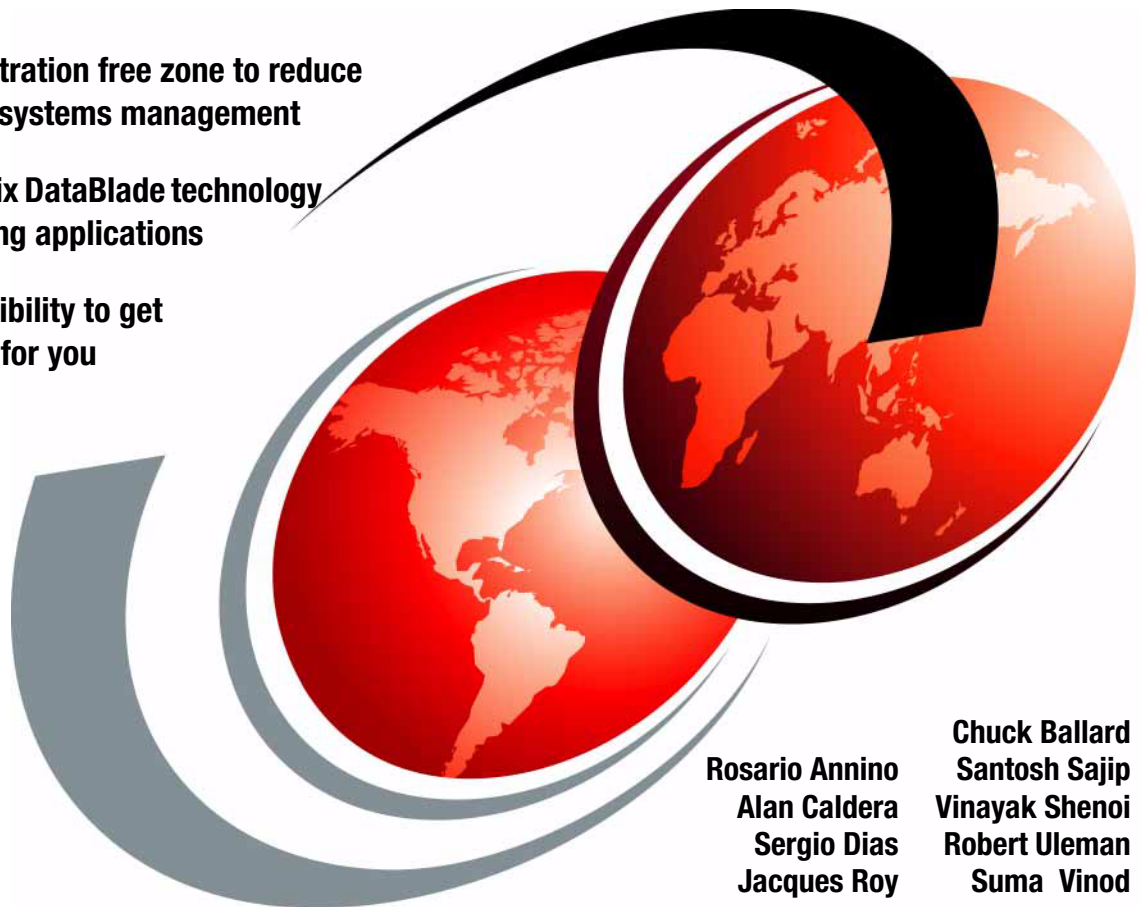


Customizing the Informix Dynamic Server for Your Environment

An administration free zone to reduce the cost of systems management

IBM Informix DataBlade technology for extending applications

Robust flexibility to get the best fit for you



Rosario Annino
Alan Caldera
Sergio Dias
Jacques Roy

Chuck Ballard
Santosh Sajip
Vinayak Sheno
Robert Uleman
Suma Vinod



International Technical Support Organization

**Customizing the Informix Dynamic Server
for Your Environment**

June 2008

Note: Before using this information and the product it supports, read the information in “Notices” on page xi.

First Edition (June 2008)

This edition applies to Version 11 of the IBM Informix Dynamic Server.

© Copyright International Business Machines Corporation 2008. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Noticesxi
Trademarks	xii
Preface	xiii
The team that wrote this book	xv
Become a published author	xviii
Comments welcome	xix
Chapter 1. IDS and your business	1
1.1 An introduction to IDS	2
1.2 Your business environment	3
1.2.1 Transaction processing	4
1.2.2 Enabling business decisions	5
1.3 IDS capabilities	6
1.4 Chapter abstracts	10
1.5 Summary	14
Chapter 2. Optimizing IDS for your environment	15
2.1 Informix Dynamic Server solutions	16
2.1.1 Available server solutions	16
2.1.2 IDS features and tools	17
2.2 Server deployment	24
2.2.1 Business environment	24
2.2.2 Features for customizing the environment	25
2.2.3 Installation Wizard footprint	25
2.2.4 Silent installation	27
2.2.5 Silent configuration	31
2.2.6 Remote instances administration tool	33
2.3 Mixed and changing environments	34
2.3.1 Example business environment	34
2.3.2 OLTP and DSS (data warehousing)	36
2.3.3 Configuring for DSS	36
2.3.4 Configuring for OLTP	43
2.3.5 Dynamically changing environments	47
2.4 Installing the Open Admin Tool	55
2.4.1 Preparing for the installation	55
2.4.2 Downloading the software	60
2.4.3 Installing the Open Admin Tool	63
2.4.4 Configuring the installation	67

Chapter 3. Enterprise data availability	75
3.1 Enterprise data availability solutions in IDS	76
3.1.1 High Availability Data Replication	77
3.1.2 Remote Standalone Secondary	78
3.1.3 Shared Disk Secondary	80
3.1.4 Continuous Log Restore	81
3.1.5 Enterprise Replication	83
3.2 Clustering EDA solutions	84
3.2.1 HA clusters	85
3.2.2 ER with HA clusters	86
3.3 Selecting the proper technology for your business	88
3.3.1 EDA technologies	89
3.3.2 Summary of capabilities and failover options	90
3.3.3 Recommended solutions	92
3.4 Sample scenarios	93
3.4.1 Failover and disaster recovery	93
3.4.2 Workload balancing with SDS and ER	96
3.4.3 Data availability and distribution using ER	98
3.4.4 Rolling upgrades for ER applications	100
3.4.5 Application redirection using server groups	103
3.5 Monitoring cluster activity	105
3.5.1 Checking the message log file and console	106
3.5.2 Event alarms	106
3.5.3 The onstat utility	106
3.5.4 The sysmaster database	106
3.5.5 The Open Admin Tool	107
Chapter 4. Robust administration	113
4.1 Disk management	114
4.1.1 Raw chunks versus cooked chunks	114
4.1.2 Managing dbspaces	115
4.1.3 Table types	120
4.1.4 Data partitioning	120
4.2 Predictable fast recovery	121
4.2.1 Benefits of RTO_SERVER_RESTART over CKPTINTVL	122
4.2.2 RTO: Dependent onconfig parameters	123
4.2.3 When not to use RTO_SERVER_RESTART	126
4.3 Automatic tuning	127
4.3.1 AUTO_CKPTS	127
4.3.2 AUTO_LRU_TUNING	128
4.3.3 AUTO_AIOVPS	129
4.4 Database connection security	130
4.4.1 OS password authentication	130

4.4.2	Pluggable Authentication Module	131
4.4.3	Lightweight Directory Access Protocol	132
4.4.4	Password encryption	132
4.4.5	Stored procedures (sysdbopen and sysdbclose)	133
4.4.6	Administrator-only mode	134
4.5	Controlling data access	135
4.5.1	Creating permissions	136
4.5.2	Security for external routines	136
4.5.3	Role-based access control	136
4.5.4	Label-based access control	137
4.5.5	Auditing	146
4.5.6	Data encryption	146
4.6	Backup and restore	147
4.6.1	Levels of backup	149
4.6.2	Ontape backup and restore	150
4.6.3	ON-Bar backup and restore	151
4.6.4	External backup and restore	155
4.6.5	Table level restore	157
4.6.6	Backup filters	158
4.6.7	Restartable restore	159
4.7	Optimistic concurrency	159
Chapter 5. The administration free zone		161
5.1	IDS administration	162
5.2	SQL-based administration	164
5.2.1	The sysadmin database	165
5.2.2	SQL Administration APIs	167
5.2.3	Examples of task() and admin() usage	167
5.2.4	Remote administration	173
5.3	Scheduling and monitoring tasks	174
5.3.1	Tasks	174
5.3.2	Sensors	176
5.3.3	Startup tasks	178
5.3.4	Startup sensors	179
5.4	Monitoring and analyzing SQL statements	180
5.4.1	Enabling and disabling tracing	182
5.4.2	Global and user modes of tracing	183
5.4.3	Examples of enabling and disabling tracing	184
5.4.4	Displaying and analyzing trace information	187
5.5	The Open Admin Tool for administration	192
5.6	The Database Admin System	208
5.6.1	Creating an idle timeout threshold	209
5.6.2	Developing a stored procedure to terminate idle users	210

5.6.3	Scheduling a procedure to run at regular intervals	214
5.6.4	Viewing the task in the Open Admin Tool	216
Chapter 6. An extensible architecture for robust solutions		219
6.1	DataBlades: Components by any other name	220
6.1.1	Object-relational extensibility	220
6.2	Data types that match the problem domain	221
6.2.1	Coordinates	222
6.2.2	Date types	230
6.2.3	Fractions	233
6.3	Denormalization for performance and modeling	242
6.3.1	Line shapes	244
6.3.2	Time series	248
6.3.3	Arrays	253
6.4	Business logic where you need it	253
6.4.1	Integration: Doing multiple customizations	253
6.4.2	Consistency: Deploying once, supporting all applications	255
6.4.3	Resiliency: Responding to changing requirements	256
6.4.4	Efficiency: Bringing the logic to the data	256
6.5	Dealing with non-traditional data	257
6.5.1	Virtual Table Interface and Virtual Index Interface	257
6.5.2	Real-time data	258
6.5.3	Emerging standards	259
6.5.4	A word of caution	259
Chapter 7. Easing into extensibility		263
7.1	Manipulating dates	264
7.1.1	The date functions	267
7.1.2	Functional indexes	269
7.1.3	Creating new date functions	270
7.1.4	The quarter() function	274
7.2	DataBlade API demystified	277
7.3	Java UDRs made easy	280
7.4	Development and deployment	284
7.4.1	Building a C UDR	284
7.4.2	Installation and registration	287
7.5	DataBlades and Bladelets	289
7.5.1	DataBlades included with IDS	289
7.5.2	Other available DataBlades	291
7.5.3	Available Bladelets	293

7.6 Summary	294
Chapter 8. Extensibility in action	295
8.1 Pumping up your data with iterators	296
8.1.1 Writing a C-based iterator function	297
8.1.2 Generating data with iterators	300
8.1.3 Improving performance with iterator functions	303
8.1.4 A challenge	309
8.2 Summarizing your data with user-defined aggregates	309
8.2.1 Extensions of built-in aggregates	310
8.2.2 User-defined aggregates	311
8.3 Integrating your data with SOA	320
8.3.1 SOA foundation technologies in IDS 11	320
8.3.2 Service providing with IDS 11	321
8.3.3 Service consumption with IDS 11	321
8.4 Publishing location data with a Web Feature Service	324
8.4.1 How organizations use spatial data	324
8.4.2 Maps and globes: The Spatial and Geodetic DataBlades	326
8.4.3 Basics of WFS	332
8.4.4 Installing and setting up WFS	347
8.4.5 Using WFS	353
8.4.6 WFS and spatiotemporal queries	358
8.5 Searching your database differently with Soundex	362
8.5.1 Creating the TSndx data type	364
8.5.2 Indexing the TSndx data type	367
8.5.3 Extending the base functionality	371
8.6 Summary	372
Chapter 9. Taking advantage of database events	373
9.1 Database servers and application architectures	374
9.2 Database events	377
9.3 Why use events	378
9.4 How to use events	379
9.4.1 IDS trigger capabilities	380
9.4.2 Trigger introspection	380
9.4.3 Creating a callback function	380
9.4.4 Registering a callback function	382
9.4.5 Memory duration	383
9.4.6 Named memory	384
9.4.7 Callback processing	386
9.5 Implementation options	390
9.5.1 Option A	390
9.5.2 Option B	391

9.6	Communicating with the outside world	392
9.6.1	Sending information to a file	392
9.6.2	Misbehaved functions	393
9.6.3	Calling a user-defined function	394
9.6.4	Sending a signal	396
9.6.5	Opening a network connection	397
9.6.6	Integrating message queues	398
9.6.7	Other possibilities	398
9.7	Conclusion	398
	Chapter 10. The world is relational	401
10.1	Virtual Table and Virtual Index Interfaces	402
10.1.1	The UDAM framework	403
10.1.2	Qualifiers	413
10.1.3	Flow of DML and DDL with virtual tables and indices	414
10.1.4	UDAM tips and tricks	419
10.2	Relational mashups	422
10.2.1	Web services	423
10.2.2	Amazon Web service	425
10.2.3	Test driving Amazon VTI	427
10.2.4	Amazon VTI architecture	437
10.3	WebSphere MQ virtual tables	441
10.3.1	WebSphere MQ	441
10.3.2	How Informix and other database applications use WebSphere MQ	443
10.3.3	IDS support for WebSphere MQ	444
10.3.4	Programming for WebSphere MQ	445
10.3.5	MQ table mapping functions	451
10.3.6	Transactions	454
10.4	Relational access to flat files	455
10.4.1	The ffvti architecture	456
10.4.2	Testing ffvti	458
	Appendix A. Additional material	465
	Locating the Web material	465
	Using the Web material	466
	System requirements for downloading the Web material	466
	How to use the Web material	467

The Amazon VTI example	468
The Fraction DataBlade example	476
TSndx datatype and overloads.	478
Glossary	493
Abbreviations and acronyms	497
Related publications	501
IBM Redbooks	501
Other publications	501
Online resources	503
How to get Redbooks	503
Help from IBM	504
Index	505

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®	DRDA®	MQSeries®
DataBlade®	eServer™	pSeries®
DB2 Universal Database™	General Parallel File System™	Redbooks (logo)  ®
DB2®	GPFS™	Redbooks®
developerWorks®	IBM®	WebSphere®
Distributed Relational Database Architecture™	IMS™	
	Informix®	

The following terms are trademarks of other companies:

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

Adobe, and Portable Document Format (PDF) are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

EJB, Enterprise JavaBeans, J2EE, Java, Java runtime environment, JavaBeans, JavaSoft, JDBC, JDK, JRE, JVM, Solaris, Sun, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Excel, Internet Explorer, Microsoft, PowerPoint, Virtual Earth, Visual Basic, Visual C++, Visual Studio, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Pentium 4, Pentium, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

In this IBM® Redbooks® publication, we provide an overview of some of the capabilities of version 11 of the IBM Informix® Dynamic Server (IDS), referred to as IDS 11, that enable it to be easily customized for your particular environment. The focus is on ease of administration and application development, since these are two of the primary areas for enabling customization. We describe and demonstrate the customization capabilities of IDS 11 with examples to show how easily it can be done.

We certainly realize that there are many other functions, features, and advantages of using IDS that come into play as you customize. However, although an in-depth focus on all of them is beyond the scope of this document, we briefly glance at some of them.

IDS 11 provides blazing online transaction processing (OLTP) performance, legendary reliability, and nearly hands-free administration to businesses of all sizes. It also offers significant advantages in availability, manageability, security, and performance.

Replication capabilities enable customers to link stores to distribution centers, distribution centers to corporate headquarters, or fast and reliable dissemination of any information across a global organization.

All these capabilities can result in a lower total cost of ownership (TCO). For example, many of the typical database administrator operations are self-managed by the IDS database, making it near hands free. The administration activities can also be controlled within an application via the SQL API. IDS customers report that they are using one-third or less of the staff typically needed to manage other database products. Shortened development cycles are also realized due to rapid deployment capabilities and the choice of application development environments and languages.

There are also flexible choices for business continuity with replication and the Continuous Availability (CA) Feature for shared disk cluster solutions. This means that there is no necessity for a “one-size-fits-all” solution. You can customize IDS to *your* environment.

The Continuous Availability Feature offers significant cost savings with support for cluster solutions, providing scalability to meet growing business demands, and failover recovery from any server to ensure continuous business operations. It provides the ability for a secondary server to automatically take over in the case of a system failure, accessing the same data disk.

Through the use of IBM Informix DataBlade® technology, the capabilities of the database can be extended to meet specific organizational requirements. All types of data can be managed, including text, images, sound, video, time series, and spatial. You can develop applications that use this technology to gain business advantages in ways that were not previously possible or practical, and typically at a lower cost.

All of this calls for a data server that is flexible and can accommodate change and growth in applications, data volume, and numbers of users. It must also be able to scale in performance as well as in functionality. The new suite of business availability functionality provides greater flexibility and performance in backing up and restoring an instance, automated statistical and performance metric gathering, improvements in administration, and reductions in the cost to operate the data server.

The technology used by IDS enables efficient use of existing hardware and software, including single- and multi-processor architectures. It also helps you keep up with technological growth, including the requirement to support complex applications, which often calls for the use of nontraditional or *rich* data types that cannot be stored in simple character or numeric form.

Built on the IBM Informix Dynamic Scalable Architecture (DSA), IDS provides one of the most effective solutions available, including a next-generation parallel data server architecture that delivers mainframe-caliber scalability, manageability and performance, minimal operating system overhead, automatic distribution of workload, and the capability to extend the server to handle new types of data.

IDS delivers proven technology that efficiently integrates new and complex data directly into the database. It handles time-series, spatial, geodetic, Extensible Markup Language (XML), video, image and other user-defined data side by side with traditional data to meet today's most rigorous data and business demands. It also helps businesses lower their TCO by using their well-regarded general ease of use and administration as well as its support of existing standards for development tools and systems infrastructure. IDS is a development-neutral environment and supports a comprehensive array of application development tools for rapid deployment of applications under Linux®, UNIX®, and Microsoft® Windows® operating environments.

The team that wrote this book

This book was produced by a team of specialists from around the world working with the International Technical Support Organization (ITSO), in San Jose, California.



Chuck Ballard is a Project Manager at the ITSO in San Jose, California. He has over 35 years experience, holding positions in the areas of product engineering, sales, marketing, technical support, and management. His expertise is in the areas of database, data management, data warehousing, business intelligence, and process re-engineering. He has written extensively on these subjects, taught classes, and presented at conferences and seminars worldwide. Chuck has both a bachelor degree and a master degree in industrial engineering from Purdue University.



Jacques Roy is a member of the IBM worldwide sales enablement organization. He is the author of “IDS.2000: Server-Side Programming in C” and the lead author of “Open-Source Components for IDS 9.x.” Jacques is also the author of multiple technical IBM developerWorks® articles on a variety of subjects. He is a frequent speaker at data management conferences, IDUG conference, and users group meetings.



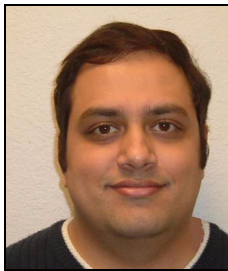
Rosario Annino is an IBM Certified Solutions Expert for IDS and DB2® for Linux, UNIX, and Windows (LUW). He is located at the Informix Lab in Bedfont, London, England. With more than 15 years experience in the software industry, Rosario has held positions as a programmer, analyst, and data warehouse administrator. He is currently in a technical support group providing level 2 support for the Informix product line. He is moving into Project Management and is already covering positions as a project manager. Rosario received a master degree in computer science in 1996 from the University of Catania, Italy.



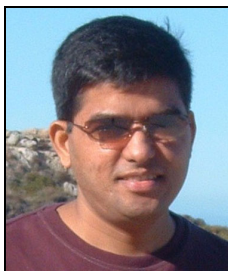
Alan Caldera is currently a team lead for the IBM Informix DataBlade development group and member of the IDS Architecture Board. He has over 20 years of experience in the IT industry as a software developer, database administrator, and consultant. Alan joined the Informix Software Professional Services Organization in 1998 as a consultant, working with customers and business partners on DataBlade implementations, application design, replication, and IDS performance tuning. He holds a bachelor degree in computer science from Indiana University.



Sergio Dias is a Software Engineer with certifications in IBM DB2 and Informix database servers, and over 25 years experience in hardware and software development, consulting, technical support, and software quality assurance (QA). He joined Informix Software, Brazil in 1995, and then moved to Miami as an Advanced Support Engineer for Latin America, the Caribbean, and Canada. Sergio currently works on the Informix QA team on High Availability and Data Replication technologies. He holds a Bachelor of Science (BS) degree in Electronics Engineering from the Instituto Tecnológico de Aeronáutica in Brazil.



Vinayak Sheno is team lead for the IBM Informix Dynamic Server SQL and Extensibility components. He has led the development of many features in the SQL, distributed queries, DRDA®, Java™, and extensibility components of IDS. Vinayak previously contributed to the Redbooks publication *Informix Dynamic Server V10 . . . Extended Functionality for Modern Business*, SG24-7299. He holds a master degree in computer science from California State University in Sacramento.



Santosh Sajj is a Senior Software Engineer with the IBM Informix Resolution team, in San Jose, California. He began his career as a member of the Informix XPS Advanced Support team in 1998 and now has over 13 years of experience in the software development and technical support field. Santosh holds a master degree in computer science from the University of Pune in India.



Robert Uleman is a member of the Information Management Worldwide Technical Sales team, focusing on spatial and spatio-temporal data management. He is a frequent contributor to publications and conferences for the geographic information systems (GIS) and location-based services (LBS) industries. Robert worked for Informix Software as a product development manager, responsible for the Geodetic, R-tree, Time Series, Video, and Image DataBlades. In his 24-year career, he has developed software products for GIS, image processing, and geophysical data processing. He holds master degrees in exploration geophysics (Stanford University) and applied physics (Delft University of Technology).



Suma Vinod is a Senior Software Engineer in the IBM World Wide Resolution team providing advanced technical support, product development, training, and defect fixing for Informix products. She joined the Informix Advanced Support team in July 1998 and has about 13 years of experience in software development and support. Suma holds bachelor degrees in computer science and engineering from Kerala University in India.

Thanks to the following people who have either contributed directly to the content of this book or to its development and publication:

- ▶ A special thanks to:
 - *Alexander Koerner* for his valuable contributions to this book and ongoing support of the ITSO. Alexander is a member of the IBM worldwide sales enablement organization, located in Munich, Germany.
 - *Donald Payne* for his significant contribution to this book, particularly in the area of Informix DataBlades. Donald is an IT Specialist with the WorldWide Enablement Center in New York, NY, USA.
 - *Prasad Mujumdar* for his valuable contributions and technical guidance in the area of user-defined access methods. Prasad is a senior member of the IDS development team, located in San Jose, CA, USA.
 - *Keshava Murthy* for his valuable contributions and technical guidance in the area of user-defined access methods. Keshava is an IDS SQL/Extensibility Architect, located in San Jose, CA, USA.

- ▶ Thanks also to the following people for their contributions to this project:
 - From IBM Locations Worldwide
 - Cindy Fung, Software Engineer, IDS Product Management, Menlo Park, CA
 - Pat Moffatt, Program Manager, Education Planning and Development, Markham, Ontario, Canada
 - From the International Technical Support Organization
 - Mary Comianos, Publications Management
 - Emma Jacobs, Graphics
 - Deanna Polm, Residency Administration

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an e-mail to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400



IDS and your business

The IBM Informix Dynamic Server (IDS) is good for your business. In this chapter, we give you a brief overview of some of the capabilities of IDS, so that you can better understand why. As you read the remainder of this book, you will see, in much detail, how IDS can be customized and optimized for your particular business environment, as a robust database management solution that can help you meet your business requirements.

1.1 An introduction to IDS

IDS delivers significant capabilities, many of which are unique. Many are also high in the list of critical capabilities that demanding businesses require. The requirements include capabilities to help manage business performance, the flexibility to change as the business changes, and increasing transaction performance to meet growing business volumes. The requirements also include extensibility to support new data types that enable increasingly robust applications and availability to meet the expanding hours of business common in a global business environment.

Such capabilities can be found in key criteria categories as in the following examples:

- ▶ Fast and easy implementation and version migration
- ▶ Minimal administration resource requirements
- ▶ Blazing online transaction processing (OLTP) performance
- ▶ Small footprint, embedability, and low operational resource usage
- ▶ Powerful extensibility for robust application development

In this book, we discuss some of the capabilities of a robust database management system (DBMS) and how they are delivered with IDS version 11 (IDS 11). As examples, IDS offers the following capabilities and features among others:

- ▶ Provides less and easier administration to run and manage
- ▶ Can automatically take corrective actions and perform self tuning
- ▶ Can administer multiple instances through a single database connection
- ▶ Enables automated systems monitoring and maintenance
- ▶ Provides a non-blocking checkpoint
- ▶ Is built on a dynamic scalable architecture
- ▶ Has a continuous availability feature for cluster solutions
- ▶ Has multiple high availability options to meet business demand
- ▶ Can be administered by command line, SQL, or the Open Admin Tool (OAT)
- ▶ Has enterprise replication for online, real-time reliable data distribution

With such capabilities, you can for perform the following tasks, for example:

- ▶ Perform implementation and maintenance with minimal resources
- ▶ Easily embed IDS in application systems
- ▶ Create your own “administration free zone” (See Chapter 5, “The administration free zone” on page 161, for details.)

Through the use of IBM Informix DataBlade technology, the capabilities of the database can be extended to meet specific organizational requirements. All types of data can be managed, including text, images, sound, video, time series,

and spatial. In addition, you can develop applications that use this technology to gain business advantages in ways that were not previously possible or practical.

All of this calls for a data server that is flexible and can accommodate change and growth, in applications, data volume, and numbers of users. It must be able to scale in performance and functionality. The IDS suite of business availability functionality provides greater flexibility for backing up and restoring an instance, automated statistical and performance metrics gathering, improvements in administration, and reductions in the cost to operate and maintain.

The technology used by IDS enables efficient use of existing hardware and software, including single- and multi-processor architectures. It also helps you keep up with technological growth, including such requirements as more complex application support, which often calls for the use of nontraditional or *rich* data types that cannot be stored in simple character or numeric form.

To summarize, IDS has significant functionality, flexibility, and capability:

- ▶ Administration that is faster and easier to save you time, money, and resources.
- ▶ Extensibility that is integrated to enable robust applications and systems solutions
- ▶ Applications that can be developed more, faster, and with fewer resources than with standardization, and a wide range of DataBlades

IDS has all this, and better yet, it can all be easily customized to support you and your particular business environment. How? Let us take a look.

1.2 Your business environment

What are your business environment and processing requirements? As we all know, there are many different business environments and many specific requirements that can make even similar environments seem different. In this section, we list some of the more common business requirements. While there are many variations, most are quite similar. We offer the following list of requirements categories to simply provide a starting point to help you in understanding the significant capabilities of IDS in supporting them. Then, your task will be to bridge this information to your particular business requirements.

- ▶ OLTP to drive the operational environment
- ▶ Packaged and custom applications
- ▶ Web-based and embedded applications
- ▶ Data analysis for decision support, through data warehousing and query
- ▶ Non-traditional data types

- ▶ Global availability for information on demand
- ▶ Mixed and changing environments

In supporting these requirements, two primary types of processing are most prevalent, OLTP and Decision Support Systems (DSS). These two types of processing are typically used in each of the environments in some form or another. In the following sections, we provide a brief overview of each to promote a consistent understanding.

1.2.1 Transaction processing

OLTP refers to an environment that is concerned with processing transactions. The transactions are processed online and typically one at a time, as they happen, rather than, say, being stored in a group or a batch and processed at a later time.

From a simplistic point of view, a *transaction* can be defined as a unit of work that consists of receiving input data, performing some processing on that data, and delivering a response. An OLTP system is one that manages the execution of these transaction applications. All of this sounds fairly straightforward, except that, over time, expanded meanings have been given to the term OLTP, as in the following examples:

- ▶ A category of transactions, with specific characteristics
- ▶ An environment for processing transactions online
- ▶ A category of applications, with specific characteristics
- ▶ A process with specific characteristics
- ▶ A category of tools used to manage the transaction processing
- ▶ A set of specific requirements for transaction completion and recovery
- ▶ A specific database organization that best supports transaction processing
- ▶ Processing characteristics that differentiate OLTP from other categories

Therefore, you must be familiar with all of these particulars as you discuss OLTP. For example, the following characteristics, among others, differentiate OLTP:

- ▶ Processing the data results in new, or changed, data.
- ▶ The duration of processing a transaction is relatively short.
- ▶ The volume of data processed in a single transaction is small.
- ▶ It typically deals with large numbers of transactions.
- ▶ A transaction recovery process is required so that data is not lost.
- ▶ The number of database accesses during a transaction is small.
- ▶ Processing time must be short, since it is performed online.
- ▶ The entire transaction must be completed successfully or rolled-back.

OLTP databases then must be capable of enabling the high performance processing that is required. That is, they need to be designed and configured to

provide fast read/write access. To support this, the data records read and written should have the following characteristics:

- ▶ Relatively small in size
- ▶ Indexed for fast access
- ▶ Lockable
- ▶ Persistent
- ▶ Recoverable
- ▶ Highly available

1.2.2 Enabling business decisions

Decision Support Systems refers to data processing applications that are concerned with performing data analysis to support the decision making process. These transactions can be processed online or in a batch. The timeliness requirement of this category of applications is determined by the business. That is, how quickly the information is required or how quickly the decision must be made.

Most typically the performance of DSS transactions is not as critical as with OLTP. This is because many DSS applications are also used simply for reporting a summary of the results of, for example, the daily business operational processes. These types of DSS transactions, which are non-exception transactions, can be processed immediately or delayed, perhaps even overnight in a batch reporting environment.

DSS applications are more typically read-only applications and, therefore, do not have the stringent recovery and availability requirements as with OLTP. Even if it is an online DSS application, and it fails for any reason, it can simply be rerun after recovery. However, a DSS application can also be run online and thus must be system managed.

The set of characteristics and requirements for DSS can be quite different from OLTP, as in the following examples:

- ▶ The applications typically access multiple databases and servers.
- ▶ The duration of the application processing is relatively long.
- ▶ The volume of data processed by the application is large.
- ▶ Typically fewer, longer running applications are processed.
- ▶ A recovery process might not be required in a read-only environment.
- ▶ The number of database accesses during a transaction is larger.
- ▶ Processing time can be long, even when performed online.
- ▶ Application results need to be repeatable.

DSS databases then must be capable of enabling multiple large tables to be scanned, read, joined, and analyzed, and particularly quickly when run online. The database typically has the following characteristics:

- ▶ Relatively large data volume
- ▶ Might require full table scans
- ▶ Federated to support multiple heterogeneous databases
- ▶ Access data across multiple servers
- ▶ Join heterogeneous data sources
- ▶ Highly available when supporting online analysis
- ▶ Scalable to handle large and growing volumes of users and data

There are many similar, but different, operating environments in the business world that are required simply to handle the differing business requirements. Throughout this book, we describe the capabilities of IDS that enable it to be customized to support all of them. The message that we want to convey is that IDS can be customized to support your environment.

1.3 IDS capabilities

We have just discussed some of the typical business environments and some of the typical requirements for them. But can all of them be met by the IDS database management system? The answer is...*definitely yes*.

In this book, we discuss and describe many of the IDS capabilities for customizing your environment. However, it is not our intention to provide detailed function and feature descriptions, but simply to discuss some of them in the context of how and where to use them and how you can use them to help you customize IDS to support your particular business requirements.

In this section, we provide a brief review of the positioning of IDS, which is summarized in Figure 1-1, and its capabilities.

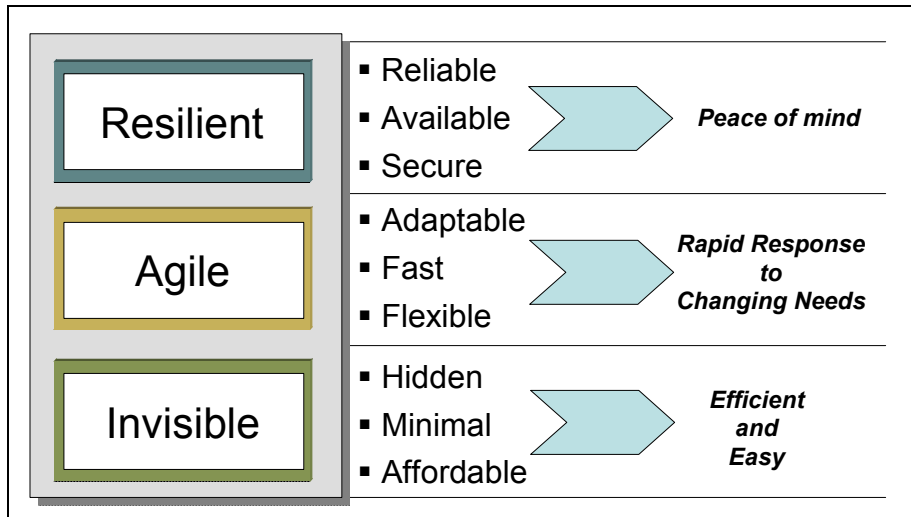


Figure 1-1 IDS capabilities positioning

Each of the categories and capabilities upon which the IDS positioning is based is briefly described as follows:

► Resilient

The business environment today is more global in nature, and the demand for high availability of systems and applications is growing dramatically to enable support of that environment. IDS capabilities are also growing to meet those demands.

– Reliable

IDS offers a broad spectrum of business continuity options to protect your data server environment. Some business situations require backup servers without duplicating data, while others need full and independent copies of the entire processing environment for failover and workload balancing around the world.

– Available

IDS 11 offers flexible choices that work seamlessly together for an availability solution to fit nearly any situation. For example, the IDS continuous availability feature enables you to build a cluster of IDS instances around a single set of shared storage devices. All instances synchronize memory structures, providing nearly seamless failover options at a fraction of the cost of having full replicas of the primary data server environment. Properly written applications can easily leverage this architecture for load balancing or practically uninterrupted data services, even in the event that one or more servers fail.

IDS extends High-Availability Data Replication by supporting multiple secondary sites, enabling you to create a failsafe, multi-site global availability plan while maximizing IT investment. Each remote replica can also be used for read access, providing more options for load balancing and improved performance. The IDS Continuous Log Restore capability extends backup and restore tactics, offering log recovery to a backup server. This is often an ideal availability solution, offering a more automated and highly available option than simple backup and restore.

- Secure

IDS supports open, industry-standard security mechanisms such as roles, password-based authentication, and relational database management system (RDBMS) schema authorizations. These open standards ensure flexibility and security with easy validation and verification. Column-level encryption and Pluggable Authentication Modules (PAMs) are also available. The Advanced Access Control Feature offers cell-, column-, and row-level label-based access control (LBAC). This means access to data can be controlled down to the individual cell level.

- ▶ Agile

To meet these business demands, businesses are going through significant change. It is not just a one-time change to support a new business need, but an ongoing change. The pace of business grows faster and faster, and business systems must continually change to keep up with the pace.

- Adaptable

Customized deployment provides support for multiple triggers on tables and views for application flexibility and compatibility.

- Fast

IDS is known for blazing fast OLTP performance. Such capabilities as the last committed isolation level and non-blocking checkpoint maximize concurrency for application performance. File system direct I/O approximates I/O performance on raw devices. Optimizer directives enable improved SQL operation performance and automatic statistics collection helps optimize data queries. These capabilities both enable significant performance gains and reduce infrastructure costs.

- Flexible

For example, the Web Feature Service application programming interface (API) in IDS 11 makes it even easier to use location-based services or location-enabled IT services. It implements an Open GeoSpatial Consortium Web Feature Service (OGC WFS) API in IDS to enable Web-driven applications to interact with location-based data provided by the IBM Informix Spatial and Geodetic DataBlade modules, which also support the Web Feature Service API.

▶ Invisible

Business systems and applications must not only keep pace with the changing business environment, but they must do so with minimal impact on the day-to-day business operations. That is, they must do so with minimal effort, low costs, and increased ease of use. One way to do that is with increased automation and self-maintenance to minimize resource costs.

– Hidden

Database administrative tasks can be controlled from within the application, permitting seamless integration for solutions. Tasks, such as space management, monitoring and manipulating memory, session management and many other activities can be executed by using SQL statements within your application with the SQL Administration API.

– Minimal

A customizable installation footprint can be created via the Deployment Wizard, enabling you to install only the data server functionality that you need, which reduces the cost and size of your solution and distribution.

– Affordable

Adding features and functionality to database management servers often results in complexity and the requirement for additional administrative duties. With IDS, that burden is relieved with administration being performed within an application or with powerful APIs and GUIs.

That is the positioning of IDS from a high level perspective. Now we discuss some examples of the special features of IDS:

- ▶ High availability to provide the systems access and support that is required to successfully compete in a global environment
- ▶ Significant security and encryption, such as LBAC and Common Criteria certification
- ▶ Spatial and Geodetic Web services for location-based services

IDS provides significant extensibility with the DataBlade technology.

- ▶ Further reduction in total cost of ownership (TCO) with improved administration functions, which are easy, yet powerful, and can be scheduled, or immediately executed automatically based on systems monitoring and programmatic alerts
- ▶ Advanced application development, with XML and SOA, for standardization and reduced cost through reusable code modules
- ▶ Enhanced solutions integration, an Admin API, and a customizable footprint that results in faster and easier administration with minimum resource requirements

Ready to learn more? These are the topics discussed and described in this book, so read on.

1.4 Chapter abstracts

In this section, we provide a brief description of the topics found in the remainder of this book. We provide a good overview of what you will find in the chapters that follow. For those who prefer to pick and choose the topics of special interest, this section will help you select and prioritize your reading.

Chapter 2: Optimizing IDS for your environment

In this chapter, we discuss topics that are related to the installation and configuration of IDS that enable you to optimize it for your environment. This chapter includes the following topics:

- ▶ How easily IDS can be customized to fit into many environments
- ▶ The new Deployment Wizard for minimizing the IDS footprint (disk space)
- ▶ Creating a customized silent installation on multiple mach machines
- ▶ The flexibility of IDS in a mixed environment, such as dynamically changing from OLTP to DSS or from DSS to OLTP
- ▶ Network infrastructure and configuration of the connections to meet requirements
- ▶ Installing and configuring the OAT

Chapter 3: Enterprise data availability

In this chapter, we provide an overview of the enterprise data availability (EDA) solutions and possible combinations of the technologies on which EDA is based. We also discuss the technologies and solutions that are available to provide the best implementation to satisfy your business requirements. In addition, we provide sample scenarios for better understanding of EDA. Upon completion of this chapter, you will have a better understanding of the technology and solutions that are available to you with IDS.

To satisfy our objectives, we discuss the following topics:

- ▶ EDA solutions
- ▶ Clustering EDA solution
- ▶ Selecting the proper technology for your business
- ▶ Sample scenarios
- ▶ Monitoring cluster activity

Chapter 4: Robust administration

The database server administrator (DBSA) plays a crucial role in the proper functioning of the database systems. IDS contains a rich set of features that provide DBSAs with a framework to create a robust environment for a high performance database system.

In this chapter, we discuss the following topics:

- ▶ *Disk management* in regard to the benefit from direct I/O on cooked chunks and how to optimize the dbspace layout and select the partitioning techniques and table types best suited for your environment
- ▶ *Performance and autonomic features* in terms of how to meet your recovery time objective (RTO), and benefit from self tuning techniques such as automatic checkpoint, automatic LRU and AIO tuning.
- ▶ *Security* in terms of how to implement database connection security, privileges on database objects using role-based access control (RBAC), and multi-level user and data access policies using LBAC
- ▶ *Backup and restore* in regard to meeting your recovery point objective (RPO) using the most suitable backup/restore technique for your configuration

Chapter 5: The administration free zone

IDS 11 provides a framework to automatically schedule and monitor database activities, to take corrective actions, and even to tune itself. Much of these features are enabled because the administration is SQL-based. Therefore, routines can be written to monitor, analyze, and change configuration parameters dynamically, based on the requirements of the IDS implementation. It is sometimes positioned as a “set it and forget it” environment because the system does much of the administration. That is also the genesis for referring to the environment as the *administration free zone*.

This framework is a significant enhancement and provides significant benefits. For example, the reduction in administration resources is a significant contributor to the low TCO of an IDS implementation.

A Web-based GUI administration tool called the *Open Administration Tool* is also available for IDS 11. This tool uses the new features in IDS 11 to provide a simple interface for performing the IDS administration tasks.

In this chapter, we provide a brief description of these features and how they make administration simple and automated in IDS 11.

Chapter 6: An extensible architecture for robust solutions

In this chapter, we begin to change the subject matter, moving from topics on the installation and configuration of the data server to the customization of its functional capabilities.

By extending the data server with new data types, functions, and application-specific structures, developers build solutions that achieve the following tasks:

- ▶ Take advantage of data models that closely match the problem domain
- ▶ Depart from strict relational normalization to achieve better performance
- ▶ Implement powerful business logic in the data tier of the software stack
- ▶ Handle new types of information

We call these solutions *robust* because the elegance and economy of their architecture gives them higher performance, maintainability, responsiveness, and flexibility in the face of changes in environment, assumptions, and requirements.

Chapter 7: Easing into extensibility

In this chapter, we describe how to get started with extensibility in IDS. We demonstrate simple examples on how to manipulate dates and discuss how to create user-defined routines (UDRs) by using C, SPL, or Java.

The basis for functional customization in IDS is the ability to add components, which are packages that contain data types, functions, index methods, and whatever else is needed to teach the data server new tricks. These components are referred to as *DataBlades*, which are, in fact, extensions of the database.

An alternative to creating DataBlades is to take advantage of extensions that are already written, which are known as either *Bladelets* (small DataBlades) or example code. These extensions are available in a fashion similar to open-source code, in that they come with source code but are not supported by IBM or the original author.

For those who want to develop their database extensions in the C programming language, IDS contains a comprehensive set of header files, public data type structures, and public functions via the DataBlade API.

Chapter 8: Extensibility in action

In this chapter, we show how to reduce the load on client applications for generating data sent to the database, how to aggregate it in the database so that it does not have to come back to the client, and how to improve performance by using views.

We also discuss how to consume Web services and provide details about new capabilities for geospatial mapping and location-based services using the Web Feature Service (WFS) available in IDS 11. One of the solutions for providing this platform independence is the Open GeoSpatial Consortium's WFS specification. It provides a generic method for accessing and creating geographic data via a Web service. A WFS provides the following capabilities:

- ▶ Query a data set and retrieve the features
- ▶ Find the feature definition
- ▶ Add features to, or delete them from, a data set
- ▶ Update features in a data set
- ▶ Lock features to prevent modification

In addition, to complete the discussion on extensibility, we include *service-oriented architecture (SOA)*. An SOA is a collection of services on a network where the services communicate with one another in order to carry out business processes. The communication can either be data passing, or it can trigger several services implementing an activity. The services are loosely coupled, have platform independent interfaces, and are fully reusable.

Chapter 9: Taking advantage of database events

In this chapter, we discuss the ability to add processing based on events that occur in the database. This capability allows you to better integrate the database in the architecture of a business solution. The result can be faster performance, simpler design and implementation, faster time to production, and response to business needs.

IDS has the unique capability of registering functions that will be executed when specific events occur. An IDS event lives in the context of a database connection. This means that the events and callback functions are specific to a user session, which then must register the callbacks before the events are generated. IDS 11 includes new stored procedures that are executed when a database is opened or when it is closed.

The logic of which external business partner is contacted following a specific event can be kept outside the application that generated the event. This way, if new business partners are added or some are removed, the application does not change. Since the logic is in the database, multiple applications can take advantage of the database logic. This simplifies the applications and provides an additional way to reuse code.

Chapter 10: The world is relational

In this chapter, we provide an discussion of the Virtual Table Interface (VTI) and Virtual Index Interface (VII) features provided by IDS. In the business world, there is still a significant volume of application data that is not stored in relational

tables, and there is a requirement to integrate this data with the data that is stored in relational databases to satisfy business operations.

IDS provides a rich framework for application developers to integrate non-relational data sources into the relational model and thereby enabling SQL query capabilities on data in these non-relational data sources. We show a few examples that illustrate the power of this framework and provide possible starting points for how to customize IDS to build complex data integration applications.

With the advent of Web services, SOA, and Web 2.0, it is evident that centralized sources of data are a thing of the past. More and more we see disparate data sources being joined in order to extract interesting information from the huge volumes of data. This is also evident in the current trend of Web 2.0 applications called *mashups*. The idea is to be able to integrate multiple Web services, or for that matter, any non-relational data source, into IDS and be able to query the data by using simple SQL statements.

1.5 Summary

We have now provided an overview of the material presented in this book. This guide will assist you in choosing and prioritizing your reading selections. In the remainder of this book, we discuss and describe the capabilities of IDS in a way that will make it easier for you to see how they can be applied and enable you to customize IDS to your specific business environment, so that you can start taking advantage of the benefits.



Optimizing IDS for your environment

In this chapter, we discuss topics related to the installation and configuration of the IBM Informix Dynamic Server (IDS) that enable you to optimize it for your environment. This chapter includes the following topics:

- ▶ How easily IDS can be customized to fit into many environments
- ▶ The new Deployment Wizard for minimizing the IDS footprint (disk space)
- ▶ The creation of a customized silent installation on multiple machines
- ▶ The flexibility of IDS in a mixed environment, such as dynamically changing from online transaction processing (OLTP) to Decision Support Systems (DSS), or from DSS to OLTP

We also discuss the network infrastructure and how to configure the connections to meet your requirements.

Let us start by discussing the IDS server solutions that are available today for your consideration.

2.1 Informix Dynamic Server solutions

The IDS delivers proven technology that efficiently integrates new and complex data directly into the database. It has outstanding functionality for application and Web development, information integration, and high performance. Most of these capabilities are unique in the industry. Thanks to the IDS architecture and robust features, it not only maintains, but accelerates its lead over other data servers in the market today. These features enable clients to use information in new and more efficient ways to create a business advantage.

In this section, we provide a brief description of the different server solutions that are available and of a large number of additional products that allow Informix to satisfy a wide range of business needs.

2.1.1 Available server solutions

Four independent server product solutions are in use today by IBM Informix customers. However, of those four, only IDS is currently being actively developed. For completeness, the four product solutions are positioned as follows:

- ▶ Informix Dynamic Server for all solution environments
- ▶ Extended Parallel Server (XPS) for large data warehouse environments
- ▶ Informix Standard Engine (SE) for small- to medium-sized OLTP environments that desire minimal data administration requirements
- ▶ Informix OnLine for small to medium-sized OLTP environments that also require additional functionality such as multi-media and application development support

We describe each of these server solutions as follows:

- ▶ *IBM Informix Dynamic Server* provides blazing fast OLTP performance and legendary reliability. In particular, IDS 11 offers significant improvements in availability, manageability, security, and performance over previous versions. In addition, it has features that can enable administration to be performed nearly hands-free in many instances.
- ▶ *IBM Informix Extended Parallel Server* is a high-end database server. It provides scalable data warehousing for the largest, most demanding, business-critical environments, and enables the integration of multiple traditional and Web-based business systems. XPS includes fast data loading and comprehensive data management to support efficient decision-making in complex environments.

- ▶ *IBM Informix Standard Engine* is an embeddable database server that provides an ideal solution for developing small- to medium-sized applications that need the power of Structured Query Language (SQL) without the database administration requirements. However, it is important to understand that the SQL provided here can be considered quite primitive when compared to the SQL provided with IDS. It integrates seamlessly with Informix application development tools, as well as third-party development tools that are compliant with the Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC™) standards.

SE is a low-maintenance and high-reliability database solution that supports the AIX®, HP UNIX, Linux, other UNIX, Sun™ Solaris™, True64 UNIX (Compaq), and Windows operating environments.

- ▶ *IBM Informix OnLine* is an easy-to-use, embeddable, relational database server for low-to-medium workloads. It features superior OLTP support with the assurance of data integrity. It provides rich multimedia data management capabilities, supporting the storage of a wide range of media such as documents, images, and audio. In addition, it supports a wide variety of application development tools, along with a large number of other third-party tools, through support for the ODBC and JDBC industry standards for client connectivity.

For more details about these products, see the Informix product family Web page for IDS 11 at the following address:

<http://www-306.ibm.com/software/data/informix/>

Also visit the IBM library Web site to download the users guide:

<http://www-306.ibm.com/software/data/informix/pubs/library/>

2.1.2 IDS features and tools

In this section, we briefly describe the features and tools that are provided by IBM Informix to increase server capability. Be aware that not all of these features and tools are available with all of the server solutions that we previously described. The focus is on IDS. For more specific details about the tools that are support by all server solutions, refer to the specific IBM Informix product descriptions. The features and tools can be segmented into five primary categories:

- ▶ Database administration
- ▶ Information integration
- ▶ Performance
- ▶ Web development
- ▶ Application development

In the following section, we provide brief descriptions of each of these categories:

► Database administration

In this category, there are tools that provide comprehensive graphical administration of the IBM Informix servers. These tools simplify such tasks as system configuration, backup and restore, system monitoring, and schema editing. This category includes the following primary administration tools:

– Open Administration Tool (OAT) for IDS 11

This set of open source code, written in PHP, enables visual administration of the IDS 11. For example, it provides the ability to administer multiple database server instances from a single location. Many tasks can be performed with the OAT, of which the most important are defining and managing automated tasks through the SQL Administration application programming interface (API), and creating and displaying performance histograms for analysis and tuning. You can also easily plug in your own extensions to the OAT to create any additional functionality that is required.

– Server Studio

This tool is a comprehensive multiplatform tool suite for IBM Informix. Developed by AGS Ltd. By using this tool, database administrators and developers can improve efficiency of daily database tasks such as application development, schema and storage space management, performance monitoring and optimization, event response management, and server maintenance. With Server Studio, you can maximize the availability, performance, and manageability of your entire IBM Informix infrastructure from a single point of control, regardless of whether the database servers are in one location or at remote sites. Server Studio also provides a comprehensive configuration auditing and change management solution for data servers to preserve your database investment.

► Information integration tools

This category includes tools that enable transparent access to data from servers. It includes the following tools among others:

– Informix Enterprise Gateway Manager

This tool is an SQL-based gateway that allows Informix tools, applications, and databases to interoperate transparently with non-Informix databases. It makes the underlying target database management system (DBMS) appear to both client and server applications as an instance of the IDS. For example, it allows distributed joins with Informix, Oracle®, and Sybase data in a single SQL statement.

- Informix Enterprise Gateway Manager with DRDA

This tool provides a UNIX-based connectivity solution for IBM relational databases. For example, it enables read-and-write access to IBM data from UNIX-based applications. Its transparent connectivity gives users full read-and-write access to a wealth of information that was previously available only through lengthy batch processing operations, file transfers, or cumbersome proprietary gateways. You can also execute distributed joins to integrate data from multiple databases.

- ▶ Performance tools

This category of tools is designed to dramatically increase server performance. The following tools are included in this category:

- Informix MaxConnect

This tool improves system scalability and performance by increasing the number of users that can simultaneously connect to an Informix database server. By multiplexing client connections to a smaller number of network connections, Informix MaxConnect moves the management of user connections away from the Informix database server, significantly reducing operating requirements on the database server while increasing system scalability. It reduces CPU requirements and optimizes the use of operating system and network resources.

- I-Spy

This tool is a smart data warehouse monitoring and optimization tool that is designed for IBM Informix databases. It helps warehouse administrators and architects increase the business value of enterprise data warehouses through enhanced utilization efficiency, design improvements, and lower maintenance costs. I-Spy sits transparently between the database and the client, helping the administrator monitor and adjust database resources and client query usage.

- Server Studio Sentinel

This tool was developed by AGS Ltd. The Performance Edition provides a comprehensive, easy-to-use issue discovery, analysis, and response automation solution that enables rapid identification and remediation of database performance problems in the IBM Informix data server environment. This powerful multiplatform agent-less tool helps pinpoint the underlying causes of IBM Informix data server performance bottlenecks and resource contention issues for fast and efficient resolution of database problems before they seriously impact users.

Sentinel automates 24x7 monitoring of IDS operations in real time, provides over 160 real-time operational parameter measurements to be used to monitor the IDS data servers and host systems on which they reside. It also performs dynamic capture of SQL statements running on

data servers, based on user login name, client host, Informix session ID, or SQL statements execution statistics.

▶ Web development

This category contains a tool that allows easy development, management, and deployment of database applications for the Web.

– Data Director for Web

This tool provides a model-driven development environment that is designed explicitly for creating powerful database applications that can grow with your business. It addresses both evolving enterprise needs and increasingly diverse technical requirements. It also automates all of the data access operations of the client application. By using this tool, developers can easily incorporate sophisticated functionality without having to be database programming experts.

▶ Application development tools

This category contains tools that enable you to create a wide range of powerful business applications quickly, including Web-ready, dynamic content management, and Java-based systems. The following tools are a sampling of those that are included in this category:

– Informix 4GL

This tool consists of an integrated rapid development system, interactive debugger and compiler. It is a comprehensive fourth-generation application development and production environment that provides power and flexibility without the need for third-generation languages such as C and COBOL. It creates sophisticated database applications, in three easy packages with one consistent interface. Used together, Informix 4GL Rapid Development System and Informix 4GL Interactive Debugger provide an environment for developing applications, while Informix 4GL Compiler provides high-performance application execution in the production environment.

– Client Software Development Kit (Client SDK)

This tool offers a single package of several APIs that are optimized for developing applications for Informix servers. By using this tool, developers can write applications in the language that they prefer and build one application to access multiple IBM Informix databases.

– IBM Informix Connect

This tool is a runtime connectivity product that includes the runtime libraries of the Informix Client Software Development Kit (Client SDK). These libraries are required by applications that run on client machines when accessing Informix servers. Informix Connect is needed when finished applications are ready to be deployed.

- IBM Informix ESQL/C

This tool provides the convenience of entering SQL statements directly into the C language source code. Developers can use SQL to issue commands to the IBM Informix server and to manage the result sets of data from queries. The ESQL/C compiler takes ESQL/C programs and enables them to use ESQL/C libraries, so that they can communicate with the database engine.

- IBM Informix ESQL/COBOL

This tool is an SQL API with which you can embed SQL statements directly into COBOL code. It consists of a code preprocessor, data type definitions, and COBOL routines that can be called. In addition, it can use both static and dynamic SQL statements. When static SQL statements are used, the program knows all the components at compile time.

- IBM Informix Java Database Connectivity (JDBC)

This tool is the JavaSoft™ specification of a standard API that allows Java programs to access DBMSs. Informix JDBC V3.0 is a platform-independent, industry-standard Type 4 driver that provides enhanced support for distributed transactions. It is optimized to work with the IBM WebSphere® Application Server.

- IBM Informix SQL

This database application development system provides the speed, power, and security that are required by both large and small database applications. It features a suite of five application development tools, including a schema editor, menu builder, SQL editor, forms builder, and report writer. It also provides rapid development for green-screen terminals and applications that do not require programming language. It includes easy reporting capabilities for quick data analysis and enables quick access to data for evaluation.

- Python driver

This tool makes IDS and Python an excellent combination for innovative developers who want to build high performance, custom applications.

- PHP driver

PHP is the fastest growing technology for building dynamic Web applications. It provides high performance and allows PHP code to use IDS as a data repository.

– DataBlade modules

These modules extend the capabilities of IDS with user-defined objects. The following DataBlade modules are available:

- C-ISAM DataBlade module

This module is a library of C functions that efficiently manage Indexed Sequential Access Method (ISAM) files. It was developed specifically to help you add relational database management system (RDBMS) features to your C-ISAM environment or to assist you in migrating C-ISAM applications to an RDBMS environment. It also enables fast data access, includes flexible indexing options, supports large files, and provides an SQL interface to C-ISAM data through an SQL Access component. Also included is a Server Storage component that enables you to store ISAM data directly in the database server, while allowing C-ISAM programs to continue accessing this data.

- Image Foundation DataBlade module

This module provides a base on which new or specialized image types and image processing technologies can be quickly added or changed. It can store and retrieve images and metadata in the database or on remote computers and storage servers. In addition, you can transform images by using the industry-standard CVT command set.

- Excalibur Text Search DataBlade module

This module enables full text search capabilities, so that you can take full advantage of an engine that is optimized for indexing and searching text information, using proximity searches and other features. This means you can add extensive text-searching to many existing applications. It delivers full-text indexing, including extensive support for fuzzy-search logic, which is especially important when indexing scanned text. Plus it provides rapid query results and offers support for various document types including ASCII, Word, Microsoft Excel®, Microsoft PowerPoint®, HTML, PDF, and WordPerfect.

- Basic Text Search DataBlade module

By using this module, you can search words and phrases in an unstructured document repository that are stored in a column of a table. This module is considered the evolution of the Excalibur module.

- Binary DataBlade module

This module includes the binary18 and binary var data types that allow you to store binary-encoded strings, which can be indexed for quick retrieval. The Binary DataBlade module comes with string manipulation functions to validate the data types and bitwise operation functions with

which you can perform bitwise logical AND, OR, or XOR comparisons or apply a bitwise logical NOT to a string.

- Node DataBlade module

This module gives you the ability to classify the actual row data in a defined hierarchy. You can consider the table as a hierarchical tree and can easily navigate up, down, left, and right to find the parent, left and right node, neighbors, depth, children, and parents in the tree. You can compare tree levels and generate new tree levels and twigs based on a given parent.

- Geodetic DataBlade module

With this module, you have the ability to manage geospatial information referenced by latitude-longitude coordinates, supporting global space and time-based queries without the limitations inherent in map projections. This module manages spatial data by using geographic information systems (GIS) technologies.

The Informix Geodetic DataBlade module is best used for global data sets and applications. It provides a robust, well-crafted C-language API, so that you can build new functions that use the same data structures and internal interfaces used by SQL functions already provided.

- Spatial DataBlade module

This module expands the IBM IDS object-relational data server to provide industry leading SQL-based spatial data types and functions that can be used directly through standard SQL queries or with client-side GIS software, such as that from ESRI and MapInfo. This module enables organizations to transform both traditional and location-based data into important information to help gain a competitive advantage. These data types can store spatial data such as the location of a landmark, a street, or a parcel of land.

- TimeSeries DataBlade module

This module expands the functionality of the database by adding sophisticated support for the management of time-series and temporal data. A *time series* is any set of data that is accessed in sequence by time that can be processed and analyzed in a chronological order.

- TimeSeries Real Time Loader DataBlade module

This module is a data loader that works in conjunction with IBM Informix-NAG Financial DataBlade technology to achieve greater analytical performance than is possible with either traditional relational databases or stand-alone real-time analysis software.

- Video Foundation DataBlade module

This module is an open and scalable software architecture that allows strategic third-party development partners to incorporate specific video technologies, such as video servers, external control devices, compression codecs, or cataloging tools, into complete database management applications with the Informix Dynamic Server. You can manage video content and metadata, or information about the content.

- Web DataBlade module

This module enables you to create Web applications that dynamically retrieve data from an IBM Informix database. By creating HTML pages that include Web DataBlade module tags and functions, you can execute SQL statements to retrieve information for the Web page and format the results in HTML for display.

- DataBlade Developers Kit

This kit is an easy-to-use graphical interface for developing DataBlade modules.

In this section, we provided a general overview of the major features and tools that are integrated with IBM Informix database servers. For more information, see the IBM Informix library at the following links:

- ▶ Informix product family page

<http://www-306.ibm.com/software/data/informix/>

- ▶ Informix library

<http://www-306.ibm.com/software/data/informix/pubs/library/>

2.2 Server deployment

In this section, we discuss the capability of Informix to adapt to a particular environment. We consider the business environment and requirements of a development company that uses IDS as the data repository. As examples, we show how IDS can be installed with just 100 MB of disk space, how you can add and remove IDS features, a how you can install and configure IDS without any interaction from the user, to demonstrate the flexibility of IDS.

2.2.1 Business environment

In this example, we analyze the business requirements of a company that develops applications that use IBM Informix as the data repository. The development company sells the application, along with IDS, and provides

support for both the application and for IDS. In this example, the development company wants a procedure to easily install and configure the environment to use their applications. This particular company has the following environment:

- ▶ There are a few thousand users.
- ▶ Many users have limited Informix skills.
- ▶ The users also have limited hardware resources.
- ▶ Each user buys the application and the Informix server database.
- ▶ The development company supports the applications and the administration of the Informix database.

The development company also wants an installation procedure that has the following characteristics:

- ▶ Requires little space
- ▶ Easily installs and configures the database with almost no interaction from the users
- ▶ Creates the same directory structure for all the users
- ▶ Has a remote administration tool that can provide easy administration of all the Informix instances

2.2.2 Features for customizing the environment

In this section, we describe one possible solution for the scenario described in 2.2.1, “Business environment” on page 24, and demonstrate how easily it can be adapted to any particular environment.

To do this, we use the following features:

- ▶ Installation Wizard to reduce the Informix installation space
- ▶ Silent installation to reduce the interaction with the user who installs Informix
- ▶ Sysadmin database to create the same directory structure and configure Informix without interaction from the users
- ▶ The OAT to manage all the instances remotely

2.2.3 Installation Wizard footprint

The IDS installation currently provides users with two installation options:

- ▶ Typical installation
- ▶ Custom installation

With IDS 11, the custom installation has been improved. To do so, IDS is now divided into multiple components based on functionality. Each component is then

divided into subcomponents to allow more granularity. By using the Installation Wizard, or Deployment Wizard, you can select only the Informix functionality and features that you need, thereby reducing the space that is used by Informix during the installation. The Installation Wizard checks the dependencies between components and guides you to the correct selection of functionalities. Vendors who embed IDS into their applications can now install only the minimum functionality and features to run their applications. For example, they can install only the base server without other components, which reduces disk space to approximately 100 MB.

Important: You must choose the custom installation to access the option to remove the IDS components.

In Figure 2-1, you can see the Informix Deployment Wizard component tree.

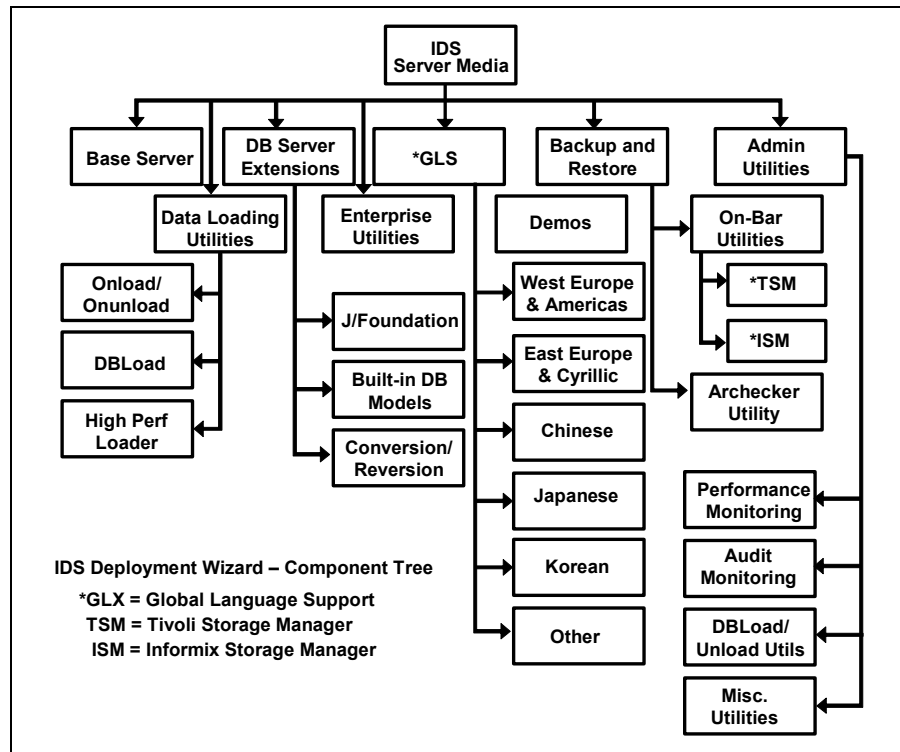


Figure 2-1 Deployment Wizard - Component tree

During the custom installation, you see a window like the one shown in Figure 2-2. On this window, you can select only the components that you need for your particular environment, thereby reducing the space needed for the installation.

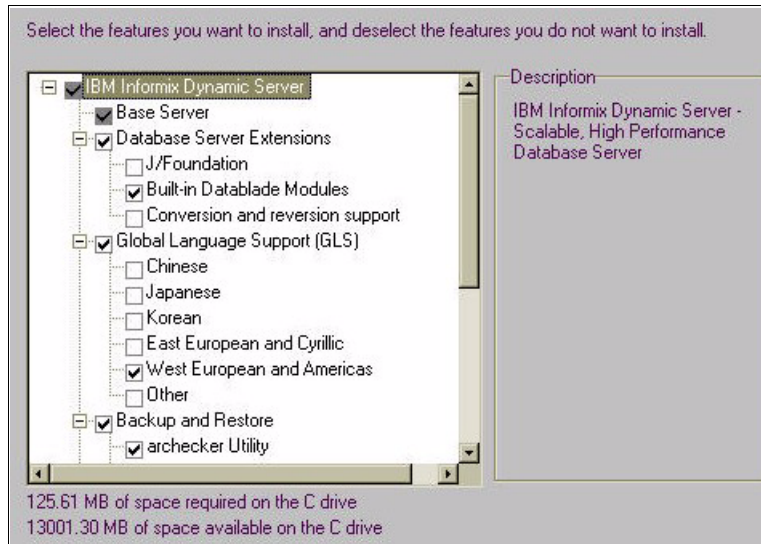


Figure 2-2 Custom installation footprint

For more information about the Installation Wizard, and IDS features and components, see the *IBM Informix Dynamic Server Installation Guide for Microsoft Windows*, G251-2776, or *IBM Informix Dynamic Server Installation Guide for UNIX and Linux*, G251-2777.

2.2.4 Silent installation

A *silent installation* is an installation method that requires no user interaction with the setup program. The installation is driven by a response file named *server.ini* (on Windows) or *responsefile.ini* (on UNIX or Linux) that declares the desired actions. Silent installation has the following advantages:

- ▶ There is no user interaction during the installation.
- ▶ The installation can be run by a person with no Informix skills.
- ▶ It is easy to replicate the same installation on different machines.

To use the silent installation, perform the following actions:

1. Install the IDS and record the preferences in a response file.

To re-use the GUI custom-setup installation configuration to install IDS, in the same way, on other machine, you must capture the responses from a graphical installation by using the following steps:

- a. Open a command window.
- b. Go to the directory where the Informix installation media resides.
- c. Be sure to have the setup.exe (Windows) or ids_install (UNIX or Linux) in the correct directory.
- d. Execute the following command depending on your platform:

- Windows:

```
setup.exe -r -f1"C:\temp\silent.ini"
```

If you run this command, without the option -fl, the response file is created in the Windows directory. The option -r means record the installation actions.

- UNIX or Linux:

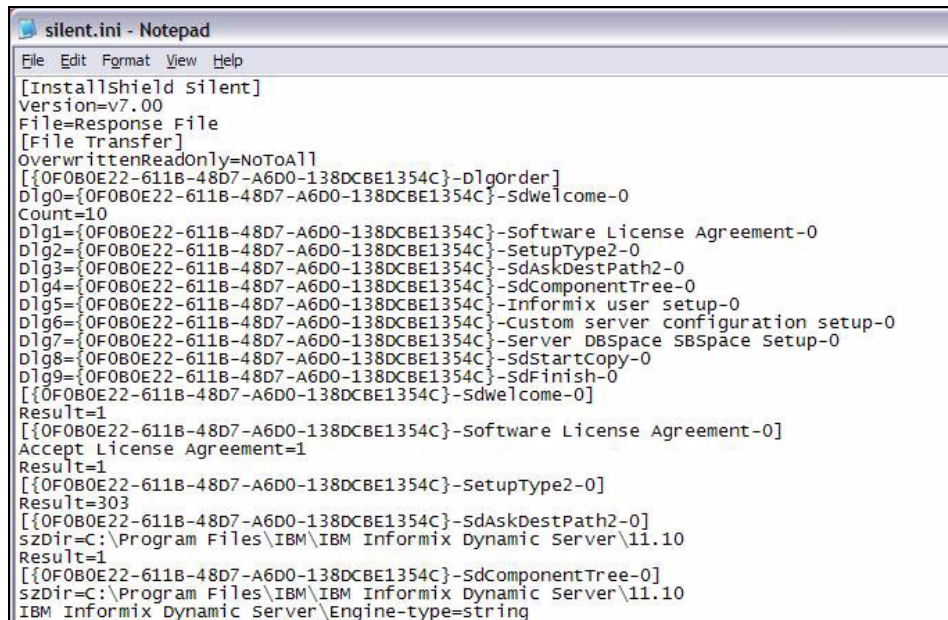
```
ids_install -gui -record responsefile.ini
```

The option -record captures the installation actions in the response file.

These commands start the GUI installation and record your choices in the response file.

After the installation starts and you accept the Software Licensing Agreement, you can choose the desired type of installation. To install only some Informix components (and minimize the footprint), choose **custom installation**. Then select the Informix component to be installed as described in 2.2.3, "Installation Wizard footprint" on page 25. When the installation finishes, check the response file.

Figure 2-3 shows an excerpt from the response file. You do not need to read the example, but simply use it to be familiar with the type of output to be received.



```
[Install]Shield silent]
Version=v7.00
File=Response File
[File Transfer]
OverwrittenReadOnly=NoToAll
[{{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}}-DlgOrder]
Dlg0={{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}}-Sdwelcome-0
Count=10
Dlg1={{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}}-Software License Agreement-0
Dlg2={{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}}-SetupType2-0
Dlg3={{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}}-SdAskDestPath2-0
Dlg4={{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}}-SdComponentTree-0
Dlg5={{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}}-Informix user setup-0
Dlg6={{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}}-Custom server configuration setup-0
Dlg7={{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}}-Server DBSpace SBSpace Setup-0
Dlg8={{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}}-Sdstartcopy-0
Dlg9={{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}}-SdFinish-0
[{{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}}-Sdwelcome-0]
Result=1
[{{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}}-Software License Agreement-0]
Accept License Agreement=1
Result=1
[{{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}}-SetupType2-0]
Result=303
[{{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}}-SdAskDestPath2-0]
szDir=C:\Program Files\IBM\IBM Informix Dynamic Server\11.10
Result=1
[{{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}}-SdComponentTree-0]
szDir=C:\Program Files\IBM\IBM Informix Dynamic Server\11.10
IBM Informix Dynamic Server\Engine-type=string
```

Figure 2-3 Response file

In addition to the response file, the manifest file %INFORMIXDIR%\etc\manifest.inf can help you quickly see the features and components that are currently installed.

2. Install the IDS by using the response file.

Now you are ready to use the response file to replicate your installation in other machines, without any interaction from the user. To do this, you need the following items:

- The IDS installation software package
- The response file silent.ini (Windows) or responsefile.ini (UNIX or Linux)

In the next machine to be installed, perform the following actions:

- a. Open a command window.
- b. Change directory and go to the directory that contains the Informix installation software.
- c. Be sure that you have the setup.exe (Windows) or ids_install (UNIX or Linux) file in the correct directory.

d. Execute the following command depending on your platform:

- Windows:
`setup.exe -s -f1"c:\TEMP\responsefile.ini"`
- UNIX or Linux:
`ids_install -gui -record responsefile.ini`

3. Check the silent log file.

There is little to show regarding a silent installation because it is *silent*. That is, there is nothing that shows the progress or that it has completed. The only way that you can discover that it has completed is that the command prompt returns. After the installation finishes, check the log file to know whether the installation was successful. The log file resides in `%INFORMIXDIR%\logs\IDS_Install_date_time.log`.

Figure 2-4 shows how the log file should appear. Again, it is not required that you read the file, but just be familiar with how the output appears.

```
IDS_Install_10_17_2007_17_44_11.log - WordPad
File Edit View Insert Format Help
Log file created - Date and Time: 10-17-2007, 17:44:11.
IBM Informix Dynamic Server 11.10

Install mode selected: Custom
Domain: domain_name
Install destination directory: C:\informix\ids11tc1

***** Attention *****
The Server will be automatically initialized. Any existing data will be lost.
If you do not want to lose the existing data, click Back to change the installation options or click Cancel to abort the installation.

When you initialize a server, a shortcut is added to the Start menu. To run commands for an initialized
server, click Start - All programs - IBM Informix Dynamic Server 11.10 - <server_name>.

Server Details
Server name: ol_ids11tc1
Service port name: svc_custom
Service port number: 9088
Server number: 0

Total size of all the features selected is approximately 211 MB.
Total disk space available at the destination directory is approximately 14917 MB.

Installation log
C:\informix\ids11tc1\logs\IDS_Install_10_17_2007_17_44_11.log

SNMP Service could not be located on this system.

Shortcut Creation
...Complete.

Registry entry creation
...Complete.

Feature manifest creation
...Complete.

Branding Product Binaries
...Complete.

Configuration file for Instance Manager
...Complete.

Starting Server Instance
...Complete.
```

Figure 2-4 Silent log

2.2.5 Silent configuration

At this point, IDS has been installed without interaction with the user. All the installations that have executed the silent installation have a similar environment. That is, they have the same INFORMIXDIR, same onconfig file, and same sqlhost. Any other requisite directories are established by the system administrator.

Now we demonstrate how some administration tasks can be executed with minimum interaction with the user. For example, to execute the tasks in the following list, we use the sysadmin database:

- ▶ Add a new dbspace named dbs1, with a size of 100 MB, in the \$informixdir\chunks directory.
- ▶ Add a new dbspace for the logical log named logdbs, with a size of 100 MB, in the \$informixdir\chunks directory.
- ▶ Add a new dbspace for the physical log named physdbs, with a size of 100 MB, in the \$informixdir\chunks directory.
- ▶ Add three logical logs in the dbspace logdbs with a size of 10 MB.
- ▶ Remove the first two logical logs from the rootdbs.
- ▶ Alter the physical log file from the default value to 30 MB.
- ▶ Change the RTO_SERVER_RESTART from the default value to 60.
- ▶ Execute a checkpoint.

All these tasks can be done, but we do not describe the details here. For more information about the sysadmin database, refer to Chapter 5, “The administration free zone” on page 161.

Now we show how you can create a script that performs all the administrative tasks mentioned previously without interaction from the user by using the following steps. The user must only execute the script from the user Informix.

1. Create a file named tailor_admin.sql.
2. Write the code as shown in Example 2-1.

Example 2-1 tailor_admin.sql

```
database sysadmin;
execute function admin("create dbspace", "dbs1",
"$INFORMIXDIR\chunks\dbs1", "100MB", "0");
execute function admin("create dbspace", "logdbs",
"$INFORMIXDIR\chunks\logdbs", "100MB", "0");
execute function admin("create dbspace", "physdbs",
"$INFORMIXDIR\chunks\physdbs", "100MB", "0");
```

```

execute function admin("add log", "logdbs", "10MB");
execute function admin("add log", "logdbs", "10MB");
execute function admin("add log", "logdbs", "10MB");
execute function admin("drop log", "1");
execute function admin("drop log", "2");
execute function admin("drop log", "3");
execute function admin("alter plog", "physdbs", "30 MB");
execute function admin("ONMODE", "wf", "RTO_SERVER_RESTART=60");
unload to "command_history.txt" select * from command_history;

```

The last line of Example 2-1 generates a file named `command_history.txt` that contains a list of all commands that the administration API ran. You can see the results of each of the commands. The file `command_history.txt` should look much like the example in shown in Figure 2-5.

3. Execute the configuration tasks.

From user `informix`, open a command window and execute the following command:

```
dbaccess - tailor_admin.sql
```

4. Check the result of the commands that have been executed.

Analyze the file `command_history.txt` that was generated by the script `tailor_admin.sql`. The file `command_history.txt` should look much like the example shown in Figure 2-5. The sixth column contains the error codes, which we have highlighted with the ovals. In this example, the commands executed correctly. Therefore the error codes that are displayed are zeros.

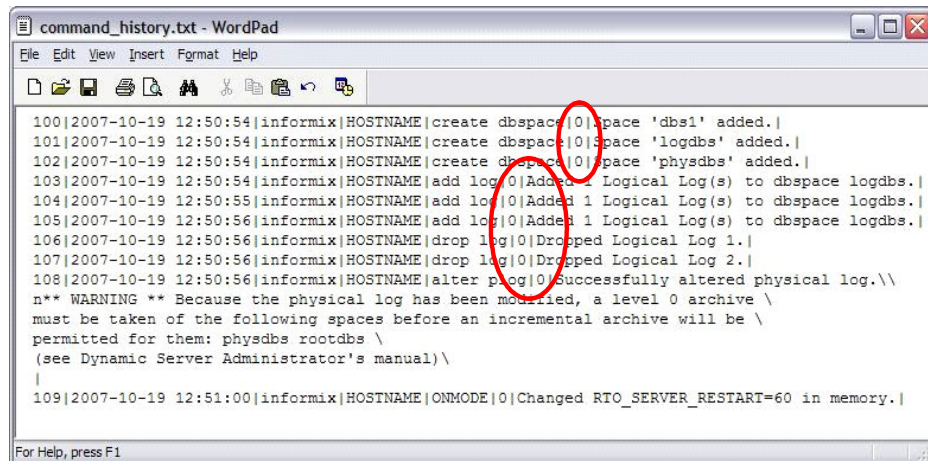


Figure 2-5 Command history output

If the commands were not executed correctly, the error codes would not be zero as demonstrated in Figure 2-6. Here we added a second command to create the same dbpace. As you can see in line 101, the error code is not a zero, but a -1. Included in this line is a brief description of the error, which in this scenario indicates that the space was not created, because it already exists.

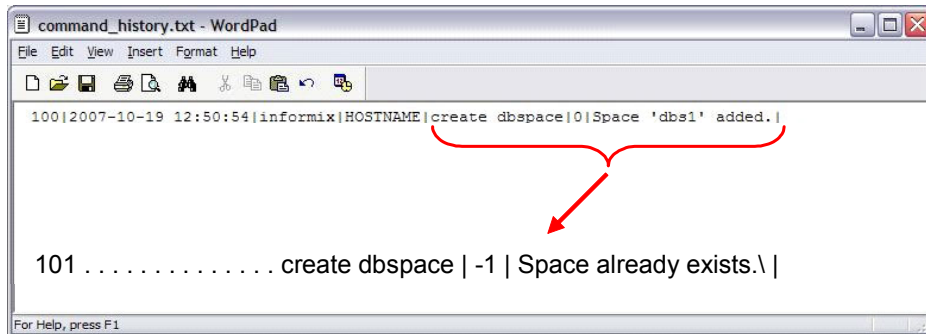


Figure 2-6 Command history output with error

When the sixth column is different from zero, you must analyze the error and fix it manually. For a description of the error, open an Informix command window and execute the following command:

```
finderr error_number
```

You have now completed the silent configuration.

2.2.6 Remote instances administration tool

Now you should have many users with a working environment, configured to work with your application. What is desired now is an easy way to manage all these instances remotely. In this scenario, the OAT can be useful. In fact, you can configure the OAT to manage all the instances and check the status of the instances remotely by using the following tasks:

1. Install and configure the OAT.

For more information about this task, see 2.4, “Installing the Open Admin Tool” on page 55.

2. Add a new connection in the OAT.

For each instance that you want to manage remotely, you must add a connection in the OAT. The host name requested in the OAT new connection form, see Figure 2-22 on page 70, must be the machine name or the IP address of the machine where the instance resides.

3. Connect to the server.

Open the OAT main page. In the quick login box, select the group, insert the password, and click **Get Servers**. Then select the instance name. The field on the right is filled in automatically as shown in Figure 2-23 on page 72. Click the **Login** button in the bottom left corner of the window.

4. Manage the instance.

At this point, you should be connected to the instance that you selected as shown in Figure 2-24 on page 73. Here you can manage the instance remotely. In addition, you can check the logs, spaces, checkpoints, message log, and much more.

2.3 Mixed and changing environments

One of the characteristics of IDS is the capability to adapt itself to a large range of different environments. In this section, we focus our attention on two particular environments, OLTP and DSS, that could be considered the upper and lower bound of the environment range. OLTP applications are typically designed to be highly performing and deal with small volumes of data. DSS applications are typically query oriented, longer in duration, and deal with much larger volumes of data. DSS is typically associated with, and retrieves data from, a data warehousing environment. Some of the primary differences between these two environments are summarized in Table 2-1 on page 36.

In this section, we show how IDS can be dynamically changed from supporting one environment to supporting another environment, without a requirement to reboot the instance, demonstrating the flexibility of IDS. This type of mixed environment is more typically supported with medium-sized instances. For large DSS or OLTP systems, it might be better to consider a dedicated machine for each environment, or to reboot the instance to change the onconfig file when moving between those environments.

2.3.1 Example business environment

Here we consider an example business environment with a requirement during the night to do the following tasks:

- ▶ Load a massive amount of data in the IDS database server.
- ▶ Execute applications that analyze the data.
- ▶ Read entire tables to generate data aggregations, reports, or both.

During the day, the activities in the IDS database server are different. For example, a large number of users execute queries to analyze the data and access reports generated from the DSS environment the previous night. In general, in this example, we can consider a mixed environment one that performs DSS during the night and OLTP during the day.

An important consideration in the mixed environment is the database server performance. That performance depends on several factors, such as hardware resources, operating system tuning, and more importantly, the database server configuration. The mixed environments that we have described are so different that it is typically suggested to use two different server configurations to provide maximum performance in both. Informix has the capability to dynamically adapt itself to both environments, thus removing the bottleneck described.

Several activities, such as the following activities, are involved in configuring a mixed environment:

- ▶ Understand the differences between OLTP and DSS.
- ▶ Tune the two environments separately.
- ▶ Find the onconfig parameter values that provide maximum performance for each environment.
- ▶ Do performance tuning for a mixed environment.

Some onconfig parameters, such as RA_PAGES and RA_THRESHOLD, cannot be changed dynamically. For those parameters, you must perform a tuning activity to find a value that can provide acceptable performance for both the environments, even though it means that performance might not be truly optimal for either workload. However, you can also choose to optimize one workload knowing that the other will be less optimal.

Always execute small changes, and then analyze the results. Keep in mind that with IDS, you can change dynamically some of the onconfig parameters, so that you can schedule the changes every time you need to switch from one environment to the other.

Note: In the following sections, we provide values that can be considered as starting points for performance tuning activity. The values came from practical experience and have shown that they are a reasonable starting points from which to begin tuning a configuration. For more information about this topic, refer to the *IBM Informix Dynamic Server Performance Guide*, G229-6385.

2.3.2 OLTP and DSS (data warehousing)

Table 2-1 contains examples of some of the considerations for, or typical differences between, OLTP and DSS environments.

Table 2-1 OLTP and DSS

OLTP	DSS
Many users	Few users
Few rows read per transaction	Many rows read per transaction
Index access data	Sequential scan access data
Fast response time (less than 2 seconds)	Long response time (minutes/hours)
Large resident memory	Small resident memory
Small virtual memory	Large virtual memory
Read cache rate of 95% or better	Read cache rate below 80%
Write cache rate 85% or better	Write cache rate below 80%
No parallel database query (PDQ)	PDQ
Multiple network ports	Single network port
Network protocol TCP/IP	If local, network protocol shared memory
Fast recovery time is important	Fast recovery time is not as important
Checkpoint duration is important	Checkpoint duration is usually not a factor
Checkpoint execution more frequent (seconds)	Checkpoint execution less frequent hours

2.3.3 Configuring for DSS

In this section, we discuss some of the major criteria that can impact the DSS environment, which are to be considered as you configure for optimization. As examples, give consideration to the following items:

- ▶ Achieving optimum memory utilization
- ▶ Using light scans
- ▶ Using PDQ
- ▶ The type of fragmentation to be used
- ▶ Maximizing the I/O throughput

Important: In subsequent sections of this chapter, where monitoring is discussed, we typically used the **onstat** or **oncheck** commands. Many monitoring tasks can also be performed by simply using SQL select statements against the sysmaster database. However, be aware that performing a select on the sysmaster results in latches (locks) on the sysmaster tables. Since these tables are used by the engine, that could introduce performance problems. Therefore, we recommend that this not be a best practice.

Let us look at the requirements in more detail as follows:

► Memory utilization

The objective is to maximize the size of virtual memory. Usually the value of the SHMVIRTSIZE is set to 75% of the memory available, but this approach is not completely correct. You should allocate only the memory that is necessary for your environment. Therefore, you need to find the point at which, during the peak time (the instance of time at which resource utilization is highest), IDS will not add any segments, while if you reduce the size of SHMVIRTSIZE, IDS will add another segment.

The following method is a good approach to find the value of SHMVIRTSIZE:

- a. Set SHMVIRTSIZE to 50% of your memory.
- b. During the peak time, check how many additional segments have been added. You can use the **onstat -g seg** command. If no additional segments were added, then reduce the SHMVIRTSIZE. When you notice additional segments, proceed to the next step.
- c. Use the following formula:
$$\text{new_SHMVIRTSIZE} = \text{old_SHMVIRTSIZE} + (\text{number_of_segments_added} * \text{SHMADD})$$
- d. When you find the optimal SHMVIRTSIZE, try to reduce a little bit the size to see if an additional segment is created. This is a way to double check that your calculations are correct.

► Light scan

The light scan is a mechanism that bypasses the traditional reading process. Pages are read from disk and put in the buffer cache in the resident shared memory segment. The light scan reads the page from disk and puts it in the light_scan_buffers, which reside in the virtual segment of the shared memory. It then reads the data in parallel, providing a significant increase in performance when compared with scanning large tables.

The number of light scan buffers is defined by the following equation:

$$\text{light_scan_buffers} = \text{roundup}((\text{RA_PAGES} + \text{RA_THRESHOLD}) / (\text{MAXAIOSIZE} / \text{PAGESIZE}))$$

MAXAIOSIZE: MAXAIOSIZE is an Informix internal parameter and is platform dependent. In general, it is in the area of about eight pages.

As you can see, RA_PAGES and RA_THRESHOLD can impact the number of light scan buffers, and they cannot be changed dynamically. You can consider creating dbspaces that are dedicated to the DSS activity, giving them a larger page size. When increasing the PAGESIZE, IDS increases the number of light scan buffers (see the previous equation). The page size must be a multiple of the operating system page size, but not greater than 16 kilobytes (KB). Place attention on the size of your row. Each page can contain a maximum of 255 rows. Therefore, if the row size is small and the page size is large, you can risk to lose disk space. To know the maximum row size, use the following command:

```
oncheck -pt databasename:tablename
```

Then check the line “Maximum row size.”

To create a dbspace with a customized page size in KB, you can use the following command:

```
onspaces -c -d DBspace [-t] [-k pagesize] -p path -o offset -s size [-m path offset]
```

– BUFFERPOOL

The BUFFERPOOL configuration parameter specifies the values for BUFFERS, LRUs, LRU_MAX_DIRTY, and LRU_MIN_DIRTY for both the default page size buffer pool and for any non-default pages size buffer pools. However, if you create a dbspace with a non-default page size, the dbspace must have a corresponding buffer pool. For example, if you create a dbspace with a page size of 8 KB, you must create a buffer pool with a page size of 8 KB. The BUFFERPOOL onconfig parameter can be useful to reduce the number of buffers and force IDS to use the light scan. For a DSS environment, you can set the buffers to a low number, for example 5000.

```
BUFFERPOOL  
size=8K,buffers=5000,lrus=8,lru_min_dirty=50,lru_max_dirty=60
```


► PDQ

Another key factor in DSS queries is to read the pages in parallel. To do this, you must activate the PDQ. There are primarily six variables that enable you control PDQ:

– PDQPRIORITY

This environment variable sets a reasonable or recommended priority value.

– MAX_PDQPRIORITY

This variable limits the PDQ resources that the database server can allocate to any one DSS query. MAX_PDQPRIORITY is a factor that is used to scale the value of PDQ priority set by users.

– DS_TOTAL_MEMORY

This variable specifies the amount of memory available for PDQ queries.

– DS_MAX_SCANS

This variable limits the number of PDQ scan threads that the database server can execute concurrently.

– DS_MAX_QUERIES

This variable specifies the maximum number of queries that can run concurrently.

– DS_NONPDQ_QUERY_MEM

This variable increases the amount of memory that is available for a query that is not a PDQ.

The following formulas are used in PDQ. A good understanding of these formulas can help to find the best setting for the PDQ parameters for your environment:

– Memory quantum

Memory is granted in units called a *quantum*. A *quantum unit* is the minimum increment of memory that is allocated to a query. The memory quantum is calculated by using the following formula:

$$\text{memory quantum} = \text{DS_TOTAL_MEMORY} / \text{DS_MAX_QUERIES}$$

– Minimum amount of decision-support memory

When you assign a value to the configuration parameter DS_MAX_QUERIES, the database server sets the minimum amount of decision-support memory according to the following formula:

$$\text{min_ds_total_memory} = \text{DS_MAX_QUERIES} * 128 \text{ KB}$$

When you do not assign a value to DS_MAX_QUERIES, the database server uses the following formula instead, which is based on the value of VPCLASS cpu or NUMCPUVPS:

$$\text{min_ds_total_memory} = \text{NUMCPUVPS} * 2 * 128 \text{ KB}$$

– Resources allocated

When a query requests a percentage of PDQ resources, the database server allocates the MAX_PDQPRIORITY percentage of the amount requested, as the following formula shows:

$$\text{Resources allocated} = (\text{PDQPRIORITY}/100) * (\text{MAX_PDQPRIORITY}/100)$$

– Memory for query

The amount of memory that is granted to a single parallel database query depends on many system factors. However, in general, the amount of memory granted to a single parallel database query is proportional to the following formula:

$$\text{memory_grant_basis} = (\text{DS_TOTAL_MEMORY}/\text{DS_MAX_QUERIES}) * (\text{PDQPRIORITY}/100) * (\text{MAX_PDQPRIORITY}/100)$$

– Maximum number of scan threads per query

You can limit the number of concurrent scans by using the DS_MAX_SCANS. In fact, the resources that users can assign to a query are calculated by the following formula:

$$\text{scan_threads} = \min(\text{nfrags}, (\text{DS_MAX_SCANS} * (\text{pdqpriority}/100) * (\text{MAX_PDQPRIORITY}/100))$$

In this formula:

- *nfrags* is the number of fragments in the table with the largest number of fragments.
- *pdqpriority* is the PDQ priority value that is set by either the PDQPRIORITY environment variable or the SET PDQPRIORITY statement.

– Amount of shared memory for PDQ

Use the following formula as a starting point for estimating the amount of shared memory to allocate to decision-support queries:

$$\text{DS_TOTAL_MEMORY} = \text{p_mem} - \text{os_mem} - \text{rsdnt_mem} - (128 \text{ KB} * \text{users}) - \text{other_mem}$$

In this formula:

- *p_mem* is the total physical memory that is available on the host computer.
- *os_mem* is represents the size of the operating system, including the buffer cache.
- *resdnt_mem* represents the size of Informix-resident shared memory.
- *users* is the number of expected users (connections) specified in the NETTYPE configuration parameter.
- *other_mem* is the size of memory used for other (non-IBM Informix) applications.

For more information, see the *IBM Informix Dynamic Server Performance Guide*, G229-6385.

In general, as a starting point for DSS environment, we set the values shown in Table 2-2.

Table 2-2 PDQ values for DSS

Parameter name	Value
PDQPRIORITY	100
MAX_PDQPRIORITY	100
DS_TOTAL_MEMORY	90% of SHMVIRTSIZE
DS_MAX_SCAN	Usually left as the default value

You can monitor the PDQ behavior by using the `onstat -g mgm` command.

PDQ queries use memory from the Virtual Shared Memory segments, not from the BUFFERS.

► **DBSPACETEMP**

This variable defines more DBSPACETEMP to allow parallelism. Also consider how much additional space is needed. For example, hash joins can use a significant amount of memory and can potentially overflow to temporary space on disk. You can use the following formula to estimate the amount of memory that is required for the hash table in a hash join:

$$\text{hash_table_size} = (32 \text{ bytes} + \text{row_size}) * \text{num_rows_table}$$

► **Fragmentation**

There are many considerations when fragmenting. For example, you must understand the workload and then consider how to fragment both the table and the indexes based on that workload.

For more information, refer to the *IBM Informix Database Design and Implementation Guide*, G251-2271. Also see the “Fragmentation guidelines” chapter in the *IBM Informix Dynamic Server Performance Guide*, G251-2296, for a discussion on how to plan, design, and execute a fragmentation scheme. In addition, see the *IBM Informix Guide to SQL: Syntax*, G229-6375.

When implemented, you can use the **onstat -D** command output to see the I/O workload on each of the fragments. The objective is to see balanced I/O across all fragments in the table. At the system level, you can use **sar -d**, or a similar utility, to monitor the I/O activity.

► Data load activity

For more information about the database server and high-performance loading, see the *IBM Informix High-Performance Loader User's Guide*, G229-6377. The High Performance Loader (HPL) offers two load modes, deluxe mode and express mode. *Express mode* uses special buffers called *light append buffers*, and can be faster. *Deluxe mode* is more flexible, but uses normal buffers and can be slower than express mode.

– Nonlogging tables

Alter a table from logging to nonlogging. The two table types are STANDARD (logging tables) and RAW (nonlogging tables). You can use any loading utility, such as **dbimport** or HPL, to load raw tables.

The advantage of nonlogging tables is that you can load large data warehousing tables quickly because they do not use CPU and I/O resources for logging. They avoid problems, such as running out of logical log space, and they are locked exclusively during an express load, so that no other user can access the table during the load.

To alter a table from logging to nologging mode, execute the following command:

```
ALTER TABLE tablename TYPE(RAW)
```

To alter a table from nonlogging to logging mode, execute the following command:

```
ALTER TABLE tablename TYPE(STANDARD)
```

► Network

In client/server communications, use a network protocol together with a network programming interface to connect and transfer data between the client and the database server. Often in a DSS environment, you can have applications that connect to the database server and run a set of jobs to elaborate the data contained in tables. Those applications can reside in the same machine where the database server resides. This can reduce the network time to transfer the data from one machine to another.

In this scenario, you can consider the possibility to create a kind of connection between the application and the server that can provide maximum speed. IDS supports several connection types, but the ones that provide fast access are shared memory (UNIX or Linux) and the named pipe (Windows). These connections provide fast access to a database server, but can pose some security risks.

In an OLTP environment, we suggest that you use TCP/IP connections for security reasons. For more information related to the networking and security, see the *IBM Informix Dynamic Server Administrator's Guide*, G229-6359.

2.3.4 Configuring for OLTP

In this section, we discuss the primary factors that can impact the OLTP environment to be considered for achieving maximum performance.

Keep in mind that, in an OLTP environment, you want to accomplish the following tasks:

- ▶ Tune the `onconfig` parameters to have fast response time.
- ▶ Read and write buffer cache rates above 90%.
- ▶ Take fast checkpoints.
- ▶ Have a short recovery time objective (RTO).
- ▶ Maximize I/O throughput.
- ▶ Optimize fragmentation strategy.
- ▶ Optimize index utilization.

We look at the primary OLTP factors detail in the sections that follow.

Onconfig parameters and performance goals

In an OLTP environment, we want to achieve the goals as described in Table 2-2 on page 41. The `onconfig` parameters have a key role in achieving those goals. Performance tuning is needed to find the values that provide the best performance for your environment.

Here we describe some of the important parameters that are involved in the OLTP configuration. However, this section is not about performance tuning and that topic is not included in this book. For information about performance tuning, see the *IBM Informix Dynamic Server Performance Guide*, G229-6385, and the *IBM Informix Dynamic Server Administrator's Reference*, G229-6360. For some of the `onconfig` parameters, we provide an initial value, but this does not mean they are the most suitable values for your particular implementation.

► Cache rate

The cache read rate should be above 90%. To obtain this goal simply increase the BUFFERS. That is, after making sure that there is also sufficient memory available. The objective is to find the point where adding more buffers no longer has an impact on the performance, while removing buffers will impact the performance. Use **onstat -p** to monitor the read and write the buffer cache rate.

► LRUs

The LRU field specifies the number of least recently used (LRU) queues in the shared-memory buffer pool. You can tune the value of LRUs, in combination with the LRU_MIN_DIRTY and LRU_MAX_DIRTY fields, to control how frequently the shared-memory buffers are flushed to disk. Setting LRUs too high might result in excessive page-cleaner activity.

- LRU_MAX_DIRTY specifies the percentage of modified pages in the LRU queues at which the queue is cleaned.
- LRU_MIN_DIRTY specifies the percentage of modified pages in the LRU queues at which page cleaning is no longer mandatory.

The following vales can be considered as typical initial values:

- 1 LRU for every 5000 buffers or 4 LRUs for each CPU virtual processor
- LRU_MAX_DIRTY = 10
- LRU_MIN_DIRTY = 5

Set the onconfig parameter AUTO_LRU_TUNING to 1, which enables IDS to tune the LRUs automatically, and use the **onstat -R** command to monitor LRU activity.

► CLEANERS

This parameter specifies the number of page-cleaner threads that are available. A typical suggested value is LRUs/2.

► LOCKS

This parameter specifies the initial size of the lock table. The lock table holds an entry for each lock that is used by a session. If the number of locks that sessions allocate exceeds the value of LOCKS, the database server increases the size of the lock table. The initial value is set as follows:

$250 * \text{number_of_users}$ during the peak time

► RA_PAGES

This parameter specifies the number of disk pages to attempt to read ahead during sequential scans of data records. Read-ahead can greatly speed up database processing by compensating for the slowness of I/O processing relative to the speed of CPU processing. A suggested initial value is 32.

- ▶ **RA_THRESHOLD**
This parameter is used with **RA_PAGES** when the database server reads during sequential scans of data records. **RA_THRESHOLD** specifies the read-ahead threshold. That is, the number of unprocessed data pages in memory that signals the database server to perform the next read-ahead. A suggested initial value 30 pages.
- ▶ **SHMVIRTSIZE**
This parameter specifies the initial size of a virtual shared-memory segment. A suggested initial value is:
 $32000 + \text{expected number of users} * 800$
- ▶ **CKPTINTVL**
This parameter specifies the frequency, expressed in seconds, at which the database server checks to determine whether a checkpoint is needed. A suggested initial value is 120.
- ▶ **LOGBUFF**
This parameter specifies the size in KB for the three logical-log buffers in shared memory. If you log user data in smart large objects, increase the size of the log buffer to make the system more efficient. If you set **LOGBUFF** too high, the database server can run out of memory and shut down during recovery. Check the logical-log section of the **onstat -l** command output to tune the correct value. A suggested initial value is 256.
- ▶ **PHYSBUFF**
This parameter specifies the size in KB of the two physical log buffers in shared memory. Check the physical log section of the **onstat -l** command output to tune the correct value. A suggested initial value is 512.
- ▶ **RTO_SERVER_RESTART**
This parameter enables use of RTO standards to set the amount of time, in seconds, that IDS has to recover from a problem after being restarted, and brings the server into an online or quiescent mode. For OLTP, a low value is suggested, such as from 60 to 90 seconds.
- ▶ **AUTO_CKPTS**
This parameter allows the server to trigger checkpoints more frequently to avoid transaction blocking. A suggested value is 1.
- ▶ **AUTO_AIOVPS**
This parameter enables the database server to automatically increase the number of AIO VPs and page cleaner threads when the database server detects that the I/O workload has outpaced the performance of the existing AIO VPs. A suggested value is 1.

- ▶ **AUTO_LRU_TUNING**

This parameter enables automatic LRU tuning. A suggested value is 1.

For more information, see the following manuals:

- ▶ *IBM Informix Dynamic Server Performance Guide*, G229-6385
- ▶ *IBM Informix Dynamic Server Administrator's Reference*, G229-6360

Fragmentation

Fragmentation is a data distribution scheme used by the database server to distribute rows or index entries to data fragments. The expression-based distribution schemes put rows that contain particular specified values in the same fragment. A fragmentation expression defines the criteria for assigning a set of rows to each fragment, either as a range rule or some arbitrary rule. A remainder fragment can be specified that holds all rows that do not match the criteria for any other fragment, although a remainder fragment reduces the efficiency of the expression-based distribution scheme.

In an OLTP environment, expression-based distribution schemes are typically used for the selection of rows. By using the expression, IDS identifies the fragments that contain the rows that are involved in the query and then uses the index to search inside the fragment. If the query does not validate the expression of the distribution schemes, all the fragments must be scanned. For more information and references, see the fragmentation discussion in the bulleted list on page 41.

Index utilization

Typically the queries involved in the OLTP environment do not request a scan of the entire table. Instead indexes are typically used to select the rows that are needed to process the query. In large OLTP environments the database administrator analyzes the tables when they are created. However, it can be difficult to continue monitoring their usage. Therefore, new users or new applications can query the tables by using a different WHERE condition that is not yet optimized. This action can generate sequential scans, which in some circumstances, can create significant performance problems because it increases both the number of pages that are read from disk and the number of locks.

OLTP performance can be increased by removing the sequential scans using the following methods:

- ▶ Use the **onstat -p** command to check the value of the seqscans field. If seqscans has a high value, say more than 200, you must investigate and determine which tables have a high number of sequential scans.

- ▶ To find the name of the tables that have a high number of sequential scans, execute a select from the sysmaster database as described in Example 2-2. This select only shows the tables with more than 200 sequential scans that were performed since the database instance was started or since the last **onstat -z** command was executed.

Example 2-2 Finding tables with sequential scans

From dbaccess, connect to sysmaster then execute the following select:

```
SELECT
dbsname,tabname,b.partnum,pf_dskreads,pf_dskwrites,pf_seqscans FROM
systabnames as a, sysptntab as b WHERE pf_seqscans > 200 AND
a.partnum=b.partnum
```

output:

dbsname	database_name
tabname	table_name
partnum	2097154
pf_dskreads	265432456
pf_dskwrites	543678954
pf_seqscans	34000

- ▶ After you identify a table with a high number of sequential scans, monitor the activity on the table to determine if there are missing indexes. You can do this by using the OAT as described in the list item “SQL Trace” on page 202 of Chapter 5, “The administration free zone” on page 161.

This information offers good suggestions of where to focus to achieve the maximum OLTP performance.

Important: At times, it can be more efficient to run a sequential scan rather than using the index. This is particularly true for small tables. You can create a index and then, by using the optimizer directives, you can modify the behavior of the optimizer and force IDS to use, or avoid, the index. Do some testing and compare results for the two methods to determine which works best in your particular implementation.

2.3.5 Dynamically changing environments

In this section, we explore the actions to take to dynamically change environments, without bouncing the instance. In this sample scenario, we consider changing between OLTP and DSS environments.

To do this, the **onconfig** parameters that must be changed to pass from one environment to the other must be identified, and they should be placed in one of the following categories:

- ▶ Dynamically changeable parameters *can* be changed dynamically by using **onmode** commands. Examples are the PDQ parameters of RTO_SERVER_RESTART, AUTO_CKPTS, AUTO_AIOVPS, and AUTO_LRU_TUNING.
- ▶ Static parameters *cannot* be changed dynamically. Examples are RA_PAGES, RA_THRESHOLD, BUFFERS, LOCKS, and CLEANERS. For these parameters, find mid-point values that can provide the best performance for both environments.

One important consideration is the source of data to which users need access. For example, consider a telecommunications company that saves all the customer call detail, such as phone number, call start and finish time, duration, location, and so on. Some users might need access to all the detail, but many others might not. They might only need a summarized version of the data, which in this scenario, might include data such as the total number of calls per day, average number of calls per hour, average duration of a call, and so on. This is true, for example, when considering OLTP and DSS users.

For a scenario that includes OLTP and DSS users, data tables might be categorized into the following three groups:

- ▶ The DSS group contains aggregate tables for the DSS users. For this category, create the tables in dbspaces with large page sizes.
- ▶ The OLTP group contains tables with detailed data for the OLTP users that contain the detailed data. Create these tables in dbspaces with a smaller page size, such as 2k or 4k.
- ▶ The OLTP and DSS tables are used by both OLTP and DSS users. Create the tables in dbspaces with a page size somewhere between the sizes of the OLTP and DSS dbspaces.

This classification of the tables gives the possibility to use different dbspaces and different BUFFERPOOL sizes, and to set parameters based on the results of the performance tuning activity.

For example, consider a 32-bit operating system with 2,000 pages. The onconfig file can contain several different BUFFERPOOL configurations, for example:

- ▶ For DSS:
BUFFERPOOL size=6k,buffers=2000, lrus=2, lru_min_dirty=60,
lru_max_dirty=50

- ▶ For OLTP:
BUFFERPOOL size=2k, buffers=500000, lrus=100, lru_min_dirty=10,
lru_max_dirty=5
- ▶ For OLTP and DSS:
BUFFERPOOL size=4k, buffers=10000, lrus=8, lru_min_dirty=40,
lru_max_dirty=30

The values of the BUFFERPOOL parameters can be changed based on the results of the performance tuning activities. For example, you can specify different buffers, LRUs, LRU_MIN_DIRTY, and LRU_MAX_DIRTY. Each page can contain a maximum of 255 rows, and the BUFFERPOOL size must be a multiple of the page size of the operating system. Therefore, if the row size is small and the BUFFERPOOL size is large, there can be a loss of disk space. To check this, use the **oncheck** command to obtain the maximum row size:

```
oncheck -pt databasename:tablename
```

Now we look at how to dynamically switch between environments.

Dynamically changing from OLTP to DSS

Suppose that there is a requirement at midnight every night to change from the OLTP environment to the DSS environment. Such a change requires changing some of the **onconfig** parameters, as in the following list:

- ▶ Activate the PDQ and change the following parameters:
 - MAX_PDQPRIORITY from 0 to 100
 - DS_TOTAL_MEMORY from 0 to 500000
 - DS_MAX_SCAN from 10 to 50
 - DS_MAX_QUERIES from 1 to 50
- ▶ Change the following parameters:
 - RTO_SERVER_RESTART from 60 to 1200
 - AUTO_CKPTS from 1 to 0
 - AUTO_AIOVPS from 1 to 0
 - AUTO_LRU_TUNING from 1 to 0

To change dynamically from OLTP to DSS, consider Example 2-3 on page 50. In this example, we use generic values that must be changed for your environment. The appropriate values that are used can be determined from your performance tuning activity.

The first step is to create a file named *oltp2dss.sql*. In that file, place the SQL script that is shown in Example 2-3.

```
database sysadmin;

INSERT INTO ph_task(
tk_name,tk_type,tk_group,tk_description, tk_execute,
tk_next_execution,tk_start_time, tk_stop_time,tk_frequency)
VALUES(
"oltp2dss_RTO","TASK","SERVER", "change RTO for DSS",
"execute function admin('ONMODE', 'wf', 'RTO_SERVER_RESTART=1200');",
DATETIME(00:00:00) HOUR TO SECOND, DATETIME(00:00:00) HOUR TO SECOND,
NULL, INTERVAL ( 1 ) DAY TO DAY
);

INSERT INTO ph_task(
tk_name,tk_type,tk_group,tk_description, tk_execute,
tk_next_execution,tk_start_time, tk_stop_time,tk_frequency)
VALUES(
"oltp2dss_CKPTS","TASK","SERVER", "change AUTO_CKPTS for DSS",
"execute function admin('ONMODE', 'wf', 'AUTO_CKPTS=0');",
DATETIME(00:00:00) HOUR TO SECOND, DATETIME(00:00:00) HOUR TO SECOND,
NULL, INTERVAL ( 1 ) DAY TO DAY
);

INSERT INTO ph_task(
tk_name,tk_type,tk_group,tk_description, tk_execute,
tk_next_execution,tk_start_time, tk_stop_time,tk_frequency)
VALUES(
"oltp2dss_LRU","TASK","SERVER", "change AUTO_LRU_TUNING for DSS",
"execute function admin('ONMODE', 'wf', 'AUTO_LRU_TUNING=0');",
DATETIME(00:00:00) HOUR TO SECOND, DATETIME(00:00:00) HOUR TO SECOND,
NULL, INTERVAL ( 1 ) DAY TO DAY);

INSERT INTO ph_task(
tk_name,tk_type,tk_group,tk_description, tk_execute,
tk_next_execution,tk_start_time, tk_stop_time,tk_frequency)
VALUES(
"oltp2dss_VPS","TASK","SERVER", "change AUTO_AIOVPS for DSS",
"execute function admin('ONMODE', 'wf', 'AUTO_AIOVPS=0');",
DATETIME(00:00:00) HOUR TO SECOND, DATETIME(00:00:00) HOUR TO SECOND,
NULL, INTERVAL ( 1 ) DAY TO DAY);

INSERT INTO ph_task(
tk_name,tk_type,tk_group,tk_description, tk_execute,
tk_next_execution,tk_start_time, tk_stop_time,tk_frequency)
```

```
VALUES(
"oltp2dss_MaxPDQ","TASK","SERVER", "change MAX_PDQPRIORITY for DSS",
"execute function admin('ONMODE', 'D', '100');",
DATETIME(00:00:00) HOUR TO SECOND, DATETIME(00:00:00) HOUR TO SECOND,
NULL, INTERVAL ( 1 ) DAY TO DAY);
```

```
INSERT INTO ph_task(
tk_name,tk_type,tk_group,tk_description, tk_execute,
tk_next_execution,tk_start_time, tk_stop_time,tk_frequency)
VALUES(
"oltp2dss_Max_Scan","TASK","SERVER", "change DS_MAX_SCANS for DSS",
"execute function admin('ONMODE', 'S', '50');",
DATETIME(00:00:00) HOUR TO SECOND, DATETIME(00:00:00) HOUR TO SECOND,
NULL, INTERVAL ( 1 ) DAY TO DAY);
```

```
INSERT INTO ph_task(tk_name,tk_type,tk_group,tk_description,
tk_execute,
tk_next_execution,tk_start_time, tk_stop_time,tk_frequency)
VALUES(
"oltp2dss_Max_Queries","TASK","SERVER", "change DS_MAX_QUERIES for
DSS",
"execute function admin('ONMODE', 'Q', '50');",
DATETIME(00:00:00) HOUR TO SECOND, DATETIME(00:00:00) HOUR TO SECOND,
NULL, INTERVAL ( 1 ) DAY TO DAY);
```

```
INSERT INTO ph_task(
tk_name,tk_type,tk_group,tk_description, tk_execute,
tk_next_execution,tk_start_time, tk_stop_time,tk_frequency)
VALUES(
"oltp2dss_DS_TOTAL_MEMORY","TASK","SERVER", "change DS_TOTAL_MEMORY for
DSS",
"execute function admin('ONMODE', 'M', '500000');",
DATETIME(00:00:00) HOUR TO SECOND, DATETIME(00:00:00) HOUR TO SECOND,
NULL, INTERVAL ( 1 ) DAY TO DAY);
```

The SQL script only needs to be executed once. Then the IDS scheduler can execute the command every night at midnight as specified in the ph_task table. To execute the script, from user informix, open a command window and execute the following command:

```
dbaccess - oltp2dss.sql
```

To check the results, from user informix, run **dbaccess** and connect to the database sysadmin. Execute the select as shown in Example 2-4.

Example 2-4 Checking the ph_task table

```
select tk_name,tk_type,tk_group,tk_description, tk_execute,
tk_next_execution,tk_start_time, tk_stop_time,tk_frequency
FROM ph_task WHERE tk_name MATCHES "oltp2dss*";
```

Output

tk_name	oltp2dss_CKPTS
tk_type	TASK
tk_group	SERVER
tk_description	change AUTO_CKPTS for DSS
tk_execute	execute function admin('ONMODE', 'wf', 'AUTO_CKPTS=0');
tk_next_execution	2007-11-02 00:00:00
tk_start_time	00:00:00
tk_stop_time	
tk_frequency	1 00:00:00

Double check that the tk_execute and tk_next_execution fields are correct as you specified in your script. Check all the other tasks that you wrote in the script, as shown in Example 2-5.

Example 2-5 Checking the table command_history

```
select * from command_history
```

cmd_number	131
cmd_exec_time	2007-12-31 00:00:00
cmd_user	informix
cmd_hostname	NA
cmd_executed	ONMODE
cmd_ret_status	0
cmd_ret_msg	OK

After the first scheduled execution, check the command_history table for the new tasks that are created. Look at the cmd_ret_status field. If it is different from zero, the command failed.

At this point, you should be able to change dynamically from the OLTP environment to the DSS environment.

Dynamically changing from DSS to OLTP

To change from the DSS to the OLTP environment, follow the same process that is described in “Dynamically changing from OLTP to DSS” on page 49. After performing the performance tuning activities for the OLTP environment, change the values of the parameters.

Instead of using the `oltp2dss.sql` script, use the `dss2oltp.sql` script as described in Example 2-6. Primarily, the following fields need to be changed:

- ▶ `tk_name`
- ▶ `tk_description`
- ▶ `tk_next_execution`
- ▶ `tk_start_time`

Example 2-6 dss2oltp.sql script

```
database sysadmin;
```

```
INSERT INTO ph_task(
tk_name,tk_type,tk_group,tk_description, tk_execute,
tk_next_execution,tk_start_time, tk_stop_time,tk_frequency)
VALUES(
"dss2oltp_RTO","TASK","SERVER", "change RTO for OLTP",
"execute function admin('ONMODE', 'wf', 'RTO_SERVER_RESTART=60');",
DATETIME(07:00:00) HOUR TO SECOND, DATETIME(07:00:00) HOUR TO SECOND,
NULL, INTERVAL ( 1 ) DAY TO DAY
);
```

```
INSERT INTO ph_task(
tk_name,tk_type,tk_group,tk_description, tk_execute,
tk_next_execution,tk_start_time, tk_stop_time,tk_frequency)
VALUES(
"dss2oltp_CKPTS","TASK","SERVER", "change AUTO_CKPTS for OLTP",
"execute function admin('ONMODE', 'wf', 'AUTO_CKPTS=1');",
DATETIME(07:00:00) HOUR TO SECOND, DATETIME(07:00:00) HOUR TO SECOND,
NULL, INTERVAL ( 1 ) DAY TO DAY
);
```

```
INSERT INTO ph_task(
tk_name,tk_type,tk_group,tk_description, tk_execute,
tk_next_execution,tk_start_time, tk_stop_time,tk_frequency)
VALUES(
"dss2oltp_LRU","TASK","SERVER", "change AUTO_LRU_TUNING for OLTP",
"execute function admin('ONMODE', 'wf', 'AUTO_LRU_TUNING=1');",
DATETIME(07:00:00) HOUR TO SECOND, DATETIME(07:00:00) HOUR TO SECOND,
NULL, INTERVAL ( 1 ) DAY TO DAY);
```

```

INSERT INTO ph_task(
tk_name,tk_type,tk_group,tk_description, tk_execute,
tk_next_execution,tk_start_time, tk_stop_time,tk_frequency)
VALUES(
"dss2oltp_VPS","TASK","SERVER", "change AUTO_AIOVPS for OLTP",
"execute function admin('ONMODE', 'wf', 'AUTO_AIOVPS=1');",
DATETIME(07:00:00) HOUR TO SECOND, DATETIME(07:00:00) HOUR TO SECOND,
NULL, INTERVAL ( 1 ) DAY TO DAY);

```

```

INSERT INTO ph_task(
tk_name,tk_type,tk_group,tk_description, tk_execute,
tk_next_execution,tk_start_time, tk_stop_time,tk_frequency)
VALUES(
"dss2oltp_MaxPDQ","TASK","SERVER", "change MAX_PDQPRIORITY for OLTP",
"execute function admin('ONMODE', 'D', '0');",
DATETIME(07:00:00) HOUR TO SECOND, DATETIME(07:00:00) HOUR TO SECOND,
NULL, INTERVAL ( 1 ) DAY TO DAY);

```

```

INSERT INTO ph_task(
tk_name,tk_type,tk_group,tk_description, tk_execute,
tk_next_execution,tk_start_time, tk_stop_time,tk_frequency)
VALUES(
"dss2oltp_Max_Scan","TASK","SERVER", "change DS_MAX_SCANS for OLTP",
"execute function admin('ONMODE', 'S', '10');",
DATETIME(07:00:00) HOUR TO SECOND, DATETIME(07:00:00) HOUR TO SECOND,
NULL, INTERVAL ( 1 ) DAY TO DAY);

```

```

INSERT INTO ph_task(tk_name,tk_type,tk_group,tk_description,
tk_execute,
tk_next_execution,tk_start_time, tk_stop_time,tk_frequency)
VALUES(
"dss2oltp_Max_Queries","TASK","SERVER", "change DS_MAX_QUERIES for
OLTP",
"execute function admin('ONMODE', 'Q', '1');",
DATETIME(07:00:00) HOUR TO SECOND, DATETIME(07:00:00) HOUR TO SECOND,
NULL, INTERVAL ( 1 ) DAY TO DAY);

```

```

INSERT INTO ph_task(
tk_name,tk_type,tk_group,tk_description, tk_execute,
tk_next_execution,tk_start_time, tk_stop_time,tk_frequency)
VALUES(
"dss2oltp_DS_TOTAL_MEMORY","TASK","SERVER", "change DS_TOTAL_MEMORY for
OLTP",
"execute function admin('ONMODE', 'M', '10000');",

```



```
DATETIME(07:00:00) HOUR TO SECOND, DATETIME(07:00:00) HOUR TO SECOND,  
NULL, INTERVAL ( 1 ) DAY TO DAY);
```

The script only needs to be executed once. The IDS scheduler will execute the command every morning at 07:00:00 a.m. as you specified in the `ph_task` table.

At this point, you should be able to dynamically change from the DSS to the OLTP environment.

2.4 Installing the Open Admin Tool

In this section, we briefly describe how to install and configure the OAT and perform such tasks as connecting to an existing instance. For more details about the functionality of OAT, see 3.5.5, “The Open Admin Tool” on page 107, and Chapter 5, “The administration free zone” on page 161.

2.4.1 Preparing for the installation

In this section, we present the typical tasks that required when preparing to install the OAT:

1. Verify the prerequisites:

The OAT for IDS requires that the following products to be already installed and configured:

- A Web server, for example Apache
- IBM I-Connect, CSDK (3.00), or IDS
- PHP 5 compiled with PDO, PDO_SQLITE, GD and SOAP (5.2.2) enabled
- Informix PDO Module

This module is already included in PHP 5.1 or later, and it is automatically enabled when you run the configuration.

2. Download the software.

You can download the software for free from the following Web sites:

- Apache
<http://httpd.apache.org/download.cgi>
- PHP
<http://www.php.net/downloads.php>

- I-Connect and CSDK

<http://www14.software.ibm.com/webapp/download/nochargesearchquery.jsp>

- Informix PDO module

This module is already included in PHP 5.1 or later. For older PHP releases, download the PDO module form the following link:

http://pecl.php.net/package/PDO_INFORMIX/download/

- XAMPP

This is an Apache distribution package that contains Apache, PHP, Mysql, and Perl. You can download it from the following address:

<http://www.apachefriends.org/en/xampp.html/>

3. Install IDS, Client SDK, or I-Connect.

To install IDS, Client SDK, or I-Connect, refer to the installation guide, in the Informix library at the following Web address, for the particular product that is being installed:

<http://www-306.ibm.com/software/data/informix/pubs/library/>

4. Install Apache and PHP.

To install Apache and PHP, you have two choices:

- Use the application named XAMPP, which automatically installs and configures the following packages:
 - Apache
 - PHP
 - Mysql
 - Perl

For more information, see the following Web address:

<http://www.ibm.com/developerworks/linux/library/l-xampp/>

Note: After the installation, you must configure XAMPP to work with Informix.

- Install and configure Apache and PHP individually.

To only install Apache and PHP, refer to the following developerWorks article depending on the operating environment for step-by-step instructions:

- Windows:

<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0607bombardier/>

- UNIX or Linux:

<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0606bombardier/>

Verifying the installation

At this point, you have installed and configured the Web server, PHP, and Informix server or client. Before proceeding with the installation of OAT, you must verify that the Web server, PHP, and Informix can communicate with each other. To verify the installation:

1. Open the Apache configuration file `/apache_install_path/conf/httpd.conf`.
2. Search for the variable `DocumentRoot` to determine which directory is used to search the Web pages.
3. Open the `DocumentRoot` directory and perform the following steps:
 - a. Create a file named `phpinfo.php`.
 - b. In the file, type the following code:

```
<?php phpinfo(); ?>
```
4. Be sure that Apache is running.
5. Open a Web browser and type the *either* of the following URLs in the address bar:

```
http://localhost/phpinfo.php  
http://machine_name/phpinfo.php
```

If Apache and PHP are working properly, you see a page with PHP and Apache. Scroll down the page until you see the Informix section, as shown in Figure 2-7 on page 58, which means that, in PHP, the Informix module is active.

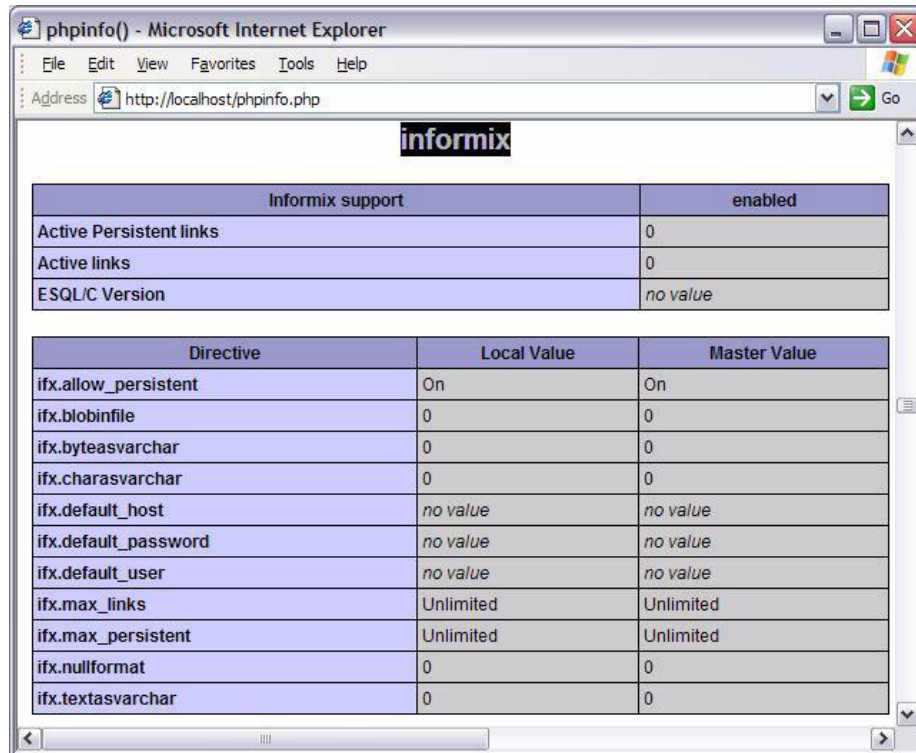


Figure 2-7 PHP Informix module

6. Be sure that the Informix instance is online.
7. Create the stores_demo database.
8. Create a file named select.php and place the code shown in Example 2-7 in the file. Change instance_name, username, and your_password to the correct values.

Example 2-7 Connect and select to IDS by using PHP

```

<?php
$ifxdbname="stores_demo@instance_name";
$ifxdbuser="username";
$ifxpassw="your_password";

$conn_id= ifx_connect($ifxdbname, $ifxdbuser, $ifxpassw);
if ($conn_id) {print("connected successfully");}
else {print ("error: $conn_id");};

$res_id=ifx_query("select * from customer",$conn_id);

```

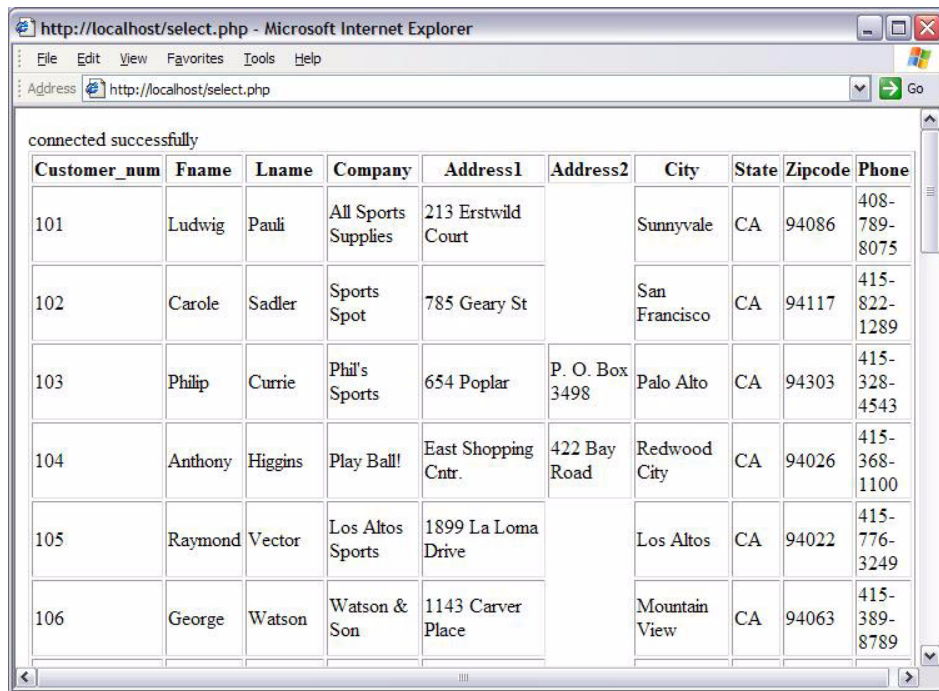
```
ifx_htmltbl_result($res_id,"border=\"1\"");
ifx_free_result($res_id);
?>
```

9. Open a Web browser and enter either of the following URLs:

<http://localhost/select.php>

<http://machinename/select.php>

If everything is working properly, you should see the output as shown in Figure 2-8.



The screenshot shows a Microsoft Internet Explorer window with the address bar set to <http://localhost/select.php>. The page content displays the text "connected successfully" above a table with 10 columns: Customer_num, Fname, Lname, Company, Address1, Address2, City, State, Zipcode, and Phone. The table contains six rows of customer data.

Customer_num	Fname	Lname	Company	Address1	Address2	City	State	Zipcode	Phone
101	Ludwig	Pauli	All Sports Supplies	213 Erstwild Court		Sunnyvale	CA	94086	408-789-8075
102	Carole	Sadler	Sports Spot	785 Geary St		San Francisco	CA	94117	415-822-1289
103	Philip	Currie	Phil's Sports	654 Poplar	P. O. Box 3498	Palo Alto	CA	94303	415-328-4543
104	Anthony	Higgins	Play Ball!	East Shopping Cntr.	422 Bay Road	Redwood City	CA	94026	415-368-1100
105	Raymond	Vector	Los Altos Sports	1899 La Loma Drive		Los Altos	CA	94022	415-776-3249
106	George	Watson	Watson & Son	1143 Carver Place		Mountain View	CA	94063	415-389-8789

Figure 2-8 Select to stores demo

You have now verified that the Web server, PHP, and Informix are installed and configured properly. Proceed with installing the OAT.

- c. If the registration is successful, you receive a validation message like the example in Figure 2-10. Click **Continue**.

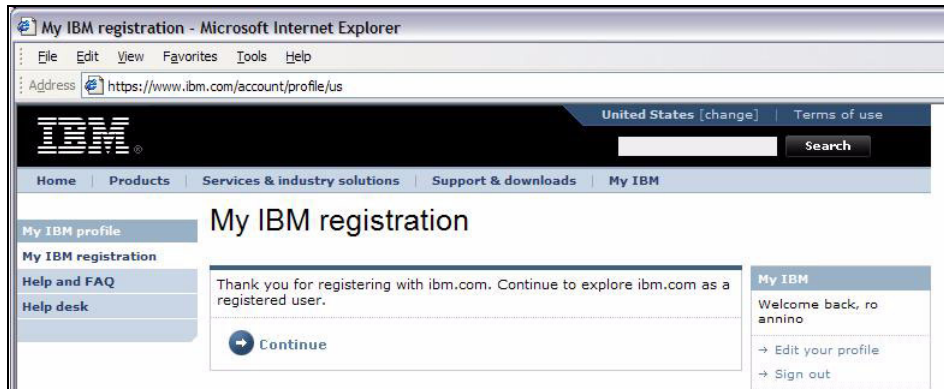


Figure 2-10 Registration confirmation message

2. On the IBM Informix Free Product Download main page (Figure 2-11), scroll down to the IBM Informix Open Admin section. Select the OS version and documentation that you want to download, and click **Download Now** at the bottom of the page.

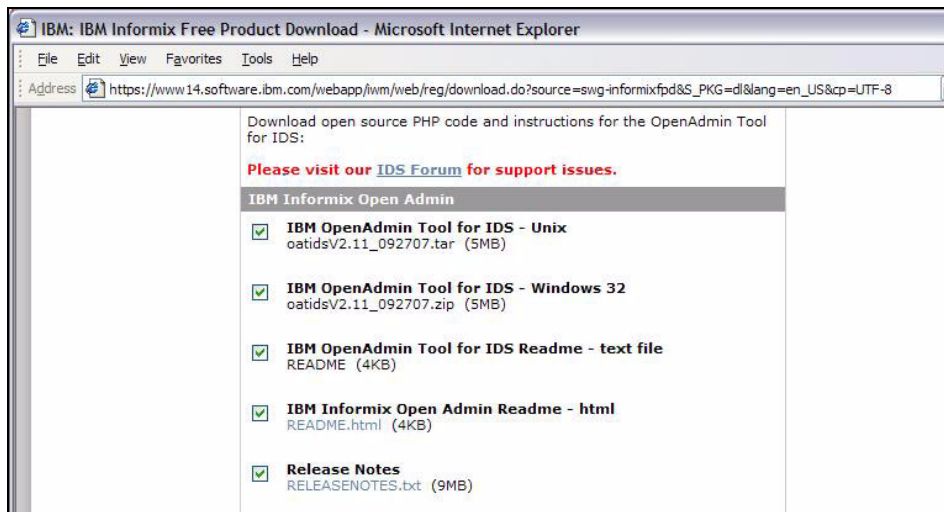


Figure 2-11 OAT download form

3. A Java applet starts and prompts you for the location to download the file (Figure 2-12).

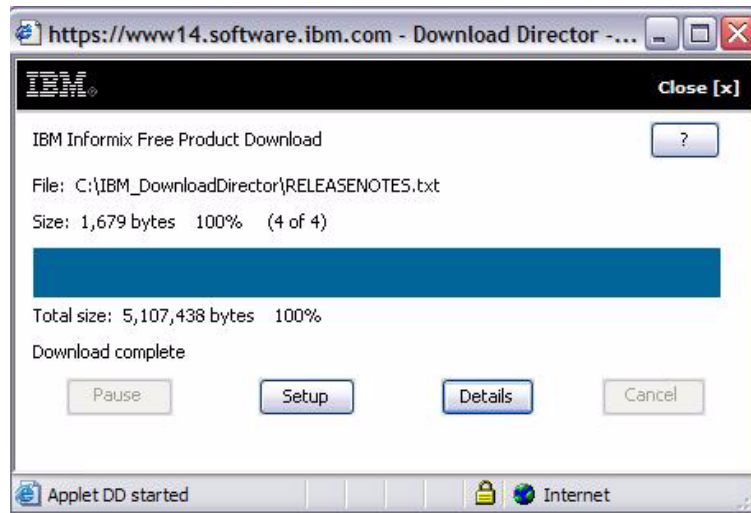


Figure 2-12 Java applet

If you click the button **Detail**, you can see the file downloaded, the location, and size, as shown in Figure 2-13.

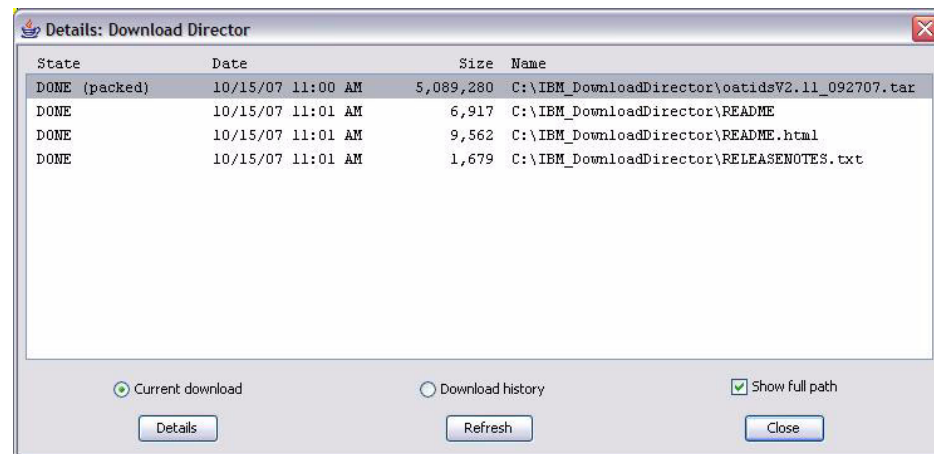


Figure 2-13 Details download directory

2.4.3 Installing the Open Admin Tool

To install the OAT:

1. In the Web server root directory, create a new subdirectory named OAT. Extract the OAT package into the /web server document root directory/OAT/.
2. Change the ownership of the OAT/install directory to the user and group that runs the Apache (httpd) server.

For the installation and configuration, use the local host for all the examples that involve an action with Apache. However, you can use the machine name instead of the local host in all the commands described in this section, and the results will be the same.

1. Open the Web browser and type the following URL in the address bar:
`http://localhost/oat/install/`
2. On the installation Welcome Page (Figure 2-14), select the **I accept the terms in the license agreement** check box and click **Next**.

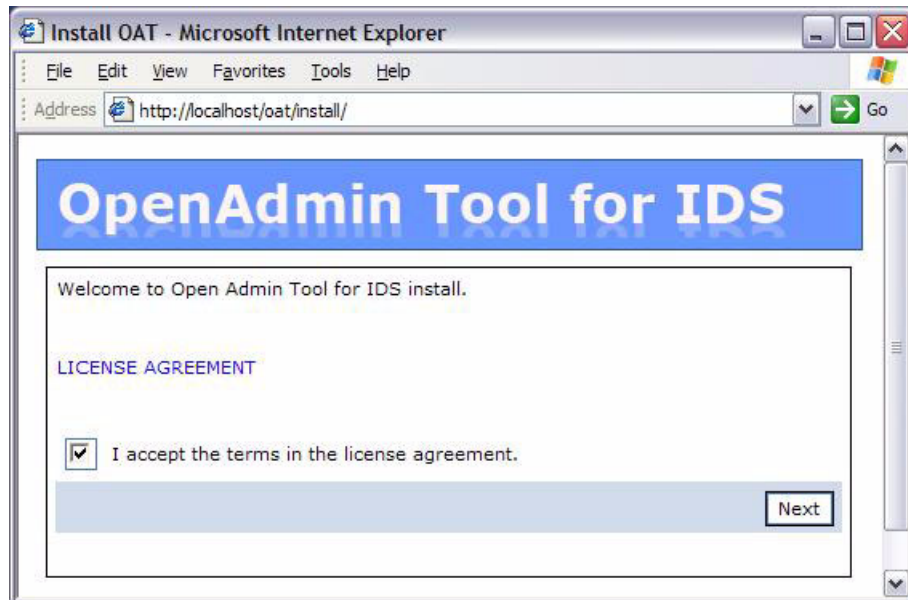


Figure 2-14 Welcome to Open Admin Tool for IDS install

- OAT checks that the PHP modules that are required are correctly installed. The Required PHP Modules page (Figure 2-15) shows the OAT check module results. You should have a page similar to this one. A red “X” in one or more PHP modules indicates a problem that must be corrected to proceed with the OAT installation. Click **Next** to continue the installation.

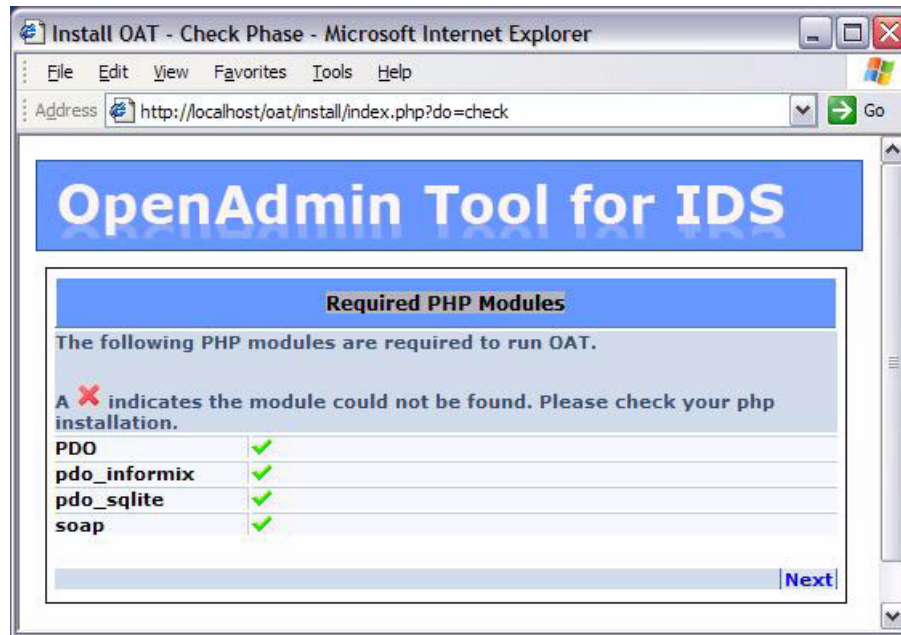


Figure 2-15 Required PHP Modules

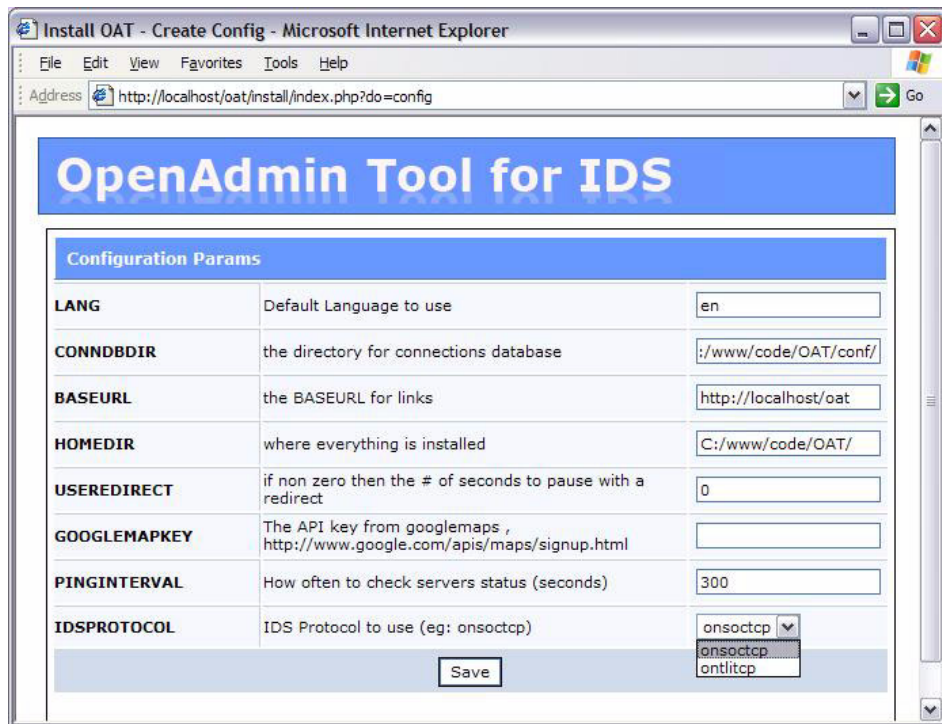
- The Configuration Parameters Web Page is displayed as shown in Figure 2-16 on page 65. On this page, you can change some of the configuration parameters, such as language, the directory where the connection database resides, the OAT home directory, and connection protocol. Change the parameter as required, or keep the default values.

Googlemapkey

For a graphical representation of the location of the server, complete the GOOGLEMAPKEY field (Figure 2-16). To begin, you must sign in at the following Web site:

<http://www.google.com/apis/maps/signup.html>

The signup process provides a key that you can enter in the GOOGLEMAPKEY field. From there, you can determine, for example, the specific latitude and longitude of the server. When you finish, click **Save**.



Configuration Params		
LANG	Default Language to use	en
CONNDBDIR	the directory for connections database	:/www/code/OAT/conf/
BASEURL	the BASEURL for links	http://localhost/oat
HOMEDIR	where everything is installed	C:/www/code/OAT/
USEREDIRECT	if non zero then the # of seconds to pause with a redirect	0
GOOGLEMAPKEY	The API key from googlemaps , http://www.google.com/apis/maps/signup.html	
PINGINTERVAL	How often to check servers status (seconds)	300
IDSPROTOCOL	IDS Protocol to use (eg: onsoctcp)	onsoctcp

Save

Figure 2-16 Configuration parameters

Creating the connection database

Create the connection database that contains all the information to connect to the IDS instances:

1. On the Create connections database page (Figure 2-17), click **Next**.

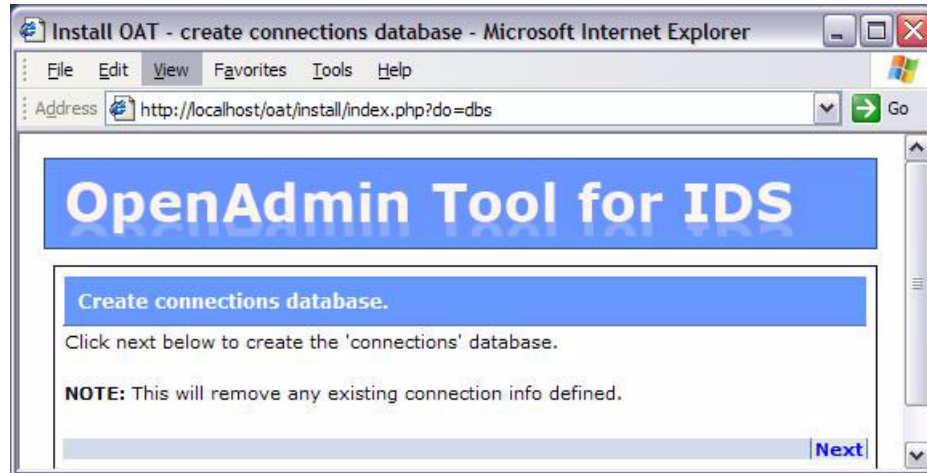


Figure 2-17 Create connections database

2. It takes few seconds to create the connection database. OAT shows a message that informs you whether the database creation was successful. A page shows a message indicating that the OAT database was created successfully such as the example in Figure 2-18. Click **Next**.

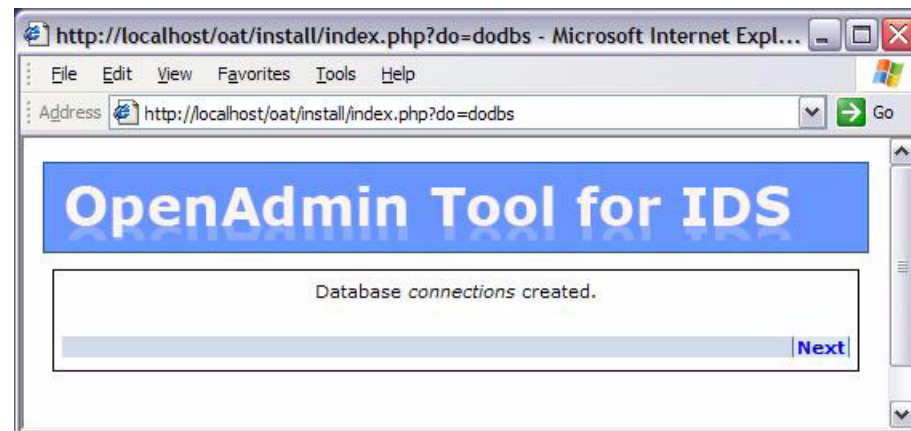


Figure 2-18 Database connection created

3. A page shows a message indicating that the installation of OAT completed successfully such as the example in Figure 2-19. Click **HERE** to start the OAT configuration.

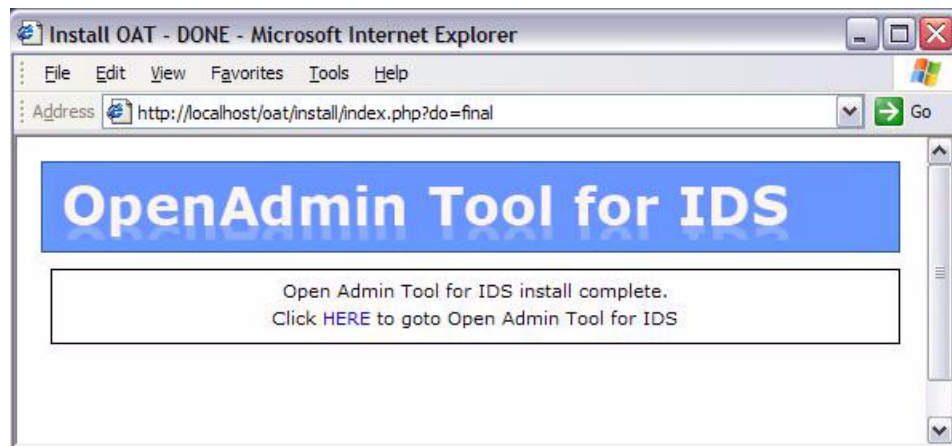


Figure 2-19 Installation completed

2.4.4 Configuring the installation

In this section, we explain how to configure OAT to open a connection to an IDS instance by performing the following tasks:

1. Open the OAT configuration page.
2. Add a group, but only the first time.
3. Add a connection to an IDS instance.
4. Connect from OAT to the IDS instance.
5. Open the OAT main page to manage the instance.

Let us look at each of these steps in more detail.

Opening the OAT configuration page

Open a Web browser and enter the following link:

<http://localhost/oat/admin/>

The OAT Admin page opens (Figure 2-20) on which you can perform the following actions:

- ▶ Change the OAT configuration parameters (refer to Figure 2-16 on page 65).
- ▶ Add a new OAT administrator group.
- ▶ Add new connections to IDS instances.
- ▶ Associate a location map to the IDS instances.
- ▶ Connect to an IDS instance.

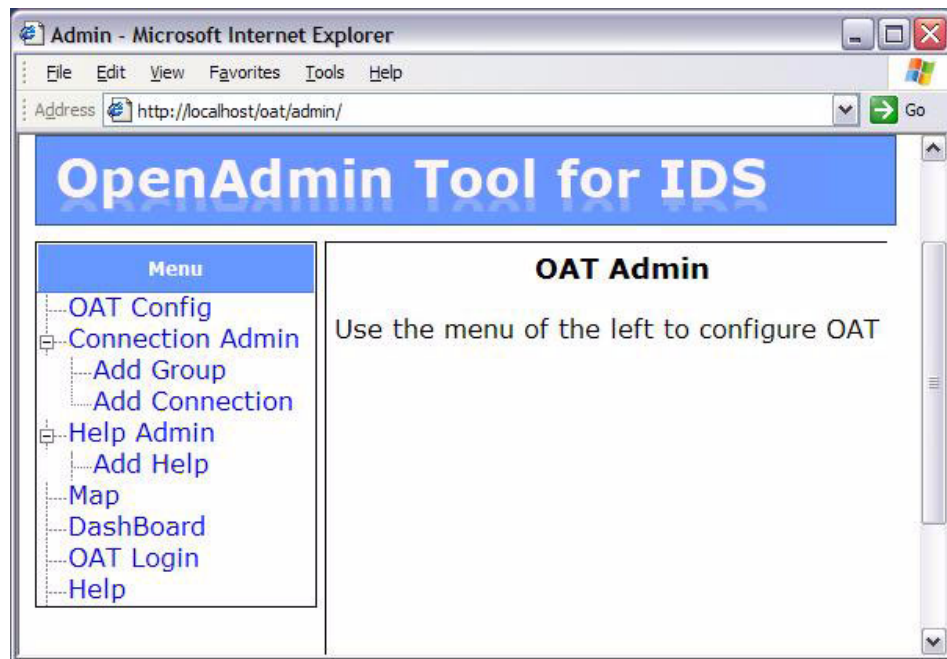


Figure 2-20 OAT configuration - Main page

Adding a new group

To add a new group, on the OAT Admin page, under Menu, click **Add Group**. The Add a Group page (Figure 2-21) is displayed. Add the administrator group name and password and click **Add**. Take note of the group name and password, which are required when you want to add a new connection.

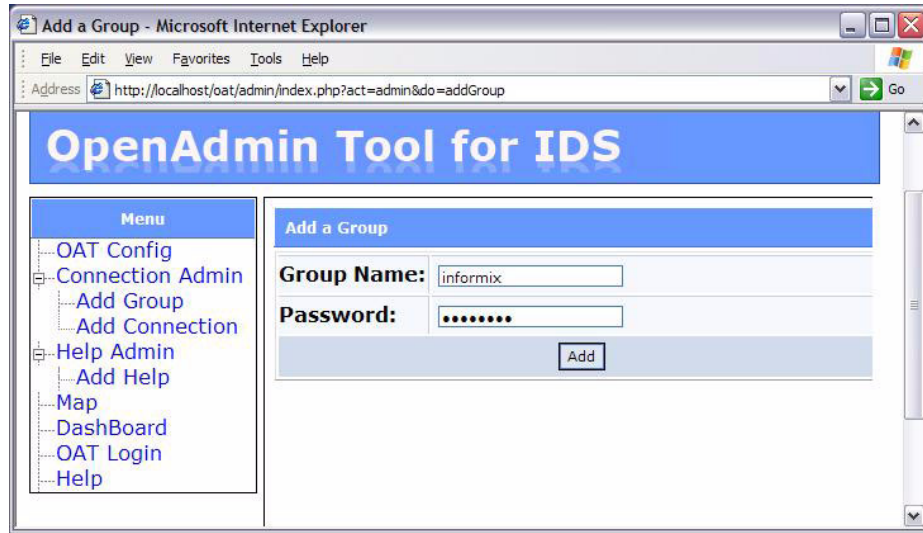
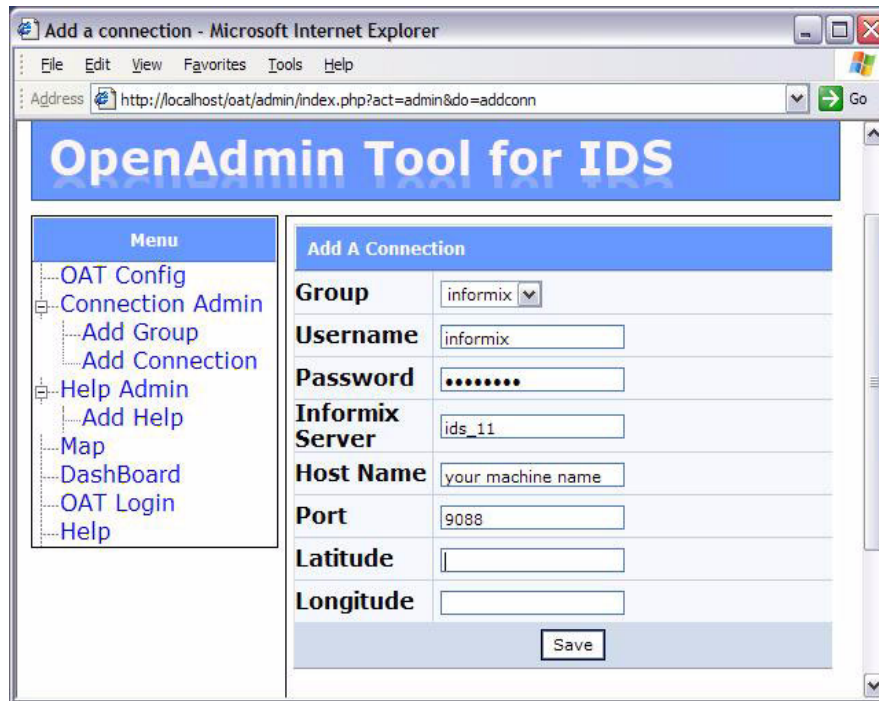


Figure 2-21 OAT configuration - Adding a new group

Adding a connection to the IDS instance

To create a new connection, under Menu, click **Add Connection**. On the Add a Connection page (Figure 2-22), complete the form and click **Save**.



The screenshot shows a web browser window titled "Add a connection - Microsoft Internet Explorer". The address bar shows the URL: `http://localhost/oat/admin/index.php?act=admin&do=addconn`. The page header is "OpenAdmin Tool for IDS". On the left is a "Menu" with options: OAT Config, Connection Admin (expanded), Add Group, Add Connection, Help Admin, Add Help, Map, DashBoard, OAT Login, and Help. The main content area is titled "Add A Connection" and contains a form with the following fields: Group (dropdown menu with "informix" selected), Username (text input with "informix"), Password (password input with 7 dots), Informix Server (text input with "ids_11"), Host Name (text input with "your machine name"), Port (text input with "9088"), Latitude (empty text input), and Longitude (empty text input). A "Save" button is located at the bottom right of the form.

Figure 2-22 OAT configuration - Adding a new connection

To find the Host Name, connect to the IDS server and enter the following command:

```
hostname
```

To find the port number, connect to the IDS server, open the file services, and search for the instance name. Depending on the operating environment, it can be found in either of the following directories depending on your platform:

- ▶ For Windows, in `C:\WINDOWS\system32\drivers\etc\services`
- ▶ For UNIX or Linux, in `/etc/services`

You should see the information as shown in Table 2-3.

Table 2-3 File services

Portname	Portnumber	Protocol	Comment
svc_custom	9088	tcp	#instance_name

The port number is in the second column. Sometimes the port name and port number used by the instance are not written in the file services. In this case, enter the following command:

```
onstat -g ntt
```

Then you see the output that is shown in Example 2-8.

Example 2-8 onstat -g ntt

```
IBM Informix Dynamic Server Version 11.10.TC1    -- On-Line -- Up 00:01:41 -- 21696 Kbytes
```

```
global network information:
```

```
#netscb connects    read    write  q-limits  q-exceed alloc/max
7/ 7      0      0      0/ 45    10/ 0    0/ 0
```

```
Individual thread network information (times):
```

```
netscb thread name  sid    open    read    write address
ccdf728              18 17:47:28
cc60bf8              17 17:47:28
cbc7c48              16 17:47:27
cc5d8a0              15 17:47:27
c939a28 soctcp1st        4 17:47:20          IBM-175C909B405.ibm.com|9088|soctcp
c917d50 soctcpio        3 17:47:20
c9008a0 soctcpoll        2 17:47:20
```

The port number is in the row that is related to the thread name, soctcp1st.

If you are not planning to use the location map, you can leave the Latitude and Longitude fields empty. If you are using the location map, those values can be determined, for example, by using Google Maps. For more information, refer to “Googlemapkey” on page 65.

When the form is complete, click **Save**.

Opening the OAT main page

Click **OAT Login** or type the following URL in the address bar of the browser:

`http://localhost/OAT/index.php`

Then perform the following steps:

1. Select the group.
2. Type the user name and password.
3. Click the **Get Servers** button.
4. Select the instance name.

The fields for Server Details are completed automatically, as shown in Figure 2-23.

5. Click **Login**.

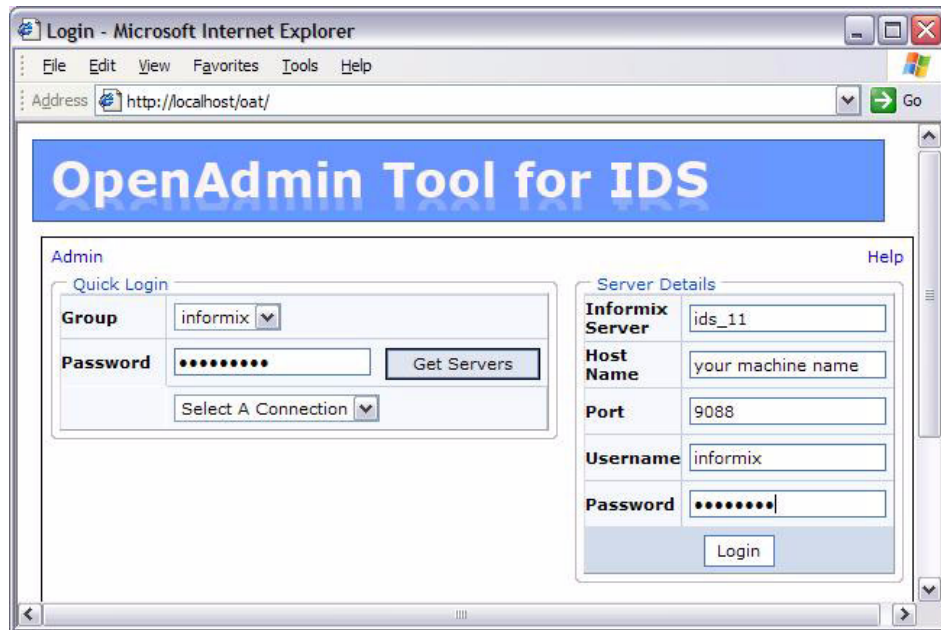


Figure 2-23 OAT Login

If the login is successful, you are redirected to the OAT main page, which is shown in Figure 2-24.

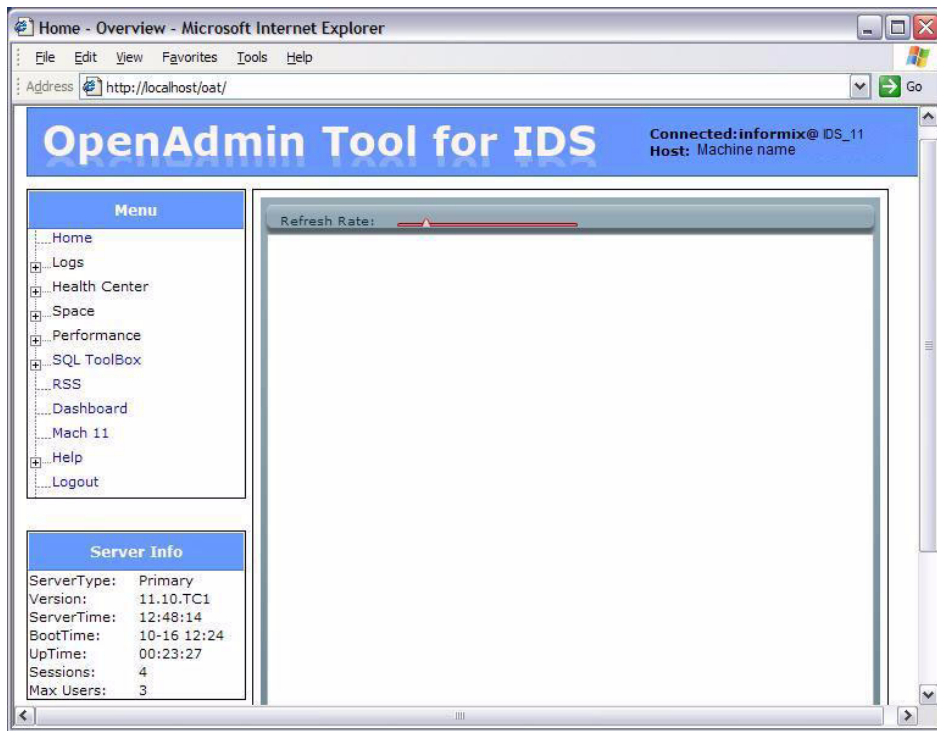


Figure 2-24 OAT main page

At this point you have completed the configuration and can now monitor and manage the instance by using the OAT. For more details about managing the instance, refer to Chapter 5, “The administration free zone” on page 161.



Enterprise data availability

In the world of information on demand, more and more companies find a need for reliable, uninterrupted, and continuous access to corporate information that is dispersed across all corners of the enterprise. Business continuity became a matter of survival since downtime costs for organizations in segments, such as financial market, credit card sales, home shopping, or airline reservations, could amount to thousands or even millions of dollars in loss of revenue.

Informix Dynamic Server (IDS) provides what we refer to in this book as *enterprise data availability* (EDA). EDA is provided by a suite of technologies that enable uninterrupted access to corporate information, and provide data replication, workload balancing, high availability, failover, and disaster recovery for important enterprise systems.

EDA is comprised of the high availability (HA) and data replication solutions that are embedded in IDS. Such solutions include High Availability Data Replication (HDR), Remote Standalone Secondary (RSS), Shared Disk Secondary (SDS), Continuous Log Restore (CLR), and Enterprise Replication (ER). These solutions are key to enabling an effective, flexible, and efficient way to maximize availability of data, provide disaster recovery, and ensure consistent delivery of that data wherever and whenever it is needed.

In this chapter, we provide an overview of the EDA solutions and possible combinations of the technologies on which they are based. We discuss the technologies and solutions that are available to provide the best implementation

to satisfy your business requirements. We also provide sample scenarios for better understanding.

In this chapter, we do not give details about how to configure and implement the solutions presented here because such information is well beyond the scope of the chapter. However, upon completion of this chapter, you will have a better understanding of the technology and solutions that are available to you with IDS.

More information: Refer to the Redbooks publication *Informix Dynamic Server 11: Extending Availability and Replication*, SG24-7488, which contains additional detail about existing availability and replication features of IDS. A softcopy of that book is available for download from the following Web page:

<http://www.redbooks.ibm.com/abstracts/sg247488.html?open>

3.1 Enterprise data availability solutions in IDS

IDS provides many innovative features to support high availability and replication of data.

High Availability Data Replication is extremely robust, having been part of IDS for over ten years. However, with HDR, there can only be one secondary instance. At this time, the user can only write to the primary instance, which might not enable the desired degree of load balancing.

Enterprise Replication is a powerful offering that enables solutions with enhanced flexibility. For example, a database administrator (DBA) can replicate as many or as few tables as desired. Multiple servers can be created, all of which stay synchronized with each other. As another long-time feature with IDS, ER delivers more enhanced features and improved functionality with each release.

The latest requirement is to have both the ease of use of HDR and the extensibility and one-to-many relationships of ER. With IDS 11, this functionality has been delivered with two new replication technologies, Remote Standalone Secondary and Shared Disk Secondary servers. Additionally, a new Continuous Log Restore feature makes it possible to manually maintain a backup system.

In this section, we provide a brief overview of the high availability and data replication technologies that are embedded in IDS. With this information, you will have a better understanding of how to apply and enable these EDA features to address your specific business and application needs.

3.1.1 High Availability Data Replication

HDR is a data replication and high availability solution that is fully integrated within the data server. HDR is easy to set up and administer. It works between two IDS server instances and requires a homogeneous environment where both of the computers in the HDR pair must be on same hardware architecture, operating system (OS), and IDS version, as illustrated in Figure 3-1.

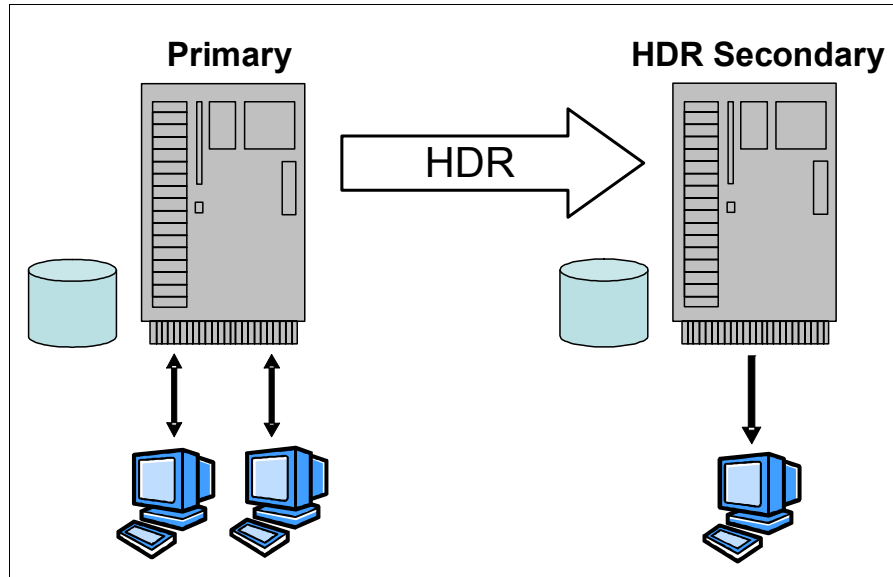


Figure 3-1 High Availability Data Replication

HDR employs a log record shipping technique to transfer the logical log records from the primary server to the secondary server. The secondary server is in perpetual roll-forward mode, so that data on the secondary server remains current with data on the primary server.

HDR can be configured to operate in synchronous (SYNC) or asynchronous (ASYNC) mode. In *SYNC mode*, we can guarantee that, when a transaction is committed on the primary server, its logs have been successfully transmitted to the HDR secondary server. In this case, the performance of the primary might be affected by the performance of the secondary server or network. Checkpoints in HDR are required to be synchronous, so that the primary and the secondary servers can switch roles. In *ASYNC mode*, transactions committed on the primary and transmission of logs to the secondary are independent. This can provide better performance, but it brings with it the risk of possibly losing transactions.

HDR uses a half-duplex communications protocol, meaning that the primary requires an acknowledgment (ACK) from the HDR secondary before sending the next buffer. This requirement can affect the performance of the primary server if, for any reason, the secondary does not send the ACK promptly.

HDR provides manual or automatic failover. If the primary server fails, the HDR secondary server automatically takes over and switches to a standard or primary server allowing minimal disruption to the clients. When the original primary server becomes available, it is synchronized when HDR is restarted.

The HDR secondary server can be used for read-only operations while in a functional HDR pair. As such, read-only applications, such as reports, can be executed against the secondary instance, thus reducing the load on the primary server. It can also be used as a hot backup server for additional availability in case of unplanned outages or disaster recovery scenarios.

3.1.2 Remote Standalone Secondary

Similar to HDR, RSS servers can provide geographically remote, application-accessible full copies of the primary instance. Logical logs are continuously transmitted from the primary server and applied to the database on the RSS server, as shown in Figure 3-2. RSS requires a homogeneous environment, as does HDR.

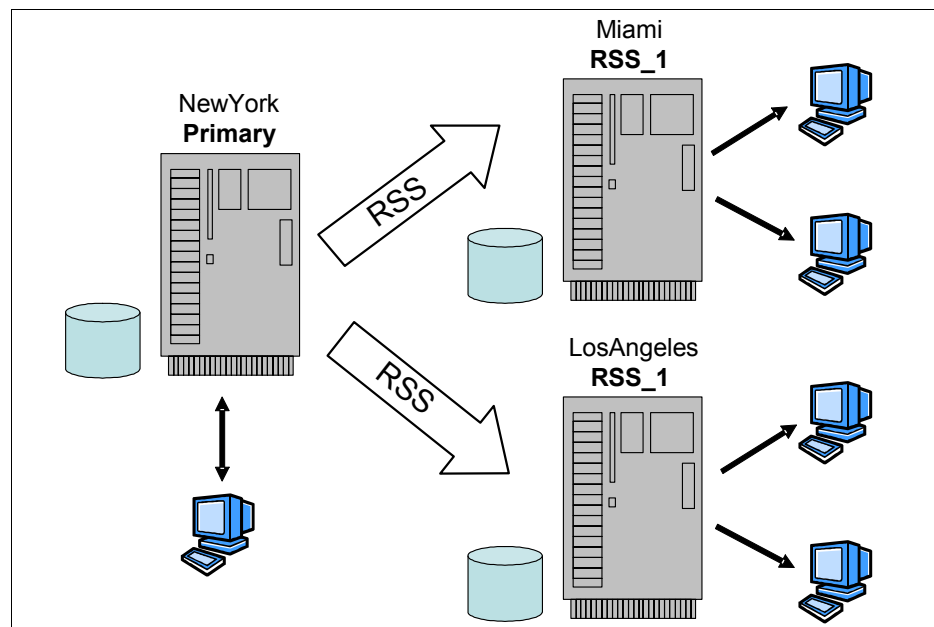


Figure 3-2 Remote Standalone Secondary

RSS is different from HDR. As examples, RSS only uses asynchronous transmissions of logs and checkpoints, RSS servers cannot be promoted directly to a primary, and one or more RSS servers can be created.

Instead of using the half-duplexed communications protocol of HDR, RSS servers use a fully duplexed protocol provided by the server multiplexer (SMX) communications interface that supports encrypted multiplexed network connections between servers in high availability environments. SMX provides a reliable, secure, high-performance communication mechanism between database server instances.

Using full duplexed communication results in RSS servers has little impact on the primary server performance.

Multiple RSS servers in geographically diverse locations can be used to provide faster query response than if all the users had to access the primary server. The application traffic that only reads the data can be sent to local RSS servers. For example, RSS servers can feed data to Web applications that do not require up-to-the-minute data. If the applications must update the data, they connect to the primary server. Otherwise, they read the data from the local RSS server. This configuration reduces network traffic and the time required by the application to access the data.

As shown in Figure 3-2 on page 78, remote application servers can access local database servers to minimize latency and improve performance.

RSS can also be used as multiple remote backup servers for additional availability in the event of unplanned outages or any catastrophe at the location of the primary or other HA secondary servers.

3.1.3 Shared Disk Secondary

Unlike HDR and RSS, SDS servers access the same physical disk as the primary server. They provide increased availability and scalability without the need to maintain multiple copies of the database, which results in lowering data storage costs, as illustrated in Figure 3-3.

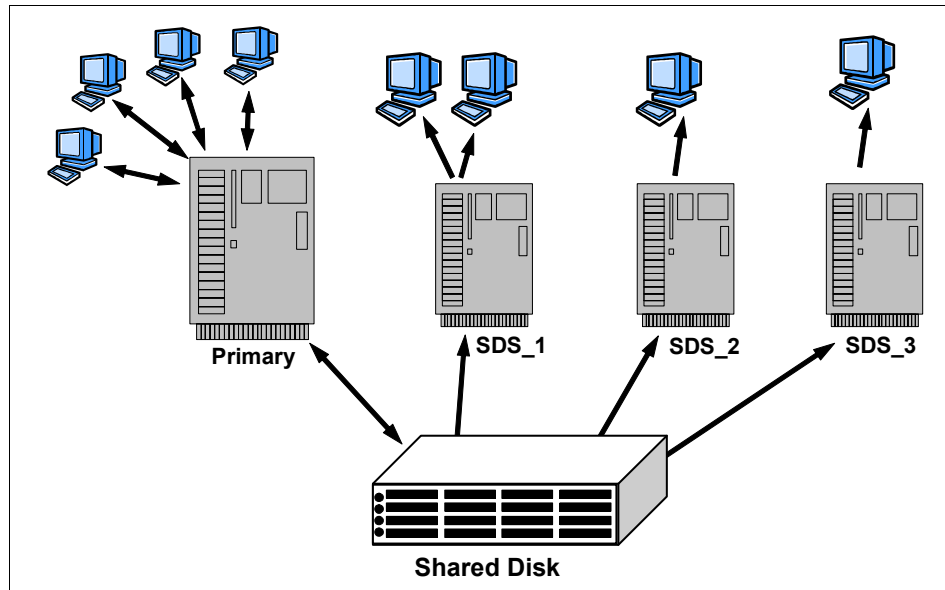


Figure 3-3 Shared Disk Secondary

SDS requires a homogeneous environment, as does HDR.

Like RSS servers, SDS servers also use the SMX layer, which is an internal component that is implemented to support the full duplexed communication protocol. SDS does not support synchronous mode, which is similar to RSS, but different from HDR.

The SDS architecture provides the ability to set up multiple database servers sharing the entire dbSPACE set that is defined by a primary database server. It can be used for defining database servers on the same physical machine or different machines with an underlying shared file system.

Multiple SDS servers provide the opportunity to dedicate specific SDS servers for specific tasks, such as data warehousing as a DSS-oriented server or Web application server with an online transaction processing (OLTP) approach, with the appropriate differences in the configuration for parallel database query (PDQ) and memory requirements. The SDS environment can also be used simply for

work balancing, by spreading the existing company applications across the SDS servers in the infrastructure to achieve a better throughput.

An SDS server can be made available quickly. When configured, an SDS server joins an existing system and is ready for immediate use.

The benefits of this feature in terms of resources, in comparison with HDR and RSS, are a significantly lower requirement on disk space and a slight reduction in network traffic. The simple requirements for setup and configuration do not bind additional DBA resources. In addition, much better load balancing and workload partitioning can be achieved by dynamically adding and removing SDS servers in an existing infrastructure.

Shared disk file systems: Several shared disk file systems are available in the market that guarantee concurrent use by different systems in a high availability cluster. For example, the IBM General Parallel File System™ (GPFS™) is a high performance shared disk file system that can provide fast, reliable data access from all servers for AIX and Linux cluster systems. Similarly, other shared disk technologies, such as Veritas Storage Foundation Cluster File System, Redundant Array of Independent Disks (RAID), and Storage Area Network (SAN), can also be used to set up an SDS cluster. However, we do not recommend the use of a mounted Network File System (NFS) for the Shared Disk Secondary servers, for performance reasons.

3.1.4 Continuous Log Restore

CLR is used as a robust way to set up a hot backup of a database server for increased availability in case of unplanned outages or disaster recovery scenarios. The hot backup of the primary IDS server is maintained on the backup server, which contains similar hardware, OS, and an identical version of IDS.

To configure a backup server by using CLR, a physical backup of the primary server is created, and the backup copy is transported to the backup server. The backup is then restored on the backup server. After the restore is complete, the backup server is ready for logical recovery. In the event that a logical log on the primary server becomes full, it is backed up and then transported to the backup server where logical recovery is performed. Figure 3-4 on page 82 illustrates the operation of CLR.

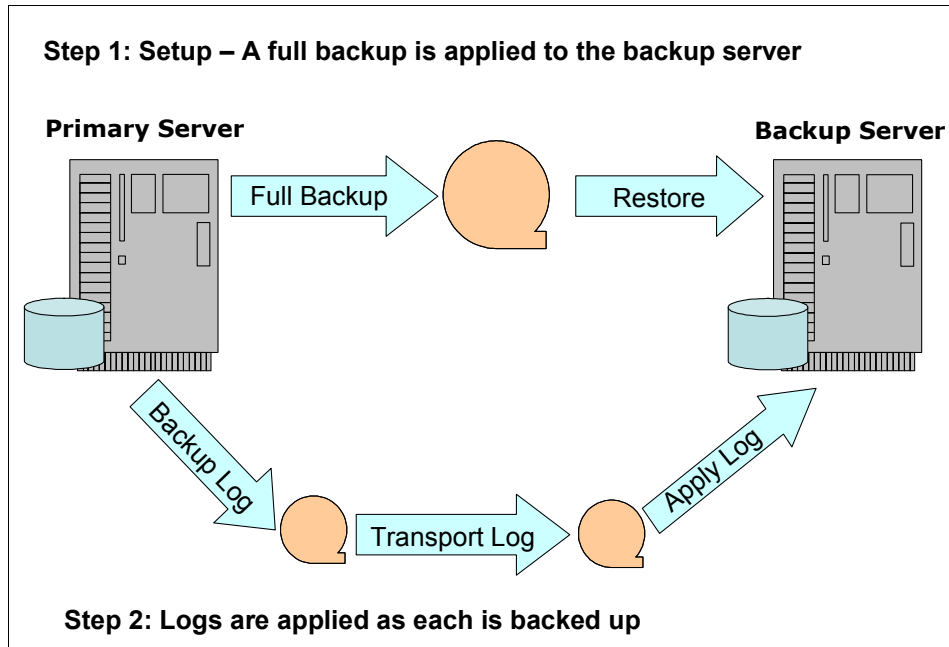


Figure 3-4 Continuous Log Restore

Should the primary server become unavailable, a final log recovery is performed on the backup server, which is brought up in online mode as the primary server.

CLR is useful when the backup database server is required to be fairly current, but the two systems need to be completely independent of each other for reasons such as security and network availability. CLR can also be useful when the cost of maintaining a persistent network connection is too high. With CLR, log files are manually transferred to a backup database server where they are restored.

3.1.5 Enterprise Replication

ER provides reliable propagation of configurable selected data across multiple IDS servers within complex network topologies, such as the one shown in Figure 3-5.

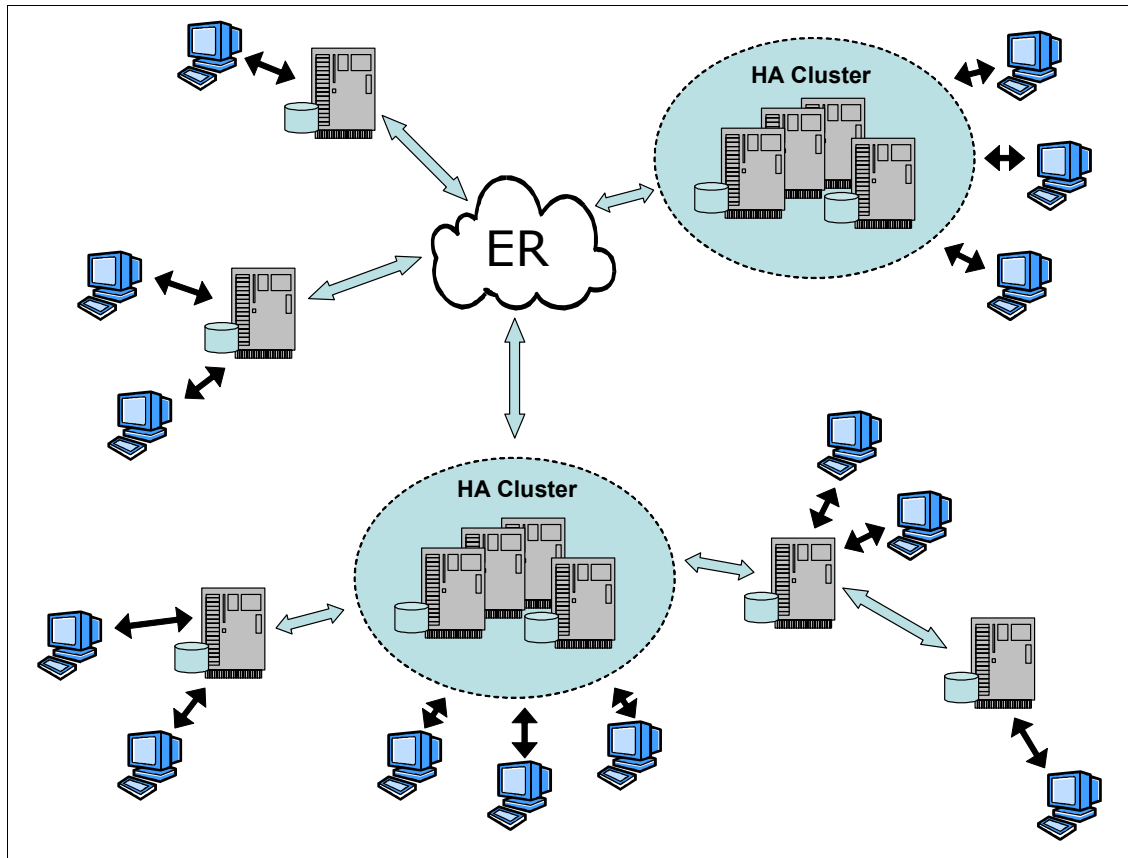


Figure 3-5 Enterprise Replication

ER is an asynchronous, log-based data replication solution. It works with both homogeneous or heterogeneous environments, meaning that each of the computers that run ER servers can be on same or different hardware architectures and OS and use different versions of IDS. For example, you can replicate the data from IDS 11 (32-bit) on Linux to IDS 10 on Solaris (64-bit).

ER can be configured to replicate data immediately, at certain intervals or point in time. It can also be used to replicate individual tables or subsets of tables rather

than the entire database or instance. In addition, each ER definition can target different specific instances, rather than all instances in the ER system.

The flexible architecture of ER allows organizations to customize their replication environment based on business requirements and models as in the following examples:

- ▶ Primary-target replication, where the flow of information is in one direction, usually for the purpose of data dissemination or consolidation
- ▶ Update-anywhere replication, where changes made on any location are replicated to all other participating database servers, often used for workload distribution

ER provides mechanisms to easily set up and deploy replication for systems with large numbers of tables and servers. It also provides support for Online Schema Evolution that allows modifications in replication definitions or replicated tables for an active ER system without interrupting the data replication.

ER offers an effective mechanism for replication within network topologies with fully-connected database servers and not-directly-connected database servers in a hierarchical tree of servers. Depending on the volume of data, the distance between the servers and the network facilities that are available, ER can be configured to use a hierarchical tree or forest of trees topology, in a way that the network traffic and database server processing could be highly reduced. For example, if replication for a large number of servers across continents is required, then a fully-connected topology might not be feasible for all the servers because of insufficient network bandwidth for the data volume. Therefore, in this case, an ER system can benefit from an hierarchical topology.

ER is not an instance-wide replication. Therefore, the disk space requirement for each IDS instance depends on that database server usage and other business needs.

All the features of ER can result in a wide spectrum of benefits, including reliable and fast replication of data across a distributed or global organization, improved data availability, capacity relief, and increased performance.

3.2 Clustering EDA solutions

Depending upon the business needs for high availability, failover, disaster recovery, data distribution and sharing, workload balancing, and capacity relief, you can choose to configure your database servers by using various combinations of HDR, RSS, SDS, CLR, and ER technologies. All the EDA

solutions in IDS can coexist and work together to satisfy an extremely wide range of business and application requirements for important systems.

The EDA features are built within the server and can interact with each other making IDS a powerful database server yet that is simple to configure and administer, thus minimizing DBA activity and overall cost of ownership.

3.2.1 HA clusters

Figure 3-6 shows an *HA cluster*, which is a combination of all the possible HA solutions including CLR, HDR, RSS, and SDS nodes. Depending on the business needs, the HA cluster can include more RSS and SDS nodes, or it can be a subset of the configuration shown.

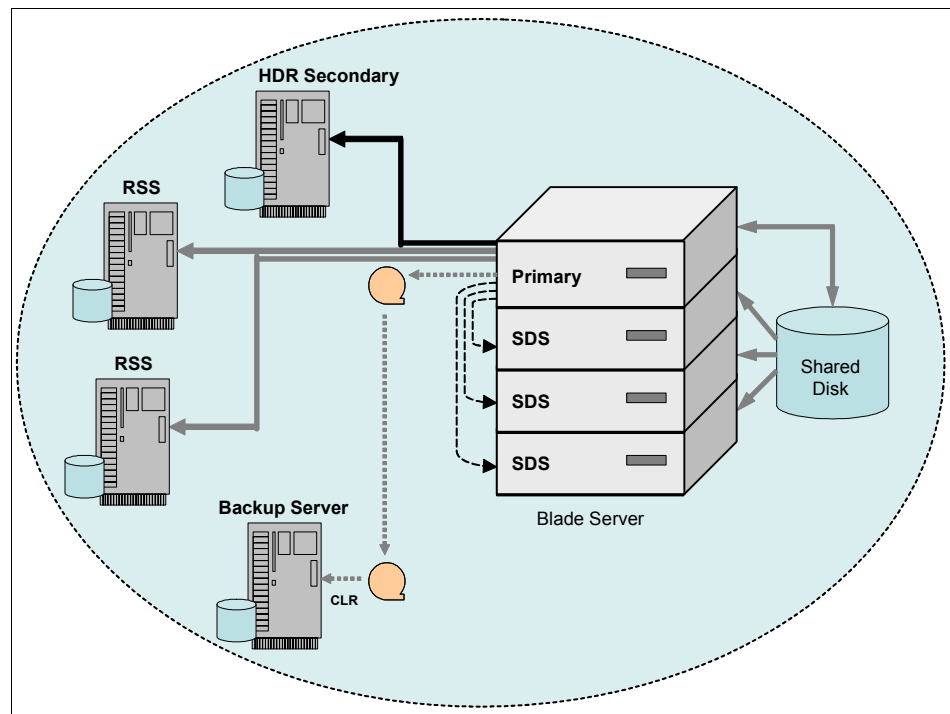


Figure 3-6 HA cluster with CLR, HDR, RSS, and SDS servers

The HA cluster can provide such capabilities as high availability, failover, disaster recovery, and workload balancing. It can also support planned or unplanned outages.

Typically, planned outages are required in situations where one of the servers in the HA cluster is scheduled for maintenance such as in the hardware or OS.

Unplanned outages occur when events, such as loss of power or a disaster situation, occurs.

In these situations, a DBA can choose the available failover options. These include failover of the primary to a secondary, switching roles between the primary and secondary, switching the secondary to another type, changing a server to standard mode, or just removing a server from the HA cluster.

3.2.2 ER with HA clusters

IDS supports mixing the HA technologies with ER, so that ER can seamlessly work with HDR, RSS, SDS or CLR.

HA clusters can participate in an ER configuration. For example, any of the database servers in an ER network can be part of an HA cluster. This might be used to ensure that key nodes, such as the ER root nodes in a hierarchical topology, are highly available, so the flow of data is not interrupted.

In this case, only the primary server of an HA cluster is included in the replication network. The backup or secondary servers are not directly configured for ER.

The order of creating the systems does not matter. An HA cluster can be created and then added to an ER network, or any stand-alone system in an ER network can be converted to an HA cluster.

What matters is to ensure that paths are available, so that the failure of any single system does not leave sets of systems cut off from one another. For example, in Figure 3-7 on page 87, if one of the central servers, in New York or London, is not available, then none of the regional servers will be able to connect to the rest of the enterprise. In this case, each central server and its regional servers are good candidates to be HA clusters.

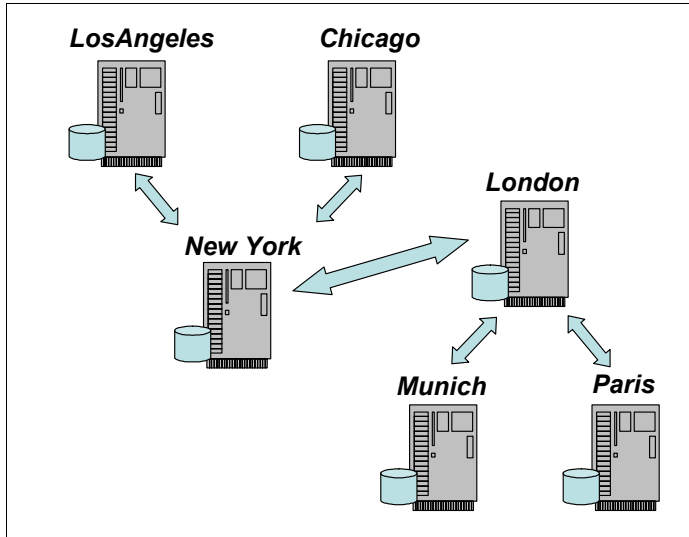


Figure 3-7 ER root nodes without HA clusters

The fault tolerance capability of the configuration can be improved by adding HA clusters to both root nodes (New York and London), as shown in Figure 3-8.

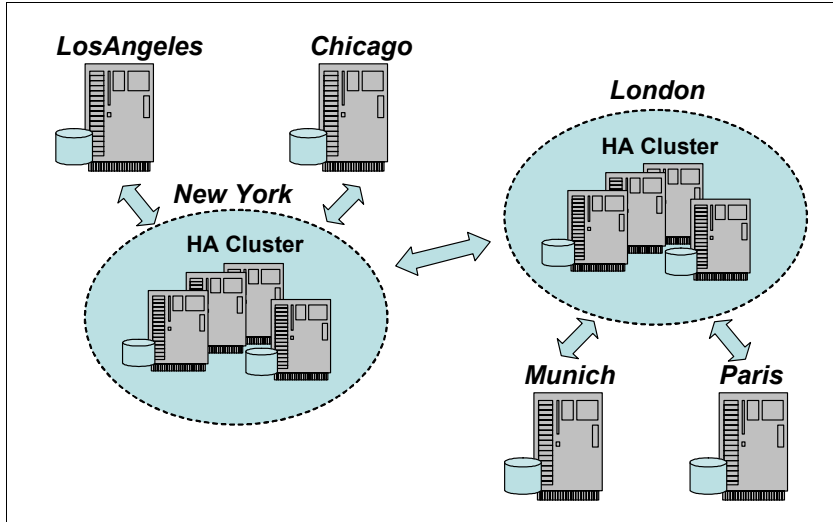


Figure 3-8 ER root nodes with HA clusters

Figure 3-9 illustrates a combination of all the HA and ER technologies in IDS. It shows three HA clusters in an ER network. It also shows how the headquarters (HQ) and each region (Region1 and Region2) of a company can be self sufficient, with respect to their HA needs, and still share the data over ER.

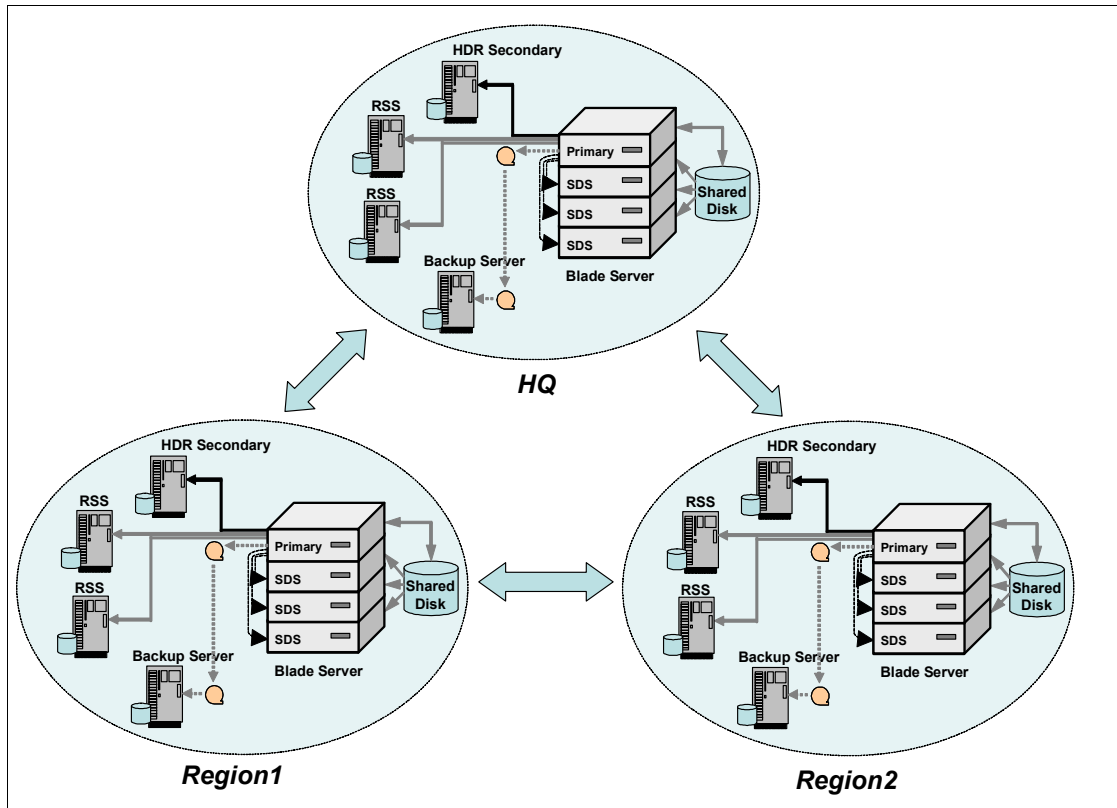


Figure 3-9 Combination of HA clusters and an ER network

In this scenario, the down time of an ER network can be significantly reduced by the use of HA technologies.

3.3 Selecting the proper technology for your business

In this section, we recapitulate the characteristics and advantages for each of the EDA solutions. We then provide suggestions and directions regarding when and where these technologies, or combinations of these technologies, can be best used.

3.3.1 EDA technologies

In the current demanding environment, more and more companies strive to achieve and maintain high availability from their IT systems because any interruption in accessing their systems can cause loss of revenue. Another important requirement pursued is to obtain high performance from their distributed systems that, depending on the company and business environment, might be spread worldwide across many continents. Of course, these goals need to be achieved with minimum cost. Therefore the high availability, high performance, and minimum total cost of ownership (TCO) become a challenging trio of demands.

The high availability requirement can be obtained with the failover, disaster recovery, and data distribution capabilities provided by HDR, RSS, SDS, CLR, and ER technologies included in IDS. In the same way, the high performance goal can be achieved by using IDS and all of its built-in features that make it a fast database server, along with the capacity relief, workload balancing, and data sharing capabilities provided by the HA and ER solutions of IDS.

High availability and high performance requirements not only depend on the database server, but also on the other components that comprise the system such as hardware, OS, network, and the application. Nevertheless, the Informix EDA solutions discussed in this chapter can be used as an important part of the overall solution.

There is no single solution to satisfy the requirements of everyone. This is because the TCO and the level of availability and performance requirements differ from one organization to another. Therefore, it is best to understand the capabilities for each of the EDA technologies of IDS to help determining the best strategy for your particular environment.

3.3.2 Summary of capabilities and failover options

Table 3-1 summarizes the capabilities and failover options for the EDA technologies of IDS. You can use this table to assist you in selecting the proper solution for your business needs.

Table 3-1 Comparison of capabilities and failover options for EDA technologies

Capabilities and failover options	HDR	RSS	SDS	CLR	ER
Provides high availability	Yes	Yes	Yes	Yes	Yes
Provides failover	Yes	Yes	Yes	Yes	No
Provides disaster recovery	Yes	Yes	Yes	Yes	No
Provides server redundancy	Yes	Yes	Yes	Yes	Yes
Provides data redundancy	Yes	Yes	No	Yes	Yes
Provides workload balancing or capacity relief	Yes	Yes	Yes	No	Yes
Provides workload partitioning	No	No	No	No	Yes
Provides dissemination and consolidation	No	No	No	No	Yes
Primary/source to secondary/backup/target distance (near/far)	Near	Both	Near	Both	Both
Supports Online Schema Evolution for application upgrades	No	No	No	No	Yes
Setup and configuration difficulty	Easy	Easy	Easy	Easy	Custom
Deployment difficulty	Easy	Easy	Easy	Easy	Easy
Disk layout of primary or source compared to secondary, backup, or target	Identical	Identical	Shared	Identical	Custom
Disk space dependency	High	High	Low	High	Custom
Shared disk system dependency	No	No	Yes	No	No
Fully duplexed (SMX) versus half duplexed communication	Half	Fully	Fully	N/A	N/A
Network dependency	High	Medium	Low	None	Custom
Supports complex network topologies with fully-connected and non-directly-connected IDS servers	No	No	No	N/A	Yes

Capabilities and failover options	HDR	RSS	SDS	CLR	ER
Homogeneous versus heterogeneous environment (hardware architecture, OS, IDS version)	Homo	Homo	Homo	Homo	Hetero
Asynchronous versus synchronous replication	Both	Async	Async	N/A	Async
Instance-wide versus customized replication or copy	Instance	Instance	Instance	Instance	Custom
Primary or source and secondary, backup, or target are like mirror images	Yes	Yes	Yes	Yes	No
Number of primary or source servers	One	One	One	One	Many
Number of secondary, backup, or target servers	One	Many	Many	Many	Many
Read-only versus read-write operations on secondary, backup, or target	Read - only	Read - only	Read - only	None	Read - Write
Can be part of an HA cluster	Yes	Yes	Yes	Yes	N/A
Only one shared primary server for any existing secondaries in an HA cluster	Yes	Yes	Yes	Yes	N/A
Only primary of HA cluster participates in ER	N/A	N/A	N/A	N/A	Yes
Coexists and works together with other EDA solutions	Yes	Yes	Yes	Yes	Yes
Secondary or backup can become a primary server	Yes	No	Yes	No	N/A
secondary or backup can become a standard server	Yes	Yes	No	Yes	N/A
Secondary can become an HDR secondary	N/A	Yes	No	N/A	N/A
Secondary can become an RSS	Yes	N/A	No	N/A	N/A
Secondary can become an SDS	No	No	N/A	N/A	N/A

3.3.3 Recommended solutions

EDA technologies can be combined to suit a variety of business situations. Table 3-2 illustrates examples of combinations that can be used to address different application and business requirements.

Table 3-2 Recommended solutions for various requirements

Application or business requirement	Recommended solution
Periodic requirement to increase reporting capacity.	Use SDS or RSS servers. If the amount of data is large and maintaining multiple copies is difficult, then use SDS servers.
You are using SAN devices, which provide ample disk hardware availability, but you are concerned about server failures.	Use SDS servers.
You are using SAN devices, which provide ample disk hardware mirroring, but you also want a second set of servers that can be brought online if the primary operation should fail (and the limitations of mirrored disks are not a problem).	Consider using two blade centers running SDS servers at the two sites.
You want to have a backup site a moderate distance away, but cannot tolerate any loss of data during failover.	Consider using two blade centers with SDS servers on the primary blade center and an HDR secondary on the remote server.
You want a highly available system in which no transaction is ever lost, but that must also have a remote system on the other side of the world.	Consider using an HDR secondary located nearby running in SYNC mode and an RSS server on the other side of the world.
You want a high availability solution, but because of the networks in your region, there is a large latency.	Consider using an RSS server.
You want a backup site, but you do not have any direct communication with the backup site.	Consider using CLR with backup and recovery.
You can tolerate a delay in the delivery of data as long as the data arrives eventually. However, you need quick failover in any case.	Consider using SDS servers with hardware disk mirroring in conjunction with ER.
You need additional write processing power, can tolerate some delay in the delivery of those writes, need something highly available, and can partition the workload.	Consider using ER with SDS servers.

3.4 Sample scenarios

The EDA technologies described in this chapter can practically be present in any business environment that requires capabilities such as high availability, failover, disaster recovery, workload balancing, data sharing, and distribution. Some of those environments that might fit closely to your business include OLTP, Web applications, embedded applications, decision support, data warehousing, and information on demand systems.

In this section, we discuss sample scenarios for the mentioned environments. One of them might just be the solution that you need to satisfy your business requirements.

3.4.1 Failover and disaster recovery

Today, failover capability alone is not the only requirement for data centers. We have all witnessed natural and man-made calamities, and it is necessary for everyone to be ready for disaster recovery. Traditional backup and restore provides disaster recovery capability, but this traditional mechanism can be slow and require a significant amount of preparation and execution.

In this example, we show a complex failover and disaster recovery strategy by using the HA solutions that are available in IDS. This layered availability strategy should provide maximum availability to survive a regional disaster.

The first layer provides availability solutions to deal with transitory local failures. For example, this might include having a couple of blade servers attached to a single disk subsystem running SDS servers. Placing the SDS servers in several locations throughout your campus makes it possible to provide seamless failover in the event of a local outage.

You might want to add a second layer to increase availability by including an alternate location with its own copy of the disks. To protect against a large regional disaster, you might also consider configuring an HDR secondary server located some distance away, perhaps hundreds of miles. You might also want to make the remote system a blade server or some other multiple-server system. By providing this second layer, if a failover should occur and the remote HDR secondary became the primary, then it is possible to easily start SDS servers at the remote site.

However, even a two-tiered approach might not be enough. A hurricane in one region can spawn tornadoes hundreds of miles away. To protect against this, consider adding a third tier of protection, such as an RSS server located one or more thousand miles away. This three-tier approach, depicted in Figure 3-10, provides for additional redundancy that can significantly reduce the risk of an outage.

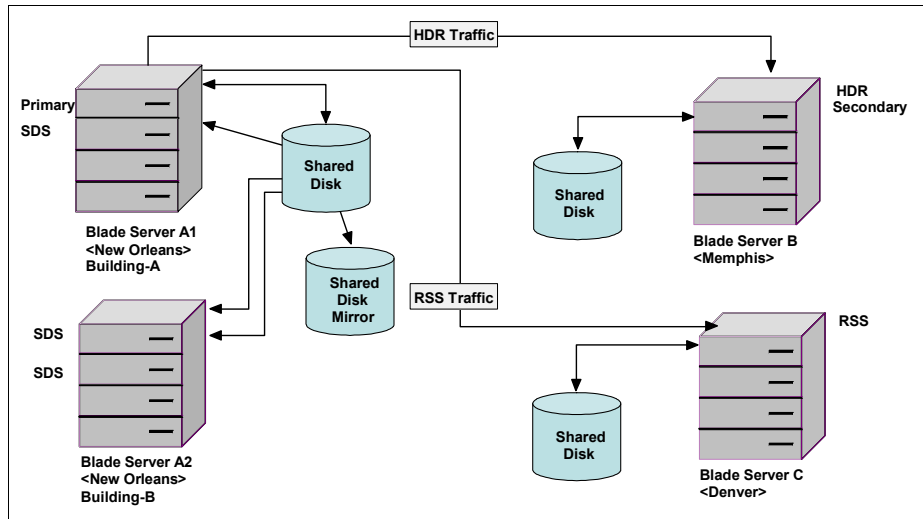


Figure 3-10 Configuration for three-tiered server availability

Now suppose that a local outage occurred in Building-A on the New Orleans campus. Perhaps a pipe burst in the machine room causing water damage to the blade server and the primary copy of the shared disk subsystem. You can switch the role of the primary server to Building-B by running the `onmode -d` command to make the primary server name on one of the SDS servers running on the blade server in Building-B. This causes all other secondary nodes to automatically connect to the new primary node, as shown in Figure 3-11 on page 95.

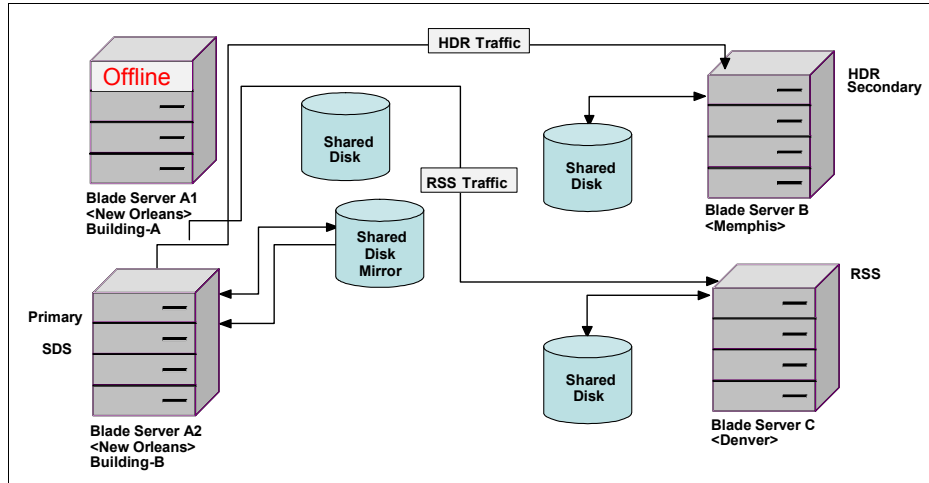


Figure 3-11 First tier of protection

Should there be a regional outage in New Orleans such that both building A and building B are both lost, then you can shift the primary server role to Memphis. In addition, you might also want to make Denver an HDR secondary and possibly add more SDS servers to the machine in Memphis. Figure 3-12 illustrates this scenario.

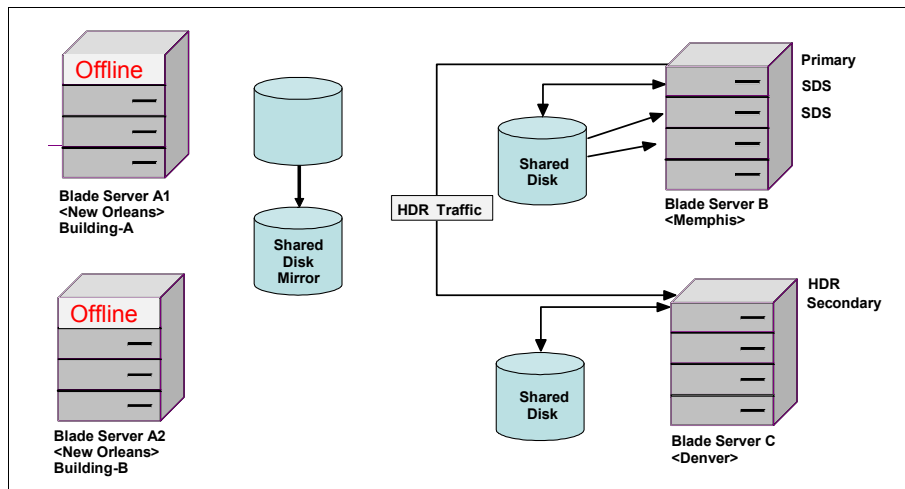


Figure 3-12 Second tier of protection

An even larger outage that affects both sites requires switching to the most remote system, as illustrated in Figure 3-13.

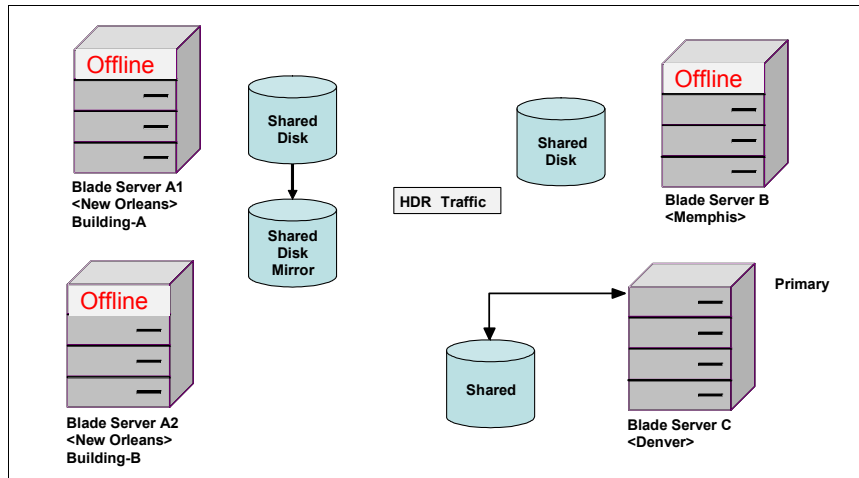


Figure 3-13 Third tier of protection

3.4.2 Workload balancing with SDS and ER

In this section, we describe the scenario an organization that needs OLTP servers for their production and needs to analyze that production data by using data warehousing solutions. The organization has a large amount of disk capacity in the form of shared disks and SAN, but has limited processor capacity.

During normal operations, there is enough capacity to satisfy the needs of the OLTP clients that mostly perform read-only access. Figure 3-14 shows the initial setup using SDS nodes at each site, along with ER.

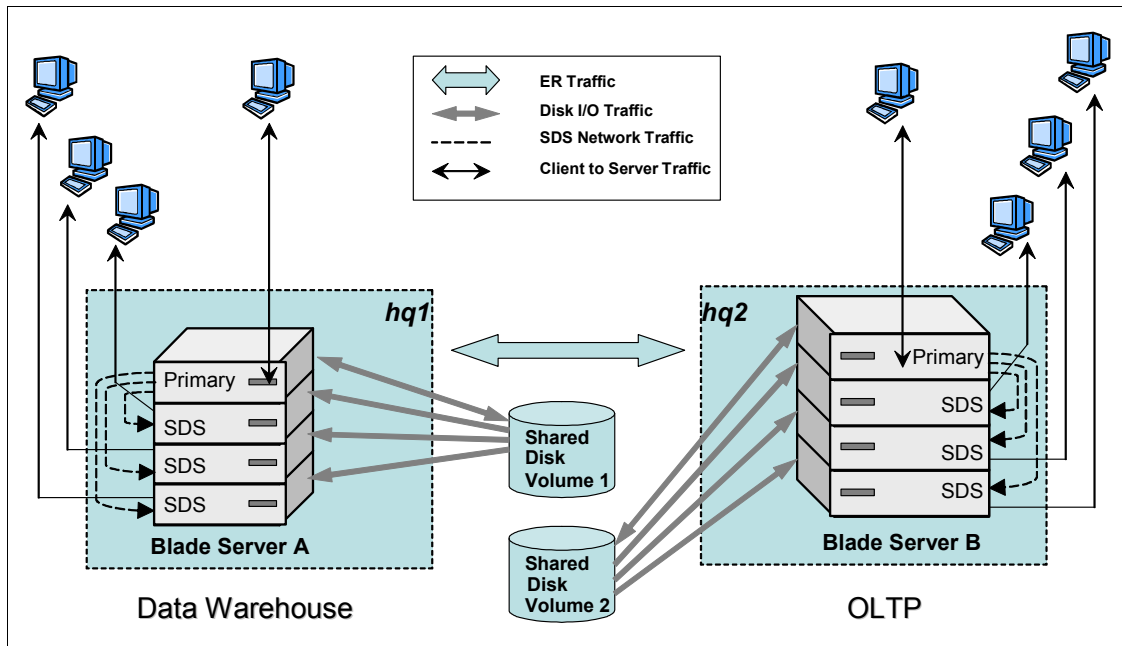


Figure 3-14 Organization with data warehouse and OLTP using SDS and ER

Over time, this company might face a need for increasing the capacity at the OLTP site to satisfy additional client applications doing read-only operations. To satisfy this need, the company can simply switch some of the SDS nodes at the data warehouse site to work connected to the primary at the OLTP site and then redirect some of the clients to those SDS nodes as illustrated in Figure 3-15.

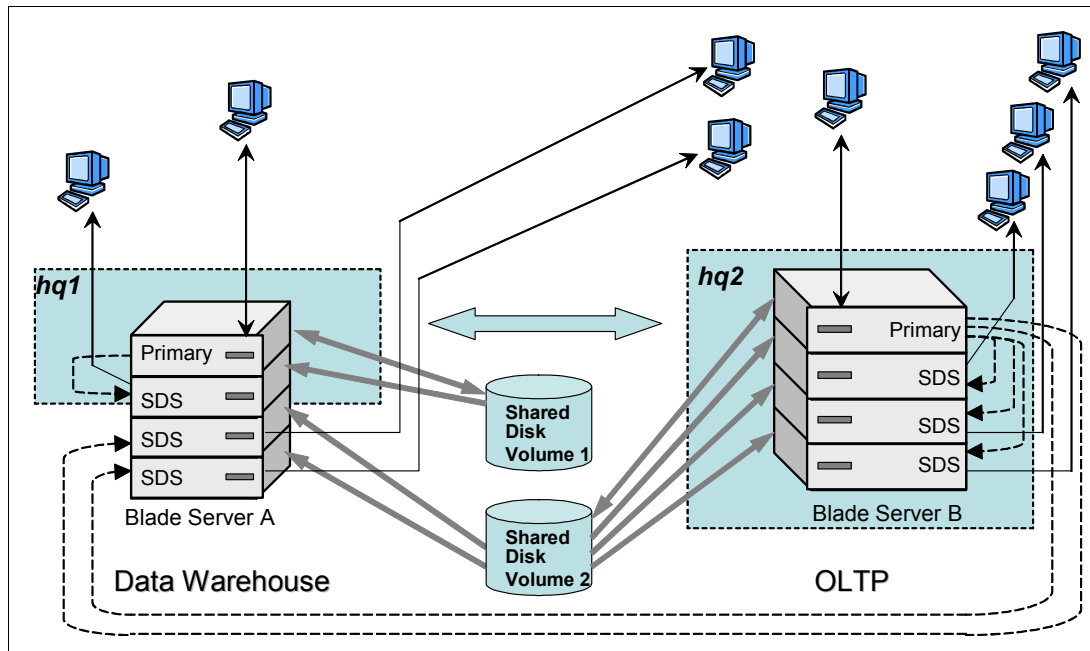


Figure 3-15 Handling transient need for extra OLTP client requirements

3.4.3 Data availability and distribution using ER

Database replication is important because it enables enterprises to provide users with access to current data where and when they need it. It can also provide a wide spectrum of benefits, including improved performance when centralized resources get overloaded, increased data availability, capacity relief, and support for data warehousing to facilitate decision support.

An enterprise typically has a number of requirements for using data replication. One key to designing an ER implementation is a clear understanding of those requirements. Some organizations might want to increase fault tolerance by duplicating the critical information in more than one location. Or, some might want to increase data availability and performance of their application by using local replicas to avoid wide area network (WAN) network traffic.

In the following sections, we provide examples of replication business models that organizations might want to implement in their environment.

Call center

Organizations might want a consistent set of records that can be updated at any site in a peer-to-peer fashion. That is an update-anywhere replication. This capability allows users to function autonomously and continue to function even when other servers or networks in the replication system are not available.

In this scenario, an international corporation needs 24x7 availability to support centers in various locations around the world. For example, a call is routed to one of the three call centers, depending on the time of day and call load-balancing requirements. Each call center then needs the ability to update records and have them replicated to the other call centers. When a call is made, it might be routed to Sydney, Australia, and the information taken is then replicated to other call centers in such locations as London and Miami. Therefore, each call center would have the information captured at the other call centers.

Retail pricing

A company might want to use a data dissemination model where data is updated in a central location and then replicated to multiple, read-only sites. This method is useful when information must be distributed, for example, to multiple sales locations.

For example, a hotel chain might want to send reservation information to the various hotels, or a book store chain headquarters might need to send updated price lists of available books to its stores on a nightly basis. To ensure that this data is consistent, the stores have read-only access to the information while the headquarters has read-write capability.

Data warehousing and DSS

Another business model that companies might want to use is data consolidation. With data consolidation, data is updated at multiple sites and replicated to a central, read-only site for DSS, accounting, or centralized reporting systems. This method gives data ownership and location autonomy at the branch level.

An example of such environment is a retail store chain that throughout the day gathers point-of-sale information. At the end of the business day, the stores must transmit the data to the headquarters, where it is consolidated into the central data warehouse to be used in various business intelligence processes, such as trend analysis and inventory control systems.

Human resources system

Organizations might also want to use the workload partitioning model where users at different sites can update data only in their own table partition, but can view data in the entire table. This gives database administrators (DBAs) the flexibility of assigning data ownership at the table-partition level.

For example, an HR system where the European site has ownership of its partition and can modify employee records for personnel in its region. Any changes to the data are then replicated to the U.S. sites. While the European site can query or read the other partitions, it cannot update them. Similarly, the U.S. sites can change data only within their own respective partitions, but can query and read data in all partitions. Any changes to the U.S. data are replicated to the European site.

3.4.4 Rolling upgrades for ER applications

ER supports schema evolution for application upgrades while ER is active and working. If an ER configuration has multiple clients connected to each of the servers and if there is a schema upgrade of any of the replicated tables or database, the new version of the clients can be deployed in a phased manner.

As an example, Figure 3-16 shows how two ER nodes, New York and London, replicate a database table, Table X, which has four columns. A set of applications, App V1, are connected to each of the ER nodes and execute transactions against Table X.

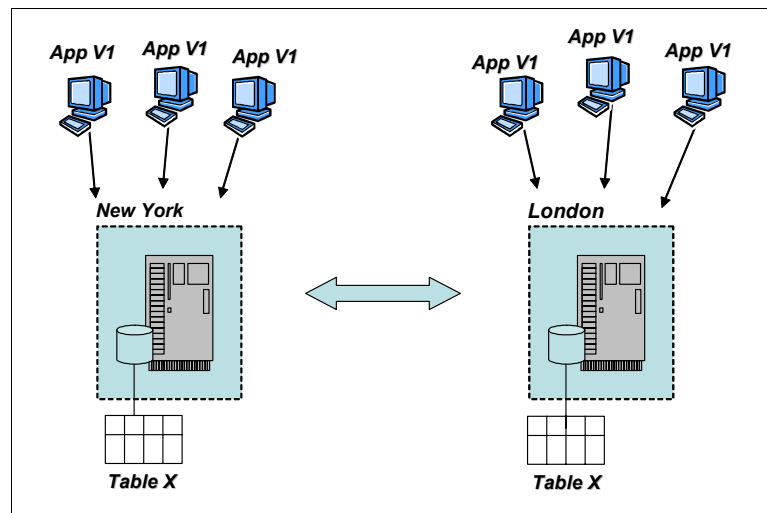


Figure 3-16 Replicated table with applications executing

Now we can add two columns to Table X on the both the servers. A new version of the application (App V2) is also developed to access the new schema of Table X. We can do this by first migrating all the applications App V1 running against the London ER node to connect to the New York node. Then we change the Table X schema, using the SQL command ALTER TABLE, to add two columns. ER is still active and Table X still gets any changes that have been applied at the ER node New York as illustrated in Figure 3-17.

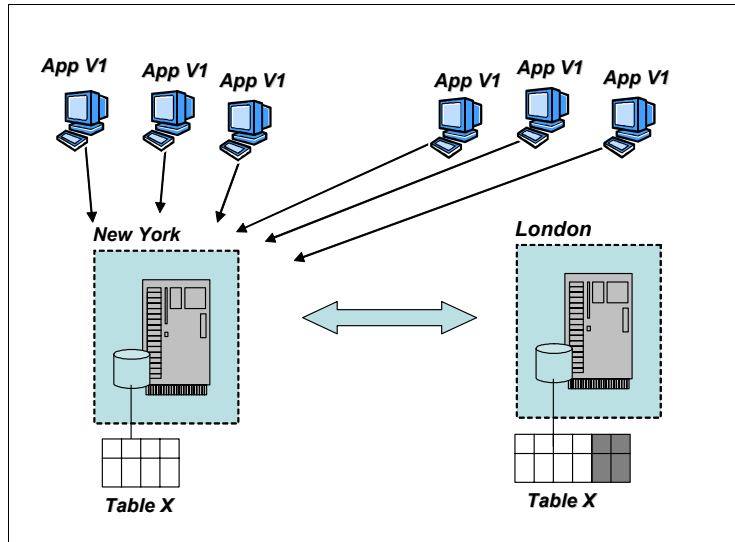


Figure 3-17 Altering the table at one ER node with older client versions migrated

Now that the ER node London has a new table schema, we can retire application App V1 and deploy the new application App V2 at both the sites. But all the new applications are now connecting to the ER node London as shown in Figure 3-18. Table X can also be changed to add the two extra columns at ER node New York, but we must re-master the replicate to include the new columns for both of the replicate participants.

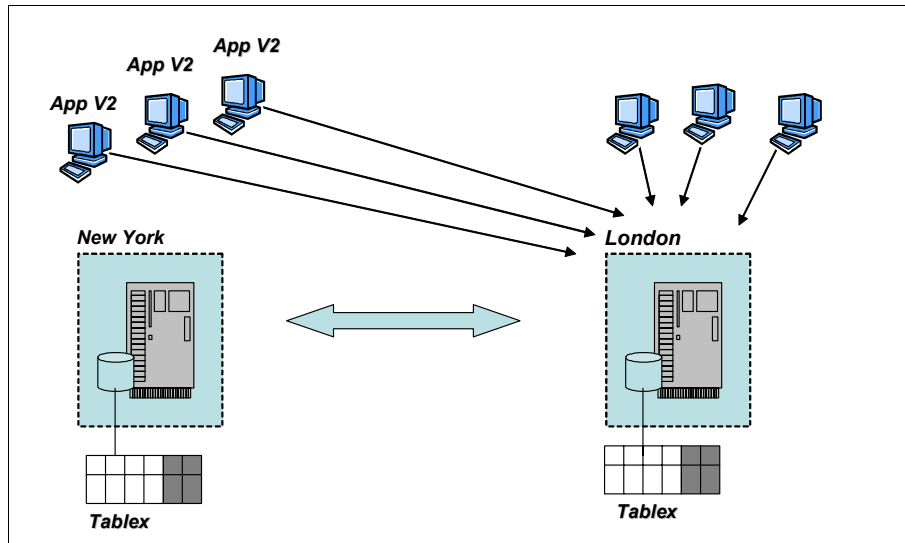


Figure 3-18 Deployment of new version of application that is aware of the new schema

Finally, we can load balance the application instances at both the ER nodes as shown in Figure 3-19.

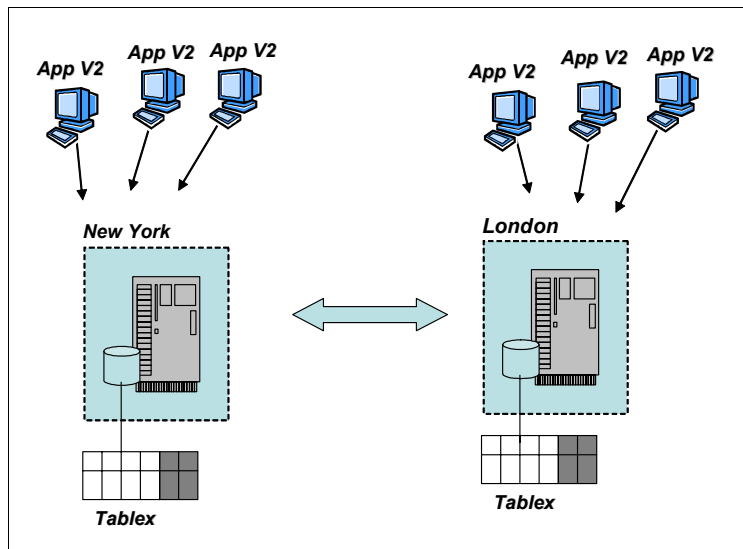


Figure 3-19 Redistributing the applications to the ER nodes

Thus without any server downtime, we can smoothly upgrade the applications from one version to another version.

3.4.5 Application redirection using server groups

The HA technologies of IDS support an automatic client redirection feature, which makes failover transparent to the application. One of the methods used by applications to automatically redirect clients to a primary server in an HA cluster is by using server groups.

Alternative: There are other application redirection mechanisms, such as using `INFORMIXSERVER` and `DBPATH` environment variables, but those methods are not described in this example.

By using a database server group, you can treat multiple related database servers as one logical entity to establish client/server connections, or to simplify the redirection of connections to database servers. Figure 3-20 illustrates the concept of a server group.

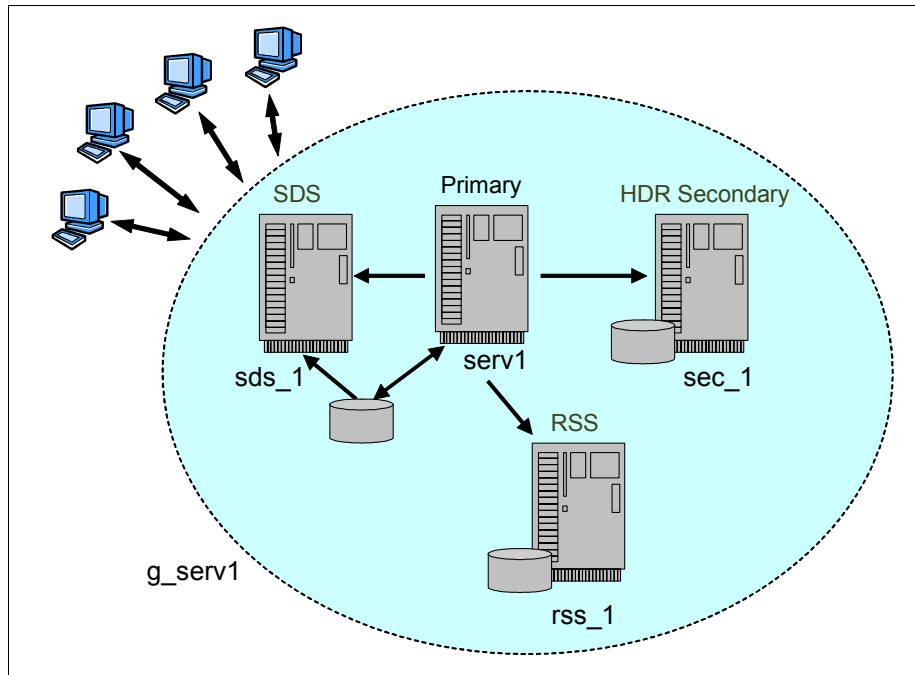


Figure 3-20 Server group

In Figure 3-20, the primary, HDR secondary, RSS, and SDS instances are configured as one logical instance named `g_serv1`. This configuration is accomplished through entries in the client and server `SQLHOSTS` files as shown in Table 3-3.

Table 3-3 `SQLHOSTS` file with server group syntax

DBSERVERNAME	NETTYPE	HOSTNAME	SERVICE NAME	Options
<code>g_serv1</code>	<code>group</code>	<code>-</code>	<code>-</code>	<code>i=100</code>
<code>serv1</code>	<code>onsoctcp</code>	<code>host1</code>	<code>serv1_tcp</code>	<code>g=g_serv1</code>
<code>sec_1</code>	<code>onsoctcp</code>	<code>host2</code>	<code>serv1_sec_tcp</code>	<code>g=g_serv1</code>
<code>sds_1</code>	<code>onsoctco</code>	<code>host3</code>	<code>sds_1_tcp</code>	<code>g=g_serv1</code>
<code>rss_1</code>	<code>onsoctcp</code>	<code>host4</code>	<code>rss_1_tcp</code>	<code>g=g_serv1</code>

It is important to mention that from an application perspective, there is no difference between a primary and a secondary database servers other than that the secondary server is read only. Depending on the business needs, applications can use both primary and secondary database servers to maximize application performance. In practical terms, applications can be written so that updates and modifications occur at the primary instance (*serv1*), while report functions or read-only applications are directed to the secondary instances (*sec_1*, *sds_1* and *rss_1*).

However, from a failover perspective, if `INFORMIXSERVER` is defined as *g_serv1*, the Informix connectivity libraries will always redirect an application connection to the primary instance defined as part of the server group. The application does not need to be recompiled or restarted. Transactions that were in-flight must be restarted since the newly promoted primary does not have any of the shared memory buffer information. In this case, the application connection is never redirected to a secondary server, but only to the current primary server.

3.5 Monitoring cluster activity

There are several ways to monitor the status of components and features related to high availability and ER servers. These are the main ways for monitoring HDR, RSS, SDS, CLR and ER:

- ▶ Message log file (`MSGPATH`) and console device/file (`CONSOLE`)
- ▶ Event alarms
- ▶ `onstat` utility
- ▶ `sysmaster` database
- ▶ Open Administration Tool (OAT)

In this section, we briefly describe the first four methods and then show an example of the MACH 11 feature of the OAT.

More information: For more details about using the message log file, console, event alarms, `onstat` utility, and `sysmaster` database for monitoring availability and replication features of IDS, refer to the Redbooks publication *Informix Dynamic Server 11: Extending Availability and Replication*, SG24-7488, which is available in softcopy at the following address:

<http://www.redbooks.ibm.com/abstracts/sg247488.html?Open>

3.5.1 Checking the message log file and console

The message log file, defined by the ONCONFIG variable MSGPATH, and the console device or file, defined by the ONCONFIG variable CONSOLE, contain informational messages that pertain to the status and errors of IDS. A DBA can find important information about the status of HA and ER servers by monitoring these files.

3.5.2 Event alarms

IDS uses the event alarm feature to report situations that require the attention of a DBA. The events can have many severity levels, such as *Attention*, *Emergency* or *Fatal*. To use the event alarm feature, set the ALARMPROGRAM configuration parameter to the full path name of one of the shell scripts provided with IDS, as shown in Example 3-1.

Example 3-1 ALARMPROGRAM in ONCONFIG file

```
ALARMPROGRAM /usr/informix/etc/alarmprogram.sh
```

You can either customize the ALARMPROGRAM scripts provided with IDS, or write and use your own shell script, batch file, or binary program.

For further details about configuring the event alarms for HA and ER servers, refer to the *IBM Informix Dynamic Server Administrator's Guide*, G229-6359, or to the *IBM Informix Dynamic Server Enterprise Replication Guide*, G229-6371.

3.5.3 The onstat utility

The **onstat** utility is the primary tool for monitoring the status of IDS. Similar to most of the other server features, **onstat** provides several options for checking the status of HA and ER servers.

3.5.4 The sysmaster database

The sysmaster database provides a set of tables related to HA and ER servers. The data in these tables is a reflection of the information that can be found with the **onstat** utility, and they can be easily accessed with the SQL interface. The database also provides the interface to plug in status data to existing monitoring solutions provided by either third-party, open-source, or inhouse applications.

3.5.5 The Open Admin Tool

The OAT is currently a PHP-based open source project from IBM that can be used for monitoring HA cluster servers via a Web-based interface. In this section, we provide an example of how to use the cluster functionality of the OAT.

More information: You can also use OAT for monitoring and administering other features and components of IDS. For more details about OAT, refer to 2.4, “Installing the Open Admin Tool” on page 55, and 5.5, “The Open Admin Tool for administration” on page 192.

Consider that you have an HA cluster with one primary database server (*newyork*), an SDS (*newyork_c1*), an HDR secondary (*miami*), and an RSS (*losangeles*). You have already configured the servers by using the Connection Admin page and included the latitude and longitude for every server. After a login to the Informix server *newyork* in the OAT, you see an overview page.

To access the Cluster functionality of OAT, under Menu in the left pane of the overview page, click **Mach 11**.

Click **Find Clusters** to start the Cluster Discovery. A message window opens as shown in Figure 3-21. Click **OK** to proceed with the discovery.

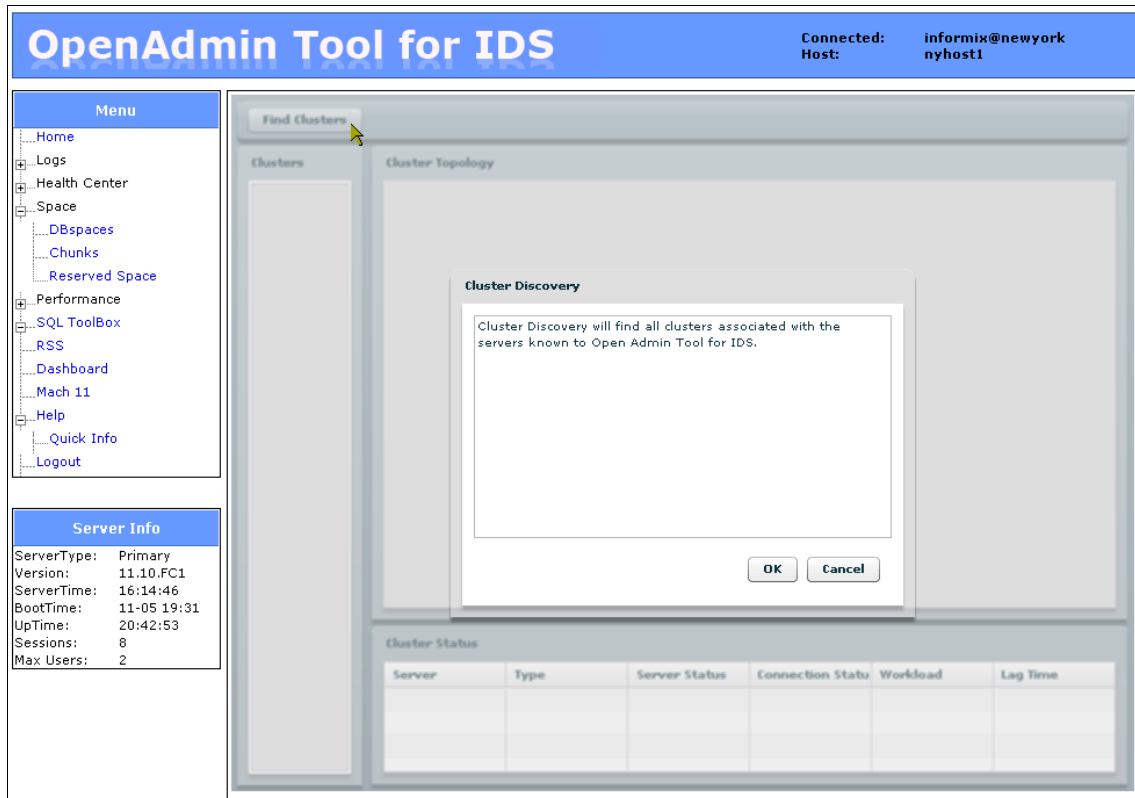


Figure 3-21 Cluster page with Cluster Discovery message

Another message window (Figure 3-22) opens that indicates that this action might take a few minutes to complete.

The screenshot displays the OpenAdmin Tool for IDS interface. At the top, a blue header bar contains the title "OpenAdmin Tool for IDS" on the left and connection information "Connected: informix@newyork" and "Host: nyhost1" on the right. The main interface is divided into several sections:

- Menu:** A vertical list of navigation options including Home, Logs, Health Center, Space (with sub-items DBspaces, Chunks, Reserved Space), Performance, SQL ToolBox, RSS, Dashboard, Mach 11, Help, Quick Info, and Logout.
- Server Info:** A box containing server details: ServerType: Primary, Version: 11.10.FC1, ServerTime: 16:14:46, BootTime: 11-05 19:31, UpTime: 20:42:53, Sessions: 8, and Max Users: 2.
- Find Clusters:** A button at the top of the main content area.
- Cluster Discovery Message Window:** A central dialog box with the title "Cluster Discovery" and the text "Cluster Discovery is looking for primary servers. This may take a few minutes...". It features a circular loading icon and an "OK" button.
- Cluster Status Table:** A table at the bottom right with the following structure:

Server	Type	Server Status	Connection Statu	Workload	Lag Time

Figure 3-22 Cluster page with message window

After discovering a new cluster, OAT shows a list of servers with the respective primary or secondary type, as depicted in Figure 3-23. Change the **Cluster Name**, if you want, and then click **OK** to continue.

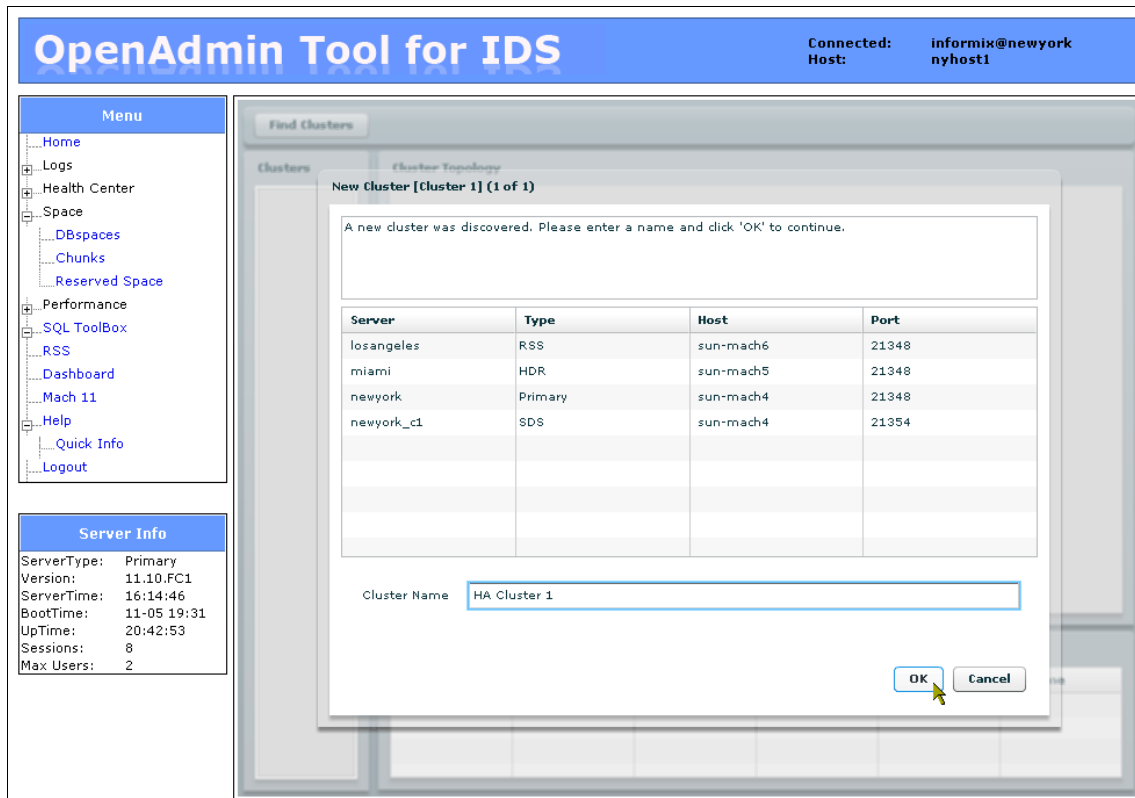


Figure 3-23 List of servers for cluster discovered

OAT indicates that the discovery is now completed, as shown in Figure 3-24. Click the created icon **HA Cluster 1**, in this example, with the cluster name chosen.

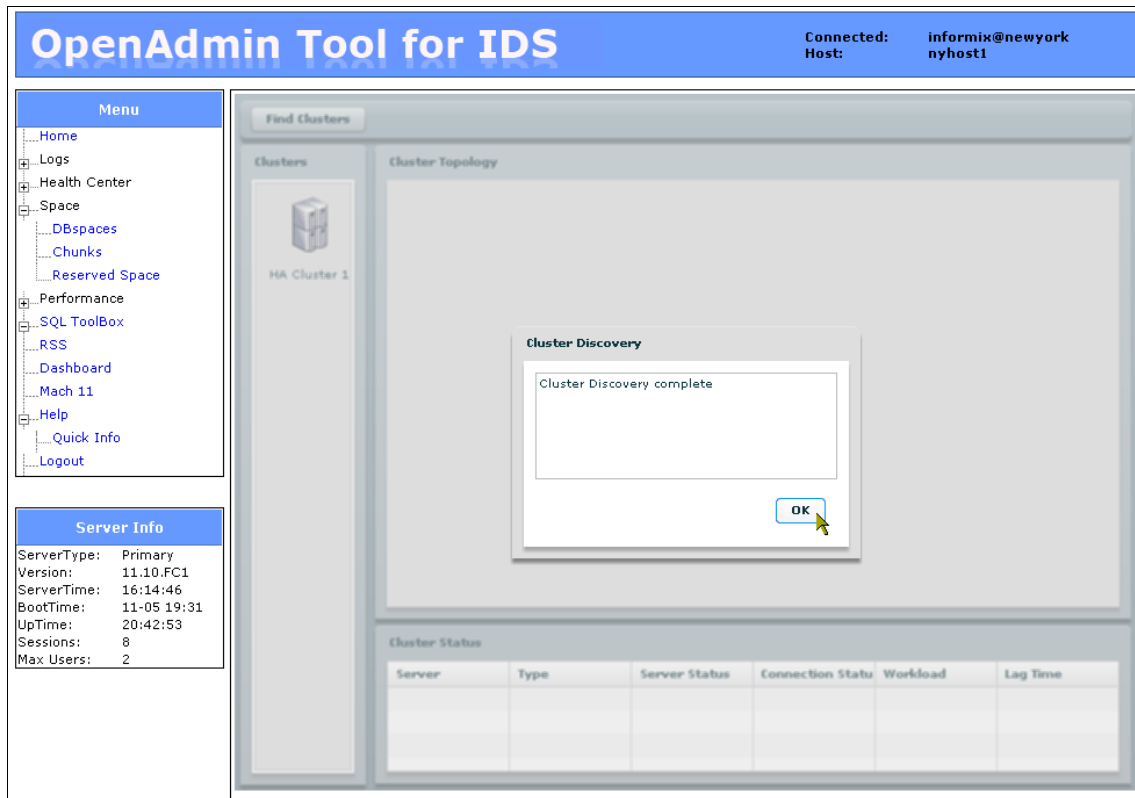


Figure 3-24 Cluster Discovery complete

The message Examining Cluster is displayed. After a few seconds, you see the cluster topology page, as depicted in Figure 3-25.

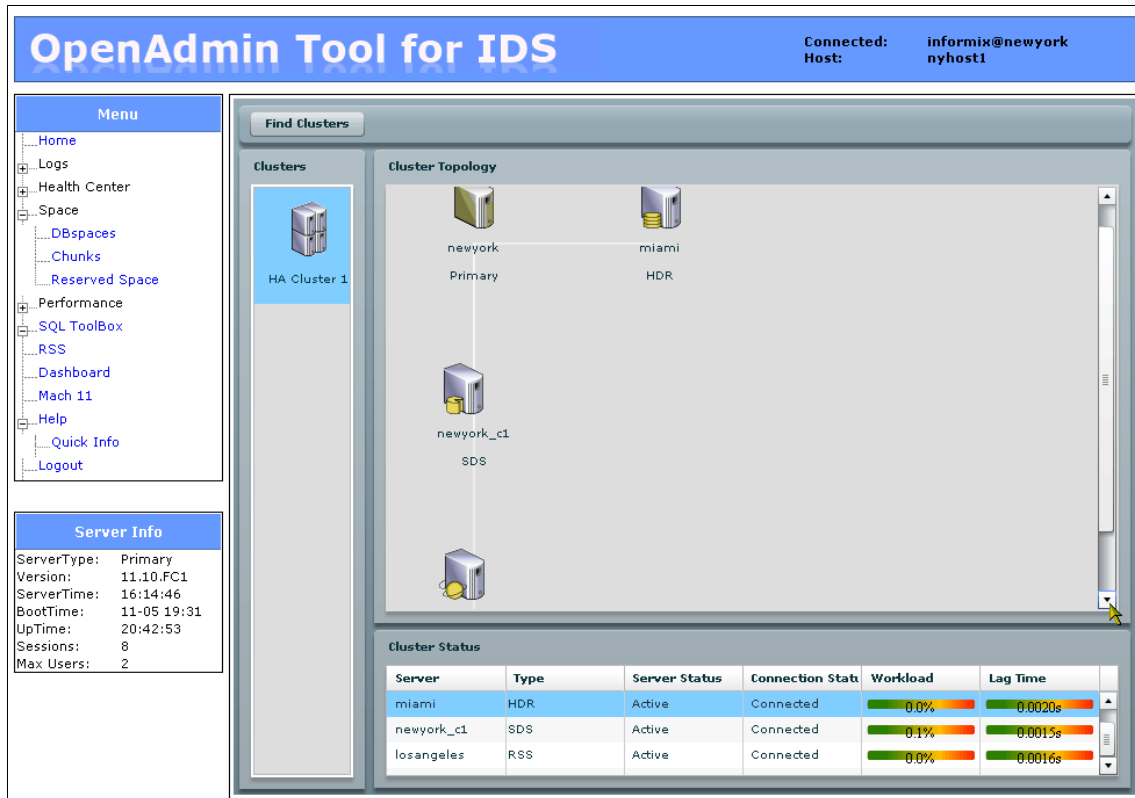


Figure 3-25 Cluster topology

You can use the cluster page to monitor the status of the servers in the HA cluster.



Robust administration

The Database Server Administrator (DBSA) plays a crucial role in the proper functioning of the database systems. An Informix DBSA has the following typical responsibilities among others:

- ▶ Installing the database software
- ▶ Designing the disk (chunk) and dbspace layout
- ▶ Fine tuning the database for optimal performance
- ▶ Implementing user and data access controls
- ▶ Handling backup and restore
- ▶ Ensuring business continuity (high availability) and data integrity

The role of the DBSA is a more powerful role than a DBA, who is typically only responsible for a particular database. The Informix Dynamic Server (IDS) contains a rich set of features that provide DBSAs with a framework to create a robust environment for a high performance database system.

In this chapter, we discuss the following topics:

- ▶ Disk management
The benefit from direct I/O on cooked chunks; how to optimize the dbspace layout and select the partitioning techniques and table types that are best suited for your environment

- ▶ Performance and autonomic features
How to meet your recovery time objective (RTO) and benefit from self-tuning techniques such as automatic checkpoint, automatic least recently used (LRU), and asynchronous input/output (AIO) tuning
- ▶ Security
How to implement database connection security, privileges on database objects using role-based access control (RBAC), and multi-level user and data access policies using label-based access control (LBAC)
- ▶ Backup and restore
Meeting your recovery point objective (RPO) using the most suitable backup and restore technique for your configuration

4.1 Disk management

Database performance relies heavily on the performance of the disk I/O subsystem. System administrators and DBAs play a crucial role in the design of an I/O system that provides optimal transaction throughput. In this section, we discuss about the disk-related components in IDS, such as chunks, dbspaces, tables, and fragmentation techniques, that can help the DBA achieve this goal.

4.1.1 Raw chunks versus cooked chunks

You can use either the standard operating system (*cooked*) files or raw disk devices as *chunks* to store data. Cooked files are easier to manage and extend but have a performance overhead due to file system caching. Buffering of data in the file system cache causes data to be copied twice. For example, during a read, data is copied from the disk to the file buffer cache and then from the file buffer cache to the application buffer. It also consumes portion of the system memory.

Raw devices, however, are much faster than cooked devices because they bypass the file system caching and use kernel AIO (KAIO). But, they are comparatively more difficult to manage and extend.

Cooked files have the benefit of a high cache hit rate, read-ahead, and asynchronous I/O. However they are not needed because the database server has its own buffering and read-ahead mechanism.

Direct I/O on cooked chunks

Some of the file system vendors support direct I/O on cooked devices, which bypasses the file system cache. Direct I/O on cooked devices eliminates the overhead of copying data twice. The data is read directly from the disk to the application buffer. However, direct I/O generally requires that the data be aligned on disk sector boundaries (512 or 1024 bytes).

Direct I/O is available on AIX, HP-UX, Solaris, Linux, and Windows, provided that it is supported by the underlying file system. Some of the file systems that support Direct I/O are Solaris UFS, VxFs, JFS, and NTFS. Refer to the man page of the `mount` command to find the argument that enables direct I/O.

IDS 11 supports direct I/O on cooked devices only if it is supported by the file system for the page size of the dbspace chunk. Direct I/O is used by default on Windows (NTFS). To enable it on UNIX, set the onconfig parameter `DIRECT_IO` to 1. If the database server is configured to use KAIO, IDS uses KAIO along with direct I/O on cooked devices. The performance of direct I/O and KAIO on cooked files is close to raw device performance.

For temporary dbspaces, use of the file system buffering for regular files can be faster than using direct I/O because there is no need to synchronize writes to ensure that writes are committed to disk. Therefore, for regular files belonging to temporary dbspaces, IDS does not use direct I/O, even if direct I/O is enabled.

Recommendations

We recommend that you use raw disk devices for the best performance. Based on your environment, requirements, and benchmark tests on your table layout, you can decide on the option that is the most optimal and convenient. Even though all your chunks are on raw devices, you might want to set `DIRECT_IO` if you load large number of rows by using flat files that are generated from heterogeneous systems. One such example is a data warehouse environment where a large number of rows is bulk loaded from various data marts running on other databases.

On Windows, both raw disks and NTFS use KAIO. Because NTFS files are a more standard method of storing data, you can use NTFS files instead of raw disks because it also supports direct I/O. Consider using raw disks if your database server requires a large amount of disk access.

4.1.2 Managing dbspaces

The design of the dbspace layout is one of the major factors that determines the database server performance. In this section, we discuss creating and managing dbspaces to get maximum benefit from the database server.

You can store highly accessed tables or critical dbspaces (root dbspace, physical log, and logical log) on your fastest disk drive to improve performance. By storing critical data on separate physical devices, you ensure that when one of the disks holding noncritical data fails, the failure affects only the availability of data on that disk.

To reduce contention for the root dbspace, the physical and logical logs can be moved out from the root dbspace to a separate disk.

Page size

The default system page size is platform dependent (4K on Windows and 2K on most UNIX platforms). However, you might want to create multiple dbspaces with differing page sizes that are multiples of the system page size. Each page size can have its own BUFFERPOOL setting in the onconfig file. The maximum allowable page size is 16K.

Larger page sizes offer the following advantages among others:

- ▶ Reduced depth of B-tree indexes, even for smaller index keys
- ▶ Decreased checkpoint time
- ▶ The grouping of long rows on the same page, which otherwise spans multiple pages for the default page size

Tblspace tblspace extents

Each dbspace contains a table space called the *tblspace tblspace* that describes all tblspaces in the dbspace. When creating a dbspace, the default first and next extent sizes for *tblspace tblspace* are 250 and 50 pages, where for non-root dbspaces, the defaults are 50 and 50 pages. If your database has a large number of tables, these defaults can cause fragmented extents, some of which might reside in non-primary chunks, which can impact performance.

At the time of disk initialization (oninit -iy), you can use the TBLTBLFIRST and TBLTBLNEXT configuration parameters to specify the first and next extent sizes for the *tblspace tblspace* belonging to the root dbspace. For non-root dbspaces, you can use the **onspaces** utility to specify the initial and next extent sizes for the *tblspace tblspace*, when creating this dbspace. The first and next extent sizes cannot be changed after the creation of the dbspace. You cannot specify these extent sizes for temporary dbspaces, sbspaces, blobspaces, or external spaces.

The number of pages in the *tblspace tblspace* is equal to the total number of table and detached index fragments including any system objects that reside in the dbspace. As shown in Example 4-1 on page 117, *dbs4* is created with an initial extent size and next extent size of 2 MB and 1 MB for *tblspace tblspace*.

The **oncheck -pe** output confirms that the first extent size is 1000 pages (for 2K page size).

Example 4-1 Specifying tblspace tblspace extents

```
$ onspaces -c -d dbs4 -p /opt/dbspaces/dbs4 -o 0 -s 10240 -ef 2000 -en 1000
```

```
$ oncheck -pe  
>>>>>
```

Chunk Pathname	Pagesize(k)	Size(p)	Used(p)	Free(p)
9 /work3/ssajip/INSTALL/dbspaces/dbs4	2	5120	1003	4117

Description	Offset(p)	Size(p)
RESERVED PAGES	0	2
CHUNK FREELIST PAGE	2	1
dbs4:'informix'.TBLSpace	3	1000
FREE	1003	4117

Multiple partitions in a single dbspace

One of the commonly used techniques for tables fragmented by expression is to fragment based on a date range to facilitate easy roll-in and roll-out of data. Each expression resides in a single dbspace. If the table has a large range of date expressions, the DBA must create a large number of dbspaces, and managing those dbspaces can be non-trivial. Also the maximum number of pages allowed in a dbspace is approximately 16 million.

You can circumvent these limitations by configuring multiple partitions in a single dbspace. Fragment elimination now eliminates partitions based on the fragmentation strategy. As shown in Example 4-2, we create a table with four partitions in dbspace dbs1 and attach a new partition, again residing on dbs1. The query plan for the SELECT shows that only part5 and part4 partitions are scanned, where the other three partitions are eliminated.

Example 4-2 Multiple partitions in a dbspace

```
CREATE TABLE shipment (item_number int, ship_date date, ship_locn  
varchar(20))
```

```
FRAGMENT BY EXPRESSION
```

```
PARTITION part1 (month(ship_date) = 1) IN dbs1,  
PARTITION part2 (month(ship_date) = 2) IN dbs1,  
PARTITION part3 (month(ship_date) = 3) IN dbs1,  
PARTITION part4 REMAINDER IN dbs1;
```

```
INSERT INTO shipment values (1, '2007-01-01', "cleveland"); -- part1  
INSERT INTO shipment values (2, '2007-02-01', "colorado"); -- part2
```

```

INSERT INTO shipment values (3, '2007-04-01', "boston");    -- part4

CREATE TABLE new1 (item_number int, ship_date date, ship_locn
varchar(20)) IN dbs1;

-- this ALTER causes row 3 to move from part4 to part5
ALTER FRAGMENT ON TABLE shipment
ATTACH new1 AS PARTITION part5 (month(ship_date) = 4) AFTER part3;

set explain on ;
select * from shipment where ship_date = '2007-04-01';
EOF

$ cat sqexplain.out
QUERY: (OPTIMIZATION TIMESTAMP: 01-22-2008 15:38:30)
-----
select * from shipment where ship_date = '2007-04-01'

Estimated Cost: 3
Estimated # of Rows Returned: 1

1) ssajip.shipment: SEQUENTIAL SCAN (Serial, fragments: 3, 4)

Filters: ssajip.shipment.ship_date = 2007-04-01

```

Blobspaces and sbspaces

If your application stores graphic or satellite images, video or audio clips, formatted spreadsheets, or digitized voice patterns, you have the option of using simple large objects (TEXT or BYTE data type) or smart large objects (BLOB or CLOB data type). TEXT and BYTE data types are stored in blobspaces where BLOB and CLOB data types are stored in sbspaces. A BLOB is a binary large object and a CLOB is a character large object.

IDS supports simple large objects primarily for compatibility with earlier versions of Informix applications. When you write new applications that need to access large objects, we recommend that you use smart large objects to hold character (CLOB) and binary (BLOB) data. CLOBs can be used only for text data, where BLOBs can be used for any binary data.

Smart large objects have the following advantages over simple large objects:

- ▶ They can store up to 4 TB as opposed to 2 GB.
- ▶ They support random access to the data.
- ▶ You can read or rewrite only specified portions of the smart large object.

- ▶ Data logging can be turned on or off unlike simple large objects where it is always on.

Temporary dbspaces

Temporary dbspaces are never backed up, nor are they physically or logically logged. Therefore, if you create a logged temporary table in a logged database, it resides only in the logged (non-temp) dbspaces of the DBSPACETEMP configuration parameter.

Decision Support Systems (DSS) or data warehouse environments generally run multiple complex queries that do large *sort*, *group*, or *join* operations. Depending on your PDQ settings, these operations can overflow to disk by creating internally generated temporary tables in these temp dbspaces. As a result, DSS queries can use significant temporary dbspace. User specified temporary tables that are non-logged also reside in these temp dbspaces. If you explicitly create a fragmented temp table and the PDQ setting is greater than 0, IDS uses parallel inserts to the temporary dbspaces if multiple dbspaces are specified in DBSPACETEMP. You get even better performance if these temporary dbspaces are on different disk controllers.

The database server keeps track of the most recently used temporary dbspace and uses the next available dbspace (in a round-robin pattern) to allocate I/O operations approximately evenly among those dbspaces. You can define a different page size for temporary dbspaces, so that the temporary tables have a separate buffer pool.

Extspaces

An extspace is a logical name that is associated with an arbitrary string that signifies the location of external data. The resource that the extspace references depends on the user-defined access method for accessing its contents. The server does not manage this space. It is created by using the **onspaces** utility.

Extspaces can be used to access the following items:

- ▶ Database tables from other vendors
- ▶ Data stored in sequential files
- ▶ Remote data stored across a network

Virtual tables/index interface and basic text search (BTS) are two areas where extspaces are used.

4.1.3 Table types

For data warehousing or other applications where a large number of rows (in millions or billions) are loaded regularly, it can take an extended time to load the rows if the table is logged. You can use the following steps to overcome this situation:

1. Drop any constraints or index on the *standard* table.
2. Use the ALTER TABLE to alter the table type from *standard* to *raw*.
3. Load the rows into the raw table by using the High Performance Loader (HPL) in express mode.
4. Perform a level 0 backup of the non-logging table.
5. Change the table type back to *standard*.
6. Add the constraints and recreate the index.

4.1.4 Data partitioning

Data partitioning is an intelligent way to distribute or fragment data or index entries across multiple disks to get the advantage of parallel disk I/O operations. It also enables the SQL operations to be segmented into subtasks and executed in parallel. The SQL optimizer can also eliminate data or index fragments based on expressions that benefit query performance. IDS supports round-robin and expression-based fragmentation.

For DSS, data can be fragmented, and the index can be non-fragmented and can reside in a different dbspace (detached index). For DSS environments that contain tables that are accessed sequentially, or if there are no columns that can be used as a fragment expression, you can choose a round-robin fragmentation. Round-robin fragmentation evenly distributes data across the disks.

For large DSS tables that contain columns representing date, region, or country, expression-based tables are most ideal because they can benefit from fragment elimination. Also fragments can be detached or attached based on, for example, the date column. For example, every month-end processing can involve detaching the fragment for the first month of last year and attaching the fragment that contains data for the new month.

For OLTP environments, you can fragment the data and index to reduce contention among sessions. Smaller tables do not need to be fragmented because the overhead of creating the scan threads to access the fragments can exceed the time taken to sequentially scan the table.

For expression fragmented tables, arrange fragmentation expressions, so that the most restrictive condition for each dbspace is tested within the expression first. The expression that is most likely to be false can be placed first, so that fewer conditions are evaluated before the database server moves to the next fragment as demonstrated in the following example.

The database server tests all six of the inequality conditions when it attempts to insert a row with a value of 25:

```
x >= 1 and x <= 10 in dbspace1  
x > 10 and x <= 20 in dbspace2  
x > 20 and x <= 30 in dbspace3
```

By comparison, only four conditions in the following expression need to be tested: the first inequality for dbspace1 ($x \leq 10$), the first for dbspace2 ($x \leq 20$), and both conditions for dbspace3:

```
x <= 10 and x >= 1 in dbspace1  
x <= 20 and x > 10 in dbspace2  
x <= 30 and x > 20 in dbspace3
```

4.2 Predictable fast recovery

In the continuous availability world of today, businesses rely on their database systems to be up and running for 24x7 every day of the year. Even a small duration of downtime can severely impact the business continuity and cause loss of revenue or credibility. If there is an unexpected outage, the database system is expected to be back online as fast as possible. The time it takes to come online depends on the transaction load at the time of the failure in relation to the last checkpoint. But, if a handle is available to the DBA to predict the time it takes for fast recovery or crash recovery to complete, they can plan accordingly.

Fast recovery time is the time from when the DBA starts the IDS server until the server comes to an online or quiescent mode. It is comprised of the following times:

- ▶ Boot up time

This is the time it takes to boot up the server infrastructure. Typically most of this time is spent in the initialization of the shared memory.

- ▶ Physical recovery time

This is the physical recovery restores the database to a physically consistent state. This time depends on the number of pages being physically restored and the I/O speed of the physical log disk.

► Logical recovery time

This time applies the logical log records to bring the database to a transactionally consistent state. This time depends on the number of log records to be applied and the I/O speed of the disk I/O where the logical log resides.

In the following sections, we describe how to set and tune the configuration parameter to predict fast recovery time. We also discuss the effects of this setting on other configuration parameters.

4.2.1 Benefits of RTO_SERVER_RESTART over CKPTINTVL

In this section, we focus on the RTO_SERVER_RESTART and CKPTINTVL configuration parameters and their benefits. These are two mutually exclusive parameters with different benefits.

The configuration parameter CKPTINTVL can be used to control the crash recovery time, but it does not take into consideration the transactional workload. The DBA must run tests to time the fast recovery based on the various settings of CKPTINTVL to arrive at an optimum setting. These tests must be run under various workload scenarios, which is not convenient.

For an environment that mandates a strict RTO policy, the RTO_SERVER_RESTART configuration parameter can be used instead of CKPTINTVL. These two parameters are mutually exclusive. If RTO_SERVER_RESTART is set, CKPTINTVL is ignored. RTO_SERVER_RESTART can be set to a value in seconds.

When the RTO_SERVER_RESTART parameter is set, IDS controls the frequency of checkpoints based on past fast recovery performance and automatically adjusts for variable workloads to ensure that IDS can meet the target fast recovery time.

Tuning RTO_SERVER_RESTART

More frequent checkpoints are triggered for a lower value of this setting. You can start with an aggressive (smaller) value of RTO_SERVER_RESTART that is suitable for your environment, and then monitor the frequency and type of checkpoints by using the **onstat -g ckp** command. If the checkpoints are too frequent and are affecting transactional performance, use the **onmode -wf** command to dynamically increase RTO_SERVER_RESTART as shown in Example 4-3 on page 123.

Example 4-3 Dynamically changing RTO_SERVER_RESTART

```
$ onstat -c | grep RTO_SERVER_RESTART
RTO_SERVER_RESTART 60

$ onmode -wf RTO_SERVER_RESTART=120
Value of RTO_SERVER_RESTART has been changed to 120.
```

4.2.2 RTO: Dependent onconfig parameters

In this section, we discuss the various onconfig parameters that can have an effect on the RTO policy.

As part of logical replay, each of the log records can generate an I/O. If that I/O requires a page to be read from disk, log replay performance will be adversely affected. Also, these random I/Os can cause unpredictable recovery time.

When RTO_SERVER_RESTART is set, IDS saves additional before images of modified pages as part of transaction management. During physical recovery, these pages are seeded into the buffer pool to eliminate random I/O and make the recovery more predictable. The random I/O is replaced by block sequential I/O. This means that now you must configure the system with an increased physical log space to accommodate all the buffer pools. Typically, this additional physical log activity has little or no impact on transaction performance.

PHYSFILE: Physical log size

For IDS 11, the PHYSFILE configuration parameter can be set to 110 percent of the total size of buffer pools listed in BUFFERPOOL for optimum fast recovery performance. During fast recovery, the server can seed the physical log pages into the buffer pool and can handle the RTO policy more reliably. Having a large physical log does not impact performance. If your system has a large buffer pool and only part of it is used for updates, then you can specify a lower value.

Changing the physical log size, location, or both can only be done online in IDS 11. You can use the **onparams** command to change the physical log size, location, or both without requiring the server to be rebooted. However, this can only be done while in quiescent or admin mode. That means it must be done when access to the data can be interrupted.

Example 4-4 on page 124 shows how to change the physical log size from 9 MB to 11 MB. This example has just the default BUFFERPOOL setting for 2K page size. If you have BUFFERPOOL settings for other page sizes, you must sum them to calculate the optimum physical log size.

There are always two physical log buffers in the shared memory. The `onstat -l` output, therefore, shows half of the `PHYSFILE` value. Double buffering permits user threads to write to the active physical log buffer, while the other buffer is flushed to the physical log on disk.

Example 4-4 Changing physical log size

```
$ onstat -c | egrep 'PHYSFILE|BUFFERPOOL'
PHYSFILE          9000          # Physical log file size (Kbytes)
BUFFERPOOL
size=2K,buffers=5000,lrus=8,lru_min_dirty=50,lru_max_dirty=60
```

```
Physical log size = 1.1 x buffer_size x buffers = 1.1 x 2K x 5000
= 11000Kb
```

```
$ onstat -l
```

```
Physical Logging
Buffer bufused  bufsize  numpages  numwrits  pages/io
P-1  16      64      15      0      0.00
      phybegin      physize  phypos   phyused  %used
      1:15525      4500    34      16      0.36
```

```
$ onparams -p -s 11000 -y
Log operation started. To monitor progress, use the onstat -l command.
** WARNING ** Because the physical log has been modified, a level 0
archive must be taken of the following spaces before an incremental
archive will be permitted for them: rootdbs
(see Dynamic Server Administrator's manual)
```

```
$ onstat -l
```

```
Physical Logging
Buffer bufused  bufsize  numpages  numwrits  pages/io
P-2  2      64      16      1      16.00
      phybegin      physize  phypos   phyused  %used
      1:20025      5500    0      2      0.04
```

```
A message is also written to the online.log:
11:14:21 Physical log size changed to 11000 KB.
```

Changing PHYSFILE and PHYSDBS: For IDS 11 and later releases, PHYSFILE and PHYSDBS in the onconfig file are only checked during the server initial creation time (oninit -i). You can change these values only by using the **onparams** command. Editing the onconfig file to change these values has no effect on the server.

PHYSBUFF

The default value for the physical log buffer size is 128 KB. When the RTO_SERVER_RESTART configuration parameter is enabled, the default physical log buffer size can be changed to 512 KB. If a smaller value is to be used, IDS writes a performance advisory message to the online.log indicating that optimal performance might not be attained, as shown in Example 4-5. Using a physical log buffer smaller than the default size impacts only performance, not transaction integrity.

Example 4-5 PHYSBUFF not optimal

10:40:18 **Performance Advisory:** The current size of the physical log buffer is smaller than recommended.
10:40:18 **Results:** Transaction performance might not be optimal.
10:40:18 **Action:** For better performance, increase the physical log buffer size to 128.

RAS_PLOG_SPEED and RAS_LLOG_SPEED

The configuration parameters RAS_PLOG_SPEED and RAS_LLOG_SPEED are used to store the rate at which the physical and logical log can be recovered during crash recovery. The units are in pages per second. Each time the server is recycled and it goes through physical and logical recovery, it updates these configuration parameters based on the time it takes for the recovery to complete. These parameters are used by the server to determine if the RTO policy can be met. They are set to 0 by default in the onconfig.std file.

RAS_PLOG_SPEED is updated during physical recovery only if the number of pages physically recovered is more than 10,000 pages. RAS_LLOG_SPEED is updated during logical recovery if the number of pages logically recovered is more than 1000 pages. This is done to get a more accurate measurement.

Important: RAS_PLOG_SPEED and RAS_LLOG_SPEED are autoupdated and are not supposed to be changed by the DBA. These values are listed in the onconfig only to inform you of the physical and logical recovery speed of your disks.

BUFFERPOOL

During fast recovery, if all the updated pages do not fit into the buffer pool, the server cannot meet the RTO_SERVER_RESTART policy because of the additional I/O. The server writes a performance advisory to increase the buffer pool size as shown in Example 4-6. Alternatively, you can decrease the value of the RTO_SERVER_RESTART configuration parameter to manage this.

Example 4-6 Advisory to increase BUFFERPOOL

15:06:23 **Performance Advisory:** The default buffer pool is very dirty prior to logical log replay.

During fast recovery roll forward, logical log pages are placed in the default buffer pool for better performance. For best recovery performance, the whole physical log and some of the logical log should be able to fit into the default buffer pool.

15:06:23 **Results:** Fast recovery performance might not be optimal.

15:06:23 **Action:** Dynamic Server automatically accounts for any performance impact on the next fast recovery occurrence. This might cause more frequent checkpoints to occur. For better performance, increase the default buffer pool size by 5%.

4.2.3 When not to use RTO_SERVER_RESTART

If you lower the value of the RTO_SERVER_RESTART, the server triggers more frequent automatic checkpoints. The frequent checkpoints do not adversely affect transaction processing because of the non-blocking functionality of checkpoints in IDS 11, but cause the cleaner threads (flush_sub) to consume more CPU cycles.

If RTO_SERVER_RESTART is set, the server does additional physical logging as described in 4.2.2, “RTO: Dependent onconfig parameters” on page 123.

Hence if your database system is a DSS, data warehouse, or any other environment that does not require a strict RTO policy, we recommend that you leave RTO_SERVER_RESTART to 0 (OFF), so that the server does not do the extra physical logging and frequent checkpointing. The default value is 0 in the onconfig.std file.

4.3 Automatic tuning

IDS 11 has the self-tuning configuration parameters of `AUTO_CKPTS`, `AUTO_LRU_TUNING`, and `AUTO_AIOVPS` that are ON by default. In the following sections, we explain why these parameters should be left with their default value and discuss scenarios when they might be set to OFF.

4.3.1 AUTO_CKPTS

The default value of `AUTO_CKPTS` is 1 (ON) in the `onconfig.std` file. When `AUTO_CKPTS` is set to 1, the server calculates the minimum physical log size using `RAS_PLOG_SPEED`.

In Example 4-4 on page 124, we set `PHYSFILE` to 11 MB, which was 110% of the total buffer pool size. In that example, we had set `AUTO_CKPTS` to 0. After we set `AUTO_CKPTS` to 1, we saw a performance advisory message in the `online.log` as shown in Example 4-7 to increase the log size to 14 MB. If the log size is less than this minimum value, automatic checkpointing is disabled. You can use the `onparams -p` command to increase `PHYSFILE` to the recommended value of 14 MB.

Example 4-7 Auto checkpoint recommended physical log size

```
14:28:42 Performance Advisory: The physical log is too small for  
automatic checkpoints.
```

```
14:28:42 Results: Automatic checkpoints are disabled.
```

```
14:28:42 Action: To enable automatic checkpoints, increase the  
physical log to at least 14000 KB.
```

The server also calculates the minimum logical log space based on `RAS_LLOG_SPEED` and writes an advisory message with the recommended value.

If automatic checkpoint is ON, the server also calculates the time it takes to flush the buffer pool and writes an advisory to the `online.log` if the physical log size is too small. By the time the flushing completes, transactions can potentially use up the remaining physical log causing checkpoints to block out transactions. The server writes this informative message, so that the DBA can take a corrective action, as shown in Example 4-8 on page 128.

Example 4-8 Physical log and BUFFERPOOL relation

14:18:41 **Performance Advisory:** Based on the current workload, the physical log might be too small to accommodate the time it takes to flush the buffer pool.

s14:18:41 **Results:** The server might block transactions during checkpoints.

14:18:41 **Action:** If transactions are blocked during the checkpoint, increase the size of the physical log to at least 14000 KB.

Apart from these useful performance advisories, AUTO_CKPTS trigger automatic checkpoints based on the past checkpoint performance and the current physical and logical log usage to avoid transaction blocking. Therefore, we recommend that you do not change the default ON setting of AUTO_CKPTS unless you are sure you want the checkpoints to occur only on the specific conditions such as 75% physical log, CKPTINTVL, and admin or archive events. If the physical and logical resources are such that automatic checkpoints are triggered every 35 seconds or less, the server disables the automatic checkpoint as shown in Example 4-7 on page 127. In such a case, you increase the resource as suggested in the *Action field* of the advisory message. By doing this, you have the benefit of automatic checkpoints.

To change the default setting, you can use the **onmode -wm** (temporary for this current server session) or **onmode -wf** (permanent change in onconfig) options to turn it OFF (value 0).

4.3.2 AUTO_LRU_TUNING

In IDS versions prior to 11, it was typical to set low thresholds for LRU_MIN_DIRTY and LRU_MAX_DIRTY. This setting limited the number of dirty buffers and improved checkpoint performance. The reduced checkpoint time blocked user transactions for a shorter duration as compared to the case where the LRU_MIN_DIRTY and LRU_MAX_DIRTY values were higher. It might even be set to a value < 1 if a large number of buffer pools were configured on the system. With IDS 11, you can now relax the LRU settings, since checkpoints do not block transactions. This should result in a dramatic increase in performance.

The following settings are a good starting point for setting the LRU flushing parameters:

- ▶ lru_min_dirty=70
- ▶ lru_max_dirty=80

You can also let IDS auto tune the LRU settings depending on its usage by setting the AUTO_LRU_TUNING configuration parameter to 1.

The database server automatically tunes LRU flushing whenever a page replacement occurs. After a checkpoint has occurred, if a page-replacement foreground write occurred during the previous checkpoint interval, the database server increases the LRU settings by 10 percent. It continues to increase the LRU flushing at each subsequent checkpoint until page-replacement foreground writes stops or until the `lru_max_dirty` for a given buffer pool falls below 10 percent. For example, if a page-replacement foreground write occurs and the LRU settings for a buffer pool are 80 and 90, the database server adjusts these to 76 and 85.5.

In addition to foreground writes, LRU flushing is tuned more aggressively whenever a page fault replaces high priority buffers and non-high priority buffers are on the modified LRU queue. Automatic LRU adjustments only make LRU flushing more aggressive. They do not decrease LRU flushing. Automatic LRU adjustments are not permanent and are not recorded in the `ONCONFIG` file.

When `AUTO_LRU_TUNING` and `RTO_SERVER_RESTART` are set and the time to flush the buffer pools during checkpoint processing takes longer than the `RTO_SERVER_RESTART` policy setting, a performance advisory is written to the `online.log` as shown in Example 4-9.

Example 4-9 Auto LRU tuning

```
Performance advisory: The time to flush the bufferpool ## is longer
than RTO_SERVER_RESTART ##.
Results: The IDS server can't meet the RTO policy
Action: Automatically adjusting LRU flushing to be more aggressive.
Adjusting LRU for bufferpool - id ## size ##k
    Old max ## min ##    New max ## min ##
```

4.3.3 AUTO_AIOVPS

The tuning of AIOVPS was based on how it can accommodate the peak number of I/O requests. The *maxlen field* of the `onstat -g ioq` command shows the largest backlog of I/O requests for each file descriptor. The *gfd* column of this output can then be used to get the *pathname* from `onstat -g ioF`, which can then be mapped to the `onstat -d` output to get the chunk name. Generally, it is not detrimental to allocate too many AIO virtual processors.

With IDS 11, you can set `AUTO_AIOVPS` to enable IDS to add AIO VPS on an as-needed basis. The default value for `AUTO_AIOVPS` is 1. When using cooked file chunks, if AIO VPs are too busy, the server automatically increases the number of flushers and AIO VPs. However, we recommend that you monitor these automatic increases and the particular environment. For example, if a large

number of AIO VPs are automatically generated on a system with a small number of disks, performance can be negatively impacted.

Even if the database setup contains only raw devices that use KAIO, the server uses AIO for writing to the `online.log` and `sqexplain.out` files or reading from the `onconfig` file or load files.

4.4 Database connection security

In this section, we discuss the many options that are available for authenticating or restricting client connections to the database server.

4.4.1 OS password authentication

If the IDS client applications specify a user name and password at connection time, IDS does OS-level authentication based on the password and shadow files or interaction with the Network Information Service (NIS). IDS also supports *trusted hosts* defined by the `hosts.equiv` and `rhosts` files. You can configure the connection security settings by using the `r` and `s` options of the fifth column entry in the `sqlhosts` file. The `s` identifies database server-side settings, and the `r` identifies client-side settings. You can choose to use a combination of `rhosts` or `hosts.equiv` lookup for the server and `.netrc` lookup for the client.

Refer to the IDS administrator's guide, *IBM Informix Dynamic Server Administrator's Guide, Version 11.1*, G229-6359-01, for more details about these settings. The following sequence shows how you can disable the `rhosts` and `hosts.equiv` lookup and force remote clients to connect only when a password is entered:

Important: Do not disable the `hosts.equiv` lookup in database servers that are used in distributed database operations.

1. A user who is trusted on the machine can connect from a remote client without specifying the `USER` clause in the `CONNECT` statement. To make the user trusted, add the remote client to the `.rhosts` file in the user's home directory:

```
> connect to 'db1@ids_server';
```

```
Connected.
```

2. An s=0 entry in the sqlhosts prevents rhosts and hosts.equiv lookup. The database server is bounced after this change in sqlhosts:

```
sqlhosts entry:  
ids_server ontlitcp      ramsay 9801 s=0
```

3. Provide a password for the remote connection to succeed:

```
> connect to 'db1@ids_server';
```

```
956: Client host or user informix@falcon1.menlo.ibm.com is not  
trusted by the server.
```

```
> connect to 'db1@ids_server' user 'user1';  
ENTER PASSWORD:
```

```
Connected.
```

4.4.2 Pluggable Authentication Module

Applications that access sensitive information from a database need a higher level of authentication than just the basic OS password authentication. In such cases, they can use application-based authentication framework, such as the Pluggable Authentication Module (PAM). PAM enables system administrators to implement different authentication mechanisms for different applications. IDS supports PAM on UNIX and Linux. You can write your own PAM service library and save it in the /usr/lib/security directory. This library can either ask for a password or throw challenges to authorize the users.

The following example shows the basic steps that are required to enable PAM. The occurrence of s= 4 in the fifth field of the sqlhosts file represents a PAM entry that includes the PAM service name and the type of authorization, password or challenge.

1. Update the sqlhosts file entry for the PAM entry:

```
ids ontlitcp ramsay 9801 s=4,pam_serv=pam_ids,pamauth=(challenge)
```

2. Add an entry to the PAM configuration file /etc/pam.conf:

```
pam_ids auth required /usr/lib/security/pam_ids.so
```

3. Write a program to create the shared library pam_ids, so that it authenticates the user by defining the function pam_sm_authenticate. This function can throw various challenges. The remaining interfaces, including pam_sm_setcred, pam_sm_acct_mgmt, pam_sm_open_session, pam_sm_close_session, and pam_sm_chauthtok, can be left in dummy status.

4. Copy pam_ids.so to the /usr/lib/security directory.
5. Provide the information for the connection to the database when prompted:

```
dbaccess test -  
Your school name (MIT):  
Your maiden name (SUZE):: SUZE  
PAM Text Info:  
Database selected.
```

An incorrect reply throws an error and prevents the user from connecting to the database as shown in the following example:

```
$ dbaccess san -  
Your school name (MIT):  
Your maiden name (SUZE):: SUZ
```

```
1809: Server rejected the connection.
```

You can find the sample shared library used in the previous example at the following Web address:

<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0704anbalagan>

4.4.3 Lightweight Directory Access Protocol

Lightweight Directory Access Protocol (LDAP) authentication in Windows is set up and configured much like the PAM setup on UNIX and Linux. The authentication module is a dynamic link library (DLL) that usually resides in the %INFORMIXDIR%\dbssodir\lib\security directory. The parameters of the module are listed in the %INFORMIXDIR%\dbssodir\pam.conf file. The source code for a fully functional LDAP authentication module and samples of the required configuration files are included in the %INFORMIXDIR%\demo\authentication directory. The sqlhosts is set up exactly like the one shown in 4.4.2, “Pluggable Authentication Module” on page 131.

4.4.4 Password encryption

The password encryption protects a password when it must be sent between the client and the database server for authentication. Communication support modules (CSMs) can be used to enable password encryption. The password encryption libraries and connection options are specified in the conccsm.cfg CSM configuration file that resides by default in the \$INFORMIXDIR/etc directory. The \$INFORMIXDIR/lib/client/csm/libixspw.so library is provided by IDS. Example 4-10 on page 133 shows a sample configuration file that uses different

libraries for client and server. The `p=0` option signifies that the password is not mandatory, but if provided, it will be encrypted.

Example 4-10 Sample `concsm.cfg`

```
SPWDCSM("client=/usr/informix/lib/client/csm/libixspw.so,server=/usr/informix/lib/csm/libixspw.so", "", "p=0")
```

Handling denial-of-service flood attacks

You can use the configuration parameters `LISTEN_TIMEOUT` and `MAX_INCOMPLETE_CONNECTIONS` to reduce the risk of a hostile, denial-of-service (DOS) flood attack:

- ▶ `LISTEN_TIMEOUT` sets the incomplete connection timeout period. The default incomplete connection timeout period is 10 seconds.
- ▶ `MAX_INCOMPLETE_CONNECTIONS` restricts the number of incomplete requests for connections. The default maximum number of incomplete connections is 1024.

If the maximum limit is reached, a message as shown in the following example is in the online message log indicating that the system is under attack:

```
100 incomplete connection at this time.  
System is under attack through invalid clients on the listener port.
```

4.4.5 Stored procedures (`sysdbopen` and `sysdbclose`)

The stored procedure, `sysdbopen`, does not fall under a security umbrella, but allows the DBA to change session properties without changing the application itself. IDS automatically executes the appropriate `sysdbopen` procedure when a user connects to a database. Similarly the procedure `sysdbclose` executes during disconnect.

The DBA can create a `user.sysdbopen` or a `public.sysdbopen` procedure. The former has precedence over the latter, if both are present. You can include any `SET` or `SQL` commands that are valid inside a regular procedure. Example 4-11 on page 134 shows how you can set different isolation levels or `PDQPRIORITY` based on the user name. It shows that a `DSS` user must run with higher `PDQPRIORITY` and with a dirty read isolation as compared to an `OLTP` user who has to run with a lower `PDQ` setting and a repeatable read isolation.

Example 4-11 sysdbopen

```
database mydb;
create procedure dss_user.sysdbopen()
set isolation to dirty read;
set pdqpriority 40;
set optcompind '2';
end procedure;

create procedure oltp_user.sysdbopen()
set isolation to repeatable read;
set pdqpriority 1;
set optcompind '1';
end procedure;

create procedure public.sysdbopen()
set pdqpriority 10;
end procedure;

-- /tmp/collect_diag can be a shell script to collect any onstats
create procedure public.sysdbclose()
system "/tmp/collect_diag";
end procedure;
```

Persistence of SET PDQPRIORITY and SET ENVIRONMENT: SET PDQPRIORITY and SET ENVIRONMENT statements are not persistent for regular user-defined routines (UDRs). Their scope is local within the UDR. For sysdbopen, these two statements are persistent until the session ends unless they are explicitly changed within the session.

4.4.6 Administrator-only mode

The online mode of the engine can be changed into an administrator only mode by using **onmode -j**. This mode only allows the DBSA group or user informix to connect to the server. Any maintenance work can be run in this mode including running any SQL or Data Definition Language (DDL) commands. **onstat** prints this mode as Single-User, as shown in Example 4-12.

Example 4-12 DBA only mode

```
IBM Informix Dynamic Server Version 11.10.F      -- Single-User -- Up
00:02:31 -- 39936 Kbytes
```

If you want individual users to connect to the server when in single user mode, you can run the following command:

```
onmode -j -U <comma_separated_user_list>
```

You can also achieve this by setting the onconfig parameter `ADMIN_USER_MODE_WITH_DBSA` to 1 and `ADMIN_MODE_USERS` to the list of such users and bouncing the server with `oninit -j`.

When the server is changed to administration mode, all sessions for users other than user *informix*, the DBSA group users, and those identified in the user list will lose their database server connection.

If you want only user *informix* to connect to the server, you can set the configuration parameter `ADMIN_USER_MODE_WITH_DBSA` to 0. A value of 1 means that the DBSA group users, user *informix*, and administration mode users, as listed in `ADMIN_MODE_USERS`, can connect when the server is in administrator only mode.

4.5 Controlling data access

IDS supports two access-management systems: Mandatory Access Control (MAC) and Discretionary Access Control (DAC).

MAC is an access control policy for information management systems that handle sensitive or classified information. Multi-level security (MLS) is a well-known implementation of MAC that addresses requirements where multiple levels of security are required for organizations such as the Department of Defense. IDS supports MLS by using the LBAC, where you can control access to rows and columns based on security labels.

DAC is an access control policy that verifies whether the user has been granted the required privileges to perform an operation. DAC is a simpler system that involves lesser overhead than MAC and can be implemented in the following ways:

- ▶ Controlling who is allowed to create databases by using the `DBC_CREATE_PERMISSION` configuration parameter
- ▶ Restricting the users who are allowed to register external UDRs by using the `IFX_EXTEND_ROLE` configuration parameter
- ▶ Controlling operations on database objects by using roles (RBAC)

In the subsequent sections, we briefly discuss these DAC options and explain with some examples how you can benefit from the LBAC functionality.

4.5.1 Creating permissions

The `DBC_CREATE_PERMISSION` configuration parameter can be used to prevent unauthorized users from creating databases. It contains the list of users who are allowed to create databases. If you only want the user `informix` to create databases, it can be set as shown in the following example:

```
DBC_CREATE_PERMISSION informix
```

4.5.2 Security for external routines

External routines with shared libraries that are outside the database server can be security risks. The `IFX_EXTEND_ROLE` configuration parameter can be set to 1 to restrict users from registering external routines. When this is set, only the DBSA can grant or revoke permissions to specific users by using the `GRANT EXTEND TO` and `REVOKE EXTEND FROM` commands. Only these users will be able to create external routines.

Only *jdoe* can create external UDRs in a database, if the following statement is the only `GRANT EXTEND` statement in that database:

```
GRANT EXTEND TO 'jdoe';
```

4.5.3 Role-based access control

A *role* is a work-task classification. Privileges on database objects are granted to roles instead of users. By assigning privileges to roles and roles to users, you can simplify the management of privileges (RABC). They can be at the database, table, routine, type, or language level.

You can also create a default role and assign that role to individual users or to `PUBLIC` on a per-database level. The default role is automatically applied when a user establishes a connection with the database. This enables a user to connect to a database without issuing a `SET ROLE` statement. The default role can also be attained by setting the role in the `user.sysdbopen` procedure. Each user that is assigned to a default role receives the privileges of that role in addition to the other privileges that are granted individually to the user.

Example 4-13 on page 137 shows that user *jdoe* who works for the sales department has a default role of *sales_role* with all permissions on the *sales* table. *maryk* works for the human resource (HR) department and has all permissions on the *employee* table but no permissions on the sales table. *jdoe* can only access the *employee* and *department* columns from the *employee* table. He cannot access any confidential columns, such as the *salary* column, from this table.

Example 4-13 Roles and default roles

```
grant connect to 'jdoe';
grant connect to 'maryk';

create role sales_role;
create role hr_role;

revoke all on employee from "public";
revoke all on sales from "public";

grant select, insert, update, delete on sales to sales_role;

grant select, insert, update, delete on employee to emp_role;
grant select (emp_name, dept_name) on employee to sales_role;

-- routine delete_emp can only be executed by emp_role
grant execute on delete_emp to emp_role;

grant default role sales_role to 'jdoe';
grant default role hr_role to 'maryk';
```

4.5.4 Label-based access control

By using LBAC, you can control read and write access to individual rows and columns of data. If your database system contains highly confidential, private, or sensitive data that needs to be secure from any unauthorized use, you might want to control and limit access to this data by implementing LBAC.

When a user attempts to access a protected table, IDS enforces two levels of access control. The first level is DAC, which you can implement by using roles (RBAC). With DAC, IDS verifies whether the user who is attempting to access the table has been granted the required privileges to perform the requested operation on that table. The second level is LBAC, which controls access at the row level, column level, or both levels.

LBAC is implemented by creating new security objects that are made up of security components, security policies, and security labels. Only the users with the built-in role DBSECADM (database security administrator) can issue DDL statements that can create, alter, rename, or drop these security objects. The scope of the DBSECADM role is across all of the databases (instance wide) unlike the user defined roles whose scope is the database in which the role is created.

The security administrator can associate security *labels* with rows or columns in a table and with users (user labels). When a user attempts to access an LBAC-protected table object, the system compares the user label to the row or column label to determine if the user can access the data. These security *labels* belong to a security policy, which in turn is defined as a set of security *components*.

In the following section, we demonstrate the implementation of an LBAC security policy by using an example of an insurance company. We go through the steps to implement LBAC security and explain them with examples. One of the first steps is to identify tables that need security at either the row level, column level, or a combination of both.

Implementing row-level LBAC

The sales department of an insurance company has a sales table that can be accessed only by the employees in the sales department. The persons that belong to a particular region, such as West and East, can only access rows from their own region unless they work at a position (for example, Director or President) that is allowed to access row from other regions. The sales table is defined as follows:

```
CREATE TABLE sales (sales_date date, sales_person varchar(10), region
varchar(10), sales int);
```

To implement row-level security on this table:

1. Create security admin.

Create a new user ID, which is `lbacadmin` in this example, for the user who is responsible for creating and maintaining LBAC. The DBSA grants the DBSECADM role to this user ID by using the GRANT command:

```
GRANT DBSECADM TO 'lbacadmin';
```

The DBSECADM role is a built-in role that only the DBSA can grant. Unlike UDRs, whose scope is the database in which the role is created, the scope of the DBSECADM role is all of the databases of the IDS instance. It is not necessary for DBSA to re-issue the GRANT DBSECADM statement in other databases on the same server. Like all built-in roles of IDS, the DBSECADM role is enabled when it is granted, without requiring activation by the SET ROLE statement, and it remains in effect until it is revoked. Only a user who holds the DBSECADM role can issue the SQL statements that can create or modify security objects.

2. Create the security component.

The lbacadmin user creates the required security components. We use all of the available types of security label components in this example:

– Set

A collection of items where the order is not important. The user label must include all the components defined for the row to read/write the row:

```
CREATE SECURITY LABEL COMPONENT departments SET {'Marketing',  
'Sales', 'Support', 'Development'};
```

– Array

An *ordered* set where the highest privilege is first and that can be used to represent a simple hierarchy. You can only read data that is at your level or below, and you can write only at your level, which is called the *no read up, no write down* approach:

```
CREATE SECURITY LABEL COMPONENT levels ARRAY [ 'Board',  
'Executive', 'Director', 'Manager', 'Engineer' ];
```

– Tree

A more complex hierarchy that can have multiple nodes and branches:

```
CREATE SECURITY LABEL COMPONENT regions TREE ('corp_wide' root,  
'East' under 'corp_wide', 'West' under 'corp_wide', 'Central'  
under 'corp_wide', 'Northern California' under 'West', ..);
```

3. Create a security policy.

Define the security policy by using all the necessary security components. In this example, we use the three components that were previously defined:

```
CREATE SECURITY POLICY policy1 COMPONENTS departments, levels,  
regions  
RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL;
```

The RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL is the default clause that causes the INSERT, UPDATE, or DELETE statement to fail if the users specify a security label in the DML statement that does not belong to them. To override it so that IDS uses the *user's* security label rather than the security label explicitly specified by the user in the DML statement, use the OVERRIDE NOT AUTHORIZED WRITE LABEL SECURITY clause.

4. Create security labels.

Define the security labels for the security policy and include the security components that comprise this label. Here we define the security labels for president, sales manager for the Western region, and a sales engineer for the Northern California region:

```
CREATE SECURITY LABEL policy1.president
COMPONENT levels 'Board',
COMPONENT departments 'Marketing', 'Sales', 'Support',
'Development',
COMPONENT regions 'corp_wide';
```

```
CREATE SECURITY LABEL policy1.west_sales_mgr
COMPONENT levels 'Manager',
COMPONENT departments 'Sales',
COMPONENT regions 'West';
```

```
CREATE SECURITY LABEL policy1.northca_sales_engr
COMPONENT levels 'Engineer',
COMPONENT departments 'Sales',
COMPONENT regions 'Northern California';
```

5. Associate the table with the security policy.

Associate the sales tables with the security policy. To apply row level protection, ALTER the table to add a column *lbac_tag* of data type *idssecuritylabel* by using the policy *policy1*:

```
ALTER TABLE sales ADD (lbac_tag idssecuritylabel DEFAULT
'NorthCA_sales_engr'), ADD SECURITY POLICY policy1;
```

DEFAULT clause: The DEFAULT clause is mandatory only if the sales table is non-empty. You must ensure that the existing rows belong to this default label.

6. Grant user labels.

The sales table that has been associated with a security policy is now protected. No users are allowed access to it until they are assigned the security labels. Now GRANT the appropriate labels to the respective users. The *president* label is assigned to *usr1*, the Western sales manager label is assigned to *usr2*, North California Sales engineer is *usr3*, and the Eastern sales manager is *usr4*:

```
GRANT SECURITY LABEL policy1.president
TO USER 'usr1' FOR ALL ACCESS;
GRANT SECURITY LABEL policy1.west_sales_mgr
TO USER 'usr2' FOR ALL ACCESS;
GRANT SECURITY LABEL policy1.northca_sales_engr
TO USER 'usr3' FOR ALL ACCESS;
GRANT SECURITY LABEL policy1.east_sales_mgr
TO USER 'usr4' FOR ALL ACCESS;
```

7. Put row-level security to work.

During every INSERT into the sales table, either you must specify the appropriate label associated with the row or, if you omit the column from the list, IDS inserts the label belonging to the user performing the INSERT. IDS provides the built-in functions SECLABEL_BY_NAME and SECLABEL_BY_COMP to insert the labels. We demonstrate how this works as follows:

- a. INSERT a row into sales as usr3 with the label west_sales_mgr. Note that usr3 does not have permission for this label because it belongs to the northca_sales_engr label. Therefore, INSERT returns an error. If the security policy is created by using the OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL clause, the INSERT successfully inserts the label northca_sales_engr and ignores the explicitly specified west_sales_mgr label. Basically the INSERT downgrades the security to the security of the user running the INSERT:

```
runas usr3:
> insert into sales values (today, "usr3", "North CA", 2000,
> SECLABEL_BY_NAME('policy1', 'west_sales_mgr'));
8247: User does not have the LBAC credentials to perform INSERT
on table (informix.sales).
```

- b. INSERT a row by using the component names. usr2 has a west_sales_mgr label but has no permissions on the *Support* component. Therefore, the INSERT fails:

```
runas usr2:
> insert into sales values (today, "usr2", "North CA", 2000,
> SECLABEL_BY_COMP('policy1', 'Manager:(Sales,Support):West'));
8247: User does not have the LBAC credentials to perform INSERT
on table (informix.sales).
```

- c. Even usr1, who is the president, cannot insert at a level lower than the presidents security level, as shown here:

```
as usr1:
> insert into sales values (today, "usr1", "North CA", 2000,
SECLABEL_BY_COMP('policy1', 'Manager:(Sales,Support):West'));
8247: User does not have the LBAC credentials to perform INSERT
on table (informix.sales).
```

- d. INSERT a row successfully as usr3 with the correct security label and use the built-in function SECLABEL_TO_CHAR to select the label:

```
as usr3:
> insert into sales values (today, "usr3", "North CA", 2000,
> SECLABEL_BY_NAME('policy1', 'northca_sales_engr'));
1 row(s) inserted.
```

```
> select sales_person, seclabel_to_char('policy1',lbac_tag),  
> region from sales;
```

```
sales_person  usr3  
(expression) Engineer:Sales:Northern California  
region        North CA
```

1 row(s) retrieved.

- e. Let us assume that usr2, who is the Western region Sales manager, wants to insert rows. If the label column is omitted, IDS automatically inserts the appropriate label that belongs to usr2. usr2 can see the row inserted by usr3 in the previous step because usr2 is higher in the hierarchy in the *levels* ARRAY and is from the West region:

as usr2:

```
> insert into sales (sales_date, sales_person, region, sales)  
> values (today, "usr2", "West", 999);
```

1 row(s) inserted.

```
> select sales_person, seclabel_to_char('policy1',lbac_tag),  
> region from sales;
```

```
sales_person  usr3  
(expression) Engineer:Sales:Northern California  
region        North CA
```

```
sales_person  usr2  
(expression) Manager:Sales:West  
region        West
```

2 row(s) retrieved.

- f. usr3 is unable to see the usr2 row because that row is at a higher level:

as usr3:

```
> select sales_person, seclabel_to_char('policy1',lbac_tag),  
> region from sales;
```

```
sales_person  usr3  
(expression) Engineer:Sales:Northern California  
region        North CA
```

1 row(s) retrieved.

- g. If `usr4`, the Eastern sales manager, inserts a row, only the rows that belong to the East region can be seen and not the remaining two rows that were inserted earlier, which are for the West region:

```
as usr4:
```

```
> insert into sales (sales_date, sales_person, region, sales)
> values (today, "usr4", "East", 999);
1 row(s) inserted.
```

```
> select sales_person, seclabel_to_char('policy1',lbac_tag),
> region from sales;
```

```
sales_person  usr4
(expression)  Manager:Sales:East
region        East
```

```
1 row(s) retrieved.
```

- h. `usr1` is the president and can view all the rows:

```
as usr4:
```

```
> select sales_person, seclabel_to_char('policy1',lbac_tag),
> region from sales;
```

```
sales_person  usr3
(expression)  Engineer:Sales:Northern California
region        North CA
```

```
sales_person  usr2
(expression)  Manager:Sales:West
region        West
```

```
sales_person  usr4
(expression)  Manager:Sales:East
region        East
```

```
3 row(s) retrieved.
```

Implementing column-level LBAC

The HR department of the insurance company wants to allow employees, managers, and the HR staff to access data in the `EMPLOYEE` table. This table contains columns with different levels of sensitivity and has the following requirements:

- ▶ Name, gender, and department are considered to be *unclassified* information and can be available to all employees
- ▶ Employee number and salary are *confidential* and are restricted to managers and HR staff
- ▶ The Social Security numbers are *highly confidential* information and are restricted to the HR staff

The following statement creates the employee table:

```
CREATE TABLE employee (emp_name varchar(20), gender char(1), dept
varchar(10), emp_no integer, salary decimal, ssn char (9));
```

To implement column-level security on this table:

1. Create the security component.

We can consider an ARRAY for the simple hierarchy of the three levels of sensitivity:

```
CREATE SECURITY LABEL COMPONENT s1c_level
ARRAY ['HIGHLY CONFIDENTIAL', 'CONFIDENTIAL', 'UNCLASSIFIED']
```

2. Create the security policy.

Define the security policy by using the component defined in step 1:

```
CREATE SECURITY POLICY access_employee_policy
COMPONENTS s1c_level
WITH IDSLBACRULES
RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL
```

3. Create the security labels.

Define the security labels from the components that belong to the policy.

```
CREATE SECURITY LABEL access_employee_policy.highconfidential
COMPONENT s1c_level 'HIGHLY CONFIDENTIAL'
```

```
CREATE SECURITY LABEL access_employee_policy.confidential
COMPONENT s1c_level 'CONFIDENTIAL'
```

```
CREATE SECURITY LABEL access_employee_policy.unclassified
COMPONENT s1c_level 'UNCLASSIFIED'
```

4. Associate the table with the security policy:

```
ALTER TABLE EMPLOYEE
ALTER emp_name SECURED WITH UNCLASSIFIED
ALTER gender SECURED WITH UNCLASSIFIED
ALTER dept SECURED WITH UNCLASSIFIED
ALTER emp_no SECURED WITH CONFIDENTIAL
ALTER salary SECURED WITH CONFIDENTIAL
```

```
ALTER ssn SECURED WITH HIGHCONFIDENTIAL
ADD SECURITY POLICY access_employee_policy
```

5. Grant user labels:

```
GRANT SECURITY LABEL access_employee_policy.HIGHCONFIDENTIAL
TO USER usr1 FOR ALL ACCESS
```

```
GRANT SECURITY LABEL access_employee_policy.CONFIDENTIAL
TO USER usr2 FOR READ ACCESS
```

```
GRANT SECURITY LABEL access_employee_policy.UNCLASSIFIED
TO USER usr3 FOR READ ACCESS
```

6. Put column-level security to work.

Depending on the level of sensitivity, only the columns that are allowed are selected, as shown in the following sequence:

a. usr1 can successfully select all the columns of the table:

```
as usr1:
> select * from employee;
```

```
emp_name          gender dept          emp_no
salary ssn
```

No rows found.

b. usr2 cannot access the highly confidential column ssn. The SELECT succeeds if usr2 selects only the allowable columns:

```
as usr2:
> select * from employee;
8245: User cannot perform READ access to the protected column
(ssn).
```

```
> select emp_no,salary from employee;
```

```
emp_no          salary
```

No rows found.

c. usr3 can access only the columns that are marked as unclassified:

```
as usr3:
> select emp_name,gender,dept from employee;
```

```
emp_name          gender dept
```

No rows found.

```
> select emp_name,ssn from employee;
      8245: User cannot perform READ access to the protected column
(ssn).
```

You can also have tables that require a combination of row level and column level security. The steps to implement this are similar to the ones that we just explained.

4.5.5 Auditing

You can use the secure auditing feature to detect any unauthorized access. This feature allows you to audit database events that access or modify data. The database system security officer (DBSSO) can configure the system to audit certain user activities and periodically analyze the audit trail. The audit event has predefined mnemonics, such as CRTB for CREATE TABLE, that can be used to define the audit masks. The **onaudit** command is used to create, modify, and maintain the audit masks and configuration.

Auditing can also be used for diagnostic purposes. For example, consider an application that drops and recreates tables where the CREATE TABLE fails intermittently with an error indicating that the table already exists, even though it was DROPPED just before creation. This means that another session is trying to run a CREATE TABLE between this session's DROP and CREATE. You can turn on auditing and audit the CREATE TABLE and DROP TABLE commands run by usr1, by entering the following commands:

```
$ onaudit -l 1
$ onaudit -p /tmp/audit
$ onaudit -a -u usr1 -e +CRTB,DRTB
```

The file created in /tmp/audit shows an entry if usr1 creates a table t1 as shown in this example:

```
ONLN|2007-11-01 11:52:17.000|ramsay|17280|ramsay_install|usr1
|0:CRTB:san:101:t1:usr1:0:-
```

4.5.6 Data encryption

You can use column-level encryption to store sensitive data in an encrypted format. After encrypting sensitive data, such as credit card numbers, only users who can provide the correct password can decrypt the data. All values in a given column of a database table can be encrypted by using the same password. You can also choose to encrypt by using different passwords, or different encryption

algorithms, on different rows of the same table. This technique is sometimes necessary to protect personal data.

Users might enter unencrypted data into columns that are meant to contain encrypted data. To ensure that data entered into a field is always encrypted, use views and INSTEAD OF triggers that internally call the encryption function.

IDS supports built-in encryption and decryption functions. The encryption functions ENCRYPT_AES and ENCRYPT_TDES each return an encrypted_data value that encrypts the data argument. Conversely, the decryption functions DECRYPT_CHAR and DECRYPT_BINARY return a plain-text data value from the encrypted_data argument. You can invoke these encryption and decryption functions from within DML statements or with the EXECUTE FUNCTION statement.

For more details and examples, refer to Chapter 5 of the Redbooks publication *Informix Dynamic Server 11: Advanced Functionality for Modern Business*, SG24-7465.

Data encryption between HDR servers

To secure transmission of data between the database servers of a High Availability Data Replication (HDR) pair, you can encrypt data. After enabling encryption, the first database server in an HDR pair encrypts the data before sending the data to the other server in the pair. The server receiving the data, decrypts the data as soon as it is received. The configuration parameter ENCRYPT_HDR is used to enable or disable HDR encryption.

Other related configuration parameters, such as ENCRYPT_CIPHERS and ENCRYPT_MAC, must be set on both the servers. For more details, refer to the *IBM Informix Dynamic Server Administrator's Reference, Version 11.1*, G229-6360-01.

4.6 Backup and restore

The DBSAs are responsible for designing and implementing the database backup strategy. They must ensure that the backup performance is optimal, so that it has minimal impact on user activity. In case of a disaster recovery, DBSAs are responsible for quickly bringing the database back online to a consistent state by restoring it from backup. In the following sections, we discuss the various IDS backup and restore techniques along with their advantages and disadvantages. This information will help you in the design of the best backup plan suitable to your environment.

You can back up and restore internally by using **ontape** or **onbar**, or externally by using a backup/restore method outside of ON-Bar.

For internal backups, transactions can continue executing on the server when the backup is running. There is a small time interval when the server blocks transactions (updates) during the archive checkpoint. However, cold restore requires the server to be in offline mode when a critical dbspace, such as root dbspace or a dbspace that contains the physical or logical log, must be restored. An **onbar** restore can take hours depending on the database size. If you have a set of important tables that require immediate access, you can run a cold restore on the dbspaces housing that data to bring the server online.

A warm restore can then be performed to restore the remaining non-critical dbspaces when the server is in online mode. This mixed restore (cold restore followed by a warm restore) can take more time than an entire cold restore because it has to run a logical restore twice (once for the cold restore and once for the warm restore). But it serves the purpose of keeping the downtime to a minimum.

The external method handles the physical backup and restore, but the logical log backup and restore still has to be done by using ON-Bar. External backup and restore can be done by using **cp**, **dd**, or **tar** on UNIX or copy on Windows, or a file backup program. If you use hardware disk mirroring and external backup and restore, the DBSA blocks the server for a short time period when the mirrors are broken. During this time, reads are still allowed but updates block. The server is then unblocked to allow updates, and **onbar** is run to back up all the logs including the current one. The offline mirrored disks can then be backed up to back up media by using external commands, during which time the server is online and accepts new connections and updates. Mirroring can be resumed after the backup is complete. You can get your system online faster with external restore than with ON-Bar.

IDS supports internal backup and restore by using the **ontape** and **onbar** utilities:

- ▶ **ontape** offers the following advantages:
 - Can perform a sequential backup and sequential restore
 - Can back up to stdio, a file, or a directory
 - Can change the logging mode of a database but the recommended way is to use **ondblog**
- ▶ **ontape** requires the following considerations:
 - Does not use a storage manager
 - Cannot specify specific storage spaces
 - Cannot do a Point-in-Time (PIT) restore from backup
 - Cannot use multiple tapes concurrently
 - Cannot restart a restore

The ON-Bar suite consists of the ON-Bar API and the Informix Storage Manager. The ON-Bar API is the Informix implementation of the client component of the Open Systems Backup Services Data Movement (XBSA) API that is defined by the X/Open organization.

ON-Bar provides the following functionality:

- ▶ Parallel backup and parallel restores
- ▶ Use of Informix Storage Manager or other storage manager vendors to track backups
- ▶ Support for concurrent multiple tapes, PIT restore, and restartable restore
- ▶ Backup and restore selected storage spaces
- ▶ Use of the sysutils database

The backup tapes produced by **ontape** and ON-Bar are not compatible. For example, you cannot create a backup with **ontape** and then restore it with ON-Bar, or visa versa. In the following sections, we discuss the various backup and restore techniques and how they can fit your needs and your environment.

4.6.1 Levels of backup

IDS supports level-0, level-1, and level-2 backups. It writes all the disk pages to backup media for level-0 backup. For level-1 backup, the data that has changed since the level-0 is written to the media. Similarly for level 2, only the data that has changed since level 1 is written to media. If you request an incremental backup where no previous incremental backup exists, ON-Bar automatically performs the lower-level backup.

We recommend that you establish a backup schedule that keeps level-1 and level-2 backups small. Schedule frequent level-0 backups to avoid restoring large level-1 and level-2 backups or many logical-log backups. The frequency of the database backup depends on the frequency, amount, and importance of the data updates.

A level-0 backup is advised after administrative events. Such events include dropping or moving a logical log, changing the size or location of the physical log, adding a dbspace, blobspace or sbpace, adding logging to a database, or dropping a chunk. You might want to schedule these events to be run during the next scheduled level-0 backup. Although you no longer need to back up immediately after adding a logical log file, the next backup should be level-0 because the data structures have changed.

4.6.2 Ontape backup and restore

If there is a small- to medium-sized database in a test or development environment that does not necessarily need parallel backup/restore or Point in Time restore, you might want to choose **ontape** for your backup strategy. You do not need to configure any storage manager in this case.

The **ontape** utility writes the backup data directly to tape media. The configuration parameters TAPEDEV and LTAPEDEV point to the tape device. When backing up to tape, ensure that an operator is available and that there is sufficient media. A backup can require multiple tapes. After a tape fills, **ontape** rewinds the tape, displays the tape number for labeling, and prompts the operator to mount the next tape. The full tapes should then be labeled and new tapes mounted.

Backup to directory and stdout

To skip the additional manual intervention required for **ontape**, you can use **ontape** to write to stdout or to directories instead of tapes. In the current environment, where the cost of disk drives are falling, backing up to disk instead of tapes is a convenient option. When TAPEDEV and LTAPEDEV point to valid directories on either a local or mounted file system, **ontape** backs up data to those directories without the interactive prompts. It generates the file name automatically as <nodename>_<servername>_L<level>_num>. If a file already exists with this name, it renames this file with the current date and time stamp.

As shown in Example 4-14, the same directory is used by two instances on the same node *ramsay*, one with SERVERNUM 10 and the other with 14. Two level-0 backups have been run on the latter instance. During restore, **ontape** looks for the file *ramsay_14_L0* in this directory. If you must restore from an older image, it can be renamed to this standard format and used for the restore.

The standard prefix can be changed by setting an environment variable IFX_ONTAPE_FILE_PREFIX. If it is set to “dev101”, **ontape** generates file names as *dev101_L0*, *dev101_L1*, and so on. The **ontape** restore also looks for files with this prefix.

Example 4-14 ontape backup to directory

```
onconfig settings:
LTAPEDEV /home/data/backup
LTAPEDEV /home/data/backup
SERVERNUM 14

$ ontape -s -L 0
File created: /home/data/backup/ramsay_14_L0
```


Please label this tape as number 1 in the arc tape sequence.
This tape contains the following logical logs:

5

Program over.

```
$ ls -ltr /home/data/backup
-rw-rw---- 1 informix informix 15342850 Oct 1 11:02 ramsay_10_L0
-rw-rw---- 1 informix informix 11042816 Nov 1 13:52
ramsay_14_20071101_135254_L0
-rw-rw---- 1 informix informix 11239424 Nov 1 22:48 ramsay_14_L0
```

LTAPEDEV and TAPEDEV can point to an existing file instead of a directory, in which case successive backups overwrite this file. It can also point to STDIO to redirect the backup to stdout. Then the **ontape** restore reads the data from stdin. This is especially efficient for setting up HDR by restoring the data to the secondary server while skipping the intermediary step of saving the data to a file or disk. You can also use this option to restore this small-sized database to a remote machine. This is known as an *imported restore*. To override the LTAPEDEV and TAPEDEV settings, you can pass the **-t** option to **ontape** as shown in Example 4-15.

Example 4-15 ontape backup/restore using pipes

```
ontape -s -L 0 -t STDIO | rsh remote1 'ontape -p -t STDIO'
```

This example requires that the INFORMIXDIR, INFORMIXSERVER, INFORMIXSQLHOSTS, and ONCONFIG environment variables are set in the default environment for this user (informix or root) on the node remote1.

4.6.3 ON-Bar backup and restore

If your environment has a large number of instances and the investment in additional disk drives for hardware disk mirroring and external backup and restore is not feasible, ON-Bar can be used for the backup strategy. A storage manager must be installed to handle the media labeling, mount requests, and storage volumes. In this section, we discuss the various methods for improving ON-Bar usability and performance.

Improving parallelism

Each ON-Bar process backs up or restores different dbspaces. To increase the degree of parallelism, the `BAR_MAX_BACKUP` configuration parameter can be increased. This value specifies the maximum number of parallel processes that are allowed for each onbar command. When the number of running processes is reached, additional processes can start only when a currently running process completes its operation. If you set `BAR_MAX_BACKUP` to 0, the system creates as many ON-Bar processes as needed. The number of ON-Bar processes is limited only by the number of storage spaces or the amount of memory available to the database server, whichever is less.

IDS 11 makes intelligent decisions regarding the ordering of dbspaces during backup and restore to achieve maximum parallelism. For example, if the largest dbspace is backed up in parallel to other smaller dbspaces, the complete system backup takes less time. During a restore, dbspaces are restored in the same order in which they were backed up, which reduces restore time.

A set of all the important tables can reside in separate dbspaces. These dbspaces, along with the dbspaces that contain the root dbspace and the logical and physical logs, can be backed more often than the remaining dbspaces.

Customizing ALARMPROGRAM

The `ALARMPROGRAM` configuration parameter can be set to the Informix-provided alarm program script `$INFORMIXDIR/etc/alarmprogam.sh` to capture certain administrative events. These events can be informative, such as log complete, backup complete, long transaction detected, or an error condition such as chunk offline or *Out of memory*.

You can customize this script to perform the following tasks:

- ▶ Change the value of `ADMINMAIL` to the e-mail address of the DBSA.
- ▶ Change the value of `PAGERMAIL` to the pager service e-mail address.
- ▶ Set the value of the parameter `MAILUTILITY` with `/usr/bin/mail` for UNIX and `$INFORMIXDIR/bin/ntmail.exe` for Windows.
- ▶ To automatically back up logical logs as they fill, change `BACKUPLOGS` to Y.

Customizing the onbar script

The onbar script is located in the `$INFORMIXDIR/bin` directory and can be customized. The Informix Storage Manager commands in the default onbar script can be removed if you are not using Informix Storage Manager, and you can add the Storage Manager initialization or cleanup commands, if needed. These can include setting any storage manager specific environment variable settings.

Monitoring ON-Bar performance

To monitor the performance of ON-Bar and your storage manager, you can use the `BAR_PERFORMANCE` configuration parameter. You can specify the level of performance monitoring and have the statistics print to the ON-Bar activity log.

`BAR_PERFORMANCE` has the following valid values:

- 0 No statistics.
- 1 Report data transfer time between IDS and the storage manager.
- 2 The time stamps in the activity and error log in microsecond precision.
- 3 Reports both microsecond time stamps and transfer statistics. The value 3 is a combination of 1 and 2.

A sample ON-Bar activity log with a `BAR_PERFORMANCE` setting of 3 during backup is shown in Example 4-16. The time stamps in the log file have the microsecond part. The transfer rate table lists the size (XBSA API column *xfer-kbytes*) and the time (XBSA API column *xfer-time*) taken to transfer the data from the XBSA to the storage manager and the size (SERVER API column *xfer-kbytes*) and transfer time (SERVER API column *xfer-kbytes*) between IDS and XBSA. The API times are the transfer time plus the time spent executing the APIs. All these times are listed per dbspace.

The second table lists the time taken to spawn the daemon or update the sysutils catalog for backup or time taken to read the bootup file in case of a warm restore. If there are performance issues, compare these times to see if the bottleneck is due to the storage manager or IDS.

Example 4-16 Transfer rate performance in the onbar activity log

```
2007-11-07 11:04:55.057311 9008 9006 /work3/ssajip/INSTALL/xps/bin/onbar_d -b
2007-11-07 11:04:55.564799 9008 9006 Archive started on rootdbs, dbs1 (Requested
Level 0).
2007-11-07 11:04:55.791489 9008 9006 Begin level 0 backup rootdbs.
2007-11-07 11:04:55.830887 9008 9006 Successfully connected to Storage Manager.
2007-11-07 11:04:56.556617 9008 9006 Completed level 0 backup rootdbs (Storage
Manager copy ID: 18950 0).
2007-11-07 11:04:56.593297 9008 9006 Begin level 0 backup dbs1.
2007-11-07 11:04:56.849028 9008 9006 Completed level 0 backup dbs1 (Storage Manager
copy ID: 18951 0).
2007-11-07 11:04:57.216304 9008 9006 Archive on rootdbs, dbs1 Completed (Requested
Level 0).
2007-11-07 11:04:57.250568 9008 9006 Begin backup logical log 9.
2007-11-07 11:04:57.283503 9008 9006 Successfully connected to Storage Manager.
2007-11-07 11:04:57.663000 9008 9006 Completed backup logical log 9 (Storage
Manager copy ID: 18952 0).
```

2007-11-07 11:04:57.832761 9008 9006 PERFORMANCE INFORMATION

TRANSFER RATES:

OBJECT NAME	XBSA API				SERVER API				
	xfer-kbytes	xfer-time	RATIO(kb/s)	API-TIME	xfer-kbytes	xfer-time	RATIO(kb/s)	API-TIME	
9	64	0.147	436	0.261	64	0.013	4967	0.014	
dbsl	62	0.105	589	0.156	62	0.000	134183	0.291	
rootdbs	10230	0.362	28233	0.440	10292	0.114	90338	0.293	
PID =	9008	10356	0.614	16859	0.856	10418	0.127	81855	0.598

2007-11-07 11:04:57.845472 9008 9006 PERFORMANCE INFORMATION

PROCESS CLOCKS:

PID	CLOCK DESCRIPTION	TIME SPENT (s)
9008	To execute the master OnBar daemon.	2.796
9008	To update 'sysutils' for Log 9.	0.036
9008	To update 'sysutils' for Dbspace dbsl.	0.036
9008	To update 'sysutils' for Dbspace rootdbs.	0.032
9008	To update 'sysutils' for Log 9.	0.024
9008	To update 'sysutils' for Log 9.	0.021
9008	To update 'sysutils' for Log 9.	0.069
9008	To update 'sysutils' for Dbspace dbsl.	0.015
9008	To update 'sysutils' for Dbspace dbsl.	0.018
9008	To update 'sysutils' for Dbspace dbsl.	0.044
9008	To update 'sysutils' for Dbspace rootdbs.	0.022
9008	To update 'sysutils' for Dbspace rootdbs.	0.042
9008	To update 'sysutils' for Dbspace rootdbs.	0.112

2007-11-07 11:04:57.857574 9008 9006 /work3/ssajip/INSTALL/xps/bin/onbar_d complete, returning 0 (0x00)

Synchronizing onbar with the storage manager

The storage manager allows you to expire any old or aborted backups. The backup objects and the metadata information is deleted from the storage manager based on an expiration policy. For example, you can expire all backups prior to a particular date or retain only certain versions of a backup. IDS has to be informed of these expired backups, so that the storage manager and IDS are synchronized. The **onmsync** utility is used for these purposes.

ON-Bar maintains a history of backup and restore operations in the sysutils database and an extra copy of the backup history in the emergency boot file. It uses the sysutils database in a warm restore and uses the emergency boot file in a cold restore because the sysutils database cannot be accessed. The emergency boot file resides in the \$INFORMIXDIR/etc directory and is named *ixbar.servernum*, where *servernum* is the value of the SERVERNUM configuration parameter.

You can use the **onmsync** utility to regenerate the emergency boot file and expire old backups.

The **onmsync** utility synchronizes the sysutils database, the storage manager, and the emergency boot file as follows:

- ▶ The missing objects in the sysutils database are updated from the emergency boot file.
- ▶ It removes the records of restores, whole-system restores, fake backups, and successful and failed backups from the sysutils database.
- ▶ It expires old logical logs that are no longer needed.
- ▶ It regenerates the emergency boot file from the sysutils database.

For the syntax and options available with **onmsync**, refer to the *IBM Informix Backup and Restore Guide, Version 11.1, G229-6361-01*.

4.6.4 External backup and restore

There is administrative overhead with external backup because you have to track and label the database objects manually or use an external source such as a third-party storage manager. If your environment only has a few instances, external backup and restore with hardware mirroring is an efficient method as the benefits of backup and restore outweigh the administrative overhead. These instances can be large DSS or data warehouse environments. However, if you can invest in the additional disk drives, disk mirroring with the external method is an ideal option.

In an external backup, the backup and restore is performed outside of the database server, except for the logical log backup and restore, which is done by using the ON-Bar utility. ON-Bar treats an external backup as equivalent to a level-0 backup. You cannot perform an external backup and then use ON-Bar to perform a level-1 backup, or visa versa because ON-Bar does not have any record of the external backup. Because the external backup is outside the scope of control of the IDS instance, the granularity of a level 1 or level 2 does not exist.

Tracking backups: To make tracking backups easier, we recommend that you back up all storage spaces in each external backup.

To back up externally with enabled hardware disk mirroring:

1. Block the server by using **onmode** as follows, so that the database server is in a consistent state at a specific point in time:

```
$ onmode -c block
```

Blocking forces a checkpoint, flushes buffers to disk, and blocks transactions that involve updates or temporary tables. During the blocking operation, users can access the database server in read-only mode.

2. Break (disable) the mirror. The mirrored disks now contain a copy of the consistent data. If there is no disk mirroring, all the chunk files in each storage space can be backed up by using **cp**, **tar**, or any third-party tools.

Also back up the administrative files, such as ONCONFIG, and the emergency boot file.

3. Unblock the server by using **onmode** as follows so that users can continue with the updates:

```
$ onmode -c unblock
```

4. Back up the logical logs including the current one by using **onbar** or **ontape** as follows:

```
$ onbar -b -l -c
```

The current logical log has the archive checkpoint information that is needed during restore.

5. Back up the offline mirrored disks to back up media by using external commands.
6. Restart disk mirroring.

The database server and ON-Bar do not track external backups. To track the external backup data, use a third-party storage manager or track the data manually. Table 4-1 shows the items that we recommend to track for an external backup.

Table 4-1 Items to track for external backup and restore

Items to track	Description or example
Full path names of each chunk file for each backed up storage space	/work/dbspaces/rootdbs (UNIX)
Object type	Critical dbspaces and non-critical storage spaces
ins_copyid_hi and ins_copyid_lo	Copy ID that the storage manager assigns to each backup object
Backup date and time	The times that the database server was blocked and unblocked
Backup media	Tape volume number or disk path name
Database server version	The database server version from which the backup was taken

IDS supports a cold or warm external restore. Refer to the *IBM Informix Backup and Restore Guide, Version 11.1*, G229-6361-01, to do a cold or warm restore from an external backup.

4.6.5 Table level restore

ON-Bar can back up and restore database objects only at the dbspace level. If a table, a set of tables, or only portions of a table (using table filters) are to be restored, you can use the **archecker** utility. **archecker** is especially useful to restore a table that has accidentally been dropped.

To restore a table, you must create an **archecker** configuration file. The environment variable `AC_CONFIG` points to this configuration file. If this variable is not set, the file used by default is `$INFORMIXDIR/etc/ac_config.std`. This configuration file contains all the **archecker** specific configuration parameters. One of these parameters is `AC_CONFIG`, which is set to the **archecker** schema file. This schema file contains the schema of the source table (table that is being restored) and the destination table. This schema file also contains an `INSERT` statement that tells the **archecker** utility the tables to extract, the dbspace or dbspaces it needs to extract to, and any table filters.

To restore a table from an onbar backup:

1. Edit the **archecker** configuration file `$INFORMIXDIR/etc/ac_config.std` to include the following basic configuration parameters:

```
AC_STORAGE      /tmp
AC_MSGPATH      /tmp/ac_msg.log
AC_SCHEMA       /home/informix/tlr.sh
```

2. Create the **archecker** schema file `/home/informix/tlr.sh` as shown in Example 4-17. The round-robin fragmented *src* table is restored from archive as an expression fragmented table *dest*. The `INSERT` statement specifies the destination and source tables. It can restore a subset of columns by specifying them in the `SELECT` list or a subset of rows using the `WHERE` clause.

Example 4-17 Creating the archecker schema file

```
database db1;
create table src (col1 int, col2 char(10)) fragment by round robin
in rootdbs, dbs1;
```

```
create table dest (col1 int, col2 char(10)) fragment by expression
mod(col1,2) = 0 in rootdbs,
remainder in dbs1;
```

```
insert into dest select * from src;
```

3. Run **archecker** to restore the table `dest` as shown in Example 4-18. Option `b` is to use the onbar driver as opposed to **ontape** (option `t`), `X` is to extract the table from archive, `v` is for verbose output, and `s` prints the status to panel.

Example 4-18 Running archecker to restore the table dest

```
$ ui archecker -bXvs
IBM Informix Dynamic Server Version 11.10.FC1
Program Name:  archecker
Version:      8.0
Released:    2007-06-11 23:42:30
CSDK:       IBM Informix CSDK Version 2.91
ESQL:       IBM Informix-ESQL Version 2.91.FN219
Compiled:    06/11/07 23:43  on SunOS 5.9 Generic_112233-12

AC_STORAGE          /tmp
AC_MSGPATH          /tmp/ac_msg.log
AC_VERBOSE          on
AC_TAPEBLOCK        62 KB
AC_IXBAR            /home/informix/INSTALL/etc/ixbar.14
Dropping old log control tables
Extracting table db1:src into db1:dest

Scan PASSED
Control page checks PASSED
Table checks PASSED
Table extraction commands 1
Tables found on archive 1
LOADED: db1:dest produced 4 rows.
Creating log control tables
Staging Log 12

Logically recovered san:dest Inserted 0 Deleted 0 Updated 0
```

4.6.6 Backup filters

The `onconfig` parameters provide the option of specifying external programs, or filters, to transform data during backup and restore with both ON-Bar and `ontape`. You can use filters for compression or other data transformations. The backup filter reads the data to be backed up, transforms it, and then returns the transformed data to the backup utility. The restore filter receives the restored data from disk, transforms it back to its original state, and then passes the data to the restore utility. You specify the filters with two new **onconfig** parameters, `BACKUP_FILTER` and `RESTORE_FILTER`.

For example, if you want to compress the archive data, the backup filter receives the data, compresses it, and then backs up the compressed data. During the restore, the restore filter decompresses the data before it is restored to the database.

```
BACKUP_FILTER      /bin/compress
RESTORE_FILTER     /bin/uncompress
```

4.6.7 Restartable restore

If a failure occurs with the database server, media, or ON-Bar during a restore, you can restart the restore from the place that it failed. You do not have to restart the restore from the beginning. The `RESTARTABLE_RESTORE` configuration parameter controls whether ON-Bar is able to keep track of the storage spaces and logical logs that were restored successfully.

You can restart the following types of restores:

- ▶ Whole system
- ▶ Point in time
- ▶ Storage spaces
- ▶ Logical part of a cold restore

Refer to the *IBM Informix Backup and Restore Guide, Version 11.1*, G229-6361-01, for more information.

4.7 Optimistic concurrency

In this section, we discuss how to prevent applications from failing with SQL errors under specific conditions.

In Committed Read isolation level, exclusive row-level locks held by other sessions can cause SQL operations to fail with a lock error when attempting to read data in the locked rows. You can use the `LAST COMMITTED` keyword option to the `SET ISOLATION COMMITTED READ` statement to reduce the risk of such locking conflicts. In contexts where an application attempts to read a row on which another session holds an exclusive lock, these keywords instruct the database server to return the most recently committed version of the row, rather than wait for the lock to be released. The `LAST COMMITTED` keywords are only effective with concurrent read operations. They cannot prevent locking conflicts or errors that occur when concurrent sessions attempt to write to the same row.

Applications that prepare a statement (by using PREPARE) before running EXECUTE can sometimes fail with the following error message:

-710 error - Table <table-name> has been dropped, altered, or renamed.

This happens when the table or tables to which the statement refers in the PREPARE get renamed or altered, possibly changing the structure of the table or even an UPDATE STATISTICS on the table. By setting the configuration parameter AUTO_REPREPARE to 1, IDS automatically re-optimizes SPL routines and prepares prepared objects again after the schema of a table referenced by the SPL routine or by the prepared object has been changed. You can also set it at a session level by using SET ENVIRONMENT IFX_AUTO_REPREPARE.

For more information, refer to the *IBM Informix Dynamic Server Administrators Reference Guide, Version 11.1*, G229-6359-01.



The administration free zone

IBM Informix Dynamic Server (IDS) is well known for its powerful and flexible administrative capabilities. In IDS 11, administration gets even better with the addition of more features.

For example, IDS 11 now has SQL-based administration. That is, now most of the administrative tasks can be performed by using SQL. It also provides a framework where SQL statements or stored procedures can be defined to execute administrative tasks, collect statistical information, and perform database system monitoring without database administrator (DBA) intervention. IDS also has the capability to trace SQL statements for a particular user or a set of users. DBAs can retrieve the trace information in several ways and in formats that are more understandable to them.

IDS 11 provides a framework to automatically schedule and monitor database activities, to take corrective actions, and even to tune itself. Many of these features are enabled because the administration is SQL-based. Therefore, routines can be written to monitor, analyze, and change configuration parameters dynamically, based on the requirements of the IDS implementation. It is sometimes positioned as a “set it and forget it” environment because the system is doing much of the administration. That is also the genesis for referring to the environment as the “administration free zone”.

The administration free zone is a significant enhancement and provides significant benefits. As an example, the reduction in administration resources is a

significant contributor to the low total cost of ownership (TCO) of an IDS implementation.

The Web-based GUI administration tool called Open Administration Tool (OAT) is also available for IDS 11. This tool uses the new features in IDS 11 to provide a simple interface for performing the IDS administration tasks.

In this chapter, we provide a brief description of these features and explain how they make administration simple and automated in IDS 11. We also discuss example scenarios for using the new functionality. Finally we show you a real life example that uses the different components of the database administration system.

5.1 IDS administration

Database servers in general require numerous administrative tasks to be performed by the DBA to keep the database management system (DBMS) running at optimal levels. The following administrative tasks are some examples:

- ▶ Archiving the system at regular intervals to save the data
- ▶ Executing update statistics on the databases and tables to ensure good query performance
- ▶ Verifying database tables and indexes for data integrity
- ▶ Adding more space to the server as needed
- ▶ Backing up logical logs
- ▶ Monitoring the system for performance degradation

In previous IDS versions, some of these administration activities, such as adding dbspaces and checking database tables and indexes, could only be performed by using command line utilities such as **onspaces**, **onparams**, and **oncheck**. Hence to administer multiple instances, a command prompt was needed with the server environment for each of the instances to enable execution of these utilities. It was not possible to administer multiple instances with a single database connection or to administer an instance remotely. In addition, these IDS versions did not provide a way to schedule a task to occur at a specific time. To schedule an administrative activity, such as archiving the system weekly, or to update statistics when the load on the server was low, the DBA had to rely on operating system (OS) cron jobs or shell scripts, which can be platform dependent.

In IDS 11, you can use a set of user-defined routines (UDRs) to do most of the administrative tasks previously performed by command line utilities, such as **finderr**, **oninit**, **onmode**, **onspaces**, **onparams**, **ondblog**, **oncheck** (-c options only),

onlog, and **onstat**. These UDRs are called *SQL Administration APIs*. If you have a connection to a database server, you can administer other database servers on the same machines or other machines using SQL. Of course, you must have access permissions on those servers.

IDS 11 also has a component called the *DBScheduler*, which by using this component, you can schedule administrative activities within the database server. You can specify when to execute a specific administration task and whether to run it at specific intervals or at predefined times. For example, you can specify that the database server is to take the archive of the system at a certain time every week without manual intervention. You can also configure the database server to detect certain situations and take corrective actions as needed. For example, when the logical logs are full, the server blocks until the logical logs are backed up or new logs are available.

You can write scripts to enable the server to detect when the logical logs are full and add a logical log dynamically. You can also define SQL scripts and store them in the database server to gather statistical information and monitor database activity. For example, you can enable the server to monitor how much memory each session is using or how many users are logged onto the system at a certain time.

You can also generate reports for later analysis. For example, you can create reports for the total memory used by sessions or for the SQL statement statistics and purge these reports every day. The DBA does not need to rely on OS cron jobs or shell scripts for these routine tasks. Because the tasks are scheduled inside the server, they are portable across all platforms.

This framework provided by IDS 11 to schedule and monitor database activities, to take corrective actions, and even tune itself is referred to as the *administration free zone*. Some of the administrative activities are already predefined in the server. That is, they are already created for you and reside in the sysadmin database. We provide a list of those predefined administrative tasks in Table 5-1.

Table 5-1 Predefined administrative tasks in IDS 11

Table	Description
mon_command_history	Monitors how much data is kept in the command history table
mon_config	Collects information about the database server's configuration file (onconfig). Only collects parameters that have changed.
mon_config_startup	Collects information about the database servers configuration file (onconfig). Only collects parameters that have changed.

Table	Description
mon_sysenv	Tracks the database server's startup environment.
mon_profile	Collects the general profile information.
mon_vps	Processes the time of the virtual processors.
mon_checkpoint	Tracks the checkpoint information.
mon_memory_system	Tracks server memory consumption.
mon_table_profile	Collects SQL profile information by table or fragment. Index information is excluded.
mon_table_names	Collects table names from the system.
mon_users	Collects information about each user.
check_backup	Checks to ensure a backup has been taken.
ifx_ha_monitor_log_replay_task	Monitors the high availability (HA) secondary log replay position.
mon_user_count	Counts the number of users.

You can expand on this list by creating new tasks and scheduling them according to your needs. The SQL Administration APIs can help with this activity. For example, to schedule an administrative task, such as adding a chunk to a dbspace, you must rely on the SQL Administration API. But to schedule an administrative task, such as executing a checkpoint, you can use the **onmode** command in the scheduler. In the sections that follow, we explain how to perform administrative tasks by using SQL and show how this helps in scheduling tasks.

5.2 SQL-based administration

In IDS 11, most administrative tasks performed by **finderr**, **oninit**, **onmode**, **onspaces**, **onparams**, **ondblog**, **oncheck** (-c options only), **onlog**, and **onstat** utilities can be performed by executing SQL commands. Using SQL-based administration facilitates administering multiple instances with a single database connection. It also makes remote administration possible. In the following sections, we see how this is implemented and describe examples of SQL administrative commands.

5.2.1 The sysadmin database

A new database, `sysadmin`, is created by default in IDS 11. If upgrading to Version 11 from a previous version of IDS, the `sysadmin` database is created automatically as part of the conversion process. To verify that the database has been created successfully, check the online log.

The `sysadmin` database is a logged database that contains tables that are used by the `DBScheduler`. These tables contain tasks that are created by the Scheduler for collecting data and monitoring system activities.

The `sysadmin` database also contains the following items:

- ▶ The built-in `admin()` function
- ▶ The built-in `task()` function
- ▶ The `command_history` table that contains information about all the `admin()` and `task()` functions that are executed

Important: Never drop or alter the `sysadmin` database, because several database components use this database.

The `sysadmin` database is created in the root `dbspace` by default. If you have a number of tasks scheduled and heavily use the SQL Administration APIs, the root `dbspace` can fill up fast. If you usually use the root `dbspace` for your databases, then you can run out of space soon. In that case, you can move the `sysadmin` database to another `dbspace` to make more space in the root `dbspace`.

To move the `sysadmin` database:

1. Make sure that the following message is displayed in the online message log after server startup:

```
SCHAPI: Started 2 dbWorker threads.
```
2. If necessary, create a new `dbspace` for the `sysadmin` database. For example, you might call it `new_dbpace`.
3. As user `informix`, run the following commands:

```
dbaccess sysadmin -  
execute function task("reset sysadmin", "new_dbpace");
```

In this example, `new_dbpace` is the name of the `dbspace` that will store the `sysadmin` database. The command returns the following message:

```
SCHAPI: 'sysadmin' database will be moved to 'new_dbpace'.  
See online message log.
```

The internal thread, `bld_sysadmin`, waits up to five minutes to obtain exclusive access to the `sysadmin` database. The progress of the `bld_sysadmin` thread is logged in the online message log.

4. Terminate the `dbaccess` session with the `close database` statement. If this operation completes successfully, the `sysadmin` database is dropped and recreated in the new `dbospace`. The Scheduler and `dbWorker` threads are started automatically. The `sysadmin` database contains the tables shown in Table 5-2.

Table 5-2 *sysadmin database tables*

Table	Description
PH_ALERT	Contains a list of errors, warnings, or information messages that must be monitored.
PH_GROUP	Contains a list of group names. Each task is a member of a group.
PH_RUN	Contains information about how and when each task was executed.
PH_TASK	Lists tasks and contains information about how and when the database server will execute each task.
PH_THRESHOLD	Contains a list of thresholds that you defined. If a threshold is met, the task can decide to take a different action, such as inserting an alert in the PH_ALERT table.

Each row in the `ph_task` table is a task or a sensor that is executed by the DBScheduler as defined. The task properties determine what SQL or stored procedures to execute, when to execute them, and where the results should be stored.

The `ph_run` table contains an entry for every execution of a task or a sensor from the `ph_task` table. You can query this table to see if a task or a sensor has been executed successfully.

The `ph_alert` table also contains user-defined and system-defined alerts. If a task or a sensor created in the `ph_task` table failed to execute, there is a row in the `ph_alert` table indicating that the SQL specified in the task failed to execute and the error it returned. These are *system-defined alerts*. You can also insert rows in to the `ph_alert` table to create an alert for when a specific event occurs. These are *user-defined alerts*.

Note: Only user `informix` has permissions to access the `sysadmin` database by default.

5.2.2 SQL Administration APIs

SQL Administration APIs are a set of UDRs that can be executed against the sysadmin database. Most of the administrative tasks can be executed by using these UDRs, which are task() and admin(). They both perform exactly the same function, but are different in the return codes provided.

The routine task() returns a descriptive message indicating the success or failure of the command. The routine admin() returns an integer whose absolute value can be used to query the command_history table in the sysadmin database to obtain more information about the command that was executed. A positive integer indicates a success and a negative integer indicates a failure. The function task() can be used to execute a command by itself and see the return message. The function admin() can be used in SQL scripts or stored procedures.

Each execution of task() and admin() gets logged into the command_history table automatically. You can query this table to retrieve information about the user who executed the command, the time the command was executed, the command itself, and the message returned when the database server completed running the command.

Note: Only the database server administrator (DBSA), root user, and informix user have permission to execute the task() and admin() functions.

5.2.3 Examples of task() and admin() usage

When your available dbspaces are filling up, you must add more space for new tables and for updating existing tables. You might receive different error message indicating that the storage space is full, depending on the task that is being performed. For example, you receive SQL error -330 if you attempt to create a database in a dbspace and where there is not sufficient space. SQL error -502 is returned if a shortage of disk space is found when the database server was building a new copy of the table with rows in clustered sequence.

Adding more space is done by using the **onspaces** command, as in previous IDS versions. We now describe how to add a dbspace using the SQL Administration API. Example 5-1 shows the commands to create a 20 MB dbspace with 0 offset by using the task() UDR.

Example 5-1 Creating dbspace dbs2 by using task()

```
database sysadmin;  
execute function task("create dbspace", "dbs2",  
"$INFORMIXDIR/chunks/dbs2", "20MB", "0");
```

Example 5-2 shows the results of the **create dbspace** command.

Example 5-2 Results for executing the above task() example

```
(expression) Space 'dbs2' added.
```

The SQL in Example 5-2 creates the dbs2 file with correct permissions in \$INFORMIXDIR/chunks as part of creating the dbspace. If a file named dbs2 already exists, then this SQL command will fail.

The same results can be obtained by executing the function admin() as shown in Example 5-3.

Example 5-3 Creating dbspace dbs2 by using admin()

```
database sysadmin;  
execute function admin("create dbspace", "dbs2",  
"$INFORMIXDIR/chunks/dbs2", "20MB", "0");
```

The SQL in Example 5-3 creates the dbs2 file with correct permissions in \$INFORMIXDIR/chunks while creating the dbspace. The only difference to task() is in the return value. Example 5-4 shows the return value.

Example 5-4 Return value from admin() example

```
(expression)
```

```
179
```

To see detailed information about the admin() function executed in Example 5-3, query the command_history table as shown in Example 5-5.

Example 5-5 SQL to query the command_history table

```
database sysadmin;  
  
select * from command_history where cmd_number=179
```

The information contained in the command_history table is returned as shown in Example 5-6. You can see that the command executed successfully and space dbs2 was added.

Example 5-6 Results from the query of the command_history table

```
cmd_number      179  
cmd_exec_time   2007-10-08 12:38:37  
cmd_user        informix
```

```
cmd_hostname    yogi
cmd_executed    create dbspace
cmd_ret_status  0
cmd_ret_msg     Space 'dbs2' added.
```

1 row(s) retrieved.

Suppose that you need to update a table and the dbspace does not have enough room to fit the table after the update. You can add more space to the dbspace by adding a chunk to it. Example 5-7 on page 169 shows how you can add a 10 MB chunk to dbspace dbs1 by using the `admin()` function.

Example 5-7 Usage example of `admin()`

```
database sysadmin;

execute function admin("add chunk", "dbs1", "/work7/chunks/chk2", 0,
"10MB");
```

When adding a chunk, the file should exist in the specified directory with the correct permissions.

Validating database tables and indexes is performed periodically in all database environments to ensure data integrity. Usually such validation is done by executing the `oncheck` command. The SQL in Example 5-8 checks all tables and fragments in `dbspace1` by using the `task()` function.

Example 5-8 SQL API for `oncheck -cD`

```
database sysadmin;

SELECT task("check data",partnum)
FROM sysmaster:systables
WHERE trunc(partnum/1048575)=1 ;
```

In IDS 11, checkpoints are non-blocking, which means that transactions can make updates while a checkpoint is active. These transactions consume physical log and logical log space during their processing. If the server is short of critical resources, such as physical log and logical log space, then the transactions are blocked to let the checkpoint finish.

In an online transaction processing (OLTP) environment, it might be more important that the checkpoints are non-blocking to facilitate more server availability. Suppose that you have an OLTP environment and do not want transactions to block during checkpoint.

If the physical log is too small and the blocking of transactions happens during checkpoint, you see performance advisories in your online.log recommending you to increase the physical log size.

To increase the physical log size, use the **onparams** command or use the SQL Administration API as shown in Example 5-9.

Example 5-9 SQL API to change physical log size

```
database sysadmin;  
execute function task("alter plog","physdbs","30 MB");  
  
execute function task("checkpoint");
```

If the logical log is too small and blocking of transactions happens during checkpoint, you see performance advisories in your online.log recommending you to increase the logical log size. To increase the logical log size, add more logical logs by using the SQL Administration API shown in Example 5-10.

Example 5-10 SQL API to add logical logs

```
database sysadmin;  
execute function task("add log", "logdbs", "15MB", "3", "true");  
execute function task("checkpoint");
```

If needed, you can also drop a logical log file to increase the amount of space in a dbspace. The database server requires a minimum of three logical log files at all times. You cannot drop a log if your logical log is composed of only three log files.

The following rules apply for dropping log files:

- ▶ If a log file is dropped that has never been written to (status A), the database server deletes it and frees the space immediately.
- ▶ If a used log file (status U-B) is dropped, the database server marks it as deleted (D). After you take a level-0 backup of the dbspaces that contains the log files and the root dbspace, the database server deletes the log file and frees the space.
- ▶ A log file that is currently in use or contains the last checkpoint record (status C or L) cannot be dropped.

To drop all logical logs in rootdbs except the current one, use the SQL Administration API shown in Example 5-11.

Example 5-11 SQL command to drop the logical logs

```
database sysadmin;
select task("drop log", number)
      from sysmaster:syslogfil
      where chunk = 1 and sysmaster:bitval(flags,"0x02")==0;
execute function task("checkpoint");
```

Some of the command line commands can also be passed as arguments to the task() and admin() functions, as shown in Example 5-12. Here the **onmode -1** command is passed as an argument to the task() function. Then **onmode -1** is executed to switch to the next logical log. You might want to switch to the next logical log file before the current log file becomes full for the following reasons:

- ▶ Back up the current log
- ▶ Activate new blobspaces and blobspace chunks

Execute the command shown in Example 5-12 to switch to the next available log file.

Example 5-12 SQL API to switch to the next logical log

```
database sysadmin;
select task("onmode", "1") from sysmaster:syslogfil
      where chunk = 1 and sysmaster:bitval(flags,"0x02")>0;
execute function task("checkpoint");
```

Suppose you are packaging the database server in your application and the database server in each package should have the space configuration as shown in Table 5-3.

Table 5-3 Space configuration

Space Name	Size
dbspace1	40 MB
dbspace2	30 MB
sbspace1	50 MB
bospace1	50 MB
physdbs	40 MB
logdbs	40 MB

Space Name	Size
tempdbs	10 MB
\$INFORMIXDIR/chunks/chunk1 in dbspace1	10 MB
\$INFORMIXDIR/chunks/chunk2 in dbspace1	10 MB

The requirement might be to add the spaces after the server comes online. To do this, develop an SQL script that creates a table that contains the name, type, path, offset, and size of the dbspaces needed and another table that contains the information about the chunks. Example 5-13 shows the SQL Administration API functions to create the dbspaces and chunks.

Example 5-13 SQL script for system setup

```

database sysadmin;

create table dbspaces
(
    type          varchar(255),
    dbspace       varchar(255),
    path          varchar(255),
    offset        varchar(255),
    size          varchar(255)
);

insert into dbspaces values
("sbspace", "sbspace1", "$INFORMIXDIR/CHUNKS/sblob1", 0 , "50 MB" );
insert into dbspaces values
("dbspace", "dbspace1", "$INFORMIXDIR/CHUNKS/dbspace1", 0 , "40 MB" );
insert into dbspaces values
("dbspace", "dbspace2", "$INFORMIXDIR/CHUNKS/dbspace2", 0 , "30 MB" );
insert into dbspaces values
("dbspace", "physdbs", "$INFORMIXDIR/CHUNKS/physdbs", 0 , "40 MB" );
insert into dbspaces values
("dbspace", "logdbs", "$INFORMIXDIR/CHUNKS/logdbs", 0 , "50 MB" );
insert into dbspaces values
("tempdbspace", "tempdbs", "$INFORMIXDIR/CHUNKS/tempdbs", 0 , "10 MB"
);
insert into dbspaces values
("blobpace", "bpace1", "$INFORMIXDIR/CHUNKS/blobdbs", 0 , "50 MB" );

create table chunks
(
    dbspace       varchar(255),

```

```

        path          varchar(255),
        offset        varchar(255),
        size          varchar(255)
    );
insert into chunks values
("dbspace1", "$INFORMIXDIR/CHUNKS/chunk1",0 , "10 MB" );
insert into chunks values
("dbspace1", "$INFORMIXDIR/CHUNKS/chunk2",0 , "10 MB" );

SELECT task( "create "|| type , dbspace, path, size, offset)
        FROM dbspaces;

select task("add chunk", dbspace, path, size, offset) from chunks;
}

```

Now create a UNIX script, as shown in Example 5-14, to initialize the server and then execute the SQL script to create the specified dbspaces and chunks. The `-w` option that is used with `oninit` makes the shell script wait until the server is online. It saves you from writing additional code to check whether the server is online before executing `dbaccess`.

Example 5-14 Shell script to call `system_setup.sql`

```

#!/bin/sh
oninit -iwy;
if [ $? -gt 0 ]
then
    echo "Error starting IDS"
    onstat -m
    exit -1
fi
dbaccess sysadmin system_setup.sql;

```

You can also perform more administrative tasks by using the SQL Administration API functions. To see a full list of available SQL Administration APIs, refer to Chapter 7 of the Redbooks publication *Informix Dynamic Server 11: Advanced Functionality for Modern Business*, SG24-7465.

5.2.4 Remote administration

SQL can be executed across different databases and instances. Because the `sysadmin` database is a database, other instances with the proper connect privileges can connect to this database. The commands that are executed against the database are SQL. Therefore, remote administration can be

accomplished quite easily. Example 5-15 shows how to check the extents of a remote database server, called `remote1`, while you are logged onto your local server.

Example 5-15 Checking extents on a remote database server

```
database sysadmin@remote1;  
execute function admin("check extents");
```

5.3 Scheduling and monitoring tasks

In IDS 11, you can schedule an SQL stored procedure or UDR by using the DB Scheduler. This gives the DBA more flexibility in managing administrative tasks. The administrative tasks can be created in the `sysadmin` database and can be scheduled to run at predefined times or as determined internally by the server. Because these tasks are in the form of SQL commands, the same tasks can be executed on instances that are on different hardware platforms.

Only task properties, not configuration parameters, define what the Scheduler collects and executes.

5.3.1 Tasks

A *task* is a means to execute a specific job at a specific time or interval. A task is executed by invoking an SQL statement, a stored procedure, a C UDR, or a Java UDR. It is a way to monitor the database server and take corrective actions as needed.

All database servers need to do checkpoints at some frequency to ensure that there is a point in time when the data in memory and disk are consistent. It is from this consistency point that the server tries to restart in the event of an unexpected outage. In IDS 11, the checkpoint frequency is determined by configuration parameters, such as `CKPTINTVL`, `RTO_SERVER_RESTART` and `AUTO_CKPTS`. These parameters are set depending on your particular needs.

In IDS 11, checkpoints are non-blocking. That is, transactions can make changes to the data while the checkpoint is in progress. The server still blocks transactions from making any updates, if it is short of critical resources, such as physical log and logical log, to complete the checkpoint.

In an OLTP environment, it is beneficial to set `AUTO_CKPTS` to avoid blocking checkpoints. Setting `AUTO_CKPTS` to 1 triggers more frequent checkpoints as needed to ensure that the transactions are not blocked while the checkpoint is in

progress. In other environments, such as Decision Support Systems (DSS), transaction blocking might not be a critical factor. Therefore, CKPTINTVL can be used to perform checkpoints at fixed intervals.

If the availability of the database server is important, you can configure RTO_SERVER_RESTART to any value between 60 and 1800. This value specifies the time in seconds in which the server should be online in the event of an expected outage. For more information about configuring and tuning these parameters, refer Chapter 4, "Robust administration" on page 113.

Suppose that you have an OLTP environment and have configured AUTO_CKPTS and RTO_SERVER_RESTART, and the physical log and logical log are big enough not to cause too frequent checkpoints. But you want to ensure that you have a consistency point everyday. You can configure the database server to do it automatically by creating a simple task that does a checkpoint every day at 5 a.m. as shown in Example 5-16.

Example 5-16 Task to do a checkpoint every day at 5 a.m.

```
INSERT INTO ph_task(
tk_name,tk_type,tk_group,tk_description,tk_execute,tk_start_time,
tk_stop_time,tk_frequency,tk_next_execution)
VALUES(
"Checkpoints","TASK","SERVER",
"Do a checkpoint frequently",
"execute function task ('onmode','c');",
DATETIME(05:00:00) HOUR TO SECOND,
NULL,
INTERVAL ( 1 ) DAY TO DAY,
DATETIME(05:00:00) HOUR TO SECOND
);
```

If you create this task at a time later than 5:00 a.m., it is executed immediately on that day, but it executes at 5:00 a.m. on all subsequent days. You can also specify a date in the tk_next_execution field if you want the task to start executing from that day.

To check if your scheduled task was executed successfully, query the ph_run table by using the tk_id of the task in ph_task. In Example 5-17, tk_id is of the task in the ph_task table that you want to check.

Example 5-17 Query to ph_run table with tk_id

```
database sysadmin;
select * from ph_run where run_task_id = tk_id;
```

DBAs must update the statistics of a database or databases periodically. Updating the statistics is important because it affects the query performance. But this is an expensive operation because of the time it takes, which depends on the number of tables and indexes in the database and the size of them. We advise that you schedule it when there are relatively few users on the database in OLTP environments and Web environments where performance is crucial. Example 5-18 shows a task that executes the **update statistics** command on a database every Sunday at 5:00 a.m. The Scheduler executes this task automatically at the specified day and time.

Example 5-18 Task to run the update statistics on the mydb database

```
INSERT INTO ph_task(
tk_name,tk_type,tk_group,tk_description,tk_execute,tk_start_time,
tk_stop_time, tk_frequency, tk_next_execution)
VALUES(
"UpdateStat_mydb","TASK","SERVER",
"Do update stasticis on mydb every Sunday at 5am",
"database mydb; update statistics high;",
DATETIME(05:00:00) HOUR TO SECOND,
NULL,
"7 0:00:00",
"2007-11-04 05:00:00",
);
```

Improvements are expected in the next release that will provide even more flexibility and power with this feature.

5.3.2 Sensors

Sensors are specialized tasks that are more geared toward collecting data. Sensor data can be purged for status logging or later analysis. Sensors provide a portable and simple way of collecting information without using OS calls. However, because of their purging capability, they are useful in creating reports.

Example 5-19 on page 177 demonstrates the creation of a sensor to track the number of sessions on the database server every 5 minutes and to delete the data every day. This is one of the predefined tasks that is automatically created in the sysadmin database.

Example 5-19 Sensor to track the number of sessions

```
database sysadmin;
INSERT INTO ph_task
(tk_name,tk_type,tk_group,tk_description,tk_result_table,
tk_create,tk_execute,tk_start_time,tk_stop_time,tk_frequency,tk_delete)
VALUES
("mon_user_count",
"SENSOR",
"SERVER",
"Count the number of user count ",
"mon_user_count",
"create table mon_user_count (ID integer, session_count integer)",
"insert into mon_user_count select $DATA_SEQ_ID, count(*) from
sysmaster:syssessions",
NULL,NULL,
INTERVAL ( 5 ) MINUTE TO MINUTE,
INTERVAL ( 1 ) DAY TO DAY);
```

Tasks and sensors can be used for diagnostic purposes. Suppose there is a memory leak in your environment every day between 4 a.m. and 5 a.m. You must determine the sessions that are active during that time and the total memory used by these sessions to help determine the session that is causing the memory leak. Example 5-20 shows how to create a sensor that collects this information and stores it in a table, called `sess_mem`, every 60 seconds during the time frame in question.

Example 5-20 Sensor to track the total memory used by sessions

```
database sysadmin;
INSERT INTO ph_task (tk_name, tk_type,tk_group,tk_description,
tk_result_table,tk_create,tk_execute,tk_start_time,tk_stop_time,
tk_delete,tk_frequency,tk_next_execution)
VALUES
("Total Session Memory",
"SENSOR",
"SERVER",
"Total Session Memory.",
"sess_mem",
"create table sess_mem (sid integer, memtotal int, memused int,
updatedate datetime year to second ); ",
```

```
“insert into sess_mem select sid,memtotal,memused,current from
sysmaster:sysscb1st where dbinfo(‘UTC_TO_DATETIME’,connected)>current
- interval(1) hour to hour“,
“04:00:00”, “05:00:00”,NULL,
INTERVAL ( 1 ) minute TO minute, DATETIME(04:00:00) HOUR TO SECOND);
```

Important: The tk_create and tk_delete fields in the ph_task table are enabled only for *sensors*. Even if you specify a value for these fields when you create *tasks*, it will be ignored.

5.3.3 Startup tasks

A startup task is executed once when the database server starts. Startup tasks are executed 60 seconds after the server comes online by default. You can configure the startup tasks to run at a different time by specifying a value in the tk_frequency field.

An example of a start up task is to check the table spaces and extents of a database server. When you create a table, the database server allocates a fixed amount of space to contain the data to be stored in that table. When this space fills, the database server must allocate space for additional storage. The physical unit of storage that the database server uses to allocate both the initial and subsequent storage space is called an *extent*.

As rows are added to a table, the database server allocates disk space in units of extents, which is a block of physically contiguous pages from the dbspace. Even when the dbspace includes more than one chunk, each extent is allocated entirely within a single chunk, so that it remains contiguous.

Contiguity is important to performance. When the pages of data are contiguous, and when the database server reads the rows sequentially during read-ahead, light scans, or lightweight I/O operations, disk-arm motion is minimized.

Because table sizes are not known, the database server cannot preallocate table space. Therefore, the database server adds extents only as they are needed, but all the pages in any one extent are contiguous for better performance. In addition, when the database server creates a new extent that is adjacent to the previous one, it treats both as a single extent.

Monitoring disk usage by table is particularly important when you are using table fragmentation, and you want to ensure that table data and table index data are distributed appropriately over the fragments.

Example 5-21 shows how to check the extents of a database server automatically each time the server starts. The SQL Administration API is used to check the extents and is created as a start up task by inserting a row into the `ph_task` table.

Example 5-21 Startup task to check extents

```
database sysadmin;
INSERT INTO ph_task
( tk_name, tk_type, tk_group, tk_description, tk_execute,
tk_start_time, tk_stop_time, tk_frequency )
VALUES
("mon_disk_history",
"STARTUP TASK",
"TABLES",
"Monitor data via the oncheck -ce command",
"execute function task('check extents') ",
NULL,
NULL,
INTERVAL ( 1 ) DAY TO DAY);
```

You can also use startup tasks to turn on diagnostic flags at server startup time. Suppose that you need to set `AFDEBUB` 5 minutes after the server starts. `AFDEBUB` is set for hanging the engine in case of an assertion failure to enable you to collect diagnostic information. Example 5-22 shows how to create a startup task for this.

Example 5-22 Task to set AFDEBUB at system startup

```
database sysadmin;
INSERT INTO ph_task(
tk_name, tk_type, tk_group, tk_description, tk_execute,
tk_start_time, tk_stop_time, tk_frequency)
VALUES(
"AFDEBUB", "STARTUP TASK", "SERVER", "Set AFDEBUB to 1", "onmode -A 1",
NULL, NULL, "0 00:05:00");
```

5.3.4 Startup sensors

A startup sensor, like a startup task, is executed once when the database server starts. Startup sensors are executed 60 seconds after the server comes online by default. You can configure the sensor by specifying a time in the `tk_frequency` field.

Example 5-23 shows an example of a startup sensor. This is a predefined sensor in IDS 11 to track the database server startup environment.

Example 5-23 Startup sensor to track the database server startup environment

```
INSERT INTO ph_task (
tk_name,
tk_type,
tk_group,
tk_description,
tk_result_table,
tk_create,
tk_execute,
tk_stop_time,
tk_start_time,
tk_frequency,
tk_delete )
VALUES (
"mon_sysenv",
"STARTUP SENSOR",
"SERVER",
"Tracks the database servers startup environment.",
"mon_sysenv",
"create table mon_sysenv (ID integer, name varchar(250), value
lvarchar(1024))",
"insert into mon_sysenv select $DATA_SEQ_ID, env_name, env_value FROM
sysmaster:sysenv",
NULL,
NULL,
"0 0:01:00",
"60 0:00:00" );
```

5.4 Monitoring and analyzing SQL statements

The ability to monitor and analyze SQL statements executed on a database server is particularly useful when troubleshooting performance problems. It is important to identify which application or user is consuming most of the database resources, such as memory, disk I/O, CPU, and locks. By using this information, the DBA can analyze system performance, make changes in the system configuration, and make suggestions to developers for changing the application logic to enable improvement in the performance of the database system.

IDS 11 provides methods to identify performance bottlenecks. The SQL Query Drill-Down feature enables you to collect different levels of trace information about the SQL statements and displays the information in a format that is understandable by the DBA. The Query Drill-Down feature helps answer the following questions among others:

- ▶ How long do SQL statements take?
- ▶ How many resources are individual statements using?
- ▶ How long did statement execution take?
- ▶ How much time was involved waiting for each resource?

By default this feature is turned off, but you can turn it on for all users or for a specific set of users. When this feature is enabled with its default configuration, the database server tracks the last 1000 SQL statements that ran, along with the profile statistics for those statements.

Be aware that the memory required by this feature is quite large if you plan to keep historical information. The default amount of space required for SQL history tracing is 2 MB. You can expand or reduce the amount of storage according to your requirements or disable SQL history tracing.

The following information is an example of what the SQL trace output shows:

- ▶ The user ID of the user who ran the command
- ▶ The database session ID
- ▶ The name of the database
- ▶ The type of SQL statement
- ▶ The duration of the SQL statement execution
- ▶ The time this statement completed
- ▶ The text of the SQL statement or a function call list (also called stack trace) with the statement type
- ▶ Statistics including:
 - Number of buffer reads and writes
 - Number of page reads and writes
 - Number of sorts and disk sorts
 - Number of lock requests and waits
 - Number of logical log records
 - Number of index buffer reads
 - Estimated number of rows
 - Optimizer estimated cost
 - Number of rows returned
- ▶ Database isolation level

In the following sections, we explain how to use this feature to enable tracing for SQL statements and how to view and use the trace information with different methods.

5.4.1 Enabling and disabling tracing

There are two ways to enable and configure SQL tracing. One way is to use the SQLTRACE configuration variable in the \$INFORMIXDIR/etc/\$ONCONFIG file to specify tracing at server startup time. Another way is to use the SQL Administration API commands that were described in 5.2.2, “SQL Administration APIs” on page 167, when the server is online.

The SQLTRACE onconfig variable

Use the SQLTRACE configuration parameter to enable and control the default tracing behavior when the database server starts. The information that you set includes the number of SQL statements to trace, tracing mode, trace level, and trace size. This is useful when you want the tracing to be on at the same time that the database server comes online.

Any user who can modify the \$INFORMIXDIR/etc/\$ONCONFIG file can modify the value of the SQLTRACE configuration parameter and affect the startup configuration.

The following syntax can be specified with SQLTRACE:

```
SQLTRACE [Level=off|low|med|high],[Ntraces=number of traces],[Size=size of each trace buffer],[Mode=global|user]
```

SQLTRACE has the following range of parameters and values:

► Level

This parameter determines the amount of information traced:

- *Off* specifies no SQL tracing occurs and is the default.
- *Low* captures statement statistics, statement text, and statement iterators.
- *Medium* captures all of the information included in low-level tracing, plus table names, the database name, and stored procedure stacks.
- *High* indicates that all of the information included in medium-level tracing is captured, plus host variables.

► Ntraces

This parameter indicates the number of SQL statements to trace before reusing the resources. The range is from 500 to 2147483647.

► Size

This parameter specifies the number of KBs for the size of the trace buffer. If this buffer size is exceeded, the database server discards the saved data. The range is 1 KB to 100 KB.

► Mode

This parameter specifies the type of tracing that is performed:

- *Global* is for all users on the system.
- *User* is for users who have tracing enabled by an Administration API task() function. Specify this value if you want a sample of the SQL that a small set of users are running.

The SQL Administration API

The SQL Administration API functions provide more flexibility in configuring the SQL tracing parameters. If you do not want to set the SQLTRACE configuration parameter to turn on the tracing at server startup time or you decide to make changes to the initial settings, you can use the SQL Administration API functions.

The built-in SQL Administration API functions or task() and admin() from the sysadmin database provide the same functionality as the configuration variable SQLTRACE. However, setting or changing the tracing values by using the API functions does not require you to restart the server. With tracing enabled or disabled, using these API functions is effective only until the engine is restarted. After the engine is restarted, the SQLTRACE setting from the configuration file is used.

Do you know of a way to enable tracing when the database server starts without using the SQLTRACE onconfig variable?

Tip: Use the Scheduler to enable tracing when the database server starts.

5.4.2 Global and user modes of tracing

There are two modes of tracing: global and user. In *global mode*, the tracing is enabled for all sessions that are running on the system. In *user mode*, tracing is enabled only for the users that you have asked the system to trace.

Global tracing is useful in the following situations:

- ▶ When doing a quick comparative analysis on resource usage of all the sessions that are running
- ▶ When you are unsure of which specific user or users to trace and you must identify the sessions to trace

Setting adjustment: Adjust the trace settings to make the trace buffers big enough on systems where there are many users.

User-mode tracing is useful in the following situations:

- ▶ When you must narrow tracing to a specific set of users or just one user
- ▶ When you know which user or user sessions to trace

Because the tracing data is stored in memory buffers, setting the tracing only to required sessions gives you more control of the tracing memory usage.

5.4.3 Examples of enabling and disabling tracing

In this section, we look at some scenarios where we must enable and disable the different modes of SQL tracing.

Performance is more important in environments such as OLTP and Web, than in environments such as DSS. If you see a performance problem in your environment, use the SQL tracing feature to help narrow down the sessions or users who are causing it.

When you notice a performance degradation, but are not sure what sessions to trace, switch on the global mode of tracing. Monitor the sessions and SQL history trace information to help identify the sessions that do not need tracing. You can disable tracing at the session level by using the SQL Administration API functions.

Example 5-24 specifies the database server to gather low-level trace information for all sessions with default values. The server collects about 1000 KB of trace information for 1000 SQL statements.

Example 5-24 Switching on low-level global tracing

```
SQLTRACE Level=low,Mode=global
```

As shown in Example 5-25, you should see a line in the message log, when the server starts indicating that the tracing is switched on.

Example 5-25 Message in the log file

```
SQLTRACE: SQL History Tracing set to 1000 SQL statements.
```

To enable low-level global tracing with default values when the server is online, use the SQL Administration API shown in Example 5-26.

Example 5-26 SQL Administration API to switch on tracing with default values

```
dbaccess sysadmin -
```

```
Database selected.
```

```
> execute function task("set sql tracing on");
```

```
(expression) Global Tracing ON Number of Traces 1000 Trace Size 984  
Mode Low
```

```
1 row(s) retrieved.
```

Example 5-27 specifies that the database server is to gather medium-level trace information for all sessions. It specifies to trace 3000 SQL statements and to collect 4 KB of trace data for each SQL statement. Medium-level tracing gathers more information than low-level tracing.

Example 5-27 Switching on medium-level global tracing

```
SQLTRACE=medium,Ntraces=3000,Size=4,Mode=global
```

Example 5-28 shows how you can gather the same information by using the SQL Administration API function of task().

Example 5-28 Switching on sql trace by using task()

```
dbaccess sysadmin -
```

```
Database selected.
```

```
> execute function task("set sql tracing on", "3000", 4 , "med",  
"global");
```

```
(expression) Global Tracing ON Number of Traces 3000 Trace Size 4056
Mode Med
```

```
1 row(s) retrieved.
```

After monitoring the system for a while, it is possible to determine which sessions do not need tracing. Suppose that you decide that you want to disable tracing for a particular session, for example session 20. Use the SQL Administration API shown in Example 5-29 to disable tracing.

Example 5-29 Disabling tracing for session 20

```
database sysadmin -
```

```
Database selected.
```

```
> execute function task("set sql user tracing off", 20);
```

```
(expression) SQL user tracing off for sid(20)
```

After enabling global tracing, you might decide that the informix user sessions are acceptable. To disable tracing for all sessions of user informix, use the SQL shown in Example 5-30.

Example 5-30 Switching off tracing by using task()

```
database sysadmin;
select task("set sql user tracing off", session_id)
FROM sysmaster:sysessions
WHERE username not in ("informix");
```

When you know which user sessions must be traced, switch on the user mode of tracing. You can monitor the sessions by a specific user or set of users to help identify the sessions that are causing a performance bottleneck.

Example 5-31 shows how to enable user-mode tracing when the database server starts. The actual tracing does not start until an SQL Administration API is executed to indicate which user sessions to trace.

Example 5-31 Switching on low-level user tracing

```
SQLTRACE=high,Ntraces=2000,Size=2,Mode=User
```

To enable the user-mode tracing after the server comes online, use the SQL shown in Example 5-32.

Example 5-32 Enabling user-mode tracing by using task()

```
dbaccess sysadmin -
```

```
Database selected.
```

```
> execute function task("set sql tracing on",2000,2, "high", "user");
```

When user-mode tracing is enabled by the SQLTRACE onconfig parameter, use the task() or admin() function to start the tracing. For example, the SQL in Example 5-33 starts tracing for all user sessions of users *usr1* and *usr2*.

Example 5-33 Starting tracing for usr1 and usr2

```
select task("set sql user tracing on", session_id)
FROM sysmaster:syssessions
WHERE username in ("usr1", "usr2");
```

The tracing is switched on for all user sessions of *usr1* and *usr2* by using the tracing values specified in the SQLTRACE onconfig parameter.

To start tracing a particular session, when the user-mode tracing is enabled, use the SQL shown in Example 5-34.

Example 5-34 Starting tracing session 31

```
database sysadmin;
execute function task("set sql user tracing on", 31);
```

5.4.4 Displaying and analyzing trace information

In this section, we discuss methods of displaying and analyzing trace information.

The onstat command

The **onstat** option **-g his** is available to retrieve the SQL history tracing information. The output of the **onstat -g his** command contains trace information for a few SQL statements and prints the output as a single trace.

The **onstat -g his** command shows the host variable values for a statement if the tracing has captured them and those same values cannot be found in the

sysmaster tables. If you must see the host variable values for a statement, the only way you can see them is to use **onstat**.

Example 5-35 shows the output of the **onstat -g his** command. It has been truncated to show the trace information for one SQL statement.

Example 5-35 Output of the onstat -g his command

```
IBM Informix Dynamic Server Version 11.10.FC1      -- On-Line -- Up 00:07:20 -- 57344
Kbytes
```

Statement history:

```
Trace Level          Low
Trace Mode           Global
Number of traces     1000
Current Stmt ID     332
Trace Buffer size    984
Duration of buffer   413 Seconds
Trace Flags         0x00001611
Control Block       10bbb4028
```

Statement # 332: @ 10bbc2488

Database: 0x1000B2

Statement text:

```
SELECT FIRST 1 {+first_rows} tk_id, (tk_next_execution -
CURRENT)::INTERVAL SECOND(9) TO SECOND::char(20)::integer as tm_rem FROM
ph_task WHERE tk_id NOT IN ( ?,?,?,?,?,?,?,?,? ) AND
tk_next_execution IS NOT NULL AND tk_enable ORDER BY tk_next_execution,
tk_priority
```

Iterator/Explain

=====

ID	Left	Right	Est Cost	Est Rows	Num Rows	Type
2	0	0	8	1	16	Seq Scan
1	2	0	1	1	1	Sort

Statement information:

Sess_id	User_id	Stmt Type	Finish Time	Run Time
19	200	SELECT	16:14:06	0.0028

Statement Statistics:

Page	Buffer	Read	Buffer	Page	Buffer	Write
Read	Read	% Cache	IDX Read	Write	Write	% Cache
0	45	100.00	0	0	0	0.00

Lock Requests	Lock Waits	LK Wait Time (S)	Log Space	Num Sorts	Disk Sorts	Memory Sorts
0	0	0.0000	0.000 B	0	0	0
Total Executions	Total Time (S)	Avg Time (S)	Max Time (S)	Avg IO Wait	I/O Wait Time (S)	Avg Rows Per Sec
35	0.3202	0.0091	0.0088	0.000000	0.000000	363.1610
Estimated Cost	Estimated Rows	Actual Rows	SQL Error	ISAM Error	Isolation Level	SQL Memory
8	1	1	0	0	DR	27824

As you can see in Example 5-35, the `onstat -g his` output shows the following information:

- ▶ Trace settings information
 - This information includes the trace level, trace mode, size of the buffer, number of statements to trace, trace flags and control block. Some of these are dynamically configurable using the SQL Administration API functions.
 - From the output shown in Example 5-35, you can see that it is a low-level global trace for 1000 SQL statements. The buffer size is shown as 984 bytes, which is almost equal to 1 KB.
- ▶ Statement information
 - This information shows the statement text, the database name, iterator, and explain information for the statement. From this information, it is possible to determine the type of query, the user ID, session ID, the estimated number of rows, and the type of scans done on the tables.
 - The database name shows up correctly only for medium and high levels of tracing.
- ▶ Statement statistics
 - This information is most important because it is required for troubleshooting performance problems. It shows the time spent in page reads, buffer reads, lock waits, and I/O waits.
 - From the output in Example 5-35, you can see that there are zero page reads and 45 buffer reads. Therefore, the percentage read from the cache is 100%.

All of this information is stored in a buffer that is allocated in memory. Because the buffer size is configurable, use care when setting it. Setting the buffer size too

large takes up too much memory unnecessarily. Setting the buffer size too small truncates the trace information.

The sysmaster database

The SQL history trace information is stored in the following in-memory pseudo tables that are part of the sysmaster database.

- ▶ The *sysqltrace* table stores information related to statement statistics.
- ▶ The *sysqltrace_info* table contains information about the trace settings.
- ▶ The *sysqltrace_iter* table contains information about the iterators and the explain output.

You can query these tables to get trace information related to a specific SQL statement or SQL statements related to a specific user or session.

Suppose that you want to retrieve the SQL trace information for a particular session, for example 19, to see the SQL statements executed by that session and their statistics. Use the query in Example 5-36 to see the trace information for session 19.

Example 5-36 SQL trace information for a session

```
database sysmaster;  
select * from sysqltrace where sql_sid=19;
```

Example 5-37 shows the output for this query. The output has been truncated to show just one row of data. If many SQLs are executed in that session, the output is long, so that you can unload the output to a file and analyze it later.

Example 5-37 Output of the SQL trace information for a session

sql_id	669
sql_address	4491906104
sql_sid	19
sql_uid	200
sql_stmttype	2
sql_stmtname	SELECT
sql_finishtime	1192573934
sql_begintxtime	1192568820
sql_runtime	0.0025896
sql_pgregreads	0
sql_bfreads	45
sql_rdcache	100.0000000000
sql_bfidxreads	0
sql_pgwrites	0


```

sql_bfwrites      0
sql_wrcache      0.00
sql_lockreq      0
sql_lockwaits    0
sql_lockwtime    0.00
sql_logspace     0
sql_sorttotal    0
sql_sortdisk     0
sql_sortmem      0
sql_executions   168
sql_totaltime    1.508025200000
sql_avgtime      0.008976340476
sql_maxtime      0.0079508
sql_numioawaits  0
sql_avgioawaits  0.00
sql_totalioawaits 0.00
sql_rowspersec   386.1600247142
sql_estcost      9
sql_estrows      1
sql_actualrows   1
sql_sqlerror     0
sql_isamerror    0
sql_isollevel    1
sql_sqlmemory    27824
sql_numiterators 2
sql_database     <None>
sql_numtables    0
sql_tablelist    None
sql_statement     SELECT FIRST 1 {+first_rows} tk_id,
(tk_next_execution - C
                    URRENT)::INTERVAL SECOND(9) TO
SECOND::char(20)::integer as t
                    m_rem FROM ph_task WHERE tk_id NOT IN (
?, ?, ?, ?, ?, ?, ?, ?, ?, ?
                    ) AND tk_next_execution IS NOT NULL AND tk_enable
ORDER
                    BY tk_next_execution, tk_priority

```

You might want to compare the execution of the same statement at different times to see if there is a performance degradation. To retrieve the trace information of a particular statement, use the query in Example 5-38.

Example 5-38 SQL trace information for a particular statement

```
database sysmaster;
select * from syssqltrace a , syssqltrace_iter b where a.sql_id =
b.sql_id and a.sql_id=329;
```

You can unload the output to a file if this query has been executed a number of times.

To see a complete description of these tables, refer Chapter 8 of the Redbooks publication *Informix Dynamic Server 11: Advanced Functionality for Modern Business*, SG24-7465.

5.5 The Open Admin Tool for administration

A Web-based GUI application called the *Open Administration Tool* is available to administer the IDS. It is open source and can be downloaded from the IBM Informix Free Product Download page at the following address:

https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?lang=en_US&source=swg-informixfpd

OAT uses SQL-based administration in IDS 11 to do most of the administrative tasks. For the same reason, this tool works only with IDS 11. You can use one installation to administer multiple instances of IDS 11 that are local or remote to your machine. Because the OAT is developed by using PHP, it can be customized easily by adding new functions.

You can find instructions to install and configure the OAT in Chapter 2, “Optimizing IDS for your environment” on page 15. In this section, we focus on the different functionality of OAT.

After you log into your database server by using the login page, you can administer and manage it by using the OAT interface. Figure 5-1 shows the Menu and Map view, with the Menu on the left side.

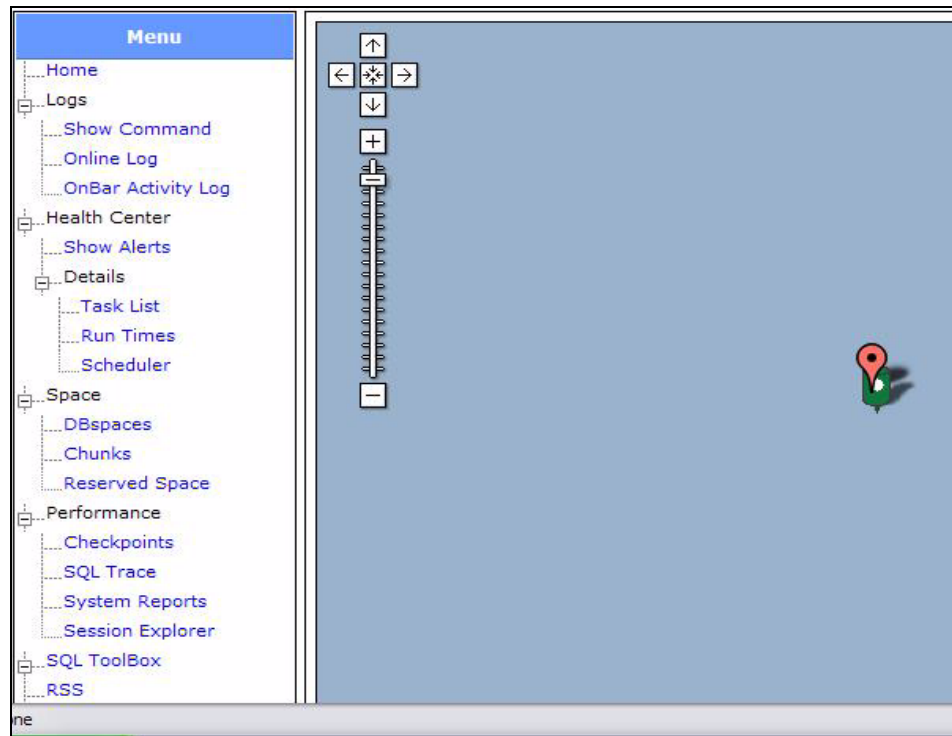


Figure 5-1 OAT Menu and Map view

Home is the first item in Menu. By clicking *Home*, you see the Map view on the right side of the page.

Logs is the next item in Menu and it contains three subitems:

- *Show Command* shows the administrative commands that were executed in the sysadmin database by using the task() or admin() function. The commands are displayed in the order of execution with the most recent ones appearing first as shown in Figure 5-2.

Admin Command History			
User	Time	Command Executed	Return Message
informix @ NA	2007-10-21 14:02:44	onmode	Checkpoint Completed
informix @ NA	2007-10-20 14:02:44	onmode	Checkpoint Completed
informix @ NA	2007-10-19 14:02:44	onmode	Checkpoint Completed
informix @ NA	2007-10-18 14:02:44	onmode	Checkpoint Completed
informix @ NA	2007-10-17 14:04:00	onmode	Checkpoint Completed
informix @ yogi	2007-10-10 16:34:44	set sql user tracing off	SQL user tracing off for sid(31).
informix @ yogi	2007-10-10 16:34:19	set sql user tracing off	ERROR could not find user.
informix @ yogi	2007-10-10 16:33:47	set sql usr tracing off	Unknown command (set sql usr tracing off).
informix @ yogi	2007-10-10 16:33:30	set sql tracing on	Global Tracing ON Number of Traces 1000 Trace Size 984 Mode Low
informix @ yogi	2007-10-10 16:33:18	set sql tracing off	SQL tracing off.
informix @ yogi	2007-10-10 15:30:55	set sql tracing on	Global Tracing ON Number of Traces 2000 Trace Size 2008 Mode High
informix @ yogi	2007-10-10 15:24:00	set sql tracing on	Global Tracing ON Number of Traces 3000 Trace Size 4056 Mode Med
informix @ yogi	2007-10-10 15:21:04	set sql tracing on	Global Tracing ON Number of Traces 2000 Trace Size 8152 Mode Med
informix @ yogi	2007-10-10 15:20:40	set sql tracing on	Global Tracing ON Number of Traces 1000 Trace Size 984 Mode Low
informix @ yogi	2007-10-10 15:20:23	set sql tracing on	Global Tracing ON Number of Traces 500 Trace Size 984 Mode Low

Figure 5-2 Show Command page

- *Online Log* displays the database server message log. The MSGPATH configuration parameter specifies the full path name of the database server message log. You should specify a path to an existing directory with an appropriate amount of space available. If you specify a file name only in the MSGPATH configuration parameter, the server creates the message log in the working directory in which you started the database server. For example, if you started the server from /usr/mydata on UNIX, the message log is written to that directory. This is the log that the server uses to write all messages, errors, and warnings.

When you view the Online Log in OAT, errors, warnings and informational messages are displayed in different colors so that you can easily notice them. Figure 5-3 shows how this log is displayed in OAT.

```

Fast poll /dev/poll enabled.
IBM Informix Dynamic Server Version 11.10.FC1N501 Software Serial Number AAA#B000000
Dynamically allocated new virtual shared memory segment (size 8192KB)
Memory sizes:resident:16384 KB, virtual:32768 KB, no SHMTOTAL limit
Direct I/O cannot be used for chunk file '/dev/dsk/c1t2d0s4'.
Dynamically allocated new virtual shared memory segment (size 8192KB)
Memory sizes:resident:16384 KB, virtual:40960 KB, no SHMTOTAL limit
Performance Advisory: The current size of the physical log buffer is smaller than recommended.
Results: Transaction performance might not be optimal.
Action: For better performance, increase the physical log buffer size to 128.
The current size of the logical log buffer is smaller than recommended.
IBM Informix Dynamic Server Initialized -- Complete Disk Initialized.
Warning: Invalid (non-existent/blobspace/disabled) dbspace listed
CETEMP: 'tmpdb1'
Checkpoint Request Non-Block Startup - rsinit.c 8853
Checkpoint started - Type Non-Block - Time to start 0.000
Wait for critical sections time 0.000
Leaving checkpoint wait
Checkpoint interval 2 opened - 4 buffers dirty
Checkpoint Completed: duration was 0 seconds.
Tue Oct 30 - loguniq 1, logpos 0xf8, timestamp: 0x52 Interval: 2

Maximum server connections 0
Checkpoint Statistics - Avg. Txn Block Time 0.000, # Txns blocked 0, Plog used 6, Llog used 1

Checkpoint interval 2 closed
Dataskip is now OFF for all dbspaces
TESTMODE: cdrStartUp have startup lock
TESTMODE: cdrStartUp: release startup lock
On-Line Mode
Building 'sysmaster' database ...
Btree scanners disabled.
Booting Language <spl> from module <>
Loading Module <SPLNULL>
Unloading Module <SPLNULL>

```

Figure 5-3 Online Log page

- *Onbar Activity Log* displays the ON-Bar activity log file. The BAR_ACT_LOG configuration parameter specifies the full path name of the ON-Bar activity log. You should specify a path to an existing directory with an appropriate amount of space available or use \$INFORMIXDIR/bar_act.log. Whenever a backup or restore activity or error occurs, ON-Bar writes a brief description to the activity log. The format of the file resembles the format of the database server message log.

You can examine the activity log to determine the results of ON-Bar actions. If you specify a file name only in the BAR_ACT_LOG parameter, ON-Bar creates the ON-Bar activity log in the working directory in which you started ON-Bar. For example, if you started ON-Bar from /usr/mydata on UNIX, the activity log is written to that directory.

The next item in the menu is the *Health Center*, which can be used to view the scheduled tasks and sensors in the DB Scheduler and their details, such as run times and return codes. You can also view the user-defined and system-defined alerts.

Health Center has the following subitems:

- *Show Alerts* shows the user-defined and system-defined alerts in the database server. This is basically the `ph_alert` table in the `sysadmin` database. Figure 5-4 shows the Alert List. The second alert in Figure 5-4 indicates that error 201 was returned when the server tried to execute the `UpdateStat_mydb` task. The third alert in Figure 5-4 indicates that `dbspace rootdbs` has not had a level 0 backup for 5 days, 4 hours, 51 minutes, and 7 seconds.

The alerts can be errors, warnings, or informational messages and can be in different states, such as new, addressed, or ignored. They can also appear in different colors depending on the `alert_color` field in the `ph_alert` table.

Alert List						
ID	Type	Message	Time	Recommendation	Alert State	
7481	WARNING	TASK NAME mon_user_count_t LOCATION ERROR – 206 The specified table (mon_user_count_t is not in the database. ERROR – 111 ISAM error: no record found.	2007-10-30 10:39:06		NEW	Re-Check Ignore
7472	WARNING	TASK NAME UpdateStat_mydb LOCATION ERROR – 201 a A syntax error has occurred.	2007-10-30 09:57:16		NEW	Re-Check Ignore
7272	WARNING	DbSPACE [rootdbs] has not had a level 0 backup for 5 04:51:07. Recommend taking a level 0 backup immediately.	2007-10-29 17:41:02		NEW	Re-Check Ignore
7275	WARNING	DbSPACE [dbs2] has not had a backup for 13816 01:41:02. Recommend taking a backup immediately.	2007-10-29 17:41:02		NEW	Re-Check Ignore
7273	WARNING	DbSPACE [DBS1] has not had a level 0 backup for 5 04:51:07. Recommend taking a level 0 backup immediately.	2007-10-29 17:41:02		NEW	Re-Check Ignore
7276	WARNING	DbSPACE [DBS3] has not had a level 0 backup for 5 04:51:07. Recommend taking a level 0 backup immediately.	2007-10-29 17:41:02		NEW	Re-Check Ignore
7274	WARNING	DbSPACE [DBS2] has never had a level 0 backup.	2007-10-29		NEW	Re-Check

Figure 5-4 Show Alerts page in Health Center

For more information about the alert colors, refer to Table 5-4 on page 197. You can selectively view the alerts depending on their Severity (color), Error Type, or State.

Table 5-4 Alert color definitions

Type	Green	Yellow	Red
Informative	Indicates a component operation status	Indicates a component operation status	Requires action.
Warning	From the database that was automatically addressed	A future event that needs to be addressed	A predicted failure is imminent. Action is necessary now.
Error	A failure in a component corrected itself	A failure in a component corrected itself but might need DBA action	A failure in a component requires DBA action.

- *Task List* shows the different tasks and sensors that are created in the Scheduler. This page shows the name, group, description, time of next execution, and frequency of execution of each task in the `ph_task` table of the `sysadmin` database. You can view this page to monitor the tasks that are created in the database and to know the names of the existing tasks when you are ready to create a new one.

On the Task List page shown in Figure 5-5 on page 197, you can see such predefined tasks as `mon_config_startup` and `mon_sysenv` as discussed in 5.1, “IDS administration” on page 162. You can view all the tasks or selectively view the tasks by using the Group to View drop-down list, which is also shown in Figure 5-5.

Name	Group	Description	Next Execution	Frequency
mon_config_startup	server	Collect information from the configuration file (on configuration changes).		
mon_sysenv	server	Tracks the current system environment.		0 00:01:0
ifx_ha_monitor_log_replay_task	server	Monitors HA secondary log replay position		
mon_user_count	server	Count the number of user count	2007-10-30 10:57	0 00:05:0
mon_user_count_t	server	Count the number of user count	2007-10-30 10:59	0 00:05:0
mon_user_count1	server	Count the number of user count	2007-10-30 10:59	0 00:05:0
Idle Timeout	user	Remove all idle users from the system.	2007-10-30 11:00	0 00:10:0
mon_checkpoint	server	Track the checkpoint information	2007-10-30 11:41	0 01:00:0
mon_memory_system	memory	Server memory consumption	2007-10-30 11:41	0 02:00:0
mon_profile	performance	Collect the general profile information	2007-10-30 13:41	0 04:00:0

Figure 5-5 Task List page

- ▶ *Run Times* shows the tasks that are executed in the system and their name, the number of times each task was executed, and the total time and average time of execution. This information is taken from the `ph_run` table in the `sysadmin` database.

You might want to look here to see when and if your scheduled tasks were executed. For example, the first line in Figure 5-6 shows that the Checkpoints task was executed 13 times, the average time of execution of this task is 4.12 seconds, and the total time of execution is 53.61 seconds. You can view all of them or selectively view them by using the Group to View drop-down list. Figure 5-6 shows all the executed tasks.

Name	Count	Average Time	Total Time
Checkpoints	13	4.12	53.61
Idle Timeout	288	0.07	21.94
UpdateStat_mydb	16	0.00	0.04
UpdateStat_mydb_1	3	0.00	0.01
UpdateStat_test1	16	14.23	227.76
check_backup	38	3.82	145.37
ifx_ha_monitor_log_replay_task	1	0.03	0.03
mon_checkpoint	855	0.33	285.80
mon_command_history	38	0.05	2.25
mon_config	38	0.54	20.87

Figure 5-6 *Run Times* page

- ▶ *Scheduler* shows part of the information from the `ph_task` table about scheduled tasks, such as start time, stop time, and frequency.

OAT also shows the *Space* details of the database server, which includes the following subitems:

- ▶ *DBspaces* shows the details of all the dbspaces, temp spaces, blobspaces, and sbspaces that are created. It also provides an interface to add these spaces to the server.

In Example 5-1 on page 167, we show how a dbspace named *dbs2* can be added by using the SQL Administration API task() function. Figure 5-7 shows how you can do the same by using the OAT interface.

The screenshot shows the 'DBSpaces' page in the OAT interface. At the top, it indicates the connection details: 'Connected: informix@ids_1110' and 'Host: yogi.menlo.ibm.com'. Below this is a table listing existing dbspaces. The table has columns for Number, Name, Type, Size, Free, Used %, # of Chunks, and Page Size. There are three rows of data. Below the table is a 'Create a Space' form with fields for Name, Path, Offset, Size, and Type, along with a 'Create' button.

DBSpaces							
Number	Name	Type	Size	Free	Used %	# of Chunks	Page Size
1	rootdbs	DBSpace	146.48 MB	23.74 MB	83.79%	1	2 KB
2	dbs1	DBSpace	98.63 MB	48.52 MB	50.80%	2	2 KB
4	dbs3	DBSpace	10 MB	9.95 MB	0.54%	1	2 KB

Below the table, the 'Create a Space' form is visible with the following fields:

- Name:
- Path:
- Offset:
- Size:
- Type:
- Button:

Figure 5-7 *DBspaces* page

By clicking the Name column, you see detailed information about each dbspace.

Example 5-7 on page 169 explains how you can add a chunk to dbSPACE dbS1 by using the admin() function. To do the same task by using the OAT interface, click **dbS1** and select the **Admin** tab. Now you have the interface to add a chunk to dbS1. Refer Figure 5-8 to see the OAT interface for adding a chunk.

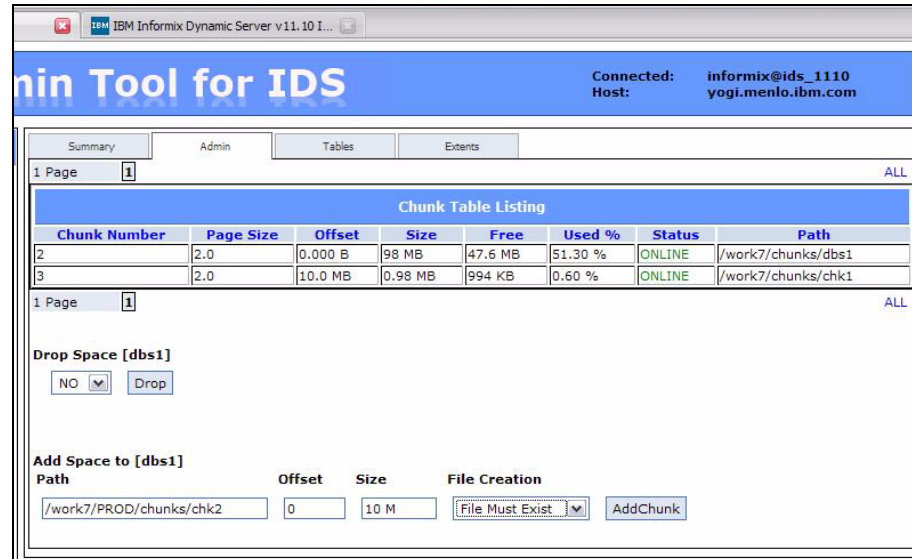


Figure 5-8 Adding a chunk

- *Chunk* shows details about all the chunks in each dbSPACE, such as size, page size, offset, free space, and used space. It also shows the reads and writes in each of the chunks.

- ▶ For *Reserved Space*, the first two tabs at the top of the page can be used to view the logical log details and the physical log details. The next tab, Admin, provides a GUI to perform a checkpoint to add and drop logical logs, and to move the physical log.

Example 5-9 on page 170, Example 5-10 on page 170, and Figure 5-11 on page 171 show SQL Administration API functions that can be used to move a physical log and to add and drop logical logs. Figure 5-9 shows the OAT interface to do these administrative tasks on the system Reserved Space.

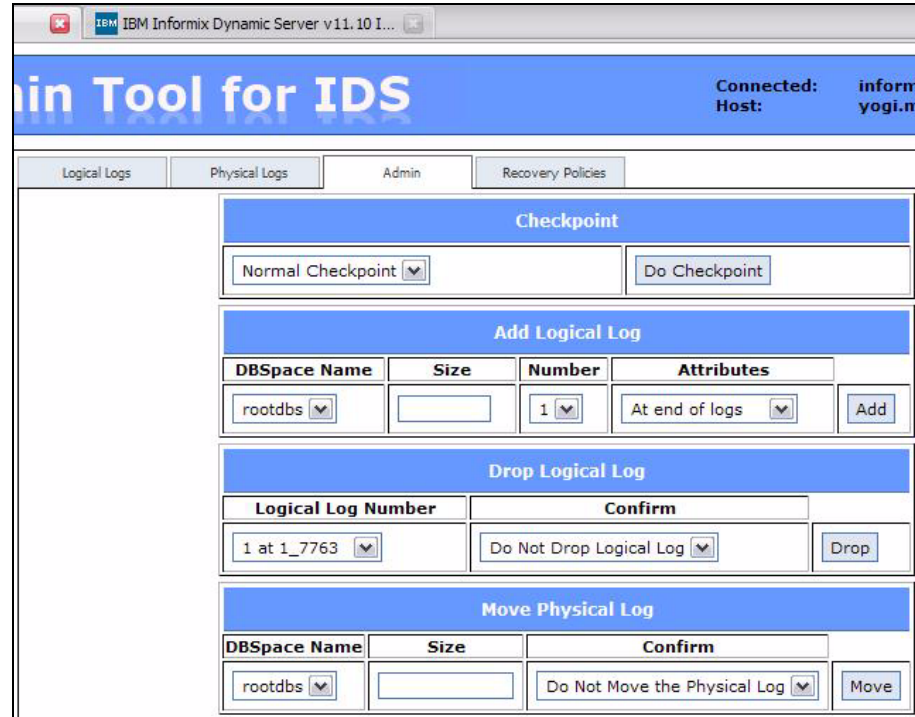


Figure 5-9 Admin tab on the Reserved Space page

The *Performance* menu item has the following subitems:

- ▶ *Checkpoints* shows detailed information about checkpoints, such as the checkpoint interval, the type, Log Sequence Number (LSN) at the time of checkpoint, what triggered the checkpoint, the flush time, and the block time. This is basically the output from `onstat -g ckp` command. The same information can also be obtained from `syscheckpoint`. This information can be analyzed by the DBAs to use in checkpoint tuning. For example, if a small physical log caused a blocking checkpoint, you can see from this output that transactions are blocked because of a small physical log and then decide to increase the physical log size.

- *SQL Trace* shows an SQL Statement Summary report (Figure 5-10) that contains a summary of the different types of SQL statements that are executed on the system. We discussed how to enable and disable tracing for SQL statements and how to view the trace information in 5.4, “Monitoring and analyzing SQL statements” on page 180. The OAT provides an interface to view detailed trace information for SQL statements.

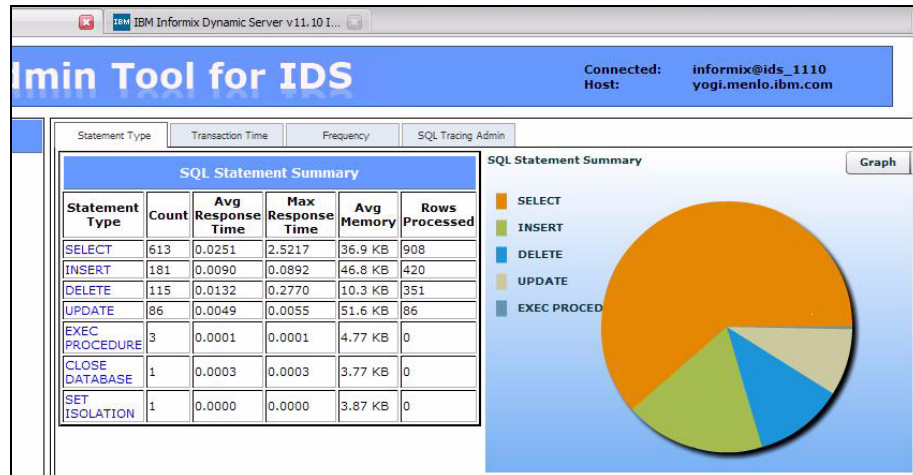


Figure 5-10 SQL Statement Summary page from SQL Trace

Usually the queries executed on a system are monitored to see if they take longer than expected. If they do, they are analyzed further to see if they are performing sequential scans, and then decisions are made regarding whether to create indexes on the tables involved. By using the OAT SQL Trace interface, you can monitor a particular type of query or all queries.

For example, to monitor the SELECT statements that are taking the longest time, you must determine if more indexes are needed on the table that is involved. From the SQL Statement Summary page, you can click the SELECT statement and drill down further to see a list of those types of statements executed on the system, along with details such as response time, I/O wait time, and lock wait time. By clicking the Drill Down button, you can see more information and statistics about each statement. You can sort the SELECT queries on the Average Run Time, as shown in Figure 5-11, to see a list of SELECT statements with the slowest ones on top.

SQL Frequency Summary						
SQL Drill Down	Count	Average Run Time	Lock Wait Time	Wait IO Time	Completion Time	SQL Statement
Drill Down	1	0.9605	0.0000	0.0000	11:33:57	SELECT count(*) FROM syssqltrace A, syssqltrace B WHERE A.sql_statement = B.sql_statement AND A.sql_id = 112808
Drill Down	1	0.9061	0.0000	0.0000	13:06:07	SELECT count(*) FROM syssqltrace A, syssqltrace B WHERE A.sql_statement = B.sql_statement AND A.sql_id = 113424
Drill Down	1	0.8643	0.0000	0.0152	12:59:43	select FIRST 10 '<form method="get" action="index.php"> <input type=submit class=button name="view" value="Drill Down"> <input type=hidden name="act" value="sqltrace"> <input type=hidden name="do" value="sqlist"> <input type=hidden name="id" value="' MAX(sql_id) "'/> </form"> as url, count(*) as cnt, TRUNC(sum(sql_runtime)/count(*),4) as avgrun, TRUNC(sum(sql_lockwttime),4) as lkwttime, TRUNC(sum(sql_totaliowaits),4) as waitio, sql_statement[1,1000] as sql_statement FROM syssqltrace WHERE sql_stmtype >0 GROUP BY sql_statement[1,1000] ORDER BY 3
Drill Down	1	0.8617	0.0000	0.0160	12:59:32	select FIRST 10 '<form method="get" action="index.php"> <input type=submit class=button name="view" value="Drill Down"> <input type=hidden name="act" value="sqltrace"> <input type=hidden name="do" value="sqlist"> <input type=hidden name="id" value="' MAX(sql_id) "'/> </form"> as url, count(*) as cnt, TRUNC(sum(sql_runtime)/count(*),4) as avgrun, TRUNC(sum(sql_lockwttime),4) as lkwttime, TRUNC(sum(sql_totaliowaits),4) as waitio, sql_statement[1,1000] as sql_statement FROM syssqltrace WHERE sql_stmtype >0 GROUP BY sql_statement[1,1000] ORDER BY 2 desc, 3 desc

Figure 5-11 SQL Drill Down

If you follow the Drill Down button for that statement, the final page is shown as in Figure 5-12. You can see details such as the tables involved, the rows processed, and the types of scans to help you to determine whether more indexes must be created on the tables involved.

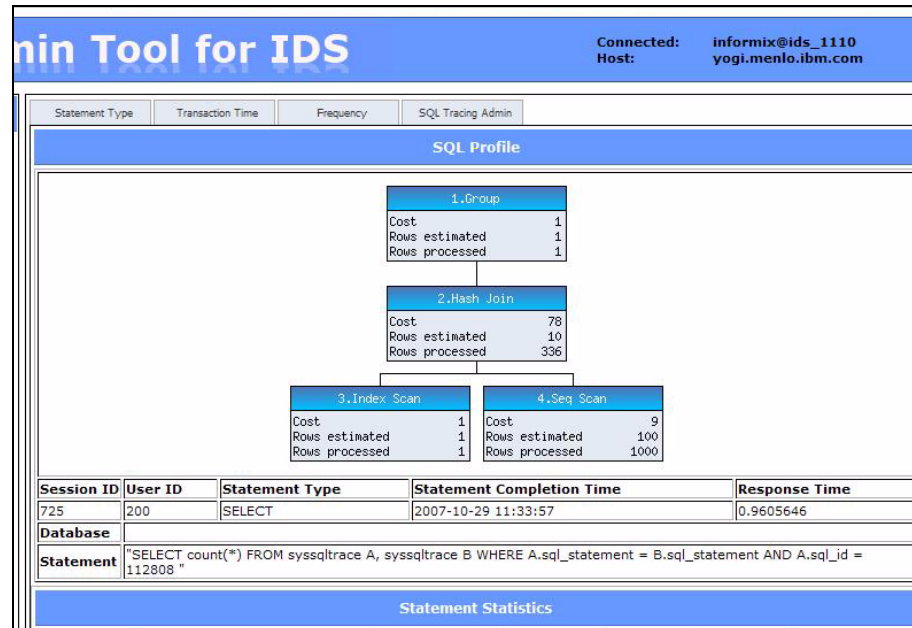


Figure 5-12 Query Drill Down

The SQL Tracing Admin tab (Figure 5-13) is the interface to enable and disable global and user mode of SQL tracing. It also shows the current tracing information.

The screenshot shows the 'SQL Tracing Admin' interface. At the top, it displays 'min Tool for IDS' and connection details: 'Connected: informix@ids_1110' and 'Host: yogi.menlo.ibm.com'. Below this, there are four tabs: 'Statement Type', 'Transaction Time', 'Frequency', and 'SQL Tracing Admin'. The 'SQL Tracing Admin' tab is active, showing a table of 'SQL Trace Info' and a 'Set SQL Tracing Options' form.

SQL Trace Info	
Number of SQL Statements Traced	1000
Traces Buffer Size	984.000 B
Oldest Traced Statement	2007-10-24 12:44:51.000
Trace Buffer Duration	02:09:34
SQL Statements Seen	68736
SQL Tracing Started	2007-10-16 14:06:38
SQL Statements Per Secon	8.84178
SQL Tracing Memory Used	1.15 MB

Set SQL Tracing Options			
Mode	Number of Traces	Trace Size	Trace Level
On	1000	984	Low

Figure 5-13 SQL Tracing Admin page from SQL Trace

- ▶ *System Reports* helps you take reports on the following items:
 - Disk space usage
 - Online log
 - Logical logs
 - System backups
 - Server memory usage
 - Table actions
 - Session list
 - SQL Statement Summary
 - SQL with most I/O time
 - Virtual processors
 - Disk I/O levels
 - Server admin commands
 - Physical logs
 - Memory pools
 - Network overview
 - Server configuration
 - View SQL caches
 - Slowest five SQL statements
 - SQL with most buffer activity
 - Databases

Some of these reports, such as SQL with most I/O time and the slowest five SQL statements, are useful in troubleshooting performance problems. Meanwhile, others can be used to evaluate the disk space needed and general system monitoring.

- ▶ *Session Explorer* shows a list of active sessions and their details.

SQL Tool Box is a powerful menu in OAT and contains the following subitems:

- ▶ *Databases* shows a list of databases (Figure 5-14). When each database is clicked, the tables and SPLs created in them are displayed. The tables can be further examined to see the column details and the data.

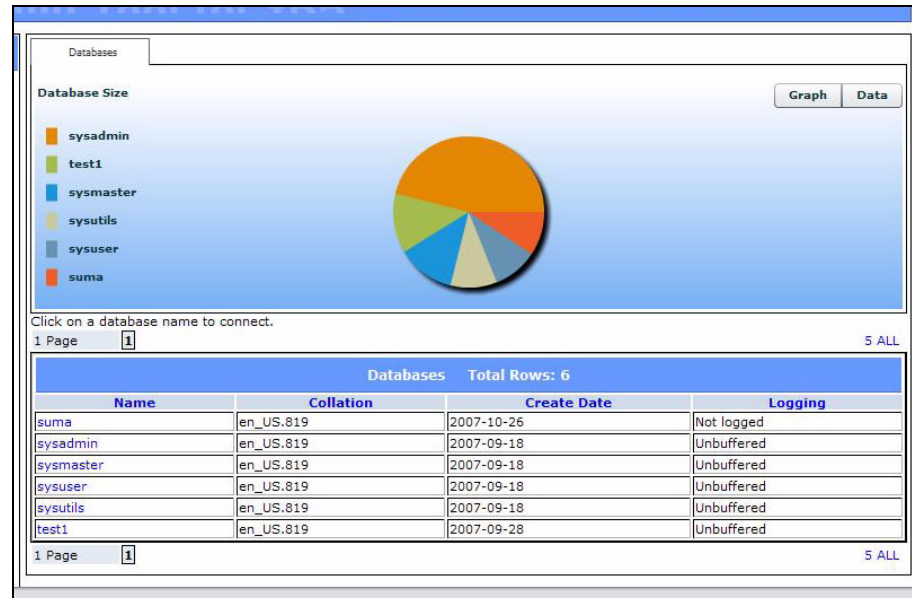


Figure 5-14 Databases

- *Schema Browser* shows the same view as when you click each database. It shows the tables and their details (Figure 5-15). You can also browse the tables by using the Browse button to see the different rows in it.

Include Database Catalog Tables? Submit

Display Options: **Text/Clob:** Show All Text **Byte/Blob:** Ignore Byte

2 Pages 1 2 > <

5 10 15 ALL

Tables													Total Rows: 19	
Browse	Name	Create Date	TabId	Partnum	Rowsize	NRows	NIndexes	Locklevel	Fextsize	Nextsize	Pages Used	View		
	command_history	2007-10-30	100	0x00100108	30784	0	2	Page	16	16	0	No		
	mon_checkpoint	2007-10-30	113	0x00100125	118	0	1	Page	16	16	0	No		
	mon_config	2007-10-30	108	0x0010011F	1035	0	0	Page	16	16	0	No		
	mon_memory_system	2007-10-30	112	0x00100124	36	0	0	Page	16	16	0	No		
	mon_onconfig	2007-10-30	109	NONE	1159	0	0	B	0	0	0	Yes		
	mon_prof	2007-10-30	116	0x00100129	18	0	0	Page	16	16	0	No		
	mon_profile	2007-10-30	117	NONE	30	0	0	B	0	0	0	Yes		
	mon_sysenv	2007-10-30	118	0x00100122	1282	0	0	Page	16	16	0	No		
	mon_table_names	2007-10-30	110	0x00100120	307	0	1	Page	16	16	0	No		

Figure 5-15 Schema Browser

- *SQL* shows the SQL Query Editor like the example in Figure 5-16. You can write an SQL statement or import from a file and execute it against any of the databases in the server.

Imin Tool for IDS

Connected: informix@ids_1110
Host: yogi.menlo.ibm.com
Database: sysadmin

SQL

SQL Query Editor: (one statement at a time)

Text/Clob Column Option: Show All

Byte/Blob Column Option: Ignore Column

Import query from: Browse...

Import

For queries with order by, group by, subqueries, distinct, unions, and aggregates, the number of rows to fetch is limited to:

100 Change Reset

Run Query Save Query to File Clear

Figure 5-16 SQL Query Editor

The OAT also provides Really Simple Syndication (RSS) feeds, which facilitates automatic checking possible through an RSS reader. The *RSS* menu item shows the RSS feeds that are available. RSS feeds are available for chunks, databases, dbspaces, environment variables, logical logs, onconfig, online log, physical log, sessions, SQL statements, and virtual processors.

The *Dashboard* menu item shows a graphical view of the memory, space, locks, and transactions in the server.

Mach11 shows the Mach11 cluster topology. It is explained in more detail in Chapter 3, “Enterprise data availability” on page 75.

5.6 The Database Admin System

The Database Admin System is a framework that can simplify many tasks for DBAs, application developers and end users. In addition, these tasks can be easily integrated into a graphical admin system, such as, the OAT for IDS.

In this section, we examine how a DBA can take advantage of the different components of the Database Admin System to solve a real life problem. The problem the we explore is to how to remove users who have been idle for more than a specified length of time, but only during work hours. Prior to the database admin system, a DBA used several different operating system tools, such as shell scripting and cron. In addition, if this is a pre-packaged system, these new scripts and cron entries must be integrated into an installed script. In addition, this must be portable across all supported platforms.

If you are to use the Database Admin System, you only have to add a few lines to your schema file and you are done. Since this is only SQL, it has the advantage of being portable across different flavors of UNIX and Windows.

We use the following components:

- ▶ Database Scheduler
- ▶ Alert System
- ▶ User Configurable Thresholds
- ▶ SQL Administration API

We break the problem into the following separate parts:

1. Creating a tunable threshold for the idle time out
2. Developing a stored procedure to terminate the idle users
3. Scheduling this procedure to run at regular intervals

Lastly we view pages that show the completed work in the OAT for IDS. These pages graphically shows all the tasks in the system and allow users to drill down and see the scheduling details and parameters of a specific task.

5.6.1 Creating an idle timeout threshold

To create a threshold that can be easily changed, we insert a row into the `ph_threshold` table in the `sysadmin` database. This table stores all the threshold or configuration values that are used by the scheduler. When creating a threshold, we must supply the information provided in Table 5-5.

Table 5-5 Columns in the `ph_threshold` table

Column	Description
<code>name</code>	The parameter name
<code>task_name</code>	The name of the task in the <code>ph_task</code> table associated with this threshold
<code>value</code>	The value associated with the parameter
<code>value_type</code>	The data type of this parameter (STRING or NUMERIC)
<code>description</code>	A description of what this threshold does

We insert a new parameter into the `ph_threshold` table that can be updated to reflect the current idle timeout period, saving the work of re-writing the stored procedure if conditions change as shown in Example 5-39. We then allow the OAT to display this as a configurable item for the task.

Example 5-39 Insert statement for our threshold value

```
INSERT INTO ph_threshold
  (name,task_name,value,value_type,description)
VALUES
  ("IDLE TIMEOUT", "Idle Timeout","60","NUMERIC",
  "Maximum amount of time in minutes for non-informix users to be
idle.");
```

5.6.2 Developing a stored procedure to terminate idle users

When creating a stored procedure to be called by the scheduler, it can optionally take in two parameters that inform the procedure of when it was invoked and the procedure's unique task ID as shown in Example 5-40. This is often useful when passing information to other parts of the Database Admin System, such as, the Alert System.

Example 5-40 Passing two parameters to the stored procedure

```
CREATE FUNCTION idle_timeout( task_id INT, task_seq INT) ;
```

Next we retrieve the thresholds from the `ph_threshold` table. We do this by using a simple select and casting the result into our desired data type (an integer). See Example 5-41.

Example 5-41 Select statement to retrieve the threshold from `ph_threshold` table

```
SELECT value::integer
       INTO time_allowed
       FROM ph_threshold
       WHERE name = "IDLE TIMEOUT"
```

The main part of our stored procedure is the SELECT statement (Example 5-42 on page 210) to find all users who have been idle for more than a specified number of minutes. From the `systcblst` table, we select the last time a thread has executed on a virtual processor. If this time is longer than our predetermined idle threshold and this thread is an `sqlexec` thread (that is, not a system thread), then we pass the session ID (`sid`) to the SQL Administration API function `admin()`. The `admin()` function has been set up to call `onmode -z` to terminate a session.

Example 5-42 Main SELECT statement

```
SELECT admin("onmode","z",A.sid), A.username, A.sid, hostname INTO rc,
sys_username, sys_sid, sys_hostname
FROM
sysmaster:syrstcb A , sysmaster:systcblst B, sysmaster:syscblst C
WHERE
A.tid = B.tid AND C.sid = A.sid AND lower(name) in ("sqlexec") AND
CURRENT - DBINFO("utc_to_datetime",last_run_time) > time_allowed UNITS
MINUTE AND lower(A.username) NOT IN( "informix", "root")
```

The last part of the stored procedure checks the return code of the `admin()` procedure to see if the session successfully terminated. If the session terminated successfully, then an alert is inserted, logging the termination of an idle user as shown in Example 5-43. The optional arguments to the stored procedures that are used to uniquely identify the task and the task sequence are required by the alert system. This allows the alert system to know who generated this alert and when this alert was generated. Several other items are required by the alert system, such as the `alert_type` (ERROR, WARNING, INFO) or the `alert_color` (ERROR, WARNING, INFO), and a message indicating what has happened.

Example 5-43 INSERT statement to add an alert to ph_alert table

```
INSERT INTO ph_alert( ID, alert_task_id,alert_task_seq, alert_type,
alert_color, alert_state, alert_object_type,
alert_object_name,alert_message, alert_action ) VALUES
(0,
task_id,
task_seq,
"INFO",
"GREEN",
"ADDRESSED",
"USER",
"TIMEOUT",
"User "||TRIM(sys_username)||"@"||TRIM(sys_hostname)||
" sid("||sys_sid||")"||
"terminated due to idle timeout.",
NULL
);
```

After an alert is created, the OAT for IDS shows it under the Health Center menu option of Show Alerts. If you do not see this alert at first, make sure that you have selected the ADDRESSED check box. This problem is automatically managed by the stored procedure. Therefore, the alert is already addressed. The second line in Figure 5-17 shows this alert.

Severity		Error Type		State			
<input checked="" type="checkbox"/> RED	<input checked="" type="checkbox"/> YELLOW	<input checked="" type="checkbox"/> ERROR	<input checked="" type="checkbox"/> WARNING	<input checked="" type="checkbox"/> NEW	<input checked="" type="checkbox"/> ADDRESSED	<input checked="" type="checkbox"/> ACKNOWLEDGED	<input type="button" value="View"/>
<input type="checkbox"/> GREEN		<input type="checkbox"/> INFO		<input type="checkbox"/> IGNORED			
1 Page		1				5 ALL	

Alert List						
ID	Type	Message	Time	Recommendation	Alert State	
6337	WARNING	TASK NAME mon_user_count_t LOCATION ERROR - 206 The specified table (mon_user_count_t is not in the database. ERROR - 111 ISAM error: no record found.	2007-10-26 13:24:06		NEW	Re-Check Ignore
6336	INFO	User suma@yogi sid(689) terminated due to idle timeout.	2007-10-26 13:19:29		ADDRESSED	Re-Check Ignore
6294	WARNING	TASK NAME UpdateStat_mydb LOCATION ERROR -201 A syntax error has occurred.	2007-10-26 09:57:16		NEW NEW	Re-Check Ignore
6097	WARNING	DbSPACE [dbs2] has never had a level 0 backup. Recommend taking a backup immediately.	2007-10-25 17:41:02			Re-Check Ignore
6098	WARNING	DbSPACE [DBS2] has not had a backup for 13812 01:41:02. Recommend taking a backup Immediately.	2007-10-25 17:41:02		NEW	Re-Check Ignore

Figure 5-17 Show Alerts page showing that session 689 has been terminated

Complete stored procedure

Example 5-44 shows the complete stored procedure for the scenario that we defined.

Example 5-44 Complete stored procedure

```

/*
*****
* Create a function which will find all users that have
* been idle for the specified time. Call the SQL admin API to
* terminate those users. Create an alert so we can track which
* users have been terminated.
*****
*/
CREATE FUNCTION idle_timeout( task_id INT, task_seq INT)
RETURNING INTEGER

```

```

DEFINE time_allowed INTEGER;
DEFINE sys_hostname CHAR(16);
DEFINE sys_username CHAR(257);
DEFINE sys_sid      INTEGER;
DEFINE rc           INTEGER;

{*** Get the maximum amount of time to be idle ***}
SELECT value::integer
  INTO time_allowed
  FROM ph_threshold
     WHERE name = "IDLE TIMEOUT";

{*** Find all users who are idle longer than the threshold ***}
FOREACH SELECT admin("onmode","z",A.sid), A.username, A.sid,
hostname
  INTO rc, sys_username, sys_sid, sys_hostname
  FROM sysmaster:sysrstcb A , sysmaster:systcblst B,
     sysmaster:sysscbllst C
  WHERE A.tid = B.tid
     AND C.sid = A.sid
     AND lower(name) in ("sqlexec")
     AND CURRENT - DBINFO("utc_to_datetime",last_run_time) >
time_allowed UNITS MINUTE
     AND lower(A.username) NOT IN( "informix", "root")

{*** If we successfully terminated a user log ***}
{*** the information into the alert table ***}
IF rc > 0 THEN
  INSERT INTO ph_alert
    (
      ID, alert_task_id,alert_task_seq,
      alert_type, alert_color,
      alert_state,
      alert_object_type, alert_object_name,
      alert_message,
      alert_action
    ) VALUES (
      0,task_id, task_seq,
      "INFO", "GREEN",
      "ADDRESSED",
      "USER","TIMEOUT",
      "User "||TRIM(sys_username)||"@ "||TRIM(sys_hostname)||
      " sid("||sys_sid||")"||
      " terminated due to idle timeout.",
      NULL
    )

```

```

);
    END IF
  END FOREACH;
  RETURN 0;
END FUNCTION;

```

5.6.3 Scheduling a procedure to run at regular intervals

The last part is to schedule the `idle_timeout` stored procedure that was previously created to run at regular intervals. This is accomplished by inserting a row into the `ph_task` table in the `sysadmin` database (see Table 5-6).

Table 5-6 Relevant columns in `ph_task` table

Column	Description
<code>tk_name</code>	The name of the task. Must be a unique name.
<code>tk_type</code>	The type of task (TASK, SENSOR, STARTUP TASK, and STARTUP SENSOR).
<code>tk_group</code>	The name of the group to associate the task with, for organization purposes. See the <code>tk_group</code> table for more details.
<code>tk_description</code>	A comment to describe what this task is doing.
<code>tk_execute</code>	A function name of the SQL statement to execute.
<code>tk_start_time</code>	Time of day to start executing this task.
<code>tk_stop_time</code>	Time of day to stop executing this task.
<code>tk_frequency</code>	Time of day to stop executing this task.

While many scheduling options are available, we keep it simple. In Example 5-45, the `idle_timeout` procedure is set to run every day between the hours of 6 a.m. and 6 p.m. at 10 minute intervals. While most of the insert statements look straightforward, the one key point to highlight is that the `tk_execute` column is receiving the name of the stored procedure to run.

Example 5-45 Task to schedule the `idle_timeout`

```
INSERT INTO ph_task
(
tk_name,
tk_type,
tk_group,
tk_description,
tk_execute,
tk_start_time,
tk_stop_time,
tk_frequency
)
VALUES
(
"Idle Timeout",
"TASK",
"USER",
"Remove all idle users from the system.",
"idle_timeout",
DATETIME(06:00:00) HOUR TO SECOND,
DATETIME(18:00:00) HOUR TO SECOND,
INTERVAL ( 10 ) MINUTE TO MINUTE
);
```

5.6.4 Viewing the task in the Open Admin Tool

After these three components are built, you can see how the OAT for IDS shows the tasks. Looking at the current overview of the task in Figure 5-18, the task that was added, Idle Timeout, is displayed in the overview.

Cron Task List			
Name	Group	Description	Next Execution
mon_config_startup	server	Collect information about database servers configuration file (onconfig). This will only collect parameters which have changed.	
mon_sysenv	server	Tracks the database servers startup environment.	
ifix_ha_monitor_log_replay_task	server	Monitors HA secondary log replay position	
mon_user_count1	server	Count the number of user count	2007-10 12:09
mon_user_count	server	Count the number of user count	2007-10 12:12
mon_user_count_t	server	Count the number of user count	2007-10 12:14
Idle Timeout	user	Remove all idle users from the system.	2007-10 12:19
mon_checkpoint	server	Track the checkpoint information	2007-10 12:41
mon_profile	performance	Collect the general profile information	2007-10 13:41
mon_vps	cpu	Process time of the Virtual Processors	2007-10 13:41

Figure 5-18 Idle Timeout task on the Task List page

Drilling down into the details of the Idle Timeout task, you see a complete list of scheduling details along with any associated parameters. Figure 5-19 shows the details of this task. In this example, we have one parameter called IDLE TIMEOUT.

The screenshot displays the 'Task Details' configuration interface for the 'Idle Timeout' task. The interface includes the following fields and options:

- Task Name:** Idle Timeout
- ID:** 26
- Description:** Remove all idle users from the system.
- Start Time:** 6 : 0 :00
- Stop Time:** 18 : 0 :00
- Frequency:** 0 Days 0 Hours 10 Minutes
- Monday:** Enabled
- Tuesday:** Enabled
- Wednesday:** Enabled
- Thursday:** Enabled
- Friday:** Enabled
- Saturday:** Enabled
- Sunday:** Enabled

A 'Save' button is located at the bottom right of the configuration area.

Figure 5-19 Details of task Idle_Timeout



An extensible architecture for robust solutions

In this chapter, we begin to change the subject matter, by moving from topics about the installation and configuration of the data server to the customization of its functional capabilities. The dimension of customization forms the boundary for the topics that are presented in the remainder of this book.

By extending the data server with new data types, functions, and application-specific structures, developers build solutions that have the following characteristics:

- ▶ Take advantage of data models that closely match the problem domain
- ▶ Depart from strict relational normalization to achieve better performance
- ▶ Implement powerful business logic in the data tier of the software stack
- ▶ Handle new types of information

We call these solutions *robust* because the elegance and economy of their architecture gives them higher performance, maintainability, responsiveness, and flexibility in the face of changes in environment, assumptions, and requirements. Here, we introduce *object-relational extensibility*, which is the concept that makes it all possible. We also show how the Informix Dynamic Server (IDS) has taken this key ingredient farther than any other data server, putting the full power of customization in the hands of developers and users, to enable customization of IDS for your environment.

6.1 DataBlades: Components by any other name

The basis for functional customization in IDS is the ability to add components, which are packages that contain data types, functions, index methods, and anything else that enhances or expands the functional capabilities. Component-based software is ubiquitous, from Web browsers, such as Firefox, and office productivity tools, such as Excel, to development platforms, such as Eclipse.

While relational database management systems (RDBMS) have long allowed for scripting through stored procedures, they are mostly large, closed, monolithic programs with no facility for adding new capabilities. IDS has the well-crafted architecture, tools, and interfaces to make it an effective container for components big and small, allowing it to respond to any data management challenge. Instead of the plugins, add-ons, and add-ins of other software, for IDS we use the term *DataBlade* or, more generically, *extension*. However, they are basically other words for the same idea of *software components*.

6.1.1 Object-relational extensibility

The basis for these data server components is *object-relational extensibility*, which entails combining the power of object-oriented programming with the proven data management advantages of the relational model. In application programming languages, such as Java, there are facilities for creating classes and objects with inheritance, encapsulation, function polymorphism, and other concepts that together support object-oriented programming. This is a style that, when used skillfully, results in maintainable, flexible, and reusable code.

A problem arises when such code must connect to a database. That is, the relational model forces us to map the problem structure to a schema consisting of tables with visible columns whose types come from a limited set of fundamental alphanumeric types, whose connections are encoded with primary and foreign keys, and whose general structure is a poor match with the object-oriented picture.

To address this impedance mismatch, in the late 1990s, several enhancements were made to SQL to make it possible to build a schema from elements that more closely match the object class structure of the application. Row types, collection types, user-defined routines (UDRs), typed tables, and table inheritance are some of these enhancements. They enhanced the expressive power of SQL and brought some benefits of object-oriented design to databases. However, it is useful to remember that IDS is still much a relational database and that SQL is still SQL. Therefore, the object-relational capabilities tend to have more to do with relational capabilities and less with object-orientated capabilities.

More significant, however, is that IDS went further along this general trend and came up with such features as opaque types, virtual table and index interfaces, and user-defined aggregates. It also came up with a well-documented, powerful DataBlade API to support the development of high-performance user-defined types (UDTs) and UDRs implemented in C. While these capabilities may or may not look like object orientation, they are critical enablers of extensibility. Extensibility is the key ingredient that makes IDS the right data foundation for such a wide variety of applications.

In this book, the emphasis is on these programming-based extensibility features rather than the more widely supported object-oriented enhancements to SQL. They make entirely new classes of applications possible and allow the data server to perform its central role in the solution architecture by hosting its share of application logic.

In the following sections, we explore, at a high level, this power of extensibility and show how it supports a more robust software solution architecture. The chapters that follow give real examples in greater detail.

6.2 Data types that match the problem domain

One of the great advantages of object-oriented programming is that the problem can be modeled in code that describes what it is about in terms of the application domain and hides the details of the implementation. The details of the implementation can even change without affecting anything that calls or uses it. To develop in this way, among other things, you need a language that lets you define new classes of objects. IDS supports SQL statements that let you do just that with UDTs in a few different ways. These are the model building blocks.

6.2.1 Coordinates

To illustrate coordinates, consider an example from the world of geographic or spatial data. In many situations, we want to manage data that represents locations in the real world. The most common way to represent a location is by a set of coordinates. *Coordinates* are numeric values that give you the distance from some reference origin in each of two or three dimensions. To keep it simple, assume that we are dealing with two dimensions in space, as you do when you use a paper map. In that case, a point location is mathematically represented as a coordinate pair (x,y) as illustrated in Figure 6-1 on page 222.

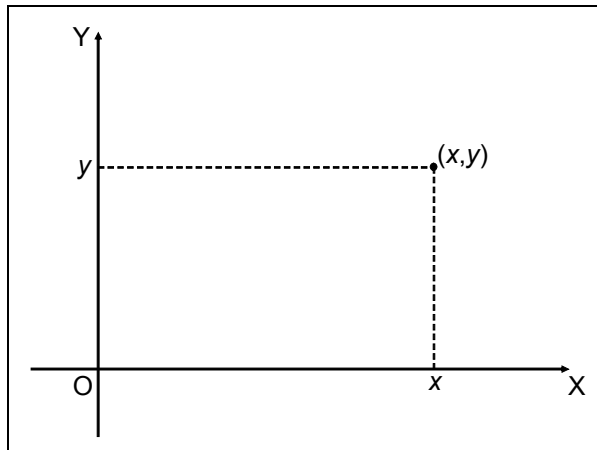


Figure 6-1 A point (x,y) in a two-dimensional coordinate system

In a relational table, you can easily represent this, as shown in Example 6-1.

Example 6-1 A table with spatial coordinate columns

```
CREATE TABLE sites (id INT, name VARCHAR(40), ..., x FLOAT, y FLOAT);  
INSERT INTO sites VALUES (1, 'one', ..., 20.0, 10.0);
```

...

It is not difficult to think of interesting spatial queries that are easily expressed in SQL, such as finding the distance to a given location by using a simple Pythagorean theorem calculation, as illustrated in Figure 6-2.

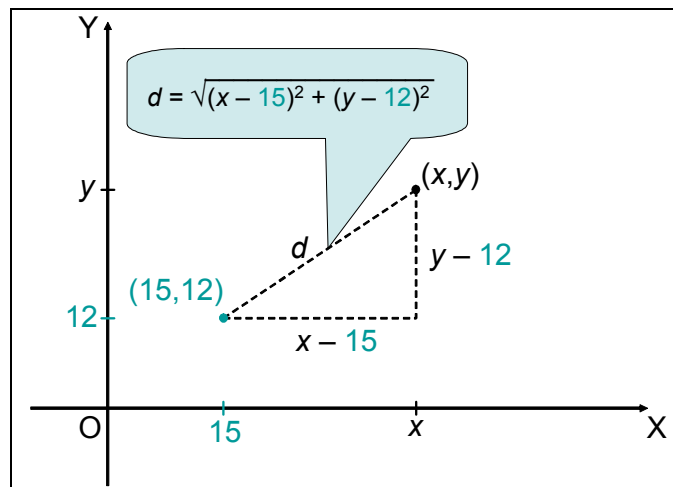


Figure 6-2 Calculation of distance, d , between two points

Example 6-2 shows the corresponding spatial query.

Example 6-2 A spatial query on coordinate columns

```
-- The given location is (x,y) = (15,12)
SELECT id, name, Sqrt((x-15)*(x-15)+(y-12)*(y-12)) AS distance
FROM sites ORDER BY 3;
```

id	name	distance
1	one	5.38516
...

Even better, we can write user-defined routines (UDRs) in SPL, C, or Java that make such queries easier to express and more self-documenting. We can also put the implementation in one place to be used by every client application. For the sake of simplicity, Example 6-3 uses SPL to implement a distance function. The output is the same as in Example 6-2, and therefore, it is not repeated here.

Example 6-3 An SPL function for distance on coordinate columns

```
CREATE FUNCTION Distance(x1 FLOAT, y1 FLOAT, x2 FLOAT, y2 FLOAT)
RETURNING FLOAT
WITH (NOT VARIANT)
RETURN Sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
```

```
END FUNCTION
```

```
SELECT id, name, Distance(x, y, 15, 12) AS distance  
FROM sites ORDER BY 3;
```

Still, the problem is that, for nearly all purposes, *x* and *y* only have meaning when used together. Individually, they mean (almost) nothing. Moreover, the distance function takes four FLOAT arguments. But, is the relationship between them, for example, $x_1 - y_1 - x_2 - y_2$ or $x_1 - x_2 - y_1 - y_2$? It can quickly become confusing if we add more arguments, for example, for working in three dimensions or to carry along information about the coordinate system for each location.

Named row types

A solution is to combine the coordinates into a single *named row type* and apply the Distance function to it as shown in Example 6-4. Again, the query result is the same as in Example 6-2, and therefore, it is not repeated here.

Example 6-4 A spatial table and function using a row type

```
CREATE ROW TYPE Point (x FLOAT, y FLOAT);  
CREATE FUNCTION Point (x FLOAT, y FLOAT)    -- Create a constructor  
RETURNING Point                            -- function for better  
WITH (NOT VARIANT)                        -- syntax  
    RETURN ROW(x, y)::Point;  
END FUNCTION;  
  
CREATE FUNCTION Distance (point1 Point, point2 Point)  
RETURNING FLOAT  
WITH (NOT VARIANT)  
    RETURN Sqrt((point1.x-point2.x)*(point1.x-point2.x) +  
                (point1.y-point2.y)*(point1.y-point2.y));  
END FUNCTION  
  
CREATE TABLE sites (id INT, name VARCHAR(40), ..., location Point);  
INSERT INTO sites VALUES (1, 'one', Point(20, 10));  
...  
SELECT id, name, Distance(location, Point(15, 12)) AS distance  
FROM sites ORDER BY 3;
```

To make it more realistic, we can search for points in a specified rectangular window, for example, to retrieve the points needed to draw a map as shown in Example 6-5.

Example 6-5 Selecting points from a rectangular window

```
SELECT * FROM sites
WHERE location.x BETWEEN 15 AND 35
      AND location.y BETWEEN 5 AND 25;
```

This still puts too much of the implementation (the BETWEEN predicates) in the application. To avoid this, we can define a new type, called Box. It represents a rectangular area, whose edges are parallel to the coordinate axes by its southwest (or lower-left) and northeast (or upper-right) corners as shown in Figure 6-3.

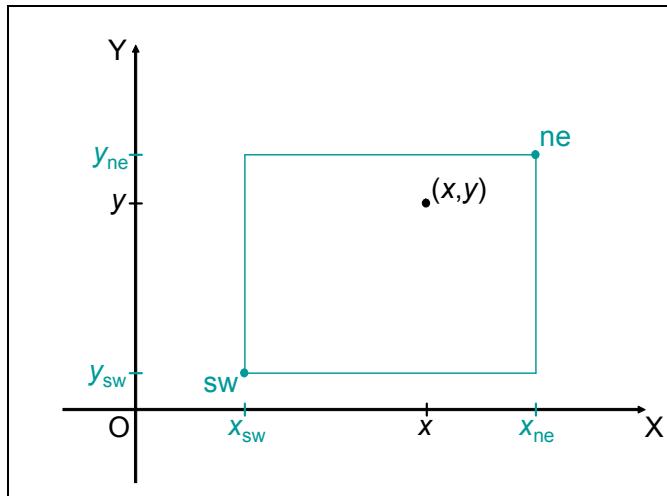


Figure 6-3 Expressing a window query by using a box object

We can then use it to define a `Within` function, which we use in turn to express the same query as in Example 6-5 on page 225. This is illustrated in Example 6-6.

Example 6-6 A window query using a `Box` data type and `Within` predicate function

```
CREATE ROW TYPE Box (sw Point, ne Point);
CREATE FUNCTION Box (sw Point, ne Point)  -- Constructor function
RETURNING Box                            -- for better syntax
WITH (NOT VARIANT)
    RETURN ROW(sw, ne)::Box;
END FUNCTION;

CREATE FUNCTION Within (loc Point, window Box)
RETURNING BOOLEAN
WITH (NOT VARIANT)
    IF (loc.x BETWEEN window.sw.x AND window.ne.x AND
        loc.y BETWEEN window.sw.y AND window.ne.y) THEN
        RETURN 't';
    ELSE
        RETURN 'f';
    END IF
END FUNCTION;

SELECT * FROM sites
WHERE Within (location, Box (Point (15, 5), Point (35, 25)));
```

All of this is from a problem modeling perspective, because it is closer to the way we think of maps and coordinate systems with points, distances, and rectangles. The spatial column has the type `Point`, and the `Distance` function simply takes two `Point` arguments. For a query that selects on points that lie within a given rectangular window, we use a `Box` data type and apply a `Within` predicate. Further, we have created reusable functions, `Distance` and `Within`. These functions give the significant benefit of having a single implementation of our logic, in the part of the software stack that is closest to the data. Applications can use them, and any changes and enhancements apply to those applications without difficult roll-out and installation of updates.

Naturally, to do this right, we need to create an index on a `Point` column that supports the `Within` predicate. If these queries must perform well and perhaps carry out more complex calculations, we want to implement the functions in a lower-level language such as C. But ignore these considerations for the moment, because there is a more important issue that has to do with the use of row types.

Opaque types: Encapsulation and better performance

Row types are helpful in organizing the database schema and making a data model a better representation of the application domain so that it is easier to understand. Some useful features, such as table inheritance, depend on row types. As such, they are an important feature of object-relational data servers. However, when it comes to the next level in extensibility, which is building powerful, reusable components, row types fall short.

The main problem with row types, aside from performance limitations, is that they offer no encapsulation. The implementation of the `Point` data type as two `FLOAT` elements is visible for all to see, and likewise for the `Box` type. In effect, row types and their elements are merely a twist on tables and their columns. Row types tend to be inextricably tied to the data model and database schema for which they were originally developed. This means that application code can access the elements of the row type directly (for example, `location.x`), which makes it impossible for the implementation to evolve with changing requirements.

For example, what if our experience of using the data types that we developed in the previous examples suggests any of the following improvements:

- ▶ Compress the coordinate values in `Point`, currently two `FLOAT` elements taking 16 bytes, when storing them on disk.

Perhaps there is not much to be gained in a single point value, but you can see that it can make a big difference for geometric shapes with many points, such as lines and polygons.

- ▶ Use `INTEGER` coordinates.

For performance and algorithmic robustness, some spatial software packages internally use integer, not floating-point coordinates.

- ▶ Implement a `Box` as (`southwest corner`, `width`, `height`) instead of (`southwest corner`, `northeast corner`).

This is largely a matter of taste, but sometimes one implementation can be more efficient than the other.

- ▶ Implement a `Box` as (`center point`, `width`, `height`).

For some map display software, the center of the window is a much more important location than the corners. Therefore, it might make sense to reflect that in the implementation.

- ▶ Implement a `Box` as (`center point`, `radius`).

This might look like a mistake, but in some situations, it makes little sense to talk about rectangles, while searching over a circular area is highly efficient. Yet mapping applications expect to deal with boxes.

If existing applications access row type elements directly through expressions, such as `window.ne.x`, then our implementation must always include these elements. Even if you provide additional functions, sometimes called *accessors* or *get and set methods*, to get to the elements (so the `window.ne.x` becomes `X(NE(window))`), you cannot be sure that applications rigorously adhere to that style and never use the `type.element` notation. That is, you cannot hide information about the type's implementation.

To protect applications from implementation changes and, conversely, to avoid freezing the implementation because of existing applications, which in short is to achieve encapsulation, use a different kind of UDT called *opaque types*. In IDS, an opaque type is a type whose implementation is not known or visible to the data server nor, therefore, to any user or application that is using SQL. All interaction between the server, or the client, and the type goes through UDRs, including a prescribed set of callback UDRs called *support functions*.

Usually an opaque type is implemented as a data structure with associated code in C, but C++ and Java are also possible. A rich, stable DataBlade API gives C implementations the power to do almost anything. For example, they can perform IDS-specific memory management, file and stream I/O, connection management, tracing and exception handling, and smart large object manipulation.

Indexing: For all but the simplest extensions, it is important to support any new data types by an index. Good data modeling is helpful, but offers little practical value if the data server cannot perform queries fast. Consider the type of information that is represented by a new data type that does not lend itself to B-tree indexing (because it cannot be ordered numerically or alphabetically). In this case, a new secondary access method must be added, by using the *Virtual Index Interface* (VII). The only realistic method for implementing an access method is to use a tight, procedural language such as C. Therefore, it is not a big step to use C for the type itself.

Let us go back to the example with `Point` and `Box`. Figure 6-4 shows how the principle of encapsulation applies in this context. The specific implementation of each type as a C structure with supporting functions (functions not shown) is hidden inside a box whose border is opaque. Only the public interfaces (UDRs) are visible to the developer or user.

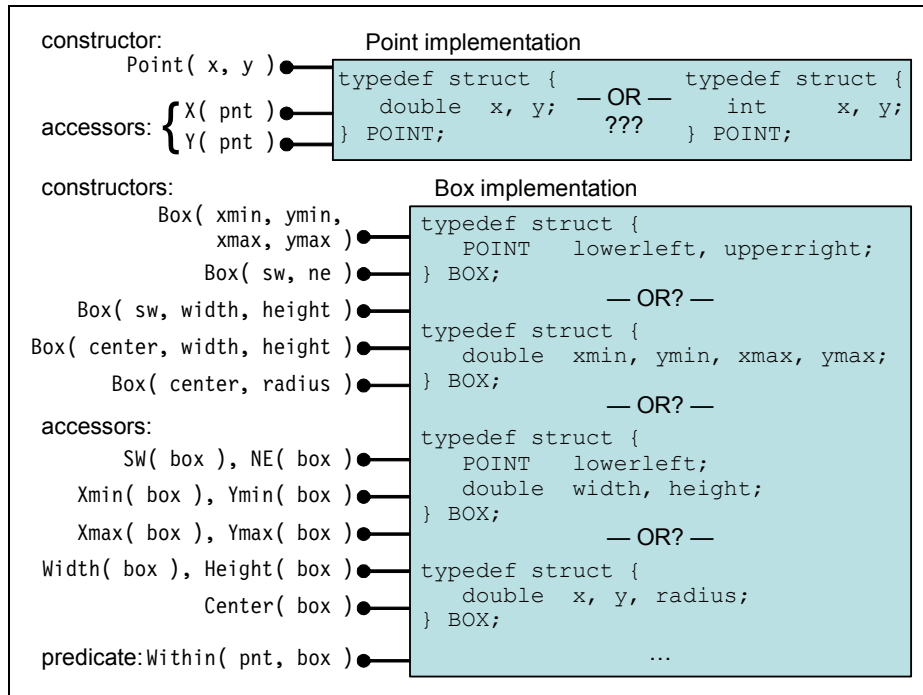


Figure 6-4 Example of Point and Box as opaque data types with encapsulation

Multiple implementations for each are possible, hidden from the application code. Only the public methods or interfaces, indicated by the ball-on-stick symbols, can be used by the application code. Many alternative interfaces can be published, regardless of actual implementation.

Of course, there is nothing new in this to the object-oriented application programmer. But in traditional SQL data servers, you cannot practice information hiding in this manner. In regard to the SQL for implementing UDRs and queries, nothing much changes except that we can no longer use the `type.element` notation in our SPL UDRs. Instead, we can only use the public interfaces that are defined for the type, as shown in Example 6-7.

Example 6-7 Distance and Within functions with opaque types

```
CREATE FUNCTION Distance (point1 Point, point2 Point)
RETURNING FLOAT
WITH (NOT VARIANT)
RETURN Sqrt((X(point1)-X(point2))*(X(point1)-X(point2)) +
(Y(point1)-Y(point2))*(Y(point1)-Y(point2)));
END FUNCTION
```

```

CREATE FUNCTION Within (loc Point, window Box)
RETURNING BOOLEAN
WITH (NOT VARIANT)
  IF (X(loc) BETWEEN X(SW(window)) AND X(NE(window)) AND
      Y(loc) BETWEEN Y(SW(window)) AND Y(NE(window))) THEN
    RETURN 't';
  ELSE
    RETURN 'f';
  END IF
END FUNCTION;

```

In reality, as alluded to previously, these functions are more likely to be implemented in C, along with the opaque types themselves and indexing support.

A real DataBlade component is the result of packaging a number of UDTs, UDRs, access methods, custom catalog tables, documentation, demos, data, client-side header files, and anything else that makes it useful into something that can be distributed, installed into an IDS instance, and registered into a database by using standard IDS tools. Nothing in this description requires that a DataBlade be large, complex, sophisticated, or unique.

Sometimes a single UDR, or a simple type that can be a variation on a built-in type, with one or two accompanying functions, can greatly impact the performance, maintainability, and robustness of an entire group of applications. To illustrate this, the following sections describe three small but real examples. The first two examples are extensions of the data server's support for time-based data types. The third is a new numeric data type that is illustrated with an extensive application example.

6.2.2 Date types

With some variations, all relational data servers have a collection of date and time data types. IDS has DATE, TIME, and DATETIME. This is useful in many situations. However, when a DATE column, for example, must do more than simply record date values and return them to the application on request, the limitations of the type can get in the way. When used in expressions and business logic, dates often need to be differentiated according to their significance to the business process. The regular DATE type does not do this.

A delivery service

Imagine a delivery service that has overnight, second-day, and ground (five days) service and delivers only on business days. Simple business logic can benefit from a variation on the DATE type that properly differentiates between business

days and non-business days (weekends and holidays). In the delivery operations database, such columns as `pick_up_date`, `estimated_delivery_date`, and `actual_delivery_date` can have a type of `BusDate`, for example. This type supports the same date representations as `DATE`, and it understands date-interval arithmetic. But only dates of actual business days are valid values, and only business days count in interval computations. This allows easy implementation of the following rules:

- ▶ Reject valid but wrong dates as `pick_up_date` values. Trying to enter `'11/03/2007'::BusDate` (assuming a USA locale) generates an error, because it is a Saturday, which is not a business date.
- ▶ Compute `estimated_delivery_date` from `pick_up_date` and `service_type`. For example, if `service_type` is `'ground'` and `pick_up_date` is `'12/20/2007'` (a Thursday), the expression `pick_up_date + 5` gives `'12/28/2007'` (which is Friday of the following week), skipping the weekend and Christmas Day.
- ▶ Assuming that the difference between estimated and actual delivery date is important (for customer complaint resolution or internal quality control), the expression `actual_delivery_date - estimated_delivery_date` yields the difference in business days, not elapsed calendar days.

The `BusDate` UDT and its `BusInterval` companion are conceptually simple types that can have a big impact on application consistency and simplicity. Most systems implement the kind of logic outlined previously in application code. However, such code must be reused or re-implemented consistently in many applications, which is a much more complex challenge than making it available once, in the data server.

This is not to say that the implementation itself is trivial. Knowing about holidays requires the maintenance of a calendar table, which must be accessible for localization purposes and annual updates. However, that only amplifies the point. It is better to maintain the calendar in the database, and run the computations that use it in the data server. This is preferred to having the calendar in multiple applications or requiring those applications to retrieve the information from the database for their computations.

Finally, because we are discussing a fairly simple twist on a built-in data type, it is possible to gain most of the benefits of extensibility without implementing entirely new types. All it takes is some UDRs that operate on normal `DATES`:

- ▶ A check constraint on `IsBusinessDay(pick_up_date)` validates entries.
- ▶ `NextBusinessDay(pick_up_date, 5)` computes `estimated_delivery_date`.
- ▶ `BusinessDays(actual_delivery_date, estimated_delivery_date)` calculates a delivery delay.

In some cases, it is obvious that a new UDT is required. In others, anything but a UDR or two is clearly overkill. Sometimes, however, it is a matter of preference

more than of objective criteria to decide when a new data type is required and when a few well-chosen and well-named functions will do. A UDT takes more effort to develop initially. The advantage is that, when it is implemented, it becomes an entirely natural extension of the data server's type system, leading, among other things, to more natural expressions in queries: `date + 5` rather than `NextBusinessDay(date, 5)`. And more natural and readable expressions can pay off in reduced development time and increased maintainability of application code.

The next example takes all its power from a single UDR.

The birthday club

Many restaurants have a birthday club (BC), which is essentially a customer loyalty list. The benefit to the customer is that they get an announcement close to their birthday, inviting them to come in and enjoy a free order of something.

Assume that we have a table called `BC_members`, that records the contact information and date of birth (in a `DATE` column called `dob`) for all members. The logic behind the mailing is simple enough. That is, once a week, run a query (shown in Example 6-8) to retrieve names and addresses for all BC members whose birthdays are coming up in a week, which is between 8 and 14 days from now.

Example 6-8 BC mailing batch query (flawed attempt)

```
SELECT name, address FROM BC_members
WHERE dob BETWEEN TODAY+8 AND TODAY+14;
```

Unfortunately, this query not give quite the results we want, because those birth dates have years attached to them, and none of them will be in the future. In reality, we want to use the birth *days* (without the year), not birth *dates*, since we really do not care how old the members are. Of course, for a different business process, such as purging a kids-only club of members who have become adults, the year may be significant.

In this case, a single UDR might provide the solution. `NextBirthday(dob DATE)` returns a `DATE` value that is the actual date, in the current or the next year, of the next birthday for someone born on the given date. This is not simply the original date of birth stripped of its year and extended again with the current year, as in the following expression:

```
EXTEND(EXTEND(dob, MONTH TO DAY), YEAR TO DAY)
```

In contrast, this function makes sure that any birthday earlier on the calendar than today is placed in the following year. Perhaps more interestingly, it also ensures that any leap day date of birth (February 29) is mapped to an

appropriate date in common years, such as February 28. Otherwise, those born in leap years might only get their invitation once every four years.

Example 6-9 shows the new weekly mailing query and examples of the output of NextBirthday.

Example 6-9 BC mailing batch query (improved) with sample birthdays

```
SELECT name, address FROM BC_members
WHERE NextBirthday( dob ) BETWEEN TODAY+8 AND TODAY+14;
```

```
If TODAY is 07/15/2006:
  NextBirthday( '10/24/1991' ) returns 10/24/2006
  NextBirthday( '04/12/2000' ) returns 04/12/2007
  NextBirthday( '02/29/1996' ) returns 02/28/2007
If TODAY is 07/15/2007:
  NextBirthday( '02/29/1996' ) returns 02/29/2008
```

This simple database extension, a single function, can tremendously simplify an application and improve its performance by applying the tricky but simple logic right where the data is. This query still requires a table scan because there is no way to index a function result that depends on the current date (NextBirthday must be declared a VARIANT function). But that is still much better than having to retrieve all rows into the application to apply the one-week filter there.

A further improvement, of course, is to find a way to express this query in a way that can be supported by an index. This requires a new data type and is left as an exercise for you. If you develop a solution, contact IBM developerWorks by sending an e-mail from the Feedback page at the following Web address:

<https://www.ibm.com/developerworks/secure/feedback.jsp>

Additional useful date manipulation examples are provided 7.1, “Manipulating dates” on page 264.

We now turn to a more elaborate example involving a new numeric type.

6.2.3 Fractions

A *cadastre* is an agency that records units of land and their ownership. This involves maintaining both the legal description and geographic survey of each parcel (the map), as well as the ownership and other rights (such as mineral or grazing rights) that apply to the parcel, including the recording of ownership transactions. Naturally, a cadastre requires industrial-strength systems and software for managing legal information, transactions, and spatial surveys and maps. A small industry segment of IT vendors specializes in these types of

software, which in turn rely on more general-purpose tools, such as geographic information systems (GIS) and data servers, such as IDS.

Some cadastral departments are faced with peculiar requirements. For example, one department, an IDS user, received complaints about the apportionment of ownership (and the property tax liability that goes with it) when a parcel had not one, but many owners. This arises frequently when a parcel is owned by a family and passed down through the generations. The problem stemmed from the use of the DECIMAL (or NUMERIC) type, commonly used in databases of all types, to represent the ownership fraction.

FLOAT and REAL: The floating-point data types FLOAT and REAL, being *approximate numeric types*, are generally avoided in databases unless they represent data of a scientific or physically measured nature, such as the geographic coordinates discussed in 6.2.1, “Coordinates” on page 222.

Fractions, and more generally rational numbers, cannot always be represented as a fixed-point decimal number without rounding. In school, we learned about continuing decimal fractions (as in $1/3 = 0.333\dots$). But in a digital computer, we must round to the number of digits given in the data type of the column as demonstrated in the following example.

In this example, *A*, *B*, and *C* each have one-third interest in a 1000 m² family property. They decide to subdivide the property and sell the back third. The new ownership interest of *A*, *B*, and *C* is calculated as follows:

fractional: $(1/3) \times (2/3) = 2/9$; $(2/9) \times 1000 = 222 \text{ m}^2$
decimal: $0.33 \times 0.67 = 0.221$; $0.221 \times 1000 = 221 \text{ m}^2$

If you see the idea behind this problem and appreciate its importance in a real-world situation as well as the relief that a well-chosen UDT can bring, you may want to skip the to 6.3, “Denormalization for performance and modeling” on page 242. However, if you need a little more convincing or want a better feel for the specifics of a rational-number type, read on.

Parcel ownership: A worked-out example

To put a little more substance behind the argument, consider the following table of parcels, defined as in the following statement:

```
CREATE TABLE parcels
    (parcel INTEGER, value MONEY, tax_rate DECIMAL(5,3));
```

Table 6-1 summarizes the results of the statement.

Table 6-1 Sample parcels table

parcel	value	tax_rate
1	\$1,000,000.00	0.015
4	\$789,243.16	0.015

The parcels' ownership is recorded in the Ownership table (Example 6-10 and resulting Table 6-2). Each owner's interest in each parcel is a fractional amount (up to 1), recorded (for illustration purposes) as both a DECIMAL(5,3) number and a fraction.

Example 6-10 The ownership table structure

```
CREATE TABLE ownership
(
    parcel      INTEGER,
    owner       VARCHAR(255),
    interest_dec DECIMAL(5,3),
    interest_fr  Fract
);
```

Table 6-2 Sample ownership table - Initial state

parcel	owner	interest_dec	interest_fr
1	A	1.000	1/1
4	B	1.000	1/1

The Fract data type

The Fract data type is a custom UDT that helps solve this particular customer's problem. It can be implemented in several ways. In the spirit of the discussion in "Opaque types: Encapsulation and better performance" on page 227, we assume that it is an opaque type. Therefore, the implementation is hidden from the server. For example, the data structure in C, and the corresponding SQL type definition, can be as shown in Example 6-11.

Example 6-11 Internal representation and SQL definition of the Fract data type

```
typedef struct          /* Internal representation in C */
{
    mi_integer numerator;
    mi_integer denominator;
}
Fraction;

create opaque type Fract  -- UDT definition in SQL
(
    internallength = 8,
    alignment      = 4
);
```

An important design choice is implicit in this structure. The *domain* is the set of rational numbers m/n , with m and n as whole numbers (integers) representable by 4-byte signed integers, that is, in the range $-2,147,483,647$ to $2,147,483,647$. Alternatives are 8-byte (mi_int8) or short (mi_smallint) integers.

Note: We use the name *Fract*, not “Fraction” to avoid any collisions with the IDS SQL keyword FRACTION (as in DATETIME YEAR TO FRACTION(3)). The C language has no such reserved word, so that we can freely use the full name.

In general, with IDS, you can use any name you choose. However, the SQL parser might return an error in some situations if the name matches a keyword, unless it is used as a delimited identifier (enclosed in double quotation marks; environment variable DELIMIDENT must be set). It is better to avoid any potential for collision and not use keywords. For more information about keywords, refer to the *IBM Informix Guide to SQL: Syntax*, G229-6375-01.

The preferred convention is to apply a unique three-character prefix to all type names and some or all other identifiers (such as names of functions and operator classes) associated with a given DataBlade. Therefore, we have the types *ST_Point* in the Spatial DataBlade, *GeoPoint* in the Geodetic DataBlade, the function *bts_contains* in Basic Text Search, and so on.

In addition to the internal representation, a UDT needs external representation: a binary one for exchange with applications (here, probably the same set of two 4-byte integers) and a text one. The obvious choice of text representation is something such as “*n/d*” (with numerator *n* and denominator *d*), as in the “1/1” values in the *Ownership* table. What is left is a possible convention for the sign, such as always attaching the sign to the numerator, so that it appears in front of the fraction: ‘*[-]n/d*’ with *n* meaning “not negative” and *d* meaning “positive.” On input, both *n* and *d* might be negative. Therefore, internally the value is constructed as follows:

$$\text{numerator} = \text{sign}(n/d) |n|, \text{ denominator} = |d|$$

For an opaque type, the support functions that convert between the type’s internal and external representations are identified by casts. In this case, a few additional casts and constructors are useful to connect the *Fract* type to the other numeric types:

- ▶ **IMPLICIT CAST** (*Fract* AS **DECIMAL**) turns 3/8 into 0.375 and lets any fraction participate easily in numeric expressions as in the following example:

```
SELECT o.interest_fr*p.value*p.tax_rate FROM ...;
```
- ▶ **IMPLICIT CAST** (**INTEGER** AS *Fract*) turns 7 into 7/1, which is convenient in the special case of whole values that must be treated as fractions.
- ▶ **FUNCTION** *Fract*(numerator **INTEGER**, denominator **INTEGER**) **RETURNING** *Fract* is a useful constructor from the integer numerator and denominator values, without going through a text representation. *Fract*(3, 6) returns 1/2.

A good implementation automatically simplifies the fraction in all functions that return a Fract value.

Finally, the Fract type needs all the appropriate functions to make it as usable as any numeric type:

- ▶ Relational operators equal (=), notequal (<> and !=), greaterthan (>), greaterthanorequal (>=), lessthan (<), and lessthanorequal (<=) automatically associated with their operator symbols
- ▶ A compare function, for server operations such as sorting and indexing
- ▶ Arithmetic operators plus (+), minus (-), times (*), divide (/), and negate (unary -)
- ▶ Additional algebraic functions Abs (absolute value) and Recip (reciprocal) (others are possible)

Important for this application are the arithmetic operations, which preserve the rational nature of the result, as shown in Example 6-12.

Example 6-12 Arithmetic operations on Fract values

```
SELECT 7/8 - 2/3 AS decimal, '7/8'::Fract - '2/3' AS fraction
FROM systables WHERE tabid=1;
```

```
decimal    0.208333333333333
fraction   5/24
```

Tax bills and ownership transactions

Now that we have the workings of our Fract data type, let us go back to the cadastral example. From the ownership records and the tax rate and parcel values, we can compute the tax bill for each owner, as shown in Example 6-13. Note the use of a collection subquery to show the list of parcels associated with each owner, in the notation (parcel, interest_dec, interest_fr). For example, (4,0.143,1/7) means parcel 4, with ownership interest of 0.143 (decimal) or 1/7 (fractional).

Example 6-13 Computation of owners' tax bills

```
SELECT
  o1.owner                AS owner,
  count(o1.parcel)        AS num_parcel,
  MULTISET
  (
    SELECT o2.parcel, o2.interest_dec, o2.interest_fr
    FROM ownership o2
    WHERE o2.owner = o1.owner
```



```

)
Sum (o1.interest_dec*p.rate*p.value) AS tot_owntax_dec,
Sum (o1.interest_fr *p.rate*p.value) AS tot_owntax_fr
FROM
ownership o1, parcels p
WHERE
p.parcel = o1.parcel
GROUP BY
o1.owner
ORDER BY
o1.owner;

```

Table 6-3 summarizes the results of the sample computation.

Table 6-3 Owners' tax bills for initial state of ownership table

owner	num_parcel	parcels	owntax_dec	owntax_fr
A	1	(1,1.000,1/1)	\$15,000.00	\$15,000.00
B	1	(4,1.000,1/1)	\$11,838.65	\$11,838.65

At this point, with each owner owning the entire parcel outright, there is no difference between the tax bills computed by using decimal or fractional interest.

Likewise, we can compute for each parcel the total owners' interest and tax assessed as shown in Example 6-14.

Example 6-14 Computation of total tax and owners' interest per parcel

```

SELECT
o.oid
Sum (o.interest_dec)
Sum (o.interest_fr)
Sum (o.interest_dec*p.rate*p.value)
Sum (o.interest_fr*p.rate*p.value)
AS parcel,
AS interest_dec,
AS interest_fr,
AS totaltax_dec,
AS totaltax_fr
FROM
ownership o, parcels p
WHERE
p.oid = o.oid
GROUP BY
o.oid
ORDER BY
o.oid;

```

Table 6-4 summarizes the results of the computation.

Table 6-4 Total owners' interest and tax per parcel for initial state

parcel	interest_dec	interest_fr	totaltax_dec	totaltax_fr
1	1.000	1/1	\$15,000.00	\$15,000.00
4	1.000	1/1	\$11,838.65	\$11,838.65

As expected, the total of all owners' interest in each parcel is exactly 1, and the total tax assessed for each parcel is the same regardless of how the owner's interest is recorded.

Now let us apply some transactions:

- ▶ Owner B transfers 1/7 of parcel 4 to D.

In terms of the ownership table, this means subtracting 1/7 from B's interest in parcel 4 and adding 1/7 to D's interest in parcel 4. Alternatively, since there is no record yet for an owner D with an interest in parcel 4, we can create one with 1/7 as the owner's interest.

Now rerun the tax computations, shown in Table 6-5. The totals per parcel have not changed, but the owners' tax bills have. Also a discrepancy has crept in between the decimal- and fraction-based tax computations. In Table 6-5, this is indicated by the following calculation:

error = the value in the ownertax_dec column – the value in the ownertax_fr column

The result is subsequently posted in the error column for the owner of that particular row.

Table 6-5 Owners' tax bills for after B transfers 1/7 of 4 to D

owner	num_parcel	parcels	ownertax_dec	ownertax_fr	error
A	1	(1,1.000,1/1)	\$15,000.00	\$15,000.00	
B	1	(4,0.857,6/7)	\$10,145.72	\$10,147.41	-\$1.69
D	1	(4,0.143,1/7)	\$1,692.93	\$1,691.24	\$1.69

As shown, the total tax for parcel 4 has not changed. However, due to rounding to three digits of the ownership interest, D is paying more than necessary, while B is paying less by the same amount. Of course, whether rounding really affects the end result depends on the number of digits kept (the DECIMAL(m,n) column declaration), the parcel value, and the tax rate.

- Owner A subdivides parcel 1 and distributes its ownership evenly over the six owners A, B, C, D, E, and F. Then, the six owners sell 1/5 of the parcel to owner G.

In the ownership table, delete any rows having to do with parcel 1. Then create new ownership records for each of the new owners, with a 1/6 interest in parcel 1. Next, since the six owners end up with their original interest in only $(1 - 1/5) = 4/5$ of the parcel, multiply their interest by that factor. Then create a new record reflecting G's 1/5 interest in parcel 1.

Again, rerun the tax computations. Table 6-6 shows the results.

Table 6-6 Owners' tax bills for after subdividing parcel 1 and selling off a part

owner	num_parcels	parcels	ownertax_dec	ownertax_fr	error
A	1	(1,0.134,2/15)	\$2,010.00	\$2,000.00	\$10.00
B	2	(4,0.857,6/7), (1,0.134,2/15)	\$12,155.72	\$12,147.41	\$8.31
C	1	(1,0.134,2/15)	\$2,010.00	\$2,000.00	\$10.00
D	2	(4,0.143,1/7), (1,0.134,2/15)	\$3,702.93	\$3,691.24	\$11.69
E	1	(1,0.134,2/15)	\$2,010.00	\$2,000.00	\$10.00
F	1	(1,0.134,2/15)	\$2,010.00	\$2,000.00	\$10.00
G	1	(1,0.134,2/15)	\$3,000.00	\$3,000.00	

Now, everyone except G is paying too much, although B's overpayment is reduced and D's increased by the same amount due to their joint ownership of parcel 4. This general overpayment is reflected in the per-parcel numbers shown in Table 6-7.

Table 6-7 Total owners' interest and tax per parcel after subdividing parcel 1

parcel	interest_dec	interest_fr	totaltax_dec	totaltax_fr	error
1	1.004	1/1	\$15,060.00	\$15,000.00	\$60.00
4	1.000	1/1	\$11,838.65	\$11,838.65	

While the total for parcel 4 is correct in both cases, masking the overpayment of one owner as it is compensated by the underpayment of another, the total for parcel 1 shows a discrepancy. That is, the total owner's interest is computed by using decimal values is greater than 100%, and the total tax for that parcel is off by \$60. This is clearly a nonsensical situation. While the actual tax error depends on the specifics of the parcel and the number of digits recorded, it is not possible

to avoid the erroneous total owners' interest. No matter what precision we choose for the `interest_dec` column, each individual value will be rounded up and the total will be too high. However, the fractional approach is correct regardless of the accumulation of subdivisions and partial sell-offs, always reflecting the ownership interest, parcel value, and tax rate to the penny.

A national cadastre had this problem, was under a legislative mandate to solve it, and did so by extending the database with a new data type for fractions. How many databases around the world suffer from the same inaccuracies, with real financial consequences?

6.3 Denormalization for performance and modeling

One aspect of matching the data server's data types, in the object-oriented spirit, to the problem domain is to let go of the relational mandate for normalization. Lack of normalization in a database design is generally frowned upon if it is the result of ignorance or product limitations, but *denormalization* is a time-honored technique, especially in fields such as data warehousing. Denormalization is the deliberate relaxation of normalization rules after designing a conceptually normalized schema. Many instances of denormalization involve data redundancy (repeating non-key attributes in multiple rows or tables). In this section, we discuss a different type of denormalization, namely the violation of the relational prohibition of non-atomic attributes, that is set-valued columns.

To illustrate the principle, consider a personnel department database with the following employee tables, not normalized (as depicted in Example 6-15 and Table 6-8 on page 243) and normalized (as depicted in Example 6-16, Table 6-9 on page 243, and Table 6-10 on page 243).

Example 6-15 An employees table

```
CREATE TABLE employees
(
  id          CHAR(5)          PRIMARY KEY,
  name       VARCHAR(255) NOT NULL,
  ...
  jobs_held  SET(VARCHAR(30))
);
```

Table 6-8 An employees table

id	name	...	jobs_held
12C34	Alice Jones	...	Developer, Manager
5G678	Bob Smith	...	Clerk, Administrator, Manager

Example 6-16 Employees tables, normalized

```
CREATE TABLE emp_names
(
  id    CHAR(5) PRIMARY KEY,
  name  VARCHAR(255) NOT NULL
);
CREATE TABLE emp_jobs
(
  id      CHAR(5),
  seq_no  INTEGER,
  job     VARCHAR(30) NOT NULL,
  PRIMARY KEY (id, seq_no)
);
```

Table 6-9 Normalized employee names table emp_names

id	name
12C34	Alice Jones
5G678	Bob Smith

Table 6-10 Normalized employee jobs table emp_jobs

id	seq_no	job
12C34	1	Developer
12C34	2	Manager
5G678	1	Clerk
5G678	2	Administrator
5G678	3	Manager

The structure shown in Example 6-15 on page 242 and Table 6-8 violates First Normal Form (1NF) due to the non-atomic jobs_held column. One of the reasons this is undesirable in a relational model is that it makes it difficult to ask questions such as, “Which employees have management experience?”, or, in

terms of a query, search for employees whose `jobs_held` set contains an element `Manager`. Alternatively, in the normalized schema, it is a simple matter to find the rows in `emp_jobs` that contain `Manager` in the `job` column.

Under some conditions, however, this type of denormalization is exactly what is needed, enhancing both the performance of the system and the expressive power of the logical data model. The condition that applies is that the normalization-violating, non-atomic values are in practice accessed exclusively (or mostly) in their entirety, not by pulling out individual elements. The `jobs_held` column in the `employees` table holds sets of values that are individually entered by a person. Each element is relevant independent of the other elements in the set. This is generally not the case for numeric arrays of physically observed or automatically recorded values, where processing tends to be done on the entire array, or large subsets of it. We discuss examples in the following sections.

6.3.1 Line shapes

Going back to the spatial example of 6.2.1, “Coordinates” on page 222, it is easy to imagine additional geometric shapes that could be modeled. One of these is a *curve*, an arbitrary one-dimensional object in the two-dimensional plane. While there are other possibilities, curves are commonly represented or approximated by a shape consisting of a series of concatenated line segments and usually called a *linestring* (a string of line segments), *polyline* (many segments), or simply *line* (a completely different use of the word from Euclidian geometry, where a line is a straight line extending infinitely in both directions). The points at which the line segments that make up the linestring begin, end, and connect are called *vertices* (singular *vertex*). Figure 6-5 shows the following examples:

- a A curve approximated by a line; vertices indicated by open circles
- b A line with six vertices that crosses itself
- c A line that is closed, meaning its begin vertex and end vertex are the same point

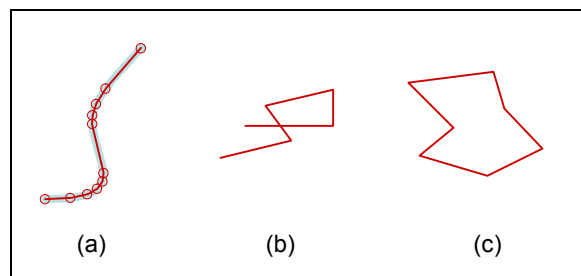


Figure 6-5 Three examples of lines

As with points and boxes, we mathematically represent each vertex in a linestring by its (X,Y) coordinates, listing all the vertices that define the line in order, from begin to end, uniquely defines the entire line. That is, we use an array of ordered pairs, as shown in Figure 6-6.

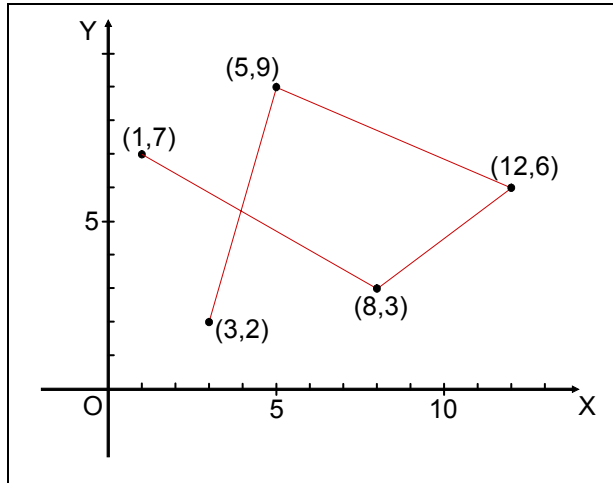


Figure 6-6 Line defined by array of ordered pairs $\{(3,2),(5,9),(12,6),(8,3),(1,7)\}$

In applications, it is highly unlikely that we want to find all lines that have a vertex at (21,6). Instead, we want to find lines that cross a given line, intersect or are contained within a given region, lie within a given distance of a given point, and so on. Each of these possibilities requires examining the entire line and, more importantly, considering all points lying on the line segments that connect the vertices. That is, the natural object that matches the problem domain most closely is the whole line, not the individual vertex.

In a strictly relational data server, with no ability to create extended types, this presents a problem. Because a line can have an arbitrary number of vertices (two or more), we cannot resort to separate columns for the individual X and Y coordinates in the main table.

A 1NF solution requires an auxiliary table to accommodate the variable number of vertices for each line, and might look something like what is shown in Example 6-17, Table 6-11, and Table 6-12.

Example 6-17 Normalized line table with auxiliary coordinate table

```
CREATE TABLE lines(id INTEGER PRIMARY KEY, name VARCHAR(255) NOT NULL);
CREATE TABLE lines_coords
(
  id      INTEGER,
  vtx_no  INTEGER,
  x       FLOAT NOT NULL,
  y       FLOAT NOT NULL,
  PRIMARY KEY (id, vtx_no)
);
```

Table 6-11 shows the sample main line table for Figure 6-6.

Table 6-11 Main line table

id	name
117	open loop

Table 6-12 shows the sample auxiliary line coordinate table for Figure 6-6.

Table 6-12 Auxiliary line coordinate table

id	vtx_no	x	y
117	1	3	2
117	2	5	9
117	3	12	6
117	4	8	3
117	5	1	7

This normalized representation is perfectly correct and can work, but it has two serious defects. First, it forces any application or database procedure to reassemble each line from its individual coordinate rows, potentially thousands or even millions of them, before it can work on any spatial expression or process. This hugely complicates all code and makes it difficult to determine the solution when examining the schema and the application logic. This is a classic consequence of the mismatch between the application object domain and the relational model.

Second, its performance and overhead are prohibitively bad. The solution involves an index scan over `id` in table `lines_coords` for each line found to get the coordinates. Each vertex has its own row, with all the overhead that entails, rather than a single row for each line. In addition, there is not much chance of a spatial indexing scheme to help spatial searches. In fact, solutions based on normalized models have been marketed in the past, including by major database vendors, but were quickly seen as of only academic interest and of no practical value precisely because of these defects.

With extensibility, the right design is easy to see. (How you implement a variable-length data type that can get arbitrarily large is another matter, which we are not concerned with here). An opaque UDT, `Line`, can represent the entire shape as a single column value, as in Example 6-18 and Table 6-13.

Example 6-18 Line table with UDT (constraints omitted)

```
CREATE TABLE lines(id INTEGER, name VARCHAR(255), shape Line);
```

The results of the `lines` table creation in Example 6-18 are depicted in Table 6-13. It is the sample line table, `lines`, with UDT (arbitrary text representation shown).

Table 6-13 Sample line table, lines, with UDT

id	name	shape
117	open loop	{{(3,2),(5,9),(12,6),(8,3),(1,7)}

With the `Line` data type, we incur the row and column overhead only once for each line. We also have the ability to create a spatial index on the `shape` column and apply predicates and functions, such as `Within` and `Distance` (see Example 6-7 on page 229) to formulate spatial queries. Best of all, the schema is simple and matches how we think of geometric shapes.

Fundamentally, the encapsulation provided by the opaque data type renders the value atomic from the point of view of the logical schema. There is no difference in principle between a `Line` value that contains an array of ordered pairs and, say, a `VARCHAR` value that contains an array of characters. Nor is there a difference between such a `Line` value and a `FLOAT` that contains 64 bits in several groups (sign, mantissa, exponent) with a specific meaning.

The next example is less obviously a modeling improvement and more purely a performance trick.

6.3.2 Time series

A *time series* is a time-indexed array, that is, a linear sequence where each element represents an observation at a specific point in time. Each element's time point is later than that of the preceding element. The elements themselves can be simple numeric values, such as a measured temperature. Or they can be an entire vector or record of many different values, such as the day's open, close, high, and low prices and trading volume for a specific security traded on a stock exchange. The time interval from element to element can be regular and predictable (for example, a daily precipitation measure) or irregular (for example, individual stock trades).

Macroeconomic (such as inflation and unemployment) and finance (such as stock symbols and indices) data are classic examples of time series. Figure 6-7 shows a graph of two regular time series and the share price (at market close) of the symbols IBM and ORCL. The time series is regular in that it is sampled at predetermined intervals (daily at close of trading), even though no observations are recorded on weekends and holidays.

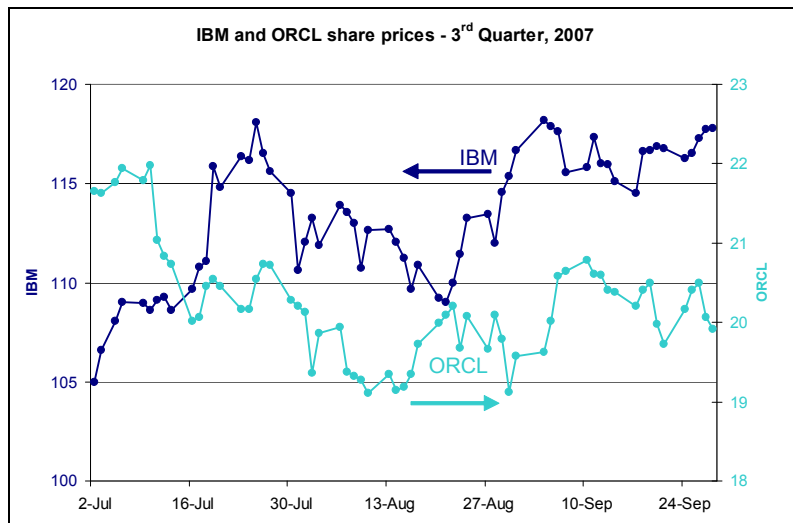


Figure 6-7 Regular time series: Daily share prices at close
Separate Y-axes: IBM (dark) on left, ORCL (light) on right

The relational and, with a few specialized exceptions, most common way to handle a time series is to give each observation or element its own row. Depending on the number and volatility of sources (such as telemetry devices and stock symbols), each source can have its own table, or all sources of the same type of information can share a single table.

Example 6-19 shows the single-table approach for share price data, tracking any number of quantities or variables, including the obvious high, low, open, and close prices.

Example 6-19 Sample time series table for share prices

```
CREATE TABLE share_prices
(
  symbol      CHAR(6),
  market_date DATE,
  high       MONEY,
  low        MONEY,
  open       MONEY,
  close      MONEY,
  volume     INTEGER,
  ...
  PRIMARY KEY (symbol, market_date)
)
```

Table 6-14 shows the first three days' worth of data from the two time series of Figure 6-7 on page 248.

Table 6-14 Sample contents of table share_prices

Symbol	market_date	high	low	open	close	volume	...
IBM	7/2/2007	\$105.01
ORCL	7/2/2007	\$21.65
IBM	7/3/2007	\$106.58
ORCL	7/3/2007	\$21.63
IBM	7/5/2007	\$108.05
ORCL	7/5/2007	\$21.77
...

For moderate amounts of data, this works fine. But as the table grows, and especially as the rate at which the data comes in accelerates, the overhead of recording each sample in its own row hurts the server's ability to keep up and to return query results quickly. Figure 6-8 on page 250 shows the conceptual data volume for a database managing 3,000 stock symbols over 24 years, tracking 65 different quantities or variables, such as open, close, high, low, and volume.

In Figure 6-8, the total number of rows is $3,000 \times 24 \times 250 = 18,000,000$ in a single or multiple tables. Each row contains a symbol, time stamp, and 65 other columns.

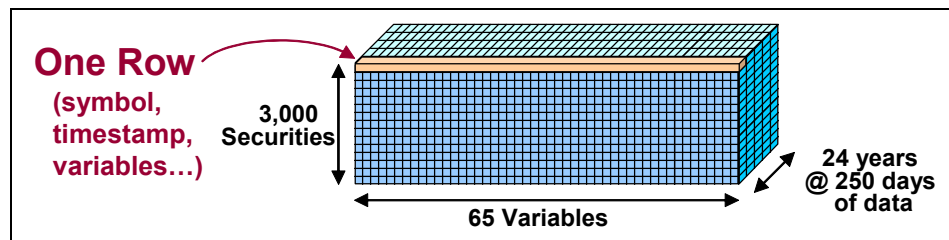


Figure 6-8 Volume of rows in a relational representation of share price time series

In contrast, assume that we have a TimeSeries data type, which the Line type of 6.3.1, “Line shapes” on page 244, can hold a data array of arbitrary length. Unlike the Line type with its fixed (X,Y) coordinate pair elements, it can handle array elements of any structure, as long as that structure has been defined as a row type. The table now looks something like the one represented by Example 6-20 and Table 6-15 on page 251.

Important: What follows is loosely based on the design and capabilities of the IBM Informix TimeSeries DataBlade product, but it only discusses the general principles, not the specific implementation of the DataBlade. Do not use this book as a reference for the product or draw conclusions from this discussion about its behavior, capabilities, or implementation.

Example 6-20 Table for share prices using time series UDT

```
CREATE ROW TYPE DailyStats
(
    market_date DATE,
    high        MONEY,
    low         MONEY,
    close       MONEY,
    volume      INTEGER,
    ...
);
CREATE TABLE share_prices_ts
(
    symbol CHAR(6) PRIMARY KEY,
    history Timeseries( DailyStats )
);
```

Note: The notation `TimeSeries(DailyStats)` defines a time series of elements that are `DailyStats` row type objects. `TimeSeries` here is a *type constructor*, which is analogous to a collection type constructor such as `SET(<row-type>)`. IDS supports the creation of type-constructor opaque types, and they have been used by IBM to create DataBlade products. However, they are not described in public documentation nor do we discuss them further in this book.

Table 6-15 Sample contents of table `share_prices_ts`

symbol	history
IBM	{{(07/02/2007,....,\$105.01,...),(07/03/2007,....,\$106.58,...),(07/05/2007,....,\$108.05,...), ...
ORCL	{{(07/02/2007,....,\$21.65,...),(07/03/2007,....,\$21.63,...),(07/05/2007,....,\$21.77,...), ...
...	...

Now the data quantity, in terms of rows, looks like a two-dimensional stack as in Figure 6-9, rather than the three-dimensional volume of Figure 6-8 on page 250. Here the total number of rows is 3,000. Each row contains two columns. Symbol and time stamp are not repeated for each element in the time series array.

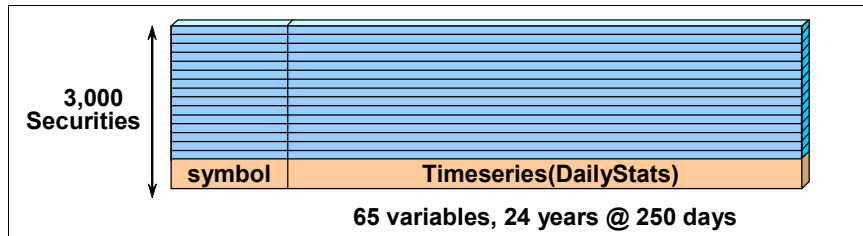


Figure 6-9 Stack of rows in UDT-based representation of share price time series

This two-column, denormalized structure embodies several optimizations:

- ▶ Because there is only one row for each tracked source (stock symbol, in the example), each symbol is only recorded once, not in every one of the 6,000 rows that comprise its time series.
- ▶ If the time series is regular, we can also omit the time stamp from the time series elements. We only need external mapping (a *calendar*) between date and array index. This not only takes care of matching the start of the time series to a specific date, but can account for regular gaps, such as weekends, and exceptions, such as holidays and other non-trading days. The calendar can be shared among all the time series.

In the normalized, relational case, rows that make up a time series are spread in unpredictable fashion over the disk. Unless there is a mechanism to keep the table clustered as it grows, we cannot expect the rows that make up a single time series to be stored close together or in order. (In a way, Figure 6-8 on page 250 is misleading: rows are not stored in a three-dimensional volume, but end-to-end in pages on disk.)

With a TimeSeries UDT, however, we can implement separate, highly optimized storage for these variable-length arrays. For example, using a B-tree as the internal structure governing the pages for each time series object. This not only optimizes access to all the elements of a given time series, but also makes retrieval of the information for a given date or range of dates highly efficient.

As mentioned previously, this use of extensibility primarily addresses performance. But what about the problem modeling aspect? Clearly, if you want to analyze individual time series with specialized statistical and signal processing algorithms, the time series array is a natural data structure for your application, and retrieving individual time series or time-bracketed subsections of them is easy in this implementation. But queries that cut across time series, as in the following example, are more difficult to express:

Which stock had the highest volume relative to its average volume on November 1?

Moreover, an off-the-shelf analysis package, such as a spreadsheet program, might not understand the custom UDT. This is where some of the advanced extensibility facilities that set IDS apart from the rest come in. Using the *Virtual Table Interface* (VTI), discussed in Chapter 10, “The world is relational” on page 401, we can make a table that is implemented as shown in Example 6-20 look like the table in Example 6-19. This gives us the following benefits:

- ▶ The performance advantages of denormalized, specialized array storage
- ▶ Array-object manipulation for applications that can handle them
- ▶ Standard-client access through a relational, normalized virtual-table view
- ▶ Flexible semantics for non-array-based queries through a virtual table

In the next section, we briefly discuss how the time series idea can be generalized to other areas.

6.3.3 Arrays

An obvious generalization of the time series idea is to allow storage and manipulation of arrays and matrices (multidimensional arrays), not just time-indexed ones. Many statistical and measured-data processing applications can benefit from these types. All considerations of 6.3.2, “Time series” on page 248, apply.

IDS already has a collection type, LIST, that can manage ordered sets of arbitrary element structures. It can even manage lists of elements that are themselves LIST objects, which would simulate a multidimensional array. However, the collection types were implemented as a simple extension to SQL to aid the management of small collections of manually entered values, not arbitrarily large arrays of automatically collected information. In addition, their semantics are limited and not tailored to specific applications such as statistical analysis.

Therefore, if you want a high-performance, scalable solution that supports matrices as well and allows you to add functions that specifically support your application, an opaque UDT-based implementation is the right choice. IBM used this approach to create data types for vectors and matrices whose elements are double-precision numbers for the IBM Informix NAG DataBlade module (implementing a subset of the Numerical Algorithms Group subroutine library; withdrawn as an IBM product in 2005).

6.4 Business logic where you need it

In the previous sections, we show that extensibility brings modern component techniques to the data server, helps match the data server’s data types and their semantics to the problem domain, and can improve performance through denormalization. In this section, we briefly highlight more benefits of extensibility that can affect the performance, ease of development, consistency, and maintainability, in short the robustness, of software solutions.

6.4.1 Integration: Doing multiple customizations

Many application-level solutions exist for specific problem domains, and many of these solutions require special services from the data management layer. Commonly, they have been implemented through special-purpose middleware, which translates the domain-specific needs (sometimes in the form of a domain-specific query language that looks like a superset of standard SQL) of the application to the generic capabilities (read, standard SQL) of the data server.

Such solutions exist for spatial information, text search, XML handling, digital media content, and a host of more specialized areas. To the extent that they are well implemented and do not violate best practices for data management, they can work quite well. For a specific client, the data access translation layer can hide the specifics of the underlying data server, making the application more database-independent. Alternatively, because the client interface to this layer is usually a product-specific API and not more-or-less standard SQL, it is more difficult for additional client programs to take advantage of the its capabilities.

Real problems occur, however, when a solution requires queries that involve more than one of these unusual types of information. Consider a simple query for a location-based service, for example, in which we search for restaurants that serve a specific dish and are located in a given search area. Let us say that our restaurants table has the columns `menu`, which contains XML or PDF documents that can be searched through a text index, and `location`, a spatial point whose location with respect to a search area can be checked through spatial predicates. In this case, the query looks as shown in Example 6-21.

Example 6-21 A simple location-based service query

```
SELECT name, address, phone FROM restaurants r
WHERE
  Within( r.location, Circle( :my_location, 5000 )) AND
  Contains( r.menu, 'Peking Duck' );
```

Assume that the table contains a million restaurants, 5000 serving Peking Duck, and 100 that are in a 5 km-radius search area. Only two restaurants both serve Peking Duck and are in the search area. Ideally, the optimizer has enough information to determine that the spatial `Within` predicate is more selective, runs an index scan on the spatial index, and applies the text `Contains` predicate as a filter on the 100 rows in the intermediate result set.

The data server performs the least possible amount of work and returns only the desired results to the client in response to a single query. Even if the situation is not ideal, performance suffers only a little. Most text search implementations require an index, in which case, it is not possible to apply the `Contains` predicate as a filter. The text index must be used for the primary index scan, and the `Within` predicate will be applied as a filter. In our example, this means that the intermediate result set has 5000 rows, not 100. Still, the data server can handle this sort of situation efficiently and only returns the two-row final result to the application.

If we must execute this query through two distinct middleware processes, our code becomes complicated and inefficient, as illustrated in Figure 6-10. The circled numbers in the figure correspond to the steps described in the text that follows.

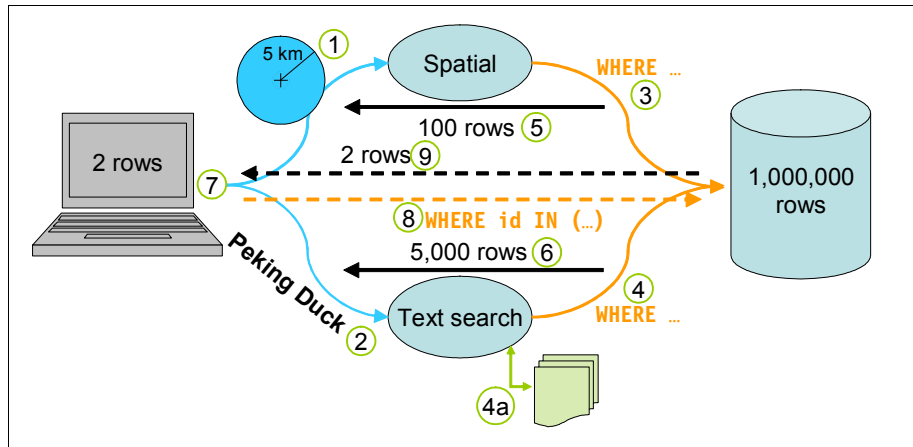


Figure 6-10 Multiple-middleware query

The application must formulate two separate middleware requests and send one (1) to the spatial and the other (2) to the text search middleware. Each middleware process submits its own translated query to the database (3, 4), some of which might involve access to local, proprietary index files (4a), and retrieves the rows found (5, 6). The application receives the results returned by the middleware processes and merges them into an intersection (7), keeping only those rows that occur in both result sets. To reduce network load, the application might request only primary key results on the first pass and issue a third query (8) to retrieve the full row contents for the final result set (9).

The case for an extensible data server is obvious: improved performance, reduced network bandwidth requirements, and vastly simplified application architecture.

6.4.2 Consistency: Deploying once, supporting all applications

Another aspect of enterprise solution architecture that has traditionally favored implementing business logic in the database is that it is the one level in the software stack that is shared among all applications. Thus, any logic implemented there is automatically applied to all processes and transactions and does not depend on tightly controlled deployment of software modules and their updates to all client platforms.

With the current preference for *service-oriented architecture* (SOA), it is tempting to conclude that the need for application logic in the data server has gone away. After all, instead of relying on client-side software (with all its deployment headaches) for the implementation of various bits of business logic, we invoke a *service* somewhere on the enterprise service bus (ESB). But services in SOA are loosely coupled, high-level-of-abstraction computing units. Their flexibility comes at a cost in performance that can be prohibitive for certain types of logic, in particular those that require accessing or selecting from large collection of data, or combining data from many different tables.

Thus, the need for a data server platform with the ability to support any kind of business logic never goes away. And extensibility, of course, is the key ingredient that makes this ability truly powerful and universally flexible.

6.4.3 Resiliency: Responding to changing requirements

DataBlade technology supports customization to environments and requirements that vary from site to site, as does this kind of component-based extensibility, which allows us to respond to changes over time. In both cases, it is difficult or impossible to come up with a design that applies to all places and for all time. Therefore, it is an important characteristic that DataBlade components are relatively small, self-contained software packages, implemented by relatively small teams working on relatively small projects over a relatively short time. Even for major products, such as the TimeSeries and Geodetic DataBlades, this is true when compared to the development effort that goes into the data server as a whole.

Moreover, DataBlade development and integration are based on a well defined and documented set of interfaces, SQL statements, and library functions. Their code and installation scripts can be kept completely separate from the IDS product itself. This not only means that partners and users can create their own DataBlades without worrying that they might stop working or need to be rebuilt with every new release of IDS. It also means that the DataBlades themselves can go through rapid evolution without requiring changes in IDS.

In short, by enabling the development of small, manageable, independent components, IDS extensibility lets in-house and independent software developers, as well as IBM itself, innovate quickly in response to changing times.

6.4.4 Efficiency: Bringing the logic to the data

One of the most obvious performance gains from implementing business logic in the data server is that it eliminates the need to retrieve large amounts of data when the end result is not those data records themselves but some derived,

much smaller result. A simple example of this is an aggregate such as an average. That is, if the desired result is an average column value over a group of rows, it is much more efficient to let the data server perform that computation than to send the entire group of rows over the network to the client so that it can compute the average.

Stored procedures written in SPL have long been used for this kind of server-side processing. However, SPL is limited as an algorithmic language, and performance for computationally intensive steps is not as good as in a mainstream procedural language such as C. Moreover, the set of standard SQL data types is limiting for many problem domains. Often, this has forced processing outside the database. Extensibility simply means that the limitations no longer apply. Business logic can now be implemented in the software tier where it makes the most sense, not where the limitations of traditional SQL and SPL force it.

6.5 Dealing with non-traditional data

A strong case for the object-relational extensibility movement in the 1990s was the need to manage data that did not fit the traditional model for relational databases, of business data that was mostly entered by hand, through forms and the like. An explosion in environmental and other scientific data collection, GPS devices, media (text documents, pictures, sound, and video), and other sources all pointed to the need for databases to evolve to deal not only with the new data structures, but also with the dramatic increase in sheer data volume that these automatically recorded data represent. In this section, we briefly discuss a few of the challenges and the tools that IDS provides to meet those challenges.

6.5.1 Virtual Table Interface and Virtual Index Interface

Some types of data inherently do not fit the relational model, need to be managed in a denormalized fashion, nor live in devices outside the control of the data server. Alternatively, relational access of normalized table structures through the unified interface of standard SQL is of huge value to application development productivity and architectural robustness.

Enter the *Virtual Table Interface*, a specification and interface that gives developers the power to make anything look like a regular table, including such diverse objects as the following types:

- ▶ A hydrological stream gauge
- ▶ An operating system's file system
- ▶ An in-memory structure for ingesting high-volume, real-time data streams
- ▶ Another table that contains a time series column

In some cases, it can be regarded as a database view with added powers. In other cases, it makes accessible through SQL what cannot normally be mapped to a relational schema.

Nearly identical to the VTI in interface and internal mechanism is the *Virtual Index Interface*, which has a different purpose. With the VII, developers can create new indexing methods to manage those new types of data for which the traditional B-tree index is inadequate. That is, what VTI does for *primary access methods* (tables), VII does for *secondary access methods*.

Like UDTs and UDRs, user-defined access methods are an important tool in the solution developer's toolbox. They convey the same benefits of performance, simplicity, flexibility, and so on to the overall solution. Many database products support views, database federation, and external data sources, which to one degree or another do some of what VTI can do. Likewise, many database products support the extension of the B-tree index to other types of data, as long as that data type can be mapped to a linear value domain. Only IDS, however, has VTI and VII, which like opaque data types, represent the extra level of power and performance that are required to turn the underlying concept from an interesting research topic to a practical mechanism for robust solutions. We discuss VTI and VII further in Chapter 10, "The world is relational" on page 401.

6.5.2 Real-time data

Because recorded-data volumes quickly dwarf anything that has ever been stored in databases, a new consideration emerges. It involves inserting, indexing, and making available large amounts of data immediately, as it is generated and time-stamped with the real-world clock time. This often means that traditional transaction processing, with log records, rollback capability, and locks, might be too expensive or even inappropriate. If a delay occurs, it is generally impossible to catch up. Instead, new techniques are needed that guarantee that the data server keep up with the incoming data stream, possibly sacrificing the comfortable integrity guarantees of old-style OLTP. We present two examples in the following sections.

Stock trades

In the previous time series example, we kept track of daily market data, which is a cumulative measure of all the individual trades that go on during a day. But what if we want to record and analyze all those individual transactions, as soon as they come in on one of the proprietary financial-data feeds?

It was for just this application that the Real-Time Loader DataBlade, which works specifically with the TimeSeries DataBlade, was developed. It employs tricks such as shared-memory storage of incoming data, making that in-memory

structure immediately available for queries through a VTI-based primary access method, and flushing accumulated data into TimeSeries values in regular tables, batching up thousands of individual updates into a single transaction. The Real-Time Loader allows IDS to process and keep up with hundreds of thousands of individual stock trade events per second.

GPS-enabled devices

Cars and trucks, PDAs and mobile phones, jets and missiles are pretty much all moving and movable things that now locate themselves by using a global positioning system (GPS) and report their changing position as they move.

Imagine tracking the millions of subscribers, updated every few seconds, for a mobile phone service provider. In many ways, it is exactly the same problem as that of the stock trades of the previous section, and the same tricks apply. We can keep up with hundreds of thousands of moving objects whose position is updated every second. This includes not only updating each object's position as the new coordinates come in, but querying the new configuration for spatial relationships to detect conditions such as objects on a collision course, objects of a specific description being closer than a specified distance, or objects entering or leaving a specified area.

6.5.3 Emerging standards

One of the characteristics of non-traditional types of data is that the standards that govern them and make them usable across systems are either nonexistent or just emerging. This means that we must be prepared over time to change our interfaces, add or rename functions, support new or changing external representations, and generally let our design evolve.

As stated earlier, it is easier to do so when the implementation is in a separate component than it would be as an integral part of the data server. This not only applies to the developer of the component but also to the user. If properly implemented, with a complete registration script, and perhaps version-stamped type data structures to allow for changes while maintaining backward compatibility, it is easier to upgrade a DataBlade than the entire server.

6.5.4 A word of caution

Because this chapter is about data server extensibility and its benefits, perhaps the enthusiasm it conveys leads you to think that DataBlades are the answer to almost every challenge. Certainly, many of us who were introduced to this technology and building DataBlades in the early days were guilty of this unwarranted exuberance. Of course, no technological breakthrough is the

solution to all problems. Instead, its proper place is as just another tool in the IT practitioner's toolbox. It is one that enhances our options in choosing the right place at which to implement business logic, keeping in mind considerations such as performance, maintainability, consistency, and resiliency. In this chapter, we have attempted to sketch some of the considerations that favor the decision to use UDTs, UDRs, and the other elements that make up DataBlade technology. Here are a few final words that may temper the tendency to dive headlong into an all-DataBlades approach.

First and foremost, a data server is about managing data. Apparently tautological, this adage is worth keeping in mind in the context of IDS extensibility, which effectively makes the data server a platform that we can program to perform any task we want. But an application server is a platform with similar programmability and integration capabilities. Therefore, we need a few guidelines to decide where to put the programmed logic.

Another way of expressing the rule from the previous paragraph is that the data server's first priority is to keep your data safe and preserve its integrity, and then to let you find the data you are looking for with speed and precision. This has strong implications for deciding what is appropriate for a DataBlade. For example, a spatial data type, with spatial predicate functions (for example, `Within`) supported by a spatial index, can help quickly find rows of data by spatial criteria. Alternatively, we can implement a DataBlade that adds sophisticated image processing functionality (Fourier Transforms, filters, multispectral classification) to the data server, but why burden a data management program with this kind of highly specialized, CPU-intensive computational load?

One question is used to separate the promising ideas from the merely curious and bad ones, when the novelty of the DataBlade idea prompted many to propose DataBlades for everything: "Does it go in the `WHERE` clause?" While not every function that is not an index-supported predicate is useless, it forces us to think a little harder before proposing an extension that does not come with index-supported query capability.

Another way to look at it is to think of the *Model-View-Controller* (MVC) style of application architecture. In MVC, the Model is often understood to be embodied in the database and its schema. In most of the examples in the first part of this chapter, the enhanced modeling power that a well-chosen data type can provide is one of the main arguments. When a particular function seems too much like a View or Controller feature, perhaps it does not belong in the DataBlade.

Naturally, there will always be gray areas. Sometimes it is convenient to find the same functionality in the data tier as in other tiers of the software stack. But in

prioritizing what should, rather than what could, be implemented, a good guideline is to look for elements that offer the following support:

- ▶ Enhance the fit between the data model and the problem being modeled
- ▶ Support fast queries (predicate functions, index support)
- ▶ Reduce the volume of data returned (aggregates, summary properties)
- ▶ Help implement triggers and stored procedures for automated business logic

With this, the case for extensibility should be clear. The rest of this book looks at specific applications of extensibility and the details of some of the facilities and tools IDS that provides.



Easing into extensibility

In this chapter, we describe how to get started with extensibility in the Informix Dynamic Server (IDS). We demonstrate simple examples on how to manipulate dates and discuss how to create user-defined records (UDRs) using C, SPL, or Java. Finally, we discuss the DataBlade modules that are available from IBM or downloadable from other sites.

7.1 Manipulating dates

IDS includes two date data types: DATE and DATETIME. DATE represents a particular day, and DATETIME represents a specific moment with a precision in the range of year to fraction of a second. IDS provides the following functions to manipulate these types:

- ▶ DATE(VARCHAR(10)) returning DATE
This function takes a character string in the format specified by the environment variable DBDATE and returns a DATE type. The default format for the US English locale is "MDY4".
- ▶ DATE(DATETIME) returning DATE
This is the same function as the previous one, but takes as input a DATETIME of any precision.
- ▶ DATE(INTEGER) returning DATE
The INTEGER argument represents a number of days since 31 December 1899.
- ▶ DAY(DATE) returning INTEGER
The DAY function returns the day of the month as an INTEGER.
- ▶ DAY(DATETIME)
This is the same function as the previous one, but takes as input a DATETIME of any precision.
- ▶ EXTEND(DATE, precision) returning DATETIME
The EXTEND function adjusts the precision of the DATE argument and returns the appropriate DATETIME. Since this is a little vague, here is an example:
`EXTEND(DATE(1), YEAR TO SECOND)`
- ▶ EXTEND(DATETIME, precision) returning DATETIME
This is the same function as the previous one, but operates on a DATETIME instead of a DATE.
- ▶ MONTH(DATE) returning INTEGER
MONTH extracts the month number from the DATE specified as an argument.
- ▶ MONTH(DATETIME) returning INTEGER
This function extracts the month from a DATETIME of any precision.

- ▶ WEEKDAY(DATE) returning INTEGER
The WEEKDAY function returns an INTEGER that represents the day of the week for the specified DATE. It starts with zero representing Sunday and goes on up to six representing Saturday.
- ▶ WEEKDAY(DATETIME) returning INTEGER
This is the same function as the previous one, but operates on a DATETIME.
- ▶ YEAR(DATE) returning INTEGER
This function extracts the year from the DATE specified as an argument.
- ▶ YEAR(DATETIME) returning INTEGER
This is the same function as the previous one, but operates on a DATETIME.
- ▶ MDY(INTEGER, INTEGER, INTEGER) returning DATE
This function creates a DATE based on the three INTEGER arguments. These arguments specify the month, day, and year, respectively. Note that the year is a four-digit integer.
- ▶ TO_CHAR(DATE, VARCHAR(?)) returning VARCHAR(?)
This function takes a DATE and a format argument and returns a character string representing the date formatted as requested. The format string can include the following types:
 - %A Weekday name
 - %B Month name
 - %d Day of the month as a decimal number
 - %Y Year as a 4-digit number
 - %R Time in 24-hour notation
- ▶ TO_CHAR (DATETIME, VARCHAR(?)) returning VARCHAR(?)
Same as previous example.
- ▶ TO_DATE(VARCHAR(?), VARCHAR(?)) returning DATE
This is the reverse operation of TO_CHAR and uses the same format string as the second argument.
- ▶ LAST_DAY(DATE)
This function takes a DATE and returns the last day of the month that its argument specifies.
- ▶ LAST_DAY(DATETIME)
This one is similar to the previous function, but it operates on a DATETIME argument.

- ▶ `ADD_MONTHS(DATE, INTEGER)` returning `DATE`
 This function takes a `DATE` and a `INTEGER` argument that can be used to specify the number of months to be added or subtracted from the `DATE` provided. The day and month time units in the first argument might specify the last day of the month, or the resulting month might have fewer days than the day in the first argument. In either case, the returned value is the last day of the resulting month. Otherwise, it is the same day of the month.
- ▶ `ADD_MONTHS(DATETIME, INTEGER)` returning `DATETIME`
 This is the same as the previous function, but it operates on a `DATETIME` value instead of a `DATE`.
- ▶ `NEXT_DAY(DATE, VARCHAR(??))` returning `DATE`
 This function takes a `DATE` and a quoted string that represents the English name for the day of the week and returns the earliest calendar date that is later than the date specified and falls on the day of the week specified. Valid abbreviations are 'SUN', 'MON', 'TUE', 'WED', 'THU', 'FRI', and 'SAT'.
- ▶ `NEXT_DAY(DATETIME, VARCHAR(??))` returning `DATETIME`
 This has the same function as the previous one, but it takes a `DATETIME` argument. The value returned is a `DATETIME YEAR TO FRACTION(5)`.
- ▶ `MONTHS_BETWEEN(DATE, DATE)` returning `DECIMAL`
 This function takes two `DATE` arguments and computes a `DECIMAL` value that represents the number of months between them based on a 31-day calendar. If the two arguments have the same day or fall on the last days of their respective months, a whole number is returned.
- ▶ `MONTHS_BETWEEN(DATETIME, DATETIME)` returning `DECIMAL`
 This is the same as the previous function, but it operates on two `DATETIME` values. The *hour*, *minute*, and *second* time units are also included in the fractional part of the calculation.
- ▶ `DBINFO('UTC_CURRENT')` returning `INTEGER`
 This variant of the `DBINFO` function returns the current time expressed as the number of seconds since 1 January 1970. The time returned corresponds to the current time zone setting in your environment.
- ▶ `DBINFO('UTC_TO_DATETIME', INTEGER)` returning `DATETIME`
 This variant of the `DBINFO` function returns the `DATETIME` representation (year to second) of a value that is generated by `DBINFO('UTC_CURRENT')`.

In addition to these functions, the following four environment variables impact the processing of dates:

▶ **DBDATE**

This variable provides the end-user format of a date. It is described in the *IBM Informix Guide to SQL: Reference*, G251-2283.

▶ **DBCENTURY**

This variable defines how to expand the year when a date is entered as a two-digit number instead of four. The acceptable values are R, P, F, and C. They stand for Current, Previous, Future, and Closest, respectively. The value R is the default if DBCENTURY is not set. DBCENTURY is described in the *IBM Informix Guide to SQL: Reference*, G251-2283.

▶ **GL_DATE**

This variable provides for the ability to internationalize DATE output. It is described in the *IBM Informix GLS User's Guide*, G229-6373.

▶ **GL_DATETIME**

This variable provides for the ability to internationalize DATETIME output. It is described in the *IBM Informix GLS User's Guide*, G229-6373.

Finally, IDS defines two built-in functions that return the current date value. CURRENT returns a DATETIME value, and TODAY returns today's date.

7.1.1 The date functions

The functions described in 7.1, “Manipulating dates” on page 264, provide functionality for input, output, formatting, and information extraction. The first interesting usage to be discussed is the input of dates based on character strings.

The function DATE() receives a character string as input, but processes it differently depending on the setting of DBDATE and DBCENTURY. Let us start with DBCENTURY.

The default value for DBCENTURY is R. This means that the century is determined by the century of the current date. Example 7-1 assumes the default US English locale.

Example 7-1 DATE function with DBCENTURY = R

```
SELECT DATE("9/2/92") FROM systables WHERE tabid =1;
```

(constant)

09/02/2092

1 row(s) retrieved.

If DBCENTURY is set to P, the inferred century is the one that is closest to the current date as described in Example 7-2.

Example 7-2 DATE function with DBCENTURY = P

```
SELECT DATE("9/2/92") FROM systables WHERE tabid = 1;
```

(constant)

09/02/1992

1 row(s) retrieved.

The other possible variation on date conversion is provided by the DBDATE environment variable. For the US English locale, it defaults to "MDY4/". This means that the elements of a date string are separated by a forward slash (/) and their order is month, day, and year. Note that the year is expected to be a four-digit year, but can be completed according to the rules set by DBCENTURY. You can change the value of DBDATE to use an international date format.

The DBDATE value is "Y4MD-" as shown in Example 7-3 on page 269. In addition to affecting the input of dates as character strings, this value also affects how dates are converted back into character strings.

Example 7-3 DBDATE display

```
SELECT order_date FROM orders WHERE order_num = 1001;
```

```
order_date
```

```
2007-05-20
```

```
1 row(s) retrieved.
```

For a more elaborate date display, you can use the TO_CHAR function and provide a format as previously described and shown in Example 7-4.

Example 7-4 TO_CHAR example invocation

```
SELECT TO_CHAR(order_date, “%d %B %Y”)
FROM orders
WHERE order_num = 1001;
```

```
(expression) 20 May 1998
```

```
1 row(s) retrieved.
```

You can use some of the provided functions to extract values, such as the month and the day, and use those values when defining table fragmentation by expression. You can also use them for grouping in SQL statements. For example, to determine how many orders you have per month, you can use the statement shown in Example 7-5.

Example 7-5 Extracting components from dates

```
SELECT YEAR(order_date) year, MONTH(order_date) month, COUNT(*) count
FROM orders
GROUP BY 1,2
ORDER BY 1,2;
```

This type of grouping can be useful in all sorts of reporting. You can do much more if you are willing to take advantage of the basic extensibility features of IDS.

7.1.2 Functional indexes

IDS V9.x and higher support the concept of a functional index. This means that you can create an index on the result of a function. Then you can use that index to speed up the processing of the queries that include the function in their SQL statements.

The built-in functions were created before IDS added extensibility features. Therefore, you cannot create an index on the result of a built-in function. However, to make it work, you can wrap the built-in function in an SPL function. For example, to create an index on the month, you can create an SPL function such as the one that is shown in Example 7-6.

Example 7-6 Functional index SPL function

```
CREATE FUNCTION udr_month(dt DATE)
RETURNING INTEGER
WITH(NOT VARIANT)
RETURN month(dt);
END FUNCTION;
```

With this wrapper function, you can create an index such as the one shown in Example 7-7.

Example 7-7 Creating a functional index

```
CREATE INDEX orders_months_ids ON orders(udr_month(order_date));
```

Then you can use an SQL statement such as the one shown in Example 7-8 that takes advantage of the index.

Example 7-8 Select statement that uses a functional index

```
SELECT * FROM orders WHERE udr_month(order_date) = 6;
```

7.1.3 Creating new date functions

You can extract additional information from a date, such as the week of the year, the week of the month, and the quarter. Let us start with the day of the year function. Having such a function allows you to report activities by week, without having to write specific stored procedures for each report or to write custom application code. You can build this function by using the built-in functions that are included in IDS. This makes it surprisingly simple, such as the one shown in Example 7-9.

Example 7-9 Day_of_year SPL implementation

```
CREATE FUNCTION day_of_year(dt DATE)
RETURNS INTEGER
WITH(NOT VARIANT)
RETURN(1+dt-MDY(1,1, YEAR(dt)));
END FUNCTION;
```

The key to the implementation of this function is to know that a date is actually an integer that represents the number of days since 31 December 1899. This means that if we have the date for January 1, it becomes a simple subtraction.

You can also use the function in an EXECUTE FUNCTION statement, setting a value in a function or stored procedure. Or you can use it in an SQL statement, such as the one shown in Example 7-10.

Example 7-10 Using day_of_year in a SELECT statement

```
SELECT order_date, day_of_year(order_date) d_o_y
FROM orders
WHERE order_num = 1001;
```

```
order_date    d_o_y
```

```
05/20/1998    140
```

```
1 row(s) retrieved.
```

The week of the year function is slightly trickier. It is a similar calculation, except that we must divide by seven days per week, as shown in Example 7-11.

Example 7-11 Week_of_year SPL implementation

```
CREATE FUNCTION week_of_year(dt DATE)
RETURNS INTEGER
WITH(NOT VARIANT)
```

```
DEFINE day1 DATE;
DEFINE nbdays INTEGER;
LET day1 = MDY(1,1,YEAR(dt));
LET nbdays = dt - day1;
RETURN 1+(nbdays+WEEKDAY(day1))/7;
END FUNCTION;
```

The key to this function is to understand the offset provided by the WEEKDAY built-in function. The WEEKDAY function returns zero for Sunday up to six for Saturday. If January 1 is a Sunday, then we know that January 8 is the following Sunday, week 2. If January 1 starts on any other day, this means that the first week is shorter. The WEEKDAY built-in function gives us that offset that allows us to determine the week of the year.

The week_of_year() function has a problem with the last week of one year and the first week of the next year. For example, 31 December 2004 was a Friday and

1 January 2005 was a Saturday. The week_of_year function shown in Example 7-12 demonstrates that.

Example 7-12 Executing the week_of_year function

```
EXECUTE FUNCTION week_of_year(DATE("12/31/2004"));
```

```
(expression)
```

```
53
```

```
1 row(s) retrieved.
```

```
EXECUTE FUNCTION week_of_year(DATE("1/1/2005"));
```

```
(expression)
```

```
1
```

```
1 row(s) retrieved.
```

This problem has been addressed in the ISO 8601 standard. Jonathan Leffler, of IBM, has written stored procedures that implement the standard. His stored procedure can be found on the International Informix Users Group Web site at the following address:

<http://www.iug.org>

Example 7-13 demonstrates the week of the year implementation converted to a user-defined function (UDF).

Example 7-13 Week of the Year implementation

```
CREATE FUNCTION day_one_week_one(year_param INTEGER)
RETURNING DATE
WITH (NOT VARIANT)
  DEFINE jan1 DATE;
  LET jan1 = MDY(1, 1, year_param);
  RETURN jan1 + MOD(11 - WEEKDAY(jan1), 7) - 3;
END FUNCTION;
```

```
CREATE FUNCTION iso8601_weeknum(dateval DATE DEFAULT TODAY)
RETURNING CHAR(8)
WITH (NOT VARIANT)
  DEFINE rv CHAR(8);
  DEFINE yyyy CHAR(4);
  DEFINE ww CHAR(2);
```

```

DEFINE d1w1 DATE;
DEFINE tv DATE;
DEFINE wn INTEGER;
DEFINE yn INTEGER;
-- Calculate year and week number.
LET yn = YEAR(dateval);
LET d1w1 = day_one_week_one(yn);
IF dateval < d1w1 THEN
-- Date is in early January and is in last week of prior year
    LET yn = yn - 1;
    LET d1w1 = day_one_week_one(yn);
ELSE
    LET tv = day_one_week_one(yn + 1);
    IF dateval >= tv THEN
-- Date is in late December and is in the first week of next year
        LET yn = yn + 1;
        LET d1w1 = tv;
    END IF;
END IF;
LET wn = TRUNC((dateval - d1w1) / 7) + 1;
-- Calculation complete: yn is year number and wn is week number.
-- Format result.
LET yyyy = yn;
IF wn < 10 THEN
    LET ww = "0" || wn;
ELSE
    LET ww = wn;
END IF
LET rv = yyyy || "-W" || ww;
RETURN rv;
END FUNCTION;

```

Jonathan also provides a standard-compliant procedure to calculate the day of the week. Example 7-14 shows the procedure converted to a UDF.

Example 7-14 ISO8601 weekday procedure

```

CREATE FUNCTION iso8601_weekday(dateval DATE DEFAULT TODAY)
RETURNING CHAR(10)
WITH(NOT VARIANT)
    DEFINE rv CHAR(10);
    DEFINE dw CHAR(4);
    LET dw = WEEKDAY(dateval);
    IF dw = 0 THEN
        LET dw = 7;

```

```
END IF;  
RETURN iso8601_weeknum(dateval) || "-" || dw;  
END FUNCTION;
```

Let us continue our discussion on date manipulation. We ignore the ISO 8601 standard to keep the functions simple. As you have previously seen, you can easily adapt the functions to be compliant with the standard.

To calculate the week of the month, use the same function, but instead of using January 1 as the starting date, use the first day of the month coming from the date passed as argument, as shown in Example 7-15.

Example 7-15 Week_of_month SPL implementation

```
CREATE FUNCTION week_of_month(dt DATE)  
RETURNS INTEGER  
WITH(NOT VARIANT)  
  
DEFINE day1 DATE;  
DEFINE nbdays INTEGER;  
LET day1 = MDY(MONTH(dt), 1, YEAR(dt));  
LET nbdays = dt - day1;  
RETURN 1 + (nbdays + WEEKDAY(day1))/7;  
END FUNCTION;
```

7.1.4 The quarter() function

Some database products provide a quarter() function. It typically returns a number between one and four. The problem with providing a quarter() function is that it assumes a specific calendar, the standard calendar year.

Many companies want to calculate the quarter based on their business year that is different from the calendar year. There are even some organizations that must calculate the quarter differently based on what needs to be done. For example, some school districts must calculate a calendar quarter, a school year quarter, and a business quarter.

Let us start with a simple implementation of a calendar year quarter, as demonstrated in Example 7-16.

Example 7-16 Quarter function implementation

```
CREATE FUNCTION quarter(dt DATE)
RETURNS INTEGER
WITH(NOT VARIANT)
RETURN (YEAR(dt) * 100) + 1 + (MONTH(dt) - 1)/3;
END FUNCTION;
```

In this implementation, the year is included as part of the quarter. For example, the third quarter of 2005 is represented by 200503. This simplifies the processing when an SQL statement spans more than one year. You can decide to create a different implementation, such as returning a character string instead of an integer. This is determined by your particular requirements.

As previously mentioned, you might want to calculate a quarter based on a starting date that is not January 1. This adds some complication where the calendar year might be different from the quarter year. For example, consider a corporation that starts its fiscal year on September 1. This means that 1 September 2005 is really the start of the first quarter 2006, December 1 is the start of the second quarter, and so on.

Example 7-17 shows the implementation for a year starting on September 1. It is easy to adapt this code for any starting date.

Example 7-17 Another example of a quarter function implementation

```
CREATE FUNCTION bizquarter(dt DATE)
RETURNS INTEGER
WITH(NOT VARIANT)

DEFINE yr INTEGER;
DEFINE mm INTEGER;

LET yr = YEAR(dt);
LET mm = MONTH(dt)+4; -- sept to jan is 4 months
IF mm > 12 THEN
    LET yr = yr + 1;
    LET mm = mm - 12;
END IF
RETURN (yr * 100) + 1 + (mm - 1)/3;
END FUNCTION;
```

The additional processing in this function compared to the quarter() function is that it moves the current month forward a number of months in order to do the quarter calculation that matches the business year.

Using the new functions

Now that you have these functions available, you can use them in SQL statements as though they were built into IDS. Example 7-18 shows how you execute this function in a SELECT statement via DBAccess.

Example 7-18 Quarter function usage

```
SELECT quarter(order_date) quarter, COUNT(*) count
FROM orders
GROUP BY 1
ORDER BY 1;
```

quarter	count
199802	16
199803	7

2 row(s) retrieved.

In addition, you can create indexes on the functions as shown in Example 7-19.

Example 7-19 Functional index on week_of_year

```
CREATE INDEX orders_weeks_ids ON orders(week_of_year(order_date));
```

This gives you the flexibility to have the database return the information that you are looking for. IDS extensibility can be useful in many other areas.

It is easy to extend the capabilities of IDS to provide variations of date manipulation. The result is less code to write and potentially fewer SQL statements to execute, which leads to better performance. A database is not a commodity. It is a strategic tool that can give you a business advantage.

By using these date manipulation techniques, you can adapt IDS to fit your environment. If the date functions provided here do not exactly fit your environment, you can easily modify them. The flexibility provided by IDS means that the IDS capabilities should be considered early in the design phase of an application. The result can be greater performance and scalability, and a simpler implementation.

7.2 DataBlade API demystified

For those who want to develop their database extensions in the C programming language, IDS contains a comprehensive set of header files, public data type structures, and public functions via the DataBlade application programmer interface (API). Most of the API functions, header files, and data types are prefixed with *mi*. The following classes of functions are available in the API:

- ▶ Data handling
- ▶ Session, thread, and transaction management
- ▶ SQL statement processing
- ▶ Function execution
- ▶ Memory management
- ▶ Exception handling
- ▶ Smart large-object management
- ▶ Operating system file access
- ▶ Tracing

Data handling

With the data handling class of functions, you can obtain information about the data type with which you are working, manipulate it, convert it to a different data type, convert it to a different code set, or transfer it to another computer running IDS. Functions in this class are typically used in UDRs that work with user-defined types (UDTs), complex IDS types (such as sets, lists, and collections), as well as more common tasks such as handling NULL values and SERIAL values that have been retrieved from the server.

Session, thread, and transaction management

By using the session, thread, and transaction management class of functions, you can obtain database connection information, open a new database connection, manage thread and stack usage, and obtain information about database processing state transitions. Functions in this class are used in UDRs that must execute SQL statements, perform recursion or long running operations, or must detect events such as the end of a transaction or even the end of a database session.

SQL statement processing

With the SQL statement processing class of functions, you can send SQL statements to the database server for execution. This is a multistep process inside a UDR, as you must construct the environment wherein the query is executed and processed. To do this, the UDR must perform the following tasks:

- ▶ Assemble the SQL statement and send it to the IDS kernel for execution
- ▶ Process the results that the IDS kernel returns to the UDR

- ▶ If the SQL statement returns rows (such as a SELECT statement), obtain each row of data
- ▶ For each row that is returned, process each column
- ▶ Complete the execution of the statement

For more information about processing SQL statements, see *IBM Informix DataBlade API Programmer's Guide*, G229-6365.

Function execution

With the function execution class of functions, you can obtain information about, and execute, UDRs from your routine. A UDR can be invoked through an SQL statement, such as EXECUTE FUNCTION or SELECT, and its data retrieved as with any other SQL statement. If the UDR that is called resides in the same shared object library as the caller, then it can be invoked just as any other C function. If it resides in a different shared object library or DataBlade, then you must use the Fastpath interface to call the UDR. The Fastpath interface allows a UDR to directly invoke the UDR, bypassing the overhead associated with compiling, parsing and executing an SQL statement. There are other functions in this class, with which you can obtain information about trigger execution as well as HDR status information.

Memory management

Because UDRs execute inside the IDS server context, traditional operating system memory allocation functions, such as malloc(), calloc(), realloc() and free(), should not be used. The DataBlade API provides the memory management class of functions so that you can manage IDS server memory within the UDR, and allocate and free memory from the virtual pool in IDS. These functions are mi_alloc(), mi_dalloc(), mi_zalloc(), and mi_free().

Memory that is allocated and freed during UDR execution is performed in terms of durations. The default duration for memory during a UDR is PER_ROUTINE. That is, memory allocated during the execution of the UDR is deallocated at the end or when it is explicitly freed by the UDR. Table 7-1 describes the various memory durations that are available.

Table 7-1 Memory durations

Duration	Description
PER_ROUTINE	One UDR invocation
PER_COMMAND	Subquery
PER_STMT_EXEC	Current SQL statement
PER_STMT_PREP	Prepared SQL statement

Duration	Description
PER_TRANSACTION	Current client transaction (BEGIN WORK to COMMIT / ROLLBACK WORK)
PER_SESSION	Current client session
PER_SYSTEM	IDS system global, persists until IDS is shut down

This class of functions also includes the ability to create *named memory*. Named memory is memory allocated from IDS shared memory, but instead of accessing it purely by address, you can assign a name and a duration to the memory. This makes it easier to write functions that can share state across multiple invocations of UDRs and user sessions. There are also functions provided to lock and unlock memory to help with concurrency.

Important: Some objects, such as large objects and long varchars, provide special functions for allocating and deallocating memory that is associated with those objects. Consult the *IBM Informix DataBlade API Function Reference*, G229-6364, for more information about functions related to these objects to determine the method to allocate and deallocate memory.

Exception handling

With the exception handling class of functions, you can trap and manage events that occur within IDS. The most common events are errors and transaction events such as an instruction to rollback the current transaction. You can find examples of this type of function in Chapter 9, “Taking advantage of database events” on page 373.

Smart large-object management

By using the smart large-object management class of functions, you can create, access, modify, and delete smart large objects in IDS. A *smart large object* is a large object that can hold up to 4 terabytes (TB) of data in a single object that supports recoverability and random access to its data. This differs from the traditional simple large object (BYTE or TEXT) that provides access on a “all or nothing” basis.

Operating system file access

The operating system file access class of functions provides file management access and manipulation similar to what operating system file access functions provide. The difference is that the DataBlade API functions periodically yield the virtual processor to limit the effects of blocking I/O. Examples of these operations

are `mi_file_open()`, `mi_file_seek()`, `mi_file_read()`, `mi_file_write()`, and `mi_file_close()`.

Tracing

The tracing class of functions allows for the embedding and enablement of tracing messages during UDR run time. With this facility, you can create trace classes and define levels of trace messages that can be embedded in your UDR. You can also specify the file name to which the trace output is written. All IBM DataBlades include a special trace function to help diagnose problems that are related to that DataBlade.

When constructing a C UDR, remember to include the header file `mi.h`. This reference includes most of the common definitions and other header files that will be used in most UDRs. For more information about these functions, header files and data types, consult the following manuals in the IDS 11 Information Center at the following address:

<http://publib.boulder.ibm.com/infocenter/idshelp/v111/index.jsp?topic=/com.ibm.start.doc/welcome.htm>

- ▶ *IBM Informix DataBlade API Programmer's Guide, G229-6365*
- ▶ *IBM Informix DataBlade API Function Reference, G229-6364*

7.3 Java UDRs made easy

IDS supports Java as a language to write UDFs, aggregates, and opaque types. The API is tailored on the Open Database Connectivity (ODBC) interface, which makes it easy for Java programmers to learn to write extensions for IDS.

You should be aware of the limitations to Java UDRs, including those as follows:

- ▶ Commutator functions
A UDR can be defined as the same operation as another one with the arguments in reverse order. For example, `lessthan()` is a commutator function to `greaterthanorequal()` and vice versa.
- ▶ Cost functions
A cost function calculates the amount of resource a UDR requires based on the amount of system resources it uses.
- ▶ Operator class functions
These functions are used in the context of the Virtual Index Interface (VII).

- ▶ Selectivity functions
A selectivity function calculates the fraction of rows that qualify for a particular UDR that acts as a filter.
- ▶ User-defined statistics functions
These functions provide distribution statistics on an opaque type in a table column.
- ▶ User-defined table functions (Virtual Table Interface (VTI))
These functions are used in the context of creating virtual tables.

These types of functions must be written in C.

Before you start to use Java UDRs, you must configure IDS for Java support. In brief, you must ensure the following tasks:

- ▶ Include a default sbspace (SBSPACENAME onconfig parameter).
- ▶ Create a jvp.properties file in \$INFORMIXDIR/extend/krakatoa.
- ▶ Add or modify the Java parameters in your onconfig file.
- ▶ (Optional) Set some environment variables.

A Java UDR is implemented as a static method within a Java class. For example, to create a quarter() function in Java, see the sample code provided in Example 7-20.

Example 7-20 Java quarter function

```
import java.lang.*;
import java.sql.*;
import java.util.Calendar;

public class Util {
    public static String jquarter(Date my_date) {
        Calendar now = Calendar.getInstance();
        now.setTime(my_date);
        int month = now.get(Calendar.MONTH);
        int year = now.get(Calendar.YEAR);
        int q = month / 3;
        q++;
        String ret = year + "Q" + q;
        return(ret);
    }
}
```

In this implementation, the `jquarter()` method takes a date as argument and returns a character string in the format "yyyyQqq," where "yyyy" represents the year and "qq" represents the quarter number.

To compile the class, use the `javac` command from the Java Development Kit (JDK™). For `Util.java`, you can use the following command:

```
javac Util.java
```

If you are using Java Database Connectivity (JDBC) features or a special class, such as one that keeps track of state information in an iterator function, you must add two Java archive (JAR) files in your classpath. The following command illustrates their use:

javac -classpath

```
$INFORMIXDIR/extend/krakatoa/krakatoa.jar;$INFORMIXDIR/extend/krakatoa/jdbc.jar Util.java
```

After the Java code is compiled, you must put it in a JAR file with a deployment descriptor and a manifest file. The deployment descriptor allows you to include in the JAR file the SQL statements for creating and dropping the UDR. In this example, the deployment descriptor, possibly called `Util.txt`, can be as shown in Example 7-21.

Example 7-21 Deployment descriptor

```
SQLActions[] = {  
    "BEGIN INSTALL  
    CREATE FUNCTION jquarter(date)  
    RETURNING varchar(10)  
    WITH(parallelizable)  
    EXTERNAL NAME 'thisjar:Util.jquarter(java.sql.Date)'  
    LANGUAGE Java;  
    END INSTALL",  
  
    "BEGIN REMOVE  
    DROP FUNCTION jquarter(date);  
    END REMOVE"  
}
```

The manifest file, which we call `Util.mf`, can be as follows:

```
Manifest-Version: 1.0  
Name: Util.txt  
SQLJDeploymentDescriptor: TRUE
```

With these files, you can create the JAR file with the command:

```
jar cmf Util.mf Util.jar Util.class Util.txt
```

Before you can create the function, you must install the JAR file in the database. Identify the location of the JAR file and give it a name that is then used as part of the external name in the create statement:

```
EXECUTE PROCEDURE install_jar(  
    "file:$INFORMIXDIR/extend/jars/Util.jar", "util_jar");
```

The `install_jar` procedure takes a copy of the JAR file from the location given as the first argument and loads it into a smart binary large object (BLOB) stored in the default smart BLOB space defined in the `onconfig` configuration file. The JAR file is then referenced by the name given as the second argument.

The `CREATE FUNCTION` statement defines a function `jquarter` that takes a date as input and returns a `varchar(10)`. The modifier in the function indicated that this function can run in parallel if IDS decides to split the statement into multiple threads of execution.

The external name defines the JAR file where to find the class `Util`. This name is the one defined in the execution of the `install_jar` procedure. The class name is followed by the static function name and the fully qualified argument name.

At this point, you can use the `jquarter()` function in SQL statements or by itself in a statement such as the following example:

```
SELECT jquarter(order_date), SUM(amount)  
FROM transactions  
WHERE jquarter(order_date) LIKE "2005%"  
GROUP BY 1  
ORDER BY 1;
```

```
EXECUTE FUNCTION jquarter("09/02/2005");
```

For simple functions, such as `jquarter()`, it might be more desirable to use `SPL` or `C`, rather than `Java`. By its nature, `Java` requires more resources to run than `SPL` or `C`. However, this does not mean that it is a bad choice for extensibility.

If you do not have demanding performance requirements, it does not matter that you use `Java`. In some cases, the complexity of the processing in the function makes the call overhead insignificant. In other cases, the functionality provided in `Java` makes it a natural choice. It is much easier to communicate with outside processes or access the Web in `Java` than with any other extensibility interface. `Java` is a natural fit to access Web services. Since the UDR runs in a `Java` virtual machine (`JVM™`), you can also use any class library to do processing, such as

the parsing of XML documents. These libraries do not require any modifications to run in the IDS engine.

Choosing Java as the primary extensibility language does not exclude using either SPL or C, so do not hesitate to use Java for extensibility if you feel it is the right choice.

7.4 Development and deployment

For those of you who are new to IDS extensibility, or if a project requires more than a few types and functions, IDS provides the DataBlade Development Kit (DBDK) for the Windows platform. It is available for both 32-bit and 64-bit Windows environments.

DBDK is a graphical user interface (GUI) that includes the following parts:

- ▶ BladeSmith

BladeSmith helps you manage the project and assists in the creation of the functions based on the definition of the arguments and the return value. It also generates header files, makefiles, functional test files, SQL scripts, messages, and packaging files.

- ▶ BladePack

BladePack can create a simple directory tree that includes files to be installed. The resulting package can be registered easily in a database by using BladeManager. It assumes that you have created your project by using BladeSmith.

- ▶ BladeManager

BladeManager is a tool that is included with IDS on all platforms. It simplifies the registration, upgrade, and de-registration of DataBlades.

7.4.1 Building a C UDR

The first step in creating a C UDR is to write the code. For example, Example 7-22 shows the code that implements a `quarter()` function in C.

Example 7-22 Creating a “C” UDR

```
#include <mi.h>

mi_lvarchar *quarter(mi_date date, MI_FPARAM *fparam)
{
    mi_lvarchar *RetVal;          /* The return value. */
```

```

short      mdy[3];
mi_integer qt;
char       buffer[10];

/* Extract month, day, and year from the date */
ret = rjulmdy(date, mdy);
qt = (mdy[0] - 1) / 3; /* calculate the quarter */
qt++;
sprintf(buffer, "%4dQ%d", mdy[2], qt);
RetVal = mi_string_to_lvarchar(buffer);

/* Return the function's return value. */
return RetVal;
}

```

The first line includes a file that defines most of the functions and constants of the DataBlade API. Others include files that might be needed in some other cases. This include file is located in `$INFORMIXDIR/incl/public`.

The line that follows defines the function `quarter()` as taking a date as an argument and returning a character string (CHAR, VARCHAR, or LVARCHAR). Note that the DataBlade API defines a set of types to match the SQL types. The function also has an additional argument that can be used to detect whether the argument is NULL, as well as do other tasks.

The rest of the function is straightforward. We extract the month, day, and year from the date argument by using the ESQL/C function `rjulmdy()`, calculate the quarter, create a character representation of that quarter, and transform the result into an `mi_lvarchar` before returning it.

The next step is to compile the code and create a shared library. Assuming that the C source code is in a file called `quarter.c`, you can do this with the following steps:

```

cc -DMI_SERVBUILD -I$INFORMIXDIR/incl/public -c quarter.c
ld -G -o quarter.bld quarter.o
chmod a+x quarter.bld

```

The `cc` line defines a variable called `MI_SERVBUILD` because the DataBlade API was originally designed to be available inside the server and within application clients. This variable indicates that we are using it inside the server. We also define the location of the directory for the include file we are using.

The `ld` command created the shared library named `quarter.bld` and includes the object file `quarter.o`. The `.bld` extension is a convention that indicates that it is a

blade library. The last command, **chmod**, changes the file permission to make sure the library has the execution permission set.

Obviously these commands vary from platform to platform. To make it easier, IDS has a directory, `$INFORMIXDIR/incl/dbdk`, that includes files that can be used in makefiles. These files provide definitions for the compiler name, linker name, and the different options that can be used. These files are different depending on the platform in use. Example 7-23 shows a simple makefile to create `quarter.bld`.

Example 7-23 Makefile to create quarter.bld

```
include $(INFORMIXDIR)/incl/dbdk/makeinc.linux

MI_INCL = $(INFORMIXDIR)/incl
CFLAGS = -DMI_SERVBUILD $(CC_PIC) -I$(MI_INCL)/public $(COPTS)
LINKFLAGS = $(SHLIBFLAG) $(SYMFLAG)

all: quarter.bld

# Construct the object file.

quarter.o: quarter.c
    $(CC) $(CFLAGS) -o $@ -c $?

quarter.bld: quarter.o
    $(SHLIBLOD) $(LINKFLAGS) -o quarter.bld quarter.o
```

To use this makefile on another platform, you must change the include file name on the first line from `makeinc.linux` to `makeinc` with the suffix from another platform.

Important: Shared objects compiled in the Visual Studio® 8 (or later) environment on both 32- and 64-bit Windows platforms require that you embed the manifest for locating the Microsoft C runtime libraries.

We suggest that you install your shared library in a subdirectory under `$INFORMIXDIR/extend`. Assume that `quarter.bld` is under a subdirectory called `quarter`. Then the `quarter` function is created using the following statement:

```
CREATE FUNCTION quarter(date)
RETURNS varchar(10)
WITH (not variant, parallelizable)
external name "$INFORMIXDIR/extend/quarter/quarter.bld(quarter)"
LANGUAGE C;
```


You can use the environment variable `INFORMIXDIR` in the external name since this variable is defined in the server environment. It is then replaced by the content of the environment variable to provide the real path to the shared library. After the function is defined, it can be used in an SQL statement or called directly as shown previously.

If there is a need to remove the function from the database, it can be done with the following statement:

```
DROP FUNCTION quarter(date);
```

7.4.2 Installation and registration

All DataBlade modules follow a similar pattern for installation and registration. They are installed in the `$INFORMIXDIR/extend` directory. The built-in DataBlades are already installed in the server on their supported platforms. Any other DataBlade modules must first be installed in the data server. The installation process is described in detail in the manual *DataBlade Modules Installation and Registration Guide*, G251-2276-01. In general, the installation can include the following steps for DataBlade modules that are available separately from IDS:

1. Unload the files into a temporary directory. This might require the use of a utility, such as `cpio` or `tar`, or an extraction utility, depending on the platform that is used.
2. Execute the installation command. This command is usually called `install` on UNIX-type platforms (or `rpm` on Linux) and `setup` on Windows platforms.

Starting with DataBlade modules released in 2007, such as Spatial 8.21.xC1 and Geodetic 3.12.xC1, a single installation command can help simplify installation and provide new capabilities such as console, GUI and silent installation. This command is in the following format:

```
blade.major.minor.fixpack.platform.bin
```

Refer to the Quick Start guides that are provided with the DataBlade module for more information about how to access these options.

After the installation, there is a new directory under `$INFORMIXDIR/extend`. The name reflects the DataBlade module and its version. For example, the current version of the Spatial DataBlade module directory is named `spatial.8.21.UC1`.

You must register a DataBlade module into a database before it is available for use. The registration might create new types, UDFs, and tables and views. The registration process is made easy through the use of the DataBlade Manager

utility (**blademgr**). Example 7-24 shows the process for registering the Spatial DataBlade module in a database called *demo*.

Example 7-24 DataBlade registration

```
informix@ibmswg01:~> blademgr
IDS 11>list demo
There are no modules registered in database demo.
IDS 11>show modules
8 DataBlade modules installed on server IDS 11:
           ifxrltree.2.00           mqblade.2.0
           Node.2.0 c               LLD.1.20.UC2
bts.1.00           ifxbuiltins.1.1
           binaryudt.1.0           spatial.8.21.UC1
wfs.1.00.UC1
A 'c' indicates the DataBlade module has client files.
If a module does not show up, check the prepare log.4
IDS 11>register spatial.8.21.UC1 demo
Register module spatial.8.21.UC1 into database demo? [Y/n]
Registering DataBlade module... (may take a while).
Module spatial.8.21.UC1 needs interfaces not registered in database
demo.
The required interface is provided by the modules:
           1 - ifxrltree.2.00
Select the number of a module above to register, or N :- 1
Registering DataBlade module... (may take a while).
DataBlade ifxrltree.2.00 was successfully registered in database demo.
Registering DataBlade module... (may take a while).
DataBlade spatial.8.21.UC1 was successfully registered in database
demo.
IDS 11>bye
Disconnecting...
informix@ibmswg01:~>
```

In this example, we start by executing the blade manager program. The prompt indicates with which instance we are working. This example uses the IDS 11 instance. The first command, **list demo**, looks into the demo database to see if any DataBlade modules are already installed. The next command, **show modules**, provides a list of the DataBlade modules that are installed in the server under \$INFORMIXDIR/extend. The names correspond to the directory names in the extend directory. The blade manager utility looks into the directories to make sure they are proper DataBlade directories. This means that other directories can exist under extend, but are not listed by the **show modules** command.

You register a DataBlade with the **register** command. Upon execution, it looks for dependencies on other modules and provides the ability to register the required modules before registering the dependent module. After the work is done, the **bye** command terminates BladeManager. After the DataBlade module is registered, you can start using it in the specified database.

A GUI version of BladeManager is also available on Windows.

7.5 DataBlades and Bladelets

When it comes to taking advantage of IDS extensibility, you also need to look at the available DataBlade modules that come with IDS or are available for a license fee.

As discussed in Chapter 6, “An extensible architecture for robust solutions” on page 219, a DataBlade is a packaging of functionality that solves a specific domain problem. It can include UDTs, UDRs, and user-defined aggregates (UDAs), tables, views, and potentially a client interface. They are packaged in a way that makes it easy to register their functionality into a database.

Bladelets and example code

An alternative to using DataBlades is to take advantage of extensions that are already written, which are known as either *Bladelets* (small DataBlades) or example code. These extensions are available in a fashion similar to open-source code, in that they come with source code but are not supported by IBM or the original author.

With access to the source code, you can study the implementation and then enhance it to suit your business requirements and environment. These extensions can provide key functionality that can save time, cost, resources, and effort.

7.5.1 DataBlades included with IDS

IDS includes the following DataBlades in the \$INFORMIXDIR/extend directory in the installation:

- ▶ Binary DataBlade

With this DataBlade, you can store and index binary data up to 253 characters in length. You can use this DataBlade to store items such as IP addresses and Globally Unique IDentifiers (GUIDs).

You can also use it as a bitmap field where several low cardinality fields can be combined together and searched as a single higher cardinality field. The DataBlade provides the BINARYVAR and BINARY18 type for doing this. The BINARYVAR type is a variable length type that supports up to 253 bytes, and the BINARY18 type is a fixed-length type that stores 18 bytes. It includes such functions as bit_and, bit_or, bit_xor, and bit_complement.

- ▶ MQSeries® DataBlade

This DataBlade enables the exchange of asynchronous messages in a distributed, heterogeneous environment by offering an interface between IDS and the WebSphere MQ messaging products installed on the same machine. WebSphere MQ products are key components of the IBM enterprise service bus (ESB) to support service-oriented architectures (SOAs).

- ▶ Basic Text Search DataBlade

This DataBlade extends IDS so that you can conduct more powerful text searches than the LIKE or MATCHES SQL operators currently provide. By using Lucene indexing and query technology, this DataBlade provides the ability to create a text search index on CHAR, VARCHAR, LVARCHAR, NVARCHAR, NCHAR, and CLOB columns so that you can do fuzzy, wildcard, and proximity searches.

- ▶ Web Feature Service

This Web Feature Service (WFS) DataBlade provides an Open GeoSpatial Consortium WFS for exchanging geographic data in XML/GML using a REST (HTTP GET/POST) interface. It can be used with geospatial mapping tools that contain a WFS client interface or APIs such as Google Maps. For more information about this DataBlade, see Chapter 8, “Extensibility in action” on page 295.

- ▶ Node

With the Node DataBlade, you can manipulate hierarchies in a more efficient manner. This type includes a set of functions, such as isAncestor(), isDescendant(), isParent(), and so on, that provide information about the hierarchical relationship of two node types. This information can be useful in such applications as bill-of-material and personnel hierarchies. In some cases, you can get more than an order of magnitude faster processing as compared to traditional relational processing.

- ▶ Spatial

The Spatial DataBlade implements a set of data types and functions that allow for the manipulation of spatial data within the database. By using this DataBlade, some business-related questions become easier to answer, for example:

- Where are my stores located in relation to my distributors?
- How can I efficiently route my delivery trucks?
- How can I micro-market to customers fitting a particular profile near my worst performing store?
- How can I set insurance rates near flood plains?
- Where are the parcels in the city that are impacted by a zoning change?
- Which bank branches do I keep after the merger based on my customer locations (among other things)?

Locations and spatial objects are represented with new data types in the database, such as ST_Point, ST_LineString, and ST_Polygon. Such functions as ST_Contains(), ST_Intersects(), and ST_Within() operate on these data types.

▶ Large Object Locator (LLD)

The Large Object Locator module presents a consistent interface for storing large objects in the database (such as CLOBs or BLOBs) and others outside the database in the file system. It is a prerequisite for the Excalibur Text DataBlade and includes the following interfaces:

– SQL interface

This is a set of functions that are used in SQL statements to manipulate the LLD objects. It includes loading an LLD object from a file or writing an LLD object to a file.

– API library

This interface is used when there is need to manipulate the LLD objects in UDFs that are written in C.

– ESQL/C library

With this library, client programs that are written in ESQL/C can manipulate LLD objects.

7.5.2 Other available DataBlades

The following DataBlades are also available for license:

▶ Geodetic DataBlade

The Geodetic DataBlade module manages geographical information system (GIS) data in IDS. It does so by treating the earth as a sphere instead of a flat map. This is an important difference from the Spatial DataBlade module. Since the earth is not a perfect sphere, it is seen as an ellipsoid.

The coordinate system uses longitude and latitude instead of the simple x and y coordinates that are used for flat maps. On a flat map, distances between points with the same difference in longitude and latitude are at the same distance from each other. With an ellipsoid earth, the distance varies based on the location of the points.

► Excalibur Text DataBlade

The Excalibur Text DataBlade provides the ability to conduct textual searches of not only traditional database character columns that store comments and descriptions, but the ability to search documents (such as Microsoft Word or Adobe® PDF) as well. It supports exact match, phrase search, wildcard searches, and boolean searches. It also supports the ability to provide synonyms, stop words (words that you do not want to index), and customized character sets.

► Web DataBlade

The Web DataBlade module provides a way to create Web applications that generate dynamic HTML pages. These Web applications are called *appPages*. The appPages use SGML-compliant tags and attributes that are interpreted in the database to generate the desired resulting document. These few tags allow for the execution of SQL statements, manipulation of results, conditional processing, and error generation. The creation of pages is not limited to HTML. Any tags can be used in the generation of a page. This means that the Web DataBlade module can easily generate XML documents.

► TimeSeries DataBlade

The TimeSeries DataBlade module provides a solution to optimize the storage and processing of data based on time. It includes features, such as user-defined timeseries data types, calendars and calendar patterns, regular and irregular timeseries, and a set of functions to manipulate the data. It provides advantages over traditional relational tables by eliminating duplicate information as well as storing the data in time order. There are methods for accessing and manipulating the TimeSeries data for C, SQL, and Java.

► C-ISAM DataBlade

With this DataBlade, you can access existing ISAM files to add existing features to your C-ISAM environment or assist in migrating C-ISAM applications to IDS. It does this by providing a VTI.

► Video Foundation DataBlade

The Video Foundation DataBlade module is an open and scalable software architecture that allows third-party development partners to incorporate specific video technologies, such as video servers, external control devices, compression codes, or cataloging tools, into relational database management system (RDBMS) applications. It also extends the capabilities of IDS by providing data types and functions that can be used to manage and index

both content and metadata, while the actual video content can be maintained elsewhere.

7.5.3 Available Bladelets

Several Bladelets are available on IBM developerWorks that you can incorporate into your environment. Table 7-2 shows a sample of the available Bladelets.

Table 7-2 *Bladelets available on developerWorks*

Bladelet	Description
Exec	Provides dynamic SQL functionality with an SPL procedure.
Flat File Access Method	A complete access method that lets you build virtual tables based on operating system files.
JPEG Image Bladelet	Provides a UDT for JPEG images so that you can manipulate images and extract and search on image properties.
Multirepresentational lvarchar Opaque Type	Creates the ids_mrLvarchar opaque type, which stores character data up to 2 GB.
Regexp	Creates routines so that you can manipulate character and CLOB data by using regular expressions.
Shapes	Creates several opaque types for managing simple spatial data, including R-tree index support.
XSLT	Creates new SQL functions that allow transformation of XML documents from one format to another using XSLT style sheets.
SqlLib	SqlLib is a Bladelet that adds several other database compatibility functions to IDS. Versions are implemented in both Java and C languages.

For a more complete list of Bladelets and example code, refer to the following Web address:

http://www.ibm.com/developerworks/db2/zones/informix/library/samples/db_downloads.html

7.6 Summary

In this chapter, we have introduced basic extensibility that uses C, SPL, and Java and demonstrated how you can use it to enhance your applications to save development time and database processing time. We also introduced the DataBlade API and the power that can be harnessed to provide even richer data types and functions. In addition, we discussed the other DataBlades that can be added to customize your environment for bigger business advantage.

IDS has the extensibility components and functionality that can help as you customize IDS for your particular environment.



Extensibility in action

In previous chapters, we discuss the business case and how to get started with extensibility in Informix Dynamic Server (IDS). In this chapter, we show how you use IDS in the following ways:

- ▶ Reduce the load on client applications for generating data that is sent to the database
- ▶ Aggregate it in the database so that it does not have to come back to the client
- ▶ Improve performance by using views

We also discuss how to consume Web Services and explain new capabilities for geospatial mapping and location-based services by using the Web Feature Service (WFS) available in IDS 11. Finally, we show how you can index your data in a non-traditional way by using the “sound” of the data.

8.1 Pumping up your data with iterators

An *iterator function* is a user-defined function (UDF) that returns to its calling SQL statement several times, each time returning a value. Iterator functions can be written in SPL, C, or Java. IDS gathers these return values together in an *active set*. To access a value in the active set, you must obtain it from either an implicit or explicit database cursor. An implicit cursor can be found when calling the iterator function from a SELECT statement. An explicit cursor is created by the user for retrieving the values.

The simplest example of an iterator function is an SPL function that uses the RETURN WITH RESUME construct, which is shown in Example 8-1. This function takes two arguments that represent the start and end date and returns all of the business days (defined as Monday through Friday) between them.

Example 8-1 Simple iterator example that uses SPL

```
CREATE FUNCTION busdaycal(startdate DATE, enddate DATE)
RETURNS DATE
DEFINE loopvar INTEGER;
DEFINE numiters INTEGER;
DEFINE currdate DATE;
LET numiters = enddate - startdate;
LET currdate = startdate;
FOR loopvar = 1 TO numiters
    IF weekday(currdate) > 0 AND weekday(currdate) < 6 THEN
        RETURN currdate WITH RESUME;
    END IF;
    LET currdate = currdate + 1;
END FOR;
RETURN currdate;
END FUNCTION;
```

Sample usage:

```
EXECUTE FUNCTION busdaycal('11-19-2007','11-27-2007');
```

(expression)

```
11/19/2007
11/20/2007
11/21/2007
11/22/2007
11/23/2007
```

11/26/2007

11/27/2007

7 row(s) retrieved.

8.1.1 Writing a C-based iterator function

Iterator functions can also be written in C by using the DataBlade API. Doing so gives you the ability to construct functions that work with complex data types, examine data structures, and link to outside libraries to provide your application with powerful new capabilities. Iterators written in C are called with state information. There are three possible states of an iterator function:

- ▶ **SET_INIT**

This state is set on the first invocation of the iterator function. When the function is in this state, such tasks as memory allocation and other initializations are done. Also, no values that are returned from this state are included in the active set.

- ▶ **SET_RETONE**

This state is the iteration state where values are returned from the iterator function to IDS. The iterator function remains in this state until `mi_fp_setisdone(fparam, MI_TRUE)` is called.

- ▶ **SET_END**

This state is the end state for the iterator, as triggered by the `mi_fp_setisdone()` function discussed later in this section. When the function is in this state, any memory that was allocated in the **SET_INIT** portion should be freed. Also, as with the **SET_INIT** state, any values that are returned during this state are not included in the active set.

This state information is passed internally via the `MI_FPARAM` structure. This structure is declared for all user-defined routines (UDRs) written in C that can be accessed from SQL. This structure is commonly used to examine and modify information such as the type, length, precision, and null state of the arguments passed to the UDR. It can also examine and modify the same type of information for the return arguments from the UDR.

The iterator state information is examined and modified by using the following two functions:

- ▶ `mi_fp_request(MI_FPARAM)`

This function is an accessor function, with which you can obtain the current state of the iterator. It takes a pointer to an `MI_FPARAM` structure that was

declared in your function. The return value is one of SET_INIT, SET_RETONE, or SET_END, which we discussed earlier.

► `mi_fp_setisdone(MI_FPARAM, INTEGER)`

By using this function, you can terminate processing for the iterator. It takes a pointer to an MI_FPARAM structure that was declared in your function, as well as an INTEGER argument that can 0 (MI_FALSE) or 1 (MI_TRUE). After this function is executed, no further results are returned from the function.

Important: Make sure that your iterator UDR includes a call to `mi_fp_setisdone()`, or your function will execute indefinitely.

In addition, you can also store the address of private-state information, called *user-state information*, in a special field of the MI_FPARAM structure. IDS passes the same MI_FPARAM structure to each invocation of the UDR within the same routine sequence. When your user-state information is part of this structure, your UDR can access this across all invocations. This user-state information is examined and modified by using the following two functions:

► `mi_fp_funcstate(MI_FPARAM)`

With this function, you can obtain the user-state pointer from the MI_FPARAM structure of the UDR. It takes a pointer to an MI_FPARAM structure as its only argument. It returns a void pointer that should be cast to a pointer to your user-state information structure.

► `mi_fp_setfuncstate(MI_FPARAM, void)`

With this function, you can set the user-state pointer in the MI_FPARAM structure of the UDR. It takes a pointer to the MI_FPARAM structure for your UDR, as well as a pointer to your user-state information structure. It does not return any values.

Memory protection: When you allocate memory for the user-state information, you must protect this memory so that it is not reclaimed by IDS while it is still in use. We recommend that you define a memory duration of PER_COMMAND used with the `mi_dalloc()` memory allocation function or explicitly change the current duration via `mi_switch_mem_duration()`.

To construct a C-based iterator function:

1. Declare the iterator function so that the return value has a data type that is compatible with the items in the active set. Therefore, if your iterator function returns a set of integers, then declare the iterator function `mi_integer`.
2. Include a pointer to the MI_FPARAM structure as the *last* parameter of the C declaration. As discussed earlier, this holds the iterator status, iterator

completion flag, and the user-state pointer. Remember that this parameter is not declared in the SQL CREATE FUNCTION definition that we discuss later.

3. In the body of the iterator function, obtain the iterator status from the MI_FPARAM structure via the mi_fp_request function that we discuss earlier.
4. For each of the iterator status values of SET_INIT, SET_RETONE, and SET_END, take the appropriate actions as needed by your requirements. Example 8-2 shows a skeleton of a C-based iterator.

Example 8-2 C skeleton for iterator function

```
typedef struct myIterState1
{
/* declare the necessary components here */
}myIterState;

mi_integer MyIterator(..., MI_FPARAM *fparam)
{
    mi_integer next;
    myIterState *myState = NULL;
    switch ( mi_fp_request(fparam) )
    {
        case SET_INIT:
            /* Allocate memory in this function */
            next = myIterator_init(..., fparam);
            break;
        case SET_RETONE:
            next = myIterator_retone(fparam);
            myState = (myIterState *)mi_fp_funcstate(fparam);
            if (end condition)
            {
                mi_fp_setisdone(fparam, MI_TRUE);
                next = 0; /* value is not part of the set */
            }
            break;
        case SET_END:
            /* Deallocate any memory allocated in the init function */
            next = myIterator_end(fparam);
            break;
    }
    return(next);
}
```

5. After compiling the file or files that contain your C-based iterator into a shared library, as discussed in Chapter 7, “Easing into extensibility” on page 263,

register the iterator function as a UDF with the ITERATOR routine modifier in the CREATE FUNCTION statement, as demonstrated in Example 8-3.

Example 8-3 SQL definition for user-defined iterator function

```
CREATE FUNCTION myIterator(INTEGER)
RETURNS INTEGER
WITH (ITERATOR)
EXTERNAL NAME “$INFORMIXDIR/extend/myfuncs/myiterfunc.so(MyIterator)”
LANGUAGE C;
```

The iterator function that you created can be used in either an SQL EXECUTE FUNCTION statement or an SQL SELECT statement, as demonstrated in Example 8-4.

Example 8-4 Methods of invoking iterator functions

Method 1:

```
EXECUTE FUNCTION myIterator(100);
```

Method 2:

```
SELECT myit.val FROM TABLE(FUNCTION myIterator(100)) AS myit(val)
```

FUNCTION keyword: The keyword FUNCTION that appears in the TABLE clause for the SELECT statement is optional in IDS version 11 and later.

8.1.2 Generating data with iterators

One of the principle uses of an iterator function is to generate data directly in IDS without a client application or needing to store the values that must be returned to the client. For example, a retailer who has multiple locations might want to use an iterator function to generate the rows that needed for a SKU-store relationship table without forming individual rows in the client application.

To help demonstrate how to construct a data generation iterator, Example 8-5 on page 301 provides a UDR. This iterator function generates telephone numbers that are given an area code, an exchange, and the number of telephone numbers requested. For simplicity in the example, we use a simple one-up sequence starting at 0. However, you can customize this example in a number of ways, including using IDS collection types, such as SET or LIST for the area codes, and ranges for the exchange numbers and actual number itself. The iterator function returns the telephone number as a character field with the format

“999-999-9999”. For brevity, customary error checks for memory allocations and boundary conditions have been removed.

Example 8-5 Source code for telephone number generator

```
#include <mi.h>
#include <strings.h>
/* Declare user-state information structure */
typedef struct TelIterator1 {
mi_integer currval;
mi_integer remvals;
mi_string *initval;
} TelIterState;

UDREXPORT mi_lvarchar *GenTelNumbers(mi_integer npa, mi_integer nxx,
mi_integer numvalues)
{
    mi_lvarchar *retval = NULL;
    mi_string retstring[13];
    mi_string myscratch[5];
    TelIterState *TelIterator = NULL;

    switch (mi_fp_request( fp ))
    {
        case SET_INIT:
            /* Allocate memory for user-state information */
            TelIterator = mi_dalloc(sizeof(TelIterState), PER_COMMAND);
            memset(TelIterator, 0, sizeof(TelIterState));
            TelIterator->currval = 0;
            TelIterator->remvals = numvals;
            /* All allocations in the structure need same duration */
            TelIterator->initval = mi_dalloc(8, PER_COMMAND);
            memset(TelIterator->initval, 0, 8);
            sprintf(TelIterator->initval, "%03d-%03d-", npa, nxx);
            /* Store the user-information state */
            mi_fp_setfuncstate(fp, (void *)TelIterator);
            mi_fp_setreturnisnull(fp, 0, MI_TRUE);
            break;
        case SET_RETONE:
            /* Retrieve the user-information state */
            TelIterator = (TelIterState *)mi_fp_funcstate(fp);
            if (TelIterator->remvals-- == 0)
            {
                /* We are done, terminate the active set */
                mi_fp_setisdone(fp, MI_TRUE);
            }
    }
}
```

```

        retval = NULL;
        mi_fp_setreturnisnull(fp, 0, MI_TRUE);
    }
    else
    {
        memset(retstring, 0, sizeof(retstring));
        memset(myscratch, 0, sizeof(myscratch));
        sprintf(myscratch,"%04d", TelIterator->currval++);
        strncat(retstring, TelIterator->initval, 8);
        strncat(retstring, myscratch, 4);
        mi_fp_setreturnisnull(fp, 0, MI_FALSE);
        retval = mi_string_to_lvarchar(retstring);
    }
    break;
case SET_END:
    /* Free the memory allocated earlier */
    if (TelIterator)
        mi_free(TelIterator);
    mi_fp_setreturnisnull(fp, 0, MI_TRUE);
    break;
}
return(retval);
}

```

After compiling this file into a shared object, we register it to the database as shown in Example 8-6.

Example 8-6 Registering the telephone number generator

```

CREATE FUNCTION GenTelNumbers(INTEGER, INTEGER, INTEGER)
RETURNS LVARCHAR
WITH (ITERATOR)
EXTERNAL NAME
“$INFORMIXDIR/extend/redbook_funcs/IterExample.bld(GenTelNumbers);
LANGUAGE C;

```

Now that the function is registered, we can use it as shown in Example 8-7.

Example 8-7 Execution of the telephone number generator

```

SELECT tel.newnumber FROM TABLE(GenTelNumbers(801,438,100))
AS tel(newnumber);

newnumber 801-438-0000
newnumber 801-438-0001

```



```
newnumber 801-438-0002
newnumber 801-438-0003
...
newnumber 801-438-0097
newnumber 801-438-0098
newnumber 801-438-0099
```

```
100 row(s) retrieved.
```

8.1.3 Improving performance with iterator functions

Many customers and application vendors use IDS views to help abstract and sometimes secure access to their data. A common issue faced when using IDS views is that, when a view definition includes GROUP BY and ORDER BY elements, the entire view must be materialized in order to apply any filters to restrict the amount of data coming back to the application.

Since IDS does not provide the capability to parameterize views, we demonstrate a technique that can be used to provide the equivalent functionality. This technique involves using a combination of global variables that can be defined by using an SPL stored procedure and an SPL iterator function that accesses it. A global SPL variable has the following characteristics:

- ▶ Requires a DEFAULT value
- ▶ Can be used in any SPL routine, although it must be defined in each routine in which it is used
- ▶ Carries its value from one SPL routine to another until the session ends
- ▶ Can be of any type, but a collection data type (SET, MULTISSET, or LIST)

It is also possible to use a C UDR that declares a piece of named memory that is valid for a given session or even for all sessions, but we do not discuss this in detail here. For simplicity in showing the example, we use SPL. Example 8-8 shows how to define and access a global variable in SPL, as well as sample usage.

Example 8-8 Defining and using GLOBAL variables in SPL

```
CREATE PROCEDURE set_sessvar(globvar INTEGER)
  DEFINE GLOBAL myglobalint INTEGER DEFAULT 0;
  LET myglobalint = globval;
END PROCEDURE;
```

```
CREATE FUNCTION use_sessvar()
  RETURNS INTEGER
```

```

    DEFINE GLOBAL myglobalint INTEGER DEFAULT 0;
    RETURN((myglobalint * myglobalint) + 2);
END FUNCTION;

```

Sample usage:

```
EXECUTE PROCEDURE set_sessvar(1000);
```

Routine executed.

```
EXECUTE FUNCTION use_sessvar();
```

```

(expression)
    1000002

```

1 row(s) retrieved.

In addition to global variables, IDS has several built-in variables that can also be referenced in this manner. They come in the form of operators, such as CURRENT, SYSDATE, TODAY, and USER, but there are also ones that can be accessed via the SQL DBINFO(), such as DBINFO('sessionid'). These variables can be useful if you want to control the result set based on the user name that is using USER, but also can be maintained in a table based on the current session identifier.

We have now established a communication mechanism with session variables. Therefore, we can define a simple SPL function that uses a FOREACH loop with RETURN WITH RESUME to retrieve all of the rows in the query whose key matches the session level variable that was set. This has the effect of applying the desired filter to the portions of the query that must be materialized in full prior to being filtered. This in turn reduces the amount of rows that are typically joined in a nested-loop join and improves the overall performance of the query. To demonstrate this technique, we have prepared the sample schema shown in n Example 8-9.

Example 8-9 Table schema for parameterized view example

```

CREATE TABLE custs (
  id SERIAL NOT NULL,
  cust_name VARCHAR(50),
  PRIMARY KEY (id));

```

```

CREATE TABLE manufactures (
  id INTEGER,
  name VARCHAR(50),

```

```
PRIMARY KEY (id));
```

```
CREATE TABLE mobile_phones (  
  phone_number CHAR(12),  
  cust_number INTEGER,  
  manufact_id INTEGER,  
  PRIMARY KEY (phone_number));
```

```
CREATE TABLE call_log (  
  call_number SERIAL NOT NULL,  
  phone_number CHAR(12),  
  called_number CHAR(12),  
  call_time DATETIME YEAR TO FRACTION(5),  
  duration DECIMAL(16),  
  PRIMARY KEY (call_number));
```

```
CREATE INDEX c_log_idx ON call_log (phone_number) USING BTREE;
```

This is a simplified version of a common master-detail schema that involves views. We used mobile phone billing in the example, but there are many other types of applications that use this type of relationship. To build data for the example, we use the information in 8.1.2, “Generating data with iterators” on page 300, to generate phone numbers.

In Example 8-10, we define the initial view that gets materialized in full. The first view shown summarizes the call_log table with a count of calls and sum of the call durations by phone number. The second view combines the first view with some other metadata to expand the detail. A sample explain output for the query is also shown.

Example 8-10 View definitions and initial query plan

View definitions:

```
CREATE VIEW call_summary_view (phone_number, callcnt, total_mins) AS  
SELECT phone_number, COUNT(*) AS callcnt, SUM(duration) AS total_mins  
FROM call_log  
GROUP BY 1;
```

```
CREATE VIEW phone_bill_view (customer_name, phone_number,  
  equip_manufact, number_of_calls, total_minutes) AS  
SELECT c.cust_name AS customer_name, m.phone_number, mf.name AS  
  equip_manufact, csv.callcnt, csv.total_mins  
FROM mobile_phones m, custs c, manufactures mf, call_summary_view csv  
WHERE m.cust_number = c.id AND
```

```
m.phone_number = csv.phone_number AND
m.manufact_id = mf.id;
```

Sample query:

```
SELECT * FROM phone_bill_view WHERE phone_number = '801-555-0015'
```

Explain output:

Estimated Cost: 105

Estimated # of Rows Returned: 19

1) (Temp Table For View): SEQUENTIAL SCAN

2) informix.mobile_phones: INDEX PATH

(1) Index Keys: phone_number (Key-First) (Serial, fragments: ALL)

Lower Index Filter: informix.mobile_phones.phone_number = (Temp Table For View).phone_number

Index Key Filters: (informix.mobile_phones.phone_number = '801-555-0015')

NESTED LOOP JOIN

3) informix.manufacts: INDEX PATH

(1) Index Keys: id (Serial, fragments: ALL)

Lower Index Filter: informix.mobile_phones.manufact_id = informix.manufacts.id

NESTED LOOP JOIN

4) informix.custs: INDEX PATH

(1) Index Keys: id (Serial, fragments: ALL)

Lower Index Filter: informix.mobile_phones.cust_number = informix.custs.id

NESTED LOOP JOIN

As you can see in the explain output in the query, a temporary table is necessary to materialize the call_summary_view embedded in the definition of the phone_bill_view.

To improve the query plan, and thus performance, we define the two SPL functions to set and retrieve a session global variable, as well as a modified view

that uses the iterator function to help subset the data in Example 8-11. This iterator function essentially replaces the `call_summary_view` defined in Example 8-10, by calling the view text in a `FOREACH .. RETURN WITH RESUME` loop. This is combined with the phone number that is initialized with the `set_phone_number()` procedure. Note that we do not parameterize the `Filter_Call_Detail()` function because this gets compiled into the new view. Also we want the values to come from the session global variables, not from any parameters that are declared to the function.

Example 8-11 Iterator function and modified view definition

```
CREATE PROCEDURE set_phone_number(telnumber CHAR(12))
  DEFINE GLOBAL globphone CHAR(12) DEFAULT '000-000-0000';
  LET globphone = telnumber;
END PROCEDURE;

CREATE FUNCTION Filter_Call_Detail()
RETURNS CHAR(12), INTEGER, DECIMAL
  DEFINE GLOBAL globphone CHAR(12) DEFAULT '000-000-0000';
  DEFINE cd_phone CHAR(12);
  DEFINE cd_callcnt INTEGER;
  DEFINE cd_totmins DECIMAL;
  FOREACH SELECT phone_number, COUNT(*) AS call_cnt, SUM(duration) AS
total_mins INTO cd_phone, cd_callcnt, cd_totmins FROM call_log WHERE
phone_number = globphone GROUP BY 1
  RETURN cd_phone, cd_callcnt, cd_totmins WITH RESUME;
  END FOREACH;
END FUNCTION;

CREATE VIEW phone_bill_view_new(customer_name, phone_number,
equip_manufact, number_of_calls, total_minutes) AS
SELECT c.cust_name AS customer_name, m.phone_number, mf.name AS
equip_manufact, cd.callcnt, cd.totmins
FROM mobile_phones m, custs c, manufactures mf,
TABLE(Filter_Call_Detail()) AS cd(phone_number, callcnt, totmins)
WHERE m.cust_number = c.id AND
m.phone_number = cd.phone_number AND
m.manufact_id = mf.id;
```

Now that we have created a new view that uses this iterator function, we demonstrate how it is executed in Example 8-12. First, we initialize the value to pass to the iterator function. Next, we query the new view, still providing a `phone_number` to be queried. This is still necessary for the filter to get folded into the view execution when it attempts to retrieve data from the `mobile_phones` table.

Finally, we provide a new EXPLAIN PLAN that shows the difference in execution. The temporary table that is necessary for the materialized view is now gone and replaced with a iterator scan. The iterator scan is using an index to retrieve the summarized call data.

Example 8-12 New query plan that uses the iterator

Variable set up:

```
EXECUTE PROCEDURE set_phone_number('801-555-0015');
```

Query:

```
SELECT * FROM phone_bill_view_new WHERE phone_number = '801-555-0015';
```

Explain output:

Estimated Cost: 66

Estimated # of Rows Returned: 19

1) informix.filter_call_detail: ITERATOR SCAN

Filters: informix.filter_call_detail.phone_number =
'801-555-0015'

2) informix.mobile_phones: INDEX PATH

(1) Index Keys: phone_number (Serial, fragments: ALL)
Lower Index Filter: informix.mobile_phones.phone_number =
informix.filter_call_detail.phone_number
NESTED LOOP JOIN

3) informix.manufacts: INDEX PATH

(1) Index Keys: id (Serial, fragments: ALL)
Lower Index Filter: informix.mobile_phones.manufact_id =
informix.manufacts.id
NESTED LOOP JOIN

4) informix.custs: INDEX PATH

(1) Index Keys: id (Serial, fragments: ALL)
Lower Index Filter: informix.mobile_phones.cust_number =
informix.custs.id
NESTED LOOP JOIN

We have shown that this can be a simple but powerful technique that leverages the extensibility features of IDS to improve performance when using views.

8.1.4 A challenge

As discussed in 8.1.2, “Generating data with iterators” on page 300, one of the potential uses of iterator functions is to create new data sets. One of the biggest challenges faced by developers and database administrators is testing applications when the data sets are small and do not include all the ranges of values that should be exercised through the system. It is possible, by using the DataBlade API, to construct an iterator function that examines the structure of a table and produces test data based on the data types that it finds.

For example, the iterator function can take a table name, the number of rows desired, and an unnamed IDS ROW type that represents valid values for each column. These values can include specific values, ranges, and sequences. The return from this iterator can be an unnamed ROW type that can be inserted via an SQL INSERT statement.

This type of function would be enjoyed by many others in the IDS community. We encourage you to either submit your completed function to the International Informix User’s Group (IIUG) Software Repository or to IBM via developerWorks at the following address:

<https://www.ibm.com/developerworks/secure/myideas.jsp?start=true&domain=>

In this section, we have discussed creating iterator functions using SPL and C that can be used to generate data or to improve performance by subsetting complex views. By using this functionality, you can truly pump up your data, not only to save processing time on the client, but create data in larger volumes that can help you test applications and database layouts and tune configurables.

8.2 Summarizing your data with user-defined aggregates

An *aggregate* is a function that iterates, or operates repeatedly, over a group of rows and returns one value to a query. For each input row, an aggregate iteration receives a column value (called the *aggregate argument*), updates a variable containing the *state* of the aggregate so far, and returns this state variable as an intermediate result. After receiving the last input value, the aggregate returns the final result.

One of the more powerful features in IDS extensibility is its ability to define a new method of aggregating your data. A *user-defined aggregate* (UDA) is similar in concept to any other aggregate function that is available on the server, such as COUNT, AVG, and SUM. IDS supports extensions of aggregates in the following ways:

- ▶ Extensions of built-in aggregates
- ▶ User-defined aggregates

8.2.1 Extensions of built-in aggregates

Extensions of built-in aggregates are done by overloading internal IDS comparison and arithmetic functions and operators. This means that if you define a new data type, you can create the functions that will implement operators such as equal (=) and greater than (>). This is done by providing functions with special names, which can include the following functions:

- ▶ concat
- ▶ divide
- ▶ equal
- ▶ greaterthan
- ▶ greaterthanorequal
- ▶ lessthan
- ▶ lessthanorequal
- ▶ minus
- ▶ negative
- ▶ notequal
- ▶ plus
- ▶ positive
- ▶ times

Table 8-1 describes the various built-in aggregates and the operators that support them.

Table 8-1 Operators for IDS built-in aggregates

Aggregate	Required operators	Return type of aggregate
AVG	plus(udt, udt) divide (udt,integer)	Return type of divide()
COUNT	No new operators are required	Integer
COUNT DISTINCT	equal(udt, udt)	Boolean
DISTINCT (or UNIQUE)	compare(udt, udt)	Boolean

Aggregate	Required operators	Return type of aggregate
MAX	greaterthanorequal(udt,udt)	Boolean
MIN	lessthanorequal(udt,udt)	Boolean
RANGE	lessthanorequal(udt,udt) greaterthanorequal(udt,udt) minus(udt,udt)	Return type of minus()
SUM	plus(udt,udt)	Return type of plus()
STDEV	times(udt,udt) divide(udt,integer) plus(udt,udt) minus(udt,udt) sqrt(udt)	Return type of divide()
VARIANCE	times(udt,udt) divide(udt,integer) plus(udt,udt) minus(udt,udt)	Return type of divide()

To extend an existing IDS aggregate with a new data type, you must write the appropriate C-based UDR or UDRs that implement the required operator functions for the new data type and register them by using SQL CREATE FUNCTION statements. For a detailed example of how to implement this type of aggregate, see Chapter 15 of the *IBM Informix DataBlade API Programmer's Guide*, G229-6365.

8.2.2 User-defined aggregates

Not all possible aggregations have been provided by IDS. That is why IDS also includes the ability to create UDAs. This type of aggregate can be used, for example, to generate XML from a table stored in an IDS database or to union spatial geometries together to form new geometries. UDAs can be defined for almost all IDS data types (built-in and opaque) with the exception of the following types:

- ▶ Collection data types (LIST, MULTISSET, SET)
- ▶ Unnamed row types
- ▶ Smart-large-object data types (CLOB or BLOB)
- ▶ Simple-large-object data types (TEXT or BYTE)

In order to define a UDA, you must provide aggregate support functions that implement the tasks of initializing, calculating, and returning the aggregate result.

One of the principal concepts behind the UDA implementation is that it must operate within the IDS multi-threaded architecture. This means that it must be able to merge partial results that might be created by multiple readers into a final result. There are four types of aggregate support functions:

- ▶ INIT
- ▶ ITER
- ▶ COMBINE
- ▶ FINAL

The purpose of the *optional* INIT aggregate function is to return a pointer to the initialized aggregate state. Whether the INIT function is required depends on whether the aggregate has a *simple state* or *non-simple state*.

A *simple state* is an aggregate state variable whose data type is the same as the aggregate argument. This is enough in aggregates such as the built-in aggregate SUM(), which requires only a running sum as its state. However, the AVG() built-in aggregate requires both a sum and a running count of records processed. Therefore, it is said to have a *non-simple state*, because special processing is needed after all the iterations to produce the final result.

As with the iterator functions described in 8.1.2, “Generating data with iterators” on page 300, the state must be maintained in the PER_COMMAND memory pool, so that the different threads have access to the state.

User-defined aggregate: MEAN()

Consider an example of creating an aggregate called MEAN(), which computes a mean value of the columns that have the DECIMAL data type. In the examples that follow for MEAN(), some error checking and code path efficiencies have been removed to simplify the examples.

In Example 8-13, we define a structure that holds the state of the aggregate to be used during the calculation. This structure holds the running sum and the count of the records used, and must be passed via an *mi_pointer* C data type. When this and the other support functions are registered later, we use an equivalent SQL POINTER type in the definition of the functions, for both arguments and return variables.

Example 8-13 INIT aggregate function for MEAN()

```
typedef struct Dag_state_struct {
    mi_decimal sum;
    mi_integer count;
} DagState_t;

DagState_t *dagStateInitMean(void)
```

```

{
    DagState_t *pDagState;
    mi_integer errCode;

    pDagState = (DagState_t *)mi_dalloc(sizeof(DagState_t),
                                         PER_COMMAND);
    errcode = deccvint(&pDagState->sum, 0);
    pDagState->count = 0;
    return(pDagState);
}

(void) dagStateFree(DagState_t *pDagState)
{
    if (pDagState)
        mi_free(pDagState);
}

UDREXPOR mi_pointer MeanInitializedec(mi_decimal *dummyArg, MI_FPARAM
*fp)
{
    mi_pointer retval;
    mi_integer errCode;

    retval = (mi_pointer)dagStateInitMean();
    return(retval);
}

```

Non-simple states: There are multiple types of non-simple states that an aggregate can use, including single-valued state and opaque-type state, in addition to the pointer-value state discussed in this example. Refer to Chapter 15 in the *IBM Informix DataBlade API Programmer's Guide*, G229-6365, for further discussion about these other non-simple states.

The ITER aggregate function performs the sequential aggregation or iteration for the UDA. It merges the single aggregate argument into the partial result, which the aggregate state contains. Example 8-14 shows how we implement the ITER function for the MEAN() aggregate that we previously described.

Example 8-14 ITER aggregate function for MEAN()

```

UDREXPOR mi_pointer MeanIteratedec(mi_pointer State, mi_decimal
*RowValue, MI_FPARAM *fp)
{
    mi_pointer retval;
    mi_integer errCode;

```

```

    ((DagState_t *)State)->count++;
    errCode = decadd(&((DagState_t *)State)->sum, RowValue,
&((DagState_t)State)->sum);
    retval = State;
    return(retval);
}

```

With the COMBINE aggregate function, the UDA can execute in a parallel query. In IDS, when a query that contains a UDA is processed in parallel, each thread operates on a subset of selected rows. The COMBINE aggregate function merges the partial results from two such subsets. It ensures that the result of aggregating over a group of rows sequentially is the same as aggregating over the two subsets of rows in parallel and then combining the results.

It is important to note that a COMBINE function *can* be called even when a query is not parallelized. Therefore, you must provide a COMBINE function for every UDA. Example 8-15 shows a COMBINE function for our implementation of the MEAN() aggregate.

Example 8-15 COMBINE function for MEAN()

```

UDREXPOR mi_pointer MeanCombinepointerdec(mi_pointer State1,
mi_pointer State2, MI_FPARAM *fp)
{
    mi_pointer retval;
    mi_integer errCode;

    errCode = decadd(&((DagState_t *)State1)->sum, &((DagState_t
*)State2)->sum, &((DagState_t*)State1)->sum);
    ((DagState_t *)State1)->count += ((DagState_t *)State2)->count;
    dagStateFree((DagState_t) State2);
    retval = State1;
    return(retval);
}

```

In this example, we take two aggregate states (State1 and State2), which represent two possible parallel threads, and merge the results. The COMBINE function is called for each pair of threads that IDS allocates. There is no need to write functions to handle more than two states. IDS simply uses the COMBINE function for states repeatedly until only one state remains. Before the function returns, it must also release any resources that were associated with one of the partial results.

The *optional* FINAL aggregate function performs the post-iteration tasks for the UDA. In this function, such tasks as type conversion and post-iteration calculations are performed, as well as the deallocation of any memory resources that were allocated in the INIT function, if specified. Our example MEAN() aggregate defines an INIT function, due to its non-simple aggregate state. Example 8-16 shows how we implement the FINAL function. Our example takes the State pointer that was produced by the earlier COMBINE function and returns a DECIMAL type. Note that we must allocate memory for the return type.

Example 8-16 FINAL function for MEAN()

```
UDREXPOR mi_decimal *MeanFinalpointerdec(mi_pointer State, MI_FPARAM
*fp)
{
    mi_decimal *retval;
    mi_decimal divisor;
    mi_integer errCode;
    retval = (mi_decimal *)mi_zalloc(sizeof(mi_decimal));
    if (((DagState_t *)State)->count == 0)
    {
        mi_fp_setreturnisnull(fp, 0, MI_TRUE);
    }
    else
    {
        mi_fp_setreturnisnull(fp, 0, MI_FALSE);
        errCode = deccvint(((DagState_t *)State)->count, &divisor);
        errCode = decdiv(&((DagState_t *)State)->sum, &divisor, retval);
    }
    dagStateFree((DagState_t *) State);
    return(retval);
}
```

Now that we have created all the supported UDRs for our MEAN() aggregate, we must compile these functions into a shared-object library and register them via the SQL CREATE FUNCTION statement. Example 8-17 shows how to register these four functions.

Example 8-17 Registering the aggregate support functions

```
CREATE FUNCTION MeanInitialize(DECIMAL)
RETURNS POINTER
WITH (HANDLESNULLS, PARALLELIZABLE)
EXTERNAL NAME
"$INFORMIXDIR/extend/redbook_funcs/DecimalAggs.bld(MeanInitializdec)"
LANGUAGE C;
```

```

CREATE FUNCTION MeanIterate(POINTER, DECIMAL)
RETURNS POINTER
WITH (PARALLELIZABLE)
EXTERNAL NAME
“$INFORMIXDIR/extend/redbook_funcs/DecimalAggs.bld(MeanIteratedec)”
LANGUAGE C;

CREATE FUNCTION MeanCombine(POINTER, POINTER)
RETURNS POINTER
WITH (PARALLELIZABLE)
EXTERNAL NAME
“$INFORMIXDIR/extend/redbook_funcs/DecimalAggs.bld(MeanCombinepointerdec)”
LANGUAGE C;

CREATE FUNCTION MeanFinal(POINTER)
RETURNS DECIMAL
WITH (PARALLELIZABLE)
EXTERNAL NAME
“$INFORMIXDIR/extend/redbook_funcs/DecimalAggs.bld(MeanFinalpointerdec)”
LANGUAGE C;

```

Finally, we can create our MEAN() aggregate via the SQL CREATE AGGREGATE statement, as shown in Example 8-18.

Example 8-18 User-defined aggregate definition

```

CREATE AGGREGATE Mean WITH (
    INIT = MeanInitialize,
    ITER = MeanIterate,
    COMBINE = MeanCombine,
    FINAL = MeanFinal
);

```

HANDLENULLS: The modifier HANDLENULLS is omitted from registration of the INIT function because we are not considering NULL to be valid values in the set. During the UDA execution, if a NULL value is encountered, the ITER function is not invoked. If the HANDLENULLS modifier is provided, then the ITER function is invoked, and there should be a method for handling the NULL value in that function. The INIT function must always be declared with HANDLENULLS because it always takes a NULL dummy argument.

Example 8-19 shows how the MEAN() aggregate is executed on a table called *calls*, which contains a column called *duration* of type DECIMAL.

Example 8-19 Executing the MEAN() aggregate

```
SELECT MEAN(duration) FROM calls;
```

```
          mean
98.955300000000

1 row(s) retrieved.
```

By using this template, we can also define aggregates to compute the median and mode of a set of DECIMAL values.

User-defined aggregate: MedianExact()

The median of a set of numbers is a value that is greater than or equal to half the values in the set and less than or equal to half the values. With an odd number of values, the median is unambiguously the middle value of the ordered data values. But, with an even number of values, either the value nearest the center, or a value between them, can qualify as the median.

To calculate the median exactly, you must store every value in the aggregate state. Because the UDA cannot know in advance the number of values in the set, the state must be a variable size, allocated dynamically as needed with the following aggregate:

```
mi_dalloc(size, PER_COMMAND)
```

For the sample implementation (which is available as a download from the IBM Redbooks Web site, as explained in Appendix A, “Additional material” on page 465), the state is a structure that contains a count of values and a pointer to the values. The values are stored in a binary tree. Each node in the tree contains a decimal value, the count of rows containing that value, and pointers to two nodes with a lesser and greater value, respectively.

This tree structure allows the ITER function to add values to the state in time of order $N \cdot \log(N)$ in the best case, where N is the number of values (order N^2 in the worst case, that is ordered incoming data). However, it is most easily navigated by a recursive function. In order not to exceed the size of the stack that IDS has allocated for the function, the recursive functions must use the DataBlade API `mi_call()` function to call themselves.

The count field in the node allows the UDA to store repeated values in one node, reducing memory consumption.

The INIT function allocates the state structure and initializes counters to zero.

The ITER function compares the incoming value to the value in the first or root node. If the incoming value is less than that in the node, the function recursively calls itself with the left child node. If the value is greater than in the node, the function recursively calls itself with the right child node. It repeats this until either of the following situations occurs:

- ▶ The values compare equal. (The count is incremented.)
- ▶ A node is not found or NULL. (A node is allocated with the `mi_dalloc()` function and initialized with the incoming value and a count of 1.)

The COMBINE function selects one of the two given state structures as the source and the other as the target. It takes each node from the source tree and moves it to the target tree. A move adds the node to the target if it did not already exist. Otherwise, the count in the source node is added to that of the target, and the source node can be freed. The function navigates down both source and target trees recursively. When finished, the source tree is empty, and the source state can be freed.

The FINAL function takes the number of values maintained in the state and divides it by two to get the position of the value to retrieve. It then navigates the tree in ascending order, using the count field in each node to keep track of the number of values visited. When it reaches the correct position, it retrieves the current value. Before returning, it frees the state. Both the navigation and the freeing of the state tree are recursive.

User-defined aggregate: MedianApprox()

An exact median aggregate requires memory proportionate to the number of values. In many applications, an approximate median is good enough. There are several algorithms that compute approximate *quantiles* of a data set in limited memory.

Quantile: Quantiles, or N-tiles, are the values at evenly-spaced positions in an ordered data set. $(q-1)$ quantiles divide the data set into q equal-sized pieces. A median divides a data set into two pieces, quartiles are divided into four pieces, percentiles are divided into 100, and so on.

This UDA takes a parameter to indicate the size of memory to be allocated. This is the memory size per invocation.

The INIT function allocates memory buffers of the given size for the incoming data. However, this aggregate can operate in parallel, and therefore, the INIT function can be called multiple times, allocating buffers of this size each time. A

UDA cannot easily know how many times its INIT function is called. Therefore, the UDA would consume more memory than intended.

The ITER function adds a value to a buffer. If the buffers are full, the function collapses the data to fit into fewer buffers, by sorting it and taking evenly-spaced samples. This frees the remaining buffers to receive new data.

The COMBINE function combines and frees buffers, similar to the collapse in ITER.

The FINAL function retrieves the median value, frees the buffers, and returns.

Enhancements

A more advanced UDA can take a parameter for the number of quantiles (1, 3, 100, and so on), and return those quantiles.

Some quantile algorithms enable you to compute the variance for a given memory size or vice versa. Therefore, another more advanced UDA can take a parameter indicating the allowed error in the result and compute the required memory size from it.

User-defined aggregate: Mode()

The mode of a data set is the value that occurs most frequently. This UDA must keep every value in its state until the FINAL function. In doing so, it borrows data structures and algorithms from the MedianExact() UDA.

The INIT function allocates the state structure and initializes counters to zero, which is similar to MedianExact(), except that Mode() does not need a total count of values.

The ITER function adds the incoming value to the state, which is the same as in MedianExact().

The COMBINE function combines two states, similar to MedianExact(). (Again, Mode() has no need for a total count.)

The FINAL function navigates the state tree, saving the node with the highest count as far as it goes. The difference from MedianExact() is that Mode() must navigate the entire state tree, not half of it.

Summary of UDA

In this section, we have discussed how UDAs can be used to put together your data in new ways. By doing so, you can reduce the amount of time spent transmitting data to the client to compute these types of aggregates as well as

take advantage of the IDS multithreaded architecture to compute them more quickly.

8.3 Integrating your data with SOA

At a simplistic level, a *service-oriented architecture (SOA)* is a collection of services on a network where the services communicate with one another in order to carry out business processes. The communication can either be data passing, or it can trigger several services implementing an activity. The services are loosely coupled, have platform independent interfaces, and are fully reusable.

As we discuss and introduce in Chapter 7, “Easing into extensibility” on page 263, IDS 11 provides a complete set of features to extend the database server. It includes support for new data types, routines, aggregates, and access methods. With this technology, in addition to recognizing and storing standard character and numeric-based information, the engine can, with the appropriate access and manipulation routines, easily integrate itself into any simple or complex SOA environment.

There are several ways to integrate IDS 11 into an SOA framework. You must differentiate between service providing and service consumption in addition to foundation technologies, such as XML support and reliable messaging integration. With the industry leading extensible architecture and features that are unique to Version 11, you can easily achieve an SOA integration.

8.3.1 SOA foundation technologies in IDS 11

SOAs rely on an XML-based messaging exchange. Although it is not required to be provided by the database server for SOA integration, having XML generating functions built into the server can dramatically help with the integration development tasks. IDS 11 provides a sophisticated set of XML-related functions to create and transform XML-formatted documents.

Quite often SOA frameworks need more reliable network infrastructures in addition to the established Internet protocols, such as HTTP or HTTPS in combination with SOAP. One example of a reliable messaging infrastructure is the WebSphere MQ messaging layer. IDS 11 supports the integration into a WebSphere MQ setup through the bundled WebSphere MQ DataBlade. For more information, refer to Chapter 10, “The world is relational” on page 401.

8.3.2 Service providing with IDS 11

In most SOA scenarios, the integration work is done on the application level and not so much on the database server level. Sometimes, it might be required to provide SOA-compliant access on an IDS database-object level.

IDS 11 developers have several options for providing Web services. Most of them use one of the many application development options, such as those that follow, that are available for IDS:

- ▶ Java-based Web services (through the IDS JDBC driver)
- ▶ .NET 2.0-based Web services (through the new IDS .NET 2.0 Provider)
- ▶ IBM Enterprise Generation Language (EGL)-based Web services
- ▶ PHP, Ruby, Perl, and C/C++ Web services

In addition to the typical, application development language-based Web services listed, IDS 11 is also supported by the IBM Web Services Object Runtime Framework (WORF), which allows rapid Web service development against IDS 11 based on SQL statements, such as SELECT and INSERT, and stored procedure calls.

Finally, with the introduction of the Web Feature Service for geospatial data, IDS 11 is capable of providing an Open GeoSpatial Consortium (OGC)-compliant Web service (just add an HTTP server to IDS) to integrate easily with geospatial applications that are WFS compliant. For more information about WFS, see 8.4, “Publishing location data with a Web Feature Service” on page 324.

8.3.3 Service consumption with IDS 11

In addition to providing Web services, it can be interesting for an application developer to integrate existing Web services. Such Web services can be either special business-to-business scenarios or public accessible services, such as currency conversion, stock ticker information, news, weather forecasts, search engines, and many more. You can have dynamic access to an official currency conversion service on a database level if the application must deal with this information. Or, what if an application wants to relate actual business data stored in an IDS database against news from news agencies?

The advantages of having Web services accessible from SQL include easy access through SQL and standardized APIs (for example, Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC)). They also include moving the Web service results closer to the data processing in the database server, which can speed up applications, and providing Web service access to the non-Java or non-C++ developers.

Due to the extensible architecture in IDS 11, such as the DataBlade API (for C/C++-based extensions) or the J/Foundation features (for the Java-based extension), it is easy to develop Web service consumer routines that can be run within the IDS context.

In Figure 8-1, a simple C UDR is accessing a public currency exchange Web service to retrieve up-to-date currency exchange rates (in our example, from USD to EUR) and calculate the euro value (PRICE_EURO column) dynamically.

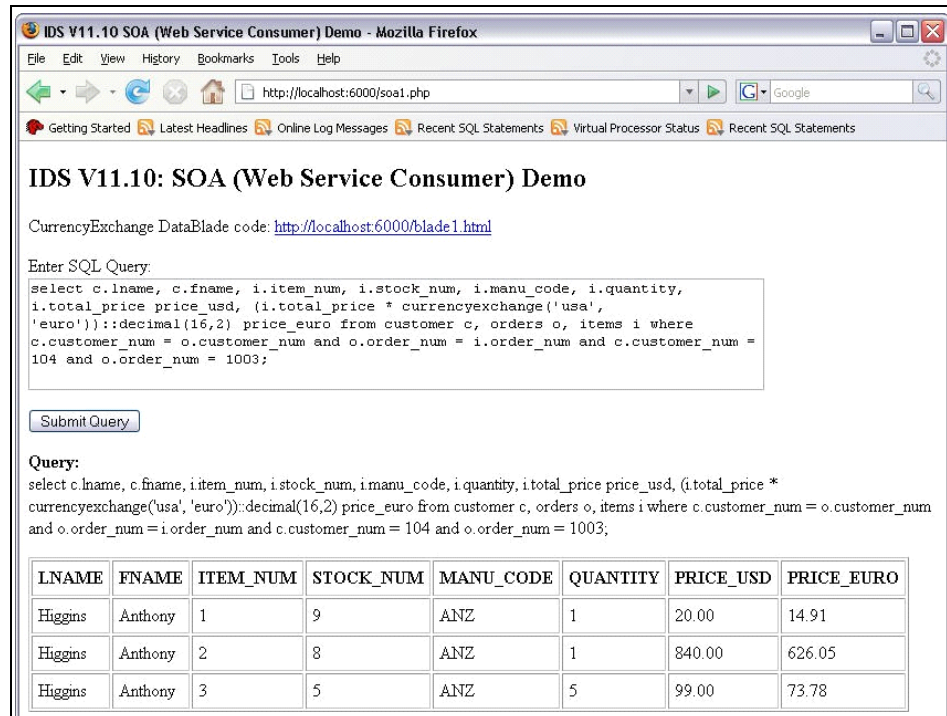


Figure 8-1 Web Feature Services

Example 8-20 illustrates the C source code of the UDR in Figure 8-1.

Example 8-20 C source code for Web service currency exchange

```
#include <ctype.h>
#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include <ifxgls.h>
#include <mi.h>

#include "CurrencyExchange.h"
```

```

#include <soapH.h>
#include "CurrencyExchangeBinding.nsmap"

UDREXPOR mi_double_precision *CurrencyExchange
(
mi_lvarchar *country1,
mi_lvarchar *country2,
MI_FPARAM *Gen_fparam
)
{
    mi_double_precision *Gen_RetVal;
    MI_CONNECTION *Gen_Con;
    struct soap *soap;
    float result = (float)0.0;

    Gen_Con = NULL;

    soap = (struct soap*)mi_alloc(sizeof(struct soap));
    if (soap == 0)
    {
        DBDK_TRACE_ERROR("CurrencyExchange", ERRORMESG2, 10);
    }

    soap_init(soap);

    Gen_RetVal =
        (mi_double_precision *)mi_alloc( sizeof( mi_double_precision ) );
    if( Gen_RetVal == 0)
    {
        DBDK_TRACE_ERROR( "CurrencyExchange", ERRORMESG2, 10 );
    }

    if (soap_call_ns1_getRate(soap, NULL, NULL,
        mi_lvarchar_to_string(country1),
        mi_lvarchar_to_string(country2),
        &result) == SOAP_OK)
        *Gen_RetVal = result;
    else
        mi_db_error_raise(Gen_Con, MI_EXCEPTION, "SOAP Fault");

    return Gen_RetVal;
}

```

Further reading

For more details about how to integrate IDS with an SOA environment, refer to the Redbooks publication *Informix Dynamic Server V10 . . . Extended Functionality for Modern Business*, SG24-7299.

8.4 Publishing location data with a Web Feature Service

Since 1997, IDS has included the ability to store, manipulate, and retrieve geographic data by using the IBM Informix Spatial and Geodetic DataBlades. A full treatment of this topic is beyond the scope of this book, but we cover enough background here to set the stage for a discussion of the WFS.

Briefly, the WFS is an open standard for retrieving and applying insert, update, and delete transactions to geographic data through an HTTP- and XML-based protocol. With IDS 11, IBM has introduced the WFS DataBlade to implement this standard. The main purpose of this section is to describe the WFS DataBlade and its use in detail. However, we begin with an overview of spatial data, its uses, and the DataBlades that help us manage this type of data.

8.4.1 How organizations use spatial data

Let us focus on maps, addresses, and geographic information. Think of national government departments such as geologic or topographic surveys, weather services, environmental protection departments, and military and intelligence agencies. Also consider local or regional government functions such as land parcel ownership records, public works, and transportation. In addition, include enterprises in industries such as power and water utilities, forestry, and oil and minerals.

These organizations have long relied on their own ability to survey and collect geographic information to create their maps. Collectively they have developed specialized tools and techniques to do so, including software tools for managing the amassed data. The general term for these software tools is *geographic information system (GIS)*. An entire industry has sprung up around this, providing tools, data, and services for everything from basic map display to advanced editing, cartographic publishing, and geographic processing and analysis.

Alternatively, many commercial and government enterprises do not have map making as their major purpose, but they manage geographic information nonetheless. Their enterprise databases all contain location information of one kind, which is addresses for customers, subscribers, members, taxpayers, suppliers, office or franchise locations, and so on. They apply quasi-spatial techniques to them, aggregating information by city, county, state, or region; matching customer to service technician or salesperson by city; setting delivery fees or insurance premiums by postal code; and so on.

Unfortunately, addresses and postal codes themselves do not support many kinds of useful spatial analysis. They cannot tell us the sites that are closest to a

given location, the nearest cross street, the number of customers that live within a given straight-line or drive-time distance, or the insured properties within the projected path of an approaching storm. For this, the addresses must first be converted to the universal currency used in a GIS, which is *spatial coordinates*. Spatial coordinates are numeric values that define a point location object that can be plotted in a coordinate reference frame and manipulated using mathematical techniques.

Fortunately, for most countries, data and services are available that make possible the conversion from text-based address to coordinate-based point location, which is called *geocoding*. A full discussion of geocoding is beyond the scope of this book. For our purposes, it is enough to know that we can turn addresses into point locations.

Geographic technology and mainstream IT

When our addresses become geographic point locations, we can apply all the same GIS techniques that are developed for the map-making industries. Because the addresses live in enterprise databases, those same databases must manage the geographic point objects, and the GIS tools must support those databases. GISs have traditionally been stand-alone applications that support specialized users and departments. Meanwhile, the map-making organizations that use them have been moving toward integrating these capabilities with enterprise-level functions and IT systems. Again, this requires that enterprise databases handle geographic data, and that GIS tools take advantage of that.

Both developments have led to the evolution of GIS from a specialized niche to an integrated aspect of mainstream IT, and to the addition of geographic, or spatial, data management capabilities to enterprise-level database products. Data server vendors, such as IBM have, teamed up with GIS software vendors, such as ESRI and MapInfo (now a unit of Pitney Bowes), to make the integration as seamless as possible.

Recently, the trend toward mainstream use of geographic technologies and map visualization has accelerated. Mapquest, Google, Yahoo, and Microsoft mapping and route planning sites have put the power of this approach in the hands of the general public. Even corporate CEOs are now asking their IT departments why they cannot see their own data this way. Mashup and Web services developments have made advanced capabilities easier to access and integrate than ever before. They have raised expectations that we can apply them to all our business processes and data. IDS is meeting the challenge with new capabilities, such as the WFS DataBlade, building on the spatial data management foundation it has had for over a decade. Let us now look at this foundation.

8.4.2 Maps and globes: The Spatial and Geodetic DataBlades

Extensibility gives us the power to match the data server's capabilities to the application problem domain. In this case, that domain is the representation (physical or human-defined) of objects (buildings, pipes, addresses, countries, parcels, watersheds, weather systems, mineral deposits, mobile phones, vehicles, and so on) in the real world as geometric shapes referenced to a coordinate system that describes their location on the surface of the earth. Thus, the location indicated by an address, or recorded by a GPS device, becomes a point. A road, pipeline, or stream is represented by a linear object that consists of connected, straight line segments. A parcel, county, or flood zone is a polygon.

Of course, an integral part of the representation of a real-world object is information about its non-spatial characteristics. Such characteristics include the name, size, value, color, owner, and many more items that can usually be encoded as traditional alphanumeric values. In GIS parlance, the entire data representation, spatial and nonspatial, of a real-world thing is a *feature*, and therefore, the name of the standard, which is the Web *Feature Service*. A feature's database equivalent is a row, either from a single table or constructed as the result of a join.

Interfaces and standards

There are many ways to design an interface for managing geographically referenced geometric shapes. For example, you can consider which the shapes to allow, what to call them, the methods and predicates they should support, and how to encode them in binary and ASCII form. Fortunately, standards are emerging for this. The reigning standard for geographic data in SQL databases is the *OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option*, which is commonly known as the *Simple Feature Specification (SFS)*, from the *OGC*.

For more information, refer to the following Web address:

<http://www.opengis.org/standards/sfs>

Since the background to this is the map-making tradition that underlies nearly all GIS practices, the specification is for shapes in the two-dimensional plane only, where straightforward Euclidian geometry (the kind taught in secondary school) applies. The term *simple* here refers to the limitation that each feature has a single geometric representation that is an instance of one of the types defined in the standard. There is no provision for managing objects that are complex, that is, composed of multiple geometric shapes (each potentially with its own set of attributes), but treated as a single value for programming and data transfer purposes.

The Spatial DataBlade is a faithful implementation of the SFS. In addition, it adds some functionality that allows for tighter integration with the software from one particular GIS software partner ESRI. We look at this DataBlade first. Then we briefly discuss the Geodetic DataBlade, which views the world rather differently, and therefore does not conform to the same standard interface (although it incorporates elements of the standard). What follows is not a substitute for the respective user guides of these products, but enough of a summary to help you understand the WFS DataBlade without referring to the more extensive documentation.

The Spatial DataBlade

As you would expect after reading Chapter 6, “An extensible architecture for robust solutions” on page 219, the Spatial DataBlade introduces new data types (UDTs), functions (UDRs), and various other items including aggregates (UDAs), casts, and index support. The spatial data types are arranged in a type hierarchy, as illustrated in Figure 8-2.

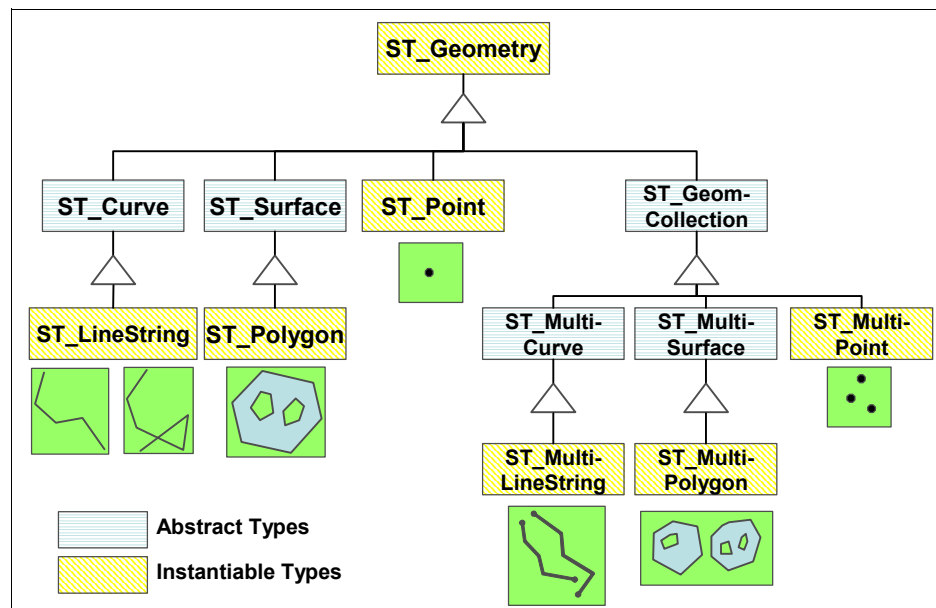


Figure 8-2 Spatial DataBlade type hierarchy

The basic types of geometry, which are point, line (here called linestring), and polygon, should be familiar. Note the following aspects of the type hierarchy as well:

- ▶ The additional types shown on the right side are collections of the ones on the left. They are defined as separate types because SQL does not have an efficient programming model for collections that need to be handled as atomic values.
- ▶ All types derive from the supertype `ST_Geometry`, which mostly supports function polymorphism. Any function defined on the supertype is available for the subtype, or any combination of subtypes. Thus, the boolean function `ST_Intersects(ST_Geometry, ST_Geometry)`, which determines whether two geometric shapes have any points in the plane in common, works for two polygons, a line and a multipolygon, any other combination of subtypes.
- ▶ Some types are defined for logical consistency and future expansion but do not play a role in table definition or queries. For example, a new subtype of `ST_Curve`, such as `ST_CircularArc`, can be defined in the future. Currently, however, all curves are represented by strings of concatenated (straight) line segments, and the same goes for the boundaries of polygons.
- ▶ We can define a column of type `ST_Geometry` and populate it with any combination of points, lines, and polygons (an `ST_Geometry` value is always an instance of one of the subtypes). In practice, this is rarely done, and most GIS software requires that each column contain only one type of geometry.

The SFS defines specific formats for representing values of these types in binary and text form. With no hint of modesty, these are designated *Well-Known Binary* (WKB) and *Well-Known Text* (WKT), respectively. For example, the WKT string in Example 8-21 defines a line segment. The choice of separators (spaces and commas) and delimiters (parentheses) is determined by the standard. The format is self-explanatory. A line segment is defined by two points (*vertices*, the plural form of *vertex*, in official usage), and each point is defined by a coordinate pair (X, Y).

Example 8-21 WKT representation of a line segment

```
'LINESTRING(523875.0 3792688.0, 523921.0 3792654.0)'
```

An additional representation defined by the OGC is *Geography Markup Language* (GML), a type of XML. GML is the basis for the WFS specification. You will see much more on GML in the discussion of WFS in the remainder of this chapter.

An aspect of spatial data that adds complexity is that coordinates, such as those in Example 8-21, only mean something in terms of a real location on the earth if the coordinate reference frame is known. That is roughly, what is the origin of the coordinate axes and what is the unit along those axes. The Spatial DataBlade has facilities for defining such *Spatial Reference Systems* (SRSs), in terms of map

projections and their several parameters as well as the assumptions that are made about the shape of the planet.

Each SRS is designated by a numeric ID, the SRID. Every ST_Geometry object carries with it the SRID of the SRS in which its coordinates are defined. SRIDs only have local significance in the current database. The map projection on which the SRS is based, however, can be referenced to an outside authority to ensure more reliable interactions between applications. We do not describe this mechanism further, except to say that it must be carefully managed. In the WFS, which by definition is about communication between systems, SRIDs are not used, but similar principles apply.

The Spatial DataBlade implements many functions to operate on its data types. The most important of these are the spatial operators, which are the predicate functions that go in the WHERE clause to support queries that select rows on the basis of spatial criteria. (For more information, see Chapter 6, “An extensible architecture for robust solutions” on page 219.) Because the WFS standard builds on the SFS, it is not surprising to find that the set of spatial operators in the WFS matches the set in the SFS. These operators are described in Table 8-2 on page 339. Of course, these operators are backed up by a capable spatial index (the R-tree index), without which none of this would be of any practical interest.

The Geodetic DataBlade

In the preceding discussion, we carefully avoid one serious problem, which we now acknowledge is a major limitation of all GIS software. That is the assumption that a map is a faithful representation of the earth as a flat earth. In many cases, the assumption is not as bad as it sounds. If you are responsible for a city or county, the curvature of the earth is unlikely to cause any problems in software that represents coordinates as X and Y in a well-chosen map projection.

However, all projections from the round surface of the earth to the flat plane of a map introduce distortions, and these become more severe as the area covered by the projected reference system grows. For example, Greenland, which is about one-quarter the land area of Brazil, looks as big as South America on many world maps. Naturally, this causes more frequent problems now that organizations are consolidating data from what used to be isolated, mostly local projects into seamless enterprise databases.

Finally, and more dramatically, all maps have edges. If you travel far enough in one direction, eventually you fall off the edge. No map can account for the fact that you can travel around the world in any direction without ever encountering an edge, and this is why no map can serve as a useful model for global applications.

None of this is helped by substituting the usual global coordinates, latitude and longitude (usually in reverse order), for the X and Y of a projected map

coordinate reference system. This is because the latitude and longitude are angles and do not resemble at all the rectilinear measures encoded in X and Y. Point locations expressed in latitude and longitude are always valid and will be plotted in the right place on any map based on them (no matter how distorted that map looks). Meanwhile, serious problems arise when you connect those points to draw a line or polygon boundary, or calculate a distance.

Figure 8-3 illustrates one of the problems introduced by assuming the earth is flat, resulting in erroneous distance measurements along nonsensical trajectories. The straight-line distance depicted in view (a), on the left, is meaningless. Not only does it traverse most of the northern hemisphere when a much shorter path is available, the actual path described by the straight line is not particularly well defined and utterly dependent on the particular projection used. In view (b), on the right, the path shown is truly the great-circle, shortest path on the globe, which is shown in this picture (itself a projection, but only for display purposes) as a curved line. On the round earth, there is no such thing as a straight line.

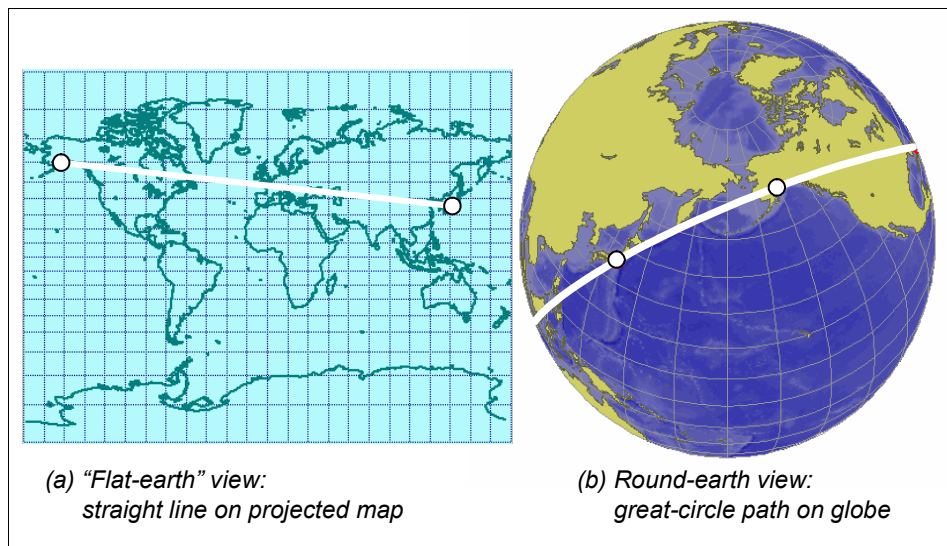


Figure 8-3 Distance from Anchorage to Tokyo from a flat earth and round earth view

Of course, this has been known for a long time, but maps are such useful and convenient tools and have served humanity so well for so long, that we tend to forget their limitations. Globes, which are arguably a more spatially accurate representation of the earth, are too difficult to carry and impossible to get at large enough scales to show sufficient detail. When software tools came along to help the geography and cartography communities, the standard practice of making maps was carried over as the fundamental paradigm for GIS software. None of

the objections to globes applies in the virtual realm of software. Only the geometric models and computations are considerably more complex. Therefore, implementing an efficient round-earth geometry engine requires mathematical skills that exceed those of most software developers.

However, it can be done. The Geodetic DataBlade incorporates a highly efficient geometric calculation engine inherently based on a round-earth model. That is the fundamental characteristic of the Geodetic DataBlade. The name refers to *geodesy*, the science and practice of measuring the shape of the earth, and is an impressive-sounding euphemism for round earth. For databases managing truly global data such as satellite imagery, weather models, or airline flight paths, there is no substitute. For less esoteric applications that still require coverage of large areas or clusters of data scattered around the globe, it can greatly improve the accuracy of spatial analysis and relieve the headaches associated with managing many different map projection-based SRSs.

There are many parallels between Spatial and Geodetic DataBlades. For the present purpose, it is sufficient to assume that they are roughly equivalent in terms of the kinds of geometric shapes they manage and the kinds of spatial predicates and operations they support. A few differences, however, are worth pointing out.

Because the SFS is inherently based on a flat-earth model, it is not possible to make a round-earth product conform to the standard without compromising the point of its existence. The Geodetic DataBlade, which predates the publication of the SFS, does not comply with it, although it does support the WKT representation. Unfortunately, this noncompliance with an accepted standard means that no mainstream GIS or mapping products directly support it.

Important: While the Geodetic DataBlade is based on a model for the earth that is a spherical shape in a three-dimensional space, it does not manage three-dimensional geometric objects (solids) or perform spatial analysis in three dimensions. All geometric shapes represented by this product are confined to the surface of the conceptual earth. Curved though it may be, that surface is only a two-dimensional space, which is why two coordinates are sufficient to locate yourself on it. The Geodetic DataBlade is *not* a 3-D product.

In addition to its round-earth approach, another aspect makes the Geodetic DataBlade unique. It incorporates the time dimension in a way that enables fast processing of queries based on both spatial and temporal criteria, by applying a single index to both dimensions. Thus, the spatial data types defined by the Geodetic DataBlade are in fact truly spatiotemporal data types. In many applications, from the changing ownership of land parcels via the satellite-based observation of natural phenomena to the tracking of moving vehicles and mobile

devices, the time dimension is at least as important as the spatial criteria. In database terms, both predicates are equally selective. This makes it difficult to optimize such queries, and the single-index approach solves this problem.

Example 8-22 shows such a spatiotemporal query. With a suitably defined R-tree index, this query on both space and time is resolved by a single index scan.

Example 8-22 A round-earth spatiotemporal query

Table definition: images(..., footprint GeoPolygon, ...)

```
SELECT * FROM images
WHERE
  Intersect
  (
    footprint,
    'GEOPOLYGON(((35,-123),(36,-122),(36,-124))),ANY,
    (2007-06-12 00:00:00.,2007-06-13 23:59:59.))'
  );
```

This query suggests the management of a library of satellite images. The search is for images whose spatial footprint (meaning, the area of the earth covered by the image) overlaps with a small area in California (latitude is the first coordinate, longitude the second) and that were taken during in a two-day period in June, 2007. Note that, by using the GeoPolygon data type, the footprint values themselves can stretch over a period of time (a time range), not just a single point in time.

As indicated previously, the unique design of the Geodetic DataBlade, with its undoubted superiority in handling certain kinds of data and scenarios, also renders it nonstandard and difficult to access using common tools. As we shall see, this is one reason why the WFS can be so valuable. It abstracts access away from the specifics of the SQL interface. If a WFS request includes a filter based on a time attribute as well as spatial criteria, the WFS implementation is free to map that filter to an integrated, high-performance Geodetic query under the covers.

Now that we have a basic understanding of the two DataBlades on which the WFS operates, we are ready to look at the WFS itself.

8.4.3 Basics of WFS

As previously noted, organizations traditionally have turned to GIS tools, such as those available from ESRI to work with spatial data managed by IDS. With the explosion of location data in today's applications, there is the need for more

platform independence and less reliance on heavy application interfaces to work with geographic data.

One of the solutions for providing this platform independence is the OGC WFS specification. It provides a generic method for accessing and creating geographic data via a Web service. A WFS provides the following capabilities:

- ▶ Query a data set and retrieve the features.
- ▶ Find the feature definition.
- ▶ Add features to a data set.
- ▶ Delete features from a data set.
- ▶ Update features in a data set.
- ▶ Lock features to prevent modification.

While a *Web Mapping Service* (WMS), another OGC specification, returns a map image to a client, the output of a WFS is an XML document that uses GML to encode the spatial information so that it can be parsed and processed by using traditional XML parsing techniques. Requests to a WFS can be made via either an HTTP GET method by using key-value pairs or an HTTP POST method by using an XML document. A WFS, backed up by a spatially enabled data server such as IDS with the Spatial DataBlade, can answer such queries as those that follow:

- ▶ Find all the Italian restaurants within one mile of my location and rank them by diner rating.
- ▶ Tell me whether my container is still in the port.
- ▶ Show all the earthquakes in this area that were above 5.0 magnitude, that occurred this century.
- ▶ Show me the areas of common bird migrations in Canada.
- ▶ Show me the areas forecasted to have severe weather in the next 24 hours.

These queries can contain purely spatial requests or be combined with traditional relational fields to form a rich query environment to map geographic data, provide location-based services, and perform complex spatial analysis.

Three types of WFS are discussed in the OGC Web Feature Service, with the Transactional extension, (OGC WFS-T) version 1.1.0 specification upon which the IBM Informix WFS DataBlade is based:

- ▶ **Basic WFS**

This is the minimum implementation. It provides for the creation of a read-only WFS that responds to queries and returns results.

- ▶ Transaction WFS

In addition to the Basic WFS, this type of WFS allows for the creation, modification, and deletion of features that are stored in the WFS. It optionally can provide support to lock a feature to prevent modification.

- ▶ XLink WFS

In addition to the Basic WFS, this WFS provides for the ability to embed XLinks (similar to hyperlinks) in the WFS. It may or may not include the ability to do Transaction operations.

The WFS specification has some specific terminology that needs to be put in database terms for the discussion here. A *namespace* refers to an IDS database, a *feature type* refers to a table, a *feature* refers to a row, and a *feature id* refers to a table primary key. All features presented by the WFS must be uniquely identifiable, and the identifier must consist of a single column primary key.

The WFS DataBlade implements a Transaction WFS that supports the following operations:

- ▶ GetCapabilities
- ▶ DescribeFeatureType
- ▶ GetFeature
- ▶ Transaction

The GetCapabilities operation is responsible for defining what the service can provide. It lists the operations that it supports, the spatial types it can operate on, query predicates, output formats, and feature types that are available in that particular instance. Example 8-23 on page 335 shows the two methods of requesting a GetCapabilities document as well as a possible response.

In the XML response, the following sections are shown:

- ▶ ServiceIdentification

This section provides information about the WFS service itself.

- ▶ ServiceProvider

This section provides information about the organization operating the WFS server.

- ▶ Operation

This section provides metadata about the operations implemented by this WFS service.

- ▶ FeatureTypeList

This section defines the list of feature types that are available. It also shows the operations and the probable location of spatial data for each feature type.

► SupportsGMLObjectType

This section shows the list of GML objects that it supports. Currently only simple features, such as points, lines and polygons, are supported by this implementation. Complex GML types are not yet supported.

► FilterCapabilities

This section defines which Common Query Language (CQL) and spatial operations are supported in this WFS instance.

This operation is typically used by geospatial mapping tools that contain WFS clients to determine what is available from the offered service. It also provides information to a prospective consumer about the data to see the types of data sets that are offered and the kinds of query capabilities they can use against the service.

Example 8-23 WFS GetCapabilities invocation and response

HTTP GET Method:

```
http://wfs.somegeo.net/geoevents/wfsdriver?SERVICE=WFS&VERSION=1.1.0&REQUEST=GetCapabilities
```

HTTP POST Method:

```
<?xml version="1.0" ?>
<wfs:GetCapabilities
  service="WFS"
  version="1.1.0"
  xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/wfs ../wfs/1.1.0/WFS.xsd"
</wfs:GetCapabilities>
```

Sample response from GetCapabilities request:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<wfs:WFS_Capabilities version="1.1.0" updateSequence="0"
  xmlns:ows="http://www.opengis.net/ows"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/wfs
  http://schemas.opengis.net/wfs/1.1.0/wfs.xsd">
<ows:ServiceIdentification>
  <ows:ServiceType>WFS</ows:ServiceType>
```

```

    <ows:ServiceTypeVersion>1.1.0</ows:ServiceTypeVersion>
    <ows:Title>IBM Informix Web Feature Service</ows:Title>
    <ows:Abstract>This is a test abstract value for the IBM Informix WFS
Datablade</ows:Abstract>
    <ows:Fees>None</ows:Fees>
    <ows:AccessConstraints>None</ows:AccessConstraints>
  </ows:ServiceIdentification>
  ...
  <wfs:FeatureTypeList>
  <wfs:FeatureType xmlns:gatgr="http://wfs.somegeo.net/geoevents">
    <wfs:Name>geoevents:congress110</wfs:Name>
    <wfs:Title>congress110</wfs:Title>
    <wfs:Abstract>No abstract provided</wfs:Abstract>
    <ows:DefaultSRS>EPSG:4326</ows:DefaultSRS>
    <ows:WGS84BoundingBox>
      <ows:LowerCorner>-179.147340 17.884813</ows:LowerCorner>
      <ows:UpperCorner>179.778470 71.352561</ows:UpperCorner>
    </ows:WGS84BoundingBox>
    <wfs:OutputFormats>
      <wfs:Format>text/xml; subtype=gml/3.1.1</wfs:Format>
      <wfs:Format>text/xml; subtype=gml/2.1.2</wfs:Format>
    </wfs:OutputFormats>
    <ows:Operations>
      <Query />
      <Insert />
      <Update />
      <Delete />
    </ows:Operations>
  </wfs:FeatureType>
  </wfs:FeatureTypeList>
  ...
</wfs:WFS_Capabilities>

```

The DescribeFeatureType operation is responsible for defining the data types found in each type of feature that is served in the WFS. Its output is an XML based schema that can be used to correctly identify each field in the feature returned. The DescribeFeatureType operation can be called with zero to many type names. If no type name is provided in the request, a description of all feature types are returned to the user.

Example 8-24 shows a sample of how to invoke DescribeFeatureType for HTTP GET/POST methods, as well as an example response. In the XML response, each field is mapped to the appropriate XML schema type. All IDS built-in types are supported with the exception of CLOB, BLOB, BYTE, TEXT, LIST, SET and

unnamed ROW types. The UDTs used by the Spatial and Geodetic DataBlades are mapped to the appropriate GML type declarations. Other UDTs are mapped to their respective character representations.

Example 8-24 WFS DescribeFeatureType invocation and response

HTTP GET Method:

```
http://wfs.somegeo.net/polboundaries/wfsdriver?SERVICE=WFS&VERSION=1.1.0&REQUEST=DescribeFeatureType&TYPENAME=congress110
```

HTTP POST Method:

```
<?xml version="1.0" ?>
<wfs:DescribeFeatureType
  service="WFS"
  version="1.1.0"
  xmlns:polboundaries="http://wfs.somegeo.net/polboundaries"
  xmlns:wfs="http://www.opengis.net/wfs"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/wfs
  ../wfs/1.1.0/WFS.xsd">
  <wfs:TypeName>congress110</wfs:TypeName>
</wfs:DescribeFeatureType>
```

Sample response:

```
<?xml version="1.0">
<xsd:schema targetNamespace="http://wfs.somegeo.net/polboundaries"
  elementFormDefault="qualified" version="0.1">
<xsd:import namespace="http://www.opengis.net/gml"
  schemaLocation="http://schemas.opengis.net/gml/3.1.1/base/gml.xsd"/>
<xsd:element name="congress110" type="polboundaries:congress110_Type"
  substitutionGroup="gml:_Feature"/>
  <xsd:complexType name="congress110_Type">
    <xsd:complexContent>
      <xsd:extension base="gml:AbstractFeatureType">
        <xsd:sequence>
          <xsd:element name="se_row_id" type="xsd:integer"
            minOccurs="0" maxOccurs="1"/>
          <xsd:element name="state" nillable="true" minOccurs="0"
            maxOccurs="1">
            <xsd:simpleType>
              <xsd:restriction base="xsd:string">
                <xsd:maxLength value="2"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:element>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>
```

```

        </xsd:element>
        <xsd:element name="cd" nillable="true" minOccurs="0"
maxOccurs="1">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:maxLength value="2"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:element>
        <xsd:element name="lsad" nillable="true" minOccurs="0"
maxOccurs="1">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:maxLength value="2"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:element>
        <xsd:element name="name" nillable="true" minOccurs="0"
maxOccurs="1">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:maxLength value="90"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:element>
        <xsd:element name="lsad_trans" nillable="true"
minOccurs="0" maxOccurs="1">
            <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                    <xsd:maxLength value="50"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:element>
        <xsd:element name="boundary"
            type="gml:MultiPolygonPropertyType" nillable="true"
            minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
</xsd:schema>

```

The GetFeature operation is the heart of the WFS server. It is responsible for retrieval and presentation of the data requested. GetFeature requests are based

on CQL (another OGC specification). A GetFeature request can consist of one to many queries that will be returned as a unioned set in the XML response. For each Query element, the property names (column names), the feature type (table name), and sort order can be requested. It is important to note that joins between tables are not supported. The query must resolve to a single table. The filter element in Query can contain both spatial and relational predicates, as shown in Table 8-2.

Table 8-2 CQL predicate operators valid in WFS 1.1

CQL predicate	SQL equivalent or definition
PropertyIsLessThan	<
PropertyIsGreaterThan	>
PropertyIsLessThanOrEqualTo	<=
PropertyIsGreaterThanOrEqualTo	>=
PropertyIsEqualTo	=
PropertyIsNotEqualTo	!=
PropertyIsLike	LIKE
PropertyIsBetween	BETWEEN
PropertyIsNull	IS NULL
BBOX	This is similar to an intersection operation performed on a polygon that is defined by a spatial bounding box bordering the southwest and northeast corners of the target area.
Equals	This is a test for spatial equality.
Disjoint	This is a test to see if two geometries do not intersect, overlap, or touch each other.
Intersect	This is a test to see if two geometries touch or intersect. This is equivalent to Not Disjoint(...).
Touches	This is a test to see if the interiors of the two geometries do not intersect and the boundary of either geometry intersects the other's interior or boundary.
Crosses	This is a test to see whether the boundaries of two geometries cross each other. This test is typically done between the spatial types of MultiPoint/LineString, LineString/LineString, MultiPoint/Polygon, and LineString/Polygon.

CQL predicate	SQL equivalent or definition
Contains	This is a test to see if the second geometry is completely contained by the first geometry provided. The boundary and interior of the second geometry are not allowed to intersect the exterior of the first geometry.
Within	This is a test to see if the first geometry is completely within the second geometry. The boundary and the interior of the first geometry are not allowed to intersect the exterior of the second geometry.
Overlaps	This is a test to see if the intersections of the geometries result in a value of the same dimension as the geometries that is different from both of the geometries. For example, if the intersection of two polygons results in a polygon, then the Overlaps() operation returns a true value, but if a point or linestring is generated from the intersection, then the operation returns a false value.
DWithin	This is a test to find objects that are within a stated distance of the geometry provided.
Beyond	This is a test to see if the objects are outside the stated distance. It is equivalent to saying Not DWithin(...).
And	This a logical operator to combine two CQL predicates to test to see if both operations are true. It is equivalent to the SQL AND operation.
Or	This is a logical operator to test two CQL predicates to see if either operation is true.
Not	This is a logical operator that functions as a negator to any operation listed in this table, with the exception of GmlObjectId.
GmlObjectId	This operator takes a specific feature identifier as an attribute. It can be used to retrieve a specific row from your feature table.

In addition to the CQL operators listed in Table 8-2, a GetFeature request can take additional parameters, which are listed in Table 8-3.

Table 8-3 GetFeature parameters valid in WFS 1.1

GetFeature parameter	Description
OUTPUTFORMAT	This parameter is used to control the format of the output.
RESULTTYPE	This parameter is used to determine whether the actual features are returned (<i>results</i>) or if a <i>count</i> of qualifying features should be returned.
MAXFEATURES	This parameter is used to control the maximum number of features returned.

GetFeature parameter	Description
TYPENAME	A list of type names to query. This is the only mandatory element to a query. If no other parameters are specified, all feature instances are returned in the response document.
FEATUREID	This parameter can be an enumerated list of feature instances to be returned. They are specified by their feature identifier.
FILTER	This describes a set of features to retrieve. The filter is defined using the predicates defined in Table 8-2 on page 339. There should be one filter specification for each type name requested in the TYPENAME parameter. This parameter is mutually exclusive with FEATUREID and BBOX.
BBOX	This parameter is a comma-separated list of coordinates that represent the southwest and northeast corners of a spatial bounding box. It is mutually exclusive with FEATUREID and FILTER.
SORTBY	This parameter is used to specify the property names (column names) by which the feature result list should be ordered when returned in the XML response document.
PROPERTYNAME	This parameter is used to list the property names (column names) that should be presented in the XML response document. The default is all properties.

Example 8-25 show examples of the two possible forms for a GetFeature request. The first example encodes the transaction by using the KVP syntax that can be sent by an HTTP GET method, while the second example shows an XML document that can be sent via a HTTP POST method. In both examples, the feature type *quakes* is being queried via a spatial bounding box for a maximum of 200 features. The output is returned in an XML document.

Example 8-25 WFS GetFeature example

HTTP GET Method:

```
http://wfs.somegeo.net/geoevents/wfsdriver?SERVICE=WFS&VERSION=1.1.0&REQUEST=GetFeature&TYPENAME=quakes&BBOX=-85.7,30.50,-80.0,35.5&MAXFEATURES=200
```

HTTP POST Method:

```
<?xml version="1.0" ?>
<GetFeature
  xmlns="http://www.opengis.net/wfs"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  service="WFS"
  version="1.1.0"
```

```
outputFormat="GML3"
maxFeatures="200"
<Query typeName="geoevents:quakes">
  <ogc:Filter>
    <ogc:BBOX>
      <ogc:PropertyName>event</ogc:PropertyName>
      <gml:Box srsName="EPSG:4326">
        <gml:coordinates>
          -85.628,33.9166 -84.224,35.0707
        </gml:coordinates>
      </gml:Box>
    </ogc:BBOX>
  </ogc:Filter>
</Query>
</GetFeature>
```

Figure 8-4 shows a sample WFS GetFeature transaction that was entered into a browser. We have shown a sample section of the XML output, for familiarity with the format of the output. All documents that are returned from a successful GetFeature operation have the number of features returned, a time stamp, and a bounding box element that shows the entire spatial extent from all feature types requested. This spatial extent is based on all of the type names (table names) requested and not purely on the result of the transaction. Each row that satisfies the query is returned as a gml:featureMember with its unique identifier.


```

<?xml version="1.0" encoding="ISO-8859-1" ?>
- <wfs:FeatureCollection xmlns="http://192.168.0.2:8080/gatgr" xmlns:gatgr="
  xmlns:gml="http://www.opengis.net/gml" xmlns:xsi="http://www.w3.org/
  http://192.168.0.2:8080/gatgr/wfsdriver?SERVICE=WFS&VERSION=1.1
  http://www.opengis.net/wfs http://schemas.opengis.net/wfs/1.1.0/v
  http://schemas.opengis.net/gml/3.1.1/base/feature.xsd" numberOfFea
- <gml:boundedBy>
- <gml:Envelope srsName="EPSG:4326">
  <gml:lowerCorner>-179.147339953 17.8848130299</gml:lowerCorner>
  <gml:upperCorner>179.778469974 71.352560638</gml:upperCorner>
  </gml:Envelope>
</gml:boundedBy>
- <gml:featureMember>
- <congress110 gml:id="congress110.1">
  <gatgr:state>01</gatgr:state>
  <gatgr:cd>05</gatgr:cd>
  <gatgr:lsad>C2</gatgr:lsad>
  <gatgr:name>5</gatgr:name>
  <gatgr:lsad_trans>Congressional District</gatgr:lsad_trans>
- <gatgr:boundary>
- <gml:MultiPolygon srsName="EPSG:4326" srsDimension="2">
- <gml:PolygonMember>
- <gml:Polygon srsName="EPSG:4326" srsDimension="2">
- <gml:exterior>
- <gml:LinearRing>
  <gml:posList dimension="2">-86.9056539628 34.453554027
    86.9056150708 34.4542220655 -86.9053139928 34.46!

```

Figure 8-4 Sample WFS GetFeature output shown in Microsoft Internet Explorer®

The Transaction operation, as the name suggests, allows for the creation (INSERT), modification (UPDATE), and deletion (DELETE) of features that are stored in the WFS. The INSERT and UPDATE Transaction operations are required to be formed by using an XML document, while DELETE has the ability to be sent via GET using key-value pairs. Example 8-26 on page 344 shows three sample operations that can be done and a sample of the XML response that comes back from a Transaction operation.

The first example inserts a row into a table that contains restaurant ratings. In this statement, we request that the database generate a new value for the feature identifier. This identifier is returned as part of the TransactionResponse XML document in *InsertResults* returned at the end of the transaction. All fields in the table must be specified in the INSERT transaction. When you want to specify a NULL field, specify an empty tag (for example *<column_name />*). You can specify one to many feature instances in each insert transaction.

Important: Tables in which new identifiers will be requested via a WFS INSERT transaction must contain either an *IDS serial* or *serial8* column as the primary key for the table.

The second example shows an UPDATE transaction where the location of a given feature instance has changed. The UPDATE transaction must contain at least one Property tag that contains a Name/Value pair to indicate the column and value that is being changed in the feature instance. The predicate uses CQL similar to the GetFeature request. In this example, we retrieve the row based on a feature identifier (*natlrestaurants.43546*). The number of rows updated by the transaction is displayed in the XML TransactionResponse document. If no rows qualify for the filter, it is not considered an error, and zero is returned for the number of rows updated.

The third example shows a DELETE transaction where we remove the identified restaurant from our feature type. The DELETE transaction only needs to contain a predicate (Filter) that uses CQL similar to the GetFeature request. Like the UPDATE example discussed earlier, we retrieve the row based on a feature identifier (*natlrestaurants.44223*). The number of rows removed from the table is displayed in the XML TransactionResponse document. As with UPDATE, if no rows qualify for the filter specified, it is not considered an error. Unlike INSERT and UPDATE, since DELETE only requires a FILTER specification, it can be specified by using the KVP syntax.

Example 8-26 WFS Transaction operations

```

<?xml version="1.0" ?>
<wfs:Transaction
  service="WFS"
  version="1.1.0"
  ...
  <wfs:Insert idgen="GenerateNew" handle="Stmt 1"
    <natlrestratings>
      <id/>
      <restaurant_id>43546</restaurant_id>
      <foodquality>3</foodquality>
      <servicequality>3</servicequality>
      <comments>Lasagna is great</comments>
      <visit_time>2007-10-07T19:35:00.00000</visit_time>
    </natlrestratings>
  </wfs:Insert>
  <wfs:Update typeName="natlrestaurants">
    <wfs:Property>
      <wfs:Name>mobilrater:locn</wfs:Name>
      <wfs:Value>
        <gml:Point>
          <gml:pos>-86.145633 37.325453</gml:pos>
        </gml:Point>
      </wfs:Value>
    </wfs:Property>
  </wfs:Update>
</wfs:Transaction>

```

```

    <ogc:Filter>
      <ogc:GmlObjectId gml:id="natlrestaurants.43546"/>
    </ogc:Filter>
  </wfs:Update>
  <wfs:Delete typeName="natlrestaurants">
    <ogc:Filter>
      <ogc:GmlObjectId gml:id="natlrestaurants.44223"/>
    </ogc:Filter>
  </wfs:Transaction>

```

Sample response:

```

<?xml version="1.0" ?>
<wfs:TransactionResponse
  version="1.1.0"
  ...
  <wfs:TransactionSummary>
    <wfs:totalInserted>1</wfs:totalInserted>
    <wfs:totalUpdated>1</wfs:totalUpdated>
    <wfs:totalDeleted>1</wfs:totalDeleted>
  </wfs:TransactionSummary>
  <wfs:InsertResults>
    <wfs:Feature handle="Stmt 1">
      <ogc:FeatureId fid="natlrestratings.132457364"/>
    </wfs:Feature>
  </wfs:InsertResults>
</wfs:TransactionResponse>

```

To better understand how all this fits together, see Figure 8-5 for a description of the architecture and processing flow that occurs.

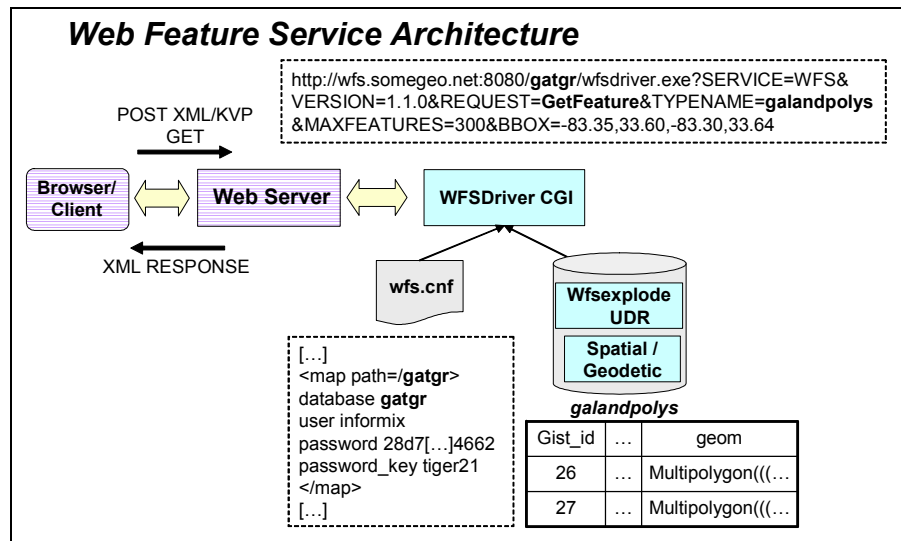


Figure 8-5 WFS DataBlade implementation

A typical WFS conversation has the following flow as illustrated in Figure 8-5:

1. The Web server (IBM HTTP Server or other Common Gateway Interface (CGI) compliant server) receives the HTTP GET/POST request from a client. A client can be a Web browser or custom program.
2. The Web server examines its configuration file (for example, httpd.conf) and invokes the CGI program **wfsdriver** (wfsdriver.exe on the Windows platform).
3. The CGI program reads the configuration file (wfs.cnf) and obtains information about the location of IDS client libraries, database name, user name, and the encrypted password with which to connect to the database.
4. The CGI program connects to IDS and calls the **WFSExplode()** UDR in the shared object library (wfs.bld).
5. The **WFSExplode()** UDR processes the transaction and formats an XML document to be returned to the **wfsdriver** CGI program. If any errors occurred during processing, an XML error document is returned.
6. The XML document is returned to the client.

8.4.4 Installing and setting up WFS

The Web Feature Service DataBlade is installed when you perform a default installation or custom installation that includes the built-in DataBlades in the IDS installer.

To create a WFS service:

1. Install IDS 11 or later.
2. Install Informix Client Software Developer Kit (SDK) version 3 or later.
3. Install any companion blade that works with WFS, which is Spatial DataBlade version 8.21.xC1 (or later) or Geodetic DataBlade version 3.12.xC1 (or later). It is not mandatory for either of these blades to be installed for WFS to be functional. Prior releases of the Spatial DataBlade and Geodetic DataBlade lack the GML publishing or parsing functions to properly interoperate with WFS.
4. Install a CGI-compliant Web server such as IBM HTTP Server.
5. Create a database with logging enabled.
6. Create a smart large object space of at least 50 MB. This is because a temporary large object is instantiated if the return is larger than 32K. This is also required by either the Spatial or Geodetic DataBlade prior to registration. Example 8-27 shows an example command to create the smart large object space.

Example 8-27 Command to create the smart-object space

```
onspaces -c -S sbspace -p C:\IFMXDATA\demo_tcp\sbspace.000 -o 0 -s 50000
```

7. Register the WFS DataBlade module and either the Spatial or Geodetic DataBlades as shown in Example 8-28.

Example 8-28 Registering the WFS and Spatial DataBlades

```
demo11_tcp>list wfsdemo
There are no modules registered in database wfsdemo
demo11_tcp>show modules
      ifxrltree.2.00          LLD.1.20.TC2
      mqblade.2.0           binaryudt.1.0
      bts.1.00              wfs.1.00.TC1
      ifxbuiltins.1.1       Node.2.0 c
      geodetic.3.12.TC1     spatial.8.21.TC1
A 'c' indicates DataBlade module has client files.
If a module is not found, check the prepare log.
demo11_tcp>register wfs.1.00.TC1 wfsdemo
```

```

Register module wfs.1.00.TC1 into database wfsdemo? [Y/n]Y
Registering DataBlade module... (may take a while).
DataBlade wfs.1.00.TC1 was successfully registered in database
wfsdemo.
demo11_tcp>register ifxrltree.2.00 wfsdemo
Register module ifxrltree.2.00 into database wfsdemo? [Y/n]Y
Registering DataBlade module... (this may take a while).
DataBlade ifxrltree.2.00 was successfully registered in database
wfsdemo.
demo11_tcp>register spatial.8.21.TC1 wfsdemo
Register module spatial.8.21.TC1 into database wfsdemo? [Y/n]Y
Registering DataBlade module... (may take a while).
DataBlade ifxrltree.2.00 was successfully registered in database
wfsdemo.
demo11_tcp>bye

```

8. Create a directory that has the same name as your database to be used by the CGI program in which the Web server is installed. This directory contains a copy of the **wfsdriver** configuration file (wfs.cnf) and the wfsdriver CGI program.
9. Run **wfssetup**, which is located in the \$INFORMIXDIR/extend/wfs.1.00.xC1/wfsdriver directory in your installation. This is a UNIX script. Therefore, users on Microsoft Windows must create the wfs.cnf manually, like the one shown in Example 8-29. This file must be placed with a copy of wfsdriver/wfsdriver.exe in the CGI directory created in step 8.

Example 8-29 Sample wfs.cnf file

```

<global>
debug_file C:\temp\wfsdriver.log
debug_level 2
</global>
<setvar>
INFORMIXDIR C:\PROGRA~1\IBM\IBMINF~1\11.10
INFORMIXSERVER demo11_tcp
</setvar>
<map path=/wfsdemo>
database wfsdemo
user informix
password 3b7d082236d445796cdfc33377400d699
password_key demokey
</map>

```

The password_key value shown in Example 8-29 can be generated by using the **wfspwcrypt** utility found in the same wfsdriver directory mentioned earlier in this step. Example 8-30 shows the usage for this utility.

Example 8-30 Usage and example for the wfspwcrypt utility

Usage for wfspwcrypt:

```
wfspwcrypt <database> <user> <key>
```

Example:

```
$ cd $INFORMIXDIR/extend/wfs.1.00.UC1/wfsdriver
$ wfspwcrypt wfsdemo informix demokey
```

```
Enter password for user 'informix': *****
Enter password again: *****
```

```
password 5b29c069a79c2c9efc9f366a4d08c15e
password_key demokey
```

Important: If you are using Microsoft Windows as the Web server, you must ensure that the INFORMIXDIR and INFORMIXSERVER values declared in the wfs.cnf match the values as set by using the **setnet32** utility. This utility is included in both the Informix-Connect and Informix Client SDK.

10. Configure the Web server for the address:port that you want your service to be broadcast on and include the ScriptAlias line similar to the one shown in Example 8-31.

Example 8-31 ScriptAlias entry in httpd.conf

```
ScriptAlias /wfsdemo "C:\Program Files\IBMHttpServer/wfsdemo"
```

11. If you have existing data that you want to present, skip to the next step. In Appendix A, "Additional material" on page 465, we provide links to sample data for you to download. This data is compatible with the Spatial DataBlade. To load it, you use the **loadshp** utility.

Attention: The shapefile utilities **loadshp**, **infoshp**, and **unloadshp** require the environment variable LD_LIBRARY_PATH in the UNIX environment. These utilities are installed with the Spatial DataBlade and placed in the \$INFORMIXDIR/bin directory.

Example 8-32 shows a **loadshp** session. For further information about the shapefile utilities, see Appendix A of the *IBM Informix Spatial DataBlade Module User's Guide*, G229-6405.

Example 8-32 The loadshp utility to load sample data

Usage for loadshp:

```
loadshp -o { create | init } -l <tablename,colname> -f <filename>
      -D <database> [ -s <server> ] [ -u <username> -p <password>
]
      [ -b <begin_row> ]
      [ -e <end_row> ]
      [ -c <commit_interval> ]
      [ -in <dbspace> ]
      [ -put <sbspace_list> ]
      [ -ext <initial_extent_size> ]
      [ -next <next_extent_size> ]
      [ -ic ]
      [ -noidx ]
      [ -srid <srid> ]
      [ -log [ <directory> ] ]
```

```
loadshp -o create -l landplaces,geom -f places -D wfsdemo -in
datadbs -srid 4
```

Beginning transaction.

```
Committing after row 1000 of 23435.
Committing after row 2000 of 23435.
Committing after row 3000 of 23435.
Committing after row 4000 of 23435.
Committing after row 5000 of 23435.
```

...

```
Committing after row 20000 of 23435.
Committing after row 21000 of 23435.
Committing after row 22000 of 23435.
Committing after row 23000 of 23435.
Committing work.
```

Inserted 23435 row(s).

Rejected 0 row(s).

Building B-tree index on column landplaces.se_row_id.


```
Defining primary key constraint on column landplaces.se_row_id.  
Building R-tree index on column landplaces.geom.  
Updating statistics for table landplaces.  
Elapsed time 0:01:02.7
```

12. If your table has a spatial column type, declare the table in `sde.geometry_columns` via an INSERT statement. This table is created when either the Spatial or Geodetic DataBlade is registered to the database. This allows WFS to understand the spatial reference system that is being used in that particular table. The **loadshp** utility, which is included with the Spatial DataBlade, automatically creates a row in this table when tables are created or initialized. The table contains the following columns:
- `f_table_catalog`
This column typically contains the name of the catalog (database) when used with software such as ESRI ArcGIS. For a table to be enabled with WFS, this value needs to be set to WFS.
 - `f_table_schema`
This column contains the schema owner name for the table.
 - `f_table_name`
This column contains the table name.
 - `f_geometry_column`
This column stores the point, linestring or polygon. If there are multiple such columns in the table, there must be one row in this table per column.
 - `storage_type`
This column can be set to NULL.
 - `geometry_type`
This column must be set to the type of geometry stored in the column. The following, more common values are valid for this column. For a more complete listing, refer to Appendix F of the *IBM Informix Spatial DataBlade Module User's Guide*, G229-6405.
 - 1** ST_Point
 - 3** ST_LineString
 - 5** ST_Polygon
 - 11** ST_MultiPolygon
 - `coord_dimension`
This column can be set to NULL.

- `srid`

This column contains the spatial reference system in which the geometry is created. In the Spatial DataBlade, this value must be contained in the *sde.spatial_references* table. In the Geodetic DataBlade, this value must be contained in the *geospatialref* table. These tables are created when either the Spatial or Geodetic DataBlades are registered to the database. In Example 8-33, we use `srid=4`, which corresponds to the unprojected WGS-84 latitude/longitude coordinate reference system.

Example 8-33 Inserting a row into the geometry_columns table for WFS

```
INSERT INTO 'sde'.geometry_columns
VALUES('WFS','informix','landplaces', 'geom', NULL, 1, NULL, 4);
```

1 row(s) inserted.

In Example 8-32, we used the **loadshp** utility to load a data set. It automatically created an entry in *sde.spatial_references* that we can modify with a simple UPDATE statement to change the *f_table_catalog* column to be the value WFS.

13. Register the table (feature type) via the WFSRegister() UDR (Example 8-34 on page 353). This UDR examines the structure of the table requested. It ensures that the table contains a single column primary key and does not contain any columns that cannot be mapped to the XML schema response, such as those discussed in the DescribeFeatureType transaction in Example 8-24 on page 337. If a spatial data type column is displayed in the table, it must be described in the *sde.geometry_columns* table, which we described earlier. It is also important that an RTree index be created for this column, so that the spatial bounding box for the table can be known.

If all these conditions are met, WFSRegister() places a row in the table *wfstables*. This row also contains the following three columns of metadata that can be updated via an SQL UPDATE statement. This metadata is in the XML response for a GetCapabilities transaction.

- `tab_title`

This is the title of the table that is presented in a geospatial mapping tool.

- `tab_abstract`

This is an abstract that can contain a more elaborate description of the table.

- `tab_keywords`

These are keywords to aid in searching the catalog.

Example 8-34 WFSRegister and updating table metadata

```
EXECUTE FUNCTION WFSRegister('landplaces');

(expression)
OK

UPDATE wfstables SET tab_title='US Landmarks from 2000 Census',
tab_abstract = 'This dataset contains all US Landmarks from the 2000
Census.', tab_keywords = SET{'Landmarks','US','2000','Points'}
WHERE regtabname = 'landplaces';

1 row(s) updated.
```

Similar metadata fields are found in the wfsmetaserviceid table. These fields are displayed in the ServiceIdentification section in the XML response for a GetCapabilities transaction.

14. Verify connectivity by using a GetCapabilities request from your Web browser. The request has the following format:

```
http://yourhostname:[http_port]/map
name/wfsdriver[.exe]?SERVICE=WFS&VERSION=1.1.0&REQUEST
=GetCapabilities
```

8.4.5 Using WFS

There are several ways that you can use the WFS type of service. A common application is *geospatial mapping*, where features that are stored in the database can be sent to a tool such as The Carbon Project's Gaia 3, which you can download from the following Web address:

<http://www.thecarbonproject.com/gaia.php>

While Gaia cannot connect to databases, it knows about OGC Web Services, such as WFS, WMS, and Web Coverage Service (WCS), and mapping services, such as Yahoo! Maps and Microsoft Virtual Earth™. It also knows about processing common geospatial file formats, such as GML, ESRI Shapefiles (shp), GoogleEarth (kml), AutoDesk (dxf), and MapInfo (mif). Any combination of these types can be used in layers to create a rich display for analysis and annotation.

Figure 8-6 shows a Gaia 3 window with multiple layers defined by using WFS. The base map is provided by Yahoo! Maps. The WFS layers shown are US Airports (denoted by the blue airplane symbols), Landmarks, Water Polygons and Landmark Polygons from the US Census TIGER data, as well as earthquake data from USGS. The earthquake points are stored by using the Geodetic DataBlade and scaled by their relative magnitude.

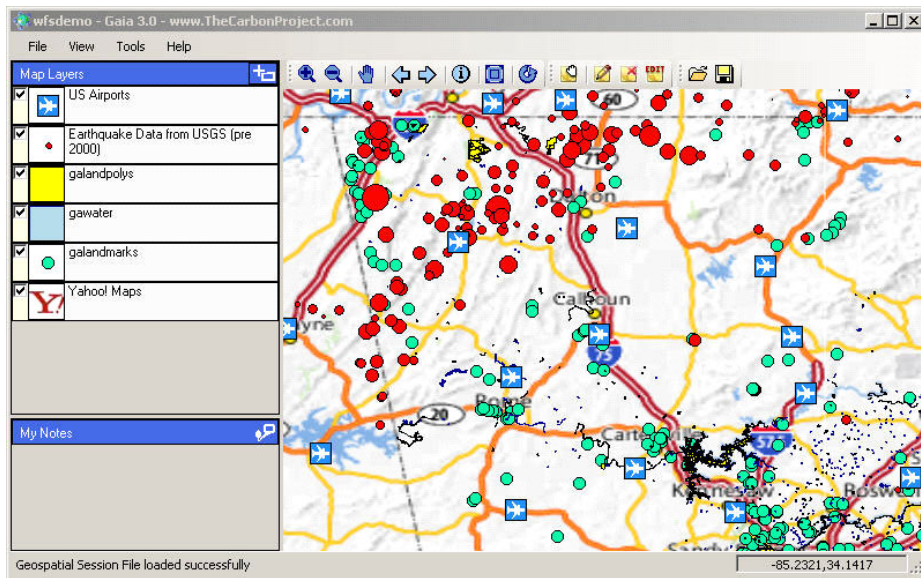


Figure 8-6 Gaia 3 window courtesy of The Carbon Project

We add these layers by selecting **Tools** → **Add Layer**. In Figure 8-7, we start with the base layer of the Yahoo! Maps, which we select from the Add Layer window. The figure shows an example of how to add the IDS WFS service to Gaia. We click the pink plus sign (+) button shown in the Add Layer to Map window. Then the Add an OGC Service to the List window opens in which we entered the following values:

- ▶ For Name, we type IDS DEMO WFS.
- ▶ For URL, we type `http://192.168.0.2:8080/wfsdemo/wfsdriver`.
- ▶ For Service Type, we select the **WFS** radio button.
- ▶ For Version, we select **1.1.0** from the drop-down list.

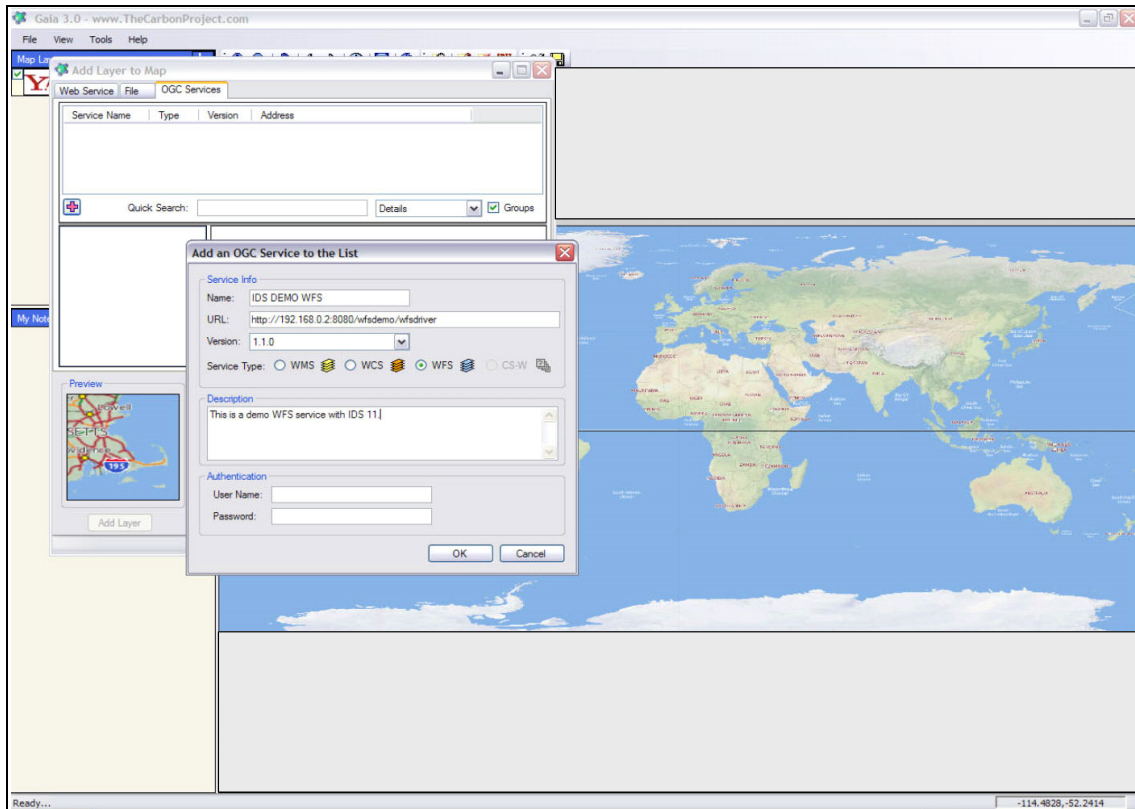


Figure 8-7 Adding the IDS WFS service to Gaia (courtesy of the Carbon Project)

We click the **OK** button at the bottom of the Add an OGC Service to the List window. Then Gaia retrieves the GetCapabilities transaction document and presents the list of layers to be selected as shown in Figure 8-8. We highlight one of the layers shown in the middle left, making sure to select **GML3** for the GML Version shown at the right. GML2 is more verbose, especially when dealing with complex polygons and can increase the amount of time needed to process the XML response. In the figure, we also limit the number of features to 100 and select **Use Bounding-Box Filter**. This limits the area in which features are displayed based on the current map view.

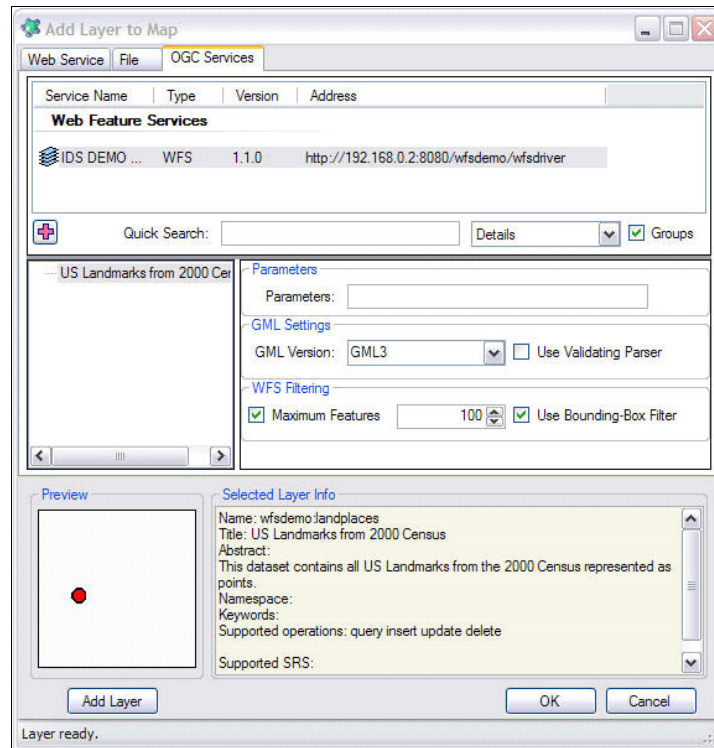


Figure 8-8 Layer presentation window from Gaia (courtesy of the Carbon Project)

While the specific details of adding services to geospatial mapping tools might vary, the basic information, such as URL, Service Type, and Version, are fairly typical across a number of geospatial mapping tools that contain a WFS client. Other tools and possibly future versions of this tool shown might enable other spatial operations and more complex queries to be specified to allow richer presentation and the ability to conduct analysis.

Location-based services using WFS

In addition to the geospatial mapping applications previously discussed, the WFS DataBlade can also be used in location-based services (LBS) applications. LBS services are based on a desired location. You can construct these applications by using the GetFeature operation and the DWithin (Distance Within) predicate.

Consider a GPS-enabled mobile phone-based application that can show you the restaurants within a given radius of your current location. These restaurants can be selected by type and ranked by user rating. Example 8-35 shows a sample DWithin query along with the SQL that is generated by the WFSExplode() UDR that such an application might use.

Example 8-35 Location-based services with DWithin

```
<?xml version="1.0" ?>
<GetFeature
  service="WFS"
  version="1.1.0"
  maxFeatures="5"
  ...
  <Query typeName="natlrestaurants">
    <ogc:PropertyName>mobilera: name</ogc:PropertyName>
    <ogc:PropertyName>mobilera: address</ogc:PropertyName>
    <ogc:PropertyName>mobilera: rating</ogc:PropertyName>
    <ogc:Filter>
      <And>
        <PropertyIsEqualTo>
          <PropertyName>mobilera: resttype</PropertyName>
          <Literal>Italian</Literal>
        </PropertyIsEqualTo>
        <DWithin>
          <PropertyName>mobilera: locn</PropertyName>
          <gml:Point>
            <gml:pos>-84.35 31.764</gml:pos>
          </gml:Point>
          <Distance units='mi'>1</Distance>
        </DWithin>
      </And>
    </ogc:Filter>
    <ogc:SortBy>
      <ogc:SortProperty>
        <ogc:PropertyName>mobilera: rating</ogc:PropertyName>
        <ogc:SortOrder>DESC</ogc:SortOrder>
      </ogc:SortProperty>
    </ogc:SortBy>
  </Query>
</GetFeature>
```

```
</Query>
</GetFeature>
```

Equivalent SQL:

```
SELECT FIRST 5 name, address, rating FROM natlrestaurants
WHERE ST_Intersects(locn, ST_Buffer(ST_GeomFromGML('<gml:Point
xmlns:gml="http://www.opengis.net/gml"><gml:pos>-84.35
31.764</gml:pos></gml:Point>',4),1609.344))
AND resttype = 'Italian'
ORDER BY rating DESC;
```

Note: In version 11, the WFS DataBlade only supports translation of the mile (mi), nautical mile (nm), meter (m), kilometer (km), and foot (ft) distance-type arguments with the DWithin and Beyond CQL predicates. In the Spatial DataBlade, the unit of measure depends on the spatial reference system specified. If the spatial reference system does not specify a linear unit of measure, it does the calculation based on degrees of latitude, which gives a less precise result.

After or during the meal, the user can upload comments, ratings, or both of the establishment by using an INSERT operation such as the one shown in Example 8-26 on page 344.

8.4.6 WFS and spatiotemporal queries

The Geodetic DataBlade is capable of not only indexing based on space, but altitude and time as well. These values are stored as ranges, with the altitude range being two double-precision values and the temporal range stored as two IDS datetime values. The values used in the ranges can be equal, but it must be a valid range (start value <= end value). The literal value ANY can be used for either range and represents an open non-null value. Consider the SQL representation of the Geodetic type GeoPoint in Example 8-36. This example shows how an earthquake might be represented, including the location, magnitude (stored in the altitude range), and time of the event.

Example 8-36 Sample GeoPoint output

```
SELECT FIRST 3 id, event FROM recent_quakes;

id      1
event  GeoPoint((52.388,177.872),(3.5,3.5),(2007-11-16
18:44:18.00000,2007-11-16 18:44:18.00000))
```



```
id      2
event  GeoPoint((-22.074,-69.628),(4.7,4.7),(2007-11-16
17:05:22.00000,2007-11-16 17:05:22.00000))
```

```
id      3
event  GeoPoint((-9.820,108.670),(4.9,4.9),(2007-11-16
15:00:04.00000,2007-11-16 15:00:04.00000))
```

3 row(s) retrieved.

In addition to the spatial extent represented by the latitude and longitude coordinates, the WFS DataBlade supports access to the additional extents (altitude and time). It represents each part of the data type as a separate XML field in the featureMember response as shown in Example 8-37.

Example 8-37 WFS GetFeature response showing GeoPoint mapping

```
<wfs:FeatureCollection ...>
  <gml:boundedBy>
    <gml:Envelope srsName="EPSG:4326">
      <gml:lowerCorner>-134.9548 -35.20453</gml:lowerCorner>
      <gml:upperCorner>45.08138 84.613659</gml:upperCorner>
    </gml:Envelope>
  </gml:boundedBy>
  <gml:featureMember>
    <recent_quakes gml:id="recent_quakes.1">
      <geoevents:event>
        <gml:Point srsName="EPSG:4326" srsDimension="2">
          <gml:pos>52.388 177.872</gml:pos>
        </gml:Point>
      </geoevents:event>
      <geoevents:event_valt_any>>false</geoevents:event_valt_any>
      <geoevents:event_valt_bottom>3.5</geoevents:event_valt_bottom>
      <geoevents:event_valt_top>3.5</geoevents:event_valt_top>
      <geoevents:event_vtime_any>>false</geoevents:event_vtime_any>
      <geoevents:event_vtime_begin>
        2007-11-16T18:44:18.00000
      </geoevents:event_vtime_begin>
      <geoevents:event_vtime_end>
```

```

                2007-11-16T18:44:18.00000
            </geoevents:event_vtime_end>
        </recent_quakes>
    </gml:featureMember>
    ...
</wfs:FeatureCollection>

```

Table 8-4 explains how each field is mapped from the GeoObject type, with *columnname* (such as event in Example 8-35) representing the name of the column that is a GeoObject data type.

Table 8-4 *GeoObject field mapping in WFS*

WFS featureMember field name	Description
columnname	Contains the spatial extent encoded in GML.
columnname_valt_any	True if the altitude range is set to the ANY value. Otherwise, it is False.
columnname_valt_bottom	Contains the bottom value of the altitude range as extracted by the Bottom() function in the Geodetic DataBlade.
columnname_valt_top	Contains the top value of the altitude range as extracted by the Top() function in the Geodetic DataBlade.
columnname_vtime_any	True if the temporal range is set to the ANY value. Otherwise, the value is False.
columnname_vtime_begin	Contains the beginning of the temporal range as extracted by the Begin() function in the Geodetic DataBlade.
columnname_vtime_end	Contains the end of the temporal range as extracted by the End() function in the Geodetic DataBlade.

These fields can be used in combination to produce complex spatiotemporal queries. When testing a range of values and using the CQL predicate PropertyIsBetween, it is only necessary to test either member of the range. The *columnname_valt_any* and *columnname_vtime_any* columns are, by default, set to ANY time range or altitude range when a spatial extent is queried.

Example 8-38 shows a GetFeature request that queries the three elements of the spatiotemporal type GeoPoint described in Example 8-37. In the example, we are searching for all earthquakes that were greater than a magnitude of 3.0, occurred between 1 January 1960 and 15 June 2001, and occurred in a specific area of the country, that is denoted by a spatial bounding box. The SQL produced by the WFSExplode() UDR combined with the Geodetic DataBlade is also shown to demonstrate how a complex spatiotemporal query is formed. When this query is executed, the R-Tree index on the *event* column is used in the space and time dimensions.

Example 8-38 GetFeature request with spatiotemporal type

```
<?xml version="1.0" ?>
<GetFeature
  service="WFS"
  version="1.1.0"
  maxFeatures="5"
  ...
  <Query typeName="geoevents:quakes">
    <Filter>
      <And>
        <PropertyIsBetween>
          <PropertyName>
            geoevents:event_vtime_begin
          </PropertyName>
          <LowerBoundary>
            1960-01-01T00:00:00.00000
          </LowerBoundary>
          <UpperBoundary>
            2001-06-15T23:00:00.00000
          </UpperBoundary>
        </And>
        <BBOX>
          <PropertyName>geoevents:event</PropertyName>
          <gml:Box srsName="EPSG:4326">
            <gml:coordinates>-83.619512081281,32.55895192298
            -83.2100178067909,32.8956934075937</gml:coordinates>
          </gml:Box>
        </BBOX>
        <PropertyIsGreaterThan>
          <PropertyName>
            geoevents:event_valt_bottom<
          </PropertyName>
          <Literal>3.0</Literal>
        </PropertyIsGreaterThan>
      </Filter>
    </Query>
  </GetFeature>
```

```
        </And>
    </Filter>
</Query>
</GetFeature>
```

SQL:

```
SELECT FIRST 5 event_id, GeoAsGML(event), IsAny(AltRange(event)),
Bottom(AltRange(event)), Top(AltRange(event)), IsAny(TimeRange(event)),
Begin(TimeRange(event)), End(TimeRange(event))
FROM quakes
WHERE Intersect(event, GeoEnvelopeFromGML('<gml:Box
srsName="EPSG:4326"><gml:coordinates>-83.619512081281,32.55895192298
-83.2100178067909,32.8956934075937</gml:coordinates></gml:Box>',0))
AND Intersect(TimeRange(event), '(1960-01-01 00:00:00.00000,2001-06-15
23:00:00.00000')::GeoTimeRange)
AND Bottom(AltRange(event)) > 3.0;
```

Summary

In this section, we have discussed enabling the publication of location-based data by using the WFS DataBlade in combination with the Spatial and Geodetic DataBlades. This opens a new capability for your business environment to use geospatial mapping tools to present colorful maps for analysis and presentation. It also gives you the ability to provide location-based services over the World Wide Web without having to learn complex spatial SQL functions and predicates or designing complex protocols for exchanging data.

8.5 Searching your database differently with Soundex

Many applications often require the need to search on character strings based on their *sound*. This functionality, called *Soundex*, has been built into search windows to help search for proper names and street addresses because spelling is often subjective, and mistakes in annotation are quite common. Unfortunately for application designers, it is usually necessary to compromise the relational integrity of the data by introducing redundant columns to tables that hold a form of Soundex code. This is further complicated by the fact that any change in the column holding the actual value requires an update to the associated sound, and every application must specify a reference to the sound column in order to search against it.

Because applications should be able to use this function as transparently as possible, automatic updating of the Soundex signature is traditionally achieved

by using triggers and stored procedures. In short, the work, which can be complex depending on the database design, is left to the database administrator.

Several algorithms have been developed for searching based on the Soundex code, such as the following examples:

- ▶ Russell Soundex
- ▶ NYSIIS
- ▶ Celko Improved Soundex
- ▶ Metaphone/Double-Metaphone
- ▶ Daitch-Mokotoff Soundex

For our example Soundex DataBlade, we use a simple consonant sound algorithm to illustrate what is possible. You can investigate these other alternative algorithms and modify the sample code accordingly to achieve the desired results in precision.

In the examples in this section, we use English surnames in an employee table to help illustrate the power of using this type of extensibility.

In Example 8-39, a standard SQL query on our example Soundex data type has asked for employees where the name equals “Smith”. Note that “Smythe,” “Smyth,” “Smithie,” and “Smithy” are all returned in addition to “Smith”. This is due to the fact that comparison takes place on the sound attributes rather than the actual text contents of the name column.

Example 8-39 SQL query for a name

```
SELECT empid, name
FROM employee
WHERE name = 'Smith'
```

empid	name
13	Smith
15	Smythe
17	Smithie
19	Smyth
20	Smithy

5 row(s) retrieved.

In the sections that follow, we demonstrate how to create this new data type along with the functions that allow it to be indexed, how to create indexes on this type, and extending the functionality even further to explore regular expression matching on the new data type.

The actual blade takes a relatively small amount of code. The majority of the code is in the sound generation function and regular expression functions. To make the example easier to follow, error checking and code path efficiencies are excluded.

8.5.1 Creating the TSndx data type

Before we can define a table that stores the character data in this fashion, we must create an opaque type to represent it. In this example, we create a new opaque type called TSndx. For simplicity, we define it as a fixed-length opaque type that can be indexed on an IDS standard 2K page size. Example 8-40 shows how we define the data structure in C.

Example 8-40 C code for TSndx

```
/* C code required to define the TSndx structure */

typedef struct {
    mi_char data[256];
    mi_char sound[30];
} TSndx;
```

We must also define this type to IDS, which we do by using the CREATE OPAQUE TYPE command, as shown in Example 8-41.

Example 8-41 TSndx type creation within IDS

```
CREATE OPAQUE TYPE TSndx (
    internallength = 286, /* 256 chars + 30 sound chars */
    alignment = 4        /* Byte alignment within the server */
);
```

All data types require a way to convert from the externally given format to an internal representation, and back again. For example, an integer stored as an SQL SMALLINT with the value 32767 does not require 5 bytes to store the value. Instead it uses 2 bytes for the value internally and converts it back to five characters when it is returned from the database.

Similarly, a spatial point, such as (40.7487, -73.986) is parsed on input and is stored as two floating point values internally. When it is retrieved from the database, the point is represented as two numbers separated by a comma surrounded by parentheses. The TSndx type requires similar functionality. To achieve this, we create an input function for computing the sound value and register it to the database as shown in Example 8-42.

Example 8-42 Source code for input function and creating of function in IDS

```
UDREXPOR TSndx *TSndxInput (mi_lvarchar *InValue)
{
    TSndx *result;
    mi_char *textval;
    mi_char *textsnd;
    mi_char *thesound;
    mi_char *Sndx(char *);

    textval = mi_lvarchar_to_string(InValue);

    result = (TSndx *)mi_alloc(sizeof(TSndx));
    strncpy(result->data, textval, strlen(textval));
    thesound = Sndx(textval);
    strncpy(result->sound, thesound, strlen(thesound));

    mi_free(thesound);
    mi_free(textval);

    return(result);
}

CREATE FUNCTION TSndxIn(lvarchar)
RETURNS TSndx
EXTERNAL NAME '$INFORMIXDIR/extend/Sndx/Soundex.bld(TSndxInput)'
LANGUAGE C;
```

Similarly, we create an output function so that the database knows which part of the opaque type to return when it is retrieved as demonstrated in Example 8-43.

Example 8-43 Source code for the output function and function creation in IDS

```
UDREXPOR mi_lvarchar *TSndxOutput(TSndx *value)
{
    return (mi_string_to_lvarchar(value->data));
}

CREATE FUNCTION TSndxOut(TSndx)
RETURNS lvarchar
EXTERNAL NAME '$INFORMIXDIR/extend/Sndx/Soundex.bld(TSndxOutput)'
LANGUAGE C;
```

One important issue remains. If we are going to use TSndx as a character column, then we almost certainly expect to use it in situations for which we use a

character type. This presents a problem because TSndx is a completely different type. And logically, the only way for us to use characters in place of TSndx is to convert every character string to a TSndx before storing it in the database.

Instead of having to convert (or cast) from TSndx to char and back again when necessary, IDS allows us to provide what are referred to as *implicit casts*. That means that in the right context, IDS does not return an incorrect data type error. Instead we give it permission to make certain assumptions and do the casting (or converting) for us. See Example 8-44.

Example 8-44 Implicit cast definition for automatic type conversion

```
CREATE IMPLICIT CAST (lvarchar AS TSndx with TsndxIn);  
  
CREATE IMPLICIT CAST (TSndx AS lvarchar with TSndxOut);
```

Now that we have defined how the data will be stored and displayed, we can create a table and insert data into it. Example 8-45 shows how to create the table with a TSndx column and the INSERT statements. We use character strings in the INSERT, even though the *name* column is of type TSndx. With the casting functions defined in Example 8-44, IDS can convert this data automatically, and the user is unaware that the name column is much more than a character type.

Example 8-45 Creating the employee table and INSERT statements

```
CREATE TABLE employee (  
    id          INTEGER,  
    name        TSndx,  
    salary      MONEY,  
    dob         DATE  
);  
INSERT INTO employee VALUES(1, 'Davis', 10000, '12/01/1981');  
INSERT INTO employee VALUES(2, 'Dilbert', 20000, '12/02/1982');  
INSERT INTO employee VALUES(3, 'Dogbert', 30000, '12/03/1983');  
INSERT INTO employee VALUES(4, 'Duffies', 40000, '12/04/1984');  
INSERT INTO employee VALUES(5, 'Genes', 50000, '12/05/1985');  
INSERT INTO employee VALUES(6, 'Jonbert', 60000, '12/06/1986');  
INSERT INTO employee VALUES(7, 'Laurence', 70000, '12/07/1987');  
INSERT INTO employee VALUES(8, 'Jones', 80000, '12/08/1988');  
INSERT INTO employee VALUES(9, 'MacLoud', 90000, '12/09/1989');  
INSERT INTO employee VALUES(10, 'Lawrence', 100000, '12/10/1990');  
INSERT INTO employee VALUES(11, 'McLewid', 110000, '12/11/1991');  
INSERT INTO employee VALUES(12, "Ratbert", 120000, '12/12/1992');  
INSERT INTO employee VALUES(13, "Smith", 130000, '12/01/1993');  
INSERT INTO employee VALUES(14, "Toffies", 140000, '12/02/1994');  
INSERT INTO employee VALUES(15, "Smythe", 150000, '12/03/1995');
```



```
INSERT INTO employee VALUES(16, "Doofus", 160000, '12/04/1996');
INSERT INTO employee VALUES(17, "Smithie", 170000, '12/05/1997');
INSERT INTO employee VALUES(18, "Jonmach", 180000, '12/06/1998');
```

If we now select from the employee table, shown in Example 8-46, we see the output equivalent as though name were a simple character type.

Example 8-46 Listing of the employee table

```
SELECT * FROM employee;
```

empid	name	salary	dob
1	Davis	\$10000.00	12/01/1981
2	Dilbert	\$20000.00	12/02/1982
3	Dogbert	\$30000.00	12/03/1983
4	Duffies	\$40000.00	12/04/1984
5	Genes	\$50000.00	12/05/1985
6	Jonbert	\$60000.00	12/06/1986
7	Laurence	\$70000.00	12/07/1987
8	Jones	\$80000.00	12/08/1988
9	MacLoud	\$90000.00	12/09/1989
10	Lawrence	\$100000.00	12/10/1990
11	McLewid	\$110000.00	12/11/1991
12	Ratbert	\$120000.00	12/12/1992
13	Smith	\$130000.00	12/01/1993
14	Toffies	\$140000.00	12/02/1994
15	Smythe	\$150000.00	12/03/1995
16	Doofus	\$160000.00	12/04/1996
17	Smithie	\$170000.00	12/05/1997
18	Jonmach	\$180000.00	12/06/1998

18 row(s) retrieved.

8.5.2 Indexing the TSndx data type

Now that we have defined how to store and display this data, we need a way to search for it. With IDS, you can overload the comparison operators that are used by the Btree index in order to build indexes on UDTs, which is simple to do. We start by supplying a comparison function and an overload for the Equal() operator, shown in Example 8-47 on page 368.

Example 8-47 Code for the compare and equals

```
/* Provide a compare function that all routines can use */
UDREXPOR mi_integer TSndxCompare(TSndx *value1, TSndx *value2)
{
    mi_integer val;
    if (val = strcmp(value1->sound, value2->sound))
        return((val > 0) ? 1 : -1);
    return(val);
}

UDREXPOR mi_boolean TSndxEqual(TSndx *value1, TSndx *value2)
{
    return((mi_boolean)(0 == TSndxCompare(value1, value2)));
}

CREATE FUNCTION Compare(TSndx, TSndx)
RETURNS integer
WITH (NOT VARIANT)
EXTERNAL NAME '$INFORMIXDIR/extend/Sndx/Soundex.bld(TSndxCompare)
LANGUAGE C;

CREATE FUNCTION Equal(TSndx, TSndx)
RETURNS boolean
WITH (NOT VARIANT)
EXTERNAL NAME '$INFORMIXDIR/extend/Sndx/Soundex.bld(TSndxEqual)
```

What is probably not evident from this example is that the normal equal (=) operator used in SQL is now available to compare sound values. The full listing of all SQL and C code in Appendix A, “Additional material” on page 465, shows the complete implementation with the addition of the overloads for >, <, >=, <=, and != operators. While it is debatable whether a string sounds better or worse than any other, if you can order the sounds in the database, then you can index them and gain performance benefits.

Creating functional indexes

As previously mentioned, IDS provides a number of indexing methods. Any ordinal value can be indexed with a standard B-Tree, as long as the compare() functions exist. You previously have seen how simple that is. Other types, such as spatial data, might use an R-Tree index, which is also provided by IDS. In this example, we do not index in more than one dimension.

Assuming that we have different criteria from the normal compare() functionality, how can we index on this value? We can do this by using the *functional indexes* feature. With functional indexes, you can create an index on the result of a

function, and our function returns an integer value representing the *sound* of the text. We can also index on the length of the name, number of consonants, and text language, but the internals are less important than the fact that you can do it. In Example 8-48, by indexing on the sound itself, we can ensure that names, such as *Duffies* and *Toffies*, are stored close to each other internally in IDS.

Example 8-48 Source for TSndxValue UDR

```
UDREXPORT mi_integer TSndxValue(TSndx *value)
{
    if (value->sound[0] == '0')
    {
        value->sound[0] = '-';
    }
    return(atoi(value->sound));
}
CREATE FUNCTION TSndxValue(TSndx)
RETURNS integer
WITH (NOT VARIANT)
EXTERNAL NAME '$INFORMIXDIR/extend/Sndx/Soundex.bld(TSndxValue)
LANGUAGE C;
```

To display the sound value for each row in the table, use the SELECT statement as shown in Example 8-49.

Example 8-49 Employee names with their sound values

```
SELECT name, TSndxValue(name) AS namesound FROM employee;
```

name	namesound
Davis	180
Dilbert	15941
Dogbert	17941
Duffies	180
Genes	620
Jonbert	62941
Laurence	5420
Jones	620
MacLoud	3751
Lawrence	5420
McLewid	3751
Ratbert	41941
Smith	-31
Toffies	180

Smythe -31
 Doofus 180
 Smithy -31

18 row(s) retrieved.

Now we can create a functional index on the table, as shown in Example 8-50.

Example 8-50 Creating a functional index

```
CREATE INDEX employee_fidx1 ON employee (TSndxValue(name))
USING BTREE;
```

Now compare how data might be stored using an index on the employee name as opposed to the an index on its associated sound. Figure 8-9 shows that the similar-sounding employees are grouped together, from a *sound perspective*, using the functional index and alphabetically using a regular index. Note that *Duffies* and *Davis* are stored near *Toffies* and *Doofus*. In the example, the spread of the data is greatly exaggerated at three rows per page to emphasize how having to search for *n* similar-sounding objects can mean having to read *n* pages. In this case, no intelligence can be built around the data type, which might mean a worst-case scenario of having to perform a full table sequential scan.

Page #	Alphabetically Ordered data		Page #	Sound of Name	Sound Ordered Data
1	Davis Dilbert Dogbert	➔	1	-31 -31 -31	Smithie Smith Smythe
2	Doofus Duffies Genes	➔	2	180 180 180	Duffies Toffies Doofus
3	Jonbert Jones Jonmach	➔	3	180 620 620	Davis Genes Jones
4	Laurence Lawrence MacLoud	➔	4	3751 3751 5420	MacLoud McLewid Lawrence
5	McLewid Ratbert Smith	➔	5	5420 6236 15941	Laurence Jonmach Dilbert
6	Smithie Smythe Toffies	➔	6	17941 41941 62941	Dogbert Ratbert Jonbert

Figure 8-9 How data might be physically stored using two different indexes

8.5.3 Extending the base functionality

We now have what is considered an acceptable solution to the sound-search function. However, we are not finished. Code is available in the public domain that we can use to extend the functionality of the new Soundex data type. For example, we can use regular expression syntax to extend the range of search possibilities on the data. Example 8-51 shows how to use a simple regular expression parser and overload the SQL LIKE operator to enable the user to use wild cards and optional sounds to make this new data type a powerful search facility.

Example 8-51 Overloading the SQL LIKE operator

```
UDREXPOR mi_boolean TSndxRE(TSndx *p_string, mi_lvarchar *p_pattern)
{
    mi_boolean    flag, strmatch(char *, char *);
    mi_char      *pattern, *patternsound;
    register     char c;

    pattern = mi_lvarchar_to_string(p_pattern);
    patternsound = Sndx(pattern);
    flag = strmatch(patternsound, p_string->sound);
    mi_free(pattern);
    mi_free(patternsound);
    return(flag);
}

CREATE FUNCTION Like(TSndx, lvarchar)
RETURNS boolean
WITH (NOT VARIANT)
EXTERNAL NAME '$INFORMIXDIR/extend/Sndx/Soundex.bld(TSndxRE)
LANGUAGE C;
```

Now that IDS knows what is meant by LIKE when referring to a TSndx data type, we can use the SQL LIKE operator in queries. Example 8-52 shows the query: "Find all names beginning with the "J" sound and ending with an "S" sound."

Example 8-52 Usage of LIKE with sound values

```
SELECT * FROM employee WHERE name LIKE 'j*s'
```

empid	name	salary	dob
5	Genes	\$50000.00	12/05/1985
8	Jones	\$80000.00	12/08/1988

2 row(s) retrieved.

Example 8-53 shows the query: “Find all names beginning with either a “J” sound or “D” sound, and containing either “F”, “L”, or “B” sounds in the middle, and ending with any sound.”

Example 8-53 A more complex LIKE query with sound values

```
SELECT * FROM employee
WHERE name LIKE '[JD]*[FLB]*';
```

empid	name	salary	dob
1	Davis	\$10000.00	12/01/1981
2	Dilbert	\$20000.00	12/02/1982
3	Dogbert	\$30000.00	12/03/1983
4	Duffies	\$40000.00	12/04/1984
6	Jonbert	\$60000.00	12/06/1986
14	Toffies	\$140000.00	12/02/1994
16	Doofus	\$160000.00	12/04/1996

7 row(s) retrieved.

The addition of a regular expression comparison as an alternative comparison operator was arbitrary. You can easily extend the comparison functions to include fuzzy matching or partial equality, for example, “Find me all the records that have an 85% match (whatever you might want that to mean).” You might also add an attribute to the data type indicating nationality, such as a Belgian application that can store the name as Flemish, German, French, or English, in which the Soundex routine is customized to work optimally based on sound rules for the language of interest. Considerations can be made for dialects as well, allowing for dropped consonants and extended vowels.

8.6 Summary

In this chapter, we have shown how you can use the power of IDS extensibility to look at your data in new ways and create data sets by using iterators. We have also shown you have can use iterator and aggregate functions to increase performance. In addition, we have shown how you can customize your environment to consume and provide Web services using IDS. In this changing IT landscape, extensibility can play a key role in reducing the time needed to produce applications yet provide rich presentation capabilities for your business needs.



Taking advantage of database events

In this chapter, we discuss the ability to add processing based on events that occur in the database. With this capability, you can better integrate the database in the architecture of a business solution. The result can be faster performance, simpler design and implementation, faster time to production, and response to business needs.

9.1 Database servers and application architectures

Database servers are not used in a vacuum, rather they are typically part of an application architecture. This architecture then places demands and requirements on the use of the database server. In a simple environment, an application might only access the database server and perhaps get or set some values. But, there are other architectures that are much more complex.

We see this in many environments. For example, application servers might run Java beans and Enterprise JavaBeans™ (EJBs) that communicate with each other based on the demands of the application. These components can be distributed over a large network and communicate with each other by using mechanisms, such as messages, or lower level communications, such as opening a network socket or sending a signal.

We also hear a lot about new types of applications, such as those called *mashups*. These applications take advantage of public interfaces that provide specific functionality to manipulate the information being processed to show it differently. Or they use such interfaces to add other information sources to complement the data and generate new information and new business insights.

The typical database server used in a software architecture can be considered an *end node*. That is, a software component sends a request to the database, and then the database accesses the appropriate tables and returns the result. In this example, the database server is a data repository and can be considered a persistence storage mechanism. In this type of environment, the database server capabilities become less and less important, with the result being that the database servers become commodities.

The Informix Dynamic Server (IDS) provides significant other capabilities that can benefit the implementation of a software architecture, as we have demonstrated in several chapters of this book. We can take it one step further and integrate the database into the flow of the architecture. Here, it is not only used as an end node, but as an intermediate node that can obtain information from multiple sources and provide a complete set of results or final result. This way, we can reduce the overall complexity of the application code since it does not have to access multiple sources and join the information together to get to the final result. In addition to reducing complexity, it can optimize the use of multiple resources, such as the network bandwidth, and increase the system's scalability. Figure 9-1 on page 375 illustrates this integrated design.

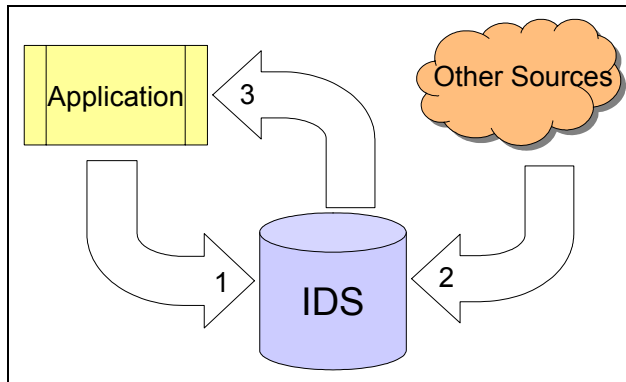


Figure 9-1 Integrated design

Figure 9-1 represents is the ability of IDS to access outside sources to complete the information provided by the database. The following types of information, as examples, can be provided:

- ▶ Validation of an address during an INSERT through a Web service
- ▶ Collection of geocoding information during a SELECT
- ▶ Discovery of any outstanding traffic violation
- ▶ Understanding credit ratings, history, or both
- ▶ Learning about creditor actions

With this information, you can create what is called a *data-centric mashup*.

The approach can also enable optimization by joining the outside data with the database tables, by taking advantage of the optimized code of the server. It can also save operations since the server can keep track of what was retrieved and not retrieve the same information multiple times.

It is common in the travel industry, for example, to have an application that needs to access another system, often a mainframe, to complete a reservation. By using this new approach, the problem can be solved as illustrated in Figure 9-2 on page 376.

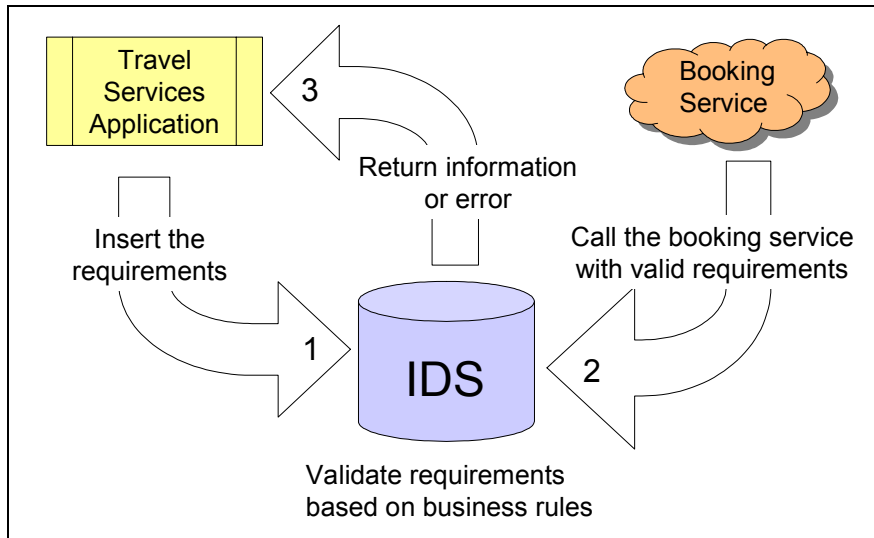


Figure 9-2 Travel service architecture

Without the events, the application does not have to access the database to validate the travel request based on the business rules established. If the request is outside the guidelines, it can either be rejected or flagged with a warning.

The application then needs to access the booking service to request the appropriate booking. This can result in an error due to no availability or a return of the itinerary. The application then must store the resulting itinerary in the database and return the information to the user.

Putting the processing in the database simplifies the application, since it eliminates multiple calls to the database. It also makes the workflow available to other applications that require it, thus reducing duplication of effort. Another benefit is that, if some of the workflow changes, such as using a different booking service or adding another step in the process, it can be changed in the database without touching the customer facing applications.

We can take this approach one step further and communicate with the outside world based on the modifications made to the database tables. In this case, we must be concerned about the completion of these modifications, which is where database events come in to play. The following types of operations can be performed:

- ▶ Placing an order to a supplier based on an inventory change
- ▶ Using outside resellers, such as amazon.com, overstock.com, and others, to sell overstock merchandise

- ▶ Sending alerts to business managers or monitoring systems based on the database activities

Figure 9-3 illustrates the communication with resellers.

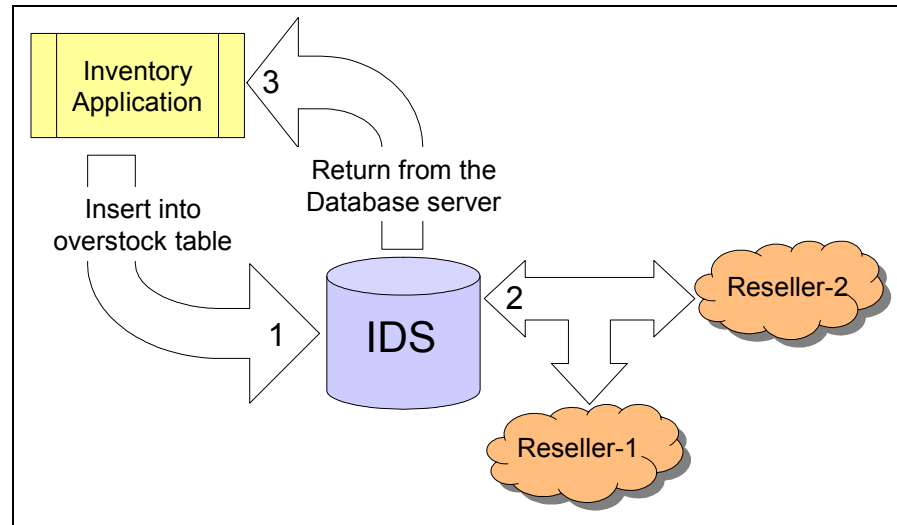


Figure 9-3 Integrating resellers in the inventory process

One advantage of this approach is that if the resellers change, are removed, or are added, the inventory application itself does not need to be changed or even aware of these changes.

In these type of situations, we must know the status of a transaction before executing the action. It is easy to see that if we send an order based on database activities, the database operation must have completed successfully. In some other cases, such as system monitoring, we might want to report on failed transactions. This is discussed further in 9.3, “Why use events” on page 378.

9.2 Database events

IDS has the unique capability of registering functions that will be executed when specific events occur. The IDS documentation refers to these functions as *callbacks*.

The concept of callback functions is found in many software graphical user interfaces (GUIs), XML parsers, programming, and so on. The key is that it is a clean way to handle asynchronous events that occur in the database system.

Table 9-1 lists the event types that are handled by the IDS DataBlade API.

Table 9-1 DataBlade API event types

Event type	Occurrence
MI_Exception	Raised when the database server generates an exception (warning or error).
MI_EVENT_SAVEPOINT	Raised after the cursor flushes within an explicit transaction.
MI_EVENT_COMMIT_ABORT	Raised when the database server reaches the end of a transaction in which work was done.
MI_EVENT_END_XACT	Raised when the database server reaches the end of a transaction. Alternatively, if a hold cursor is involved, raised only after the hold cursor is closed. (Use MI_EVENT_COMMIT_ABORT instead.)
MI_EVENT_END_STMT	Raised when the database server completes the execution of a current SQL statement, or for statements associated with a cursor, when the cursor is closed.
MI_EVENT_POST_XACT	Raised just after the database commits or rolls back a transaction if work was done in the transaction or if an MI_EVENT_END_XACT event was raised.
MI_EVENT_END_SESSION	Raised when the database server reaches the end of the current session.

The IDS implementation provides the flexibility to distinguish between events, so that callback functions can be used in a variety of contexts. In the case of communicating with the outside world, the events of interest are likely to be MI_EVENT_COMMIT_ABORT and MI_EVENT_END_XACT.

9.3 Why use events

Why should we use events instead of simply doing what we need to communicate with the outside world? The short answer is called *transaction boundaries*.

Any communication with the outside world must be initiated only when the state of the work performed in the database is known. That is, the transaction status. This does not necessarily mean we do not want to communicate with the outside world when the transaction aborts.

Imagine that someone starts a transaction and accesses confidential information in a table. If only committed transactions are handled, we do not generate an alert if a rollback were issued on the current transaction.

The use of database server events provides another benefit in that the application does not need to be concerned with these tasks. This means that if the business needs change, such as adding additional resellers, there is no need to change the application itself because only the database processing changes. This benefit is compounded when multiple applications are subject to common rules, because the change then occurs at one place rather than in multiple applications.

9.4 How to use events

An IDS event lives in the context of a database connection. This means that the events and callback functions are specific to a user session. A session must then register callbacks before the events are generated. IDS 11 includes new stored procedures that are executed when a database is opened or closed. But, these procedures must be named `sysdbopen()` and `sysdbclose()`. It is possible to have multiple copies of these procedures, one per user. You can also create a procedure for the PUBLIC user, which applies to any user.

By using this approach, you can register an event when the connection is established, in addition to setting any connection attributes, such as the isolation level. The processing must be limited to the general information that can be gathered from the environment when the event occurs. However, this general approach is likely too restrictive to be useful.

Another way to register events is to relate them to specific tables. This means that the registration of an event is done through a trigger on a specific table. Example 9-1 shows the CREATE TRIGGER syntax.

Example 9-1 CREATE TRIGGER example

```
CREATE TRIGGER eventTab1
INSERT ON tab1
BEFORE (EXECUTE PROCEDURE registerMyCallback() )
FOR EACH ROW (EXECUTE PROCEDURE processRow() )
```

In Example 9-1, we execute an action for each row that is processed in the statement execution. Since we want to process events, we must register a callback, which is done in the BEFORE action. Note that this action is executed even if the triggering statement does not process any rows. This is acceptable since the processing is performed in the FOR EACH ROW action. If nothing is

processed at that level, the callback finds that no action needs to be performed and completes its execution.

IDS provides an extensive sets of trigger capabilities. We review some of these capabilities in the next section.

9.4.1 IDS trigger capabilities

IDS provides the ability to create triggers based on the following SQL operations:

- ▶ DELETE
- ▶ INSERT
- ▶ SELECT
- ▶ UPDATE

The SELECT trigger capability is unique to IDS. It was added in the IDS 9.21 time frame, sometime during the year 2000. Then later IDS 10.0 introduced the ability to create triggers on views (INSTEAD OF triggers).

9.4.2 Trigger introspection

In the IDS 10.0 time frame, the DataBlade API was enhanced to support a new feature referred to as *trigger introspection*. This feature provides the ability, from within a C function, to determine if the function is executing within a trigger, the type of trigger it is executing in, and the before (old) and after (new) image of the row being processed.

The new row is available in the case of INSERT and UPDATE operations. The old row is available for the DELETE and UPDATE operations.

That is what can be done with triggers. Now we return to the callback functions. There are two questions that need to be answered. They are, how can a callback function be created and how can it be registered? Let us take a look.

9.4.3 Creating a callback function

A *callback function* is a function that follows a specific calling convention. Example 9-2 shows the function signature.

Example 9-2 Callback function signature

```
MI_CALLBACK_STATUS MI_PROC_CALLBACK  
<function_name>(MI_EVENT_TYPE eventType, MI_CONNECTION *conn, void  
*eventData, void *userData)
```

A callback function returns a status indicator for how to continue handling the event when the callback completes. Table 9-2 lists the possible return values.

Table 9-2 *Callback status return values*

Value	Description
MI_CB_EXC_HANDLED	Return status indicates that the callback has successfully handled the event, and there is no need to continue with event handling.
MI_CB_CONTINUE	This is the only status other than an exception that callback can return. IDS continues looking for other callbacks to execute for the given event. If an exception callback returns this status, and no other register callback exists, the DataBlade API aborts the user-defined record and any current transaction.

The callback function accepts the following arguments:

- ▶ The event type that triggered the callback
- ▶ The connection on which the event occurred
- ▶ A pointer to an event type structure
- ▶ A pointer to any user data (This data pointer is defined when the callback is registered.)

The last argument is particularly useful since memory can be allocated that is passed to the callback function to provide any type of information that is desired. We return to it in 9.4.5, “Memory duration” on page 383.

The body of the callback is just like any C user-defined function (UDF). It contains calls to DataBlade API functions, standard C functions, and possibly system calls. Example 9-3 shows a simple callback that allows for testing to make sure the callback is called as expected.

Example 9-3 *Simplest callback function*

```
MI_CALLBACK_STATUS MI_PROC_CALLBACK
  cbfunc0(MI_EVENT_TYPE event_type, MI_CONNECTION *conn,
          void *event_data, void *user_data)
{
  mi_integer change_type;
  mi_string *str;

  change_type = mi_transition_type(event_data);
  DPRINTF("logger", 80, ("Entering cbfunc0()"));
  switch(change_type) {
    case MI_BEGIN: str = "BEGIN";
```

```

        break;
    case MI_NORMAL_END: str = "COMMIT";
        break;
    case MI_ABORT_END: str = "ROLLBACK";
        break;
    case MI_ERROR: str = "ERROR";
        break;
    default: str="UNKNOWN!";
}
DPRINTF("logger", 80, ("\ttransition: %s", str));
DPRINTF("logger", 80, ("Exiting cbfunc0()"));
return(MI_CB_CONTINUE);
}

```

In this callback, we write to a trace file that the callback was called and indicate the type of event that occurred. We obtain the latter information by calling the DataBlade API function `mi_transition_type()` by using the argument `event_data` as input to it.

Much more can be added to the callback to send information to the outside world. The possibilities are discussed in 9.6, “Communicating with the outside world” on page 392. The information is usually obtained through the `user_data` parameter that is passed when registering the callback.

9.4.4 Registering a callback function

The DataBlade API provides a set of functions to register, unregister, enable, disable, and retrieve callbacks. Previously we described how to register a callback by calling a user-defined procedure in the BEFORE action of a trigger. The routine can also be called explicitly as part of a transaction. Example 9-4 shows a simple callback registration code.

Example 9-4 Callback registration

```

mi_integer registerCallback0()
{
    MI_CALLBACK_HANDLE *cbhandle;

    cbhandle = mi_register_callback(NULL, MI_EVENT_END_XACT, cbfunc0,
        NULL, NULL);
    if (cbhandle == NULL)

```



```
mi_db_error_raise(NULL, MI_EXCEPTION,  
    "Callback registration failed", NULL);  
return(0);  
}
```

Example 9-5 shows the creation of the user-defined procedure.

Example 9-5 Procedure creation

```
CREATE PROCEDURE registerCallback()  
EXTERNAL NAME  
"$INFORMIXDIR/extend/callbacks/callbacks.bld(registerCallback0)"  
LANGUAGE C;
```

The `mi_register_callback()` function in Example 9-4 takes five arguments. You can find the details of these arguments in the *IBM Informix DataBlade API Function Reference*, G229-6364. The second argument defined is the event that the callback processes, the third argument is the address of the callback function, and the fourth argument is a pointer to a user-defined block of memory.

The block of memory can be of any format as long as both the registration function and the callback agree on its content. Our example passes a NULL pointer, but other information can be included, such as the context of the registration (trigger on the table and the type of trigger (DELETE, INSERT, SELECT, UPDATE)).

IDS has the concept of memory duration for blocks of memory allocated by a user-defined routine (UDR). This concept must be well understood so that we do not end up with either invalid pointers or memory that stays allocated longer than needed. This is the subject of the next section.

9.4.5 Memory duration

When a UDR allocates memory, it uses a default memory duration if none is mentioned explicitly. By default, the memory allocation is valid for the duration of a routine execution. When the routine executes a return, the memory can be reclaimed by the database server. This is not always required.

Table 9-3 lists the memory durations that are available.

Table 9-3 Memory durations

Duration	Explanation
PER_ROUTINE	For the duration of the UDR execution
PER_COMMAND	For the duration of the execution of the current subquery
PER_STMT_EXEC	For the duration of the execution of the current SQL statement
PER_STMT_PREP	For the duration of the current prepared SQL statement
PER_TRANSACTION	For the duration of one transaction
PER_SESSION	For the duration of the current client session
PER_SYSTEM	For the duration of the database server execution

The DataBlade API provides a set of functions for memory allocation. For example, the `mi_alloc()` function allocates memory based on the current default memory duration. This default can be changed with the command `mi_switch_mem_duration()`. Another way to control the memory duration is to use the `mi_dalloc()` functions that take an additional memory duration argument.

Regardless of which memory duration you are using, it is a good practice to free the memory explicitly, using `mi_free()`, if possible.

When writing callback routines and passing information through memory, the most likely duration used is `PER_SESSION` since we need to use the information generated during the transaction after the transaction completes.

You might know that your callback needs to generate information that is reused between transactions, such as some initialization values of a total count of activities, or even between sessions. In this case, you might have to use a `PER_SYSTEM` memory duration, which you most likely use with another memory allocation scheme, named memory, which is discussed in the next section.

9.4.6 Named memory

IDS has another memory allocation mechanism that has been around since the early years of IDS 9.x, specifically called *named memory*. Named memory is a way to allocate a global memory block. This means that the memory allocated through this mechanism can be retrieved by any session. The DataBlade API provides four functions to manipulate named memory, and all four function names are prefixed with `mi_named`.

If the memory is used only at the session level, it is convenient to use the session identifier as part of the name used for the memory block. The code in Example 9-6 shows how named memory can be allocated.

Example 9-6 Named memory allocation

```
MI_CONNECTION *sessionConnection;
mi_integer sessionId;
NamedMemory_t *pmem;
mi_string buffer[32];
. . .

sessionConnection = mi_get_session_connection();
/* Retrieve the session ID */
sessionId = mi_get_id(sessionConnection, MI_SESSION_ID);
/* Retrieve or create session memory */
sprintf(buffer, "session%d", sessionId);
if (MI_OK != mi_named_get(buffer, PER_SESSION, &pmem)) {
    /* wasn't there, allocate it */
    if (MI_OK != mi_named_zalloc(sizeof(NamedMemory_t), buffer,
                                PER_SESSION, &pmem)) {
        mi_db_error_raise(NULL, MI_EXCEPTION,
                          "Logger memory allocation error", NULL);
    }
    /* initialize the memory structure */
    . . .
}
```

After defining local variables, we retrieve the database connection for the current session. This allows us to execute the `mi_get_id()` function and retrieve the session identifier. We then use this session identifier to create a unique name for the entire server.

At this point, we are ready to start manipulating the named memory block. The first *if* statement test is to see if named memory with the name found in the variable `buffer` has already been allocated. If it has been allocated, the memory block is retrieved in the variable `pmem`.

If the named memory is not allocated, we use the `mi_named_zalloc()` function to allocate the named memory block. This function also initializes the content of the memory block to zero.

The named memory block can contain pointers to other memory blocks. These additional memory blocks do not need to be allocated as named memory since we already have a means to retrieve where they are. However, the callback

function might have to clean up that memory before exiting, by using the `mi_free()` function.

If there is a need to free up the named memory block, it can be done easily as shown in Example 9-7.

Example 9-7 Freeing named memory

```
sessionConnection = mi_get_session_connection();
/* Retrieve the session ID */
sessionId = mi_get_id(sessionConnection, MI_SESSION_ID);
/* Retrieve or create session memory */
sprintf(buffer, "session%d", sessionId);
if (MI_OK == mi_named_get(buffer, PER_SESSION, &pmem)) {
    /* was there, free it it */
    mi_named_free(buffer, PER_SESSION);
}
```

This code is similar to the allocation code found in Example 9-6. The interesting part is that the `mi_named_free()` function requires the additional argument that indicates the memory duration of the named memory block. This implies that multiple named memory blocks with the same name can exist as long as they have different duration.

The most common use of callbacks is to use information about the row processed, manipulate it, and send it to a destination. This means that we use a trigger that processed each row and a callback at the end of the transaction. Both the row processing routines and the callback must have access to the same memory. When we register the callback, we can give it a user memory buffer. To ensure that it is the same as the one used by the row processing routines, we use named memory with an agreed upon name. This way, each piece of code has a common memory pointer.

9.4.7 Callback processing

According to the IDS documentation, the registration of a callback survives until one of the following conditions is met:

- ▶ The connection on which the callback is registered closes. Either the UDR exits, or the `mi_close()` function executes.
- ▶ The DataBlade API calls the callback, which happens for state-transition callbacks when one of the following events occurs:
 - `MI_EVENT_SAVEPOINT`
 - `MI_EVENT_COMMIT_ABORT`

- MI_EVENT_POST_XACT
 - MI_EVENT_END_STMT
 - MI_EVENT_END_XACT
 - MI_EVENT_END_SESSION
- You explicitly unregister the callback with the `mi_unregister_callback()` function.

Because this might seem a bit confusing, we provide a few examples. We look at a few possibilities by using the event type `MI_EVENT_END_XACT`.

The first test involves using the callback code shown in Example 9-3 on page 381 and registering for the `MI_EVENT_END_XACT` event in the `BEFORE` action of a trigger. Example 9-8 shows the `INSERT` operations.

Example 9-8 Testing multiple events

```
CREATE TRIGGER mytrig
INSERT ON mytab
BEFORE (EXECUTE PROCEDURE registerCallback0() );

INSERT INTO mytab VALUES(0, "one row");
BEGIN;
INSERT INTO mytab VALUES(0, "row two");
INSERT INTO mytab VALUES(0, "row three");
COMMIT;
```

The first `INSERT` is in an automatic transaction. The trigger executes and registers the trigger. The transaction completes, and we find one call to the callback function in the tracing file.

In the second part, there are two `INSERT` statements that are part of the same transaction. The trigger is executed twice in the transaction. Therefore, the callback function is registered twice. The result is that one transaction calls two callbacks resulting in two new entries in the tracing file.

We need to do one more test, which is to remove the trigger and register the callback in each case, as shown in Example 9-9.

Example 9-9 Explicit callback registration test

```
EXECUTE PROCEDURE registerCallback0();
INSERT INTO mytab VALUES(0, "one row");
BEGIN;
EXECUTE PROCEDURE registerCallback0();
INSERT INTO mytab VALUES(0, "row two");
```

```
INSERT INTO mytab VALUES(0, "row three");
COMMIT;
INSERT INTO mytab VALUES(0, "row four");
```

The first statement registers the callback function, and its execution ends with the following error message:

```
7514: MI_EVENT_END_XACT callback can only be registered inside a
transaction.
```

The first statement executes properly without executing the callback. The statements that are included between BEGIN and COMMIT execute properly resulting in one call to the callback function. The last INSERT also executes properly but without a call to the callback function.

These tests tell us the following information:

- ▶ Multiple callbacks can be registered for the same event, even if it is the same callback function.
- ▶ A registered callback is called once and then is automatically removed.

Since we are likely to want our use of callbacks to be transparent to application code, we always use the trigger method of using callbacks. This means that, if you expect transactions to include multiple operations or multiple tables that register callbacks, you might have to add indicators to ensure that a specific callback is called only once. You can do this easily by using indicators in the named memory block indicating if the callback has already executed. Another method is to have an indicator that keeps track of which callback is already registered and avoid duplicated registration. Example 9-10 shows the second method.

Example 9-10 Single callback registration

```
typedef struct NamedMemory {
    mi_integer registered;
    . . .
} NamedMemory_t;

mi_integer registerCallback1()
{
    MI_CALLBACK_HANDLE *cbhandle;
    MI_CONNECTION *sessionConnection;
    mi_integer sessionId;
    NamedMemory_t *pmem;
    mi_string buffer[32];
```

```

sessionConnection = mi_get_session_connection();
/* Retrieve the session ID */
sessionId = mi_get_id(sessionConnection, MI_SESSION_ID);
/* Retrieve or create session memory */
sprintf(buffer, "session%d", sessionId);
if (MI_OK != mi_named_get(buffer, PER_SESSION, &pmem)) {
    /* wasn't there, allocate it */
    if (MI_OK != mi_named_zalloc(sizeof(NamedMemory_t), buffer,
        PER_SESSION, &pmem)) {
        mi_db_error_raise(NULL, MI_EXCEPTION,
            "Logger memory allocation error", NULL);
    }
    /* initialize the memory structure */
}
if (pmem->registered == 0) {
    cbhandle = mi_register_callback(NULL, MI_EVENT_COMMIT_ABORT,
        cbfunc0, pmem, NULL);
    if (cbhandle == NULL)
        mi_db_error_raise(NULL, MI_EXCEPTION,
            "Callback registration failed", NULL);
    pmem->registered = 1;
}
return(0);
}

```

Start by defining the structure to be used for the named memory. The key component is to have a flag that indicates if the callback has been registered. The size of the structure depends on how the memory will be used.

The code continues with the creation or retrieval of the named memory as shown in Example 9-6 on page 385. Since the memory is initialized to zero, the registered indicator is automatically set, indicating that the callback has not been registered.

After the named memory is retrieved, we test the registered indicator to see if we must register the callback. If we need to register the callback, we do so by passing the named memory pointer as the user data argument. Then we change the indicator after verifying that the registration was successful.

This method implies that the callback itself will either reset the registered indicator to zero or free the named memory. Otherwise, the callback is registered only once for the life of the named memory block.

Now that we know how to create and register callbacks, let us look at how to architect a solution that makes the database server a more active participant in

the architecture of a solution. Two implementation options are described in the following sections.

9.5 Implementation options

There are multiple options when implementing event processing. Two of the more typical options are described in the following sections. We refer to them as Option A and Option B.

9.5.1 Option A

Figure 9-4 illustrates event processing using Option A.

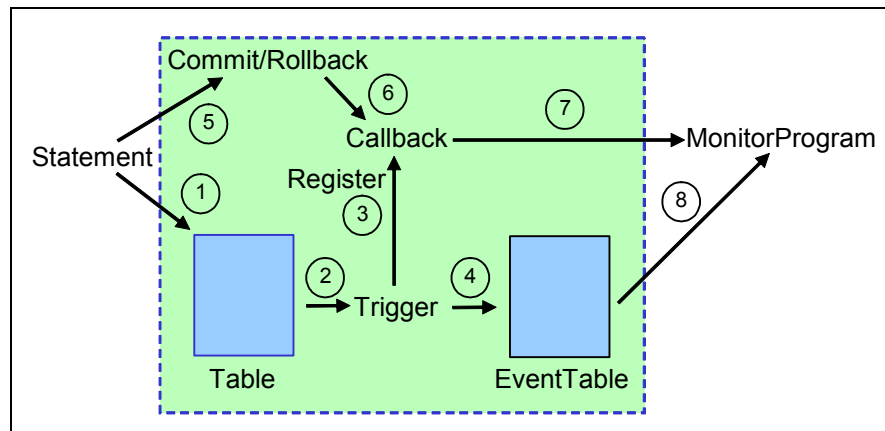


Figure 9-4 Event processing, Option A

We described the first part of this option earlier in this chapter. It is represented by the following steps:

1. The statement executes and accesses a table.
2. The BEFORE trigger is called.
3. The callback function is registered for the desired event.

At this point each row is processed. This implementation involves writing the result of the processing to an event table. That table can simply keep a character representation of the result or can be more complex, including multiple columns of varied types. The processing steps continue as follows:

4. Process each row from the statement.
5. IDS generates a COMMIT or ROLLBACK event.
6. Execute the callback.

Since all rows have been processed, the callback can be minimal and simply perform the following steps:

7. Send a message or a signal to an outside monitoring program.
8. The outside program is started and reads the event table.

This model only works when processing COMMIT events. If you want to process ROLLBACK events by using this architecture, you will find the event table empty. The event table (eventTable) is empty because all the write operations to it are in the context of the current transaction. If the transaction is aborted, the operations are rolled back, which includes removing all the data from the event table. This means that you need a different approach when you want to generate some events, even when transactions are rolled back.

9.5.2 Option B

This implementation option does not depend on a database table. It is necessary when the objective is generate events with information when a rollback occurs. One way to do this is to write to an external file. The DataBlade API provides a set of functions to manipulate files. By using this technique, the file can be read back after a ROLLBACK and still provide the information to an outside program. When performing this option, a specific file name can be agreed on, or a message that provides the path to the result file can be used. However, a more efficient method is to take advantage of named memory.

Figure 9-5 illustrates event processing using Option B.

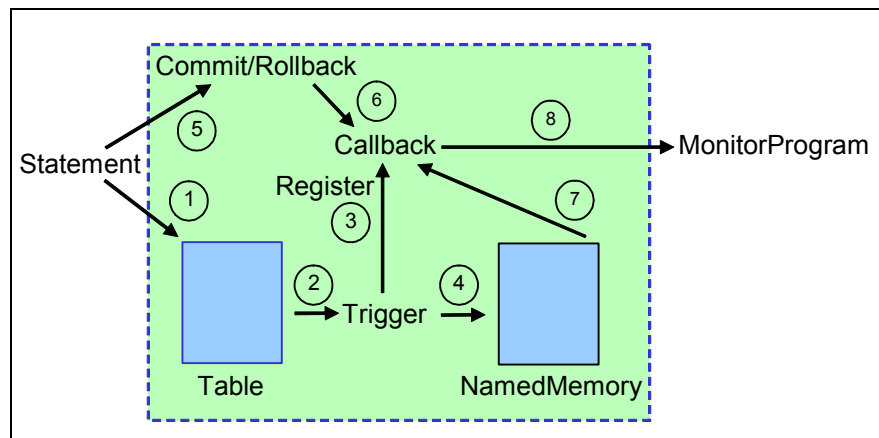


Figure 9-5 Event processing, Option B

The processing for Option B is similar to Option A:

1. The statement executes and accesses a table.
2. The BEFORE trigger is called.
3. The callback function is registered for the desired event.

At this point, each row must be processed. This is where the processing differs from Option A. Here, the result of processing each row is written to named memory. A flexible implementation requires the dynamic allocation of memory and the chaining of these memory blocks. Since this new memory allocation must last longer than the execution of the procedure, the `mi_dalloc()` function must be used to require a memory duration different from the default.

The processing steps continue:

4. Process each row from the statement.
5. IDS generates a COMMIT or ROLLBACK event.
6. Execute the callback.

The callback needs to retrieve the processed information:

7. Read the named memory, free the memory blocks, and so on.
8. Send the results to the outside entity.

9.6 Communicating with the outside world

To complete the discussion on using events, let us look at the following communication options with the outside world:

- ▶ Sending the information to a file
- ▶ Calling a user-defined function (UDF)
- ▶ Sending a signal
- ▶ Using a socket connection
- ▶ Using message queues

9.6.1 Sending information to a file

Sending information to a file is the easiest way to send information to the outside world because the DataBlade API provides functions that emulate the operating system file access system calls. The problem with this approach is that an outside entity must monitor the file to understand the changes. This is the approach used in the IBM developerWorks article “Event-drive fined-grained auditing with Informix Dynamic Server”, which you can find at the following Web address:

<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0410roy/>

The following DataBlade API functions are for file manipulation:

- ▶ `mi_file_allocate()`
- ▶ `mi_file_close()`
- ▶ `mi_file_errno()`
- ▶ `mi_file_open()`
- ▶ `mi_file_read()`
- ▶ `mi_file_seek()`
- ▶ `mi_file_sync()`
- ▶ `mi_file_tell()`
- ▶ `mi_file_to_file()`
- ▶ `mi_file_unlink()`
- ▶ `mi_file_write()`

It might be difficult to manage the information provided by the events. How can the events be removed from the file once its processing is completed?

It makes sense that, instead of writing to a specific file, the callback writes to a specific directory. Each event then generates a new file, and unique file names can easily be generated by using a timestamp value. This way, events can be archived or removed after they have been processed.

By itself, this approach is less useful in an event-driven architecture. However, used in conjunction with signals or other methods, it provides the functionality that is needed with the additional benefit of preserving the events in permanent storage.

9.6.2 Misbehaved functions

Before investigating the other options, we review how the functions execute within the Informix database server.

IDS is based on a multithreaded architecture that uses processes to execute the multiple execution threads. This has two major implications:

- ▶ When a function executes, it keeps control of a virtual processor until it returns or until it executes a DataBlade API function. This means that, if a function executes an I/O operation through the standard operating system calls, the virtual processor that runs this function blocks until the system call completes.
- ▶ If a function is re-scheduled, it might run in a different process that invalidates global variables and other objects. This means that, if a function has a socket open and gets rescheduled, the file descriptor that represents the socket is invalid if the function is not rescheduled on the same virtual processor where the socket was opened.

Avoid these misbehaved types of functions if possible. However, the DataBlade API provides mechanisms to handle these types of functions. When creating a UDF, there is an option to identify the type of virtual processors it must run on, called a *user-defined virtual processor*. In addition to identifying the virtual processor in the CREATE FUNCTION statement, an entry must be added in the onconfig file to define the target virtual processor. IDS already takes advantage of this capability for XML processing and in the basic text search DataBlade. Example 9-11 shows the definitions used for the virtual processors.

Example 9-11 User-defined virtual processor definition

```
VPCLASS idxmlvp,num=1
VPCLASS bts,noyield,num=1
```

As you can see in the VPCLASS definition, it is possible to start more than one of the virtual processors. This brings us back to the second problem discussed in the beginning of this section. To solve that problem, the DataBlade API provides the functions `mi_module_lock()` and `mi_udr_lock()` to request that a function be rescheduled on the same virtual processor on which it initially ran.

Here a user-defined virtual processor is created by using a similar VPCLASS declaration. Assuming the newly created VPCLASS is called *mycallback*, a function can be forced to run on that processor by using a declaration like the one shown in Example 9-12.

Example 9-12 Using the CPU modifier

```
CREATE FUNCTION myFunc()
RETURNING . . .
WITH (CPU=mycallback)
. . .
```

Any function that is misbehaved should use these mechanisms. But what about the callback function? It is not a registered UDF. From a callback function, it is possible to call a UDF that restricts where it executes, which is the subject of the next section.

9.6.3 Calling a user-defined function

Considering what we have discussed in the previous, it seems appropriate to call a UDF from the callback, which is possible with a set of functions that the DataBlade API provides. The functionality is called the *fastpath interface*.

A UDF can be called with no consideration for its implementation. This means that the callback can call a UDF written in C, Java, and SPL.

Since a declared UDF is being called, this function can be declared as running on a specific type of virtual processor as explained in 9.6.2, “Misbehaved functions” on page 393. This enables control on how the code is run.

A Java UDF can also be called. Since the Java runtime environment™ (JRE™) included in IDS is a complete implementation, the full power of the language can be used to do anything the language allows. This includes opening sockets, sending signals, and so on. Java does not have all the functionality available in C. However, this should not be a problem since the callback is written in C and can do any processing unavailable in Java before calling the Java UDF.

The DataBlade API provides the following set of functions under the fastpath interface:

- ▶ `mi_cast_get()`
- ▶ `mi_func_desc_by_typeid()`
- ▶ `mi_routine_end()`
- ▶ `mi_routine_exec()`
- ▶ `mi_routine_get()`
- ▶ `mi_routine_get_by_typeid()`
- ▶ `mi_td_cast_get()`

The fastpath interface provides a more convenient (and faster) way to call a function than to use an EXECUTE FUNCTION SQL statement. As when using SQL statements, the fastpath interface allows functions to be called without having to compile them in the shared library. The function can come from any DataBlade module that is registered in the current database.

When using the fastpath interface, the calls follow this general sequence:

1. Retrieve a reference to the desired function.
2. Execute the function.
3. Release the resources associated with the function descriptor.

The code shown in Example 9-13 demonstrates this sequence of operations.

Example 9-13 Operations using the fastpath interface

```
MI_FUNC_DESC *fn;
MI_CONNECTION *conn;
. . .
conn = mi_get_session_connection();
fn = mi_routine_get(conn, 0, "writeFile(lvarchar, lvarchar)");
(void) mi_routine_exec(conn, fn, &ret, buffer, pcur->xml);
mi_routine_end(conn, fn);
```

To retrieve a function, a database connection is required. The one from the session in which we are running can be used. Having the connection, the function descriptor that corresponds to the function name and signature provided as an argument can be retrieved. In this example, we are looking for the following function:

```
writeFile(lvarchar, lvarchar)
```

If the function is not found, the `mi_routine_get()` functions return a NULL pointer. With the function descriptor, the function can be called and passed the proper arguments. The `mi_routine_exec()` function returns a pointer to the value that is returned by the execution of the target function.

When finished, the memory associated with the function descriptor can be released.

9.6.4 Sending a signal

On UNIX systems, it is possible to send a signal to another process (monitoring process) running on the same machine. This is done by using the `kill()` system call. The following signals can be used for this purpose:

- ▶ SIGUSR1: User-defined signal 1
- ▶ SIGUSR2: User-defined signal 2

The `kill()` system call takes only two arguments: the process number where to send the signal and the signal being sent. Because of this, some issues must be addressed when using this capability.

First, know which process identifier you want to target. Depending on the operating system, it is possible to find information about the running processes by using several methods. A simple, platform independent way is to agree on a specific file where the process ID is written when the program is started. The callback function can then use the DataBlade API functions to read the file to retrieve the process number and send the signal.

Since there are two user-defined signals, information can be conveyed based on which signal is sent. For example, one signal can be used for committed transactions and the other signal can be used for rollbacks.

Sending a signal from a callback function should not make it a misbehaved function. However, we recommend that you treat it as though it were misbehaved.

After the monitoring process receives the signal, it knows to return to the database server to retrieve the details of the event. If you are only concerned with committed transactions, the easiest way is to have a communication table

that the monitoring process reads to handle the events. After an event is handled, the row is removed from the table.

When you also want to handle transaction rollbacks, the solution is slightly more complicated since it is not possible to write the event to a communication table. If you do so, the inserted row or rows disappear due to the rollback.

One solution is to write the event to an external file. This is done by defining a directory for event files. After receiving a signal, the monitoring program can start processing the event files from the given directory. When an event is processed, the event file can be removed from the directory or moved to another location.

Another solution is to keep the information in named memory. After receiving the signal, the monitoring process gets the information from named memory and processes the events. Of course, the monitoring program cannot access named memory directly. Therefore, this implementation must include additional UDRs to provide this access.

Earlier in this chapter, we discussed allocating named memory on a session basis. This does not work for this solution since the monitoring program does not know which session issued the signal. The named memory must be allocated at a higher duration (`PER_SYSTEM`) and shared by all the sessions that generate events. The management and access to the named memory become more complex since we must be concerned about concurrency control using the DataBlade API functions `mi_lock_memory()`, `mi_try_lock_memory()`, and `mi_unlock_memory()`.

9.6.5 Opening a network connection

Another way to communicate with a monitoring process is to open a network connection to it. In this case getting a machine name or address and a port number is required. There are several ways to provide this information. For example, it can be provided through a file or even as an argument to the callback registration function.

The DataBlade API does not provide functions to establish a network connection. An easy way to work around this issue is to use the fastpath interface to call a UDF written in Java.

A C UDF can still be used to access the network. This function is definitely a misbehaved function. Therefore, a user-defined virtual processor is required.

A UDF runs as a thread in a virtual processor. Over time, a thread can be rescheduled to run on a different virtual processor. This means that any open file descriptor, including network connections, become invalid when you change the

virtual processor. If you define multiple instances of the user-defined virtual processor to run the UDF that includes the network connection, the function can fail at times due to scheduling. To work around this issue, use the DataBlade API function `mi_udr_lock()` to ensure that the function does not migrate to another virtual processor during its execution.

9.6.6 Integrating message queues

IDS includes the MQSeries DataBlade module, which is described in 7.5.1, “DataBlades included with IDS” on page 289. The MQSeries DataBlade can be a simple way to integrate IDS into an SOA environment.

Message queues are much simpler to integrate into an event-processing scheme since the communication with the message queue is part of the transaction. This means that, if there is a transaction rollback, the message is not sent to the outside world.

The use of message queues does not require a callback function. The trigger can simply either call the appropriate function or INSERT into a message queue table if it is set up accordingly.

9.6.7 Other possibilities

Other options are possible, such as using an HTTP connection to communicate with the World Wide Web. Another approach is to access Web services. A Web service can easily be accessed through IDS by using Java UDFs which is explained in 8.3.2, “Service providing with IDS 11” on page 321.

The use of Java UDFs should not be underestimated. Most communication toolkits made available by Web-based companies include a Java interface that can be used within IDS.

9.7 Conclusion

Integrating IDS in the architecture of a solution can simplify the design and improve performance. It can also provide an additional separation from business processes and a specific business application.

The logic of which external business partner is contacted following a specific event can be kept outside the application that generated the event. This way, if new business partners are added or some are removed, the application does not change. Since the logic is in the database, multiple applications can take

advantage of the database logic. This simplifies the applications and provides an additional way to reuse code.

In this chapter, we described how you can take advantage of the power of IDS extensibility to gain a business advantage. The more you learn about the capabilities of IDS, the more you can improve your efficiency. However, integrating IDS into a solution architecture from the start is the best approach.



The world is relational

In this chapter, we discuss of the Virtual Table Interface (VTI) and Virtual Index Interface (VII) features that are provided by the Informix Dynamic Server (IDS). In business today, a significant volume of application data is still not stored in relational tables, while there is a requirement to integrate this data with the data that is stored in relational databases to satisfy business operations. IDS provides a rich framework for application developers to integrate non-relational data sources into the relational model, thereby enabling SQL query capabilities on data in these non-relational data sources.

We also show a few examples that illustrate the power of this framework and provide possible starting points for how to customize IDS to build complex data integration applications.

10.1 Virtual Table and Virtual Index Interfaces

Traditionally relational databases (RDBs) have provided query capabilities for data that is stored and organized in relational tables. Databases understand relational schemas and have sophisticated indexing mechanisms to efficiently access the underlying data. They provide SQL capabilities to query, dissect, and examine the data. IDS has extended this model to enable integration of non-relational data into the relational model and thus extend the power and value of business applications. The VTI is a framework defined to implement gateways to non-relational data into IDS. Popular implementations of VTI include `ffvti` (external file access) and message queue (MQ) VTI tables.

In addition to providing a VTI framework, IDS provides a framework for application developers to extend or implement new indexing schemes. If the more typical B-Tree or RTREE implementations do not satisfy the needs of an application, then there is a requirement for a new indexing mechanism. The VII is a framework to extend or define new indexing schemes. Popular implementations of VII include the Excalibur Text (ETX) and Basic Text Search (BTS) DataBlades.

IDS refers to VTI and VII as *user-defined access methods* (UDAMs). An access method consists of software routines that open files, retrieve data into memory, and write data to permanent storage, such as a disk. IDS natively supports built-in access methods that work with relational data stored in IDS tables. However, UDAMs are plug-ins that allow IDS to understand non-relational data or data stored outside of IDS relational tables.

VTI is also called a *user-defined primary access method*. A *primary access method* provides a relational table interface for direct read and write access. A primary access method reads directly from and writes directly to source data. It provides a means of combining data from multiple sources in a common relational format that the database server, users, and application software can use. IDS relational tables use native built-in primary access methods.

VII is also referred to as a *user-defined secondary access method*. A *secondary access method* provides a means of indexing data for alternate or accelerated access. An index consists of entries, each of which contains one or more key values, and a pointer to the row in a table that contains the corresponding value or values. The secondary access method maintains the index to coincide with inserts, deletes, and updates to the primary data. Indexes created on IDS relational tables use BTREE as secondary access method by default. The BTREE access method is native to IDS and falls under the built-in secondary access methods.

UDAM: Unless otherwise stated, we use the acronym UDAM for both the VTI and VII frameworks.

10.1.1 The UDAM framework

The UDAM framework consists of a set of the following components:

- ▶ Purpose functions, flags, and values
- ▶ System catalogs
- ▶ Descriptors
- ▶ DataBlade APIs that work with the descriptors

IDS communicates with the UDAM through the invocation of an appropriate purpose function. Information about SQL statement specification and state is passed to the access method using various descriptors. The information held in the descriptors can be accessed and manipulated by using DataBlade APIs that operate on the descriptors.

Purpose functions are user-defined routines (UDRs) with predefined signatures that are implemented in the C language. The sequence of purpose function invocations depends on the type of SQL statement. We discuss this topic in more detail in the remaining sections of this chapter.

Purpose functions are implemented by the access method developers. They fill in for the built-in purpose functions that are otherwise called for normal relational tables. Since IDS does not understand the data handled by the access method, it relies on purpose functions to transform data into the relational format.

SQL syntax to define UDAM

Now we look at the SQL extensions with which we can define new access methods and create tables by using those access methods.

Example 10-1 shows the syntax for creating a UDAM in IDS.

Example 10-1 UDAM create syntax

```
CREATE <PRIMARY | SECONDARY> ACCESS_METHOD <method name>
(
  <purpose function> = <external_routine> |
  <purpose value> = < string value | numeric value > |
  <purpose flag>
);
```

The CREATE ACCESS_METHOD statement registers the UDAM with IDS and provides a list of capabilities and purpose functions implemented by the UDAM. Purpose function names must be provided at UDAM create time, and the underlying UDRs must be registered beforehand.

Table 10-1 lists commonly used purpose functions, values, and flags.

Table 10-1 Purpose functions, values, and flags

Purpose word	Description	Category
am_sptype	A character that specifies from what type of storage space a primary or secondary-access method can access data.	Value
am_keyscan	A flag that, if set, indicates that am_getnext returns rows of index keys for a secondary-access method. If a query selects only the columns in the index key, the database server uses the row of index keys that the secondary-access method puts in shared memory, without reading the table.	Flag
am_unique	A flag to set if a secondary-access method checks for unique keys.	Flag
am_readwrite	A flag to set if a primary-access method supports data changes. The default setting, not set, indicates that the virtual data is read-only.	Flag
am_parallel	A flag that the database server sets to indicate which purpose functions or methods can execute in parallel in a primary or secondary-access method.	Flag
am_costfactor	A value by which the database server multiplies the cost that the am_scancost purpose function or method returns for a primary or secondary-access method.	Value
am_create	Name of a user-defined function (UDF) or method (UDR) name that creates a virtual table or virtual index.	Function
am_drop	Name of UDF or method (UDR) name that drops a virtual table or virtual index.	Function
am_open	Name of UDF or method (UDR) name that makes the virtual table or virtual index available for access in queries.	Function
am_close	Name of a UDF or method (UDR) name that reverses the actions of the am_open function.	Function
am_insert	Name of a UDR that inserts a row or an index entry.	Function

Purpose word	Description	Category
am_delete	Name of a UDR that deletes a row or an index entry.	Function
am_update	Name of a UDR that updates a row or an index entry.	Function
am_stats	Name of a UDR that builds statistics based on the distribution of values in the access method.	Function
am_scancost	Name of a UDR that calculates the cost of qualifying and retrieving data.	Function
am_check	Name of a UDR that tests the physical structure of a table or performs an integrity check on an index.	Function
am_beginscan	Name of a UDR that sets up a scan.	Function
am_endscan	Name of a UDR that reverses the setup that am_beginscan initializes.	Function
am_rescan	Name of a UDR that scans for the next item from a previous scan to complete a join or subquery.	Function
am_getnext	Name of the required UDR that scans for the next item that satisfies a query.	Function
am_getbyid	Name of a UDR that fetches data from a specific physical address. am_getbyid is available only for primary access methods.	Function
am_truncate	Name of a UDR that deletes all rows of a virtual table (primary-access method) or that deletes all corresponding keys in a virtual index (secondary-access method).	Function

For a more complete list of purpose functions, values, and flags, refer to the IDS 11 information center at the following Web address:

<http://publib.boulder.ibm.com/infocenter/idshep/v111/index.jsp>

The ALTER ACCESS_METHOD syntax, shown in Example 10-2, provides a way to change the capabilities of a UDAM. The ADD, DROP, and MODIFY keywords can be used to achieve the desired change in UDAM capabilities.

Example 10-2 ALTER UDAM syntax

```
ALTER ACCESS_METHOD <method name>
ADD|DROP|MODIFY <purpose function | purpose value>
= <routine name | quoted string>
```

The DROP ACCESS_METHOD syntax, shown in Example 10-3, provides a way to drop a UDAM as long as it is not used by any virtual table or index.

Example 10-3 DROP ACCESS_METHOD

```
DROP ACCESS_METHOD <method name> RESTRICT
```

Having looked at the SQL extensions to register or manipulate UDAMs, we now look at how to create tables or indices by using the UDAM, which is shown in Example 10-4.

Example 10-4 CREATE TABLE/INDEX using UDAM

```
CREATE TABLE ...  
USING <access method name>  
(  
    <identifier>=<value>,  
    <identifier>=<value>,  
    ...  
)  
  
CREATE INDEX ... <index key specification> <operator class name>  
USING <access method name>  
(  
    <identifier>=<value>,  
    <identifier>=<value>,  
    ...  
)
```

The USING clause associates an access method with the virtual table or virtual index. The <identifier>=<value> block is optional and can be used to pass configuration or customization parameters to the access method. This information is accessed by the UDAM by using the descriptors that are passed to the purpose functions. This mechanism provides access method developers extra flexibility in designing a UDAM, so that a single UDAM can be used for multiple virtual tables or indices.

System catalogs

IDS uses a set of system catalog tables to store the metadata information about a UDAM and associated virtual tables and indices. The catalogs have the following names:

- ▶ informix.sysams
- ▶ informix.sysprocedures
- ▶ informix.systables
- ▶ informix.sysindices

informix.sysams is the primary catalog that holds the metadata information about UDAM obtained from the CREATE ACCESS_METHOD statement.

Example 10-5 shows the entry in sysams for a HASH access method.

Example 10-5 informix.sysams schema

```
> select * from informix.sysams where am_name = 'hash';
```

am_name	hash
am_owner	informix
am_id	3
am_type	P
am_sptype	D
am_defopclass	0
am_keyscan	0
am_unique	0
am_cluster	0
am_rowids	1
am_readwrite	1
am_parallel	0
am_costfactor	1.000000000000
am_create	116
am_drop	0
am_open	117
am_close	118
am_insert	119
am_delete	121
am_update	120
am_stats	0
am_scancost	122
am_check	0
am_beginscan	123
am_endscan	0
am_rescan	124
am_getnext	125
am_getbyid	126
am_build	0
am_init	0
am_truncate	0

It is clear that there is a one-to-one correspondence between the purpose function, value, and flags provided at UDAM create time and what is populated in sysams. Purpose functions are represented by the UDR identifier, instead of the actual UDR name. For example, am_create for the HASH access method is

identified by UDR identifier 116. UDR identifiers uniquely identify the UDR in the `informix.sysprocedures` catalog where UDR metadata is stored.

The `informix.sysprocedures` catalog stores metadata information about UDRs. UDAM purpose functions are implemented using C UDRs, and therefore, they have a unique entry in the `informix.sysprocedures` catalog. Example 10-6 shows the entry in `informix.sysprocedures` for the HASH `am_create` purpose function. Entries in the `informix.sysprocedures` catalog are populated by using the `CREATE PROCEDURE` or `CREATE FUNCTION` SQL statement.

Example 10-6 `informix.sysprocedures` entry for HASH `am_create`

```
> select * from sysprocedures where procid=116;
```

procname	sha_create
owner	informix
procid	116
mode	d
retsize	102
symsize	260
datasize	0
codesize	0
numargs	1
isproc	f
specificname	
externalname	(sha_create)
paramstyle	I
langid	1
paramtypes	pointer
variant	t
client	f
handlesnulls	f
iterator	f
percallcost	0
commutator	
negator	
selfunc	
internal	f
class	
stack	
parallelizable	f
costfunc	
selconst	0.00
collation	en_US.819

The `informix.systables` and `informix.sysindices` hold the metadata information about tables and indices created in a database. For virtual tables and indices, it also holds the UDAM identifier or `am_id`. Example 10-7 shows the entry in `informix.systables` for a virtual table created with the HASH access method. In this example, the `am_id` column value is 3, which uniquely identifies the HASH access method entry in `informix.sysams`.

Example 10-7 `informix.systables` entry for virtual table using HASH access method

```
> CREATE TABLE hash_table ( col1 INTEGER )  
USING hash ( mode="static", hashkey="(col1)", number_of_rows="100" );
```

```
> SELECT * FROM informix.systables WHERE tabname="hash_table";
```

tabname	hash_table
owner	vshenoi
partnum	1048840
tabid	105
rowsize	4
ncols	1
nindexes	0
nrows	0.00
created	11/07/2007
version	6881281
tabtype	T
locklevel	P
npused	0.00
fextsize	16
nextsize	16
flags	0
site	
dbname	
type_xid	0
am_id	3
pagesize	2048
ustlowts	
secpolicyid	0
protgranularity	

Descriptors

The UDAM framework defines a set of descriptors to hold information that is passed to the C functions that provide the implementation of purpose functions. While many descriptors provide support when writing purpose functions, we only discuss a few of the more important ones.

When the server calls one of the purpose functions, it passes a descriptor of the appropriate type, containing all the information that is required by the function to perform its work. Two of the descriptors, the scan descriptor and the table descriptor, also contain *userdata fields* that can be used to store information specific to the particular implementation. The userdata field is maintained or cached by the server for the duration of a UDAM operation, such as SCAN, INSERT, and DELETE. It, thereby, provides a placeholder for the UDAM to store state information or cache data between purpose function calls.

Table 10-2 shows the descriptors that are defined by the UDAM framework. The accessor function prefix column provides the DataBlade API extension function name prefixes that operate on the respective descriptors.

Table 10-2 UDAM descriptors

Descriptor	Description	Accessor function prefix
Key descriptor (MI_AM_KEY_DESC)	Holds index keys, strategy functions, and support functions (VII only)	mi_key_
Qualification descriptor (MI_AM_QUAL_DESC)	Holds WHERE clause criteria	mi_qual_
Row descriptor (MI_ROW)	Holds order and data types of projected columns	mi_row_
Row-id descriptor (MI_AM_ROWID_DESC)	Holds indexed table row location (VII only)	mi_id_
Scan descriptor (MI_AM_SCAN_DESC)	Holds SELECT clause projection	mi_scan_
Statistics descriptor (MI_AM_ISTATS_DESC)	Holds data distribution of values	mi_istats_
Table descriptor (MI_AM_TABLE_DESC)	Holds table (VTI) or index (VII) location and attributes	mi_tab_

UDAM DataBlade API extensions

UDAM developers are provided with numerous utility functions that work with descriptors passed into purpose function implementations. The access method developer is responsible for accessing the external data by using information from the descriptor. The developer is also responsible for applying the knowledge of the underlying external data store and performing the action that is appropriate for the purpose function task.

Now we look at one specific example with a focus on the `am_open` purpose function. Suppose an access method is created with the statement shown in Example 10-8.

Example 10-8 Creating an access method

```
CREATE PRIMARY ACCESS METHOD mymethod
(
  AM_BEGINSCAN = my_begin_scan,
  AM_GETNEXT = my_get_next,
  AM_OPEN = my_open,
  ...
)
```

This implies that several SQL functions, including one called `my_open`, have been registered with the database server by using the CREATE PROCEDURE/FUNCTION statement (Example 10-9).

Example 10-9 CREATE statement

```
CREATE FUNCTION my_open (ptr POINTER)
RETURNING integer
EXTERNAL NAME "$INFORMIXDIR/extend/mymethod/mymethod.bld(my_open)"
LANGUAGE C
END FUNCTION;
```

In this example, `my_open()` is defined to take an argument of type `POINTER`. The database server passes a reference to an `MI_AM_TABLE_DESC` to the function, but no SQL data type corresponds to this structure. The `POINTER` data type is provided to allow opaque types to be passed to SQL functions under these circumstances.

We look at some tasks that `my_open` must perform, which depends on the requirements of the external data store:

- ▶ Verifying that the user has authority to open the table or file
- ▶ Initializing a user data structure with information that subsequent calls will require.

Tip: Memory for user data structures should be allocated with the `mi_dalloc()` DataBlade API memory allocator call, with `PER_COMMAND` duration, so that it persists across function calls. For a discussion of memory durations provided by the DataBlade API, refer to the *IBM Informix DataBlade API Function Reference, Version 11.1*, G229-6364.

- ▶ Opening an external file and save a file descriptor, or performing the equivalent of an `mi_open()` for an external relational database
- ▶ Obtaining a row descriptor for the table, which allows subsequent function calls to process data from individual columns

This work is done with a combination of the DataBlade API accessor function calls, DataBlade API calls, and standard C. For example, to retrieve the name of the table to which an `MI_AM_TABLE_DESC` refers, we make a call to `mi_tab_name()`. We then get the name of the table owner with a call to `mi_tab_owner()`. The accessor function prefix column from Table 10-2 on page 410 lists the prefix of the function calls that operate on respective descriptors. For a complete list of DataBlade API accessor functions, refer to the IDS 11 information center at the following Web address:

<http://publib.boulder.ibm.com/infocenter/idshelp/v111/index.jsp>

How everything fits together

Assuming that we are trying to create a virtual table or index by using syntax from Example 10-4 on page 406, the following steps are performed by the database server to create the external data store:

1. Extracts the access method name from the `USING` clause.
2. Retrieves the row from `informix.sysams` that corresponds to the given access method name.
3. Retrieves the UDR identifier for the `am_create` purpose function from the `informix.sysams` row.
4. Retrieves the `informix.sysprocedures` row for the UDR identifier obtained in previous step.
5. Creates `MI_AM_TABLE_DESC` and populates all relevant fields of the structure.
6. Executes the `am_create` function based on information from `informix.sysprocedures` and passes the populated `MI_AM_TABLE_DESC` structure. This step should create the external data store.
7. Creates an entry in `informix.systables` or `informix.sysindices` for the new virtual table or index if the `am_create` function execution is successful. Records the `am_id` from `informix.sysams` row into the new `informix.systables/sysindices` entry.

10.1.2 Qualifiers

The IDS SQL optimizer evaluates all the filters as part of a WHERE clause of an SQL query and decides on the list of filters that can be pushed down to the purpose functions of UDAM for evaluation. The UDAM framework reformats those filters and stores the results in a qualification descriptor (an MI_AM_QUAL_DESC structure). The qualification descriptor, in turn, is encapsulated in a scan descriptor (an MI_AM_SCAN_DESC structure), which is passed to the access method scan purpose functions, such as am_beginscan and am_getnext.

In a typical implementation, the am_beginscan purpose function calls mi_scan_qual() to extract the qualification descriptor from the scan descriptor. The am_getnext routine uses that information to qualify each row that it returns to the server.

A qualification might be a simple predicate, as shown in Example 10-10.

Example 10-10 Qualifier predicate

```
SELECT * from hash_table where col1 = 1;
```

Alternatively, a qualification can be a Boolean expression that contains an array of MI_AM_QUAL_DESC structures. Each structure contains either a simple predicate or another array of Boolean expressions. This setup is called a *complex qualifier*. Example 10-11 shows an example of a complex qualifier with three simple qualifiers (col1 =1, col1 =10, col1 > 15) joined by an OR operator.

Example 10-11 Complex qualifier

```
SELECT * from hash_table where col1 = 1 or col1 = 10 or col1 > 15;
```

The accessor function mi_qual_issimple() tests whether an MI_AM_QUAL_DESC structure points to a simple qualifier.

The accessor functions mi_qual_nquals() and mi_qual_qual() return the total number of qualifiers and a particular qualifier descriptor based on a given index into the array of qualifier descriptors respectively when handling complex qualifiers.

Handling qualifiers

The task of the purpose functions is to break the qualification down into a series of simple predicates. It is also to assign a value of MI_VALUE_TRUE, MI_VALUE_FALSE, or MI_VALUE_NOT_EVALUATED to each qualifier by using the accessor function mi_qual_setvalue().

Example 10-12 shows an excerpt of code that handles simple and complex qualifiers.

Example 10-12 Code for complex qualifiers

```
if (mi_qual_issimple(qd))
{
    /* Execute simple, function. (Not shown.) */
    /* Test the result that the function returns. */
    if (result == MI_TRUE)
    {
        /* Set result in qualification descriptor.*/
        mi_qual_setvalue(qd,MI_VALUE_TRUE);
        return; ;
    }
    else
    {
        mi_qual_setvalue( qd,MI_VALUE_FALSE);
        return;;
    }
} /* END: if (mi_qual_issimple(qd)) */
else
{ /* Complex qualification (has AND or OR)..Loop until all functions
execute.*/
    for (i = 0; i < mi_qual_nquals(qd); i++)
        get_result(mi_qual_qual(qd, i), my_data)
    } /* END: Complex qualification (has AND or OR) */
return;;
```

When the purpose function has processed each of the simple predicates in a qualification, it can make a call to `mi_eval_am_qual()` to finish evaluating the qualification. This causes the database server to evaluate any predicates that were set to `MI_VALUE_NOT_EVALUATED` and to assign a value of `MI_VALUE_TRUE` or `MI_VALUE_FALSE` to the statement as a whole. If the set of qualifiers as a whole has a value of `MI_VALUE_TRUE`, then the row satisfies the qualification, and the purpose function can return it to the server. Otherwise, the purpose function can skip to the next row.

10.1.3 Flow of DML and DDL with virtual tables and indices

Now that you understand the UDAM framework and the surrounding data structures, we turn our focus to the flow of the purpose function calls for various Data Manipulation Language (DML) and Data Definition Language (DDL) operations.

UDAM CREATE TABLE and CREATE INDEX flow

When an application issues a CREATE TABLE or CREATE INDEX statement by using a UDAM, the database server invokes the sequence of purpose functions as illustrated in Figure 10-1.

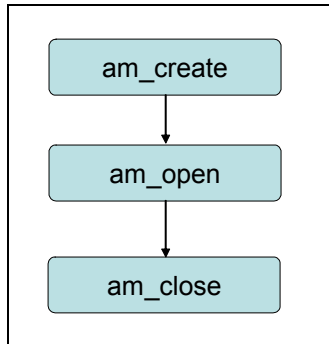


Figure 10-1 VTI create flow

Figure 10-2 shows a VII flow that differs from the VTI flow in that, for every row in the table (on which the index is created), the database server performs `am_insert` to insert the index key value into the virtual index.

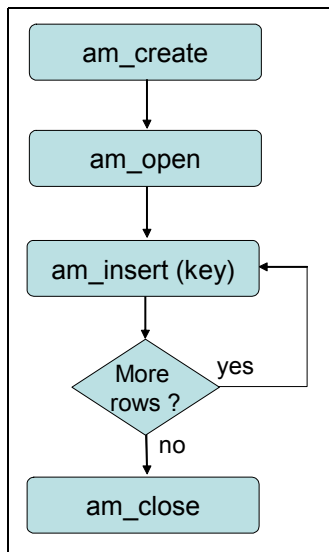


Figure 10-2 VII create flow

UDAM DROP TABLE or DROP INDEX flow

When an application issues a DROP TABLE or DROP INDEX statement to drop a virtual table or an index, the database server invokes the sequence of purpose functions as illustrated in Figure 10-3.

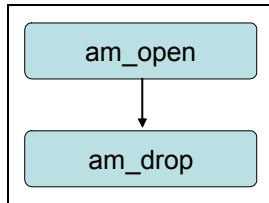


Figure 10-3 UDAM drop flow

UDAM INSERT, DELETE, UPDATE flow

The INSERT, DELETE, UPDATE of a UDAM table has the following scenarios, with different flows for each:

- ▶ INSERT, DELETE, UPDATE with row address or row ID (see Figure 10-4)

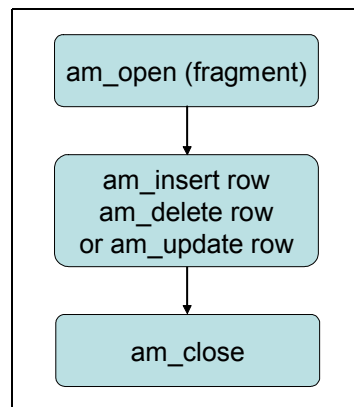


Figure 10-4 UDAM IDU with row address (row ID) flow

- ▶ INSERT, DELETE, UPDATE with UDAM table in subquery (see Figure 10-5)

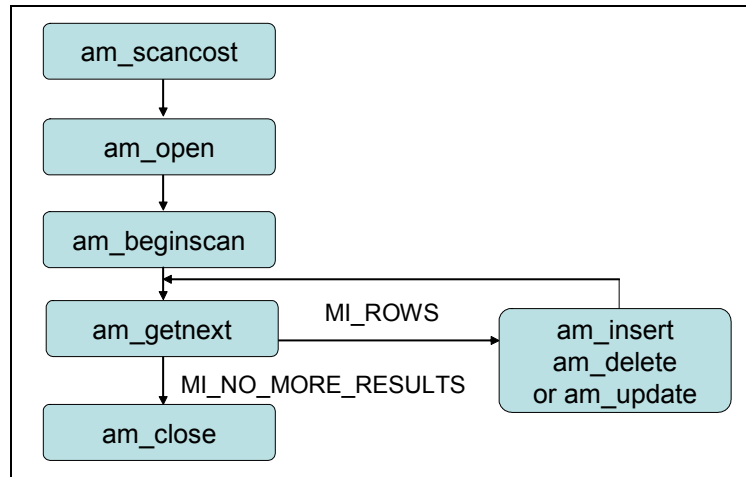


Figure 10-5 UDAM IDU in a subquery flow

- ▶ INSERT, DELETE, UPDATE with multiple rows returned by am_getnext (see Figure 10-6)

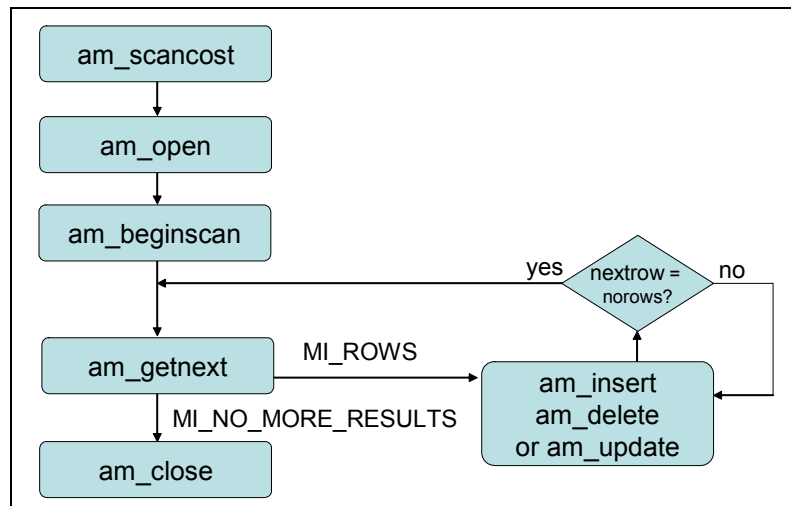


Figure 10-6 UDAM IDU with multiple rows flow

am_scancost purpose function: The am_scancost purpose function is called before am_open/am_begins to estimate the cost of scanning the table or index, given the qualifiers to be applied.

UDAM SELECT flow

The UDAM SELECT flow, shown in Figure 10-7, is self-explanatory. The `am_getnext` purpose function applies the qualifiers and returns either `MI_ROWS` (row qualifies) or `MI_NO_MORE_ROWS` (no rows qualify).

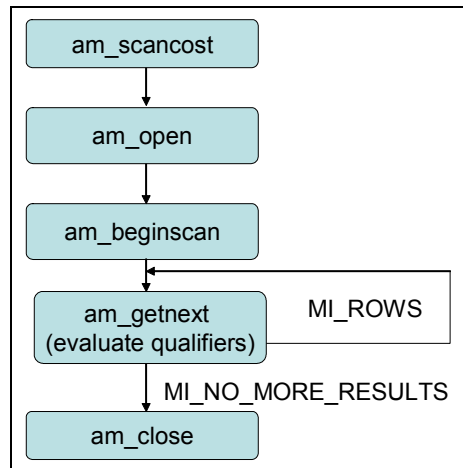


Figure 10-7 UDAM SELECT flow

UDAM oncheck flow

IDS supports **oncheck** capabilities for table and index checks on virtual tables and indices. The **oncheck** command determines if the table or index to check uses a UDAM. If it does, then **oncheck** calls the `am_check` purpose function to perform the necessary checks.

am_check: `am_check` is a purpose function that is implemented by UDAM developers. This means that IDS relies on UDAM developers to provide the checking facilities for virtual tables and indices.

UDAM truncate flow

IDS supports truncation of virtual tables via the TRUNCATE TABLE SQL statement. UDAMs that support a truncate table are required to implement the `am_truncate` purpose function and must have the `am_readwrite` flag set on the UDAM. Figure 10-8 shows the flow of truncate table execution for UDAM.

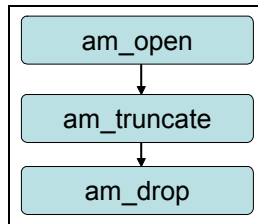


Figure 10-8 UDAM truncate flow

10.1.4 UDAM tips and tricks

In this section, we discuss the following topics to help UDAM developers handle various database server services and monitor UDAM usage:

- ▶ “Transactions”
- ▶ “Caching data” on page 420
- ▶ “Logging and recovery” on page 420
- ▶ “Locking” on page 421
- ▶ “Parallelization” on page 421
- ▶ “Onstat options for UDAM” on page 421

Transactions

Currently, the UDAM framework does not support the two-phase commit protocol. Therefore, transaction management for external data is problematic. A caveat to this restriction is that, starting with IDS v10.00xC1, XA-compliant data sources can be registered with the database server, so that they can participate in the two-phase commit protocol.

Changes in transaction state can be monitored by registering a callback to check for a “type” value of `MI_EVENT_END_STMT`, `MI_EVENT_END_XACT`, or `MI_EVENT_END_SESSION`. This makes it relatively easy to manage simple transactions by caching any updates and instantiating them only when the transaction ends (or not instantiating them if the transaction rolls back). Refer to Chapter 9, “Taking advantage of database events” on page 373.

The difficulty comes when an error occurs during a commit. If the transaction affects data from both the external data source and internal Informix tables, then

it is possible for an update to occur on one side but not the other. For now, there is no perfect way to handle this condition.

Caching data

The UDAM developer has access to the following hooks for caching information across API calls:

- ▶ Caching at scan level

The UDAM can cache information during a scan (`am_beginscan` → `am_endscan`), by using the `mi_scan_setuserdata` and `mi_scan_userdata` accessor functions on `MI_AM_SCAN_DESC` descriptor. The database server does not free this memory when the scan descriptor is destroyed. The access method can either free it at `am_endscan` time or let the server free it when the memory duration expires. A memory duration of `PER_COMMAND` is sufficient for the time a scan is open.

- ▶ Caching at open table level

The UDAM can cache information during a table or index open, (`am_open` → `am_close`), by using the `mi_tab_setuserdata` and `mi_tab_userdata` routines on an `MI_AM_TABLE_DESC` descriptor. The server does not free this memory when the table descriptor is destroyed. The UDAM can either free it at `am_close` time or let the server free it when the memory duration expires. A memory duration of `PER_STATEMENT` is sufficient for the time a table or index is open.

- ▶ Caching using named memory

The UDAM can allocate memory of any duration and associate a name with it by calling the DataBlade API routines `mi_named_alloc` or `mi_named_zalloc`. Later, UDAM can retrieve the address of that memory by calling `mi_named_get` with the same name. Eventually, the access method can free the memory by calling `mi_named_free`.

Logging and recovery

If the data associated with a virtual table or index is completely external to the database, that is if it is in an IDS external space, then it is not possible for the database server to provide logging and recovery services for it. If this capability is required for the UDAM, then it is up to the UDAM developer to implement a custom solution for achieving logging and recovery.

However, if the data is maintained in an IDS SmartBLOB space, then logging and recovery services of the SmartBLOB component can be leveraged by the UDAM. Be aware that the data must be stored in a SmartBLOB that was created with the `MI_LO_ATTR_LOG` flag turned on. Refer to the `mi_lo_create`, `mi_lo_spec_init`, and `mi_lo_specset_flags` routines in the *IBM Informix DataBlade API Programmer's Guide, Version 11.1*, G229-6365, for details.

Locking

As with logging and recovery, locking is beyond the control of the server if the data is stored in a location that is external to the server.

If the data is stored in a SmartBLOB space, however, the server provides a locking service. It is important to realize that the granularity of the locking is at the SmartBLOB level. That is if any data in a SmartBLOB is accessed, then the entire SmartBLOB is locked for the duration of the transaction. Therefore, if the external data is stored as one table per SmartBLOB, then by default, the UDAM has table-level locking.

Parallelization

To enable parallelization for a UDAM implementation, the following purpose functions (UDRs) must be parallelizable:

- ▶ am_open
- ▶ am_close
- ▶ am_beginscan
- ▶ am_endscan
- ▶ am_getnext
- ▶ am_rescan
- ▶ am_getbyid

For a SELECT query involving UDAM in parallel, these UDRs must be parallelizable. For a DELETE query involving UDAM in parallel, along with the purpose functions, the am_delete function must be parallelizable. Similarly for an UPDATE statement, the am_update function must be parallelizable. And, for INSERT statement, the am_insert function must be parallelizable.

One way to use INSERT query parallelism is to run either "INSERT INTO... SELECT" or "SELECT .. INTO TEMP .." statements in parallel.

Onstat options for UDAM

The cac option of the **onstat** utility can be used to display cached data associated with an access method. Example 10-13 shows the **onstat** command that displays the cached data for the specified access method (<access method name>). If the "am" keyword is included, but the access method name is omitted, then the cached data associated with all access methods is displayed.

Example 10-13 The onstat command to display cached data

```
onstat -g cac am <access method name>
```

The **onstat** command in Example 10-14 shows both the cached data for a virtual table and the configuration parameters that were specified when the table was created.

Example 10-14 The onstat command to display cached data and config parameters

```
onstat -g dic <table name>
```

10.2 Relational mashups

With the advent of Web services, service-oriented architecture (SOA) and Web 2.0, it is evident that centralized sources of data are a thing of the past. More and more we see disparate data sources being joined in order to extract interesting information from the huge volumes of data. This is evident in the current trend of Web 2.0 applications called *mashups*. In this section, we discuss the concept of relational mashups. The idea is to integrate multiple Web services (or for that matter, any non-relational data source) into IDS and to query the data by using simple SQL statements.

Web services rely on simple open standards, such as XML and SOAP, and are accessed through any kind of client application. Typically those applications are written in Java, C++, or C#. For those organizations who already have an existing application that is based on an SQL database and that already use business logic in the database server through UDRs, developers might want to integrate access to Web services on the SQL level.

Having Web services accessible from SQL offers the following advantages:

- ▶ Easy access through SQL and standardized APIs (as examples, ODBC and JDBC)
- ▶ Movement of the Web service results closer to the data processing in the database server which can speed up applications
- ▶ Web service access to non-Java or C++ developers

In this section, we look at how you can use the UDAM framework to access the Amazon E-Commerce Web service as a relational table. Code examples are provided as a quick start for application developers who are interested in integrating Web services with IDS.

All examples in this section have been implemented by using the following configuration:

- ▶ Red Hat Enterprise Linux ES, release 4, 32 bit
- ▶ IDS 11.10.xC1 Cheetah GA release
- ▶ gSOAP 2.7.9I for Linux
- ▶ Amazon E-Commerce WSDL

10.2.1 Web services

A *Web service* is a set of related application functions that can be programmatically invoked over the Internet. Businesses can dynamically mix and match Web services to perform complex transactions with minimal programming. Web services allow buyers and sellers all over the world to discover each other, connect dynamically, and execute transactions in real time with minimal human interaction.

Web services are self-contained, self-describing modular applications that can be published, located, and invoked across the Web:

- ▶ Self-contained

On the client side, a programming language with XML and HTTP client support is enough to get started. On the server side, a Web server and servlet engine are required. The client and server can be implemented in different environments. It is possible to Web service enable an existing application without writing a single line of code.
- ▶ Self-describing

The client and server must recognize only the format and content of request and response messages. The definition of the message format travels with the message. Therefore, no external metadata repositories or code generation tools are required.
- ▶ Modular

Simple Web services can be aggregated to form more complex Web services either by using workflow techniques or by calling lower layer Web services from a Web service implementation.

Web services can be most anything. Examples are theater review articles, weather reports, credit checks, stock quotations, travel advisories, and airline travel reservation processes. Each of these self-contained business services is an application that can easily integrate with other services from the same or different companies, to create a complete business process. This interoperability allows businesses to dynamically publish, discover, and bind a range of Web services through the Internet.

Web services consumer requirements

To call a Web service from within IDS:

1. Construct a SOAP message based on a given Web service description.
2. Send the SOAP message to the Web services provider via the required protocol (typically HTTP).
3. Receive the Web service response, parse it, and handle the results on an SQL level.

All of these tasks must be executed from the IDS SQL layer to achieve the required portability.

The gSOAP C/C++ toolkit

The gSOAP toolkit provides an easy way to generate SOAP to C/C++ language bindings combined with the advantage of a simple, but powerful API to reduce the learning curve for users who wants to get started on Web services development.

A Web service client and Web service server code can both be generated by gSOAP. In addition, gSOAP is self-contained, so that no additional libraries or products are required. This enables an easier deployment of gSOAP-based IDS extensions (DataBlades).

The gSOAP stub and skeleton compiler for C and C++ was developed by Robert van Engelen of Florida State University. See the following Web sites for more information:

- ▶ Sourceforge.net gSOAP Toolkit
<http://sourceforge.net/projects/gsoap2>
- ▶ gSOAP: C/C++ Web Services and Clients
<http://www.cs.fsu.edu/~engelen/soap.html>

gSOAP installation and configuration

To use gSOAP, first download a recent version of gSOAP for your desired development platform (UNIX, Linux, or Windows) from the following Web address:

http://sourceforge.net/project/showfiles.php?group_id=52781

Throughout the development of this book, we have used version 2.7.9l of gSOAP for Linux x86 32 bit.

After you download the gSOAP toolkit, extract the compressed file into a folder, for example `/work/gsoap-linux-2.7`. In the following sections, we refer to this gSOAP installation location as the `$(GSOAP_DIR)`.

Since we must compile C source code files, have a C-compiler installed on your development platform. For the examples in this section, we have been using gcc version 3.4.6 20060404 (Red Hat 3.4.6-3).

10.2.2 Amazon Web service

Amazon E-Commerce Service (ECS) exposes the Amazon product catalogs through an easy-to-use Web service interface. ECS offers a variety of operations or Web services that provide a range of services that deal with querying, browsing, shopping, and ordering of items from Amazon. Each operation takes in a predefined format of request and can be customized to return the level of information that we require for a response reply. We do not discuss the various operations and responses in details. If you are interested in the finer details of ECS, we encourage you to browse the documentation that is available from the following Web address:

<http://aws.amazon.com/>

ECS is free, but requires registration for an Amazon Web Services (AWS) account in order to obtain the access key and secret access key. All ECS requests require the access key to be part of the request to access Amazon data.

For the examples provided in this section, we use the `ItemLookup` and `ItemSearch` operations. The queries are restricted to Amazon only for “books.” You can download the Web Service Description Language (WSDL) for ECS from the following Web address:

<http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl?>

Tip: WSDL is an XML file that describes the Web service operation message format and the structure of request and response messages. It also describes the Web service endpoints that corresponding to each operation.

ECS `ItemLookup` operation

Given an item identifier, or for books the ISBN, the `ItemLookup` operation returns some or all of the item attributes based on the level of information requested via the response group. The `ItemLookup` operation expects the `ItemLookupRequest` message format as input and returns the `ItemLookupResponse` message format as a result of the lookup.

Table 10-3 lists some of the important parameters passed through the ItemLookupRequest message.

Table 10-3 ItemLookupRequest attributes

Parameter	Description
AWSAccessKeyId	ECS access key.
Idtype	Type of item identifier used to look up an item. We used the ISBN lookup.
ItemId	The item identifier of interest.
SearchIndex	The category in which to search. We used the category “books” for all searches.
ResponseGroup	Specifies the types of information of interest. We used ItemAttributes, BrowseNodes, and Offers to retrieve item attributes (such as Title and Author), to get a paginated response and to get the current list price and offers on the item.

In the ItemLookupResponse message, we looked at the ItemAttributes tag to retrieve the ISBN number, author name, title, and current list price for the item.

ECS ItemSearch operation

The ItemSearch operation returns a list of items that satisfy the search criteria. The search criteria can have multiple indices on which to filter. Typical searches include a search on keywords, a search for all books by a particular author, and a search by title. We used two types of search:

- ▶ Keyword, where the keyword is “All”. This translates into the most elaborate of searches where all books available on Amazon returned.
- ▶ Title, where all books matching a particular title are returned.

Table 10-4 lists some of the important parameters passed in through the ItemSearchRequest message.

Table 10-4 ItemSearchRequest attributes

Parameter	Description
AWSAccessKeyId	ECS access key.
Author	Name of author to search for.
Title	Title of book to search for.
SearchIndex	The category in which to search. We used category “books” for all our searches.

Parameter	Description
ResponseGroup	Specifies the types of information of interest. We used ItemAttributes, BrowseNodes, and Offers to retrieve item attributes (such as Title and Author), to get a paginated response and to get the current list price and offers on the item.
Keywords	Words or phrases associated with the item of interest on which to search.

In the ItemLookupResponse message, we looked at the ItemAttributes tag to retrieve ISBN number, author name, title and current list price for the item.

10.2.3 Test driving Amazon VTI

To better understand the design of the IDSAccess UDAM, try the code to see how to use the UDAM to create a VTI table and to run queries.

Note: A wsvti example is provided with this book for you to download and try. This example demonstrates the Amazon VTI capabilities. We start with gSOAP code generation and then run the actual examples. See Appendix A, “Additional material” on page 465, for details about how to download the example code from the IBM Redbooks Web site and then to use it.

Generating gSOAP code for ECS access

To generate gSOAP code for ECS access:

1. Create a working directory for the Amazon VTI files. We refer to this directory as `WSVTI` for the Web service VTI.
2. Create a subdirectory under `WSVTI` for the gSOAP generated files. We refer to this directory as `AWS` for the Amazon Web service.
3. Download the ECS WSDL file from the following Web address:

<http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl?>

Then copy the file into a local folder.

4. From a command prompt in a shell, run the following gSOAP command on the recently downloaded WSDL file:

```
${GSOAP_DIR}/bin/wsd12h -c -t ${GSOAP_DIR}/typemap.dat  
AWSECommerceService.wsdl
```

This step generates a file called *AWSECommerceService.h* in the current working directory. The `-c` option is important to generate the required C language template instead of the default C++ template.

5. From the same command prompt, generate the necessary C stubs and skeletons (.c and .h files), which integrate into an Amazon VTI implementation, by executing the following gSOAP command:

```
`${GSOAP_DIR}/bin/soapcpp2 -c -C -I`${GSOAP_DIR}/import  
AWSECommerceService.h
```

The `-C` option instructs the `soapcpp2` tool to generate the Web service client code only, and the `-c` option forces the generation of C language code instead of C++ code.

From the generated files, the more important ones are:

- *AWSECommerceService.h* contains the function prototype and C structure definitions for response and request messages.
 - *soapClient.c* contains the implementation of client library functions to invoke Amazon Web services through SOAP.
6. To test the installation, write a simple stand-alone C program to retrieve data from ECS. Copy the C source code from Example 10-15 into `${WSVTI}/test.c`. Then replace `<<Replace with Access Key>>` with your Amazon access key.

Example 10-15 C source code

```
#include "soapH.h"  
#include "AWSECommerceServiceBinding.nsmap"  
  
int main()  
{  
    struct soap soap;  
    int i = 0;  
    struct _ns1__ItemLookupResponse r;  
    struct _ns1__ItemLookup in;  
    struct ns1__ItemLookupRequest req;  
    enum _ns1__ItemLookupRequest_IdType idt =  
_ns1__ItemLookupRequest_IdType__ISBN;  
    char *isbn = "0091883768";  
    char *repGrp[5] =  
{"ItemAttributes", "Images", "EditorialReview", "Offers", "BrowseNodes"}  
;  
  
    in.Request = &req;  
    in.__sizeRequest = 1;  
    in.AWSAccessKeyId = <<Replace with Access Key>>;
```

```

req.ResponseGroup = repGrp;
req.__sizeResponseGroup = 5;
req.IdType = &idt;
req.__sizeItemId = 1;
req.ItemId = &isbn;
req.SearchIndex = "Books";
soap_init(&soap);
if (soap_call__ns1__ItemLookup(&soap, NULL, NULL, &in, &r))
    soap_print_fault(&soap, stderr);
else
{
    if (r.ns1__OperationRequest &&
        r.ns1__OperationRequest->ns1__Errors)
    {
        printf("%d\n",
r.ns1__OperationRequest->ns1__Errors->__sizeError);
        printf("%s\n",
r.ns1__OperationRequest->ns1__Errors->Error->Code);
    }
    for (i = 0 ; i < r.ns1__Items->__sizeItem ; i++)
    {
        printf("ISBN: %s\n", isbn);
        printf("Book Title: %s\n",
r.ns1__Items->ns1__Item[i].ns1__ItemAttributes->Title);
        printf("Book Price: %s\n",
r.ns1__Items->ns1__Item[i].ns1__ItemAttributes->ListPrice->Formatted
Price);
    }
}
    soap_end(&soap);
    soap_done(&soap);
    return 0;
}

```

Compile test.c by using the commands shown in Example 10-16.

Example 10-16 Compile commands

```

gcc -Wall -g -m32 -I. -I${GSOAP_DIR} -o soapC.o -c soapC.c
gcc -Wall -g -m32 -I. -I${GSOAP_DIR} -o soapClient.o -c
soapClient.c
gcc -Wall -g -m32 -I. -I${GSOAP_DIR} -o stdsoap2.o -c
${GSOAP_DIR}/stdsoap2.c
gcc -Wall -g -m32 -I. -I${GSOAP_DIR} -o test test.c soapC.o
soapClient.o stdsoap2.o

```

Running the compiled test binary should yield results similar to output shown in Example 10-17.

Example 10-17 Sample output

```
> ./test
ISBN: 0091883768
Book Title: Who Moved My Cheese?
Book Price: $20.65
```

Running the Amazon VTI example

Having downloaded gSOAP and generated code for ECS access by using gSOAP, you can test the Amazon VTI source example provided with this book:

1. Download the `wsvti` example tar ball and extract the contents of the tar file under `$(WSVTI)`. The directory contents after extraction should be similar to output shown in Example 10-18.
 - The `aws` directory is the `$(AWS)`, which contains the code generated by gSOAP for ECS access.
 - The `test` directory holds sample SQL scripts to test Amazon VTI.
 - `wsutil.c` holds utility functions for IDSAmericanAccess UDAM.
 - `wsvti.c` and `wsvti.h` contain the IDSAmericanAccess UDAM purpose function definitions and implementation.
 - `wsvti.sql` is the script used to register IDSAmericanAccess UDAM with the IDS database.
 - `mallocfix.c` is memory allocation wrapper code that is linked with the gSOAP generated code to deflect all gSOAP memory allocation calls, such as `malloc` and `free`, to IDS DataBlade API memory allocators. The IDS DataBlade API manages all memory allocations needed by a UDR, provides memory durations to efficiently control memory consumption, and performs automatic garbage collection of memory when memory duration expires. In addition, memory allocated through the DataBlade API is accessible across threads running in a multiple virtual processor environment. Hence UDRs are required to use DataBlade API memory allocators for their memory needs.

Example 10-18 Sample directory contents

```
bash-3.00$ ls -l
total 48
drwxr-xr-x  2 vshenoi rand  4096 Nov 11 15:02 aws
-rw-r--r--  1 vshenoi rand  2760 Nov 11 15:03 Makefile
-r--r--r--  1 vshenoi rand  3081 Oct 19 16:03 mallocfix.c
```



```

-rw-r--r-- 1 vshenoi rand 2756 Oct 19 10:59 README.txt
drwxrwxrwx 2 vshenoi rand 256 Nov 11 14:23 test
-rw-r--r-- 1 vshenoi rand 7416 Oct 22 13:31 wstutil.c
-rw-r--r-- 1 vshenoi rand 12908 Oct 23 13:27 wsvti.c
-rw-r--r-- 1 vshenoi rand 1456 Oct 22 21:21 wsvti.h
-rw-r--r-- 1 vshenoi rand 1871 Oct 23 10:38 wsvti.sql

```

2. Edit `wsvti.h` and change the macro `AWSACCESSKEYID` to use your Amazon access key.
3. Run the `MAKE` command in the `${WSVTI}` directory to generate the `IDSAmazonAccess UDAM` shared object. The shared object name is `wsvti.bld` and can be found in `${WSVTI}/linux-intel/`.
4. Running as user `informix`, create the `$INFORMIXDIR/extend/wsvti` directory.
5. Running as user `informix`, copy `${WSVTI}/linux-intel/wsvti.bld` to `$INFORMIXDIR/extend/wsvti`. This step ensures that the `IDSAmazonAccess UDAM` is copied into a common place where all IDS extensions reside.
6. Configure IDS to run the `IDSAmazonAccess UDAM` code in a separate virtual processor, called `wsvp`. By adding a dedicated virtual processor class to the IDS configuration, we can separate the execution of the blocking network calls of the Web service consumer UDAM from the overall IDS query processing. To enable at least one dedicated virtual processor class for that purpose, add the following line to the `ONCONFIG` file of your IDS instance:

```
VPCLASS wsvp,num=1
```

7. Start, or restart, the IDS instance.
8. Execute the `onstat -g glo` command.

You should now see the additional virtual processor listed, as illustrated in Example 10-19.

Example 10-19 The results for onstat

```
bash-3.00$ onstat -g glo
```

```
IBM Informix Dynamic Server Version 11.10.UC1 -- On-Line -- Up 00:12:31 --
36240 Kbytes
```

```
MT global info:
```

```

sessions threads vps      lngspins
0          18      12      0

```

```

          sched calls      thread switches yield 0   yield n   yield forever
total:    6376132          1198269          5762089   17341    1039
per sec:  0              0              0          0         0

```

Virtual processor summary:

class	vps	usercpu	syscpu	total
cpu	1	2.35	0.76	3.11
aio	6	0.00	0.00	0.00
lio	1	0.00	0.00	0.00
pio	1	0.00	0.00	0.00
adm	1	0.00	0.00	0.00
msc	1	0.00	0.00	0.00
wsvp	1	0.02	0.00	0.02
total	12	2.37	0.76	3.13

Individual virtual processors:

vp	pid	class	usercpu	syscpu	total
1	2065	cpu	2.35	0.76	3.11
2	2066	adm	0.00	0.00	0.00
3	2067	lio	0.00	0.00	0.00
4	2068	pio	0.00	0.00	0.00
5	2070	aio	0.00	0.00	0.00
6	2071	msc	0.00	0.00	0.00
7	2072	aio	0.00	0.00	0.00
8	2073	aio	0.00	0.00	0.00
9	2074	aio	0.00	0.00	0.00
10	2075	aio	0.00	0.00	0.00
11	2076	aio	0.00	0.00	0.00
12	2088	wsvp	0.02	0.00	0.02
		tot	2.37	0.76	3.13

-
9. Create a new database to work with as shown in Example 10-20.

Example 10-20 New database for use with example

```
bash-3.00$ dbaccess - -  
> create database tryamazon with log;
```

Database created.

10. Register the UDAM in the tryamazon database by using the wsvti.sql script as shown in Example 10-21.

Example 10-21 Sample script for testing

```
bash-3.00$ dbaccess tryamazon wsvti.sql
```

Database selected.

Routine created.

Routine created.

Routine created.

Routine created.
Routine created.
Routine created.
Routine created.
Routine created.
Routine created.

1 row(s) inserted.

Access_method created.

Database closed.

11. Run the test script provided in the `$(WSVTI)/test` directory. The test script creates two relational tables, `customer` and `orders`, and then populates data into those tables. Finally it creates the Amazon VTI table, called `AmazonTable` (shown in Example 10-22), and performs a join between all three tables.

Example 10-22 Example table create

```
CREATE TABLE AmazonTable
(
  ISBN char(10),
  TITLE lvarchar,
  PRICE decimal
) USING IDSAmericanAccess ;
```

12. The join returns the latest price of a book from Amazon based on an ISBN order from a particular customer. The query is shown in Example 10-23. Refer to “Design considerations” on page 440 for details about the need for the `USE_NL` optimizer directive when performing joins involving `AmazonTable`.

Example 10-23 Example query for Amazon access

```
SELECT --+USE_NL(AmazonTable)
order_num, AmazonTable.*, fname, lname, order_date
FROM AmazonTable, orders, customer
WHERE AmazonTable.isbn = orders.isbn
AND orders.customer_num = customer.customer_num;
```

Example 10-24 shows the results of executing the test script.

Example 10-24 Script results

```
bash-3.00$ dbaccess -e tryamazon tryit.sql
```

Database selected.

```
create table customer
(
  customer_num      serial(101),
  fname             char(15),
  lname             char(15),
  company           char(20),
  address1          char(20),
  address2          char(20),
  city              char(15),
  state             char(2),
  zipcode           char(5),
  phone             char(18),
  primary key (customer_num)
);
```

Table created.

```
create table orders
(
  order_num         serial(1001),
  order_date        date,
  customer_num      integer not null,
  ship_instruct     char(40),
  backlog           char(1),
  ISBN              char(10),
  ship_date         date,
  primary key (order_num),
  foreign key (customer_num) references customer (customer_num)
);
```

Table created.

```
load from 'customer.unl' insert into customer;
28 row(s) loaded.
```

```
load from 'orders.unl' insert into orders;
4 row(s) loaded.
```

```
CREATE TABLE AmazonTable
```

```

(
  ISBN char(10),
  TITLE lvarchar,
  PRICE decimal
) USING IDSAmericanAccess ;
Table created.

SELECT FIRST 5 * from AmazonTable;

isbn    1401908810
title   Spiritual Connections: How to Find Spirituality Throughout All
the Relat
        ionships in Your Life
price   24.95000000000000

isbn    0743292855
title   Paula Deen: It Ain't All About the Cookin'
price   25.00000000000000

isbn    1591391105
title   The First 90 Days: Critical Success Strategies for New Leaders
at All Le
        vels
price   27.95000000000000

isbn    0072257121
title   CISSP All-in-One Exam Guide, Third Edition (All-in-One)
price   79.99000000000000

isbn    0071410155
title   The Six Sigma Handbook: The Complete Guide for Greenbelts,
Blackbelts, a
        nd Managers at All Levels, Revised and Expanded Edition
price   89.95000000000000

5 row(s) retrieved.

select --+USE_NL(AmazonTable)
order_num,
AmazonTable.*, fname, lname, order_date from AmazonTable, orders,
customer where (AmazonTable.isbn = orders.isbn) and orders.customer_num
= customer.customer_num;

order_num    1001
isbn         0091883768

```

```

title      Who Moved My Cheese?
price      20.6500000000000
fname      Anthony
lname      Higgins
order_date 05/20/1998

order_num   1002
isbn        0091883768
title       Who Moved My Cheese?
price       20.6500000000000
fname       Ludwig
lname       Pauli
order_date 05/21/1998

order_num   1003
isbn        0954681320
title       Six Sigma and Minitab: A complete toolbox guide for all Six
Sigma p
            ractitioners (2nd edition)
price       49.9900000000000
fname       Anthony
lname       Higgins
order_date 05/22/1998

order_num   1004
isbn        0764525557
title       Gardening All-in-One for Dummies
price       29.9900000000000
fname       George
lname       Watson
order_date 05/22/1998

```

4 row(s) retrieved.

```

DROP TABLE AmazonTable;
Table dropped.
DROP TABLE customer;
Table dropped.
DROP TABLE orders;
Table dropped.

```

Database closed.

You can see the power of the UDAM framework in the Example 10-24 on page 434. IDSAmericanAccess UDAM can provide a relational cloak to the Amazon VTI table. SELECTing from Amazon VTI table performs an ECS operation that is translated into relational rows by the UDAM and is available for further relational operations. The join example shows how Amazon VTI table can participate in complex SQL statements and work in tandem with relation tables. The Amazon VTI table can be used in all SQL constructs where VTI tables can appear.

10.2.4 Amazon VTI architecture

The basic architecture is to expose the ItemLookup and ItemSearch ECS operations as VTI tables. You must be able to translate a query on the Amazon VTI table into Web service operations involving a search for items on ISBN or title keywords. A query on the Amazon VTI should, at a minimum, return the following information:

- ▶ ISBN number
- ▶ Title of the book
- ▶ Current list price

Figure 10-9 shows the architectural diagram of the Amazon VTI implementation.

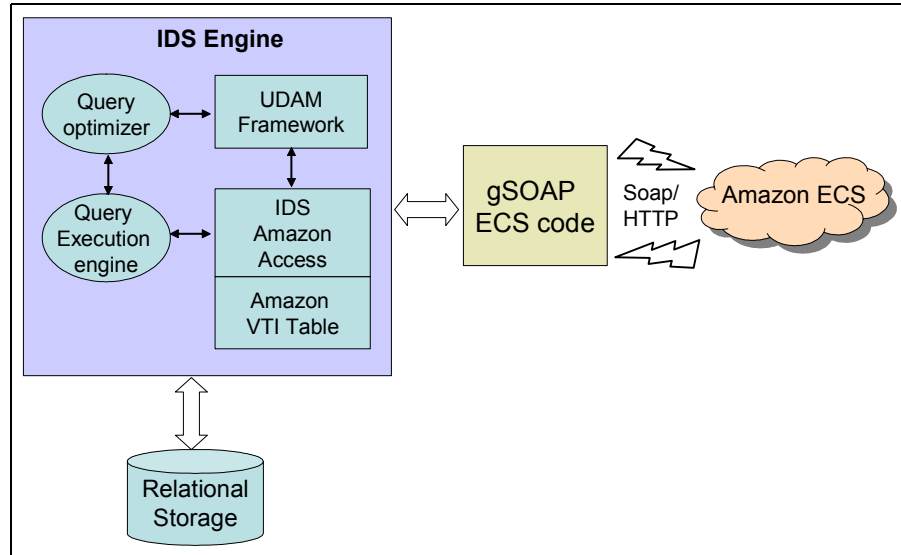


Figure 10-9 Amazon VTI implementation

The IDS SQL query optimizer and query execution engine communicate with the Amazon VTI implementation through the IDSAmericanAccess UDAM. In order to

feed data to the IDS SQL engine, the IDSAmericanAccess UDAM uses gSOAP-generated ECS access code to connect to the Amazon ECS Web service. The UDAM framework and IDSAmericanAccess UDAM provide the relational conversion functions on top of the Amazon ECS Web service.

Example 10-25 shows the Amazon VTI table schema. AmazonTable has an ISBN column that allows ECS ISBN lookup type of operations. AmazonTable also has the TITLE column that allows ECS Title Lookup type of operations. The PRICE column is an information only column (no filters allowed) that carries the ECS ListPrice of a queried item. This is a basic schema and can be expanded to include any other information available through ECS.

Example 10-25 Amazon VTI table schema

```
CREATE TABLE AmazonTable
(
  ISBN char(10),
  TITLE lvarchar,
  PRICE decimal
) USING IDSAmericanAccess ;
```

Example 10-26 shows an excerpt from `WSVTI/wsvti.sql`, which creates the IDSAmericanAccess UDAM. IDSAmericanAccess is defined as a *primary access method* with multiple purpose functions.

Example 10-26 Example access method

```
CREATE PRIMARY ACCESS_METHOD IDSAmericanAccess(
am_open = wsvti_open,
am_rescan = wsvti_rescan,
am_close = wsvti_close,
am_drop = wsvti_drop,
am_beginscan = wsvti_beginscan,
am_endscan = wsvti_endscan,
am_getnext = wsvti_getnext,
am_scancost=wsvti_scancost,
am_sptype = "A");
```

You have now seen how an SQL query generates a flow of purpose function calls in 10.1.3, “Flow of DML and DDL with virtual tables and indices” on page 414. With that in mind, Table 10-5 lists the purpose functions and the tasks that they perform.

Table 10-5 Purpose functions

Purpose function	Task description
wsvti_open	<ul style="list-style-type: none"> ▶ Allocates gSOAP request and response structures. ▶ Caches these structures in user data to be used by wsvti_beginscan, wsvti_getnext, and wsvti_endscan.
wsvti_close	<ul style="list-style-type: none"> ▶ Frees cached gSOAP structures.
wsvti_drop	<ul style="list-style-type: none"> ▶ Empty function, since we do not need to do anything when the VTI table is dropped.
wsvti_scancost	<ul style="list-style-type: none"> ▶ If qualifiers are present, returns a lower cost than if they are not present. This helps in cases where joins are performed with the VTI table being the inner table. Refer to “Design considerations” on page 440 for more details.
wsvti_beginscan	<ul style="list-style-type: none"> ▶ Decides which ECS request to use based on the qualifier columns. Refer to “Design considerations” on page 440 for more details. ▶ Does not generate any SOAP messages or communication, but initializes the SOAP engine.
wsvti_getnext	<ul style="list-style-type: none"> ▶ Converts qualifier constants into appropriate char strings holding ECS request parameters. ▶ Uses gSOAP calls to perform the actual ECS operation. ▶ Retrieves results and formats the result into relational rows using mi_row_create(). ▶ Returns the rows to the SQL engine. ▶ Returns a maximum of 200 rows for unbounded queries.
wsvti_endscan	<ul style="list-style-type: none"> ▶ Uninitializes SOAP engine.
wsvti_rescan	<ul style="list-style-type: none"> ▶ Performs a quick wsvti_endscan and wsvti_beginscan.

Design considerations

The Amazon VTI architecture has the following design considerations:

- ▶ Since we are interested only in querying data, the IDSAccess UDAM is a read-only UDAM.
- ▶ Amazon data is vast, and querying the entire data without any filtering seems unreasonable and unpractical. This means that when you run the following statement, you must limit the results returned by the ECS operation in order to have realistic query completion times:

```
SELECT * FROM AmazonTable;
```

The IDSAccess UDAM retrieves a maximum of 200 rows when there are no filters or qualifiers supplied in the query. Unbounded queries are translated into ItemSearch ECS operations with Keywords=All in the request.

- ▶ In order to filter data from ECS, you need qualifiers to be passed into IDSAccess UDAM. In turn, this means that the VTI table must have columns, so that filters on the column translate into UDAM qualifiers. Hence, AmazonTable schema provides ISBN and TITLE as the two columns on which filters can be defined.
- ▶ If there are no qualifiers, then wsvti_beginscan() uses ItemLookup with Keywords=All request. If the qualifier column is ISBN, then it uses the ItemLookup request. If the qualifier column is TITLE, then it uses the ItemSearch request with the Title pattern passed in. Complex qualifiers are not supported in this implementation, although it is easy to extend the current implementation to perform complex filtering.
- ▶ Join queries involving AmazonTable must result in the AmazonTable being the inner query of the join. Assuming the query from Example 10-23 on page 433, if AmazonTable is the outer table, then during query execution, the IDS SQL engine must fetch all rows from AmazonTable to join with the inner tables (customer, orders). Any join filters passed (AmazonTable.ISBN=orders.ISBN) are not passed into the UDAM as qualifiers, but are evaluated as a join filter. This results in an unbounded query on AmazonTable, which is impractical. Hence, AmazonTable should always be the inner table in a join. Therefore, Example 10-23 uses the --+ USE_NL(AmazonTable) optimizer directive to force the desired join order.

10.3 WebSphere MQ virtual tables

WebSphere Message Queue (MQ) software suite provides reliable messaging for distributed, heterogeneous applications to exchange information, delegate jobs, coordinate events, and create an enterprise service bus (ESB). When Informix applications use WebSphere MQ, you write custom code, manage multiple connections, and route data through your application.

IDS version 10.00.UC3, introduced built-in support for Informix applications to interact with WebSphere MQ via SQL callable functions with two-phase commit support. This eliminates development overhead and encapsulates integration complexity.

10.3.1 WebSphere MQ

In its simplest form, WebSphere MQ is a method to exchange messages between two end points. It acts as an intermediary between two systems and provides value added functionalities such as reliability and transactional semantics.

Whether you buy a book on amazon.com or enroll in e-business with ibm.com, the order event triggers a workflow of the information through multiple modules. These modules include user account management, billing, packaging and shipping, procurement, customer service, and partner services. The execution in triggered modules generates a subsequent workflow. To meet reliability and scaling requirements, it is typical to have application modules on multiple machines.

If you are using the same software on all systems, for instance an SAP® stack, the software itself usually comes with workflow management features. If the modules are running in a homogeneous environment, such as LINUX machines running WebSphere and Informix, it is easier to change information via distributed queries or enterprise replication. Alternatively, the application might be running on heterogeneous systems, such as a combination of WebSphere, DB2, Oracle and Informix. In this case, programming and setup of distributed queries or replication becomes complex, and in many cases, does not meet the application requirements.

Figure 10-10 illustrates the application integration.

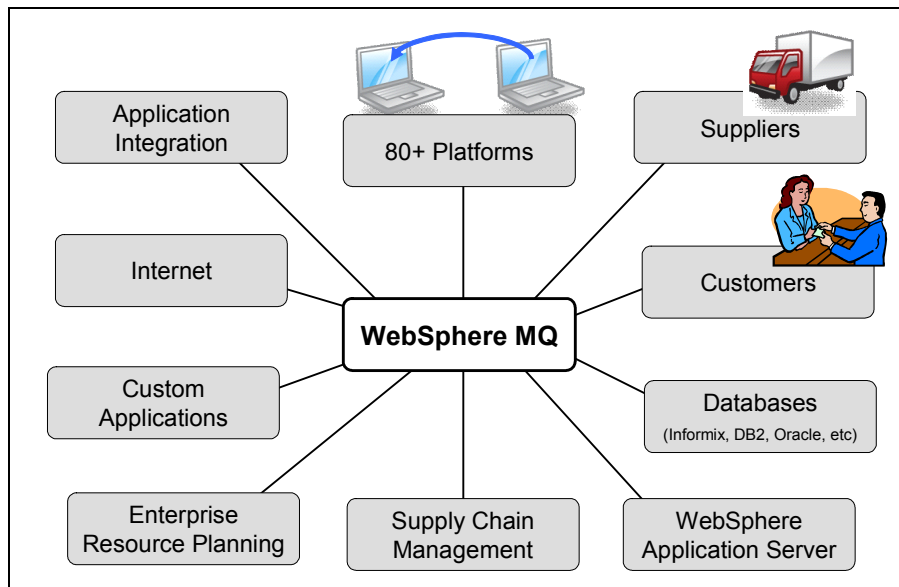


Figure 10-10 WebSphere MQ for enterprise business application integration

WebSphere MQ is designed to address integration issues such as those shown in Figure 10-10. It prefers no platform and enforces no paradigms. WebSphere MQ supports more than 80 platforms and APIs in C, C++, Java, Java Message Service (JMS), and Visual Basic®. WebSphere MQ is also the mainstay for designing an ESB for SOA.

WebSphere MQ provides a reliable store-and-forward mechanism, so that each module can send and receive messages to and from it. WebSphere MQ achieves this by persistent queues and APIs for programming. In addition, WebSphere MQ Message Broker, another software product in the WebSphere MQ product suite, provides message routing and translation services. Simplicity of infrastructure means the applications must establish message formats, queue attributes, and so on. WebSphere MQ also supports publish and subscribe semantics for queues, making it easy to send a single message to multiple receivers and subscribing messages from queue by need, similar to a mailing list.

The applications predetermine queue names, messages, and message formats, just as the two network applications agree on socket numbers. The application-to-application message exchange is asynchronous. That is, one application does not wait for another application to receive the message. WebSphere MQ ensures that the message is stored reliably and will have the

message available for target application. The target application is responsible for receiving the message from WebSphere MQ.

10.3.2 How Informix and other database applications use WebSphere MQ

Applications have many input sources, such as user entry, business-to-business transactions, workflow messages, and data in the database. A sample order entry application must store data in the Informix database and send and receive messages to WebSphere MQ. The application establishes connections with Informix and WebSphere MQ. In addition, the application uses a transaction manager to ensure reliability of the data exchange. For instance, the order saved in the database must be sent to a queue and marked as processed in the database. The order can be marked as processed only after WebSphere MQ receives the message successfully. Therefore, the interaction must have transactional protection. Figure 10-11 illustrates this type of environment.

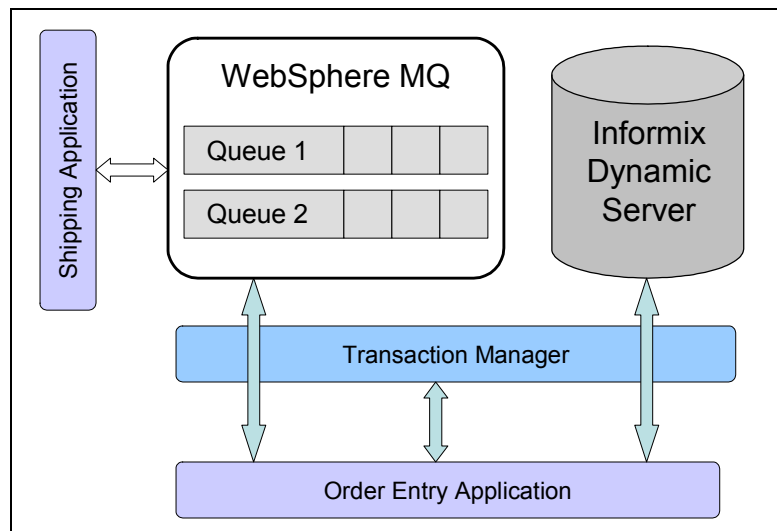


Figure 10-11 Applications using WebSphere MQ

The order entry application writes custom code to exchange messages from and to WebSphere MQ. Developing custom code every time an application wants to interact with WebSphere MQ is costly. It requires you to train the programmers for it or hire consultants to develop, debug, and maintain this code, and to modify the code for new queues and applications. The data exchanged between the database and WebSphere MQ flows is through the application, which is not efficient for high data volumes and necessitates a transaction manager.

10.3.3 IDS support for WebSphere MQ

IDS provides SQL callable functions to read, receive, send, subscribe, unsubscribe and publish, which are illustrated in Figure 10-12. These SQL callable functions expose WebSphere MQ features to the IDS application and integrate the WebSphere MQ operations into IDS transactions. That is, the fate of the WebSphere MQ operation is tied to the fate of the transaction. If the transaction is rolled back, the operations made on WebSphere MQ, messages sent or received, are rolled back. This is done by coordinating transactions at the IDS and WebSphere MQ, not by a compensating transaction. This is reliability with high performance.

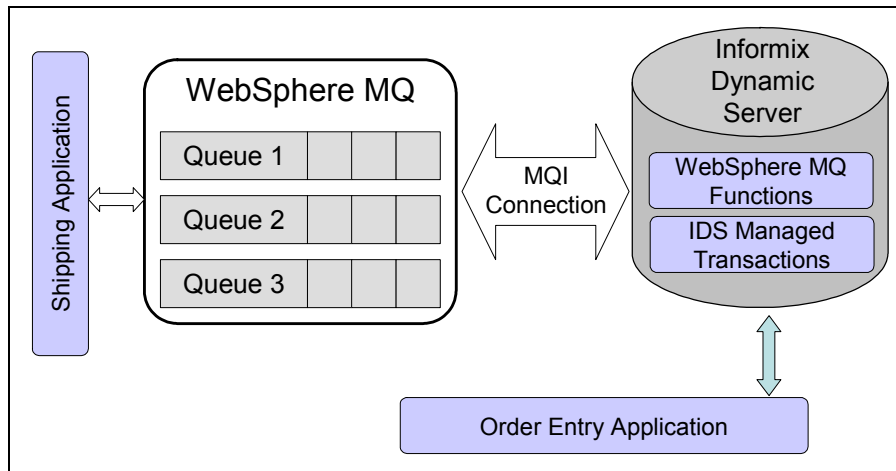


Figure 10-12 WebSphere MQ - Sending and receiving messages

Example 10-27 shows how you can easily use IDS WebSphere MQ functionality to send and receive a message to and from a WebSphere MQ queue.

Example 10-27 WebSphere MQ - Sending and receiving a message

```
select MQSend("CreditService", customerid || ":" || address || ":" ||
product ":" || orderid)
from order_tab
where customerid = 1234;

insert into shipping_tab(shipping_msg) values( MQReceive() );

create function get_my_order() returns int;

define cust_msg lvarchar(2048);
define customerid char(12);
```

```

define address char(64);
define product char(12);
define corderid char(12);
define snd_status int;

-- Get the order from Order entry application.
execute function MQReceive("OrderQueue") into cust_msg;
let customerid = substr(cust_msg, 1, 12);
let address = substr(cust_msg, 14, 77);
let product = substr(cust_msg, 79, 90);
let corderid = substr(cust_msg, 92, 103);

insert into shipping_table(custid, addr, productid, orderid)
  Values(customerid, address, product, corderid);

-- send the status to CRM application
execute function MQSend("CRMQueue", corderid || ":IN-SHIPPING") into
snd_status;
return 1;
end function;

```

When you roll back the transaction, as shown in Example 10-28, the message received is restored in the queue book order, and the row is also removed from shipping_tab.

Example 10-28 Rollback of the transaction

```

begin work;
INSERT into shipping_tab(shipping_msg)
  values (MQReceive("bookorderservice"));

rollback work; -- Undo previous statement including WMQ operation

```

10.3.4 Programming for WebSphere MQ

IDS provides functions that exposing each interface provided by WebSphere MQ, such as read, receive, send, publish, subscribe, and unsubscribe, and functions to send and receive large messages. The WebSphere MQ functions can be invoked anywhere a function can be used, such as in value clauses, projection lists, query filters, stored procedures, and triggers. In addition, with IDS, you can map a WebSphere MQ queue into an IDS table. An insert on this table translates to a send operation to the WebSphere MQ, and a select translates to either a read or a receive.

While WebSphere MQ provides simple abstractions of queues and its operations, each operation comes with many options, such as msg expiry time and retry count. IDS has abstracted these options into service, policy, and optionally correlation ID:

► Service

Service maps a queue, queue manager, and code set of the messages into the service. The table “informix”.mqiservice stores the mappings. IDS.DEFAULT.SERVICE is mapped to system default queue manager, queue named IDS.DEFAULT.QUEUE, and default code set.

► Policy

Policy defines the attributes, such as priority and expiry date, for each operation. The table “informix”.mqipolicy stores 37 attributes for each policy. IDS.DEFAULT.POLICY is the default policy. Depending on your application environment, create one or more policies.

► Correlation ID

When multiple applications share the same queue, you can control the interaction by using a correlation ID of up to 48 bytes. When the applications agree on the correlation ID for their messages, they can get messages that match their correlation ID. They work similar to filters or predicates in SQL queries. Correlation ID is not mandatory and has no default value.

Basic programming functions

Table 10-6 lists the MQ programming functions in IDS.

Table 10-6 MQ functions in IDS

Function name	Description
MQSend()	Sends a string message to a queue
MQSendClob()	Sends CLOB data to a queue
MQRead()	Reads a string message in the queue into IDS without removing it from the queue
MQReadClob()	Reads a CLOB in the queue into IDS without removing it from the queue
MQReceive()	Receives a string message in the queue into IDS and remove it from the queue
MQReceiveClob()	Receives a CLOB in the queue into IDS and remove it from the queue
MQSubscribe()	Subscribes to a topic

Function name	Description
MQUnSubscribe()	Unsubscribes from a previously subscribed topic
MQPublish()	Publishes a message into a topic
MQPublishClob()	Publishes a CLOB into a topic
CreateMQVTIRead()	Creates a read VTI table and maps it to a queue
CreateMQVTIReceive()	Creates a receive VTI table and maps it to a queue
MQTrace()	Traces the execution of MQ functions
MQVersion()	Gets the version of MQ functions

Functions for sending messages from IDS to WebSphere MQ

The following functions are for sending messages to IDS and WebSphere MQ:

- ▶ MQSend(Service, Service_Policy, Message, CorrelationID)
- ▶ MQSendClob(Service, Service_Policy, ClobMessage, CorrelationID)

You can send a message of up to 32,739 bytes to a WebSphere MQ queue. MQSendClob() behaves the same as MQSend(), except that it takes CLOB as its message parameter instead of character type. Message and ClobMessage are the mandatory parameters. IDS sends the message to the queue managed by the queue manager by using the policy in the service record entry saved in the "informix".mqiservice table.

Parameter interpretation

The following function describes how the calls are interpreted for MQSend(). The other functions follow the same pattern. When the four parameters are given, translation is straightforward and is executed as given:

```
MQSend(serviceparam, policyparam, messageparam, correlationparam)
```

The following translation applies when one or more parameters are missing:

- ▶ MQsend(messageparam) is translated as follows:

```
MQSend("IDS.DEFAULT.SERVICE", "IDS.DEFAULT_POLICY",
messageparam, "");
```

- ▶ MQsend(serviceparam, messageparam) is translated as follows:

```
MQSend(serviceparam, "IDS.DEFAULT_POLICY", messageparam, "");
```

- ▶ MQsend(serviceparam, policyparam, messageparam) is translated as follows:

```
MQSend(serviceparam, policyparam, messageparam, "");
```

Examples

Example 10-29 shows how to use these parameters.

Example 10-29 Parameters

```
SELECT MQSend("myservice", "mypolicy", orderid || ":" || address)
FROM   tab
WHERE  orderid = 12345;
```

All WebSphere MQ functions should run in a transaction. In IDS, SELECT, UPDATE, DELETE, and INSERT automatically start a new transaction. Alternatively, you can start a new transaction with a BEGIN WORK statement. Simply executing the function gives the error shown in Example 10-30.

Example 10-30 Error when executing an MQ function

```
EXECUTE FUNCTION MQSend("MyService",
    "<order><id>5</id><custid>6789</custid></order>");
```

IDS does not implicitly start a new transaction for the EXECUTE statement. Therefore, you must start a transaction explicitly as shown in Example 10-31.

Example 10-31 Starting a transaction

```
BEGIN WORK;
EXECUTE FUNCTION MQSend("MyService",
    "<order><id>5</id><custid>6789</custid></order>");
COMMIT WORK;
```

If the transaction gets rolled back, all operations on WebSphere MQ are rolled back, just as IDS rolls back its changes. See Example 10-32.

Example 10-32 Rollback of operations

```
BEGIN WORK;
INSERT INTO resultstab(sendval)
    VALUES(MQSend("MyService",
        "<order><id>5</id><custid>6789</custid></order>")
ROLLBACK WORK;
```

Read and receive functions

The following are examples of the read and receive functions:

- ▶ MQRead(Service, Policy, CorrelationID) returns lvarchar(32739).
- ▶ MQReadClob(Service, Policy, CorrelationID) returns CLOB.
- ▶ MQReceive(Service, Policy, CorrelationID) returns lvarchar(32739).
- ▶ MQReceiveClob(Service, Policy, CorrelationID) returns CLOB.

The read operation gets the message from the queue without deleting the message from the queue. The receive operation removes the message from the queue and gets the message. These functions can be called with zero or more parameters. The parameters are interpreted similar to MQSend(). The transactional behavior of the receive functions is the same as with MQSend.

MQRead() and MQReceive() can return up to 32,739 bytes. The maximum size of the message itself is a WebSphere MQ configuration parameter. The larger messages should be read or received as a CLOB. For MQ, a message is a message. Depending on the length, IDS differentiates between messages to map the messages to data types.

If a correlation ID is given, WebSphere MQ gets the next message in the queue matching the correlation ID. Otherwise a NULL message is returned. The policy determines the wait time when no applicable message is present in the queue. Therefore, by using a predefined correlation ID, multiple applications can share the same queue and for different purposes.

Consider the statements shown in Example 10-33.

Example 10-33 Read and receive functions

```
SELECT mqread('SHIPPING.SERVICE','My.DEFAULT.POLICY')
      FROM systables where tabid = 1;
```

```
SELECT mqreceive('SHIPPING.SERVICE','My.DEFAULT.POLICY')
      FROM systables where tabid = 1;
```

Publish and subscribe functions

Publishing and subscribing to a queue have an effective configuration for exchanging information between multiple applications on multiple subjects. When an order entry must go to credit card, shipping, customer relationship management (CRM), and partner applications, the order entry application publishes the order once to a queue. Target applications can subscribe to the queue and obtain the message by using either a read or receive function. Within this scheme, WebSphere MQ also supports categorizing messages into topics for finer control. For example, the order entry message can categorize the order as books, electronics, and clothing topics, for example.

You must configure the queue for publishing and define the topics. With WebSphere MQ, you can define topics statically or dynamically. The message broker provides the publish and subscribe features and must be running in addition to the queue manager. The message broker component provides message routing and message translation, easing business integration challenges.

Subscribers subscribe to a topic and specify the queue on which to receive the messages. When a publisher inserts a message on that topic into the queue, the WebSphere MQ broker routes the messages to all of the queues of each specified subscriber. The subscribers retrieve the message from the queue by using read or receive functions.

“informix”.mqipubsub table: Before using the publish and subscribe services, you must set up the “informix”.mqipubsub table. See the IDS documentation for its schema and examples.

The publish and subscribe functions are as follows:

- ▶ MQPublish(publisher_name, policyparam, message, topic, correlationid)
- ▶ MQSubscribe(subscriber_name, policy_name, topic)
- ▶ MQUnsubscribe(subscriber_name, policy_name, topic)

The publisher name and subscriber names must be defined in the “informix”.mqipubsub table. Consider the usage samples in Example 10-34.

Example 10-34 Publish and subscribe functions

```
SELECT mqSubscribe('WeatherChannel', "Weather")
      FROM systables WHERE tabid = 1;
```

```
SELECT mqPublish('WeatherChannel',
"<weather><zip>94501</zip><date>7/27/2006</date><high>89</high><low>59<
/low></weather>", "Weather")
      FROM systables WHERE tabid = 1;
```

```
SELECT mqreceive('WeatherChannel', "Weather")
      FROM systables WHERE tabid = 1;
```

MQ utility functions

The utility functions are MQVersion() and MQTrace():

- ▶ MQVersion() returns the current version of the WebSphere MQ blade in IDS.
- ▶ MQTrace(trace_level, trace_file) enables you to trace the execution path of WebSphere MQ functions and interaction between IDS and MQ. The tracing level can be from 10 to 50, in multiples of 10.

Example 10-35 shows the trace output.

Example 10-35 Trace output

```
14:19:38 Trace ON level : 50
14:19:47 >>ENTER : mqSend<<
14:19:47   status:corrid is null
14:19:47 >>ENTER : MqOpen<<
14:19:47   status:MqOpen @ build_get_mq_cache()
14:19:47 >>ENTER : build_get_mq_cache<<
14:19:47   status:build_get_mq_cache @ mi_get_database_info()
14:19:47   status:build_get_mq_cache @ build_mq_service_cache()
14:19:47 >>ENTER : build_mq_service_cache<<
14:19:47 <<EXIT : build_mq_service_cache>>
```

10.3.5 MQ table mapping functions

Invoking the WebSphere MQ functions in IDS is relatively simple. For example, IDS can map a WebSphere MQ queue to an IDS table. Performing a SELECT on the table fetches the messages in the queue, and an INSERT on the table sends the message. We discuss its usage in this section. Other operations, such as UPDATE and DELETE, on the table are not allowed.

The MQ table mapping functions are as follows:

- ▶ MQCreateVtiRead(readtable, servicename, policy, maxMessage)
- ▶ MQCreateVtiReceive(receivetable, servicename, policy, maxMessage)

A SELECT on a read table imitates MQRead(). It fetches the message without deleting it from the queue, where a SELECT on a receive table deletes the message on the queue as well. The maxMessage parameter determines the size of the column, but it also determines the type of message. A positive length creates a column. The maximum length of the message defined is 32607. Use -1 as maxMessage to retrieve the message as a CLOB, and use -2 to retrieve the message as a BLOB.

Example 10-36 shows how to create a read table.

Example 10-36 Creating a read table

```
-- Create a READ table with max message length 4096.
execute function MQCreateVTIREAD("myreadtable",
                                "myservice", "mypolicy", 4096);

-- Below is the table created by MQCreateVTIREAD() function.

create table myreadtab
  ( msg      lvarchar(4096),
    correlid varchar(24),
    topic    varchar(40),
    qname    varchar(48),
    msgid    varchar(12),
    msgformat varchar(8))
using "informix".mq (SERVICE = "myservice", POLICY = "mypolicy", ACCESS
= "READ");

-- Get the top 10 messages from the queue.
SELECT first 10 * from myreadtable;
-- INSERT a message into the table
INSERT into myreadtable values("IBM:81.98;Volume:1020");
-- SELECT the first message matching correlation id
SELECT FIRST 1 * from myreadtable where correlid = 'abc123';
```

IDS is aware of the correlation ID predicate and sends the correlation ID request to MQ. WebSphere MQ matches the correlation ID and sends the matched message.

You can create a table to transport BLOB data by using the statement shown in Example 10-37.

Example 10-37 Creating a table to transport BLOB data

```
execute function MQCreateVTIRECEIVE("mydoctable", "myservice",
"mypolicy", -2);
```

The table created by the MQCreateVTIRECEIVE() function is shown in Example 10-38.

Example 10-38 MQCreateVTIRECEIVE() created table

```
create table mydoctable
  ( msg      BLOB,
    correlid varchar(24),
    topic    varchar(40),
    qname    varchar(48),
    msgid    varchar(12),
    msgformat varchar(8)
  using "informix".mq (SERVICE = "myservice", POLICY = "mypolicy",
    ACCESS = "RECEIVE");

INSERT into mydoctable(msg) select blobcol from ordertab;

-- insert using blob, get through blob
INSERT into mydoctable(msg) values(filetoblob("/etc/passwd",
"client"));

select lotofile(msg, '/tmp/blob.dat','client') from mydoctable;
```

How not to use this feature

Both the INSERT and SELECT operations on the RECEIVE tables and INSERT on the READ tables change the data on MQ. You should be aware of these nuances, which are shown in the statements in Example 10-39.

Example 10-39 SELECT on RECEIVE tables

```
-- Find all the golf orders in the message queue.
SELECT * from myrecvorder where msg matches '%golf%';
-- Find all the orders from zip 94501.
SELECT * from myrecvorder where msg[12,15] = '94501';
-- Find all the messages greater than a correlation ID
SELECT * from myrecvorder where correlid > "abc123";
-- Find the number of messages for my correlation id
SELECT count(*) from myrecvorder;
```

To complete the two first SELECT statements, IDS retrieves *all* the messages in the service myrecvorder and then applies the filters and returns the qualified rows. The unqualified messages are lost. Use the READ tables if you want to apply these predicates, but be aware of the message fetch overhead.

connection to WebSphere MQ. When the application invokes the first WebSphere MQ function within a transaction, IDS begins a corresponding transaction at MQ. During commit or rollback, the IDS transaction manager is aware of WebSphere MQ participation in the transaction and coordinates the transaction with it.

Environment

MQ functionality is provided with IDS, and the DataBlade is installed into \$INFORMIXDIR/extend when IDS is installed. The DataBlade is registered in the database that is to invoke MQ functions. WebSphere MQ interaction is currently supported in Informix logged databases. IDS communicates with WebSphere MQ by using the server API. Therefore, WebSphere MQ must be installed on the same machine as the server. However, this WebSphere MQ can channel the messages to one or more remote WebSphere MQ servers. Each IDS instance can connect to only one WebSphere MQ queue manager.

Platform support

Table 10-7 summarizes the IDS and WebSphere MQ versions by supported platforms.

Table 10-7 IDS and WebSphere MQ platforms support

IDS version	Support platforms	WebSphere MQ version
10.00.xC3 and later	Solaris, 32 bit HP/UX (PA-RISC), 32 bit AIX, 32 bit Windows, 32 bit	Needs V5.3 and later
10.00.xC4 and later	AIX, 64 bit HP/UX (PA-RISC), 64 bit	Needs v6.0 and later
10.00.xC5 and later	Linux (Intel®), 32 bit Linux (IBM eServer™ pSeries®), 64 bit Solaris, 64 bit	Needs v6.0 and later

10.4 Relational access to flat files

So far we have looked at two VTI implementations, both of which were read only and dealt with sophisticated data stores such as WebSphere MQ and Amazon ECS. However, there are several other popular data sources that can populate flat files. Can IDS “relational-ize” flat file data?

The answer to this question takes us back to the basic objective of the UDAM framework, which is to provide an interface to integrate non-relational data into

IDS. By using the UDAM framework, it is easy to add relational query support to flat files. Interestingly enough, there is already a Bladelet, called *ffvti*, which does exactly that.

Bladelet: A Bladelet is a small, informal DataBlade module. It is meant to be ready to use and is offered complete with source code at no cost, but without support or warranty.

The *ffvti* Bladelet was written by one of the authors of this book, Jacques Roy. It provides *FFAccess* as the primary access method, which is a read-only interface to make external files look like relational tables to IDS. We extend this access method to make it read/write by adding capabilities to *INSERT* and *TRUNCATE* rows from the *VTI* table or underlying flat file.

10.4.1 The *ffvti* architecture

The *ffvti* architecture (shown in Figure 10-14) is similar to the standard UDAM-based designs. It provides *FFAccess* to UDAM, which understands how to read delimited records from a flat file and convert it into relational rows. The path of the flat file to relational and the delimiter character can be configured at *CREATE TABLE* time by using access method options in the *USING* clause.

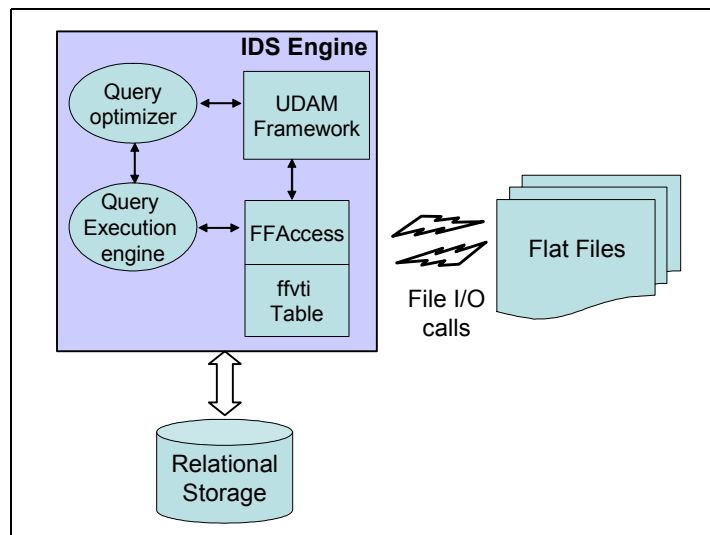


Figure 10-14 The *ffvti* architecture

Example 10-40 shows the CREATE ACCESS METHOD statement for the new fvti implementation. It has been improved with the addition of the ff_insert and ff_truncate purpose functions. The am_readwrite flag is enabled to indicate that the access method supports read and write. The am_delete and am_update functions are not defined. Therefore, DELETE and UPDATE of rows for fvti tables is not supported.

Example 10-40 CREATE ACCESS METHOD

```
CREATE PRIMARY ACCESS_METHOD FFAccess(
am_open = ff_open,
am_insert = ff_insert,
am_close = ff_close,
am_drop = ff_drop,
am_beginscan = ff_beginscan,
am_endscan = ff_endscan,
am_getnext = ff_getnext,
am_truncate = ff_truncate,
am_readwrite,
am_sptype = "A");
```

We now look at each of the purpose functions and its associated tasks, which are summarized in Table 10-8.

Table 10-8 Purpose functions and tasks

Purpose function	Task description
ffvti_open	<ul style="list-style-type: none"> ▶ Extracts the flat file name and delim character from options. ▶ Opens flat file access in read, write, or read-write mode based on the type of access on the VTI table. ▶ Allocates state information to maintain the flat file descriptor. ▶ Caches state information in the MI_AM_TABLE_DESC descriptor.
ffvti_close	<ul style="list-style-type: none"> ▶ Closes the flat file descriptor. ▶ Frees the state information.
ffvti_beginscan	<ul style="list-style-type: none"> ▶ Sets up the file seek position at the start of the file in preparation for file scan.
ffvti_endscan	<ul style="list-style-type: none"> ▶ Empty function because ffvti_begin handles the file rewind.

Purpose function	Task description
ffvti_getnext	<ul style="list-style-type: none"> ▶ Performs a buffered read on file. That is, performs read aheads of lines in a file to improve performance. ▶ Breaks up a line into delimited values and converts each value to the appropriate VTI table column type. ▶ Creates a row out of the column values by using <code>mi_row_create</code>. ▶ Applies any filters if necessary to decide whether the row must be returned to the engine. Uses <code>mi_eval_am_qual()</code>.
ffvti_insert	<ul style="list-style-type: none"> ▶ Extracts column values from the row provided by the IDS SQL engine. ▶ Converts the column values into character string by applying casting functions. ▶ Assembles buffer containing delimited characterized column values. ▶ Appends the newly formed buffer to the end of a flat file.
ffvti_truncate	<ul style="list-style-type: none"> ▶ Closes the file descriptor opened by <code>ffvti_open</code>. ▶ Reopens the file with the <code>O_TRUNC</code> option in order to truncate the file.
ffvti_drop	<ul style="list-style-type: none"> ▶ Empty function, because dropping the VTI table should not drop the flat file.

Transaction support

Transactional support for `ffvti` is not available in the current implementation. This means that the `INSERT` and `TRUNCATE TABLE` options are irreversible. Transactional support can be included by adding another layer in between the flat file and access method. However, that is beyond the scope of this chapter and does not add any value to the common usage scenarios of `ffvti`.

10.4.2 Testing `ffvti`

Having looked at the architecture of `ffvti`, we now test the implementation, by using the following steps, to exploit the powerful features provided:

1. Download the `ffvti` source and extract it to any location on your file system. We refer to this location as `FFVTI`. The examples provided with this book contain the files listed in Table 10-9 on page 459.

Table 10-9 *ffvti files*

File	Description
ffvti.c	Source code for the flat-file access method.
ffvti.h	Include file for ffvti.c and ffutil.c.
ffutil.c	Source code for utility functions for the flat-file access method.
ffvti.sql	SQL script used to create the access method.
Makefile	Makefile for the Bladelet.
tab.txt	Sample text file.
tryit.sql	Test script.
ffvti.def	Exported names for an NT DLL.
WinNT.mak	NT makefile used to create the DLL (requires Visual C++®).

2. Run the MAKE command in \${FFVTI} to generate the \${FFVTI}/bin/ffvti.bld shared object.
3. Run as user informix and create directory \$INFORMIXDIR/extend/ffvti.
4. Run as user informix and copy ffvti.bld to \$INFORMIXDIR/extend/ffvti.
5. Create a new database to work with, as shown in Example 10-41.

Example 10-41 Example create database

```
bash-3.00$ dbaccess - -
> create database tryffvti with log;
```

Database created.

6. Register the FFAccess method in the newly created database, as shown in Example 10-42.

Example 10-42 Registering the access method

```
bash-3.00$ dbaccess -e tryffvti ffvti.sql
```

Database selected.

```
CREATE FUNCTION set_tracing(lvarchar, integer, lvarchar)
RETURNING int
WITH(HANDLENULLS)
EXTERNAL NAME "$INFORMIXDIR/extend/ffvti/ffvti.bld(set_tracing)"
```

```

LANGUAGE C;
Routine created.
;
CREATE FUNCTION ff_open(ptr pointer)
RETURNING int
EXTERNAL NAME "$INFORMIXDIR/extend/ffvti/ffvti.bld(ff_open)"
LANGUAGE C;
Routine created.
;
CREATE FUNCTION ff_beginscan(ptr pointer)
RETURNING int
EXTERNAL NAME "$INFORMIXDIR/extend/ffvti/ffvti.bld(ff_beginscan)"
LANGUAGE C;
Routine created.
;
CREATE FUNCTION ff_getnext(ptr pointer, ptr2 pointer, OUT rowid int)
RETURNING int
EXTERNAL NAME "$INFORMIXDIR/extend/ffvti/ffvti.bld(ff_getnext)"
LANGUAGE C;
Routine created.
;
CREATE FUNCTION ff_endscan(ptr pointer)
RETURNING int
EXTERNAL NAME "$INFORMIXDIR/extend/ffvti/ffvti.bld(ff_endscan)"
LANGUAGE C;
Routine created.
;
CREATE FUNCTION ff_close(ptr pointer)
RETURNING int
EXTERNAL NAME "$INFORMIXDIR/extend/ffvti/ffvti.bld(ff_close)"
LANGUAGE C;
Routine created.
;
CREATE FUNCTION ff_drop(ptr pointer)
RETURNING int
EXTERNAL NAME "$INFORMIXDIR/extend/ffvti/ffvti.bld(ff_drop)"
LANGUAGE C;
Routine created.
;
CREATE FUNCTION ff_insert(ptr pointer, ptr2 pointer, OUT rowid int)
RETURNING int
EXTERNAL NAME "$INFORMIXDIR/extend/ffvti/ffvti.bld(ff_insert)"
LANGUAGE C;
Routine created.
;

```

```

CREATE FUNCTION ff_truncate(ptr pointer)
RETURNING int
EXTERNAL NAME "$INFORMIXDIR/extend/ffvti/ffvti.bld(ff_truncate)"
LANGUAGE C;
Routine created.
;
INSERT INTO systpaceclasses(name) VALUES("ffvti");
1 row(s) inserted.

```

```

CREATE PRIMARY ACCESS_METHOD FFAccess(
am_open = ff_open,
am_insert = ff_insert,
am_close = ff_close,
am_drop = ff_drop,
am_beginscan = ff_beginscan,
am_endscan = ff_endscan,
am_getnext = ff_getnext,
am_truncate = ff_truncate,
am_readwrite,
am_sptype = "A");
Access_method created.

```

Database closed.

7. Copy \${FFVTI}/tab.txt to /tmp/tab.txt. We use /tmp/tab.txt as our flat file.
8. Run \${FFVTI}/tryit.sql in the new database, as shown in Example 10-43.

Example 10-43 Trying the example

```
bash-3.00$ dbaccess -e tryffvti tryit.sql
```

Database selected.

```
EXECUTE FUNCTION set_tracing("ffvti", 80, "/tmp/trace.out");
```

```
(expression)
          0
1 row(s) retrieved.
```

```

CREATE TABLE tab
(
  a BOOLEAN,
  b VARCHAR(20),
  c DATE,
  d DATETIME year to second,

```

```
e INTERVAL hour to second,  
f DECIMAL,  
g DOUBLE PRECISION,  
h SMALLFLOAT,  
i INT8,  
j INT  
) USING FFAccess (path='/tmp/tab.txt', delim=';');  
Table created.
```

```
SELECT FIRST 3 * FROM tab;
```

```
a t  
b line 1  
c 04/17/2001  
d 2001-04-17 10:34:20  
e 10:34:20  
f 17.23000000000000  
g 3.141592600000  
h 2.180000070000  
i 123456789012  
j 1
```

```
a t  
b line 2  
c 04/18/2001  
d 2001-04-18 10:34:20  
e 10:34:20  
f 18.23000000000000  
g 4.141592600000  
h 3.180000070000  
i 223456789012  
j 2
```

```
a t  
b line 3  
c  
d  
e 10:34:20  
f 18.23000000000000  
g 4.141592600000  
h 3.180000070000  
i 223456789012  
j 3
```

```
3 row(s) retrieved.
```



```
insert into tab values ('t','new row', "01/01/2008", "2008-01-01
00:00:00", "00:00:00", 12.6, 12.6,1.0,123456789, 100);
1 row(s) inserted.
```

```
SELECT * FROM tab;
```

```
a t
b line 1
c 04/17/2001
d 2001-04-17 10:34:20
e 10:34:20
f 17.230000000000000
g 3.141592600000
h 2.180000070000
i 123456789012
j 1
```

```
a t
b line 2
c 04/18/2001
d 2001-04-18 10:34:20
e 10:34:20
f 18.230000000000000
g 4.141592600000
h 3.180000070000
i 223456789012
j 2
```

```
a t
b line 3
c
d
e 10:34:20
f 18.230000000000000
g 4.141592600000
h 3.180000070000
i 223456789012
j 3
```

```
a t
b new row
c 01/01/2008
d 2008-01-01 00:00:00
e 0:00:00
```

```
f 12.600000000000000
g 12.600000000000000
h 1.000000000000000
i 123456789
j 100
```

4 row(s) retrieved.

```
truncate table tab;
Table truncated.
```

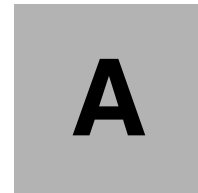
```
SELECT * FROM tab;
```

No rows found.

```
DROP TABLE tab;
Table dropped.
```

```
Database closed.
```

Example 10-43 first creates a VTI table called *tab*. Note the use of `USING FFAccess (path='/tmp/tab.txt', delim=';')`, which sets up the VTI table to use a flat file `/tmp/tab.txt` and delimiter `;`. Then we show a `SELECT FIRST 3` on the VTI to sample the first three rows in the flat file. Next, we `INSERT` a row into the VTI table, followed by a `SELECT`, to illustrate the success of the previous `INSERT`. Finally we `TRUNCATE` table, followed by a `SELECT`, to show that the flat file indeed gets truncated.



Additional material

This book refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

<ftp://www.redbooks.ibm.com/redbooks/SG247522>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247522.

Using the Web material

The additional Web material that accompanies this book includes the following files:

<i>File name</i>	<i>Description</i>
TestVTI.zip	Compressed code sample for Amazon Web Service VTI and Flat File Access VTI

Fraction Example.zip Compressed file that contains the following “fraction” files:

- **fraction.1.0.zip**: Fraction DataBlade for Windows, including built copy ready to install. Refer to 6.2.3, “Fractions” on page 233.
- **fraction.1.0.tar.gz**: Fraction DataBlade for Linux and UNIX, including built copy, ready to install, for Linux only. Refer to 6.2.3, “Fractions” on page 233.
- **FractionTestScenario.zip**: SQL scripts and command files for running the property tax scenario described in “Parcel ownership: A worked-out example” on page 235.

System requirements for downloading the Web material

The following system requirements are for downloading the Web material for two examples provided with this publication.

The Amazon VTI example

The following system configuration is recommended. It includes the minimum configuration required to install and run a single instance of the Informix Dynamic Server (IDS):

Hard disk space:	300 MB
Operating System:	Linux 32 bit (Red Hat Enterprise Linux (RHEL) 4.0)
Processor:	Pentium® 4 or later
Memory:	512 MB or higher

In addition to the above configuration, gSOAP is required to run the Amazon Web Service VTI. To use gSOAP, first download a recent version of gSOAP for your desired development platform (UNIX, Linux, or Windows) from the following Web address:

http://sourceforge.net/project/showfiles.php?group_id=52781

Throughout the development of this book, we have used version 2.7.9l of gSOAP for Linux x86 32 bit.

After downloading the gSOAP toolkit, extract the compressed file into a folder, for example /work/gsoap-linux-2.7. In the following sections, we refer to this gSOAP install location as the `#{GSOAP_DIR}`.

Since you need to compile C source code files, ensure that a C-compiler is installed on your development platform. For the examples in this section, we have been using the GCC version 3.4.6 20060404 (Red Hat 3.4.6-3).

The Fraction DataBlade example

The requirements for using the Fraction DataBlade and test scenario downloads depend on what you want to do with them. Consider the following points:

- ▶ To inspect source code and scripts (all downloads), you need any text or code editor.
- ▶ To install and run the Fraction DataBlade on Windows (fraction.1.0.zip and FractionTestScenario.zip), you need a working instance of IDS (9.x, 10.x, or 11.x) on one of the supported Windows platforms. The included installation was tested on IDS 11.10.TC2 running on Windows XP Professional.
- ▶ To install and run the Fraction DataBlade on Linux (fraction.1.0.tar.gz and FractionTestScenario.zip), you need a working instance of IDS (9.x, 10.x, or 11.x) on one of the supported Linux platforms. The included installation was tested on IDS 11.50.UCB1TL running on Ubuntu Linux 2.6.22-virtual.
- ▶ To build or rebuild the Fraction DataBlade on Windows (fraction.1.0.zip):
 - Microsoft Visual C++ .NET Version 7, Microsoft Development Environment 2003 or later
 - An installed copy of IDS 9.x, 10.x, or 11.x (for C header files)
 - (Optional) To create the distribution directory automatically, BladePack 4.20.TC1 or later, from the IBM Informix DataBlade Developers Kit
- ▶ To build or rebuild the Fraction DataBlade on Linux or UNIX (fraction.1.0.tar.gz):
 - Development tools including C compiler, linker, and standard libraries
 - An installed copy of IDS 9.x, 10.x, or 11.x (for C header and make include files)

How to use the Web material

To use the Web material, create a subdirectory (folder) on your workstation and extract the contents of the Web material zip file into that folder.

The Amazon VTI example

Extracting the TestVTI.zip file creates two new subdirectories (folders) called wsvti and ffvti. We refer to the path of this wsvti and ffvti subdirectory as `${WSVTI}` and `${FFVTI}` respectively.

Use the following steps to test the Amazon VTI source example:

1. Edit `wsvti.h` and change the macro `AWSACCESSKEYID` to use your Amazon access key.
2. Run the `MAKE` command in the `${WSVTI}` directory to generate the `IDSAmazonAccess UDAM` shared object. The shared object name is `wsvti.bld` and can be found at `${WSVTI}/linux-intel/`.
3. Running as user `informix`, create the `$INFORMIXDIR/extend/wsvti` directory.
4. Running as user `informix`, copy `${WSVTI}/linux-intel/wsvti.bld` to `$INFORMIXDIR/extend/wsvti`. This step ensures that the `IDSAmazonAccess UDAM` is copied to where all IDS extensions reside.
5. Configure IDS to run the `IDSAmazonAccess UDAM` code in a separate virtual processor, called `wsvp`. By adding a dedicated virtual processor class to the IDS configuration, you can separate the execution of the blocking network calls of the Web service consumer UDAM from the overall IDS query processing. To enable at least one dedicated virtual processor class for that purpose, add the following line to the `ONCONFIG` file of your IDS instance and restart the IDS instance:

```
VPCLASS wsvp,num=1
```

6. Create a new database to work with as shown in Example A-1.

Example: A-1 New database for use with example

```
bash-3.00$ dbaccess - -  
> create database tryamazon with log;
```

Database created.

7. Register UDAM in the `tryamazon` database using script `wsvti.sql`, as shown in Example A-2

Example: A-2 Sample script for testing

```
bash-3.00$ dbaccess tryamazon wsvti.sql
```

8. Run the test script provided in the `${WSVTI}/test` directory. The test script creates two relational tables, `customer` and `orders`. It then populates data into

those tables. Finally it creates the Amazon VTI table, called AmazonTable (shown in Example A-3) and performs a join between all three tables.

Example: A-3 Example table create

```
CREATE TABLE AmazonTable
(
  ISBN char(10),
  TITLE lvarchar,
  PRICE decimal
) USING IDSAmaonAccess ;
```

Example A-4 shows the results of executing the test script.

Example: A-4 Script results

```
bash-3.00$ dbaccess -e tryamazon tryit.sql
```

Database selected.

```
create table customer
(
  customer_num          serial(101),
  fname                 char(15),
  lname                 char(15),
  company               char(20),
  address1              char(20),
  address2              char(20),
  city                  char(15),
  state                 char(2),
  zipcode               char(5),
  phone                 char(18),
  primary key (customer_num)
);
```

Table created.

```
create table orders
(
  order_num             serial(1001),
  order_date            date,
  customer_num          integer not null,
  ship_instruct         char(40),
  backlog               char(1),
  ISBN                  char(10),
  ship_date             date,
  primary key (order_num),
```

```
        foreign key (customer_num) references customer (customer_num)
    );
Table created.
```

```
load from 'customer.unl' insert into customer;
28 row(s) loaded.
```

```
load from 'orders.unl' insert into orders;
4 row(s) loaded.
```

```
CREATE TABLE AmazonTable
(
  ISBN char(10),
  TITLE lvarchar,
  PRICE decimal
) USING IDSAmericanAccess ;
Table created.
```

```
SELECT FIRST 5 * from AmazonTable;
```

```
isbn  1401908810
title Spiritual Connections: How to Find Spirituality Throughout All
the Relationships in Your Life
price 24.95000000000000

isbn  0743292855
title Paula Deen: It Ain't All About the Cookin'
price 25.00000000000000

isbn  1591391105
title The First 90 Days: Critical Success Strategies for New Leaders
at All Levels
price 27.95000000000000

isbn  0072257121
title CISSP All-in-One Exam Guide, Third Edition (All-in-One)
price 79.99000000000000

isbn  0071410155
title The Six Sigma Handbook: The Complete Guide for Greenbelts,
Blackbelts, and Managers at All Levels, Revised and Expanded Edition
price 89.95000000000000
```


5 row(s) retrieved.

```
select --+USE_NL(AmazonTable)
order_num,
AmazonTable.*, fname, lname, order_date from AmazonTable, orders,
customer where (AmazonTable.isbn = orders.isbn) and orders.customer_num
= customer.customer_num;
```

```
order_num 1001
isbn      0091883768
title     Who Moved My Cheese?
price     20.6500000000000
fname     Anthony
lname     Higgins
order_date 05/20/1998
```

```
order_num 1002
isbn      0091883768
title     Who Moved My Cheese?
price     20.6500000000000
fname     Ludwig
lname     Pauli
order_date 05/21/1998
```

```
order_num 1003
isbn      0954681320
title     Six Sigma and Minitab: A complete toolbox guide for all Six
Sigma p
          ractitioners (2nd edition)
price     49.9900000000000
fname     Anthony
lname     Higgins
order_date 05/22/1998
```

```
order_num 1004
isbn      0764525557
title     Gardening All-in-One for Dummies
price     29.9900000000000
fname     George
lname     Watson
order_date 05/22/1998
```

4 row(s) retrieved.

```
DROP TABLE AmazonTable;  
Table dropped.
```

```
DROP TABLE customer;  
Table dropped.
```

```
DROP TABLE orders;  
Table dropped.
```

```
Database closed.
```

You have successfully installed and used the Amazon Web Service VTI sample code.

Running the Flat File Access VTI example

To run the flat file access VTI example:

1. Run the MAKE command in the `${FFVTI}` directory to generate the FFVTI UDAM shared object. The shared object name is `ffvti.bld` and can be found in `${FFVTI}/bin/linux-intel/`.
2. Run as user `informix` and create the `$INFORMIXDIR/extend/ffvti` directory.
3. Run as user `informix` and copy `ffvti.bld` to `$INFORMIXDIR/extend/ffvti`.
4. Create a new database to work with, as shown in Example A-5.

Example: A-5 Example create database

```
bash-3.00$ dbaccess - -  
> create database tryffvti with log;
```

5. Register the FFAccess method in the newly created database, as shown in Example A-6.

Example: A-6 Register the access method

```
bash-3.00$ dbaccess -e tryffvti ffvti.sql
```

6. Copy the `${FFVTI}/tab.txt` file to `/tmp/tab.txt`. We used the `/tmp/tab.txt` file as our flat file.

7. Run `#{FFVTI}/tryit.sql` in the new database, as shown in Example A-7.

Example: A-7 Try the example

```
bash-3.00$ dbaccess -e tryffvti tryit.sql
```

```
Database selected.
```

```
EXECUTE FUNCTION set_tracing("ffvti", 80, "/tmp/trace.out");
```

```
(expression)
```

```
0
```

```
1 row(s) retrieved.
```

```
CREATE TABLE tab
```

```
(
```

```
  a BOOLEAN,
```

```
  b VARCHAR(20),
```

```
  c DATE,
```

```
  d DATETIME year to second,
```

```
  e INTERVAL hour to second,
```

```
  f DECIMAL,
```

```
  g DOUBLE PRECISION,
```

```
  h SMALLFLOAT,
```

```
  i INT8,
```

```
  j INT
```

```
) USING FFAccess (path='/tmp/tab.txt', delim=';');
```

```
Table created.
```

```
SELECT FIRST 3 * FROM tab;
```

```
a t
```

```
b line 1
```

```
c 04/17/2001
```

```
d 2001-04-17 10:34:20
```

```
e 10:34:20
```

```
f 17.23000000000000
```

```
g 3.141592600000
```

```
h 2.180000070000
```

```
i 123456789012
```

```
j 1
```

```
a t
```

```
b line 2
c 04/18/2001
d 2001-04-18 10:34:20
e 10:34:20
f 18.23000000000000
g 4.141592600000
h 3.180000070000
i 223456789012
j 2
```

```
a t
b line 3
c
d
e 10:34:20
f 18.23000000000000
g 4.141592600000
h 3.180000070000
i 223456789012
j 3
```

3 row(s) retrieved.

```
insert into tab values ('t',"new row", "01/01/2008", "2008-01-01
00:00:00", "00:00:00", 12.6, 12.6,1.0,123456789, 100);
1 row(s) inserted.
```

```
SELECT * FROM tab;
```

```
a t
b line 1
c 04/17/2001
d 2001-04-17 10:34:20
e 10:34:20
f 17.23000000000000
g 3.141592600000
h 2.180000070000
i 123456789012
j 1
```

```
a t
b line 2
```

```
c 04/18/2001
d 2001-04-18 10:34:20
e 10:34:20
f 18.23000000000000
g 4.141592600000
h 3.180000070000
i 223456789012
j 2
```

```
a t
b line 3
c
d
e 10:34:20
f 18.23000000000000
g 4.141592600000
h 3.180000070000
i 223456789012
j 3
```

```
a t
b new row
c 01/01/2008
d 2008-01-01 00:00:00
e 0:00:00
f 12.60000000000000
g 12.600000000000
h 1.00000000000000
i 123456789
j 100
```

4 row(s) retrieved.

```
truncate table tab;
Table truncated.
```

```
SELECT * FROM tab;
```

No rows found.

```
DROP TABLE tab;
Table dropped.
```

```
Database closed.
```

You have now successfully installed and used the Flat File Access VTI sample code.

The Fraction DataBlade example

The downloads for this example include a complete DataBlade called *Fraction* (version 1.0) to illustrate the ideas described in 6.2.3, “Fractions” on page 233 and show the implementation details of an opaque data type. The DataBlade, when built, is fully usable for production work.

fraction.1.0.zip: Fraction DataBlade for Windows

The compressed file contains the following folder structure:

- ▶ distrib
 - extend
 - fraction.1.0

This is the distribution staging folder, which contains the complete DataBlade distribution package. To install, simply copy this folder to %INFORMIXDIR%\extend\fraction.1.0.
- ▶ install

This folder contains the BladePack control files that allow you to use BladePack to copy the right scripts and DataBlade dynamic link library (DLL) to the distribution folder. While the use of BladePack is optional, you can easily copy the scripts manually.
- ▶ scripts

This folder contains the SQL registration scripts that BladeManager requires to register the DataBlade. These scripts are to be copied to the distribution folder. In addition, it contains a script, test.sql, that creates all the DataBlade objects without going through BladeManager. It has no provision for unregistering or upgrading. Therefore, only use this script for testing purposes.
- ▶ src
 - c

This folder contains the readme.txt file that describes the source files (in the c subfolder) and provides instructions for building the DataBlade. In addition, it contains the project and solution files for Microsoft Visual Studio/C++.
 - c

This folder contains the C source and header files, as well as the .def file that defines the DLL exports.

- winnt-i386

This folder contains the .bld DataBlade DLL file that is built by Visual C++.

fraction.1.0.tar.gz: Fraction DataBlade for Linux and UNIX

The gnu-zip-compressed tar file contains the following directory structure:

- ▶ distrib

- fraction.1.0

This is the distribution staging directory, which contains the complete DataBlade distribution package. To install, simply copy this directory to \$INFORMIXDIR/extend/fraction.1.0.

- ▶ scripts

This folder contains the SQL registration scripts that BladeManager requires to register the DataBlade. These are to be copied to the distribution directory. In addition, it contains a script, test.sql, that creates all the DataBlade objects without going through BladeManager. It has no provision for unregistering or upgrading. Therefore, only use this script for testing purposes.

- ▶ src

This folder contains the readme.txt file that describes the source files (in the c subdirectory) and provides instructions for building the DataBlade. In addition, it contains the UNIX or Linux makefile (and the output from a MAKE run).

- c

This folder contains the C source and header files.

- linux-intel

This folder contains the .bld DataBlade shared library created by MAKE, as well as the object files from the compilation step.

FractionTestScenario.zip: Parcel tax example

This download lets you recreate the example discussed in “Parcel ownership: A worked-out example” on page 235. The results and tables presented there are formatted and edited results from the queries in this package; an additional step is performed in this package that is not represented in this book.

The FractionTestScenario.zip file contains the following files:

- ▶ scenario.cmd

This is the Windows command file to run the entire scenario and save the output in scenario.log. For usage instructions, type the following command:

```
scenario
```

- ▶ scenario.sh:
This file contains the UNIX or Linux shell script to run the entire scenario and save the output in scenario.log. For usage instructions, type the following command:
`./scenario`
- ▶ tax.sql:
This file contains the SQL script to create the tables used in the parcel tax scenario, as well as stored procedures for performing the steps in the scenario.
- ▶ taxtotals.sql
This file has the SQL script that contains the two queries for showing ownership per owner and per parcel.

TSndx datatype and overloads

In this section we provide the the full listing of all SQL and C code for implementation and addition of the overloads.

SQL code sample

(c) Copyright IBM Corp. 2002 All rights reserved.

This sample program is owned by International Business Machines Corporation or one of its subsidiaries ("IBM") and is copyrighted and licensed, not sold.

You may copy, modify, and distribute this sample program in any form without payment to IBM, for any purpose including developing, using, marketing or distributing programs that include or are derivative works of the sample program.

The sample program is provided to you on an "AS IS" basis, without warranty of any kind. IBM HEREBY EXPRESSLY DISCLAIMS ALL WARRANTIES EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow for the exclusion or limitation of implied warranties, so the above limitations or exclusions may not apply to you. IBM shall not be liable for any damages you suffer as a result of using, modifying or distributing the sample program or its derivatives.

Each copy of any portion of this sample program or any derivative work, must include a the above copyright notice and disclaimer of warranty.

The following are SQL commands used the the soundex DataBlade example. You will not just be able to run this file directly. Rather, use something like SQLEditor to selectively execute them.

Author: Jon Machtynger (jon.machtynger@uk.ibm.com)

```
Create type for TSndx:  
create opaque type TSndx (  
    internallength = 286,  
    alignment = 8  
);
```

```
Define the Input Function for TSndxInput:  
drop function TSndxIn (lvarchar);  
create function TSndxIn (lvarchar)  
returns TSndx  
external name '$INFORMIXDIR/extend/Sndx/Soundex.blc(TSndxInput)'  
language c;
```

```
Define the Output Function for TSndxOutput:  
drop function TSndxOut (TSndx);  
create function TSndxOut (TSndx)  
returns lvarchar  
external name '$INFORMIXDIR/extend/Sndx/Soundex.blc(TSndxOutput)'  
language c;
```

```
Provide Compare Function:  
drop function Compare (TSndx,TSndx);  
create function Compare (TSndx,TSndx)  
returns integer  
with (not variant)  
external name '$INFORMIXDIR/extend/Sndx/Soundex.blc(TSndxCompare)'  
language c;
```

```
Provide Equal Function:  
drop function Equal (TSndx,TSndx);  
create function Equal (TSndx,TSndx)  
returns boolean  
with (not variant)  
external name '$INFORMIXDIR/extend/Sndx/Soundex.blc(TSndxEqual)'  
language c;
```

```
Provide NotEqual Function:  
drop function NotEqual (TSndx,TSndx);  
create function NotEqual (TSndx,TSndx)  
returns boolean
```

```
with (not variant)
external name '$INFORMIXDIR/extend/Sndx/Soundex.bld(TSndxNotEqual)'
language c;
```

```
Provide LessThan Function:
drop function LessThan (TSndx,TSndx);
create function LessThan (TSndx,TSndx)
returns boolean
with (not variant)
external name '$INFORMIXDIR/extend/Sndx/Soundex.bld(TSndxLessThan)'
language c;
```

```
Provide GreaterThan Function:
drop function GreaterThan (TSndx,TSndx);
create function GreaterThan (TSndx,TSndx)
returns boolean
with (not variant)
external name '$INFORMIXDIR/extend/Sndx/Soundex.bld(TSndxGreaterThan)'
language c;
```

```
Provide LessThanOrEqual Function:
drop function LessThanOrEqual (TSndx,TSndx);
create function LessThanOrEqual (TSndx,TSndx)
returns boolean
with (not variant)
external name
'$INFORMIXDIR/extend/Sndx/Soundex.bld(TSndxLessThanOrEqual)' language
c;
```

```
Provide GreaterThanOrEqual Function:
drop function GreaterThanOrEqual (TSndx,TSndx);
create function GreaterThanOrEqual (TSndx,TSndx)
returns boolean
with (not variant)
external name
'$INFORMIXDIR/extend/Sndx/Soundex.bld(TSndxGreaterThanOrEqual)'
language c;
```

```
drop function Like (TSndx, lvarchar);
create function Like (TSndx, lvarchar)
returning boolean
with (not variant)
external name '$INFORMIXDIR/extend/Sndx/Soundex.bld(TSndxRE)' language
c;
```

```
Function to return internal value of sound:
drop function TSndxValue (TSndx);
create function TSndxValue (TSndx)
returning integer
with (not variant)
external name '$INFORMIXDIR/extend/Sndx/Soundex.blc(TSndxValue)'
language c;
```

C code sample

(c) Copyright IBM Corp. 2002 All rights reserved.

This sample program is owned by International Business Machines Corporation or one of its subsidiaries ("IBM") and is copyrighted and licensed, not sold. You may copy, modify, and distribute this sample program in any form without payment to IBM, for any purpose including developing, using, marketing or distributing programs that include or are derivative works of the sample program.

The sample program is provided to you on an "AS IS" basis, without warranty of any kind. IBM HEREBY EXPRESSLY DISCLAIMS ALL WARRANTIES EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow for the exclusion or limitation of implied warranties, so the above limitations or exclusions may not apply to you. IBM shall not be liable for any damages you suffer as a result of using, modifying or distributing the sample program or its derivatives.

Each copy of any portion of this sample program or any derivative work, must include a the above copyright notice and disclaimer of warranty.

Example 10-44 contains all the C code necessary to create your Soundex DataBlade.

Author: Jon Machtynger (jon.machtynger@uk.ibm.com)

Example 10-44 C code sample

```
/* Standard library includes. */
#include <ctype.h>
#include <stdio.h>
#include <string.h>
#include <stddef.h>

/* Include when accessing the Informix API. */
#include <mi.h>

typedef struct {
    mi_char          data[256];
```

```

        mi_char          sound[30];
    } TSndx;

// Define Sound-bytes

#define SOUND_S      '0'
#define SOUND_S      '1'
#define SOUND_S      '2'
#define SOUND_S      '3'
#define SOUND_S      '4'
#define SOUND_S      '5'
#define SOUND_S      '6'
#define SOUND_S      '7'
#define SOUND_S      '8'
#define SOUND_S      '9'
#define SOUND_SKIP   0

#define RE_MULTI     '*' // Alternative could be '%'
#define RE_SINGLE    '?' // Alternative could be '_'

UDREXPOR TSndx *TSndxInput (mi_lvarchar *InValue)
{
    TSndx    *result;// Return Value
    mi_char  *textval;// Container for text value
    mi_char  *textsnd;// Container for Sound Value
    mi_char  *thesound;// Sound of the text

    mi_char  *Sndx(char *);

    // Convert externally passed string to a character string
    textval = mi_lvarchar_to_string(InValue);

    // Allocate mem for return value
    result = (TSndx *)mi_alloc(sizeof(TSndx));

    // Fill the structure values
    strcpy(result->data, textval);
    thesound = Sndx(textval);
    strcpy(result->sound, thesound);

    // Clean up

    mi_free(thesound);
    mi_free(textval);
}

```

```

    // Return data type to calling statement
    return result;
}

UDREXPOR mi_integer TSndxValue(TSndx *Value)
{
    // The following works around the fact that an initial '0'
    // value may be ignored. Multiple 0's will be missing
    // substitute a starting 0 with a decimal point.
    if (Value->sound[0] == '0') {
        Value->sound[0] = '-';
    }

    return atoi(Value->sound);
}

UDREXPOR mi_lvarchar *TSndxOutput(TSndx *Value)
{
    return mi_string_to_lvarchar(Value->data);
}

// The following are functions used in comparing one sound to another

// This function effectively does a string compare on the sounds. It
// is
// arguable whether this makes sense to compare sounds, but it does
// allow
// us to index the values in a B-Tree which has performance benefits.
UDREXPOR mi_integer TSndxCompare (TSndx *Value1, TSndx *Value2)
{
    mi_integerval;

    if (val = strcmp(Value1->sound, Value2->sound))
        return (val > 0) ? 1 : -1;
    return val;
}

// All following functions use TSndxCompare

UDREXPOR mi_boolean TSndxEqual (TSndx *Value1, TSndx *Value2)
{
    return (mi_boolean)(0 == TSndxCompare(Value1, Value2));
}

```

```

UDREXPOR mi_boolean TSndxNotEqual (TSndx *Value1, TSndx *Value2)
{
    return (mi_boolean)(0 != TSndxCompare(Value1, Value2));
}

UDREXPOR mi_boolean TSndxLessThan (TSndx *Value1, TSndx *Value2)
{
    return (mi_boolean)(-1 == TSndxCompare(Value1, Value2));
}

UDREXPOR mi_boolean TSndxLessThanOrEqual (TSndx *Value1, TSndx
*Value2)
{
    return (mi_boolean)(1 > TSndxCompare(Value1, Value2));
}

UDREXPOR mi_boolean TSndxGreaterThan (TSndx *Value1, TSndx *Value2)
{
    return (mi_boolean)(1 == TSndxCompare(Value1, Value2));
}

UDREXPOR mi_boolean TSndxGreaterThanOrEqual(TSndx *Value1, TSndx
*Value2)
{
    return (mi_boolean)(-1 < TSndxCompare(Value1, Value2));
}

// This is a consonant based soundex routine. Any soft letters
// are ignored unless they play a significant part in modifying
// a hard sound (e.g. 'h' ignored unless it makes a 's' a 'sh').

char *Sndx(char *txt)
{
    mi_integerlength, i, pcntr;
    mi_charccurr, cnext, cprev, cnext2, cnext3, cnext4, sound;
    mi_char*buf, *p, assign_no_prev_char(mi_char cprev, mi_char check,
mi_char sound);

    length=strlen(txt);

    buf = (char *)mi_alloc(length + 100);

    for (pcntr=0, cprev=0, i=0;i<length;i++)

```

```

{
    cprev = (i>0) ? tolower(txt[i-1]) : 0;
    ccurr=tolower(txt[i]);
    cnext = ((i+1) < length) ? tolower(txt[i+1]) : 0;
    cnext2 = ((i+2) < length) ? tolower(txt[i+2]) : 0;
    cnext3 = ((i+3) < length) ? tolower(txt[i+3]) : 0;
    cnext4 = ((i+4) < length) ? tolower(txt[i+4]) : 0;

    if (!isalpha(ccurr) &&
        (ccurr != RE_MULTI) && (ccurr != RE_SINGLE) &&
        (ccurr != '[') && (ccurr != ']') &&
        (ccurr != '^') && (ccurr != '!'))
        continue;

    switch(ccurr)
    {
        // Soft characters
        case 'a':
        case 'e':
        case 'h':
        case 'i':
        case 'o':
        case 'u':
        case 'w':
        case 'y':
            sound=SOUND_SKIP;
            break;

        // Non Compound consonents.

        case 'b':
            sound = assign_no_prev_char(cprev, ccurr, SOUND_B);
            break;

        case 'd':
            sound = assign_no_prev_char(cprev, ccurr, SOUND_T);
            break;

        case 'f':
        case 'v':
            sound = assign_no_prev_char(cprev, ccurr, SOUND_F);
            break;

        case 'j':
            sound = assign_no_prev_char(cprev, ccurr, SOUND_SH);

```

```

        break;

case 'l':
    sound = assign_no_prev_char(cprev, ccurr, SOUND_L);
    break;

case 'm':
    sound = assign_no_prev_char(cprev, ccurr, SOUND_M);
    break;

case 'n':
    sound = assign_no_prev_char(cprev, ccurr, SOUND_N);
    break;

case 'q':
    sound = assign_no_prev_char(cprev, ccurr, SOUND_K);
    break;

case 'r':
    sound = assign_no_prev_char(cprev, ccurr, SOUND_R);
    break;

case 'z':
    sound = assign_no_prev_char(cprev, ccurr, SOUND_S);
    break;

// The beginning of the compound characters

case 'c':
    if (cnext == 'e') {
        sound=SOUND_S;
        break;
    }

    if (cnext == 'h'){
        sound=SOUND_SH;
        break;
    }

    if (cnext == 'i') {
        if ((cnext2=='o') && (cnext3=='u') && (cnext4=='s')) {
            sound = SOUND_SH;
            i += 2;
            break;

```



```

        }
        sound=SOUND_S;
        break;
    }

    sound = assign_no_prev_char(cprev, 'c', SOUND_K);
    break;

case 'g':
    if (cnext=='e') {
        sound=SOUND_SH;
        break;
    }
    sound = assign_no_prev_char(cprev, 'g', SOUND_K);
    break;

case 'k':
    if (cprev=='c') {
        sound=SOUND_SKIP;
        break;
    }
    sound = assign_no_prev_char(cprev, 'k', SOUND_K);
    break;

case 'p':
    if (cnext=='h') {
        sound=SOUND_F;
        break;
    }
    sound = assign_no_prev_char(cprev, 'p', SOUND_B);
    break;

case 's':
    if (cnext == 'h') {
        sound=SOUND_SH;
        break;
    }
    sound = assign_no_prev_char(cprev, 's', SOUND_S);
    break;

case 't':

    if ((cnext=='i') && (cnext2=='o')) {
        sound = SOUND_SH;
        break;
    }

```

```

        }
        sound = assign_no_prev_char(cprev, 't', SOUND_T);
        break;

    case 'x':
        buf[pcntr]=SOUND_K;
        buf[pcntr+1]=SOUND_S;
        buf[pcntr+2]='\0';
        pcntr += 2;
        break;

    default:
        sound = ccurr;
    }

    if (sound != SOUND_SKIP) {
        buf[pcntr]=sound;
        buf[pcntr+1]='\0';
        pcntr++;
    }
}

return (buf);
}

// Where recurring characters are present, don't force a double
// sound e.g. (madder does not sound like maddadder)

mi_char assign_no_prev_char(mi_char cprev, mi_char check, mi_char
sound)
{
    if (cprev==check)
        return SOUND_SKIP;
    else
        return sound;
}

//
// Pattern Match routine.
//
// Simplistic Regular expression handler
//
// Allows use of:
//   '%' to match any string

```

```

//  '_' to match a single character
//  [] to provide a set of matchable characters
//  ^ and ! to negate the contents of matched characters
//
// The guts of this code has been taken from GNU make.
//

UDREXPOR mi_boolean TSndxRE(TSndx *p_string, mi_lvarchar *p_pattern)
{
    mi_booleanflag;
    mi_char *pattern;
    mi_char *patternsound;
    registerchar c;
    mi_booleanstrmatch (char *, char *);

    pattern = mi_lvarchar_to_string(p_pattern);
    if (pattern == NULL) {
        mi_db_error_raise( NULL, MI_SQL, "error",
            "FUNCTION %s", "TSndxRE 1", (char *)NULL);
    }

    patternsound = Sndx(pattern);

    flag = strmatch(patternsound, p_string->sound);

    printf("Matching %s against %s gives %d\n", patternsound,
p_string->sound, flag);

    mi_free(pattern);
    mi_free(patternsound);

    return flag;
}

// Actual strmatch code to allow regular expression matching
mi_boolean strmatch (char *pattern, char *string)
{
    register char*p = pattern,
        *n = string;
    register char c;

    while ((c = *p++) != '\0')
    {
        c = tolower(c);

```

```

switch (c)
{
    case RE_SINGLE:

        if (*n == '\\0')
            return MI_FALSE;
        break;

    case RE_MULTI:

        for (c = *p++; c == RE_SINGLE || c == RE_MULTI; c = *p++)
        {
            if (c == RE_SINGLE)
            {
                if (*n == '\\0')
                    return MI_FALSE;
                else
                    ++n;
            }
        }

        if (c == '\\0')
            return MI_TRUE;

        {
            char c1 = tolower(c);
            for (--p; *n != '\\0'; ++n)
                if ((c == '[' || tolower(*n) == c1) &&
                    strmatch (p, n) == MI_TRUE)
                    return MI_TRUE;
            return MI_FALSE;
        }

    case '[':
    {
        register int not;

        if (*n == '\\0')
            return MI_FALSE;

        not = (*p == '!' || *p == '^');
        if (not)
            ++p;

        c = *p++;
    }
}

```

```

for (;;)
{
    register char cstart = c, cend = c;

    cstart = cend = tolower(cstart);

    if (c == '\0')
        return MI_FALSE;

    c = tolower(*p++);

    if (c == '-' && *p != ']')
    {
        cend = *p++;
        if (cend == '\0')
            return MI_FALSE;
        cend = tolower(cend);

        c = *p++;
    }

    if (tolower(*n) >= cstart && tolower(*n) <= cend)
        goto matched;

    if (c == ']')
        break;
}

if (!not)
    return MI_FALSE;
break;

matched::
while (c != ']')
{
    if (c == '\0')
        return MI_FALSE;

    c = *p++;
}

if (not)
    return MI_FALSE;
}

```

```
        break;

        default:
            if (c != tolower(*n))
                return MI_FALSE;
    }
    ++n;
}

if (*n == '\\0')
    return MI_TRUE;

return MI_FALSE;
}
```

Glossary

access control list (ACL). The list of principals that have explicit permission (to publish, to subscribe to, and to request persistent delivery of a publication message) against a topic in the topic tree. The ACLs define the implementation of topic-based security.

aggregate. Precalculated and prestored summaries, kept in the data warehouse to improve query performance.

aggregation. An attribute-level transformation that reduces the level of detail of available data, for example, having a Total Quantity by category of items rather than the individual quantity of each item in the category.

application programming interface (API). An interface provided by a software product that enables programs to request services.

asynchronous messaging. A method of communication between programs in which a program places a message on a message queue and then proceeds with its own processing without waiting for a reply to its message.

attribute. A field in a dimension table.

BLOB. Binary large object. A block of bytes of data (for example, the body of a message) that has no discernible meaning, but is treated as one solid entity that cannot be interpreted.

commit. An operation that applies all the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work begins.

composite key. A key in a fact table that is the concatenation of the foreign keys in the dimension tables.

computer. A device that accepts information (in the form of digitalized data) and manipulates it for a result based on a program or sequence of instructions about how the data is to be processed.

configuration. The collection of brokers, their execution groups, the message flows and sets that are assigned to them, and the topics and associated access control specifications.

continuous data replication. See *enterprise replication*.

data append. A data loading technique where new data is added to the database leaving the existing data unaltered.

data cleansing. A process of data manipulation and transformation to eliminate variations and inconsistencies in data content. Data cleansing typically improves the quality, consistency, and usability of the data.

Data Definition Language (DDL). An SQL statement that creates or modifies the structure of a table or database, for example, CREATE TABLE, DROP TABLE, ALTER TABLE, or CREATE DATABASE.

data federation. The process of enabling data from multiple heterogeneous data sources to appear as though it is contained in a single relational database. Can also be referred to “distributed access.”

Data Manipulation Language (DML). An INSERT, UPDATE, DELETE, or SELECT SQL statement.

data mart. An implementation of a data warehouse, typically with a smaller and more tightly restricted scope, such as for a department or workgroup. It can be independent or derived from another data warehouse environment.

data mining. A mode of data analysis that has a focus on the discovery of new information, such as unknown facts, data relationships, or data patterns.

data partition. A segment of a database that can be accessed and operated on independently even though it is part of a larger data structure.

data refresh. A data loading technique where all the data in a database is completely replaced with a new set of data.

data warehouse. A specialized data environment developed, structured, and used specifically for decision support and informational applications. It is subject oriented rather than application oriented. Data is integrated, non-volatile, and time variant.

database partition. Part of a database that consists of its own data, indexes, configuration files, and transaction logs.

DataBlades. These are program modules that provide extended capabilities for Informix databases and are tightly integrated with the database management system (DBMS).

DB Connect. Enables connection to several relational database systems and the transfer of data from these database systems into the SAP Business Information Warehouse.

DDL. See *Data Definition Language*.

debugger. A facility on the Message Flows view in the Control Center that enables message flows to be visually debugged.

deploy. To make the configuration and topology of the broker domain operational.

dimension. Data that further qualifies or describes a measure, or both, such as amounts or durations.

distributed application In message queuing, a set of application programs that can each be connected to a different queue manager, but that collectively constitute a single application.

DML. See *Data Manipulation Language*.

drill-down. Iterative analysis, exploring facts at more detailed levels of the dimension hierarchies.

dynamic SQL. SQL that is interpreted during execution of the statement.

engine. A program that performs a core or essential function for other programs. A database engine performs database functions on behalf of the database user programs.

enrichment. The creation of derived data. An attribute-level transformation performed by a type of algorithm to create one or more new (derived) attributes.

enterprise replication. An asynchronous, log-based tool for replicating data between IBM Informix Dynamic Server (IDS) database servers.

extenders. Program modules that provide extended capabilities for DB2 and are tightly integrated with DB2.

FACTS. A collection of measures and the information to interpret those measures in a given context.

federation. Providing a unified interface to diverse data.

gateway. A means to access a heterogeneous data source. Can use native access or Open Database Connectivity (ODBC) technology.

grain. The fundamental lowest level of data represented in a dimensional fact table.

instance. A particular realization of a computer process. Relative to the database, the realization of a complete database environment.

Java Database Connectivity (JDBC). An API that has the same characteristics as ODBC, but is specifically designed for use by Java database applications.

Java Development Kit (JDK). Software package used to write, compile, debug, and run Java applets and applications.

Java Message Service (JMS). An API that provides Java language functions for handling messages.

Java Runtime Environment (JRE). A subset of the JDK that enables you to run Java applets and applications.

materialized query table. A table where the results of a query are stored for later reuse.

measure. A data item that measures the performance or behavior of business processes.

message domain. The value that determines how the message is interpreted (parsed).

message flow. A directed graph that represents the set of activities performed on a message or event as it passes through a broker. A message flow consists of a set of message processing nodes and message processing connectors.

message parser. A program that interprets the bit stream of an incoming message and creates an internal representation of the message in a tree structure. A parser is also responsible for generating a bit stream for an outgoing message from the internal representation.

metadata. Typically called data (or information) about data. It describes or defines data elements.

MOLAP. Multidimensional OLAP. Can be called MD-OLAP. Refers to OLAP that uses a multidimensional database as the underlying data structure.

multidimensional analysis. Analysis of data along several dimensions, for example, analyzing revenue by product, store, and date.

multitasking. Operating system capability that allows multiple tasks to run concurrently, taking turns using the resources of the computer.

multithreading. Operating system capability that enables multiple concurrent users to use the same program. This saves the overhead of initiating the program multiple times.

nickname. An identifier that is used to reference the object located at the data source that you want to access.

node. An instance of a database or database partition.

node group. Group of one or more database partitions.

ODS. See *operational data store*.

online analytical processing (OLAP). Multidimensional data analysis, performed in real time. Not dependent on an underlying data schema.

Open Database Connectivity (ODBC). A standard API for accessing data in both relational and non-relational database management systems. Using this API, database applications can access data stored in database management systems on a variety of computers even if each database management system uses a different data storage format and programming interface. ODBC is based on the call-level interface (CLI) specification of the X/Open SQL Access Group.

operational data store (ODS). (1) A relational table for holding clean data to load into InfoCubes, and can support some query activity. (2) Online Dynamic Server, an older name for IDS.

optimization. The capability to enable a process to execute and perform in such a way as to maximize performance, minimize resource utilization, and minimize the process execution response time delivered to the user.

partition. Part of a database that consists of its own data, indexes, configuration files, and transaction logs.

pass-through. The act of passing the SQL for an operation directly to the data source without being changed by the federation server.

pivoting. Analysis operation where a user takes a different viewpoint of the results, for example, by changing the way the dimensions are arranged.

primary key. Field in a table that is uniquely different for each record in the table.

process. An instance of a program running in a computer.

program. A specific set of ordered operations for a computer to perform.

pushdown. The act of optimizing a data operation by pushing the SQL down to the lowest point in the federated architecture where that operation can be executed. More simply, a pushdown operation is executed at a remote server.

RSAM. Relational Sequential Access Method. The disk access method and storage manager for the Informix DBMS.

ROLAP. Relational OLAP. Multidimensional analysis using a multidimensional view of relational data. A relational database is used as the underlying data structure.

roll-up. Iterative analysis, exploring facts at a higher level of summarization.

server. A computer program that provides services to other computer programs (and their users) in the same or other computers. However, the computer that a server program runs in is also frequently referred to as a server.

shared nothing. A data management architecture where nothing is shared between processes. Each process has its own processor, memory, and disk space.

static SQL. SQL that has been compiled prior to execution. Typically provides best performance.

subject area. A logical grouping of data by categories, such as customers or items.

synchronous messaging. A method of communication between programs in which a program places a message on a message queue and then waits for a reply before resuming its own processing.

task. The basic unit of programming that an operating system controls. Also see *multitasking*.

thread. The placeholder information associated with a single use of a program that can handle multiple concurrent users. See also *multithreading*.

unit of work. A recoverable sequence of operations performed by an application between two points of consistency.

user mapping. An association made between the federated server user ID and password and the data source (to be accessed) user ID and password.

virtual database. A federation of multiple heterogeneous relational databases.

warehouse catalog. A subsystem that stores and manages all the system metadata.

xtree. A query-tree tool that enables you to monitor the query plan execution of individual queries in a graphical environment.

Abbreviations and acronyms

ACS	access control system	CRM	customer relationship management
ADK	Archive Development Kit	CS	cursor stability
AIO	asynchronous input/output	CSM	communication support module
API	application programming interface	DAC	Discretionary Access Control
AQR	automatic query rewrite	DAS	DB2 Administration Server
AR	access register	DB	database
ARM	automatic restart manager	DB2 II	DB2 Information Integrator
ART	access register translation	DB2 UDB	DB2 Universal Database™
ASCII	American Standard Code for Information Interchange	DBA	database administrator
AST	application summary table	DBDK	DataBlade Development Kit
ASYNC	asynchronous	DBM	database manager
AWS	Amazon Web Services	DBMS	database management system
BLOB	binary large object	DBSA	database server administrator
BTS	basic text search	DBSSO	database system security officer
BW	Business Information Warehouse (SAP)	DCE	distributed computing environment
CA	Continuous Availability	DCM	Dynamic Coserver Management
CCMS	Computing Center Management System	DCOM	Distributed Component Object Model
CDR	Continuous Data Replication	DDL	Data Definition Language
CGI	Common Gateway Interface	DES	Data Encryption Standard
CLI	call-level interface	DIMID	Dimension Identifier
CLOB	character large object	DLL	dynamic link library
CLP	command line processor	DML	Data Manipulation Language
CLR	Continuous Log Recovery	DMS	database managed space
CLR	Continuous Log Restore	DOS	denial-of-service
CORBA	Common Object Request Broker Architecture	DPF	data partitioning facility
CPU	central processing unit	DRDA	Distributed Relational Database Architecture™
CQL	Common Query Language		

DSA	Dynamic Scalable Architecture	HPL	High Performance Loader
DSN	data source name	HQ	headquarters
DSS	Decision Support System	HR	human resource
EAI	Enterprise Application Integration	I/O	input/output
EBCDIC	Extended Binary Coded Decimal Interchange Code	IBM	International Business Machines Corporation
ECS	E-Commerce Service	ID	identifier
EDA	enterprise data architecture	IDE	Integrated Development Environment
EDA	enterprise data availability	IDS	Informix Dynamic Server
EDU	engine dispatchable unit	II	Information Integrator
EGL	Enterprise Generation Language	IIUG	International Informix User's Group
EGM	Enterprise Gateway Manager	IMS™	Information Management System
EJB™	Enterprise JavaBean	ISA	Informix Server Administrator
ER	enterprise replication	ISAM	Indexed Sequential Access Method
ERP	Enterprise Resource Planning	ISM	Informix Storage Manager
ESB	enterprise service bus	ISV	independent software vendor
ESE	Enterprise Server Edition	IT	information technology
ETL	Extract, Transform, and Load	ITR	internal throughput rate
ETX	Excalibur Text	ITSO	International Technical Support Organization
FP	fix pack	J2EE™	Java 2 Platform Enterprise Edition
FTP	File Transfer Protocol	JAR	Java archive
GB	gigabytes	JDBC	Java Database Connectivity
GIS	geographic information system	JDK	Java Development Kit
GML	Geography Markup Language	JE	Java Edition
GPS	global positioning system	JMS	Java Message Service
GUI	graphical user interface	JRE	Java Runtime Environment
GUID	Globally Unique Identifier	JVM	Java virtual machine
Gb	gigabits	KAIO	kernel AIO
HA	high availability	KB	kilobytes
HADR	High Availability Disaster Recovery	LBAC	label-based access control
HDR	High Availability Data Replication	LBS	location-based services

LDAP	Lightweight Directory Access Protocol	PAM	Pluggable Authentication Module
LLD	Large Object Locator	PDQ	parallel database query
LPAR	logical partition	PDS	partitioned data set
LRU	least recently used	PHP	Hypertext preprocessor (general purpose scripting language)
LSN	Log Sequence Number		
LUN	logical unit number	PIB	parallel index build
Luw	Linux, UNIX, and Windows	PIT	Point-in-Time
LV	logical volume	PSA	persistent staging area
MAC	Mandatory Access Control	QA	quality assurance
MB	megabytes	RAID	Redundant Array of Independent Disks
MDC	multidimensional clustering		
MLS	multilevel security	RBA	relative byte address
MPP	massively parallel processing	RBAC	role-based access control
MQ	message queue	RBW	red brick warehouse
MQI	message queuing interface	RDB	relational database
MQT	materialized query table	RDBMS	relational database management system
MRM	message repository manager		
MTK	DB2 Migration Toolkit for Informix	RID	record identifier
		RPO	recovery point objective
MVC	Model-View-Controller	RR	repeatable read
Mb	megabits	RS	read stability
NFS	Network File System	RSAM	Relational Sequential Access Method
NIS	Network Information Service		
NPI	non-partitioning index	RSS	Really Simple Syndication
O/S	operating system	RSS	Remote Standalone Secondary
OAT	Open Administration Tool		
ODBC	Open Database Connectivity	RTO	recovery time objective
ODS	operational data store	SA	systems administrator
OGC	Open GeoSpatial Consortium	SAN	storage area network
OLAP	online analytical processing	SCB	session control block
OLE	object linking and embedding	SDK	Software Developer Kit
OLTP	online transaction processing	SDS	Shared Disk Secondary
ORDBMS	Object Relational Database Management System	SE	Standard Engine
		SFS	Simple Feature Specification
OS	operating system	SID	surrogate identifier

SMIT	Systems Management Interface Tool	WORF	Web Services Object Runtime Framework
SMP	symmetric multiprocessing	WSDL	Web Service Description Language
SMS	System Managed Space	WWW	World Wide Web
SMX	server multiplexer	XBSA	X-Open Backup and Restore APIs
SOA	service-oriented architecture	XML	Extensible Markup Language
SPL	Stored Procedure Language	XPS	Extended Parallel Server (Informix)
SQL	Structured Query Language		
SRS	Spatial Reference System		
SSJE	Server Studio Java Edition		
SYNC	synchronous		
TB	terabytes		
TCB	thread control block		
TCO	total cost of ownership		
TMU	table management utility		
UDA	user-defined aggregate		
UDAM	user-defined access method		
UDB	Universal Database		
UDF	user-defined function		
UDR	user-defined routine		
UDT	user-defined type		
URL	Uniform Resource Locator		
VG	volume group (RAID disk terminology)		
VII	Virtual Index Interface		
VLDB	very large database		
VP	virtual processor		
VSAM	virtual sequential access method		
VTI	Virtual Table Interface		
WAN	wide area network		
WCS	Web Coverage Service		
WFS	Web Feature Service		
WKB	Well-Known Binary		
WKT	Well-Known Text		
WMS	Web Mapping Service		

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 503. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Developing PHP Applications for IBM Data Servers*, SG24-7218
- ▶ *Informix Dynamic Server V10 . . . Extended Functionality for Modern Business*, SG24-7299
- ▶ *Informix Dynamic Server V10: Superior Data Replication for Availability and Distribution*, SG24-7319
- ▶ *Informix Dynamic Server 11: Advanced Functionality for Modern Business*, SG24-7465
- ▶ *Informix Dynamic Server 11 Extending Availability and Replication*, SG24-7488

Other publications

These publications are also relevant as further information sources:

- ▶ *Built-In DataBlade Modules User's Guide*, G251-2770
- ▶ *C-ISAM DataBlade Module User's Guide, Version 1.0*, G251-0570
- ▶ *Data Director for Web Programmer's Guide, Version 1.1*, G251-0291
- ▶ *Data Director for Web User's Guide, Version 2.0*, G210-1401
- ▶ *DataBlade API Function Reference*, G251-2272
- ▶ *DataBlade API Programmer Guide*, G251-2273
- ▶ *DataBlade Developer's Kit User Guide*, G251-2274
- ▶ *DataBlade Module Development Overview*, G251-2275
- ▶ *DataBlade Module Installation and Registration Guide*, G251-2276-01

- ▶ *Geodetic DataBlade Module User's Guide, Version 3.11*, G251-0610
- ▶ *Guide to SQL: Syntax*, G251-2284-02
- ▶ *IBM Informix Database Design and Implementation Guide*, G251-2271
- ▶ *IBM Informix DataBlade API Function Reference*, G229-6364
- ▶ *IBM Informix DataBlade API Programmer's Guide*, G229-6365
- ▶ Doe, Carlton. *IBM Informix Dynamic Server 11: The Next Generation in OLTP Data Server Technology*. McPress, 2007, ISBN-10 1583470751, ISBN-13 978-1583470756.
- ▶ *IBM Informix Dynamic Server Administrator's Guide, Version 11.1*, G229-6359-01
- ▶ *IBM Informix Dynamic Server Administrator's Reference*, G229-6360
- ▶ *IBM Informix Dynamic Server Enterprise Replication Guide*, G229-6371
- ▶ *IBM Informix Dynamic Server Installation Guide for Microsoft Windows*, G251-2776
- ▶ *IBM Informix Dynamic Server Installation Guide for UNIX and Linux*, G251-2777
- ▶ *IBM Informix Dynamic Server Performance Guide Version 10.0*, G251-2296
- ▶ *IBM Informix Dynamic Server Performance Guide v11.10*, G229-6385
- ▶ *IBM Informix GLS User's Guide*, G229-6373
- ▶ *IBM Informix Guide to SQL: Reference*, G251-2283
- ▶ *IBM Informix Guide to SQL: Syntax*, G229-6375
- ▶ *IBM Informix High-Performance Loader User's Guide*, G229-6377
- ▶ *IBM Informix Spatial DataBlade Module User's Guide*, G229-6405
- ▶ *IDS Administrators Guide*, G251-2267-02
- ▶ *Image Foundation DataBlade Module User's Guide, Version 2.0*, G251-0572
- ▶ *Modeling a BLM Business Case with the IBM Informix Spatial DataBlade, Version 8.10*, G251-0579
- ▶ *Modeling a Forestry Business Case with IBM Informix Spatial DataBlade, Version 8.10*, G251-0580
- ▶ *Read Me First Informix Data Director for Web, Version 2.0*, G251-0512
- ▶ *Spatial DataBlade Module User's Guide, Version 8.20*, G251-1289
- ▶ *TimeSeries DataBlade Module User's Guide, Version 4.0*, G251-0575
- ▶ *TimeSeries Real-Time Loader User's Guide, Version 1.10*, G251-1300
- ▶ *User Defined Routines and Data Types Developer's Guide*, G251-2301

- ▶ *Virtual-Index Interface Programmer's Guide, G251-2302*
- ▶ *Virtual Table Interface Programmer's Guide, G251-2303*

Online resources

These Web sites are also relevant as further information sources:

- ▶ IBM CIO article, "Improve Database Performance on File System Containers in IBM DB2 Universal Database V8.2 using Concurrent I/O on AIX," by Kenton DeLathouwer, Alan Y Lee, and Punit Shah, August 2004.
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0408lee/>
- ▶ "Non-blocking checkpoints in Informix Dynamic ServerDescription2," by Scott Lashley
<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0703lashley/index.html>
- ▶ Downloadable Bladelets and demos
http://www-128.ibm.com/developerworks/db2/zones/informix/library/samples/db_downloads.html
- ▶ Object-relational database extensibility, including DataBlades
http://www.iiug.org/software/index_ORDBMS.html
- ▶ Introduction to the TimeSeries DataBlade
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0510durty2/index.html>
- ▶ Informix Web DataBlade Architecture
<http://www.ibm.com/developerworks/db2/library/techarticle/0207harrison/0207harrison.html>

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Symbols

.NET 2.0-based Web services 321

A

access method
 primary 258, 405
 secondary 258
 UDAM 402–403
accessors 228
administration 3, 25, 162–163
 OAT 192
 remote 173
 robust 113
 SQL-based 164
administration free zone 2, 11, 161
administrator-only mode 134
ADMINMAIL 152
Advanced Access Control Feature 8
aggregate 309
 argument 309
agile 8
AIO (asynchronous input/output) 114
AIOVPS 129
AIX 455
ALARMPROGRAM configuration parameter 106, 152
allocated resources 40
ALTER ACCESS_METHOD 405
am_check 418
am_scancost 417
Amazon
 E-Commerce Service (ECS) 425
 VTI 427
 VTI architecture 437
 VTI example 430
Amazon Web service 425
Apache 56
application development 3, 9, 321
application redirection using server groups 103
appPages 292
approximate numeric types 234
archecker utility 157
architecture 312, 320, 346, 373–374, 376,

390–391, 458
 Amazon VTI 437
 ER 84
 extensible for robust solutions 219
 ffvti 456
 for robust solutions 12
 SDS 80
array 139, 244, 253
asynchronous (ASYNC) mode 77
asynchronous application-to-application message exchange 442
asynchronous event 377
asynchronous I/O 114
asynchronous input/output (AIO) 114
auditing feature 146
AUTO_AIOVPS 45, 129
AUTO_CKPTS 45, 127
AUTO_LRU_TUNING 46, 128
AutoDesk 353
automatic checkpoint 11, 114
automatic tuning 127
 AUTO_AIOVPS 129
 AUTO_CKPTS 127
 AUTO_LRU_TUNING 128
autonomic features 11

B

backup 11, 113–114, 147, 164, 170, 195–196
 database 149
 filters 158
 incremental 149
 levels of 149
 logical log 156
 ON-Bar 151
 ontape 150
 to directory and stdout 150
 tracking 155
backup and restore 18, 147
 external 155
 failover and disaster recovery 93
BAR_MAX_BACKUP 152
BAR_PERFORMANCE 153
basic text search (BTS) 119

- Basic Text Search DataBlade module 22, 290, 402
- Binary DataBlade module 22, 289
- binary large object (BLOB) 118, 283, 311, 451, 453
- blade server 93
- Bladelet 12, 289, 293, 456
- BladeManager 284, 289
- BladePack 284
- BladeSmith 284
- BLOB (binary large object) 118, 283, 311, 451, 453
 - data 452
- blobspace 118, 149
- books 426, 429
- Boolean 310
- boot up time 121
- B-tree index 116, 228, 252, 258
- BTS (basic text search) 119
- BUFFERPOOL 126
- business decision enablement 5
- business environment 24
- business intelligence, data warehousing 99
- business logic 422
- business-to-business scenarios 321

C

- C/C++ Web services 321
- CA (continuous availability) 2
- cache 189, 410, 420
- cache rate 44
- cadastre 233
- calendar 251
- call center 99
- callback function 380
- callbacks 377
- capacity relief 84
- cardinality 290
- cartographic publishing 324
- C-based iterator function 297
- CGI (Common Gateway Interface) 346
- character large object (CLOB) 118, 290, 311, 446, 449, 451
- character strings 267–268, 366
 - Soundex 362
- checkpoint 122, 127, 169, 174–175, 198, 201
- chunk 113, 149, 164, 169, 171, 173, 178, 200
 - cooked 114
 - offline error condition 152
 - raw 114
- C-ISAM DataBlade module 22, 292

- CKPTINTVL 45, 122
- CLEANERS 44
- client application 422
- Client Software Development Kit (SDK) 20
- CLOB (character large object) 118, 290, 311, 446, 449, 451
- CLR (Continuous Log Restore) 8, 82, 84–86, 89–90, 105
 - EDA 75–76, 81
- cluster activity monitoring 105
- collection 311
 - data type 303
- column-level encryption 8
- column-level security 144–145
- commit 441, 454
- Committed Read isolation level 159
- Common Criteria certification 9
- Common Gateway Interface (CGI) 346
- Common Query Language (CQL) 335
- commutator functions 280
- complex qualifier 413
- concurrency 159, 279, 397
- configuration 15, 161, 163, 171, 174, 180–183, 194–195, 209, 259, 283, 406, 422–424, 449
 - file 163, 183
 - parameters 133, 174
- Connect, Informix 20
- connection database 66
- consistency, SOA 255
- console device 105–106
- consolidation 90
- continuous availability (CA) 2
 - feature 7
- Continuous Log Restore (CLR) 8, 82, 84–86, 89–90, 105
 - EDA 75–76, 81
- cooked chunk 114
 - direct I/O 115
- coord_dimension 351
- coordinates 222
- cost functions 280
- CQL (Common Query Language) 335
- CQL operator 340
- CQL predicate PropertyIsBetween 360
- CREATE INDEX statement 415
- CREATE TABLE 146
- CREATE TABLE statement 415
- CREATE TRIGGER 379, 387
- CREATE VIEW 305, 307

customization 253

D

DAC (Discretionary Access Control) 135

data

access 135

availability 98

cache, UDAM 420

distribution 84, 410

encryption 146

 between HDR servers 147

integrity 17, 113, 162, 169

iterators 300

load activity 42

partitioning 120

 round-robin fragmentation 120

replication 75–76, 83–84

retrieval, efficiency 256

sharing 84, 93

Data Definition Language (DDL) 134, 137, 414, 454

Data Director for Web 20

Data Manipulation Language (DML) 414, 454

data partitioning, OLTP 120

data type 2–3, 12, 230, 242, 264, 277, 280,

290–291, 294, 309, 311, 410, 449

 birthday club example 232

 delivery service example 230

 parcel ownership example 235

 standards 259

 tax bills and ownership transactions 238

 TSndx 364, 367

 UDT 327

data warehousing 3, 36, 42, 99

 business intelligence 99

 denormalization 242

 IDS 16

 monitoring and optimization tool 19

 table types 120

database

administration 162

administration system 208

backup 149

connection security 130

creation 167

events 13, 377

objects 114

replication 98

search 362

 searching with Soundex 362

database management system (DBMS) 2, 18

database server 130, 163, 166–167, 170–171,
174–176, 178–186, 193–196, 199, 277, 321, 374,
378–379, 384, 389, 402, 404, 411–412, 414–415,
422

 data processing 321, 422

database server administrator (DBSA) 11, 113

database system security officer (DBSSO) 146

DataBlade 3, 12, 220, 237, 259, 263, 277,

279–280, 284–285, 287–289, 291, 294, 325, 327,
380–382, 384, 386, 394, 397–398, 424, 456, 468

 Basic Text Search 290

 Binary 289

 C-ISAM 292

 Excalibur Text 292

 Fraction example 476

 Geodetic 291, 324

 IDS 220

 modules 22, 287–288

 MQSeries 290

 Node 290

 resiliency 256

 Spatial 290–291, 324

 technology 2

 TimeSeries 292

 Video Foundation 292

 Web 292

 WFS 290

DataBlade API 228, 277–278, 285, 294, 378,
380–382, 384, 393, 395, 397, 403, 410, 420, 430

 C-based iterator functions 297

 extensions for UDAM 410

 implementation Option B 391

 iterator function 309

 mi_call() function 317

 object-relational extensibility 221

DataBlade Developers Kit 24

DataBlade Manager 287

data-centric mashup 375

DBCREATE_PERMISSION 135–136

DBMS (database management system) 2, 18

DBPATH 103

DBSA (database server administrator) 11, 113

DBSERVERNAME 104

dbspace 31, 149, 164–165, 167–170, 172, 178,
196, 199–200

 layout 113

 management of 115

- multiple partitions 117
 - page size 116
 - tblspace tblspace extents 116
 - temporary 119
 - DBSPACETEMP 41, 119
 - hash join 41
 - DBSSO (database system security officer) 146
 - DDL (Data Definition Language) 134, 137, 414, 454
 - DECIMAL data type 312
 - Decision Support Systems (DSS) 4–5, 10, 15, 34–35, 41, 47–49, 119, 133, 155, 175, 184
 - configuration for 36
 - data warehousing 36, 99
 - decision-support memory 39
 - decryption 147
 - DEFAULT clause 140
 - degree of parallelism 152
 - DELETE 343–344, 380, 421, 448, 451, 454, 457
 - denial-of-service flood attacks 133
 - denormalization, data warehousing 242
 - deployment 15
 - Deployment Wizard 9–10, 26
 - component tree 26
 - DescribeFeatureType operation 336
 - descriptor, UDAM 410
 - direct I/O on cooked chunks 115
 - disaster 75
 - disaster recovery 84–85, 90, 93, 147
 - Discretionary Access Control (DAC) 135
 - disk hardware mirroring 92
 - disk management 11, 114
 - disk mirroring 148, 151, 155
 - distributed joins 18
 - distributed queries 441
 - Distributed Relational Database Architecture (DRDA) 19
 - distribution 93, 98
 - DLL (Data Definition Language) 414
 - DML (Data Manipulation Language) 414, 454
 - domain 236
 - DRDA (Distributed Relational Database Architecture) 19
 - DROP 282, 287, 405, 416, 436, 464
 - DROP ACCESS_METHOD 406
 - DROP TABLE 146, 436
 - DS_MAX_QUERIES 39
 - DS_MAX_SCANS 39
 - DS_NONPDQ_QUERY_MEM 39
 - DS_TOTAL_MEMORY 39
 - DSS (Decision Support Systems) 4–5, 10, 15, 34–35, 47–49, 119, 133, 155, 175, 184
 - configuration for 36
 - data warehousing 36, 99
 - PDQ parameter values 41
 - dynamic SQL 293
- E**
- E-Commerce Service (ECS)
 - Amazon 425
 - gSOAP code 427
 - ItemLookup operation 425
 - ItemSearch operation 426
 - ECS (E-Commerce Service)
 - Amazon 425
 - gSOAP code 427
 - ItemLookup operation 425
 - ItemSearch operation 426
 - EDA (enterprise data availability) 10, 75, 84
 - capabilities and failover options 90
 - CLR 75–76, 81
 - ER 75–76, 83
 - HDR 75–77
 - recommended solutions 92
 - RSS 75–76, 78
 - SDS 75–76, 80
 - solutions 10
 - solutions in IDS 76
 - technologies and advantages 88
 - efficiency 256
 - EGL (Enterprise Generation Language) 321
 - encryption 9, 147
 - data 146
 - passwords 146
 - encryption, password 132
 - end node 374
 - enterprise data availability (EDA) 10, 75, 84
 - capabilities and failover options 90
 - CLR 75–76, 81
 - ER 75–76, 83
 - HDR 75–77
 - recommended solutions 92
 - RSS 75–76, 78
 - SDS 75–76, 80
 - solutions 10
 - solutions in IDS 76
 - technologies and advantages 88
 - Enterprise Gateway Manager 18

- with DRDA 19
- Enterprise Generation Language (EGL) Web services 321
- Enterprise Replication (ER) 2, 83–84, 89, 105–106, 441
 - application upgrades 100
 - clustering solutions 84
 - data availability and distribution 98
 - EDA 75–76, 83
 - with HA cluster 86
 - workload balancing with SDS 96
- enterprise service bus (ESB), SOA 442
- environment variables 267, 281
- ER (Enterprise Replication) 2, 83–84, 89, 105–106, 441
 - application upgrades 100
 - clustering solutions 84
 - data availability and distribution 98
 - EDA 75–76, 83
 - with HA cluster 86
 - workload balancing with SDS 96
- ESQL/C 21, 285, 291
- ESQL/COBOL 21
- ESRI 325, 332
 - ArcGIS 351
- ESRI Shapefiles 353
- event
 - how to use 379
 - why use 378
- event alarms 105–106
- event processing implementation Option A 390
- event processing implementation Option B 391
 - named memory 391
 - rollback 391
- event-driven architecture 393
- Excalibur Text DataBlade 291–292, 402
- Excalibur Text Search DataBlade module 22
- Extended Parallel Server (XPS) 16
- extensibility 2–3, 12–13, 263, 269, 276, 283–284, 289, 294–295, 309–310, 326
 - architecture 261
 - consistency 256
 - data server 261
 - efficiency 257
 - IDS 399
 - object-relational movement 257
- extensible architecture 12
 - for robust solutions 219
- extension 220

- extent 178
- external backup 151
- external backup and restore 148, 155
- external routines, security 136
- extspace 119

F

- f_geometry_column 351
- f_table_catalog 351
- f_table_name 351
- f_table_schema 351
- failover 75, 84–85, 93
- fast recovery 121
 - time 121
- fastpath interface 278, 394
- fault tolerance 98
- feature 334
- feature id 334
- feature type 334
- federated 6
- ffvti
 - architecture 456
 - Bladelet 456
 - external file access 402
- filter, backup 158
- FIRST 191, 435, 452, 462, 464
- flat earth 329
- flat-earth model 331
- FLOAT 234
- forest of trees topology 84
- Fract data type 236–237
- Fraction DataBlade example 476
- fragmentation 36, 41, 43, 46, 178, 269
 - data partitioning 120
- fragments 40, 169, 178
- full text search 22
- function polymorphism 328
- functional index 269, 368

G

- Gaia 3 353
- geocoding 325
- geodesy 331
- Geodetic DataBlade 256, 291, 324, 327, 331, 347
 - globes 326
 - maps 326
 - round-earth model 329
 - spatiotemporal queries and WFS 358

- srid 352
- Geodetic DataBlade module 23
- Geodetic Web services 9
- geographic analysis 324
- geographic information system (GIS) 234, 324, 326
 - software 328, 330
 - techniques 325
 - tools 325
- geographic point locations 325
- geographic processing 324
- geographic technology 325
- Geography Markup Language (GML) 328, 353, 356
- geometric shapes 326
- geometry_type 351
- GeoPoint 361
- GeoPolygon data type 332
- geospatial file format 353
- geospatial mapping 353
- get and set methods 228
- GetCapabilities operation 334
- GetCapabilities transaction 353
 - document 356
- GetFeature operation 338
- GIS (geographic information system) 234, 324, 326
 - software 328, 330
 - techniques 325
 - tools 325
- global mode 183
- global positioning system (GPS) 259
- GML (Geography Markup Language) 328, 353, 356
- Google 325
- GoogleEarth 353
- GOOGLEMAPKEY field 65
- GPS (global positioning system) 259
- GPS-enabled devices 259
- GRANT user labels 140, 145
- great-circle, shortest path 330
- GROUP BY 269, 276, 283, 303, 305, 307
- group, new 69
- gSOAP code 427
- GUI 284

H

- HA (high availability) 2, 7, 9, 75–76, 84–85, 93, 105–106, 112–113, 164
 - application redirection 103
 - cluster 85
 - sysmaster database 106

- HA cluster 85
 - ER 86
- hardware mirroring 155
- hash join, DBSPACETEMP 41
- HDR (High Availability Data Replication) 8, 84–86, 89, 91, 105, 278
 - data encryption 147
 - EDA 75–77
- Health Center 196, 212
- heterogeneous data sources 6
- hierarchical server tree 84
- hierarchical tree topology 84
- high availability (HA) 2, 7, 9, 75–76, 84–85, 93, 105–106, 112–113, 164
 - application redirection 103
 - cluster 85
 - sysmaster database 106
- High Availability Data Replication (HDR) 8, 84–86, 89, 91, 105, 278
 - data encryption 147
 - EDA 75–77
- High Performance Loader (HPL) 42, 120
- Home 193
- host variables 182
- hot backup 81
- HPL (High Performance Loader) 42, 120
- HTTP 290, 335, 423–424
- human resources system 100

I

- I/O throughput 36
- IBM Informix TimeSeries DataBlade 250
- IDS
 - administration 3
 - application development 3
 - business environment 24
 - capabilities 6
 - database events 377
 - DataBlade 220
 - decryption functions 147
 - EDA 75
 - EDA solutions 76
 - encryption functions 147
 - extensibility 3, 295, 372
 - features and tools 17
 - LIST collection type 253
 - memory management 228
 - mixed and changing environments 34

- multiple instances 192
- server deployment 24
- service consumption 321
- SOA 321
- solutions 16
- VII 401
- VTI 401
- Web service 321
- WebSphere MQ 444
- IDS (Informix Dynamic Server) 16
- IFX_EXTEND_ROLE 135
- Image Foundation DataBlade module 22
- implicit cast 366
- imported restore 151
- incremental backup 149
- index 162, 169, 176, 202, 204, 228, 269, 276, 363, 367, 402
 - functional 269, 368
 - utilization 46
- inequality conditions 121
- Informix
 - 4GL 20
 - Client Software Development Kit (SDK) 20, 349
 - Connect 20
 - DataBlade technology 2
 - Enterprise Gateway Manager 18
 - Enterprise Gateway Manager with DRDA 19
 - ESQL/C 21
 - ESQL/COBOL 21
 - Java Database Connectivity (JDBC) 21
 - MaxConnect 19
 - NAG DataBlade module 253
 - OnLine 16–17
 - SQL 21
 - Standard Engine 16–17
 - WebSphere MQ 443
- Informix Dynamic Server (IDS) 16
 - administration 3
 - application development 3
 - capabilities 6
 - database events 377
 - DataBlade 220
 - decryption 147
 - EDA solutions 76
 - encryption 147
 - enterprise data availability (EDA) 75
 - extensibility 3, 295, 399
 - features and tools 17
 - instance 84, 454–455
 - LIST collection type 253
 - shared memory 279
 - solutions 16
 - SQL optimizer 413
 - SQL-based administration 192
 - VII 401
 - VTI 401
- Informix Extended Parallel Server (XPS) 16
- informix.sysams catalog 407
- Informix-Connect 349
- INFORMIXDIR 349
- INFORMIXSERVER 103, 349
- infoshp utility 349
- INSERT 175–177, 179, 209, 211, 213, 215, 222, 309, 343–344, 351, 358, 366, 375, 379–380, 387–388, 421, 445, 448, 451–453, 456, 458, 461, 464
- INSERT, DELETE, UPDATE flow 416
- Installation Wizard 25
 - footprint 25
- instance 162, 288, 337, 441, 443, 455
- INSTEAD OF triggers 147, 380
- integration, multiple customizations 253
- invisible 9
- I-Spy 19
- ItemLookup operation 425
- ItemSearch operation 426
- iterator function 282, 296–300, 303, 307, 309

J

- Java 263, 280–283, 293–294, 296, 394, 397, 422, 442
- Java Database Connectivity (JDBC) 21
- Java UDF 395
 - Web services 398
- JavaSoft specification 21
- JDBC (Java Database Connectivity) 17, 21, 282, 321, 422, 454
 - WebSphere Application Server 21
- joins 433

K

- KAIO (Kernel Asynchronous Input/Output) 114, 130
- Kernel Asynchronous Input/Output (KAIO) 114, 130
- key descriptor 410
- key-value pairs 333

KVP syntax 341, 344

L

label-based access control (LBAC) 8–9, 11, 114, 135, 137

 column level 143

 row-level 138

Large Object Locator (LLD) module 291

last committed isolation level 8

latency, minimizing 79

LBAC (label-based access control) 8–9, 11, 114, 135, 137

 column-level 143

 row-level 138

LBS (location-based services) 357

LDAP 132

least recently used (LRU) 44, 114

light append buffer 42

light scan 37

 buffers 38

line 244

linestring 244

Linux 43, 287, 423–424, 455, 466

LIST 303, 311

LIST collection type 253

LLD (Large Object Locator) module 291

load 162, 295, 434

loadshp utility 349, 351

location-based services (LBS) 357

lock table 44

locking 421

LOCKS 44

log recovery 82

LOGBUFF 45

logical log 78, 162, 170–171, 201, 205

 backup 156

 buffer 45

 file 170

logical recovery time 122

logical restore 148

logs 194, 205

LRU (least recently used) 44, 114

LTAPEDEV 150–151

M

MAC (Mandatory Access Control) 135

MAILUTILITY 152

mainstream IT 325

Mandatory Access Control (MAC) 135

MapInfo 325, 353

Mapquest 325

mashup 14, 325, 374

 data-centric 375

 relational 422

MAX_PDQPRIORITY 39

MAXAIOSIZE 38

MaxConnect 19

MEAN() 312

MedianApprox() 318

MedianExact() 317

memory 121, 163–164, 174, 177, 180–181, 184, 189–190, 208, 278, 297–299, 301–303, 312, 315, 317–319, 383, 388–389, 402, 404, 411, 420, 430

 allocation 298, 383–385, 389, 392, 430

 duration 383

 for query 40

 management, IDS 228

 quantum 39

 usage 184

 utilization 36–37

message log file 105–106

message queue

 integration 398

 VTI tables 402

metadata 293, 305, 334, 406, 408–409, 423

Microsoft mapping 325

mirroring 148

MLS (multi-level security) 135

Mode() 319

model 391, 401–402

Model-View-Controller (MVC) 260

MQ 446

MQ Publish and Subscribe functions 449

MQ Read and Receive functions 449

MQ Table mapping functions 451

MQ Utility functions 451

MQSeries DataBlade module 290, 398

multi-level security (MLS) 135

multimedia data management 17

multiple instances 162, 164

MULTISET 303, 311

multithreaded architecture 393

MVC (Model-View-Controller) 260

N

named memory 279, 384

- event processing Option B 391
 - signal, sending 397
- named pipe 43
- named row type 224
- namespace 334
- network 42
- new group 69
- Node DataBlade module 23, 290
- non-blocking checkpoint 8, 169, 174
- nonlogging tables 42
- non-traditional data 257
- not a 3-D product 331
- N-tile (quantile) 318
- NULL values 277

O

- OAT (Open Admin Tool) 2, 10–11, 18, 25, 33, 57, 105, 107, 110, 162, 192, 208, 212, 216
 - administration 192
 - configuration page 68
 - installation 55, 63
 - installation configuration 67
 - main page 72
 - PHP 18
 - software download 60
- object-oriented application programmer 229
- object-oriented programming advantages 221
- object-relational extensibility 219–220
 - movement 257
- ODBC (Open Database Connectivity) 17, 321, 422
 - interface 280
- OGC (Open GeoSpatial Consortium) 321, 326
 - WFS-T 333
- OGC WFS API 8
- OGC WFS specification 13
- OLTP 2–5, 10, 15, 47–49, 93, 169, 174–175, 184, 258
 - configuration 43
 - data partitioning 120
 - data warehousing 36
 - stored procedure 133
 - TCP/IP connections for security 43
 - workload balancing with SDS and ER 96
- oltp2dss.sql file 49
- oltp2dss.sql script 53
- onaudit 146
- ON-Bar 195
 - API 149
- backup and restore 151, 157
 - performance monitoring 153
- onbar 148, 154
 - restore 148
 - script customization 152
- oncheck 37, 162, 164, 169, 179
- ONCONFIG 182
- onconfig 126, 128, 163, 182–183, 187, 281, 283
 - dependent parameters in RTO policy 123
 - file 31, 34, 394, 431, 468
 - parameter values 35
 - parameters 43, 48
- onconfig.std file 125
- online schema evolution 90
- OnLine, Informix 16–17
- onmode 48, 122, 128, 162, 164, 171, 175, 179, 210, 213
- onsmsync 154
- onspaces 162, 164, 167
- onstat 37, 105–106, 122, 163–164, 201, 431
 - trace information 187
- ontape 148
 - backup and restore 150
 - restore 151
- opaque row type 293
 - encapsulation and performance 227
- opaque type 228, 236–237, 293, 364–365
- Open Admin Tool (OAT) 2, 10–11, 18, 25, 33, 57, 105, 107, 110, 162, 192, 208, 212, 216
 - administration 192
 - installation 55, 63
 - installation configuration 67
 - software download 60
- Open Database Connectivity (ODBC) 17, 321, 422
 - interface 280
- Open GeoSpatial Consortium (OGC) 321, 326
 - WFS-T 333
- Open GeoSpatial Consortium Web Feature Service (OGC WFS) API 8, 290
- open standards 422
- Open Systems Backup Services Data Movement (XBSA) API 149
- operator class functions 280
- optimization 375
- optimizer 433, 437, 440
- Option A event processing 390
- Option B event processing 391
 - named memory 391
 - rollback 391

Oracle 441
ORDER BY 269, 276, 283, 303
ordered set 139
out-of-memory error condition 152

P

page 181, 189, 200
 cleaner threads 45
 size 116
PAGERMAIL 152
PAM (Pluggable Authentication Module) 8, 131
 LDAP 132
parallel database query (PDQ) 36, 39–40, 49, 80, 119
 shared memory 40
 stored procedure 133
parallelism 152, 421
parallelization 421
partitioning 113
password
 authentication 130
 encryption 132, 146
PDQ (parallel database query) 36, 39–40, 49, 80, 119
 parameter values 41
 shared memory 40
 stored procedure 133
PDQPRIORITY 39
performance 2, 184, 201, 295, 303–304, 306, 309
 features 11
 tools 19
 tuning 43
Perl Web services 321
permissions, data access 136
PHP 55
 driver 21
 OAT 18
 Web services 321
PHYSBUFF 45, 125
PHYSDBS 125
PHYSFILE 125
physical recovery time 121
Pitney Bowes 325
Pluggable Authentication Module (PAM) 8, 131
 LDAP 132
polyline 244
polymorphism, function 328
primary access method 258, 402, 405, 438

primary-target replication 84
privileges 173
 database connection 114
processes 267, 283, 383, 396, 423
PropertyIsBetween CQL predicate 360
purpose function 403
Python driver 21

Q

qualification descriptor 410
qualifier, VTI and VII 413
quantile 318
quantum 39
quantum unit 39
quasi-spatial techniques 324
query
 memory 40
 plan 305–306, 308
queues 392, 442–443, 446, 450
quotations 423

R

RA_PAGES 44
RA_THRESHOLD 45
RAS_LLOG_SPEED 125
RAS_PLOG_SPEED 125
raw chunk 114
raw disk device 114–115
RBAC (role-based access control) 11, 114, 136
read-ahead 114, 178
read-ahead threshold 45
REAL 234
real-time data 258
 GPS-enabled devices 259
 stock trades 258
Real-Time Loader DataBlade 258
recoverability 279
recovery 75, 121
recovery point objective (RPO) 114
recovery time objective (RTO) 11, 43, 45, 114
 dependent onconfig parameters 123
Redbooks Web site 465, 503
 Contact us xix
relational data sources 401
relational mashup 422
relational tables 401
remote administration 164, 173
remote instances administration tool 33

- Remote Standalone Secondary (RSS) 84–86, 89, 91, 94, 105
 - EDA 75–76, 78
 - SMX communications interface 79
- replication 27, 441
 - environment 84
 - network 86
 - of data 76
 - primary target 84
 - topologies 84
 - update anywhere 84
- resiliency 7, 256
- resources allocated 40
- response file 29
- responsefile.ini file 27
- restartable restore 159
- restore 11, 113–114, 147, 195
 - external 155
 - imported 151
 - ON-Bar 151
 - ontape 150
 - restartable 159
 - table-level 157
- retail pricing 99
- role-based access control (RBAC) 11, 114, 136
- roles 137
- rollback 279, 379, 397, 445, 454
 - event processing Option B 391
- roll-forward mode 77
- round-earth model 329, 331
- round-robin
 - fragmentation 120
 - temporary dbspace 119
- row descriptor 410
- row-id descriptor 410
- row-level security 141
- RPO (recovery point objective) 114
- RSS (Remote Standalone Secondary) 84–86, 89, 91, 94, 105
 - EDA 75–76, 78
 - SMX communications interface 79
- RTO (recovery time objective) 11, 43, 45, 114
 - dependent onconfig parameters 123
- RTO_SERVER_RESTART 45, 122
 - when not to use 126
- R-tree index 293, 329, 332
- Ruby Web services 321

S

- SAP stack 441
- sbspace 118, 149
- scalability 276, 374
- scan descriptor 410
- schema 208, 258, 304–305, 338, 407, 438, 440, 450
 - editing 18
 - editor 21
 - evolution 100
 - upgrade 100
- SDS (Shared Disk Secondary) 80, 84–86, 89–91, 105
 - architecture 80
 - EDA 75–76, 80
 - SMX communications interface 80
 - workload balancing with ER 96
- secondary access method 258, 402
- security 9, 11
 - column-level 144–145
 - components 138
 - database connection 114, 130
 - external routines 136
 - IDS 16
 - label 141
 - labels 138–139, 144
 - policy 139–140, 144
 - row-level 141
- SELECT 169, 173, 190–191, 203, 210, 213, 239, 268–271, 276, 278, 283, 296, 300, 302, 305, 307–308, 317, 375, 380, 409–410, 413, 421, 433, 435, 440, 448–453, 462–464
- selectivity functions 281
- self-maintenance 9
- sending of a signal 396
- sending of information to a file 392
- sensors 176
- serial column 343
- serial8 column 343
- server 1, 15, 24, 71, 161, 392, 431
 - deployment 24
 - group 104
 - application redirection 103
- server multiplexer (SMX) communications interface 79
- Server Studio 18
- Server Studio Sentinel 19
- server.ini file 27
- SERVERNUM 154

- service provider 424
- service-oriented architecture (SOA) 13, 256, 321, 422
 - data integration 320
 - foundation technologies in IDS 11 320
 - framework 320
- set 139
 - ordered 139
- SET ENVIRONMENT 134
- SET ISOLATION COMMITTED READ 159
- SET PDQPRIORITY 134
- SET ROLE 136
- SFS (Simple Feature Specification) 326
- shapefile utilities 349
- Shared Disk Secondary (SDS) 80, 84–86, 89–91, 105
 - architecture 80
 - EDA 75–76, 80
 - SMX communications interface 80
 - workload balancing with ER 96
- shared library 285–287, 299, 395
- shared memory 43
- SHMVIRTSIZE 45
- Show command 194
- signal, sending 396
 - named memory 397
- silent configuration 31
- silent installation 25, 27
- silent log file 30
- Simple Feature Specification (SFS) 326
- smart large object 279
- SMX communication 79
- SMX layer 80
- SOA (service-oriented architecture) 13, 256, 321, 422
 - data integration 320
 - ESB 442
 - foundation technologies in IDS 11 320
 - framework 320
- SOAP 55, 323, 422, 424, 428
 - message 424
- software components 220
- Soundex 362
 - DataBlade 363
 - examples 363
- spatial analysis 324
- spatial coordinates 325
- spatial data 324
- spatial data type 327, 331
- Spatial DataBlade 290–291, 324, 327, 329, 347, 358
 - globes 326
 - maps 326
 - shapefile utilities 349
 - SRS 328
- Spatial DataBlade module 23
- spatial extent 359–360
- spatial operators 329
- Spatial Reference System (SRS) 328
- Spatial Web services 9
- spatiotemporal query 332, 358
- SPL (Stored Procedure Language) 257, 263, 270, 283, 293, 296, 303–304, 306, 309
 - routines 160
- SQL
 - data types 257
 - Informix 21
 - level 422
 - Web services 422
 - optimizer 120, 413
 - UDAM syntax 403
- SQL Administration API 9, 18, 163–164, 167, 170, 173, 179, 182–186, 208
- SQLHOSTS file 104
- sqlhosts file 130–131
- srid 329, 352
- SRS (Spatial Reference System) 328
- standards for data types 259
- startup sensors 179
- startup tasks 178–179
- statistical information 161, 163
- statistics 162, 176, 181–182, 189–190, 203, 405
 - descriptor 410
 - functions 281
- stdout, backup to 150
- stock trades 258
- storage manager 154
- Storage Manager commands 152
- storage_type 351
- stored procedure 13, 133, 161, 166–167, 174, 211–212, 270, 272, 379, 445
- Stored Procedure Language (SPL) 257, 263, 270, 283, 293, 296, 303–304, 306, 309
 - routines 160
- subtypes 328
- support functions 228
- synchronous (SYNC) mode 77
- syntax 182, 403, 405, 412

- ALTER ACCESS_METHOD 405
- DROP ACCESS_METHOD 406
- KVP 341, 344
 - regular expression 371
- sysdbclose 133
- sysdbclose() 379
- sysdbopen 133
- sysdbopen() 379
- sysmaster database 105–106
- sysqltrace table 190
- sysqltrace_info table 190
- sysqltrace_iter table 190
- system catalog 406
- system configuration 18
- system monitoring 18
- system scalability 19
- system-defined alert 166

T

- tab_abstract 352
- tab_keywords 352
- tab_title 352
- table descriptor 410
- table functions 281
- table space 178
- table types 120
- table-level restore 157
- tables 162, 165–167, 169, 176, 189–190, 192, 202, 204, 206–207, 287, 289, 293, 374–376, 388, 402–403, 406, 409, 414, 418–419, 433, 437, 440–441, 453, 456–457
 - primary access method 258
- TAPEDEV 150–151
- tar file 430
- task 174
- tblspace tblspace extents 116
- TBLTBLFIRST 116
- TBLTBLNEXT 116
- TCP/IP
 - connections for security in an OLTP environment 43
- template 317, 428
- temporary dbspaces 119
- text search 290, 394
- The Carbon Project, Gaia 3 353
- thread 166, 210, 277, 314, 431
- time series 23, 248
- time-indexed array 248
- TimeSeries DataBlade module 23, 250, 256, 258, 292
- TimeSeries Real Time Loader DataBlade module 23
- tracing
 - global mode 183
 - onstat command 187
 - user mode 183
- transaction 4, 341
 - boundaries 378
 - processing 4
 - recovery process 4
- Transaction WFS 334
- transition 386
- tree 139
- trigger 380, 445
 - capabilities 380
 - introspection 380
- TRUNCATE 419, 456, 458, 464
- TSndx
 - data type 364, 367
- tuning 11, 113, 127, 175, 201, 309
- two-phase commit protocol 419
- type constructor 251
- type hierarchy 327

U

- UDA (user-defined aggregate) 310–314, 316, 318–319, 327
 - MEAN() 312
 - MedianApprox() 318
 - MedianExact() 317
 - Mode() 319
- UDAM
 - caching data 420
 - CREATE TABLE and CREATE INDEX flow 415
 - DataBlade API extensions 410
 - descriptor 410
 - DROP TABLE or DROP INDEX flow 416
 - INSERT, DELETE, UPDATE flow 416
 - locking 421
 - logging and recovery 420
 - oncheck flow 418
 - onstat options 421
 - parallelization 421
 - SELECT flow 418
 - SQL syntax 403
 - tips and tricks 419

- transactions 419
- truncate flow 419
- UDF (user-defined function) 287, 291, 296
 - calling 394
- UDR (user-defined routine) 12, 134, 162, 167, 174, 220, 223, 231–232, 277–278, 280, 282–284, 289, 297–298, 300, 303, 311, 327, 384, 386, 397, 403–404, 407, 412, 422, 430
- UDT (user-defined type) 231, 234, 250, 327
 - Fract data type 236
- UNIX 19, 152, 194–195, 208, 287, 349, 396
- unloadshp utility 349
- UPDATE 343–344, 380, 421, 448, 451, 454, 457
- UPDATE STATISTICS 160
- update-anywhere replication 84
- user label 140, 145
- user mode 183
- userdata field 410
- user-defined
 - alert 166
 - primary access method 402
 - record 381
 - secondary access method 402
 - statistics functions 281
 - table functions 281
 - virtual processor 394
 - network connection 397
- user-defined access method (UDAM) 402
 - SQL syntax 403
- user-defined aggregate (UDA) 310–314, 316, 318–319, 327
 - MEAN() 312
 - MedianApprox() 318
 - MedianExact() 317
 - Mode() 319
- user-defined function (UDF) 287, 291, 296
 - calling 394
- user-defined routine (UDR) 12, 134, 162, 167, 174, 220, 223, 231–232, 277–278, 280, 282–284, 289, 297–298, 300, 303, 311, 327, 384, 386, 397, 403–404, 407, 412, 422, 430
- user-defined type (UDT) 231, 234, 250, 327
 - Fract data type 236
- user-state information 298

V

- vertex 244
- Video Foundation DataBlade module 24, 292

- views 12, 287, 289, 295, 303, 305, 309
- VII (Virtual Index Interface) 13, 228, 258, 280, 401–402
 - qualifier 413
- Virtual Index Interface (VII) 13, 228, 258, 280, 401–402
 - qualifier 413
- virtual processor 164, 205, 210, 279, 393–394, 430–432
- Virtual Shared Memory segments 41
- Virtual Table Interface (VTI) 13, 252, 257, 281, 401
 - qualifier 413
- VTI (Virtual Table Interface) 13, 252, 257, 281, 401
 - qualifier 413

W

- Web DataBlade module 24, 292
- Web development 20
- Web Feature Service (WFS) 13, 290, 326, 332
 - architecture 346
 - Basic 333
 - capabilities 333
 - DataBlade 290, 325, 327
 - installation and setup 347
 - overview 332
 - publishing location 324
 - spatiotemporal queries 358
 - Transaction 334
 - XLink 334
 - Web Feature Service API 8
 - Web Mapping Service (WMS) 333
 - Web service 13, 283, 295, 325, 353, 375, 422–425, 428, 437–438, 468
 - accessing 398
 - Amazon E-Commerce Service 425
 - IDS 321
 - standards 422
 - Web services
 - .NET 2.0-based 321
 - EGL-based 321
 - Java-based 321
 - OGC 321
 - PHP, Ruby, Perl, and C/C++ 321
 - Web Services Object Runtime Framework (WORF) 321
 - WebSphere 441–442, 455
 - WebSphere Application Server 21
 - WebSphere MQ 441, 443, 455

- description 441
- IDS support 444
- Informix 443
- Informix and other database application use 443
- Message Broker 442
- programming 445
- queue manager 455
- sending messages from IDS to WebSphere MQ 447
- transactions 454
- Well-Known Binary (WKB) 328
- Well-Known Text (WKT) 328, 331
- WFS (Web Feature Service) 13, 290, 326, 332
 - architecture 346
 - Basic 333
 - capabilities 333
 - DataBlade 290, 325, 327
 - installation and setup 347
 - overview 332
 - publishing location 324
 - spatiotemporal queries 358
 - Transaction 334
 - XLink 334
- WFS Delete transaction 344
- WFS GeoPoint 358
- WFS GetFeature operation 357
- WFS GetFeature request 340, 344, 361
- WFS GetFeature transaction 342
- WFS INSERT transaction 343
- WFS Transaction operation 343
- WFS Update Transaction 344
- WFS with Transactional extension 333
- wfsdriver 346
- wfspwcrypt utility 349
- wfssetup 348
- Windows 43, 152, 208, 284, 286–287, 289, 424, 455, 466
- WKB (Well-Known Binary) 328
- WKT (Well-Known Text) 328, 331
- WMS (Web Mapping Service) 333
- WORF (Web Services Object Runtime Framework) 321
- workload balancing 75, 84, 93
- workload partitioning 90

X

- XBSA API 149

- XLink 334
- XML 284, 290, 293, 311, 422–423, 425
- XPS (Extended Parallel Server) 16

Y

- Yahoo 325



Redbooks

Customizing the Informix Dynamic Server for Your Environment

(1.0" spine)
0.875" <-> 1.498"
460 <-> 788 pages



Customizing the Informix Dynamic Server for Your Environment

An administration free zone to reduce the cost of systems management

IBM Informix DataBlade technology for extending applications

Robust flexibility to get the best fit for you

In this IBM Redbooks publication, we provide an overview of some of the capabilities of version 11 of the IBM Informix Dynamic Server (IDS), referred to as IDS 11, that enable it to be easily customized for your particular environment. Although many capabilities are available, the focus of this book is on the areas of ease of administration and application development. We describe and demonstrate these capabilities with examples to show how it can be done and provide a model as you begin your customization.

IDS 11 provides nearly hands-free administration to businesses of all sizes. It also offers significant advantages in availability, manageability, security, and performance. Built on the IBM Informix Dynamic Scalable Architecture (DSA), these capabilities can result in a lower total cost of ownership. For example, many of the typical database administrator operations are self-managed by the IDS database.

IDS customers report that they are using one-third or less of the staff typically needed to manage other database products. Shortened development cycles are also realized due to rapid deployment capabilities and the choice of application development environments and languages. There are also flexible choices for business continuity with replication and continuous availability. By taking advantage of these functions and features of IDS 11, you can customize your IDS implementation to enable you to better satisfy your particular business requirements.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks