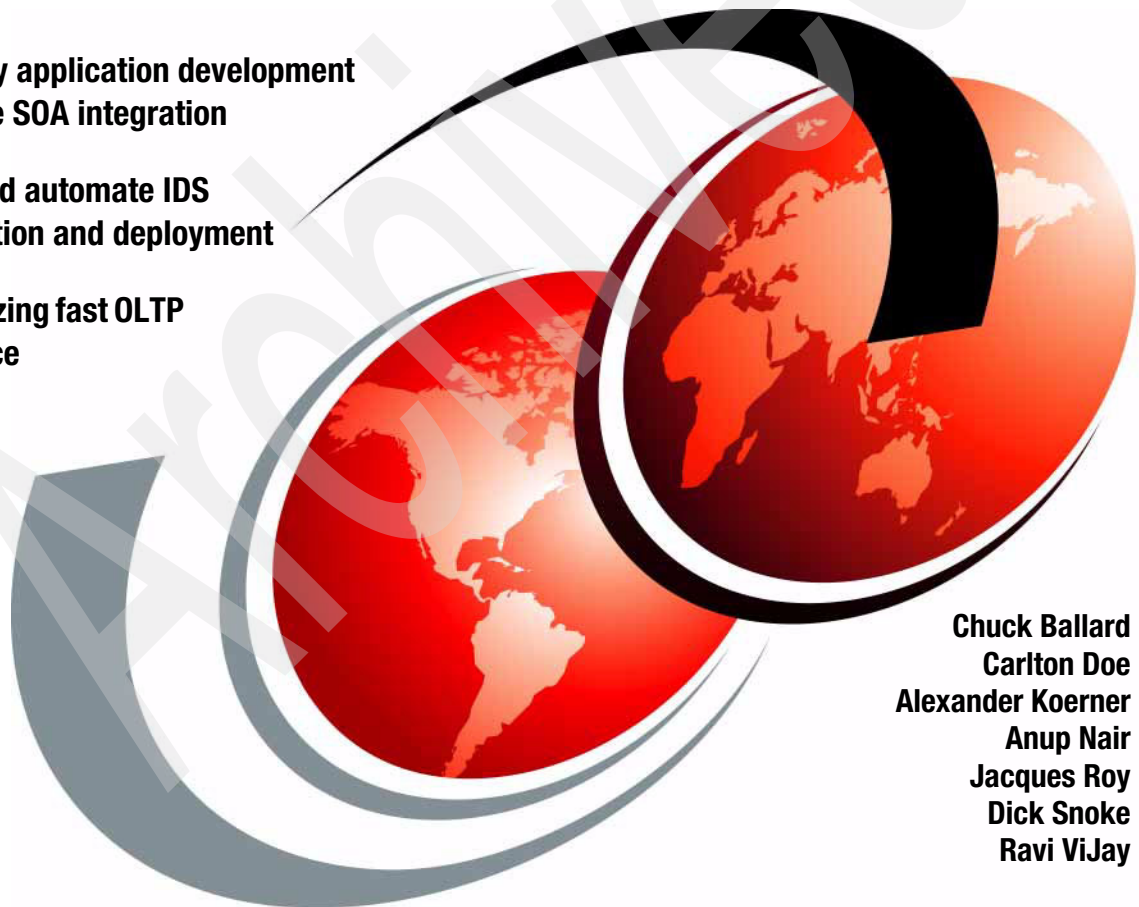


Informix Dynamic Server V10 ... Extended Functionality for Modern Business

Enable easy application development
and flexible SOA integration

Simplify and automate IDS
administration and deployment

Realize blazing fast OLTP
performance



Chuck Ballard
Carlton Doe
Alexander Koerner
Anup Nair
Jacques Roy
Dick Snoke
Ravi ViJay



International Technical Support Organization

**Informix Dynamic Server V10 . . . Extended
Functionality for Modern Business**

December 2006

Archived

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

First Edition (December 2006)

This edition applies to Version 10 of Informix Dynamic Server.

© Copyright International Business Machines Corporation 2006. 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

Notices	ix
Trademarks	x
Preface	xi
The team that wrote this IBM Redbook	xii
Become a published author	xv
Comments welcome	xvi
Chapter 1. IDS essentials	1
1.1 Informix Dynamic Server architecture	3
1.1.1 DSA components: processor	4
1.1.2 DSA components: dynamic shared memory	7
1.1.3 DSA components: intelligent data fragmentation	8
1.1.4 Using the strengths of DSA	10
1.1.5 An introduction to IDS extensibility	14
1.2 Informix Dynamic Server Editions and Functionality	22
1.2.1 Informix Dynamic Server Express Edition (IDS-Express)	23
1.2.2 Informix Dynamic Server Workgroup Edition	23
1.2.3 Informix Dynamic Server Enterprise Edition	25
1.3 New features in Informix Dynamic Server V10	26
1.3.1 Performance	27
1.3.2 Security	29
1.3.3 Administration and usability	31
1.3.4 Availability	33
1.3.5 Enterprise replication	34
1.3.6 APPLICATIONS	36
Chapter 2. Fast implementation	37
2.1 How can I get there from here	38
2.1.1 In-place upgrades	38
2.1.2 Migrations	41
2.2 Things to think about first	50
2.2.1 Physical server components	51
2.2.2 Instance and database design	56
2.2.3 Backup and recovery considerations	59
2.3 Installation and initialization	62
2.4 Administration and monitoring utilities	67
Chapter 3. The SQL language	73

3.1 The CASE clause	74
3.2 The TRUNCATE TABLE command	77
3.2.1 The syntax of the command	78
3.2.2 The DROP TABLE command versus the DELETE command versus the TRUNCATE TABLE command	78
3.2.3 The basic truncation	79
3.2.4 TRUNCATE and transactions	81
3.3 Pagination	81
3.3.1 Pagination examples	82
3.3.2 Pagination for database and Web applications	84
3.3.3 Pagination as in IDS: SKIP m FIRST n	85
3.3.4 Reserved words: SKIP, FIRST. Or is it?	86
3.3.5 Working with data subsets	89
3.3.6 Performance considerations	92
3.4 Sequences	97
3.5 Collection data types	102
3.5.1 Validity of collection data types	103
3.5.2 LIST, SET, and MULTISSET	106
3.6 Distributed query support	109
3.6.1 Types and models of distributed queries	109
3.6.2 The concept	111
3.6.3 New extended data types support	112
3.6.4 DML query support	112
3.6.5 DDL queries support	119
3.6.6 Miscellaneous query support	120
3.6.7 Distinct type query support	123
3.6.8 In summary	125
3.7 External Optimizer Directives	125
3.7.1 What are external optimizer directives	125
3.7.2 Parameters for external directives	126
3.7.3 Creating and saving the external directive	128
3.7.4 Disabling or deleting an external directive	130
3.8 SQL performance improvements	132
3.8.1 Configurable memory allocation	132
3.8.2 View folding	136
3.8.3 ANSI JOIN optimization for distributed queries	144
Chapter 4. Extending IDS for business advantages.	151
4.1 Why extensibility	152
4.1.1 Date manipulation example	152
4.1.2 Fabric classification example	153
4.1.3 Risk calculation example	155
4.2 IDS extensibility features	157

4.2.1 Data types	157
4.2.2 Routines	157
4.2.3 Indexing.	160
4.2.4 Other capabilities	160
4.3 Extending IDS	161
4.3.1 DataBlades	162
4.4 A case for extensibility.	172
Chapter 5. Functional extensions to IDS.	175
5.1 Installation and registration	176
5.2 Built-in DataBlades	177
5.2.1 The Large Object Locator module.	178
5.2.2 The MQ DataBlade module.	178
5.3 Free-of-charge DataBlades	180
5.3.1 The Spatial DataBlade module	180
5.4 Chargeable DataBlades	182
5.4.1 Excalibur Text search	182
5.4.2 The Geodetic DataBlade module	183
5.4.3 The Timeseries DataBlade module	184
5.4.4 Timeseries Real Time Loader	184
5.4.5 The Web DataBlade module.	185
5.5 Conclusion.	186
Chapter 6. Development tools and interfaces	187
6.1 IDS V10 software development overview	188
6.1.1 IBM supported APIs and tools for IDS V10.	188
6.1.2 Embedded SQL for C (ESQL/C) - CSDK	188
6.1.3 The IBM Informix JDBC 3.0 driver - CSDK	190
6.1.4 IBM Informix .NET provider - CSDK	193
6.1.5 IBM Informix ODBC 3.0 driver - CSDK	195
6.1.6 IBM Informix OLE DB provider - CSDK.	197
6.1.7 IBM Informix Object Interface for C++ - CSDK	200
6.1.8 IBM Informix 4GL	201
6.1.9 IBM Enterprise Generation Language.	205
6.1.10 IBM Informix Embedded SQL for Cobol (ESQL/Cobol)	207
6.2 Additional tools and APIs for IDS V10.	209
6.2.1 IDS V10 and PHP support	209
6.2.2 PERL and DBD::Informix	212
6.2.3 Tcl/Tk and the Informix (isqltcl) extension.	213
6.2.4 Python, Informix DB-2.2 and IDS V10.	215
6.2.5 IDS V10 and the Hibernate Java framework.	216
Chapter 7. Data encryption.	219
7.1 Scope of encryption.	220

7.2 Data encryption and decryption	220
7.3 Retrieving encrypted data	223
7.4 Indexing encrypted data	225
7.5 Hiding encryption with views	227
7.6 Managing passwords	229
7.6.1 General password usage	229
7.6.2 The number and form of passwords	229
7.6.3 Password expiration	230
7.6.4 Where to record the passwords	232
7.6.5 Using password hints	232
7.7 Making room for encrypted data	233
7.7.1 Determining the size of encrypted data.	233
7.7.2 Errors when the space is too small	235
7.8 Processing costs for encryption and decrypting	236
Chapter 8. Authentication approaches	241
8.1 Overall security policy	242
8.2 To trust or not to trust, that is the question	242
8.2.1 Complete trust.	243
8.2.2 Partial trust	244
8.2.3 No trust at all.	244
8.2.4 Impersonating a user	245
8.3 Basic OS password authentication	245
8.4 Encrypting passwords during transmission	248
8.5 Using pluggable authentication modules (PAMs)	250
8.5.1 Basic configuration	250
8.5.2 Using password authentication	251
8.5.3 Using challenge-response authentication	251
8.5.4 Application considerations for challenge-response.	251
8.6 Using an LDAP directory	251
8.7 Roles	252
8.7.1 Default roles	253
Chapter 9. Legendary backup and restore	257
9.1 IDS backup and restore technologies	258
9.1.1 Cold, warm, and hot as well as granularity	258
9.1.2 Ontape	261
9.1.3 ON-Bar utility suite	262
9.1.4 External backups.	265
9.2 Executing backup and restore operations.	266
9.2.1 Creating a backup.	267
9.2.2 Verifying backups	269
9.2.3 Restoring from a backup.	271

9.3 New functionality	274
9.3.1 Ontape to STDIO	274
9.3.2 Table Level Point-in-Time Restore (TLR)	275
Chapter 10. Really easy administration	281
10.1 Flexible fragmentation strategies	282
10.1.1 Introduction to fragmentation	282
10.1.2 Fragmentation strategies	283
10.1.3 Table and index creation	285
10.1.4 Alter fragment examples	287
10.1.5 System catalog information for fragments	290
10.1.6 SQEXPLAIN output	291
10.1.7 Applying new fragment methods after database conversion	292
10.1.8 Oncheck utility output	293
10.1.9 Fragmentation strategy guidelines	295
10.2 Shared memory management	296
10.2.1 Database shared memory	297
10.2.2 Managing shared memory	298
10.2.3 Setting SQL statement cache parameters	301
10.3 Configurable page size and buffer pools	303
10.3.1 Why configurable page size	304
10.3.2 Advantages of using the configurable page size feature	304
10.3.3 Specifying page size	306
10.4 Dynamic OPTCOMPIND	308
Chapter 11. IDS delivers services (SOA)	311
11.1 An introduction to SOA	312
11.1.1 An SOA example: an Internet bookstore	312
11.1.2 What are Web services	314
11.2 IDS 10 as a Web service provider	315
11.2.1 IDS Web services based on Enterprise Java Beans (EJBs)	315
11.2.2 IDS and simple Java Beans Web services	315
11.2.3 IDS 10 and EGL Web services	316
11.2.4 EGL Web service providing	316
11.2.5 IDS 10 and WOF (DADX Web services)	319
11.2.6 IDS 10 and other Web services environments (.NET, PHP)	335
11.3 IDS 10 as a Web service consumer	336
11.3.1 Utilizing IDS and Apache's AXIS for Web service consumption	338
11.3.2 Configuring IDS 10 and AXIS 1.3 for the examples	339
11.3.3 The IDS 10 / AXIS Web service consumer development steps	344
11.3.4 The AXIS WSDL2Java tool	344
11.3.5 A simple IDS 10 / AXIS Web service consumer example	345
11.3.6 Consume Web services with IDS and the gSOAP C/C++ toolkit	353

11.3.7 Configuration of IDS 10 and gSOAP for the examples	353
11.3.8 The IDS 10 / gSOAP Web service consumer development steps	355
11.3.9 A simple IDS 10 / gSOAP Web service consumer example	356
Appendix A. IDS Web service consumer code examples	369
IDS10 / gSOAP Web service consumer: udr.c	370
A makefile to create the StockQuote example on Linux	374
Glossary	379
Abbreviations and acronyms	383
Related publications	387
IBM Redbooks	387
Other publications	387
Online resources	388
How to get IBM Redbooks	389
Help from IBM	389
Index	391

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

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

Redbooks (logo)  ™	DB2 Universal Database™	Rational®
developerWorks®	DB2®	Redbooks™
ibm.com®	DRDA®	System p™
AIX®	Informix®	Tivoli®
DataBlade™	IBM®	VisualAge®
Distributed Relational Database	IMS™	WebSphere®
Architecture™	MQSeries®	

The following terms are trademarks of other companies:

Oracle is a registered trademark of Oracle Corporation and/or its affiliates.

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

Snapshot, and the Network Appliance logo are trademarks or registered trademarks of Network Appliance, Inc. in the U.S. and other countries.

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

ActiveX, Expression, Microsoft, Visual Basic, Visual C++, Visual C#, Visual J#, Visual Studio, Windows NT, Windows, Win32, and the Windows logo are trademarks of Microsoft Corporation 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

This IBM® Redbook provides an overview of the Informix® Dynamic Server (IDS), Version 10. IDS is designed to help businesses better use their existing information assets as they move into an on demand business environment. It provides the reliability, flexibility, and ease of maintenance that can enable you to adapt to new customer requirements.

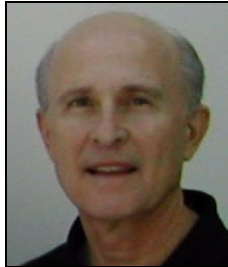
IDS is well known for its blazing online transaction processing (OLTP) performance, legendary reliability, and nearly hands-free administration for businesses of all sizes—all while simplifying and automating enterprise database deployment.

Version 10 offers significant improvements in performance, availability, security, and manageability, including patent-pending technology that virtually eliminates downtime and automates many of the tasks that are associated with deploying mission-critical enterprise systems. New features speed application development, enable more robust enterprise data replication, and enable improved programmer productivity through support of IBM Rational® development tools, JDBC™ 3.0, and Microsoft® .NET as examples. Version 10 provides a robust foundation for e-business infrastructures with optimized Java™ support, IBM WebSphere® certification, and XML and Web services support.

Ready for service-oriented architecture (SOA)? We also include descriptions and demonstrations of support that are specific to IDS for an SOA.

The team that wrote this IBM Redbook

This IBM Redbook was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Poughkeepsie Center. The team members are depicted here, along with a short biographical sketch of each.



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 Bachelors degree and a Masters degree in Industrial Engineering from Purdue University.



Carlton Doe has over 10 years of Informix experience as a Database Server Administrator and 4GL Developer before joining Informix in 2000. During this time, he was actively involved in the local Informix user group and was one of the five founders of the International Informix Users Group (IIUG). Carlton has served as IIUG President, Advocacy Director, and sat on the IIUG Board of Directors for several years. He is best known for having written two Informix Press books on administering the IDS database server as well as several IBM whitepapers and technical articles. Carlton currently works for IBM as a Sr. Certified Consulting IT Specialist in the Global Technical Partner Organization. He lives in Dallas, Texas.



Alexander Koerner is a certified Senior IT-Specialist in the IBM German Channel Technical Sales organization, based in Munich, Germany. He joined Informix in October 1989. He was instrumental in starting and leading the SAP/R3 on Informix project, developed an Informix adaptor to Apple's (NeXT's) Enterprise Object Framework, and has contributed to the success of many strategic projects across the region. Alexander currently focuses on Informix and SOA integration, and 4GL/EGL, but he also actively supports IDS, XML and DataBlade™ technology. His activities also include presentations at conferences and events such as the IBM Information On Demand Conference, IBM Informix Infobahns, regional IUG meetings, XML One, and ApacheCon. Alexander is a member of the German Informatics Society and holds a Masters degree in Computer Science from the Technical University of Berlin.



Anup Nair is a Senior Software Engineer with the IBM Informix Advanced Support team, Menlo Park, California, working on formulating core strategies for OEM/Partners using Informix products. He joined the Informix Advanced Support team in 1998 and has over 15 years of industry experience, holding positions in management, marketing, software development and technical support fields. Anup has a Bachelors degree in Computer Engineering and a Masters degree in Computer Science from Pune University and has a certificate in Project Management.



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*. He is also the author of multiple technical developerWorks® articles on a variety of subjects. Jacques is a frequent speaker at data management conferences, IDUG conferences, and users group meetings.



Dick Snoke is a Senior Certified IT Specialist in the ChannelWorks group in the Unites States. Dick has 33 years experience in the software industry. That experience includes activities such as managing, developing and selling operating systems and DBMS software for mainframes, minicomputers and personal computers. His current focus is on the IBM DBMS products and, in particular, the Informix database products. Dick also supports related areas, such as information integration and high availability solutions.



Ravi ViJay is a systems software engineer in the IBM India Software Labs, working primarily on new features of IBM Informix database releases. His areas of expertise include design and development of applications using IDS. He has also written specifications and developed integration tests for new features of IDS releases. Ravi holds a Bachelors degree in Computer Science from Rajasthan University.

Other contributors

In this section we thank others who have either contributed directly to the content of this IBM Redbook or to its development and publication.

A special thanks

The people in the following picture are from the IBM Informix SQL Development Team in Menlo Park, California. They provided written content for this IBM Redbook based on work that they performed in their development activities. We greatly appreciate their contributions.



From left to right, they are:

- ▶ Joaquim Zuzarte is currently working on the IDS SQL Engine. He also works on the Query Optimizer, with a focus the ANSI outer join query performance.
- ▶ Vinayak Shenoi is the lead SQL developer for IDS and is currently developing features in Extensibility, JDBC support, and Java inside the server.
- ▶ Keshava Murthy is the architect of the IDS SQL and Optimizer components. He has developed features in the SQL, RTREE, distributed queries, heterogeneous transaction management, and extensibility components.
- ▶ Ajaykumar Gupte is a developer of IDS SQL and Extensibility, and contributed to SQL features in DDL, DML, Fragmentation, and View processing.
- ▶ Sitaram Vemulapalli is a developer of IDS SQL and Extensibility, as well as Backup/Restore. He is currently enhancing the distributed query feature.
- ▶ Nita Dembla is a developer in the IDS optimizer group and co-developed the Pagination features in IDS. She has also worked in DB2® development.

Thanks also to the following people for their contributions to this project:

From IBM Locations Worldwide

- Demi Lee, Advisory IT Specialist, Sales and Distribution, Singapore
- Cindy Fung, Software Engineer, IDS Product Management, Menlo Park, CA
- Pat Moffatt, Program Manager, Education Planning and Development, Markham, ON Canada

From the ITSO, San Jose Center

- Mary Comianos, Operations and Communications
- Deanna Polm, Residency Administration
- Emma Jacobs, Graphics

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll 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'll 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 Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook 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 essentials

IBM Informix Dynamic Server 10 (IDS) continues a long-standing tradition within IBM and Informix of delivering first-in-class database servers. It combines the robustness, high performance, availability, and scalability that is needed by today's modern business.

Complex, mission-critical database management applications typically require a combination of online transaction processing (OLTP), batch, and decision-support operations, which include OLAP. Meeting these needs is contingent upon a database server that can scale in performance as well as in functionality. The database server must adjust dynamically as requirements change—from accommodating larger amounts of data, to changes in query operations, to increasing numbers of concurrent users. The technology should be designed to use all the capabilities of the existing hardware and software configuration efficiently, including single and multiprocessor architectures. Finally, the database server must satisfy user demands for more complex application support, which often uses nontraditional or “rich” data types that cannot be stored in simple character or numeric form.

IDS is built on the IBM Informix Dynamic Scalable Architecture (DSA). It provides one of the most effective solutions available—a next-generation parallel database 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. With version 10, IDS increases its lead over the database landscape with even

faster performance, communication and data encryption for high security environments, table-level restoration and other backup/recovery enhancements, improvements in administration reducing the cost to operate the database server and more.

IDS delivers proven technology that efficiently integrates new and complex data directly into the database. It handles time-series, spatial, geodetic, XML (Extensible Markup Language), video, image and other user-defined data side-by-side with traditional legacy data to meet today's most rigorous data and business demands. IDS helps businesses to lower their total cost of ownership (TCO) using its 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®, Microsoft Windows®, and UNIX® operating environments.

The maturity and success of IDS is built on more than 10 years of widespread use in critical business operations, which attests to its stability, performance, and usability. IDS 10 moves this already highly successful enterprise relational database to a new level.

This IBM Redbook introduces briefly the technological architecture which supports all versions of IDS and then describes in greater detail some of the new features that are available in IDS 10. Most of these features are unique in the industry and are not available in any other database server available today. With version 10, IDS continues to maintain and accelerate its lead over other database servers on the market today enabling customers to use information in new and more efficient ways to create business advantage.

1.1 Informix Dynamic Server architecture

High system performance is essential for maintaining maximum throughput. IDS maintains industry-leading performance levels through multiprocessor features, shared memory management, efficient data access and cost-based query optimization. IDS is available on many hardware platforms. Because the underlying platform is not apparent to applications, the engine can migrate easily to more powerful computing environments as needs change. This transparency enables developers to take advantage of high-end symmetric multiprocessing (SMP) systems with little or no need to modify application code.

Database server architecture is a significant differentiator and contributor to the engine's performance, scalability, and ability to support new data types and processing requirements. Almost all database servers available today use an older technological design that requires each database operation for an individual user (as examples, read, sort, write, and communication) to invoke a separate operating system process. This architecture worked well when database sizes and user counts were relatively small. Today, these types of servers spawn many hundreds and into the thousands of individual processes that the operating system must create, queue, schedule, manage/control, and then terminate when no longer needed. Given that, generally speaking, any individual system CPU can only work on one thing at a time—and the operating system works through each of the processes before returning to the top of the queue—this database server architecture creates an environment where individual database operations must wait for one or more passes through the queue to complete their task. Scalability with this type of architecture has nothing to do with the software; it is entirely dependent on the speed of the processor—how fast it can work through the queue before it starts over again.

The IDS server architecture is based on advanced technology that efficiently uses virtually all of today's hardware and software resources. Called the Dynamic Scalable Architecture (DSA), it fully exploits the processing power available in SMP environments by performing similar types of database activities (such as I/O, complex queries, index builds, log recovery, inserts, and backups/restores) in parallelized groups rather than as discrete operations. The DSA design architecture includes built-in multi-threading and parallel processing capabilities, dynamic and self-tuning shared memory components, and intelligent logical data storage capabilities, supporting the most efficient use of all available system resources. We discuss each component of DSA in the following sections.

1.1.1 DSA components: processor

IDS provides the unique ability to scale the database system by employing a dynamically configurable pool of database server processes called *virtual processors* (VPs). Database operations such as a sorted data query are segmented into task-oriented subtasks (for example, data read, join, group, sort) for rapid processing by virtual processors that specialize in that type of subtask. Virtual processor mimics the functionality of the hardware CPUs in that virtual processors schedule and manage user requests using multiple, concurrent threads, as illustrated in Figure 1-1.

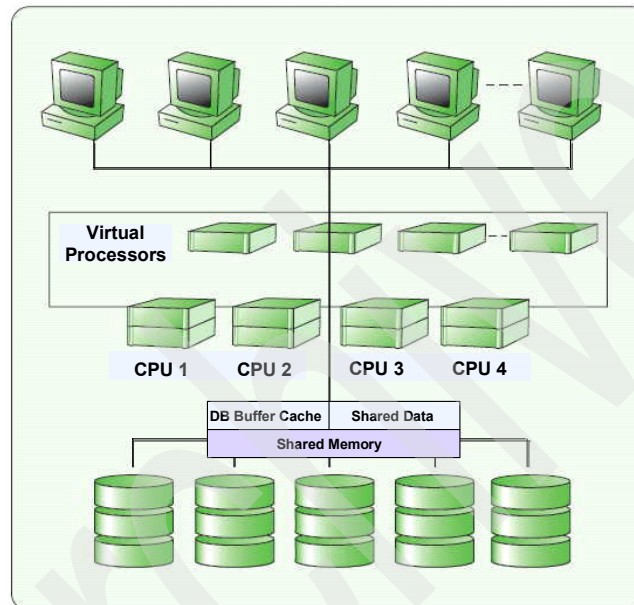


Figure 1-1 IDS pool of VPs for parallel tasks execution

A thread represents a discrete task within a database server process and many threads can execute simultaneously, and in parallel, across the pool of virtual processors. Unlike a CPU process-based (or single-threaded) engine, which leaves tasks on the system CPU for its given unit of time (even if no work can be done, and thus wasting processing time), virtual processors are multi-threaded. Consequently, when a thread is either waiting for a resource or has completed its task, a thread switch will occur and the virtual processor will immediately work on another thread. As a result, precious CPU time is not only saved, but it is used to satisfy as many user requests as possible in the given amount of time. This is referred to as *fan-in parallelism* and is illustrated in Figure 1-2.

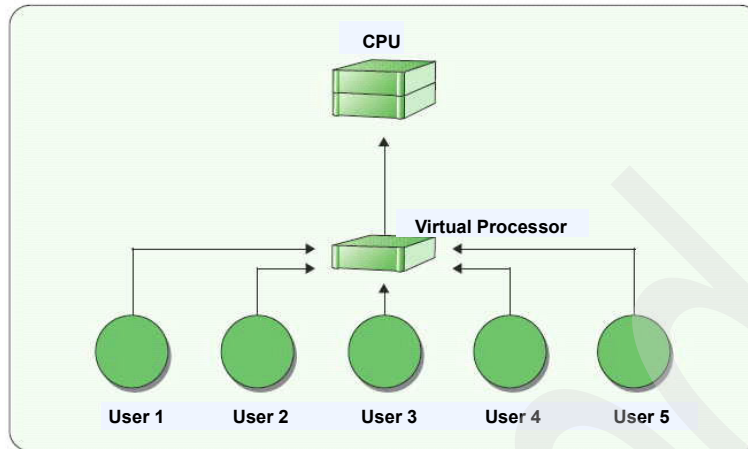


Figure 1-2 Fan-in parallelism for efficient hardware resource use

Not only can one virtual processor respond to multiple user requests in any given unit of time, but one user request can also be distributed across multiple virtual processors. For example, with a processing-intensive request such as a multi-table join, the database server divides the task into multiple subtasks and then spreads these subtasks across all available virtual processors. With the ability to distribute tasks, the request is completed quicker. This is referred to as *fan-out parallelism* and is illustrated in Figure 1-3.

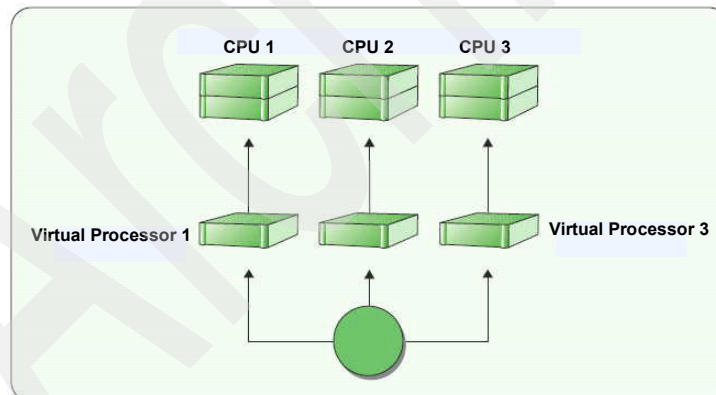


Figure 1-3 Fan-out parallelism uses many VPs to process a single SQL operation

Together with fan-in parallelism, the net effect is more work being accomplished quicker than with single-threaded architectures; in other words, the database server is faster.

Dynamic load balancing occurs within IBM IDS because threads are not statically assigned to virtual processors. Outstanding requests are serviced by the first available virtual processor, balancing the workload across all available resources. For efficient execution and versatile tuning, virtual processors can be grouped into classes—each optimized for a particular function, such as CPU operations, disk I/O, communications and administrative tasks as illustrated in Figure 1-4.

An administrator can configure the instance with the appropriate number of virtual processors in each class to handle the workload. Adjustments can be made while the instance is online without interrupting database operations in order to handle occasional periods of heavy activity or different load mixes.

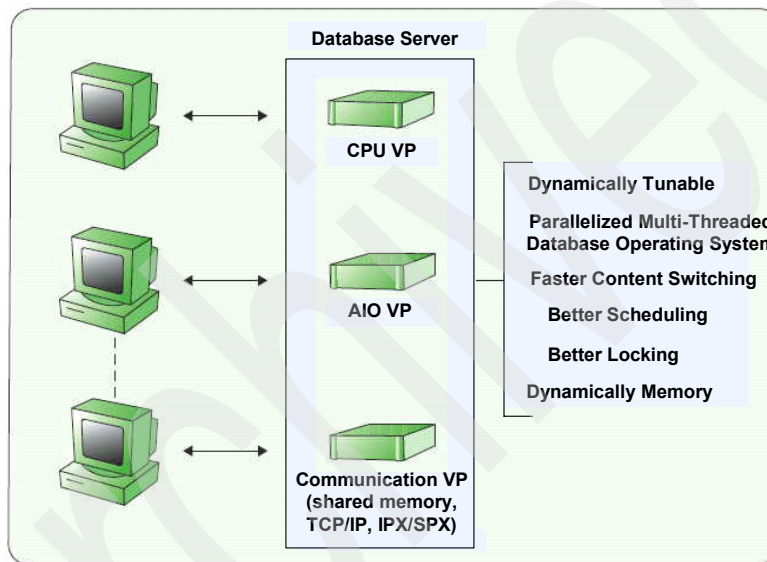


Figure 1-4 VPs are grouped into classes, optimized by function

In UNIX and Linux systems, the use of multi-threaded virtual processors significantly reduces the number of UNIX/Linux processes and, consequently, less context switching is required. In Microsoft Windows systems, virtual processors are implemented as threads to take advantage of the operating system's inherent multi-threading capability. Because IBM IDS includes its own threading capability for servicing client requests, the actual number of Windows threads is decreased, reducing the system thread scheduling overhead and providing better throughput.

In fully utilizing the hardware processing cycles, IBM IDS engines do not need as much hardware power to achieve comparable to better performance than other database servers. In fact, real-world tests and customer experiences indicate

IBM IDS only needs between 25% to 40% of the hardware resources to meet or exceed the performance characteristics of single-threaded or process-based database servers.

1.1.2 DSA components: dynamic shared memory

All memory that is used by IBM IDS is shared among the pool of virtual processors. Beyond a small initial allocation of memory for instance-level management, usually a single shared memory portion is created and used by the virtual processors for all data operations. This portion contains the buffers of, as examples, queried and modified data, sort, join and group tables, and lock pointers. What is unique to IDS is that should database operations require more (or less) shared memory, additional segments are added and dropped dynamically from this portion without interrupting user activities. An administrator can also make similar modifications manually while the instance is running. When a user session terminates, the thread-specific memory for that session is freed within the portion and reused by another session.

The buffer pool is used to hold data from the database disk supply during processing. When users request data, the engine first attempts to locate the data in the buffer pool to avoid unnecessary disk I/Os. Depending on the characteristics of the engine workload, increasing the size of the buffer pool can result in a significant reduction in the number of disk accesses, which can help significantly improve performance, particularly for online transaction processing (OLTP) applications.

Frequently used table or index data is held in the buffer pool using a scorecard system. As each element is used, its score increases. A portion of the buffer system holds these high-score elements while the remainder is used to hold less frequently used data. This segmentation of high and low use data is completely transparent to the application; it gets in-memory response times regardless of which portion of the buffer pool contains the requested element. As data elements are used less often, they are migrated from the high use to low use portion. Data buffers in this area are flushed and reused through a FIFO process.

Included in the memory structures for an instance are cached disk access plans for the IDS cost-based optimizer. In most OLTP environments, the same SQL operations are executed throughout the processing day albeit with slightly different variable conditions such as *customer number*, *invoice number*, and so on.

Each time an SQL operation is executed, the database server optimizer must determine the fastest way to access the data. Obviously if the data is already cached in the memory buffers, it is retrieved from there; if not disk access is required. When this occurs, the optimizer has to decide on the quickest way to

get the requested data. It needs to evaluate whether or not an index exists pointing directly to the requested data or if the data has been intelligently fragmented on disk restricting the possible number of dbspaces to look through. When joining data from several tables, the optimizer evaluates which table will provide the data the others will join to and so on. While not really noticeable to users, these tasks take time to execute and affect response time.

Informix Dynamic Server provides a caching mechanism whereby data I/O plans can be stored for re-use by subsequent executions of the same operation. Called, appropriately enough, the SQL Statement Cache, this allocation of instance memory stores the SQL statement and the optimizer's determination of the fastest way to execute the operation. The size of this cache is configurable as well as when an individual SQL operation is cached. Generally speaking, most configurations cache the operation after it has been executed three or more times to prevent filling the cache with single-use operations. The cache can be flushed and refreshed if needed while processing continues.

With dynamic reconfiguration of memory allocations, intelligent buffer management, caching of SQL access plans, and a number of other technologies, Informix Dynamic Server provides unmatched efficiency and scalability of system memory resources.

1.1.3 DSA components: intelligent data fragmentation

The parallelism and scalability of the DSA processor and memory components are supported by the ability to perform asynchronous I/O across database tables and indexes that have been logically partitioned. To speed up what is typically the slowest component of database processing, IDS uses its own asynchronous I/O (AIO) feature, or the operating system's kernel AIO, when available. Because I/O requests are serviced asynchronously, virtual processors do not have to wait for one I/O operation to complete before starting work on another request. To ensure that requests are prioritized appropriately, four specific classes of virtual processors are available to service I/O requests: logical log I/O, physical log I/O, asynchronous I/O and kernel asynchronous I/O. With this separation, an administrator can create additional virtual processors to service specific types of I/O in order to alleviate any bottlenecks that might occur.

The read-ahead feature enables IDS to asynchronously read several data pages ahead from disk while the current set of pages retrieved into memory is being processed. This feature significantly improves the throughput of sequential table or index scans, and user applications spend less time waiting for disk accesses to complete.

Data partitioning

Table and index data can be logically divided into partitions, or fragments, using one or more *partitioning schemes* to improve the ability to access several data elements within the table or index in parallel as well as increase and manage data availability and currency. For example, if a sequential read of a partitioned table were required, it would complete quicker because the partitions would be scanned simultaneously rather than each disk section being read serially from the top to the bottom. With a partitioned table, database administrators can move, associate or disassociate partitions to easily migrate old or new data into the table without tying up table access with mass inserts or deletes.

IDS has two major partitioning schemes that define how data is spread across the fragments. Regardless of the partitioning scheme chosen, or even if none is used at all, the effects are transparent to users and their applications. Table partitions can be set and altered without bringing down the instance and, in some cases, without interrupting user activity within the table. When partitioning a table, an administrator can specify either:

- ▶ Round-robin—Data is evenly distributed across each partition with each new row going to the next partition sequentially.
- ▶ Expression®-based—Data is distributed into the partitions based on one or more sets of logical rules applied to values within the data. Rules can be *range-based*, using operators such as “=”, “>”, “<”, “<=”, MATCHES, IN, and their inverses, or *hash-based* where the SQL MOD operator is used in an algorithm to distribute data.

Depending on the data types used in the table, individual data columns can be stored in different data storage spaces, or *dbspaces* than the rest of the table's data. These columns, which are primarily smart large objects, can have their own unique partitioning strategy that effectively distributes those specific columnar values in addition to the partitioning scheme applied to the rest of the table. Simple LOBs can and should be fragmented into simple blobspaces, however, because they are objects, as far as the instance is concerned, no further fragmentation options are possible.

Indexes can also be partitioned using an expression-based partitioning scheme. A table's index partitioning scheme need not be the same as that used for the associated table. Partitioned indexes can be placed on a different physical disk than the data, resulting in optimum parallel processing performance. Partitioning tables and indexes improves the performance of data-loading and index-building operations.

With expression-based partitioning, the IDS cost-based SQL optimizer can create more efficient and quicker plans using partition elimination to only access those table/index partitions where the data is known to reside or should be

placed. The benefit is that multiple operations can be executing simultaneously on the same table, each in its unique partition, resulting in greater system performance than typical database systems.

Depending on the operating system used, IDS can use *raw* disks when creating dbspaces to store table or index data. When raw disk space is used, IDS uses its own data storage system to allocate contiguous disk pages. Contiguous disk pages reduce latency from spindle arm movement to find the next data element. It also allows IDS to use direct memory access when writing data. With the exception of Windows-based platforms, where standard file systems should be used, using raw disk-based dbspaces provides a measurable performance benefit.

1.1.4 Using the strengths of DSA

With an architecture as robust and efficient as IBM IDS, the engine provides a number of performance features that other engines cannot match.

The High Performance Loader

The High-Performance Loader (HPL) utility can load data very quickly because it can read from multiple data sources (for example, tapes, disk files, pipes or other tables) and load the data in parallel. As the HPL reads from the data sources, it can execute data manipulation operations such as converting from EBCDIC to ASCII (American Standard Code for Information Interchange), masking or changing data values, or converting data to the local environment based on Global Language Support requirements.

An HPL job can be configured so that normal load tasks, such as referential integrity checking, logging and index builds, are performed either during the load or afterwards, which speeds up the load time. The HPL can also be used to extract data from one or more tables for output to one or more target locations. Data manipulation similar to that performed in a load job can be performed during an unload job.

Parallel data query and Memory Grant Manager

The speed with which IDS responds to a data operation can vary depending on the amount of data being manipulated and the database design. While many simple OLTP operations such as single row inserts/updates/deletes can be executed without straining the system, a properly designed database can use IDS features such as parallel data query, parallel scan, sort, join, group, and data aggregation for larger, more complex operations.

The parallel data query (PDQ) feature takes advantage of the CPU power provided by SMP systems and the IDS virtual processors to execute fan-out

parallelism. PDQ is of greatest benefit to more complex SQL operations that are more analytical, or OLAP oriented, than operational, or OLTP oriented. With PDQ enabled, not only is a complex SQL operation divided into a number of sub-tasks but the sub-tasks are given higher or lower priority for execution within the engine's resources based on the overall *PDQ-priority* level requested by the operation.

The Memory Grant Manager (MGM) works in conjunction with PDQ to control the degree of parallelism by balancing the priority of OLAP-oriented user requests with available system resources, such as memory, virtual processor capacity and disk scan threads. Each OLAP query can be constructed to request a percentage of engine resources (that is, PDQ priority level) for execution. The IDS administrator can set query type priorities, adjust the number of queries allowed to run concurrently, and adjust the maximum amount of memory used for PDQ-type queries. The MGM enforces the rules by releasing queries for execution when the proper amounts of system resources are available.

Full parallelism

The parallel scan feature takes advantage of table partitioning in two ways. First, if the SQL optimizer determines that each partition must be accessed, a scan thread for each partition will execute in parallel with the other threads to bring the requested data out as quickly as possible. Second, if the access plan only calls for "1" to "N-1" of the partitions to be accessed, another access operation can execute on the remaining partitions so that two (or more) operations can be active on the table or index at the same time. Because disk I/O is the slowest element of database operations, to scan in parallel or have multiple operations executing simultaneously across the table/index can provide a significant performance boost.

With full, integrated parallelism, IDS can simultaneously execute several tasks required to satisfy an SQL operation. As data is being retrieved from disk or from memory buffers, the IDS parallel sort and join technology takes the incoming data stream and immediately begins the join and sorting process rather than waiting for the scan to complete. If several join levels are required, higher-level joins are immediately fed results from lower-level joins as they occur as illustrated in Figure 1-5.

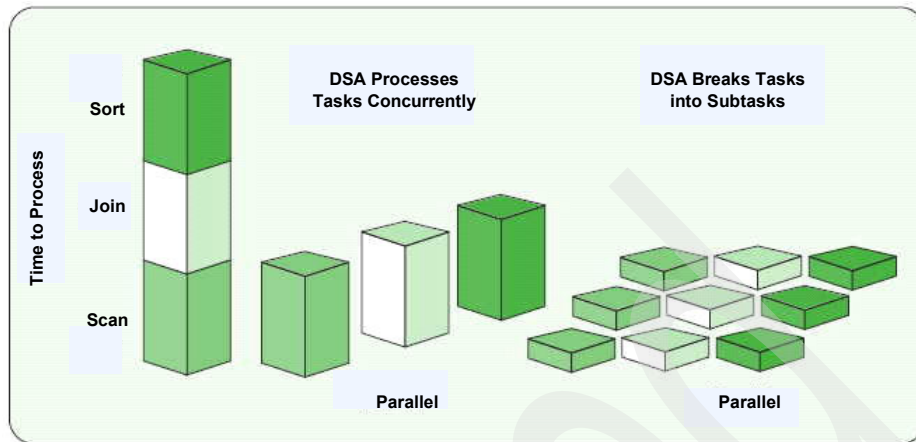


Figure 1-5 IDS integrated parallelism

Similarly, if aggregate functions such as SUM, AVG, MIN or MAX need to be executed on the data or a GROUP BY SQL operator is present, these functions execute in real-time and in parallel with the disk scan, join and sort operations. Consequently, a final result can often be returned to the requester as soon as the disk scan is completed.

Similar to the parallel scan, a parallel insert takes advantage of table partitioning allowing multiple virtual processors and update threads to insert records into the target tables in parallel. This can yield performance gains proportional to the number of disks on which the table was fragmented.

With single-threaded database engines, index building can be a time-consuming process. IBM IDS uses parallelized index-building technology to significantly reduce the time needed to build indexes. During the build process, data is sampled to determine the number of scan threads to allocate. The data is then scanned in parallel (using read-ahead I/O where possible), sorted in parallel and then merged into the final index as illustrated in Figure 1-6.

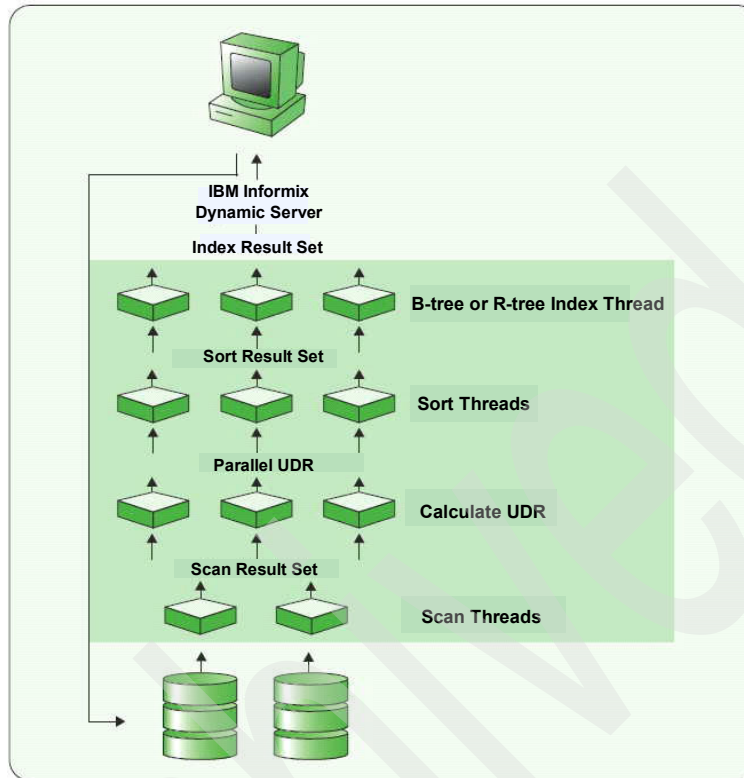


Figure 1-6 IDS parallelism reduces index build and maintenance time

As with other I/O operations already mentioned, everything is done in parallel; the sort threads do not need to wait for the scan threads to complete and the index builds do not wait for the sorts. This parallelization produces a dramatic increase in index-build performance when compared to serial index builds.

IDS cost-based optimizer

IBM IDS uses a cost-based optimizer to determine the fastest way to retrieve data from database tables or indexes based on detailed statistical information about the data within the database generated by the UPDATE STATISTICS SQL command. This statistical information includes more than just the number of rows in the table; the maximum and minimum values for selected columns, value granularity and skew, index depth and more are captured and recorded in overhead structures for the optimizer. The optimizer uses this information to pick the access plan that will provide the quickest access to the data while trying to minimize the impact on system resources. The optimizer's plan is built using estimates of I/O and CPU costs in its calculations.

Access plan information is available for review through several management interfaces so developers and engine administrators can evaluate the effectiveness of their application or database design. The SQL operations under review do not need to actually execute in order to get the plan information. By either setting an environment variable, executing a separate SQL command or embedding an instruction in the target SQL operation, the operation will stop after the operation is *prepared* and the access plan information is output for review. With this functionality, application logic and database design can be tested for efficiency without having to constantly rebuild data back to a known *good* state.

In some rare cases, the optimizer might not choose the best plan for accessing data. This can happen when, for example, the query is extremely complex or there is insufficient statistical information available about the table's data. In these situations, after careful review and consideration, an administrator or developer can influence the plan by including optimizer directives (also known as *optimizer hints*) in the SQL statement. Optimizer directives can be set to use or exclude specific indexes, specify the join order of tables, or specify the join type to be used when the operation is executed. An optimizer directive can also be set to optimize a query to retrieve only the "N" rows of the possible result set.

1.1.5 An introduction to IDS extensibility

IDS provides a complete set of features to extend the database server, including 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, manage non-traditional data structures that are either modeled more like the business environment or contain new types of information never before available for business application processing. Though the data might be considered *nonstandard*, and some types can be table-like in and of themselves, it is stored in a relational manner using tables, columns and rows. In addition, all data, data structures created through Data Definition Language (DDL) commands, and access routines recognize objected-oriented behaviors such as overloading, inheritance and polymorphism. This object-relational extensibility supports transactional consistency and data integrity while simplifying database optimization and administration.

Other database management systems (DBMS) rely on middleware to link multiple servers, each managing different data types, to make it look as though there is a single processing environment. This approach compromises not only performance, but also transactional consistency and integrity because problems with the network can corrupt the data. This is not the case with IDS. Its object-relational technology is built into the DSA core and can be used, or not, at will within the context of a single database environment.

Data types

IDS uses a wide range of data types to store and retrieve data as illustrated in Figure 1-7. The breadth and depth of the data types available to the database administrator and application developer is significant—allowing them to truly define data structures and rules that accurately mirror the business environment rather than trying to approximate it through normalized database design and access constraints.

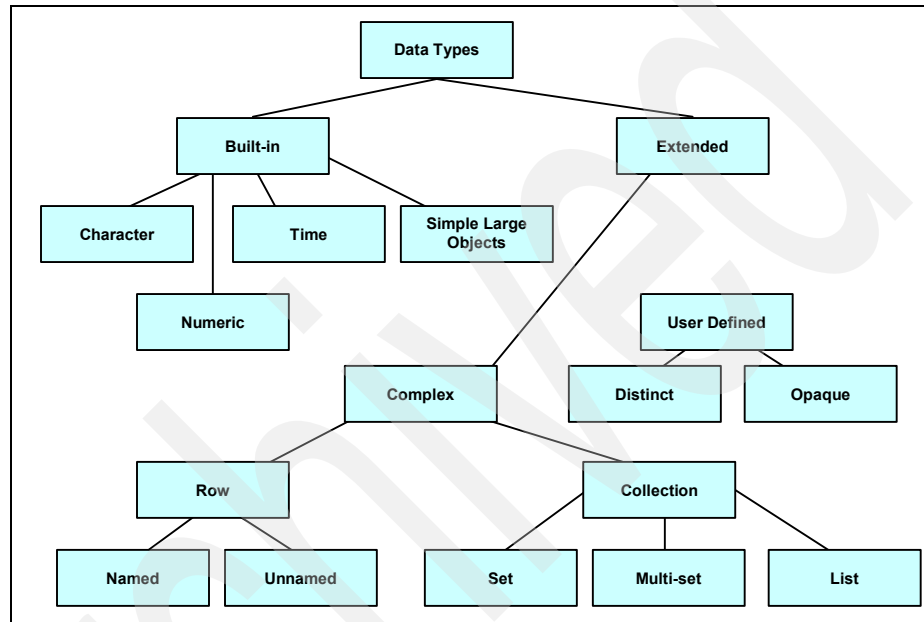


Figure 1-7 The IDS data type tree

Some types, referred to as *built-in* types, include standard data representations such as *character(n)*, *decimal*, *integer*, *serial*, *varchar(n)*, *date*, and *datetime*, alias types such as *money*, and *simple large objects (LOBs)*. Additional built-in types have been added to recent releases of IDS, including *boolean*, *int8*, *serial8*, and an even longer variable length character string, the *lvvarchar*.

Extended data types themselves are of two classes, including:

- ▶ Super-sets of built-in data types with enhanced functionality
- ▶ Types that were not originally built into the Informix database server but that, when defined, can be used to intelligently model data objects to meet business needs.

The collection type is used to store repeating sets of values within one row of one column that would normally require multiple rows or redundant columns in one or

more tables in a traditional database. The three collection types enforce rules on whether or not duplicate values or data order is significant. Collection data types can be nested and contain almost any type, built-in or extended.

With *row* data types, a new data type can be built that is composed of other data types. The format of a row type is similar to that used when defining columns to build a table—a parameter name and data type. When defined, row types can be used as columns within a table or as a table in and of themselves. With certain restrictions, a row type can be dynamically defined as a table is being created or can be inherited into other tables as illustrated in Figure 1-8.

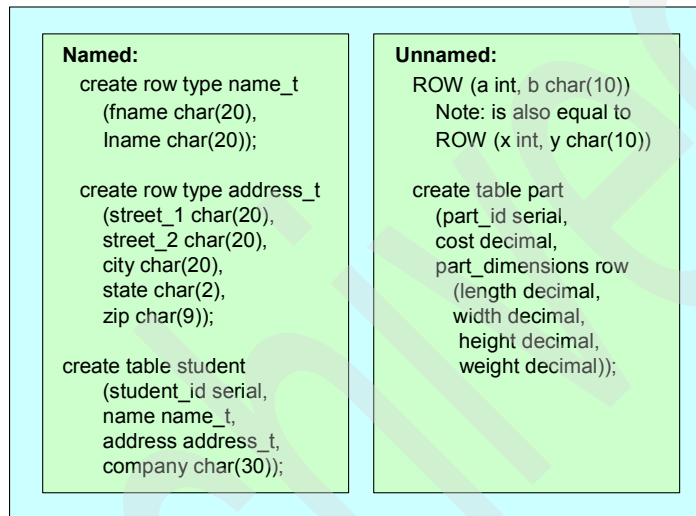


Figure 1-8 Examples of “named” and “unnamed” row types

A distinct data type is an alias for an existing data type. A newly defined distinct data type will inherit all of the properties of its parent type (for example, a type defined using a *float* parent will inherit the elements of precision before and after the decimal point) but because it is a unique type, its values cannot be combined with any other data type but its own without either “casting” the value or using a user-defined routine. Finally, *opaque* data types are those created by developers in C or Java and can be used to represent any data structure that needs to be stored in the database. When using opaque data types, as opposed to the other types already mentioned, the database server is completely dependent on the type’s creator to define all access methods that might be required for the type including insert, query, modify and delete operations.

Extended data types can be used in queries or function calls, passed as arguments to database functions, indexed and optimized in the same way as the core built-in data types. Because any data that can be represented in C or Java

can be natively stored and processed by the database server, IDS can encapsulate applications that have already implemented data types as C or Java structures. Because the definition and use of extended data types is built into the DSA architecture, specialized access routines support high performance. The access routines are fully and automatically recoverable, and they benefit from the proven manageability and integrity of the Informix database server architecture.

Data type casting

With the enormous flexibility and capability that both built-in and extended data types provide to create a database environment that accurately matches the business environment, they must often be used together. To do so requires functionality to convert values between types. This is generally done through the use of casts and, quite often, the casting process uses user-defined functions (UDF).

Casts enable a developer to manipulate values of different data types together or to substitute the value of one type in the place of another. While casts, as an identifiable function, have only been recently added to the SQL syntax, IDS administrators and developers have been using casts for some time; however, they've been hidden in the database server's functionality. For example, to store the value of the integer "12" in a table's character field requires casting the integer value to its character equivalent, and this action is performed by the database server on behalf of the user. The inverse cannot be done because there is no appropriate cast available to represent a character, such as an "a," in a numeric field.

When using "user-defined types" (UDTs), casts must be created to change values between the source type and each of the expected target data. For some types, such as *collections*, *LOBs* and *unnamed row* types, casts cannot be created due to the unique nature of these types. Casts can be defined as either "explicit" or "implicit." For example, with an implicit cast, a routine is created that adds values of type "a" to the value of type "b" by first converting the value of one type to the other type and then adding the values together. The result can either remain in that type or be converted back into the other type before being returned. Any time an SQL operation requires this operation to occur, this cast is automatically invoked behind the scenes and a result returned. An explicit cast, while it might perform the exact same task as an implicit cast, only executes when it is specifically called to manipulate the values of the two data types. While it requires a little more developer effort to use explicit casts, there are more program options available with their use based on the desired output type.

User-Defined Routines, Aggregates and Access Methods

In earlier versions of IDS, developers and administrators who wanted to capture application logic that manipulated data and have it execute within the database server only had stored procedures to work with. Although stored procedures have an adequate amount of functionality, they might not optimize performance. IDS now provides the ability to create significantly more robust and higher performing application or data manipulation logic in the engine where it can benefit from the processing power of the physical server and the DSA.

A “user-defined routine” (UDR) is a collection of program statements that—when invoked from an SQL statement, a trigger, or from another UDR—perform new domain-specific operations, such as searching geographic data or collecting data from Web site visitors. UDRs are most commonly used to execute logic in the database server, either generally useful algorithms or business-specific rules, reducing the time it takes to develop applications and increasing the applications’ speed. UDRs can be either functions that return values or procedures that do not. They can be written in *IBM Informix Stored Procedure Language* (SPL), C or Java. SPL routines contain SQL statements that are parsed, optimized and stored in the system catalog tables in executable format—making SPL ideal for SQL-intensive tasks. Because C and Java are powerful, full-function development languages, routines written in these languages can carry out much more complicated tasks than SPL routines. C routines are stored outside the database server with the path name to the shared library file registered as the UDR. Java routines are first collected into “jar” files, which are stored inside the database server as “smart large objects” (SLOs). Regardless of their storage location, C and Java routines execute as though they were a built-in component of IDS.

A “user-defined aggregate” (UDA) is a UDR that can either extend the functionality of an existing built-in aggregate (for example, *SUM* or *AVG*) or provide new functionality that was not previously available. Generally speaking, aggregates return summarized results from one or more queries. For example, the built-in *SUM* aggregate adds values of certain built-in data types from a query result set and returns their total.

An extension of the *SUM* aggregate can be created to include user-defined data types, enabling the reuse of existing client application code without requiring new SQL syntax to handle the functionality of new data types within the application. To do so, using the example of the *SUM* aggregate, would require creating (and registering) a user-defined function that would overload the “*PLUS*” function and take the user-defined data types, which needed to be added together, as input parameters.

To create a completely new user-defined aggregate requires creating and registering two to four functions to perform the following actions:

- ▶ Initialize the data working space
- ▶ Merge a partial existing result set with the result of the current iteration
- ▶ Merge all the partial result sets
- ▶ Return the final result set with the associated closure and release of system resources to generate the aggregate.

In defining the ability to work with partial result sets, UDAs can, like built-in aggregates, execute in parallel. Functions created and registered for UDAs can be written in SPL, C or Java. Like built-in aggregates, the engine wholly manages a UDA after it is registered (as either an extended or user-defined aggregate).

IDS provides primary and secondary access methods to access and manipulate data stored in tables and indexes. Primary access methods, used in conjunction with built-in data types, provide functionality for table use. Secondary access methods are specifically targeted to indexes and include B-tree and R-tree indexing technologies as well as the CopperEye Indexing DataBlade which significantly reduces the creation and maintenance of indexes on extremely large data sets. Additional user-defined access methods can be created to access other data sources. IDS has methods that provide SQL access to data in either a heterogeneous database table, an external sequential file or to other nonstandard data stored anywhere on the network. Secondary access methods can be defined to index any data as well as alternative strategies to access SLOs. These access methods can be created using the Virtual Table Interface (VTI) and the Virtual Index Interface (VII) server application programming interfaces (APIs).

DataBlades

IBM Informix DataBlade modules bring additional business functionality to the database server through specialized user-defined data types, routines and access methods. Developers can use these new data types and routines to more easily create and deploy richer applications that better address a company's business needs. IDS provides the same level of support to DataBlade functionality that is accorded to built-in or other user-defined types/routines. With *IBM Informix* DataBlade modules, almost any kind of information can be easily managed as a data type within the database server.

There is a growing portfolio of third-party DataBlade modules, or developers can use the *IBM Informix* DataBlade Developer's Kit (DBDK) to create specialized blades for a particular business need.

The following is a partial list of available IBM Informix DataBlade technologies (a current list is available at ibm.com/informix):

- ▶ *IBM Informix TimeSeries DataBlade*—This DataBlade provides a better way to organize and manipulate any form of real-time, time-stamped data. Applications that use large amounts of time-stamped data, such as network analysis, manufacturing throughput monitoring or financial tick data analysis, can provide measurably better performance and reduced data storage requirements with this DataBlade than can be achieved using traditional relational database design, storage and manipulation technologies.
- ▶ *IBM Informix NAG DataBlade*—IBM partnered with the Numerical Algorithms Group (www.nag.co.uk) to provide the ability to perform quantitative analysis of tick-based financial data within the engine itself through the use of routines from their Fortran-based library. These libraries can be applied to the analysis of currency, equity and bond instruments to identify over- and under-valued assets, implement automated trading strategies, price complex instruments such as derivatives, or to create customized products for an institution's corporate customers. Because the analysis occurs in the engine where the data is stored, response times are a fraction of those achieved by systems that must first transfer the data through middleware to a client-side application.
- ▶ *IBM Informix TimeSeries Real-Time Loader*—A companion piece to the IBM Informix TimeSeries DataBlade, the TimeSeries Real-Time Loader is specifically designed to load time-stamped data and make it available to queries in real-time.
- ▶ *IBM Informix Spatial DataBlade and the IBM Informix Geodetic DataBlade*—Provide functionality to intelligently manage complex geospatial information within the efficiency of a relational database model. The IBM Informix Geodetic DataBlade stores and manipulates objects from a “whole-earth” perspective using four dimensions—latitude, longitude, altitude and time. It is designed to manage spatio-temporal data in a global context, such as satellite imagery and related metadata, or trajectory tracking in the airline, cruise or military environment. The IBM Informix Spatial DataBlade is a set of routines that is compliant with open-GIS (geographic information system) standards, which take a “flat-earth” perspective to mapping geospatial data points. Based on ESRI technology (www.esri.com), routines and utilities, this DataBlade is better suited for answering questions such as, “how many grocery stores are within ‘n’ miles of point ‘x’?”, or “what is the most efficient route from point ‘a’ to point ‘b’?”. All IBM Informix geospatial DataBlades take advantage of the built-in IBM Informix R-tree multi-dimensional index technology, resulting in industry-leading spatial query performance. While the IBM Informix Geodetic DataBlade is a for-charge item, the IBM Informix Spatial DataBlade is available at no charge to appropriated licensed users of IDS.

- ▶ *IBM Informix Excalibur Text DataBlade*—Performs full-text searches of documents stored in database tables and supports any language, word or phrase that can be expressed in an 8-bit, single-byte character set.
- ▶ *IBM Informix Video Foundation DataBlade*—Allows strategic third-party development partners to incorporate specific video technologies, such as video servers, external control devices, codecs or cataloging tools, into database management applications. It also provides the ability to manage video content and video metadata regardless of the content's location.
- ▶ *IBM Informix Image Foundation DataBlade*—Provides functionality for the storage, retrieval, transformation and format conversion of image-based data and metadata. While this DataBlade supplies basic imaging functionality, third-party development partners can also use it as a base for new DataBlade modules to provide new functionality, such as support for new image formats, new image processing functions and content-driven searches.
- ▶ *IBM Informix C-ISAM DataBlade*—Provides two separate pieces of functionality to the storage and use of Indexed Sequential Access Method (ISAM)-based data. In those environments where the data is stored in its native flat-file format, the DataBlade provides database server-based SQL access to the data. From a user or application developer perspective, it is as though the data resided in standard database tables. The second element of functionality enables the storage and retrieval of ISAM data in/from standardized database tables while preserving the native C-ISAM application access interface. From a C-ISAM developer's perspective, it is as though the data continued to reside in its native flat-file format; however, with the data stored in a full-functioned database environment, transactional integrity can be added to C-ISAM applications. Another benefit to storing C-ISAM data in database format is gaining access to the more comprehensive backup and recovery routines provided by IDS.

The DBDK is a single development kit for Java-, C- and SPL-based DataBlades and the DataBlade application programming interface. The DataBlade API is a server-side “C” API for adding functionality to the database server, as well as for managing database connections, server events, errors, memory and processing query results. Additional support for DataBlade module developers includes the IBM Informix Developer Zone available at:

<http://www7b.boulder.ibm.com/dmdd/zones/informix/>

Developers can interact with peers, pass along information and expertise, and discuss new development trends, strategies and products. Examples of DataBlades and Bladelets, indexes and access methods are available for downloading and use. Online documentation for the DBDK and other IBM Informix products is available at:

<http://ibm.com/informix/pubs/library/>

1.2 Informix Dynamic Server Editions and Functionality

With database server technology as dynamic and flexible as DSA, it is only natural to assume that customers would be able to buy just the level of IDS functionality they need. IBM has packaged IDS into three *editions*, each tailored from a price and functionality perspective to a specific market segment. Regardless of the edition purchased, each comes with the full implementation of DSA and its unmatched performance, reliability, ease of use, availability and, depending on bundle-driven hardware or connection and scalability restrictions. Table 1-1 includes a brief comparison of the three editions and their feature set.

Table 1-1 IDS editions and their functionality

	Express Edition	Workgroup Edition	Enterprise Edition
Target Market	Midmarket companies (100-999 employees), ISVs for OEM use	Departments within large enterprises, midsized companies	Large enterprises
Function	Full-function, object-relational data server. Includes important capabilities such as: high reliability, security, usability, manageability and performance. Includes Self-healing manageability features, Near-zero administration, Allows integration into Rational Application Developer and Microsoft Visual Studio®, Support for transparent "silent" installation, Supports wide array of development paradigms, Minimal disk space requirements, Simplified installation	Includes all of the features of IDS Express plus features to handle high data loads including Parallel data query, Parallel backup and restore, High-performance loader, High-Availability Data Replication (HDR) can be purchased as an add-on option	Includes all of the features of IDS Workgroup Edition plus features required to provide the scalability to handle high user loads and provide 24x7x365 availability, including Enterprise Replication (ER), High-Availability Data Replication (HDR)

	Express Edition	Workgroup Edition	Enterprise Edition
Customizable	Installation sets common defaults	Installation offers greater flexibility	Supports greatest flexibility to allow tailoring the product to meet the most demanding environments
Scalable	2 CPUs / 4 GB RAM maximum	For V10: 4 CPU, 8 GB memory maximum For V7.31 & V9.4: 2 CPU, 2 GB memory maximum	Unlimited
Upgrade Path	Informix Dynamic Server Workgroup Unlimited Edition	Informix Dynamic Server Enterprise Edition	(Not applicable)

Note that not all license terms and conditions are contained in this document. See an authorized IBM sales associate or reseller for the full details.

1.2.1 Informix Dynamic Server Express Edition (IDS-Express)

Targeted towards small to mid-size businesses or applications requiring enterprise-level stability, manageability and security, Informix Dynamic Server Express Edition (IDS-Express) is available for systems using Linux and Microsoft Windows (server editions). Though limited to systems with 2 physical CPUs and up to 4 GB of RAM, IDS-Express has the full complement of administration and management utilities including online backup, the ability to scale to almost 8 PB (petabytes) of data storage, a reduced installation footprint and a full support for a wide range of development languages and interfaces. It cannot be used however to support Internet-based application connections. For those with very little database server skills, the installation process can be invoked to not only install the database server but also configure an operational instance with a set of default parameters.

1.2.2 Informix Dynamic Server Workgroup Edition

Informix Dynamic Server Workgroup Edition (WGE) is for any size business needing additional power to process SQL operations, efficiently manage very large databases or build a robust, fail-over system to ensure database continuation in the event of natural or man-made outage. IDS-WGE is available on all supported operating systems. Its hardware support is limited to four

physical CPUs and 8 GB of RAM. IDS-WGE cannot be used to support Internet-based application connections.

IDS-WGE has all the components, utilities, storage scalability and so on of IDS-Express but its ability to process more complicated SQL operations on larger databases is enhanced because the PDQ and MGM components discussed in the “Parallel data query and Memory Grant Manager” section of this chapter are available for use. With PDQ and the MGM, database server resources can be pre-reserved then fully deployed without interruption to process any given SQL operation. With the ability to pre-allocate sections of instance memory, more of the *sort*, *join* or *order-by* operations commonly used in larger operations can occur in memory as opposed to temporary disk space further increasing performance.

Managing large databases is not just a SQL processing problem. Typically these environments also require the ability to quickly insert or extract data as well as perform full or targeted backups. IDS-WGE includes additional functionality to do both. The HPL discussed in the “The High Performance Loader” section of this chapter can be used to execute bulk data load and unload operations. It uses DSA’s threading model to process multiple concurrent input or output data streams with or without data manipulation by other threads as the job executes. HPL jobs can be executed while tables and their indexes are active supporting user operations if desired to eliminate maintenance windows for these types of operations increasing system up-time.

Backing up (or restoring) large databases or a specific subset of the data requires very powerful and flexible backup/restore functionality. IDS-WGE includes the ON-Bar API with its ability to do partial or full instance backups and restores. In some cases, these restores can occur while the instance is online and functioning. The granularity of the restore can be to a specific second if necessary. Backups, and their associated restores, can be multi-threaded and output to multiple backup devices to reduce the amount of time required to create a backup or perform a restore.

Through the ON-Bar utility suite and the appropriate third-party tape management software, instance or logical log backups can be incorporated into the management software handling all the other backups occurring across the enterprise. Using this management software, various tape devices and jukebox configurations that exist today, or in the future to store data on high-capacity tape devices or to create backups in parallel across multiple tape devices even if they are attached to different machines can be used. The tape management software’s automated scheduling facilities can be used to schedule and execute backup operations at any desired frequency because it handles all the required communication between the engine’s backup threads and the actual backup devices relieving the IDS administrator of another task.

For data centers which do not have a full-fledged tape management system, IDS-WGE includes a limited functionality tape management system—the Informix Storage Manager (ISM) as well as support for the IBM Tivoli® Storage Manager application. With the ISM, administrators can configure up to 4 locally connected backup devices to be used by the ON-Bar utility for full or partial parallelized instance or logical log backups. There is no scheduling facility in the ISM but the backup process can still be somewhat automated through the execution of a simple shell script by the operating system's crontab or similar scheduling facility. The software maintains, with the exception of one file written out in the %INFORMIXDIR%etc directory, the metadata of what was backed up and when. With this file and the metadata about both instance and logical log backups, full or partial restores can be executed as necessary.

For customers wanting to provide continuation of database services in the event of a natural or man-made outage, the Informix High-Availability Data Replication (HDR) option can be purchased and used with IDS-WGE. With HDR, the results of data manipulation statements such as inserts, updates or deletes are mirrored in real-time to a hot stand-by server. When in stand-by mode, the mirror copy supports query operations and, as a result, can be used by report generation applications to provide data rather than the production server. Depending on the amount of reporting applications, off-loading their execution to the mirror server can provide a measurable performance improvement to day-to-day operations.

In the event the primary server is unavailable, the mirrored copy of the database server is switched to fully updateable mode and can continue to support client applications. Depending on the administrator's preference, when the primary is again available, it can either automatically return to its primary state after receiving the changes executed on the stand-by server or can continue as the new mirror and receive updates in real-time as it used to send when in primary mode.

HDR is extremely simple to set up and use and should be considered required technology for operations needing robust data availability.

1.2.3 Informix Dynamic Server Enterprise Edition

Informix Dynamic Server Enterprise Edition (IDS) includes the full feature set of the database server and can be deployed in any size environment requiring the richest set of functionality supported by the most stable and scalable architecture available in the market today. IDS has no processor, memory or disk access limitations other than those imposed by the operating system it's installed on. With its renowned stability and extensibility, IDS is the perfect database server to use for traditional, Internet-based or other rich media applications.

In addition to HDR, IDS also includes Informix Enterprise Replication (ER), an asynchronous mechanism for the distribution of data objects throughout the enterprise. ER uses simple SQL statements to define the objects to replicate and under what conditions replication occurs. ER preserves “state” information about all the servers and what they’ve received and guarantees delivery of data even if the replication target is temporarily unavailable. Data flow can be either uni- or bi-directional and several conflict resolution rule sets are included to automatically handle near simultaneous changes to the same object on different servers.

One of the greatest ER strengths is its flexibility. Unlike HDR, ER is platform and version independent; data objects from an IDS version 7 instance on Windows can be replicated to an IDS version 10 instance on an AIX® or other operating system without issue. The “replication topology” is completely separate from the actual physical network topology and can be configured to support fully-meshed, hierarchical or forest of trees/snowflake connection paths. ER can easily scale to support hundreds of nodes, each potentially with varying replication rules, without affecting regular transaction processing.

IDS supports concurrent operation of both HDR and ER giving businesses the ability to protect itself from outages as well as automatically migrate data either for application partitioning or distribution / consolidation purposes.

1.3 New features in Informix Dynamic Server V10

In this section we discuss the new features in IDS V10 that provide the extended functionality needed for modern business. IDS V10 provides significant advances in database server security, performance, availability, replication, administration and applications. We have summarized these features in Table 1-2.

Table 1-2 New features of IDS V10

Performance	Security	Admin & Usability	Availability	Enterprise Replication	Applications
Allocating memory for Non PDQ Queries	Column Level Encryption	Single User Mode	Table Level Restore	DRAUTO	JDBC 3.0 Support
Configurable Page Size	Restricting Registration of External Routines	Renaming Dbspaces	Online Index build	Replicate Resync	.Net Support

Performance	Security	Admin & Usability	Availability	Enterprise Replication	Applications
Dynamic OPTCOMPIND	Secure environment check	Ontape use of STUDIO		Template and Master Replicates	
External Optimizer Directives	PAM Authentication	Multiple fragments in one dbspace		Detecting event alarms with the event alarm program	
	Preventing Denial of service attack				

In this section, we describe the IDS V10 features in a bit more detail by category.

1.3.1 Performance

Listed here are brief descriptions of the features that can impact performance.

► **Allocating memory for Non-PDQ Queries:**

The default value of 128 k for non-PDQ queries can be insufficient for queries that specify memory intensive options such as ORDER BY and GROUP BY. Now you can specify more memory than 128 k that is allocated to non-PDQ queries by using the configurable parameter DS_NONPDQ_QUERY_MEM.

The DS_NONPDQ_QUERY_MEM value is calculated during database server initialization based on DS_TOTAL_MEMORY configurable parameter value. The maximum supported value for DS_NONPDQ_QUERY_MEM is 25% of the DS_TOTAL_MEMORY. If you set the value greater than maximum allowed value, the value is changed by database server during the processing of DS_NONPDQ_QUERY_MEM and the message is written in MSGPATH.

► **Configurable Page Size:**

You can define the page size for a standard or temporary dbspace when you create the dbspace. The page size defined must be an integral multiple of 2k or 4k with the maximum limit of 16k. The advantages that you get with this feature are Space efficiency, Increased maximum key size for UNICODE support and Access efficiency. Refer to 10.2, "Shared memory management" on page 296 for more detail about the configurable page size configuration parameter.

► **Dynamic OPTCOMPIND:**

The optimizer uses the OPTCOMPIND value to determine the method to perform the join operations for an ordered pair of tables.

Table 1-3 describes the access plan of the optimizer with respect to OPTCOMPIND values.

Table 1-3 Access Plan and OPTCOMPIND value

Access Plan	OPTCOMPIND Value
Use Index	0
If Repeatable Read than 0; else 2	1
Use lowest cost	2

You can set the value of the OPTCOMPIND dynamically within a session using SET ENVIRONMENT OPTCOMPIND command. The value that you set dynamically within a session gets precedence over the current value specified in the ONCONFIG file, but no other user sessions are affected by it. The default OPTCOMPIND setting is restored after the session terminates.

► **External Optimizer Directives:**

Optimizer directives are comments that instruct the query optimizer how to execute the query. You can create, save and reuse external optimizer directives as temporary workaround solutions to problems when you do not want to change SQL statements in queries. As examples, when your query starts to perform poorly, or when there is no access to the SQL statement, such as those in pre-built and compiled applications purchased from a vendor. An Administrator can create and store the optimizer directives in the sysdirectives catalog table. You can then use IFX_EXTDIRECTIVES environment variable or EXT_DIRECTIVES configuration parameter to enable this feature. Table 1-4 gives the IFX_EXTDIRECTIVES and EXT_DIRECTIVES combination settings.

Table 1-4 IFX_EXTDIRECTIVES and EXT_DIRECTIVES combinations

IFX_EXTDIRECTIVE value	EXT_DIRECTIVE value 0	EXT_DIRECTIVE value 1	EXT_DIRECTIVE value 2
Not Set	OFF	OFF	ON
1	OFF	ON	ON
0	OFF	OFF	OFF

Refer to 3.7, “External Optimizer Directives” on page 125 regarding the use of external optimizer directives.

1.3.2 Security

In this section, we give an overview of the security features of IDS V10.

► Column Level Encryption

You can protect the confidentiality of the data by using the column level encryption feature enhancement of IDS V10. Built-in SQL encryption functions, `encrypt_des` and `encrypt_aes`, can be used to encrypt and decrypt the data, using the latest cryptographic standards. Example 1-1 illustrates a query statement that returns the encrypted values of data that has been stored on disk. Refer to Chapter 7, “Data encryption” on page 219 for more details about this feature.

Example 1-1 Query statement returning encrypted values

```
create table agent
  (agent_num    serial(101),
   fname       char(43),
   lname       char(43));
set encryption password "encryption";
insert into agent values (113, encrypt_aes("Lary"),
encrypt_aes("Paul"));
select * from agent;
```

The encrypted output is:

agent_num	113
fname	0+wT/AAAAEAHvTRGn25s0T90c+ecVM7oIrWHM1yz
lname	0JNv/AAAAEA8mgUh8yH4ePCu06jdIXxYsa1tb3hH

► Restricting Registration of External Routines

You can use the `IFX_EXTEND_ROLE` configuration parameter to restrict the users ability to register the external routines. When the `IFX_EXTEND_ROLE` configuration parameter is set to `ON`, the users who have the built-in `EXTEND` role can create external routines. The Database Server Administrator (DBSA) can use built-in `EXTEND` role to specify which users can register the UDRs that include the `EXTERNAL NAME` clause.

User-defined routines use shared-object files that are external to the database server and that could potentially contain harmful code. The DBSA can use the `GRANT` statement to confer the `EXTEND` role on a user (typically the DBA of a local database), or can use `REVOKE` to withdraw that role from a user. The DBSA can disable this feature by setting to "off" the `IFX_EXTEND_ROLE` configuration parameter. This feature is intended to improve security and to control accessibility.

► Secure Environment Check

During instance initialization or restart, an IDS V10 instance on non-Windows platforms test file and directory ownership and permissions on a number of critical database server files and directories. This check is executed to make sure these files have not been tampered with potentially allowing unauthorized access to instance utilities and operations. If problems are found, the instance will not start and an error message will be written into MSGPATH. Some of the files checked are:

- The permissions on \$INFORMIXDIR and some directories under it. For each directory, check that the directory exists, that it is owned by user informix and the correct group, and that its permissions do not include write permissions for the group or other users.
- The permissions on the ONCONFIG file. The file must belong to the DBSA group. If the DBSA group is group informix (default), then the ONCONFIG file should be owned by user informix too; otherwise, the ownership is not constrained. The file must not have write permissions for others.
- The permissions on the sqlhosts file. Under a default configuration, the sqlhosts file is \$INFORMIXDIR/etc/sqlhosts; the owner should be user informix, the group should be either the informix group or the DBSA group, and there should be no public write permissions. If the file is specified by setting the INFORMIXSQLHOSTS environment variable, then the owner and group are not checked, but public write permissions are not permitted.
- The length of both the file specifications \$INFORMIXDIR/etc/onconfig.std and \$INFORMIXDIR/etc/\$ONCONFIG must each be less than 256 characters.

► PAM Authentication

Pluggable Authentication Module (PAM) is a standardized system for allowing the operating system administrator to configure how authentication is to be done. It allows system administrator to apply different authentication mechanisms for different applications. PAM is supported on Solaris™ and Linux, in both 32 and 64 bit modes. On HP-UX and AIX, PAM is supported in 32 bit mode only. PAM also supports challenge-response protocols. Refer to 8.5, “Using pluggable authentication modules (PAMs)” on page 250 for more details about using PAM.

► Prevent Denial of Service Attack

A denial of service attack is an attempt to make the database server unavailable to its intended users. The attack can occur when someone connects to a port reserved for an engine, but does not send data. A separate session attempts to connect to the server but the blocked listener thread, waiting for the telnet session, cannot accept the connection request of the

second session. If during the waiting period, an attacker launches an attack in a loop mode, a flood of attacks can be received on the connection.

To reduce the risk of denial of service attacks IDS provides multiple listener threads (`listen_authenticate`) to handle connections and imposes limits on the availability of the listener VP for incomplete connections. Two new configuration parameters can be used to customize this feature:

LISTEN_TIMEOUT

Sets the incomplete connection time-out period (in seconds). This is the number of seconds the server waits for the connection. The default value of `LISTEN_TIMEOUT` parameter is 10.

MAX_INCOMPLETE_CONNECTION

You can restrict the number of incomplete requests for the connection using `MAX_INCOMPLETE_CONNECTION` parameter. When the maximum value is reached, an error message stating that server might be under Denial of Service attack is written in the online message log file. The default value of the `MAX_INCOMPLETE_CONNECTION` parameter is 1024.

1.3.3 Administration and usability

In this section, we provide a brief overview of the administration and usability features of IDS V10.

► Single User Mode

Single user is an intermediate mode between quiescent mode and online mode. This is an administrator mode which only allows user informix to connect and perform any required maintenance, including the task requiring the execution of SQL and DDL statements. You can set this mode using the `-j` flag of the `oninit` and the `onmode` commands. The `oninit -j` command brings the server from offline to single user mode and `onmode -j` brings the server from online to single user mode. The server makes an entry in the message log file whenever it enters and exits the single user mode. Figure 1-9 shows an example of using `onmode` command to set the single user mode.

```
sun280-3% onmode -j
This will change mode to single user. Only a DBSA can connect
in this mode.
Do you wish to continue (y/n)? y
All threads which are not owned by user informix will be killed.
Do you wish to continue (y/n)? y
sun280-3% onstat -

IBM Informix Dynamic Server Version 10.00.UC5      -- Single-User -- Up
```

Figure 1-9 The `onmode -j` example

► Renaming Dbspaces

The need to rename the standard dbspaces might arise if you are reorganizing the data in an existing dspace. You can rename a previously defined dspace if you are user informix or have DBA privileges, and the database server is in quiescent mode. The rename dspace operation only changes the dspace name, it does not reorganize the data. It updates the dspace name in all the places where dspace name is stored, such as reserved pages on disk, system catalogs, the ONCONFIG file and in memory data structures. You can also use **onspaces** command to rename the dspace. Here are some restrictions when using the rename dspace feature:

- You cannot rename the critical spaces such as rootdbspace and space containing physical logs and logical logs.
- You cannot rename a dspace with down chunks.
- You cannot rename spaces with **onmonitor** command.

You must take a level 0 archive of the renamed space and root dspace after rename operation. Figure 1-10 shows the example of using onspaces command to rename dspace.

```
sun280-3% onmode -s
This will perform a GRACEFUL SHUTDOWN -
Do you wish to continue (y/n)? y
sun280-3% onspaces -ren dspace1 -n dspace2
Rename of Space completed successfully.

** WARNING ** A level 0 archive of Root DBSpace and the renamed
Space need to be done.
sun280-3% onstat -d

IBM Informix Dynamic Server Version 10.00.UC5      -- Quiescent -- Up 00:08:55 -- 38912 Kbytes

Dbspaces
address number  flags      fchunk  nchunks  pgsize  flags  owner  name
b7317e8  1      0x60001    1        1      2048    N B    informix rootdb
c1134f0  2      0x60001    2        1     16384    N B    informix dspace2
2 active, 2047 maximum
```

Figure 1-10 Renaming dspace using the onspaces command

► Ontape use of STDIO

If you are using ontape to backup and restore data, you can now use standard I/O instead of a tape device or disk file. This feature enables pipe-based remote backups/restores such as might be done in HDR or to a UNIX utility such as CPIO, TAR, or COMPRESS or to disk with a specific filename other than that used by L/TAPEDEV. You can set this by setting the value of configuration parameter TAPEDEV to STDIO. Refer to 9.3.1, “Ontape to STDIO” on page 274 for more detail.

► Multiple Fragments in One Dbspace

You can store the multiple fragments of the same table or index in a single dbspace. Storing multiple table fragments in a single dbspace reduces the number of dbspaces needed for fragmented table and can improve the query performance over storing each fragmented expression in a different dbspace. This way you can also simplify the management of dbspace. The following example shows the commands that create a fragmented table with partitions:

```
CREATE TABLE tab1(a int)
FRAGMENT BY EXPRESSION
PARTITION part1 (a >=10 AND a < 50) IN dbspace1
PARTITION part2 (a >=50 AND a < 100) IN dbspace2
...
;
```

1.3.4 Availability

In this section we give a brief overview of Availability features of IDS V10.

► Table Level Restore

You can restore table data from a level 0 archive to a user-defined point in time. This gives the flexibility to restore specific pieces of data without a need to perform lengthy restore of the entire archive. This feature is also useful in situations where tables need to be moved across server versions or platforms. You can also use the archecker utility to perform table level restore.

► Online Index Build

You can create and drop an index without having an exclusive lock placed on the table during the duration of the index build. This also makes the table available during index build. CREATE INDEX ONLINE and DROP INDEX ONLINE statements can be used to create and drop online indexes. You can use the CREATE INDEX ONLINE even when the reads and updates of the table are occurring. The advantages of creating indexes with CREATE INDEX ONLINE statement are:

- If you need to build new index to improve the performance of the query, you can immediately create it without placing a lock on the table.
- The query optimizer can establish better query plans, because it can update statistics on unlocked tables.
- The database server can build an index while the table is being updated.
- The table is available for the duration of the index build.

The advantages you have by dropping indexes with DROP INDEX ONLINE statement are:

- You can drop the inefficient index without disturbing ongoing queries that are using them.
- When the index is flagged, the query optimizer will not use the index for new SELECT operations on tables.

The ONLIDX_MAXMEM configuration parameter can be used to limit the amount of memory that is allocated to a single pre-image pool and a single updator log pool. The pre-image and updator log pools are shared memory pools that are created when a CREATE INDEX ONLINE statement is executed. The pools are freed after the execution of the statement is complete. You can set ONLIDX_MAXMEM in ONCONFIG file before starting the database server or you can set it dynamically using the **onmode -wm** and **onmode -wf** commands.

An example command to create an index online is:

```
CREATE INDEX cust_idx on customer(zipcode) ONLINE
```

An example command to drop an index online is:

```
DROP INDEX cust_idx ONLINE
```

1.3.5 Enterprise replication

Sites with an IDS Enterprise Edition license can use either or both of the IDS data replication features, High Availability Data Replication (HDR) or Enterprise Replication (ER). In IDS V10 a number of enhancements were made to these features:

► DRAUTO

When using HDR, earlier versions of the database server contained the DRAUTO ONCONFIG parameter which controlled the action of the mirror instance in the event of a replication failure. This was removed in V9, but has been returned in V10. Depending on the value configured for DRAUTO, when the mirror instance determines the primary instance has failed, the mirror can either:

- Remain in read-only mode until manually converted.
- Switch to primary mode and begin processing transactions automatically. When the original primary instance returns, the mirror will send its transactions to the primary then return to mirror mode.
- Switch to primary mode and begin processing transactions automatically. When the original primary instance returns, the mirror remains in primary mode and the original primary acts as the new mirror receiving all updates to make it logical consistent to the new primary.

The DRAUTO parameter should be set very carefully because network failures are treated equally as actual instance failures. Contact an authorized IBM seller to speak with their technical representatives about the implications of setting DRAUTO.

► Replicate Resync

The significant ER feature added in IDS V10 is the ability to re-synchronize the data in ER replicates. Accomplished with the `cdr check [repair]` and `cdr sync` utilities, it is no longer necessary to perform data extracts and loads or database restores to bring tables or instances to consistency. The `cdr check` utility can verify whether or not the data in a replicate instantiated on two nodes is identical. Its output includes the number of dissimilar data values as well as row count problems. With the `repair` option, the utility can automatically correct the errors. As might be expected, the `cdr sync` utility is used to completely refresh one instantiation of a replicate with the data from a master replicate. Depending on the amount of data in a replicate and the number of errors found by `cdr check`, returning the affected replicated to consistency might be faster with `cdr sync`.

► Templates and Master Replicates

When configuring ER, replicates are defined that include what data will be replicated and to which targets. In IDS V10, this process has been simplified with the introduction of templates and master replicates. Master replicates are reference copies of the replicate definitions and can be used to verify the integrity of specific instantiations of that replicate on individual nodes. Using the master, the administrator can determine if the data or target definition on a node has been changed and rebuild it as necessary.

With templates, a large number of replicates can be defined once on one node then realized on any number of additional nodes eliminating the need to manually redefine the same replicates on each one.

► Detecting event alarms with the event alarm program

It is now possible for the `ALARMPROGRAM` utility to capture and use ten ER-specific alarm classes including storage space full, subsystem failure/alert, data sync error and resource allocation conditions. Depending on how the instance administrator configures these alarms, the `ALARMPROGRAM` can automatically take corrective actions or send alerts to operation or administration staff.

1.3.6 APPLICATIONS

In this section, we give a brief overview of applications support of IDS V10:

► JDBC 3.0 support

IDS V10 supports Informix JDBC 3.0 by introducing the following features in compliance with Sun™ Microsystems™ JDBC 3.0 specifications:

- Internally update BLOB and CLOB data types using all methods introduced in the JDBC 3.0 specification.
- Specify and control ResultSet holdability, using the Informix JDBC extension implementation.
- Retrieve auto-generated keys from the database server.
- Access multiple INOUT mode parameters in Dynamic Server through the Callable Statement interface.
- Provide a valid large object descriptor and data to the JDBC client to send or retrieve BINARY data types as OUT parameters.
- J/Foundation supports JRE™ Version 1.4 and the JDBC 3.0 specification.
- Updated software Electronic Licensing.

Refer to 6.1.3, “The IBM Informix JDBC 3.0 driver - CSDK” on page 190. for more detail on JDBC 3.0 support for IDS V10.

► .NET support

The .NET provider enables .NET applications to access and manipulate data in IDS V10. All the applications that can be executed by the Microsoft .NET framework can make use of IBM Informix .NET provider. Here are some examples of applications:

- Visual Basic® .NET applications
- Visual C#® .NET applications
- Visual J#® .NET applications
- ASP .NET Web applications

Refer to 6.1.4, “IBM Informix .NET provider - CSDK” on page 193 for more detail about .NET support.

Fast implementation

There is no denying that the watchwords for businesses today are very similar to the Olympic motto—"Citius, Altius, Fortius" (faster, higher, stronger). Products and services need to be scoped, designed and brought to market before the competition. Marketing messages have to create an immediate impact to drive demand. Costs and inefficiency must be reduced to the bare minimum. A total commitment to the business or product strategy is expected unless the strategy does not appear to be working, then the business (and its employees) must immediately change course, adopt a new strategy and push it forward.

How does that apply to database server operations? It seems administrators are no different and want the shortest and fastest process for getting a server up and running. In this chapter we attempt to do exactly that by discussing topics such as upgrades and migration, design issues, installation and initialization as well as an overview of some of the IDS administration and monitoring utilities.

Understand and know this from the beginning—*there are no magic bullets or short cuts to success in this area*. The topics covered here are too broad to receive more than a cursory overview in this IBM Redbook. Further complicating matters, many of these topics are tightly interconnected yet must be discussed in some sort of a sequential order. So it might seem at times that they are not being discussed in the proper order. That said, best practices in each area will be presented with the understanding that administrators will adapt and tailor these suggestions to their own environments based on business requirements and their own hands-on experience.

2.1 How can I get there from here

A large part of the reason behind creating this IBM Redbook was to help administrators using earlier versions of IDS, other versions of Informix database servers or even other database servers migrate to IDS version 10. It makes sense then to talk about migration and upgrade paths first.

2.1.1 In-place upgrades

One of the wonderful things about administering Informix database servers is that there is always a graceful way to move from one version to another. In most cases the migration occurs “in-place” with the instance being shut down, the new binaries loaded and the instance restarted. Migration completed. The unspoken assumption is that migration and testing occurred first on a test or development server.

Depending on the current version of the database server, moving to IDS version 10 can occur through an in-place upgrade. For other versions or even other Informix servers, an interim step is required, although these steps can also happen in-place. Figure 2-1 shows the recommended upgrade paths which must be followed when moving from an earlier version of IDS. As indicated, almost all the more current versions of IDS can be upgraded directly. The one exception is an IDS version 7.24 environment using Enterprise Replication.

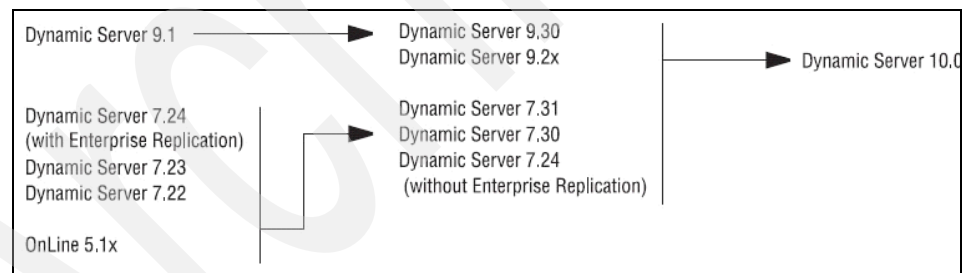


Figure 2-1 In-place upgrade paths to IDS version 10

If the installed version of the database server is somewhat older, it must first be upgraded to an acceptable interim version, preferably the most current in that family such as 7.31 or 9.3/9.4. The reason for this is that between these earlier and more current versions, a large number of structural changes occurred, not all of which are visible to an administrator. These changes need to be made to instance and database structures prior to moving to version 10.

As indicated in Figure 2-1, it is possible to upgrade from OnLine v. 5.1x by performing an incremental upgrade to IDS v.7.31 first. In following this path, the

OnLine administrator would be well served to get as much education as possible on IDS prior to going through the upgrade process. As described in Chapter 1, “IDS essentials” on page 1, the IDS architecture is radically different and, as a result, so is the administration, performance tuning and the day-to-day maintenance activities of the environment, requiring new skills and insight.

Explicit directions for executing an in-place upgrade are contained in the IBM Informix Migration Guide for each version of the instance. One important but undocumented step involves the installation of the new binaries. These should be installed in a different directory than the current \$INFORMIXDIR. Simply change \$INFORMIXDIR and \$PATH to reflect the new location so the correct binaries are used when restarting the instance. This will facilitate easier reversion if necessary. The general process of an in-place upgrade is briefly summarized in Table 2-1.

Table 2-1 The general process for executing an in-place upgrade to IDS v.10

Step	Description
Ensure adequate system resources	There should be sufficient file system space to install the new server binaries, system memory for the instance to use and free space in the rootdbs and other spaces as well as the logical logs for conversion activities.
Test application connectivity and compatibility	Tests should include all applications including storage managers, utilities as well as administrative and other scripts and functions
Prepare a backout plan in case of failure	This is a critically important step. There should be multiple full instance backups as well as other data exports if time and space permit. These backups should be verified with the <i>archecker</i> utility before the upgrade begins. System backups and copies of configuration files should be made as well. NOTE: though the database server contains regression or roll-back functionality, it might be faster (and certainly safer) to restore from backup.
Test, test and more tests	Conduct the upgrade multiple times on a test server using backups to restore the instance to a pre-upgrade state for another attempt. Test applications under as big a user load as can be created
Capture pre-migration snapshots	Using server utilities capture full configuration information about spaces and logs. Use the set explain on SQL statement to capture optimizer plans on the most important operations for comparison after upgrading.

Step	Description
Prepare the instance for the upgrade	Look for and remove any outstanding in-place table alters (not required but safer to do), close all transactions by shutting down the instance then restarting to quiescent and take the appropriate actions if using replication. CREATE ONE OR MORE LEVEL 0 BACKUPS! Verify them with the <i>archecker</i> utility.
Install and configure	Install the new binaries preferably in another directory than the current \$INFORMIXDIR. Copy/modify instance configuration files such as \$SQLHOSTS, \$ONCONFIG and others as needed.
Restart instance and monitor startup	Execute oninit -jv to bring the instance to the new single user mode. Monitor the \$MSGPATH file in real-time for any errors. If the log indicates the “sys” databases (such as sysmaster) are being rebuilt, <i>Do not attempt to connect to the instance until the log indicates that the databases have been rebuilt. Otherwise, the instance will become corrupted.</i>
Perform post-migration activities	Drop then update statistics, create a new level 0 backup, verify data integrity with the <i>oncheck</i> utility, re-enable/restart replication, restore user access

If problems arise after the upgrade, there are at least two options to revert back to the earlier version. You can use the IDS bundled reversion utility, or you can restore the level 0 backup that was created at the beginning of the upgrade process and reset the environment parameters to point to the original \$INFORMIXDIR and related structures.

Both options will work, although there are conditions which preclude using the reversion utility, including whether newly created database tables and attributes are not supported in the earlier version, if a table or index fragmentation scheme has changed, if new constraints or triggers have been created, and other conditions that are listed in the *IBM Informix Migration Guide, G251-2293*. If any of these conditions are true, in-place reversion is not possible.

The process of executing an in-place reversion is somewhat similar to upgrading, remove any outstanding in-place table alters, save copies of files and configuration information, create one or more level 0 backups, disable replication, remove or unregister any new functionality installed but not supported by the earlier version, close all transactions and quiesce the instance. The reversion command is **onmode -b older_version** where *older_version* is replaced with the number of the desired database server version (for example, 7.3). Monitor the \$MSGPATH for all reversion messages, including those about

the “sys” databases. When the process has completed, verify data integrity with the *oncheck* utility, create a level 0 backup then restore user access.

2.1.2 Migrations

The term *migration* in this chapter refers to data movement from an unsupported database server to IDS. The data could originate from an Informix Standard Engine (SE) instance or another server from the IBM, or a competitor, portfolio of products. The migration process is much more labor intensive and requires more careful preparation and planning as well as more time to execute.

The biggest task in migrating is the verification of table structures and attributes. This is also the greatest opportunity for administrators to have a big impact on performance after the migration. Because a migration requires rebuilding the environment, administrators have the opportunity to correct earlier design flaws in the database or organization of the new IDS instance. Tables and attributes can be adjusted, sized correctly and placed in dbspaces in such a way to better utilize the full power of IDS. Careful review and planning in this area can have a very positive impact on the availability, security and functionality of the instance.

In a migration it is not just the data that must be reviewed and moved into the new environment. Stored procedures and other functions built in the original database server must be migrated to IDS in order for applications to continue working properly. While data type conversions and the actual movement of data from the source server to IDS can be automated, converting procedures is usually a manual process requiring effort.

IBM has a tool to assist and even perform some of the work in converting database structures and moving data from a source server to IDS. The *IBM Migration Toolkit (MTK)* is a freely downloadable utility with a Windows and UNIX/Linux port. At the time of the writing of this book, the current URL to access the MTK was:

<http://www-306.ibm.com/software/data/db2/migration/mtk/>

Additional information as well as a quick functionality walkthrough is available through the IBM developerWorks Web site at:

<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0603geib/>

The MTK is a graphical utility guiding the administrator through the five steps migration process described in Table 2-2.

Table 2-2 The five steps of the migration process as managed by the IBM MTK

Step	Description
Specify the source database server	The MTK can either convert an existing dbschema-like file from the source server or, with connectivity properly configured, connect to the source server and extract schema and data definitions
Convert the schema and data definitions into IDS syntax	The source information can be reviewed then converted into IDS-compatible syntax. The results are written to a flat file which can be used by the MTK in a later step or by the administrator with other utilities such as <i>dbaccess</i> to create the database in the new IDS instance. Conversion errors or problems are noted in the file for handling in the next step.
Refine the IDS DDL syntax	The IDS syntax can be reviewed for accuracy. Any conversion errors or problems are highlighted so changes can be made by the administrator using the tool's interface. This is particularly helpful if the source attribute could be converted into more than one IDS datatype.
Produce data migration scripts	Extraction and load commands are generated based on the relationships between source and IDS tables and attributes. With MTK v.1.4 these scripts can only be used within the context of the MTK. New functionality to remove this constraint is planned for the next release of the MTK.
Build the new IDS environment and populate the tables	With connectivity properly configured, the MTK connects to the IDS instance and builds the databases from the syntax file. When the database is built, the MTK connects to the source server, extracts the data and loads it into the appropriate IDS tables. This requires either a JDBC interface to the IDS instance or be executed on a physical server which can connect to the target IDS instance using the <i>dbaccess</i> utility. A report is generated after the load process completes.

At the time of the writing of this book, the MTK was at version 1.4. This version fully supports schema and data migration from Oracle® servers as well as most Sybase servers. This version seamlessly handles migration of the Oracle timestamp data type to an IDS datetime data type with a precision of 5 places. It ignores any “with timezone” or “with local timezone” syntax in the Oracle definition. For Sybase migrations, only JDBC data migration connectivity is supported in this release.

Data migration utilities

Use of the MTK to move data is not required, IDS provides several data movement utilities which can be used to unload and load data. Some only work

within IDS environments, others produce or use plain ASCII files and can load files created by a non-IDS unload process. When using ASCII files, the default IDS delimiter is the “|” or “pipe” character. It is strongly recommended to use this character because it significantly reduces the chance for conflict with text inside the load/unload file.

SQL load and unload commands

The slowest and least robust in terms of functionality, is an SQL statement. It is used to identify attributes and conditions for extracting data, converting it to ASCII and (usually) populating a named flat file. Conversely, the contents of an ASCII file, whether created from an IDS instance or a competitive server, can be read and inserted into an IDS database based on an attribute list contained in the SQL statement.

Unlike the unload process which can select from multiple tables if desired, the load process can only insert into a single table. When loading, a 1:1 match of table attributes to data file elements is not required; in the load statement file elements can be mapped to specific table attributes. Obviously, any table attribute not mapped should not have a “not null” constraint or the insert process will fail.

These operations do not require an exclusive lock on the tables involved though one should be created if a static image of the source table is required during an unload or to minimize the risk of a long transaction condition when loading a large amount of data into a logged database/table.

The dbexport and dbimport utilities

Here we discuss a very commonly used set of utilities particularly for moving an entire IDS database to an instance on a non-compatible OS. *Dbexport* automatically creates a set of ASCII unload files, one for each table, as well as the complete schema definition for the database. With the **-ss** flag, it will include table fragmentation rules and extent sizing in the DDL definition file.

When exporting, an exclusive lock is required on the database in order to ensure logical consistency between tables with referential integrity constraints. With the appropriate flags, the unload files can either be created on disk or output to tape. When output to tape, the database DDL file is still created on disk so it can be edited if necessary.

The database DDL file is created in the same directory as the data unload files if unloading to disk. The file naming convention of *database_name.sql* should not be changed because *dbimport* matches the file name to the database name specified when the utility is invoked. The DDL file looks very similar to that created by the *dbschema* utility though it also includes load file information as well as load triggering control characters. This file can be edited to change

fragmentation rules or extent sizing if necessary. Attribute names can be changed as well as data types provided the new types do not conflict with the type unloaded. New constraint, index or stored procedures/UDRs can be added in the appropriate places in the DDL file.

When *dbimport* is invoked, the target database is created, then, using the DDL file, each table is created, loaded with data, followed by index, constraint or stored procedures/UDR creation. By default, the database is created in a non-logged mode to prevent a long transaction from occurring during data loads. This can be overridden though the administrator should remember that the database creation, all table loads as well as index, constraint and stored procedure/UDR creation, occurs within a **single** transaction. After the database is created and loaded, it can be converted to the desired logging mode with the appropriate *ontape* or *ondblog* command. With the **-d dbspace** flag, **dbimport** will create the database in the dbspace listed rather than in the rootdbs. When importing, the **-c** flag can be used to “continue” the load process though non-fatal errors occur.

Because these utilities translate to and from ASCII, they are relatively slow from a performance perspective. It is nice however to have a partially editable and completely transportable database capture.

The dbload utility

This utility uses ASCII data files as well as a control file to load specific pre-existing tables within a database. Because the loads can be occurring within a logged database, the control file parameters give this utility more flexibility.

These parameters include:

-l	The fully pathed file name in which errors are logged
-e number	The number of data load errors which can occur before the utility aborts processing. The load errors are logged if the -l flag is used.
-i number	The number of rows to skip in the data file before beginning to load the remaining rows. This is particularly useful for restarting a failed load.
-k	Locks the target tables in exclusive mode during the data load
-n number	The commit interval in number of rows. When loading into a logged database, this flag can be used to minimize the risk of a long transaction.
-s	Perform a syntax check of the control file without loading data. Very useful for catching typing errors when creating or editing the control file.

The control file maps elements from one or more data files into one or more table attributes within the database. The control file contains only **file** and **insert** statements with the first listing input data files and the data element to attribute maps. Each **insert** statement names a table to receive the data as well as how the data described in the **file** statement is placed into the table. Several small control files are illustrated in Example 2-1.

Example 2-1 Dbload control file examples

```
FILE #1:
file stock.unl delimiter '|' 6;
insert into stock;
file customer.unl delimiter '|' 10;
insert into customer;
file manufact.unl delimiter '|' 3;
insert into manufact;

FILE #2:
file stock.unl delimiter '|' 6;
insert into new_stock (col1, col2, col3, col5, col6)
    values
(f01, f03, f02, f05, 'autographed');

FILE #3:
file cust_loc_data
(city 1-15,
 state 16-17,
 area_cd 23-25 NULL = 'xxx',
 phone 23-34 NULL = 'xxx-xxx-xxxx',
 zip 18-22,
 state_area 16-17 : 23-25);
insert into cust_address (col1, col3, col4)
    values
(city, state, zip);
insert into cust_sort values(area_cd, zip);
```

In the first and second control file examples, the load files use a “|” (pipe) separator between data elements. In the first control file, there is a 1:1 match between data elements and the specified target table attributes so a direct insert is requested. In the second control file, the data file contains 6 elements but only 5 table attributes will be loaded. Of the 5 attributes to load, the last will receive a constant. Finally in the last control file, the data files do not use a delimiter symbol so the data elements are mapped to a control file “variable” through their position within the text string. For example, the first 15 characters are mapped

into the “city” variable. In addition, this control file specifies two tables are to be loaded from one row of input data.

Dbload supports almost all of the extensible data types. Nesting of types is not supported however.

The onunload and onload utilities

These two utilities function similarly to *dbexport/dbimport* with a major caveat. Whereas *dbimport/dbexport* worked with ASCII files, these utilities use data in the native IDS binary format. As a result, data extraction and reloading is significantly faster.

Because the utilities use data in binary form, they can only be used when moving between physical servers using the same operating system and the exact same version of IDS. Another difference is that a DDL file is not created, the table and attribute definitions as they exist in the source database are used as defaults when recreating the tables. Some of the definitions can be overridden with flags when the *onload* utility is invoked including changing the dbspace in which a table is created or an index’s fragmentation rule and renaming an index or constraint.

The High Performance Loader

The High Performance Loader (HPL) is a database server utility that allows efficient and fast loading and unloading of large quantities of data. The HPL supports exchanging data with tapes, data files and programs, and converts data from these sources into a format compatible with an IDS database. Likewise, extracted data can be published to tape, disk or other targets and converted into several data formats including ASCII, IDS binary or EBCDIC. The HPL also supports data manipulation and filtering during load and unload operations.

The HPL is actually a collection of four components:

ipload	An X-based graphical utility that you can use to define all aspects of an HPL project.
onpload	The HPL data migration executable.
onpladm	A command line interface that you can use to define all aspects of an HPL project.
onpload	A small database stored in the rootdbs that contains all project definitions.

The HPL is the fastest of the data migration utilities because its operations can be parallelized to simultaneously read or write to/from multiple devices. Setting up and using the HPL requires more work though, often at a table by table level. There is a separate configuration file in \$INFORMIXDIR/etc for the HPL where

parameters such as the number of converter threads, AIO buffers and buffer size and so on are set.

The HPL connects to an IDS instance through its network-based instance name/alias and allocates threads for each device defined for the project. As a result, it is important the network-based connection protocol be tuned to support the additional overhead of HPL operations. Unload projects do not lock the source tables in exclusive mode; however, load projects can execute in one of two modes:

- ▶ Express: target table indexes, triggers, and constraints are disabled and data is loaded using “light appends;” no records are written to the logical logs. The table is locked in exclusive mode and when the project is completed a level 0 backup is required to facilitate instance recovery.
- ▶ Deluxe: target table indexes, triggers and constraints are active and the table is accessible by other users. The inserts are logged but execute as a single transaction so a commit point needs to be set in the project definition to prevent a long transaction from occurring.

As alluded to, HPL operations are defined and executed as “projects.” Each project contains definitions for devices to read from or write to, the input and output data formats, any filters for excluding data, the actual SQL operations to execute as well as maps for describing attribute to data element correspondence. Project definitions can be created and maintained graphically through *ipload*, *ServerStudio JE (SSJE)*, or the *Informix Server Administrator (ISA) utilities*, as illustrated in Figure 2-2, Figure 2-3, and Figure 2-4, or from the command line with the *onpladm* interface.

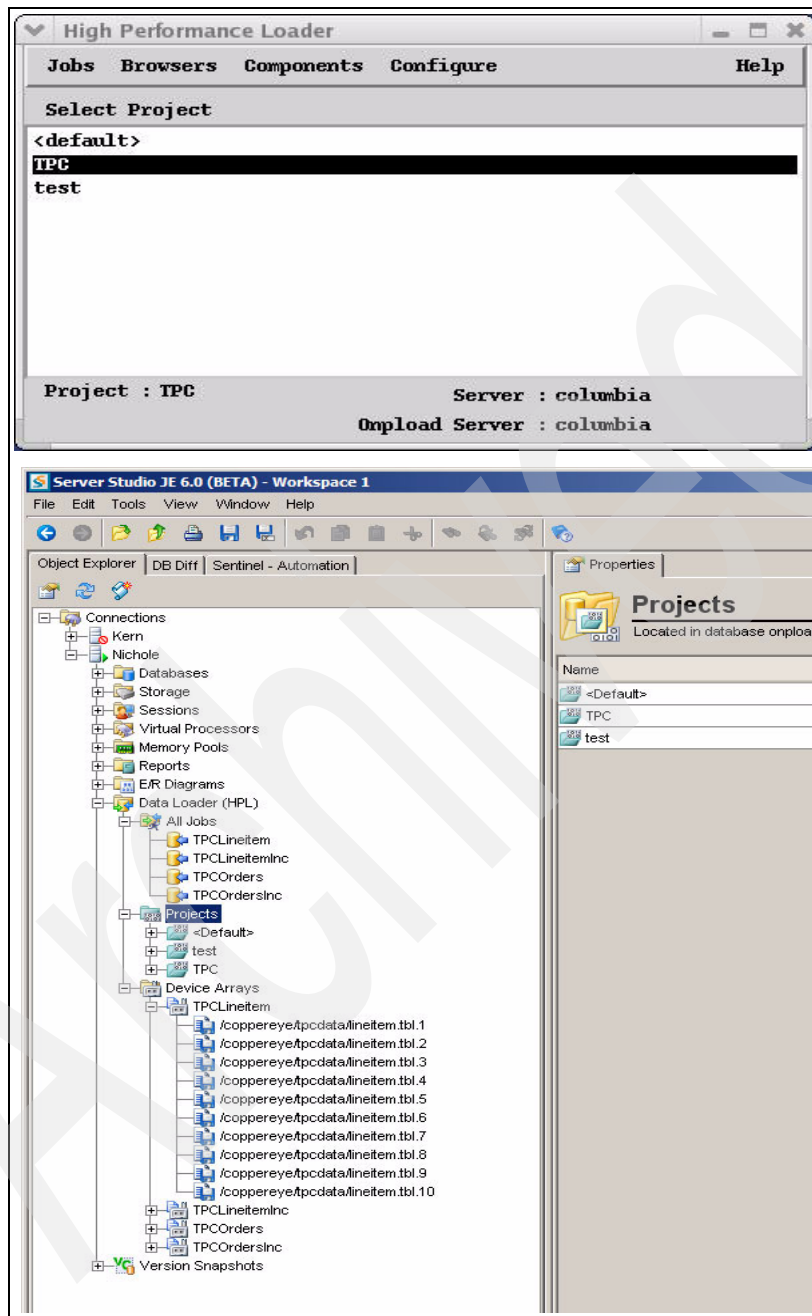


Figure 2-2 The ipload and SSJE interface for viewing projects

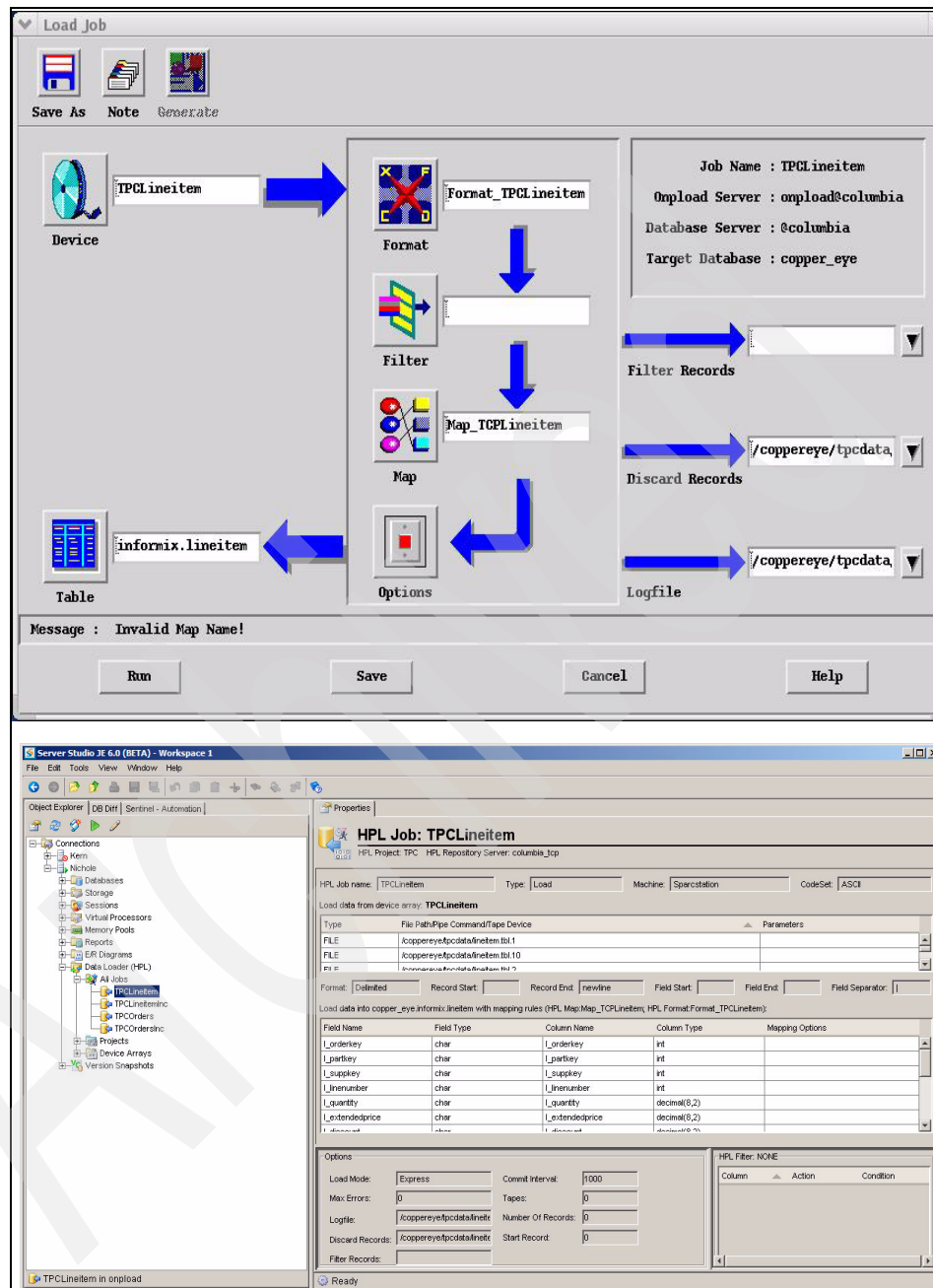


Figure 2-3 The ipload and SSJE load project flow interfaces

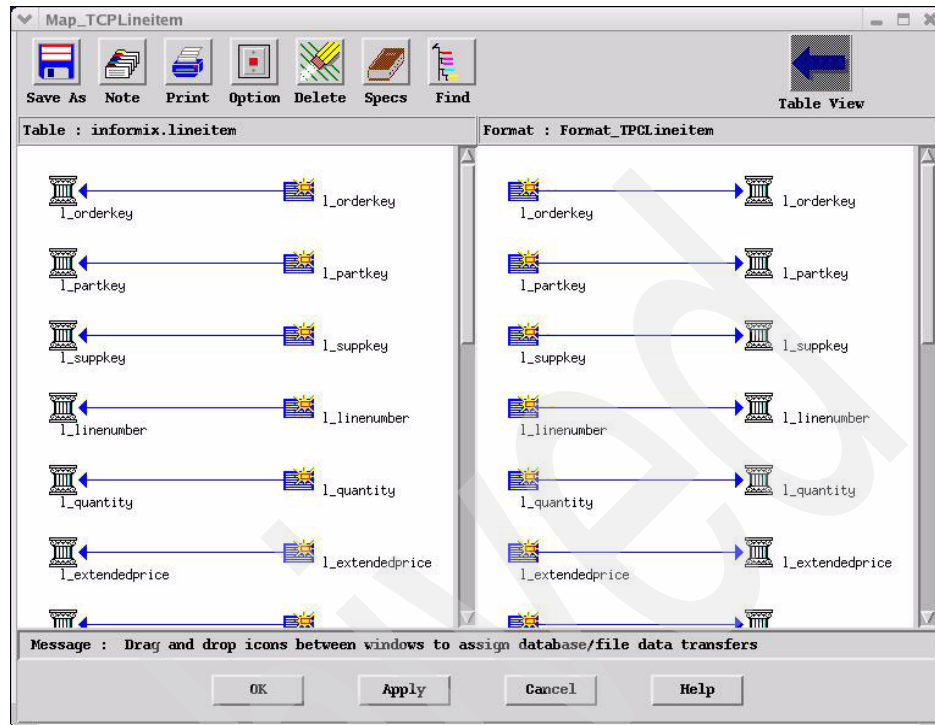


Figure 2-4 The ipload attribute to data element mapping interface

The HPL supports reading and writing to IDS binary, ASCII (fixed length and delimited) and EBCDIC formats. While the utility can load and unload huge amounts of data quickly, project performance is highly dependent on the number of devices configured, whether and what kind of data type conversion is required and the project "mode" if it is a load project. Executing HPL projects has a significant impact on instance resources so careful monitoring and tuning is advised. The *IBM Informix High-Performance Loader User's Guide*, G251-2286, includes tuning and other resource guidelines for executing HPL projects.

2.2 Things to think about first

There is a commonly used phrase to the effect that "an ounce of prevention is better than a pound of cure." This is certainly true for database environments where when built and in operation, it might be nearly impossible to "cure" problems that arise. As a result, it is critical that sufficient time and effort be spent in the planning and evaluation of the database design, the logical and physical instance implementation, backup, recovery and data retention/purge

requirements as well as the physical server resources available to the instance. This phase is similar to laying the foundation when building a house; if done in a slipshod manner or it is incomplete, the structure will fail. In a book of this type, it is impossible to cover all the best practices or inter-relationships between various aspects of data processing environments but this section highlights some of the most important design considerations and decisions that need to be made prior to initializing or migrating an IDS environment. In most cases, these factors are interactive and cyclical with changes in one affecting one or more of the others. Because there is never an unlimited amount of time or money to create a perfect environment, this process ultimately becomes one of evaluating trade-offs and making compromises. The administrator's skill and experience to correctly analyze and make sound judgements is as big a factor in the project's success as the number of physical CPUs or other factors considered in this section.

An important fact to remember when creating the overall design is that any given project will operate for much longer, and its data store will be bigger, than described in the project scope document. As a result, wise administrators account for both of these aspects when creating the instance and database designs.

2.2.1 Physical server components

In the Chapter 1, "IDS essentials" on page 1, the Dynamic Scalable Architecture (DSA) was introduced featuring its triad of components: CPU, memory, and disk. With DSA, Informix Dynamic Server is the most efficient and powerful database server available today. When designing an instance, an administrator needs to consider the amount of physical server resources and the expected workload. While generally true that "if some is good, more is better," proper design is not just throwing everything the server has into the instance but the best balance and use of the resources to meet the functional specification. This might require the administrator trying to change the hardware purchase to get the correct balance of components.

As a couple of examples, if the application will be heavily I/O dependent, rather than spend money for CPUs consider increasing the number of I/O channels and disks that can be directly addressed. With more channels and directly addressable disks, tables and indexes can be fragmented across more dbspaces enabling finer grained access routes. If the application will use a lot of data but most of it will be referenced continually by most users, invest in more memory in order to increase the number of BUFFERS. With more BUFFER memory, more data can be held in the high-priority section of the buffer pool eliminating the time and cost of doing a physical I/O operation. Finally, if the application user count will be high but the workload balanced, invest in more CPUs so more threads can be forked to support more concurrent activities.

The disk component

Of the three physical server components, disks and disk access has the greatest impact on instance performance and will mostly be the greatest source of conflict when working with other members of the project design team. IDS parallelism can only work properly when the instance design can be carried down to discrete and independent disk resources. In today's disk drive environments, most disk administrators only create a couple of "logical units" (LUNs) and put all disks into these LUNs using some RAID level. While RAID has its place and can be used to provide protection against disk failure or potentially faster data access, incorrect RAID use can dramatically impact a database environment.

RAID-level specifications are slightly vague, so some differences exist in how any given RAID level is implemented from vendor to vendor. Although seven RAID levels exist, most vendors have products in levels 0, 1, 2, 3, 5, and 6. Most vendors also have a pseudo level called "0+1" or "10", which is a combination of levels 0 and 1.

Briefly, RAID level 0 is the stripping of data across portions of several disks with the "stripe set" appearing to the OS and the instance as one physical disk. The strategy of this level is "divide and conquer." When a read call is executed against the stripe set, all devices are activated and searched for the requested data. With a small subset of the total data, each device should be able to complete its search quicker either returning the requested data or not. Unfortunately, this strength is also its greatest weakness. That is, there is not any intelligence to where the data could be. For all intents and purposes, it is written in round robin mode. Whenever a small request comes in, all devices must be active though only one will have the requested data. The consequence is a serialization of I/O operations. Just as important, there is no protection against disk failure. If one disk fails, the data it contained is lost.

RAID level 1 is the creation of at least one mirror for each primary disk allocation. When a write operation is executed, the write occurs on the primary and mirror segments. Some vendors manage the mirror writes synchronously, others write asynchronously; it is important to understand which is used by the vendor because it affects LUN performance. Some vendors also support "mirror spares" or the ability of defining one or more disks as hot stand-by devices to immediately fill in and replace a failed disk in a mirror pair so there is constantly an $N + 1$ setup even in a failure condition. Some vendors also provide the ability to create 2 sets of mirrors. The second set can be deactivated at will for some non-Informix database servers to create a full backup. IDS can use this second set for an external backup as described in Chapter 9, "Legendary backup and restore" on page 257. As with RAID level 0, its greatest strength is also its weakness. While the mirror set provides excellent protection against disk failure, these mirror sets appear to the OS and instance as a single LUN and with

(usually) only one disk, it takes longer to perform a complete scan to find the requested data.

RAID level “10” or “0+1” is a multi-disk stripe set that is also mirrored to one or more sets of disks. With this level, there is excellent protection against failure plus a smaller set of data on the disks to search. Once again, this set appears as a single LUN to the instance and there is not any intelligence to how data is striped on disk.

RAID levels 5 and 6 are very similar in design and implementation; data is stored in up to 75 percent of the disk space with the remaining 25 percent used to store what are called “parity bits.” These parity bits allow for the reconstruction of data on other drives should the data become unavailable on the primary disk.

In RAID level 5, each disk holds its own parity bits, while in level 6 each disk stores parity information for other disks. The I/O throughput is very poor because of the disk contention involved in writing the original data as well as the parity information to one or more other disks. In the event of a disk failure, reading from parity information is pathetically slow.

For most database applications if a RAID solution must be used, RAID levels 0 and 1 (and the hybrid) are the most appropriate because of their relative speed when compared to the other RAID levels. If disk performance is *not* an issue but total disk usage is, RAID level 5 could be used.

RAID is not the only option to solve data protection and access concerns though. IDS provides mirroring, striping and intelligent data fragmentation technology providing distinct technical and performance advantages over RAID.

IDS mirroring operates similarly to RAID 1 in that chunks can be mirrored so write operations execute against the primary and mirror chunks. The writes occur synchronously ensuring consistency between the primary and mirror chunks. The real benefit to IDS mirroring is that because the mirror is defined within the instance, the IDS query optimizer is aware of the mirror chunks and uses them when building access plans. Consequently, the optimizer will execute query operations against the mirror chunks while executing write operations against the primary chunks. If there are several query operations, one will execute against the primary while another is executed using the mirror chunks. The net result can be a doubling of the I/O throughput.

IDS also provides data striping or *fragmentation* functionality in two forms:

- ▶ Round-robin fragmentation similar to RAID level 0
- ▶ Intelligent placement on disk based on attribute value

IDS round-robin fragmentation is exactly like, and offers the same benefits and risks as, RAID 0. If this functionality and risk is acceptable, it does not matter

which technology is used to stripe the data to disk though it might require fewer keystrokes to use RAID level 0.

Where IDS fragmentation functionality is strongest is with the intelligent placement of data on disk based on attribute values. For example, suppose a table had an important attribute which was always included in the conditional section of SQL operations and whose values ranged from 1 to 500, the table could be fragmented with values 1 to 100 on one disk, 101 to 200 on another disk, 201 to 300 on a third disk and so on. Because the IDS optimizer knows about the fragmentation rules, they are used when creating and executing an access plan. For any operation where that attribute is listed as a condition, the optimizer knows which disk contains the data and the I/O thread is directed to just that device. I/O throughput increases as multiple operations can be executed simultaneously against different fragments of the same table. I/O throughput is increased again if IDS mirroring is used with these fragments.

When deciding on the instance physical implementation, the best solution is one that maximizes the number of directly addressable disk chunks either through direct connections or LUNs. Ideally, these discrete devices would then be used to create mirrored fragments within the instance. This is where contention with the system or disk administrator will occur. That administrator will want to minimize the configuration work they do for the database environment. The disk vendor will say that the disk farm has its own cache and will pre-read data so there is no need for intelligent data placement—as though the disk farm was smart enough to know which data was going to be asked for next! In either case, the recommendation will be to create one large RAID set and let the disk farm handle the I/O. For single threaded database servers which can only do one thing at a time, that approach might be acceptable but it is a significant stumbling block to IDS threaded, multiple concurrent operation design. If the instance only sees one device, be it a LUN or one very large disk, it will only use one I/O thread and operations will become serialized. Instance operations will be throttled by disk latency and I/O bus speed.

Whether IDS disk technology is used or not, there is one golden rule which must be followed: the drives on which the rootdbs, the logical logs, and the physical log reside **MUST** be protected against failure. An IDS instance can sustain a loss of other spaces with their tables/indexes and continue operations but if the rootdbs or the physical/logical logs are affected, the instance shuts down to protect data integrity.

The CPU component

The threaded DSA model brings an interesting wrinkle to this aspect of server planning. With non-threaded database servers the only way to achieve performance scalability is to increase the speed of the CPUs; this is an illusion though because the database server did not scale at all, some of the operations

just completed more quickly. With IDS, having a fewer number of faster/more powerful CPUs might decrease instance throughput. Performance might actually increase with more slower/less powerful processors!

The key to understanding how this works is in the types of operations the instance is executing. If the instance is only doing simple or common work such as standard I/O operations and regular user/application interactions, IDS threads are highly optimized for these operations and nothing special is required from the hardware. The instance will benefit from a larger number of slower processors and a corresponding increase in the number of CPU VPs. If however, the instance is executing a large number of Java UDRs, complex stored procedures or other server-side routines including DataBlade processing, more powerful processors would help execute these routines quicker. This does not mean that slower processors cannot support UDR and DataBlade operations, they certainly can. It is simply that they will take longer to execute the required mathematical procedures these operations require. Generally speaking, if the workload will be mixed, an environment will be better served with more, less expensive processors.

What is interesting in this discussion is the emergence of multi-core processors. With a multi-core processor, two slower processing units are combined as one physical unit. Vendors and customers benefit from reduced cooling requirements (slower processors do not get as hot) as well as some processing performance improvements from two slower processors doing more work than one more powerful (and hotter) processor. As this time, research is still being conducted to see how best to configure IDS instances on multi-core processors.

The memory component

When considering memory, there needs to be sufficient for instance, OS, and any other application operations. With IDS' newer buffer management system, the database server uses memory much more efficiently for satisfying user operations. Nevertheless, the amount of memory and how it is configured requires tuning and monitoring. This is one area where more is not necessarily better because it will have an impact on checkpoint operations as all the buffers must be flushed.

As a starting point, consider how often data will be re-used as opposed to user operations requiring unique elements as well the number of locks that might be required. Where data is re-used quite often, it needs to stay in the high priority area of the LRU queue. Create fewer queues with more buffers so a proportionally larger number of buffers are in this high priority area. Conversely, where data churn is common, a larger number of queues, perhaps with fewer buffers, will ensure data flows more quickly out of the LRU queues.

In version 10, the LOCKS \$ONCONFIG parameter is now a baseline configuration and if the instance requires more locks for processing it will dynamically allocate up to 15 batches of additional locks. These additional locks are maintained in the virtual portion of shared memory. A new configuration parameter, DS_NONPDQ_QUERY_MEM, will allocate space in the virtual portion of memory to handle **sort** and **join** SQL operations in-memory as opposed to in temporary dbspaces. Other functional enhancements also occur within or are controlled through structures in the virtual portion of instance memory so it is important SHMVIRTSIZE is configured sufficiently for the expected worst-case scenario of memory usage. If more is actually required, an additional segment can be allocated (depending on the SHMTOTAL and SHMADD \$ONCONFIG parameter values) but allocating and deallocating memory should be minimized as much as possible.

2.2.2 Instance and database design

The logical and physical design of a database and how it is implemented on disk will have a greater effect on database server throughput than almost any instance tunable parameter. Perhaps the biggest factor is minimizing disk I/O contention as well as the number of accesses to operate on or with data as much as possible. This is driven primarily by the database design.

Before creating the database, a data model and modeling theory must be selected. A modeling theory refers to how a database administrator will choose to look at relationships between data elements. Because of the advanced IDS architecture, it supports three primary modeling theories.

► Relational

This data model typifies database design most commonly used for online transaction processing (OLTP) purposes. The primary design element is small, discrete, and unique data elements that are stored in separate tables. The disadvantage to this model is that a significantly larger number of I/O operations are required to operate on data because each SQL **join** operation that retrieves a data element requires an I/O operation. The physical design of the instance (where tables are placed on disk) is of paramount importance in order to eliminate as much I/O contention as possible and, with DSA's parallelism, to maximize the number of I/O threads that feed data in real-time to other threads processing an I/O operation.

The other significant disadvantage to relational theory is that the logical database design has no resemblance to how the business thinks about or wants to use data. For example, data that is related to each other such as an order requires the basic order information to be stored in one table and components of the order such as the number of each product ordered, their prices and so on be stored in at least one (or more) other tables. This requires

the application developer to find where all the data elements for an order reside, join them properly, operate as needed on them, then write them back into all the various tables. This requires a lot of work inside the application as well as I/O operations inside the database server.

► Object-relational

Object-relational (O/R) models employ basic relational design principles, but include features such as extended data types, user-defined routines, user-defined casts and user-defined aggregates to extend the functionality of relational databases. Extending a database simply refers to using additional datatypes to store data in the database as well as a greater flexibility in the organizational and access to the data. The disadvantage to using an O/R model is that it requires thinking about data differently—the way a business uses it as opposed to breaking it up into little pieces. It also requires greater skills on the part of the designer to really understand how data is used by the business. The advantages are numerous and include a simplified design, decreased I/O requirements and greater data integrity. For example, using the order example from the relational model, in an O/R model, order components can be nested inside each other in a number of different ways so that everything is available in one row. Application development is simplified because order data exists in the database as it does in the real world—as a single object with small yet inherited parts.

► Dimensional

This data model is typically used to build data marts optimized for data retrieval and analysis. This type of informational processing is known as online analytical processing (OLAP) or decision-support processing. With this model, a very few centralized fact tables containing transactions are surrounded by dimension tables which contain descriptions of elements in the fact table. Common examples of dimensional tables include time (such as what is a week, month, or quarter?), geography (such as what stores are in district X and region Y?) and so on. In a dimensional model, I/O from the fact table is filtered primarily by joining with elements from the dimension tables. Because the number of rows in the fact tables are usually quite large, I/O operations are usually quite large and require more time to complete than operations for a day-to-day transactional database.

Each of these modeling theories has strengths and weakness, but with IDS a database designer has the ability to mix and match all three into a single database as needed to solve business needs. The context of this book prevents a more in-depth explanation of each design theory. The *IBM Informix Database Design and Implementation Guide*, G251-2271, would be a good first place for database designers to become better educated on each theory and how to use them in building a more efficient database.

As a database designer begins to build the logical model of all required tables, constraints and indexes, the designer needs to get familiar with the data and how it is used. The designer needs to understand what applications will use the data and how. Will they be read, insert or update intensive? Will the number of some data elements grow more rapidly than others and why? How long must data be retained and what is the purge expectation in terms of time, timing and concurrent access to remaining data during a purge? What is the acceptable length of time for a data access operation to complete? Based on answers to these questions and many more, the types of tables to create (strictly relational, O/R or dimensional) and operational rules for the tables (constraints or relationships to data in other tables) will become apparent.

When looking at dbspace usage, as access requirements are refined, how tables should be placed within dbspaces and where the dbspaces are created on disk will become apparent. It is in this process that the ability to control to the lowest level possible the creation and placement of LUNs mentioned in “The disk component” on page 52, becomes critical. The best and most I/O efficient database design in the world will fail if the dbspaces are all placed in a single RAID 5 LUN. Likewise, being able to create dbspaces on individual disks will be wasted if the designer creates all the most frequently used tables in one dbspace.

The process of disk and dbspace design is two parts science and three parts trial and error. Generally speaking a wise designer will divide tables and indexes in such a way to maximize IDS’ ability to execute parallel operations. Tables will be fragmented into one or more dbspaces so the optimizer can perform fragment elimination. Indexes and index-based constraints should be separated into a number of other spaces so that as index finds are executing, parallel table reads can occur. As much as possible, these dbspaces should be placed on different LUNs using different drives to minimize device contention.

In the past Informix engineering recommended a 1:1 correspondence of disk chunks to dbspaces. Some engineers even recommended rather than loading up a dbspace with a bunch of tables or fragments of tables, only putting a couple of smaller whole tables or a fragment of one larger table in a dbspace. At first look this might appear a bit absurd, but with the added flexibility it brings to the backup/restore process as well as the DATASKIP parameter, this makes sense.

As with the sizing of tables, the same growth factors need to be applied to dbspace sizing. Do not skimp on the size of the chunks and related spaces. It is always better to have more rather than less space. Trying to retroactively fix a “too small” environment is always more expensive than the hardware to build it right the first time.

2.2.3 Backup and recovery considerations

Data access speed is not the only factor which should be considered when designing the physical implementation of tables and their associated dbspaces. The ability to easily and quickly recover from a user or system failure is just as important; more so if hardware or IDS-based disk redundancy is not used.

In establishing a backup strategy, a number of factors can and should influence its creation. Among these are:

- ▶ Is it critical to be able to restore to any specific moment in time?
- ▶ What granularity of restore could be required?
- ▶ Are some tables more important to capture regularly than others? Would these have to be restored first in the event of whole system failure?
- ▶ How much data is there to back up and possibly restore in the instance?
- ▶ How easily could lost data be re-created?
- ▶ How much time is available to create a backup if the backup process were required to occur in a quiet or maintenance period?
- ▶ What and how many physical devices are available on which to create the backup? How fast are they?
- ▶ What are the retention and recycle periods for the tapes used to back up the instance?

The first step is to determine how much data loss, if any, is acceptable. There are several different types of data loss scenarios and each can require a different recovery plan. These scenarios include

- ▶ The deletion of rows, columns, tables, databases, chunks, storage spaces, or logical logs
- ▶ Intentional or accidental data corruption or incorrect data creation
- ▶ Hardware failure such as a disk that contains chunk files fails or a backup tape that wears out
- ▶ Database server failure
- ▶ Natural disaster which compromises either equipment or the operational site or just interrupts power and network access.

The next step is determining what the most logical recovery process would be based on the extent of the failure. One possible recovery plan is shown in Table 2-3.

Table 2-3 Potential recovery options based on data loss

Severity	Data Lost	Potential Recovery Plan
Small	Noncritical	Restore data when convenient using a “warm” restore.
Medium	Business critical but data was not in instance “critical” dbspaces	Use a warm restore to restore only the affected spaces immediately. Alternatively use the Table-Level Point in Time Restore feature
Large	Critical tables and dbspaces are lost	Use a mixed restore to recover critical spaces first followed by a warm restore to recover remaining spaces
Disaster	All	A cold restore to recover the most important spaces first followed by a warm restore to recover remaining spaces. Alternatively, convert the HDR secondary to “primary mode” and continue processing using that instance copy.

See Chapter 9, “Legendary backup and restore” on page 257 for a detailed explanation of cold and warm restores, as well as the Table-Level Point in Time Restore functionality. The chapter also discusses the backup and restore functionality offered by the two IDS utilities—*ontape* and the *OnBar* utility suite.

While both *ontape* and *OnBar* support warm restores of specific dbspaces, only *OnBar* supports backing up specific spaces. From a physical design perspective, this means an administrator can place more static tables in one or more dbspaces which get backed up less frequently. More active tables can be fragmented into other spaces which are backed up daily. With DATASKIP enabled and tables fragmented using expressions, even if a portion of the table is unavailable, instance operations can continue while either a warm or table-level restore is executed.

DATASKIP is the \$ONCONFIG parameter which controls how an instance reacts when a dbspace or portion thereof is down. The DATASKIP parameter can be set to **off**, **all**, or **on *dbspace_list***. **Off** means that the database server does not skip any fragments. If a fragment is unavailable, the query returns an error. **All** indicates that any unavailable fragment is skipped. **On *dbspace_list*** instructs the database server to skip any fragments that are located in the specified dbspaces.

This feature, while helpful from a business continuation perspective, should be used cautiously because it can compromise data integrity if user activities

continue for a period of time while spaces are unavailable. See the *IBM Informix Dynamic Server Administrator's Guide*, G251-2267, for more information about using DATASKIP.

As discussed in Chapter 9, “Legendary backup and restore” on page 257, if the administrator chooses to look at backup and restores at a whole-instance level, each backup utility supports three levels of backup granularity as well as administrator-driven or automatic backup of the logical logs. Logical logs backups are required for restoring to a specific moment in time or to the last committed transaction prior to a system failure.

Generally speaking, a whole-instance perspective to backup and recovery is most appropriate when data throughout the database changes on a regular basis. Full as well as incremental change backups are created on a predefined basis. Recovery (depending on the backup utility used) involves replaying the backups which make up the logical set in order to restore to a specific moment in time. If most data change occurs in a known subset of database tables, these tables should be placed / fragmented into spaces which do not contain the more static tables. The *OnBar* utility suite can then be used to execute regular backups of just these spaces with an occasional full backup to capture all spaces.

One other option to consider is that depending on instance use, it might be faster to recreate or reload data than to restore from backup. This only works in situations where the IDS instance is used for analytical rather than transactional activities. In this type of situation, an occasional full backup is created and instance recovery is achieved by restoring this backup then re-loading the source data with the same mechanism used to load the data originally. The assumption here is that the source files will be protected from loss until the full backup is created so the data is fully recoverable.

Finally, if the logical logs are not backed up as they fill, or are only backed up infrequently, the administrator should assume recovery will only be possible to the last logical set of a full and potentially 2 incremental backups. In some cases this might be the only option available. For example, if the instance is embedded inside an application being sold to customers without on-site DBAs to manage a full backup and recovery mechanism, creating a single full instance backup on a regular basis is usually the only available option. Recovery is only possible to the moment in time in which the backup was started. The application infrastructure should be designed to protect the backup files either by copying them to a different drive in the server or requiring the customer to either copy the files themselves or provide a network-accessible storage device for the application to copy the files to.

2.3 Installation and initialization

The *IBM Informix Dynamic Server Installation Guide for UNIX and Linux*, G251-2777, as well as the *IBM Informix Dynamic Server Installation Guide for Microsoft Windows*, G251-2776, are both well written, step-wise instructions on how to install and initialize an IDS instance for the first time. They, and this book are not intended to discuss and evaluate every single \$ONCONFIG parameter and how it might be configured. This section will briefly discuss the steps involved in the installation and initialization of an instance.

The first step is to review the machine-specific notes for the IDS port. These are found in the **release** subdirectory of the binary distribution. For UNIX and some Linux ports, to access this information the IDS binaries need to be un-tarred to a directory. With Windows distributions, these notes should be available directly from the installation media. Within the machine notes are recommendations for tuning the operating system to support IDS instances as well as any required OS patches that should be applied prior to installing IDS. The notes will also list any OS specific limitations such as largest addressable memory allocations in order to correctly size an instance request for Resident and Virtual shared memory.

The next step is to create the required user and group identifiers. During a Windows installation, this occurs automatically but with UNIX/Linux ports, an *informix* user ID and group must be created with the *informix* UID a member of the “informix” group. The “informix” UID is used to manage and maintain instances and databases so this password should be protected and only given to those who really need it. No general user access should ever use the “informix” UID.

Several environment variables must be set prior to installing the binaries. Of these, the most important is INFORMIXDIR or the directory into which the IDS binaries will be installed. The **bin** subdirectory of \$INFORMIXDIR must be included in the executable PATH for each UID accessing instances. This can either be set in each user's individual profiles or globally in the /etc/profile as shown in Example 2-2.

Example 2-2 Setting IDS environment variables globally in the /etc/profile

```
#### begin IDS-oriented environment variables
## last update: October 10, 2006 by Carlton

INFORMIXDIR=/opt/informix/10
export INFORMIXDIR

export PATH=$INFORMIXDIR/bin:$PATH

INFORMIXSERVER=rio_grande
```



```
ONCONFIG=onconfig.rio
EDITOR=vi
```

```
export INFORMIXSERVER ONCONFIG EDITOR
```

```
#### end of IDS-oriented environment variables
```

Other critical variables include INFORMIXSERVER or the default instance to connect to as well as ONCONFIG or the configuration file for the instance. There are a number of optional environment parameters which can be set for default language, numeric and date display and other database locale-oriented functionality. Consult the installation guide or the *IBM Informix Dynamic Server Administrators Guide*, G251-2267, for a complete listing. For Windows ports, these parameters are set as part of the installation process.

The actual installation of the IDS binaries depends on the port (Windows or UNIX/Linux) as well as how much interaction is expected between the installation process and the person executing the installation. Both Windows and UNIX/Linux binaries can be installed using a “silent” mode which does not require any interaction with someone actually executing the installation. Typically silent installations are executed in environments where the IDS database server is hidden inside another application and must be installed as a part of the larger application installation process. The application developer creates a silent installation configuration file with a number of parameter values and includes it with the IDS binaries. The application installer copies the IDS binaries and the silent configuration file into a temporary directory then calls the IDS silent installer as shown in Example 2-3.

Example 2-3 Invoking the silent installer

Windows:

```
## create $INFORMIXDIR then change into it
mkdir c:\my_application\informix
cd c:\my_application\informix

## call the IDS silent installer
setup.exe -s c:\temp\informix\silent.ini -l
c:\temp\informix\silent_install.log
```

Unix/Linux:

```
[ids_install | installserver ] -silent -options options_file.ini
```

A Windows silent installation can only occur when executed from \$INFORMIXDIR. The *options_file.ini* parameter in the UNIX/Linux example is the fully-pathed filename of the silent installation configuration file.

When not using silent installation mode, there are a number of installation modes for installing on UNIX/Linux. These include:

- ▶ Console mode (the default)
- ▶ Using a Java-based graphical installer
- ▶ Extraction using a command-line utility, usually **tar**, followed by the execution of a command-line installer
- ▶ Invoking the Java installation JAR directly
- ▶ Using the RPM Package Manager (Linux only)

All of these must be executed as **root** or a UID with **root**-level permissions because file ownership and permissions will be changed as part of the installation.

Most UNIX/Linux administrators prefer to use command-line installers so IDS provides several depending on what needs to be installed as shown in Table 2-4.

Table 2-4 Command-line installation utilities

Installation command	What it does
ids_install	Installed all options of the IDS binary. The administrator can select which components to install and verify installation location
installserver	Only installs the database server
installconn	Only installs the IConnect or base client connectivity files
installclientsdk	Installs the Client Software Development Kit or the complete set of connectivity libraries. This enables the creation of customized client connectivity modules.

For non-silent Windows installations, double-click **setup.exe** on the distribution medium. One of the first questions will be whether or not to execute a domain installation. Generally, it is NOT recommended to execute a domain installation because all UID and group creation occurs on the domain server requiring domain administration privileges. In addition, all instance access must be verified from the domain server which can be quite slow when compared to local authorization.

As part of the Windows installation, the administrator will be asked if the installer should create and initialize an instance. If so, the installer will prompt for the instance name, size and location of the rootdbs as well as the size and location of one additional dbspace. The installer will use these parameters as well as those in \$INFORMIXDIR/etc/onconfig.std to create the instance's \$ONCONFIG file and initialize the instance. When the initialization has completed, the instance should be stopped and the \$ONCONFIG and \$SQLHOSTS modified as described next.

After the UNIX/Linux installation completes, the next step is to create and modify the instance's \$ONCONFIG as well as \$SQLHOSTS files. Templates for both are in \$INFORMIXDIR/etc and should be copied as a base. Information about each \$ONCONFIG parameter is available in the *IBM Informix Dynamic Server Administrator's Guide*, G251- 2267, because are the required and optional settings for the \$SQLHOSTS file.

Note: Windows \$SQLHOSTS information is included in the Windows Registry. Refer to the Windows installation guide for instructions about locating and modifying these entries.

Each local instance name and alias as well as remote instances to which a local instance might attempt to connect to needs to be defined in the \$SQLHOSTS file. There are five columns of information in this file, the local or remote instance name/alias, nettype (or connection protocol), host name or IP address of the physical server hosting the instance, the network "service name" (which corresponds to a port definition in /etc/services) or the specific port number to use and finally any connection options such as buffer size, keep-alive, security settings and so on. An entry in the fourth (or service name) column is required for all instances (including shared memory connection definitions) and must be unique to avoid communication conflicts.

Because most inter-instance connections will occur using a network-based communication protocol, the /etc/services file will require modification to reserve a specific network port number for each instance. If a Domain Name Server (DNS) is not used for physical server identification, the /etc/hosts file (or Windows equivalent) will need to be modified to include server name and network identification.

Access to instance services when it is initialized is based on OS authentication methods and is an area where IDS has made significant improvements in the past few releases. It is now possible to use LDAP and other Pluggable Authentication Modules (PAMs) for single user sign-on which also grants instance access. Without this, instance access is granted based on trusting the UID and computer the request is coming from. There are various ways of

configuring this kind of trusted access depending on the amount of trust an administrator wants to allow. Some options involve the creation of a `/etc/hosts.equiv` file, or a `.rhosts` file. Again, the installation and administrators manual explain these options in great detail. At least one access method must be configured prior to initializing the instance however.

The next step in preparing for instance initialization is the creation of the files or raw devices which will be used for chunks. UNIX/Linux IDS ports can use either “cooked” (aka file system files) or “raw” (unformatted disk partitions or LUNs) devices for chunks. Windows ports should only use cooked files. There are advantages and disadvantages to both types of devices. While raw spaces usually provide a 10% to 15% increase in performance, they are harder to backup requiring the use of one of the IDS backup utilities or an external backup as described in Chapter 9, “Legendary backup and restore” on page 257. With cooked files, they are no different than other files in the OS and can be backed up as part of an OS backup operation provided the instance is offline (best) or at least in quiescent or single-user mode so there is not activity occurring. As mentioned earlier, cooked files have a performance impact on instance operations because of the OS overhead involved in the I/O operations.

One critical key when setting up UNIX/Linux disk storage is the use of symbolic links which point to the actual devices. Best practices dictate creating all the symbolic links in a single directory then using those links in the path component of the chunk/instance creation command. With links, if a device has to be changed, all that is required is to re-point the link and recover/rebuild the data if applicable.

The last step prior to instance initialization is the creation of the `$MSGPATH` file. One of the parameters in the `$ONCONFIG`, this is the instance log where activity, problem and other messages are recorded. It is particularly helpful to monitor this log in real-time during instance initialization to catch errors or to see when initialization has completed. Use the UNIX/Linux **touch** command (or the Windows equivalent) to create the file as fully pathed in the `$ONCONFIG`.

With all the variables set, the binaries installed, the `$ONCONFIG` and `$SQLHOSTS` files copied and modified, it is time to initialize the instance. In a separate window, start a **tail -f** command on the `$MSGPATH` file to watch initialization messages in real-time. The **-i** flag of the IDS *oninit* utility initializes (or reinitializes if the instance has been created previously) the instance. Again, best practices dictate monitoring the output from this command so the complete command would be **oninit -ivy**. If all goes well, the last line of output from this command will be a status return code of 5. This does *not* mean the instance is completely initialized and ready for use.

In the `$MSGPATH` file there will two sets of three messages each. In the first set, they will say the creation of the sysmaster, sysutils, and sysusers databases has

begun. The second set will indicate that each database has been created. The instance *must not* be used until these databases have been created. If a user session connects and attempts to execute any type of operation, the instance will become corrupted requiring reinitialization.

When the instance is initialized, additional dbspaces can be created to hold user data as well as one or more temporary dbspaces. At least one smart BLOBspace should also be created even if the database will not be using SLOBs. The purpose for creating the space listed as SBSPACENAME and SYSSBSPACENAME in \$ONCONFIG is to store optimizer statistics generated through **update statistics** commands. A 10 to 15 MB smart BLOBspace would be sufficient for these statistics. Dbspace, smart BLOBspace and chunk allocations are created with the *onspaces* utility.

With additional spaces, if more logical logs are required or simply moved out of the rootdbs along with the physical log, this can now occur. The *onparams* utility is used to create or drop logical logs or move or resize the physical log.

When the instance overhead structures have been created, user databases with all their components can be created and loaded with data if necessary. If a large amount of data needs to be loaded, create the databases in non-logged mode, load the tables then convert the database to logged mode in order to use explicit transactions. Converting an existing database to or from a logged mode can be done with the *ontape* or *ondblog* utilities. A baseline full backup should be created at this point. When completed, users can be allowed to access the instance and the real work of production monitoring and tuning begins.

2.4 Administration and monitoring utilities

Each edition of Informix Dynamic Server (described in Chapter 1, “IDS essentials” on page 1) includes a complete set of utilities to administer and monitor the instance. Most of them are command-line oriented as befits the database server’s UNIX/Linux heritage but there are two graphical utilities as well. The command-line utilities are briefly described in Table 2-5.

Table 2-5 IDS command-line utilities

Utility	Description
oncheck	Depending on the options used, <i>oncheck</i> can perform the following: Check specified disk structures for inconsistencies, repair indexes that are found to contain inconsistencies, display disk structure information, check and display information about user-defined data types across distributed databases.
ondblog	Used to change the logging mode of a database.
oninit	Used to start the instance. With the <i>-i</i> flag, it will initialize or reinitialize an instance. Depending on other flags, can bring the instance from offline to single-user, quiescent or multi-user operating modes.
onlog	Used to view the contents of a logical log. The log can either still be stored in the instance or backed up to tape/disk.
onmode	Depending on the options used, <i>onmode</i> can perform the following: Change the instance operating mode when started, force a checkpoint to occur, close the active logical log and switch to the next log, kill a user session, add or remove VPs, add and release additional segments to the virtual portion of shared memory, start or stop HDR functionality, dynamically change certain configuration, PDQ, set explain and other parameters.
onparams	Used to add or delete logical logs, move or resize the physical log and add a buffer pool to support standard dbspaces created with a larger than default page size.
onspaces	Used to manage the creation and deletion of chunks and dbspaces/BLOBspaces (both standard and temporary) in addition to enabling IDS-based mirroring, renaming spaces and changing the status of mirror chunks.

Utility	Description
onstat	Reads shared-memory structures and provides statistics about the instance at the time the command is executed. The system-monitoring interface (aka the sysmaster database) also provides information about the instance and can be queried directly through any SQL-oriented tool.

The *IBM Informix Dynamic Server Administrator's Reference*, G251-2681, has a complete explanation of each command-line utility.

IDS also provides two graphical utilities. The Informix Server Administrator (ISA) is a Web-based tool enabling an administrator to execute all instance administration and monitoring tasks as well as some limited database administration. When installing the ISA, the administrator can indicate whether it should use a previously configured Web server or the Apache server bundled as part of the ISA installation.

The ISA is particularly helpful for training new instance administrators because it removes the need to remember all 100 or so *onstat* flags and other command-line syntax. With its point and click interface, tasks can be accomplished relatively quickly. The added training bonus comes from the fact that as the ISA executes a task, the command-line equivalent is displayed so the new administrator can learn the correct syntax. Figure 2-5 illustrates the add an additional logical log ISA page.

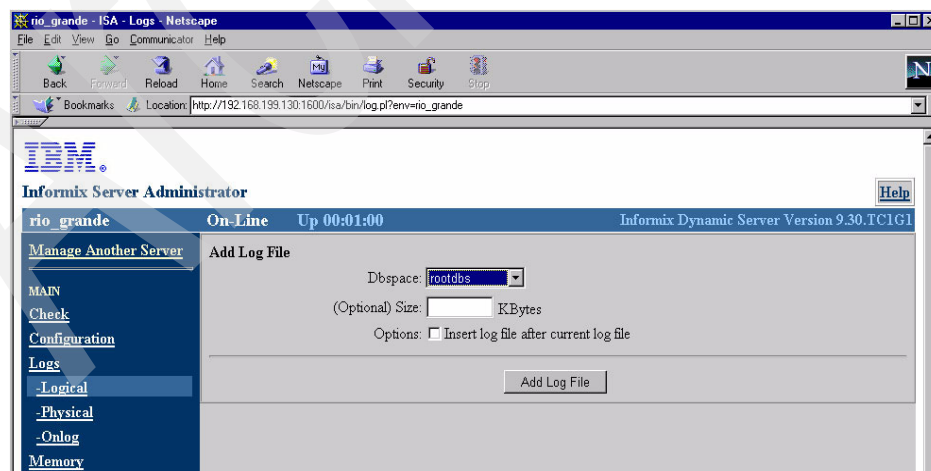


Figure 2-5 An ISA screen

The second utility, ServerStudio Java Edition (SSJE), was originally designed to be the complementary database administrators tool; it has since grown and increased its suite of functionality to include instance monitoring and administration as well as statistical gathering and trending to provide a comprehensive view of instance and database operations.

The SSJE as it exists in the IDS distribution is licensed technology from AGS, Ltd., an Informix partner. It provides basic DBA functionality while more extensive instance administration or monitoring and trending functionality modules can be added by purchasing those modules from an authorized IBM seller/reseller or directly from AGS, Ltd. Figure 2-6, Figure 2-7, and Figure 2-8 are example screen shots of the SSJE, simply for familiarity of appearance.

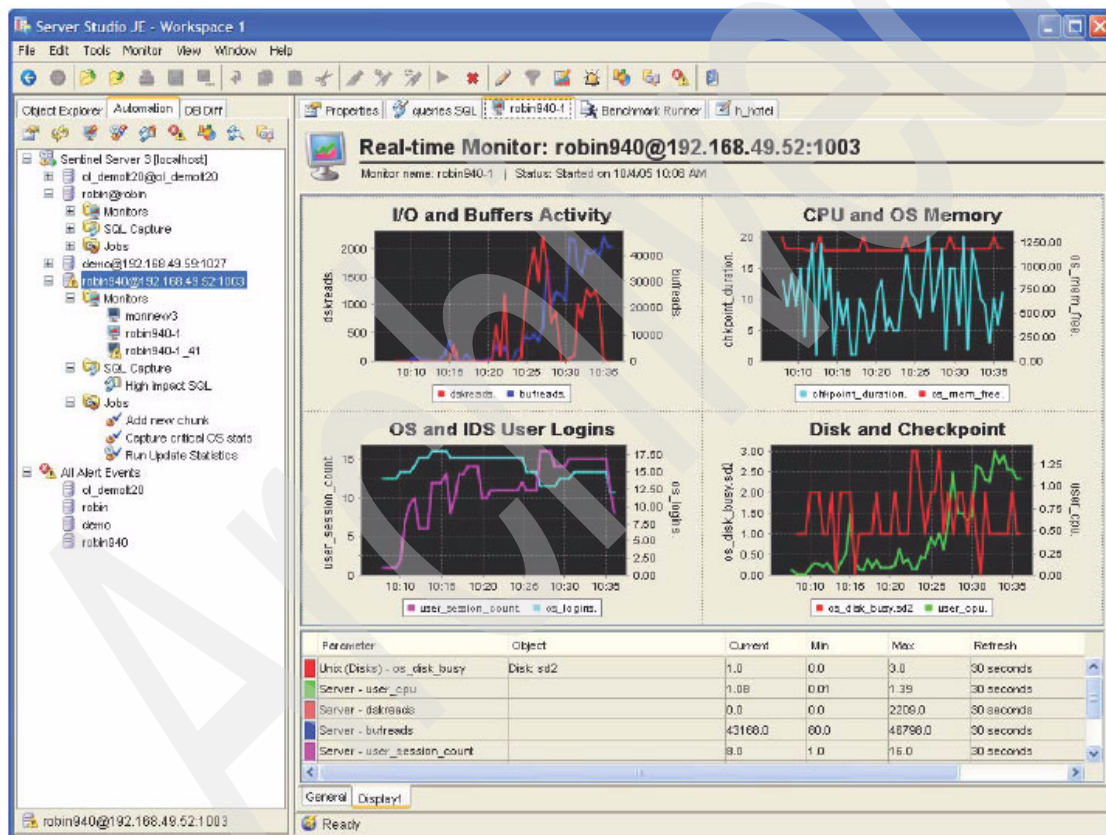


Figure 2-6 The SSJE real-time instance monitoring module

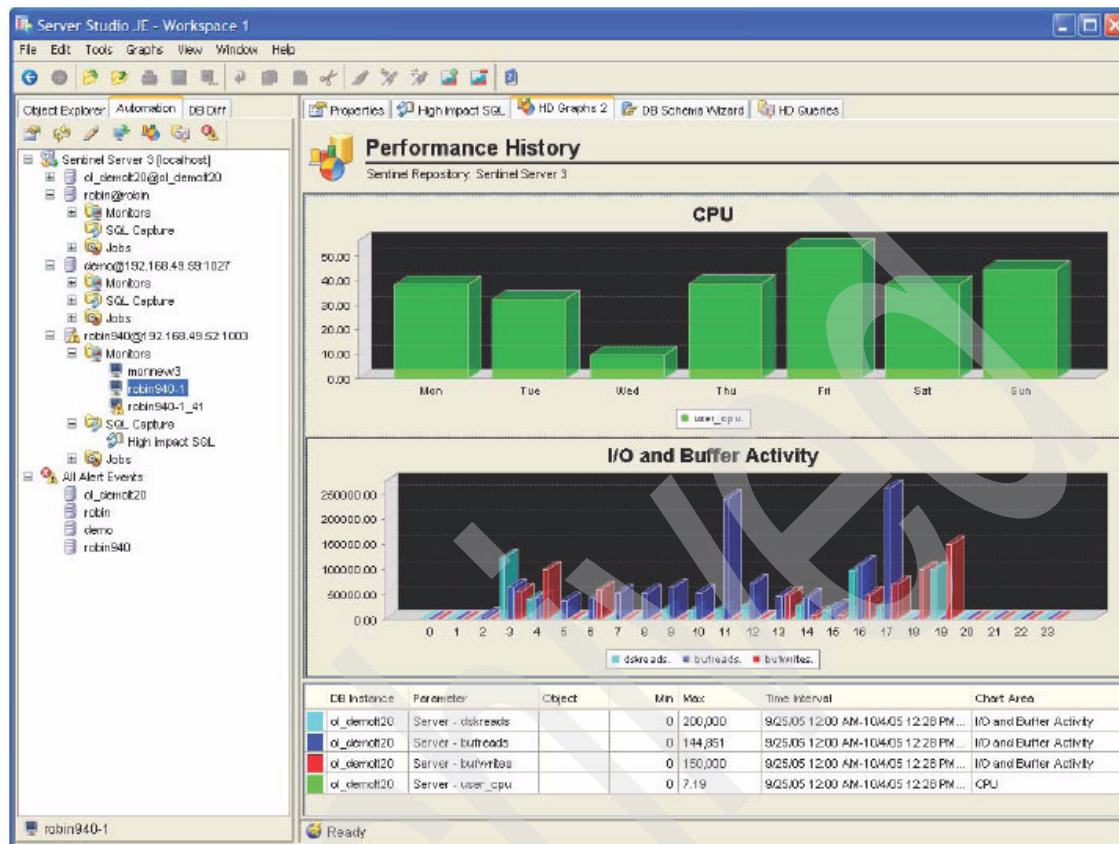


Figure 2-8 The SSJE performance trending module

Regardless of which utility is used, nothing is hidden from the administrator, all aspects of instance operations can be interrogated and analyzed if desired. While this requires work on the administrators part, it enables the administrator to perform fine-grain tuning to squeeze as much performance as possible out of the instance if required.

The SQL language

Intergalactic DataSpeak is a term some use to imply that SQL is the language most widely used to describe operations on the data in databases. You might or might not agree, but SQL is widely used. In addition, the SQL language standard committee continues to refine and expand the standard.

In this chapter, we discuss a number of recent additions to the SQL language that are recognized by IDS as well as how IDS processes SQL statements. Some of these additions were first available in IDS V9.40 and some are new in IDS V10. All are powerful extensions to the SQL language or facilities that are in the SQL standard but were not recognized previously by IDS.

3.1 The CASE clause

The CASE expression clause is a bit of procedural logic in an otherwise declarative language. By using the CASE clause you might be able to condense multiple SQL statements into a single statement or simplify the SQL in other ways. It is part of the SQL:1999 standard, and it is supported by all the major DBMS.

A CASE expression is a conditional expression which is very similar to the switch statement in the C programming language. Apart from the standard data types, the CASE clause allows usage of expression involving cast expressions and extended data types, however currently it does not allow expressions that compare BYTE or TEXT values.

Example 3-1 shows the syntax. You have the choice of two forms, one for simple operations and one for more complex operations. As per the syntax, you must include at least one when clause within the CASE expression, but any subsequent when clause and the else clause are optional. If the CASE clause does not handle the else clause and if none of the when conditions evaluate to true, the resulting value is null.

The first statement uses the CASE clause to translate the integer column type values into their corresponding character meanings. That makes the meaning of the result set much more understandable to a reader. For a program, this can protect the program from changes to the values or their meanings.

The second statement shows the simpler form of the same statement. In this one we include the else condition to handle any values that are not included in any of the other conditions of the clause. The else could have been included in the first form as well.

Using the else protects against inadvertent null values being included in the result set, which can relieve programs of having to check for nulls. If none of the conditions in the case clause are valid or true, then a null value is returned. So the else ensures that at least one condition is always met.

If more than one condition is true, the first true condition is used.

Example 3-1 The CASE clause

```
SELECT  a.tabname[1,18] table,
        b.colname[1,18] AS column,
        CASE
        WHEN b.coltype = 0 OR b.coltype = 256 THEN 'char'
        WHEN b.coltype = 1 OR b.coltype = 257 THEN 'smallint'
        WHEN b.coltype = 2 OR b.coltype = 258 THEN 'integer'
```

```

        WHEN b.coltype = 3 OR b.coltype = 259 THEN 'float'
        WHEN b.coltype = 4 OR b.coltype = 260 THEN 'smallfloat'
        WHEN b.coltype = 5 OR b.coltype = 261 THEN 'decimal'
        WHEN b.coltype = 6 OR b.coltype = 262 THEN 'serial'
        WHEN b.coltype = 7 OR b.coltype = 263 THEN 'date'
        WHEN b.coltype = 8 OR b.coltype = 264 THEN 'money'
        WHEN b.coltype = 9 OR b.coltype = 265 THEN 'null'
        WHEN b.coltype = 10 OR b.coltype = 266 THEN 'datetime'
        WHEN b.coltype = 11 OR b.coltype = 267 THEN 'byte'
        WHEN b.coltype = 12 OR b.coltype = 268 THEN 'text'
        WHEN b.coltype = 13 OR b.coltype = 269 THEN 'varchar'
        WHEN b.coltype = 14 OR b.coltype = 270 THEN 'interval'
        WHEN b.coltype = 15 OR b.coltype = 271 THEN 'nchar'
        WHEN b.coltype = 16 OR b.coltype = 272 THEN 'nvarchar'
        WHEN b.coltype = 17 OR b.coltype = 273 THEN 'int8'
        WHEN b.coltype = 18 OR b.coltype = 274 THEN 'serial8'
        WHEN b.coltype = 19 OR b.coltype = 275 THEN 'set'
        WHEN b.coltype = 20 OR b.coltype = 276 THEN 'multiset'
        WHEN b.coltype = 21 OR b.coltype = 277 THEN 'list'
        WHEN b.coltype = 22 OR b.coltype = 278 THEN 'row'
        WHEN b.coltype = 23 OR b.coltype = 279 THEN 'collection'
        WHEN b.coltype = 24 OR b.coltype = 280 THEN 'rowref'
        WHEN b.coltype = 40 OR b.coltype = 296 THEN 'blob'
        WHEN b.coltype = 41 OR b.coltype = 297 THEN 'lvarchar'
        end AS columntype,
        HEX(collength) AS size
FROM   systables a,
       syscolumns b
WHERE  a.tabid = b.tabid AND
       b.tabid > 99
GROUP BY a.tabname, b.colname, b.coltype, b.collength
ORDER BY 1
;

SELECT a.tabname[1,18] table,
       b.colname[1,18] AS column,
       CASE coltype
         WHEN 0 THEN "char"
         WHEN 1 THEN "smallint"
         WHEN 2 THEN "integer"
         WHEN 3 THEN "float"
         WHEN 4 THEN "smallfloat"
         WHEN 5 THEN "decimal"
         WHEN 6 THEN "serial"
         WHEN 7 THEN "date"
         WHEN 8 THEN "money"
         WHEN 9 THEN "null"

```

```

        WHEN 10 THEN "datetime"
        WHEN 11 THEN "byte"
        WHEN 12 THEN "text"
        WHEN 13 THEN "varchar"
        WHEN 14 THEN "interval"
        WHEN 15 THEN "nchar"
        WHEN 16 THEN "nvarchar"
        ELSE "unknown"
    end AS columntype,
    HEX(collength) AS size
FROM    systables a,
        syscolumns b
WHERE   a.tabid = b.tabid AND
        b.tabid > 99
ORDER BY 1
;

```

Another use of the CASE clause is for counting the number of rows that meet the conditions that you specify. IDS does not currently support use of multiple COUNT functions in a single select statement. However, using CASE and SUM, you can achieve the same result, as shown in Example 3-2. You cannot use the COUNT function more than once in a single SELECT, but you can use SUM as much as you want. The CASE clause allows you to count by returning a 1 for those rows that meet the required condition and null otherwise. The SUM just includes the non-null values.

Example 3-2 Counting with CASE

```

SELECT  SUM ( CASE
            WHEN account_status = 'A' THEN 1
            ELSE NULL
        END
        ) AS a_accounts,
        SUM ( CASE
            WHEN discount::float > 0 THEN 1
            ELSE NULL
        END
        ) AS nonzero_discounts
FROM    manufact
;

a_accounts nonzero_discounts
    12             5

```

1 row(s) retrieved.

You can use the CASE clause in many places in a SQL statement. Example 3-3 shows how to update a table using the CASE clause to choose the right value for each row. The power of this usage is that you can get all the updates done in a single statement and single pass through the table. Without the CASE, you would have to use multiple statements, and that would mean much more work for the DBMS.

Example 3-3 Multiple updates in one statement

```
UPDATE staff SET manager = CASE
    WHEN dept = 10 AND job = 'Mgr'
    THEN (SELECT id FROM staff WHERE job = 'CEO')
    WHEN dept = 15 AND job != 'Mgr'
    THEN (SELECT id FROM staff WHERE job = 'Mgr' AND dept = 15)
    WHEN dept = 20 AND job != 'Mgr'
    THEN (SELECT id FROM staff WHERE job = 'Mgr' AND dept = 20)
    WHEN dept = 38 AND job != 'Mgr'
    THEN (SELECT id FROM staff WHERE job = 'Mgr' AND dept = 38)
    WHEN (dept = 15 OR dept = 20 OR dept = 38) AND job = 'Mgr'
    THEN 210
    ...
END;
```

One of the factors that programmers tend to overlook while using a CASE clause is that, if there are multiple expressions inside the CASE clause evaluating to true, then only the result of the first evaluated true statement is returned and the rest of the expressions that evaluate to true are either ignored or not parsed.

In general, you can use CASE wherever there is an expression in a SQL statement. Thus, you can do limited decision-making in almost any SQL expression. Think about CASE whenever you find a set of statements doing distinct but similar operations. You might be able to condense them into a single statement.

3.2 The TRUNCATE TABLE command

A common request is for IDS to have a means to truncate a table. Truncation means removing all the rows while retaining or dropping the storage space for those rows. In IDS terms, the facility of retaining the storage space during the truncation is an option given to the user. The default behavior is to drop the storage space. It is approximately the equivalent of dropping and recreating a table.

3.2.1 The syntax of the command

TRUNCATE TABLE, or TRUNCATE as it is commonly referred, has a very simple syntax, as shown in Figure 3-1.

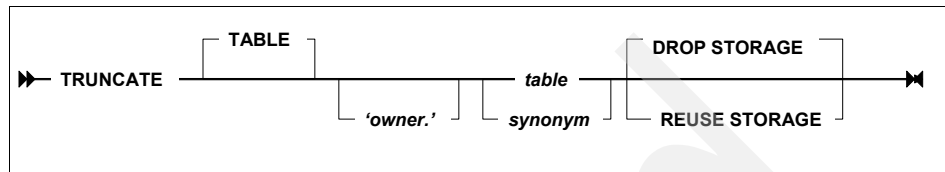


Figure 3-1 TRUNCATE TABLE syntax

The usage of the keyword TABLE along with the command TRUNCATE is optional and has been made available for the purpose of application program legibility. It has two other optional parameters, DROP STORAGE and REUSE STORAGE. DROP STORAGE can be specified to release the storage (partition extents) allocated for the table and the index, and is the default behavior. Whereas REUSE STORAGE will keep the same storage space for the table after the truncate. You can use REUSE STORAGE for those tables that are repetitively emptied and reloaded.

3.2.2 The DROP TABLE command versus the DELETE command versus the TRUNCATE TABLE command

There are similarities between the DROP TABLE, DELETE, and TRUNCATE TABLE commands.

Like TRUNCATE, the DROP TABLE removes all the data and the index. And that is where the similarities end, because DROP TABLE also removes the table and its associated reference from the database. Thus, when the DROP TABLE statement is executed the table will be non-existent in the database, and therefore inaccessible. The table will have to be recreated to again enable access.

When deleting all the data, the DELETE FROM command (without the WHERE qualifier) behaves the same way as TRUNCATE. In such a scenario DELETE has more overheads than TRUNCATE. It is typically more efficient to use the TRUNCATE statement, rather than the DELETE statement, to remove all rows from a table.

3.2.3 The basic truncation

Like any delete function, to execute TRUNCATE you must have an exclusive access to that table. TRUNCATE is very fast and efficient in scenarios where the table contains large amounts of data. This is because the deletion does not happen on individual rows, which means that no delete triggers are fired. So any logging of activity or other recording done in a trigger does not occur. This is one of the main reasons why TRUNCATE cannot be used on tables that are a part of Enterprise Replication (ER) but can be used with High Availability Data Replication (HDR).

Example 3-4 shows the truncate statement in action. The examples in this section all use a table called *dual*, which has just one column and a single row of data. TRUNCATE is not a large operation for a table with just one column and one row. However, the operation is very fast on table of any size because the individual rows are not deleted.

Example 3-4 Preparing the TRUNCATE data

```
> CREATE TABLE dual ( c1 INTEGER);
```

Table created.

```
> TRUNCATE dual;
```

Table truncated.

```
> SELECT * FROM dual;
```

c1

No rows found.

```
> INSERT INTO dual VALUES (1);
```

1 row(s) inserted.

```
> SELECT * FROM dual;
```

c1

1

1 row(s) retrieved.

The statistics about the table are not changed. So the number of rows and distribution of values that were recorded in the most recent update statistics are retained. Example 3-5 shows that the `systables.nrows` count is not affected by truncating the table.

Example 3-5 TRUNCATE and table statistics

```
> SELECT tabname, nrows FROM systables WHERE tabname = 'dual';
```

```
tabname  dual
nrows    0
```

1 row(s) retrieved.

```
> UPDATE STATISTICS FOR TABLE dual;
```

Statistics updated.

```
> SELECT tabname, nrows FROM systables WHERE tabname = 'dual';
```

```
tabname  dual
nrows    1
```

1 row(s) retrieved.

```
> TRUNCATE dual;
```

Table truncated.

```
> SELECT tabname, nrows FROM systables WHERE tabname = 'dual';
```

```
tabname  dual
nrows    1
```

1 row(s) retrieved.

```
> SELECT * FROM dual;
```

```
      c1
```

No rows found.

```
> UPDATE STATISTICS FOR TABLE dual;
```

Statistics updated.

```
> SELECT tabname, nrow FROM systables WHERE tabname = 'dual';
```

```
tabname  dual  
nrow     0
```

```
1 row(s) retrieved.
```

3.2.4 TRUNCATE and transactions

You must be careful about transaction boundaries when you truncate a table. The statement cannot be rolled back unless it is the last statement in the transaction.

The only allowed operations in a transaction after a TRUNCATE statement are COMMIT and ROLLBACK. Try running a TRUNCATE statement immediately after the first TRUNCATE and see what happens. Example 3-6 demonstrates what happens if you do try to execute a SQL statement within a transaction after a TRUNCATE statement.

Example 3-6 TRUNCATE in a transaction

```
> BEGIN WORK;
```

```
Started transaction.
```

```
> TRUNCATE dual;
```

```
Table truncated.
```

```
> SELECT * FROM dual;
```

```
26021: No operations allowed after truncate in a transaction.
```

```
Error in line 1
```

```
Near character position 17
```

3.3 Pagination

It is a common experience that when you search for a common or popular keyword on the Web through a search engine that the search returns millions of links. Of course, you would prefer that what you are looking for be at the top of that list. Apart from finding what you want quickly, the search engine also gives

you a sense of the total information that is available and lets you browse the result set back and forth.

Pagination is the ability to divide the information into manageable chunks in some order—either predetermined or user-defined. Business applications need to provide report generation to view information online or to generate hardcopy reports. Both methods require the results to be paginated. So, applications that use databases need to include a pagination strategy. In this section, we focus on the FIRST, LIMIT, SKIP clauses and the addition of the ORDER BY clause to derived tables to enable pagination using SQL.

3.3.1 Pagination examples

Here are some examples of the practical use of pagination.

Pagination in search engine results

Figure 3-2 illustrates the fetch results of a sample search query using an Internet search engine. The first search page shows details about the fetch in totality, including details such as fetch time, total records fetched, the number of result pages, and whether the fetch is from the search engine cache or from the search database. Subsequent pages would also show similar information.

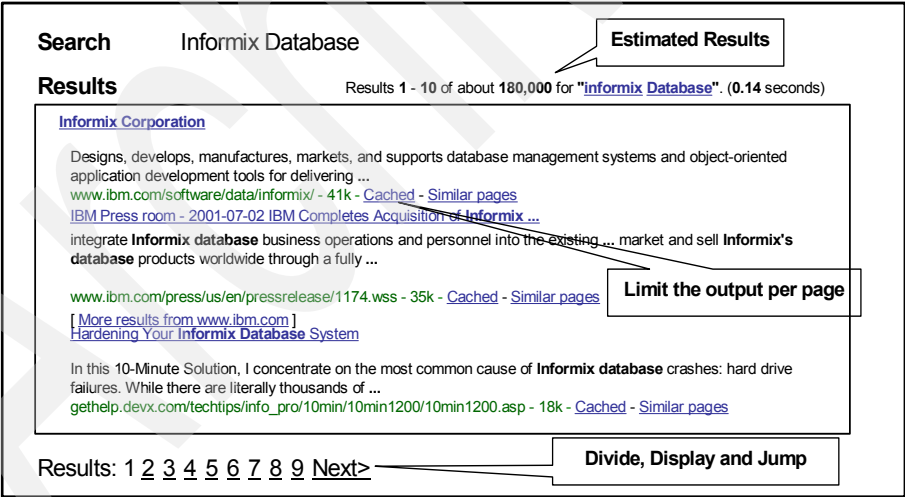


Figure 3-2 Search engine results

Pagination in business applications

Pagination is also important in back office applications. Reporting applications such as case status reports, workflow status, and order status should support pagination of the report. Users of interactive reports require control of fields, flexible sort order, and search within search results.

Figure 3-3 shows a sample business application report. It is sorted by Order Date, but there is an option to sort by other fields or change from ascending order to descending, and vice versa.

Option To Change Sort Order

Current Sort Order

Order Report for April

Order id ▲▼	Order item ▲▼	Quantity ▲▼	Order Date ▲▼	Status ▲▼
2345	Wilson Balls	32	9-03-2006	CUST-RECVD
5832	Adidas Socks	12	9-04-2006	ORD-RECVD
3474	Mouth Guard	6	9-07-2006	ORD-RECVD
3723	Adidas Shoes	1	9-12-2006	IN-SHIPPING
2383	Gamma Strings	4	9-22-2006	BACK-ORDER
2382	Wilson Balls	32	9-29-2006	ORD-RECVD

<<First <Previous Page 4 of 8 Next> Last>>

Navigating Through Result Sets

Figure 3-3 Sample business application

Pagination by the book

The word *pagination* comes from the publishing domain. When any book is published, a decision must be made concerning the method of pagination. A similar strategy is implemented as a pagination technique for database and Web applications.

Pagination is:

- ▶ The system by which pages are numbered.
- ▶ The arrangement and number of pages in a book, as noted in a catalog or bibliography.
- ▶ Laying out printed pages, which includes setting up and printing columns as well as rules and borders.

Although pagination is used synonymously with page makeup, the term often refers to the printing of long manuscripts rather than ads and brochures.

For business applications, especially interactive applications, results must be provided in manageable *chunks*, such as pages or screens. The application designer must be aware of the user context, the amount of information that can be displayed per page, and the amount of information available. Within each page or screen, the application has to indicate how far the user has traversed the result or report.

3.3.2 Pagination for database and Web applications

In book publishing, the publisher typically sets the rules. When a decision is made regarding the pagination method, the readers cannot change it. The only way to re-paginate the book is to tear it apart and rebind it. In business applications, users typically set the rules. They specify the parameters for pagination, the number of rows in each page, and ordering criteria, such as price, user, and by date. But when they get the results, they can immediately change the order, perhaps simply by a click. Because, users do change their minds.

Pagination requirements

In this section we provide brief descriptions of pagination requirements:

- ▶ **Search Criteria:** The search can be on specific keys such as customer ID, items, or orders. In some applications, a search within search results must also be supported. Depending on volume of data and performance requirements, caching or refresh strategy can be determined.
- ▶ **Result Set Browsing:** Users get the first page browsed by default, but should be able to navigate to the first, next, or previous page, or jump to a random page within the result set. If the result set is large, the number of pages to jump should be limited to avoid clutter.
- ▶ **User Context and Parameters:** Depending on the settings and previous preferences, the application might need to alter the pagination. For example, it might need to specify the number of rows to be displayed in each page and the number links for other pages.
- ▶ **Ordering Mechanism:** Every report generated has an order, order date, customer ID, shipping date, and so on. For pagination based on SQL queries, choose the keys for ORDER BY clauses carefully. It is most efficient when the database chooses an index to return sorted data.
- ▶ **Resources:** You must determine the frequency with which the application can query the database to retrieve the results versus how much can be cached in the middleware. If the data does not change, you do not require a complete snapshot or you have a way to get consistent results (for example, based on

a previous timestamp), then you can choose to issue an SQL query for every page. However, for faster responses, you can choose to cache some or all of the result set in the middleware and choose to go back to database when the requested pages are not in the cache. A developer typically needs to make these decisions.

- Performance: Response time is important for interactive applications. And, more specially, the first page display. For database queries, tune the query and index for returning first rows quickly. For more details, see 3.3.6, “Performance considerations” on page 92.

3.3.3 Pagination as in IDS: SKIP m FIRST n

IDS V10 has a FIRST, LIMIT, and SKIP clause as part of SELECT statement to enable database oriented pagination.

One of the things often required is to examine parts of lists of items. If the desired result is the first or last part of the list and if the list is not large, then the ORDER BY clause is often sufficient to accomplish the task. However, for large sets or for some section in the middle of the list, there has previously been no easy solution.

IDS V10 introduced a new Projection Clause syntax for the SELECT statement that can control the number of qualifying rows in the result set of a query:

- SKIP *offset*
- FIRST *max*

The SKIP offset clause excludes the first offset qualifying rows from the result set. If the FIRST max clause is also included, no more than max rows are returned.

The syntax requires that if both SKIP and FIRST are used together, then the SKIP must occur first.

The SKIP and FIRST options in SELECT cannot be used in the following contexts:

- In the view definition while doing a CREATE VIEW
- In nested SELECT statements
- In subqueries

3.3.4 Reserved words: SKIP, FIRST. Or is it?

The differentiating factor that can cause the SKIP and FIRST keywords to be considered as a reserved word or as a literal depends upon the usage. The syntax requires that when both SKIP and FIRST are used in a SELECT projection list then both must be followed by an integer value. The parser evaluates the status of these keywords according to the integer value that follows them.

If no integer follows the FIRST keyword, the database server interprets FIRST as a column identifier. For example, consider that we have a table named skipfirst with column names skip and first, with the data populated as shown in the first query in Example 3-7. The second query shows the interpretation of FIRST as column name and the third query interprets that as a SELECT.FIRST option.

Example 3-7 Using FIRST as column name

```
> SELECT * FROM skipfirst;
```

skip	first
11	12
21	22
31	32
41	42
51	52

5 row(s) retrieved.

```
> SELECT first FROM skipfirst;
```

first
12
22
32
42
52

5 row(s) retrieved.

```
> SELECT FIRST 2 first FROM skipfirst;
```

first
12
22

2 row(s) retrieved.

The same considerations apply to the SKIP keyword. If no literal integer or integer variable follows the SKIP keyword in the projection clause, the parser interprets SKIP as a column name. If no data source in the FROM clause has a column with that name, the query fails with an error.

Example 3-8 shows various scenarios of how the SKIP and FIRST is interpreted by the server. Query 1 interprets SKIP as the column name. Query 2 interprets SKIP as a keyword and a column name. Query 3 interprets SKIP as a keyword and FIRST as a column name. Query 4 interprets SKIP as a keyword and FIRST as both keyword and column name. Query 5 fetches the first 2 rows for the column skip. Here the token FIRST is interpreted as keyword, where as the SKIP token is interpreted as a column name. Query 6 and Query 7 establish that unlike Query 5, when a integer value is added in front of the SKIP token, it is interpreted as keyword. In Query 8 the parser expects an integer qualifier or a reserved token such as comma, from etc. after the SKIP. When none of that is found a syntax error is returned. Query 9 and Query 10 use the customer table from the stores_demo database. In Query 9 the parser tries to retrieve the column name from the table mentioned in the FROM clause, failing which the query returns an error. The Query 10 is an improvisation of Query 9 where the SKIP keyword is follow by an valid integer qualifier for the same table.

Example 3-8 Parser interpretation of SKIP and FIRST as keywords

```
> -- Query 1
> SELECT skip FROM skipfirst;

    skip
    11
    21
    31
    41
    51

5 row(s) retrieved.

> -- Query 2
> SELECT SKIP 2 skip FROM skipfirst;

    skip
    31
    41
    51

3 row(s) retrieved.
```

```
> -- Query 3
> SELECT SKIP 2 first FROM skipfirst;
```

```
first
  32
  42
  52
```

3 row(s) retrieved.

```
> -- Query 4
> SELECT SKIP 2 FIRST 2 first FROM skipfirst;
```

```
first
  32
  42
```

2 row(s) retrieved.

```
> -- Query 5
> SELECT FIRST 2 skip FROM skipfirst;
```

```
skip
  11
  21
```

2 row(s) retrieved.

```
> -- Query 6
> SELECT FIRST 2 SKIP 2 FROM skipfirst;
```

```
201: A syntax error has occurred.
Error in line 1
Near character position 21
```

```
> -- Query 7
> SELECT FIRST 2 SKIP 2 first FROM skipfirst;
```

```
201: A syntax error has occurred.
Error in line 1
```

Near character position 21

```
> -- Query 8  
> SELECT SKIP FIRST 2 first FROM skipfirst;
```

201: A syntax error has occurred.
Error in line 1
Near character position 19

```
> -- Query 9  
> SELECT skip FROM customer;
```

201: A syntax error has occurred.
Error in line 1
Near character position 24

```
> -- Query 10  
> SELECT SKIP 25 fname FROM customer;
```

fname
Eileen
Kim
Frank

3 row(s) retrieved.

3.3.5 Working with data subsets

With IDS V10, there is a way to get any subset of a list you wish. Example 3-9 demonstrates how this can be done.

Here, the state sales taxes are retrieved in groups. Query 1 gets the five states with the highest tax rates. Query 2 gets the states with the sixth through tenth highest rates. Subsequently Query 3 gets the states with the sixth through fifteenth highest rates. Thus, if you want to display items in groups, you might want to use this technique rather than a cursor in order to simplify the programming and reduce the amount of data held in the application at any time.

Example 3-9 Selecting Subsets of a List

```
> -- Query 1
```

```
SELECT FIRST 5
```

```

        SUBSTR(sname,1,15) AS state,
        (sales_tax::DECIMAL(5,5))*100 AS tax
FROM    state
ORDER BY tax desc
;

```

state	tax
Wisconsin	8.5700000000
California	8.2500000000
Texas	8.2500000000
New York	8.2500000000
Washington	8.2000000000

5 row(s) retrieved.

> -- Query 2

```

SELECT  SKIP 5 FIRST 5
        SUBSTR(sname,1,15) AS state,
        (sales_tax::DECIMAL(5,5))*100 AS tax
FROM    state
ORDER BY tax desc;

```

state	tax
Connecticut	8.0000000000
Rhode Island	7.0000000000
Florida	7.0000000000
New Jersey	7.0000000000
Minnesota	6.5000000000

5 row(s) retrieved.

> -- Query 3

```

SELECT  SKIP 5 FIRST 10
        SUBSTR(sname,1,15) AS state,
        (sales_tax::DECIMAL(5,5))*100 AS tax
FROM    state
ORDER BY tax desc
;

```

state	tax
Connecticut	8.0000000000
Rhode Island	7.0000000000

Florida	7.0000000000
New Jersey	7.0000000000
Minnesota	6.5000000000
Illinois	6.2500000000
D.C.	6.0000000000
West Virginia	6.0000000000
Kentucky	6.0000000000
Mississippi	6.0000000000

10 row(s) retrieved.

The ORDER BY clause is very important. The SKIP and FIRST logic is not executed until after the entire result set has been formed. So if you change the ordering from DESC to ASC, then you will retrieve different rows. Similarly, if you change the column that determines the ordering, you might get different rows.

The way to think about a query is to first determine that you are getting the entire set the way you want it. After that, limit what is returned by using SKIP, FIRST or both. Example 3-10 shows the differences.

Here Query 1 gets the first five states alphabetically, in reverse order. Query 2 gets the sixth through tenth states, again in normal alphabetic order. Query 3 gets the sixth through tenth states also, but this time in reverse alphabetic order.

Example 3-10 The Effects of Ordering on SKIP...FIRST

> -- Query 1

```
SELECT  FIRST 5
        SUBSTR(sname,1,15) AS state,
        (sales_tax::DECIMAL(5,5))*100 AS tax
FROM    state
ORDER BY state desc
;
```

state	tax
Wyoming	0.0000000000
Wisconsin	8.5700000000
West Virginia	6.0000000000
Washington	8.2000000000
Virginia	4.2500000000

5 row(s) retrieved.

> -- Query 2

```

SELECT  SKIP 5 FIRST 5
        SUBSTR(sname,1,15) AS state,
        (sales_tax::DECIMAL(5,5))*100 AS tax
FROM    state
ORDER BY state
;

```

state	tax
Colorado	3.7000000000
Connecticut	8.0000000000
D.C.	6.0000000000
Delaware	0.0000000000
Florida	7.0000000000

5 row(s) retrieved.

> -- Query 3

```

SELECT  SKIP 5 FIRST 5
        SUBSTR(sname,1,15) AS state,
        (sales_tax::DECIMAL(5,5))*100 AS tax
FROM    state
ORDER BY state desc
;

```

state	tax
Vermont	4.0000000000
Utah	5.0000000000
Texas	8.2500000000
Tennessee	5.5000000000
South Dakota	0.0000000000

5 row(s) retrieved.

3.3.6 Performance considerations

Pagination queries use the ORDER BY clause to get a sorted result set. While the server can sort the results in any order or combination, it takes time and memory. Wherever possible, the server tries to use available index to evaluate the ORDER BY clause. So, formulate the queries to exploit existing indexes. If the keys to the ORDER BY clause comes from multiple tables, IDS physically sorts the data. The sorting algorithm is aware of the FIRST and SKIP clause and will stop the sorting process when it has satisfied the request.

Look at the effect of ORDER BY clause on query plans that the optimizer generates, in the following cases:

Case 1

As shown in Example 3-11, by default the optimizer chooses the best plan to return all rows. Here it chooses dynamic hash join. Even though the overall performance is good, when the table is large it can take some time to return first rows.

Example 3-11 Case 1

QUERY:

```
SELECT SKIP 100 FIRST 10 *  
FROM    item, component  
WHERE   item.id = component.itemid  
ORDER BY item.id
```

Estimated Cost: 19116

Estimated # of Rows Returned: 39342

Temporary Files Required For: Order By

1) informix.item: INDEX PATH

(1) Index Keys: id scode bin (Serial, fragments: ALL)

2) informix.component: SEQUENTIAL SCAN

DYNAMIC HASH JOIN

Dynamic Hash Filters: informix.item.id = informix.component.itemid

Case 2

Now in Example 3-12, you can see that the sorting columns *item.id* and *component.partno* come FROM two tables. This forces sorting of the result set, and again the optimizer has chosen dynamic hash join. Here again there will be a delay in returning the FIRST rows to the application.

Example 3-12 Case 2

QUERY:

```
SELECT SKIP 100 FIRST 10 *  
FROM    item, component  
WHERE   item.id = component.itemid  
ORDER BY item.id, component.partno
```

Estimated Cost: 19102
Estimated # of Rows Returned: 39342
Temporary Files Required For: Order By

- 1) informix.item: SEQUENTIAL SCAN
- 2) informix.component: SEQUENTIAL SCAN

DYNAMIC HASH JOIN

Dynamic Hash Filters: informix.item.id = informix.component.itemid

Case 3

To address the issue in Case 1, set the optimization mode to FIRST_ROWS. This can be done in two ways:

1. SET OPTIMIZATION **FIRST_ROWS**;
2. SELECT {+**FIRST_ROWS**} FIRST 10 <Column Name> FROM <Table Name>;

To reset or prefer optimization to ALL_ROWS, execute the following:

1. SET OPTIMIZATION **ALL_ROWS**;
2. SELECT {+**ALL_ROWS**} <Column Name> FROM <Table Name>;

Notice, as depicted in Example 3-13, that the Optimizer has chosen index path for scanning each table and joins them using a nested loop join. This enables the server to return rows quicker without waiting for the hash table to be completely built.

Example 3-13 Case 3

QUERY: (FIRST_ROWS OPTIMIZATION)

```
SELECT FIRST 10 *  
FROM    item, component  
WHERE    item.id = component.itemid  
ORDER BY item.id
```

Estimated Cost: 1604
Estimated # of Rows Returned: 39342

- 1) **informix.item: INDEX PATH**

(1) Index Keys: id scode bin (Serial, fragments: ALL)

2) informix.component: INDEX PATH

- (1) Index Keys: itemid partno partbin (Serial, fragments: ALL)
Lower Index Filter: informix.item.id=informix.component.itemid

NESTED LOOP JOIN

Case 4

With FIRST_ROWS optimization set, the optimizer chooses a different plan, even you use columns FROM different tables in the ORDER BY clause. This is depicted in Example 3-14.

Example 3-14 Case 4

QUERY: (FIRST_ROWS OPTIMIZATION)

```
SELECT SKIP 100 FIRST 10 *  
FROM    item, component  
WHERE   item.id = component.itemid  
ORDER BY item.id, component.itemid
```

Estimated Cost: 20066

Estimated # of Rows Returned: 39342

Temporary Files Required For: Order By

1) informix.component: SEQUENTIAL SCAN

2) informix.item: INDEX PATH

- (1) Index Keys: id scode bin (Serial, fragments: ALL)
Lower Index Filter: informix.item.id =

informix.component.itemid

NESTED LOOP JOIN

Similar issues follow for queries without ORDER BY clause as well.

Case 5

When you join two tables on an equality predicate, depending on the cost, IDS can choose hash join. This is demonstrated in Example 3-15. With this method, the server has to build the complete hash table on the join before returning any results. This is not suitable for interactive pagination applications.

Example 3-15 Case 5

QUERY:

SELECT FIRST 10 *
FROM item, component
WHERE item.id = component.itemid

Estimated Cost: 504

Estimated # of Rows Returned: 39342

1) informix.item: SEQUENTIAL SCAN

2) informix.component: SEQUENTIAL SCAN

DYNAMIC HASH JOIN

Dynamic Hash Filters: informix.item.id = informix.component.itemid

Case 6

With the FIRST rows optimization, optimizer chooses different access and joins method, as shown in Example 3-16.

Example 3-16 Case 6

QUERY: (FIRST_ROWS OPTIMIZATION)

SELECT FIRST 10 *
FROM item, component
WHERE item.id = component.itemid

Estimated Cost: 1468

Estimated # of Rows Returned: 39342

1) informix.component: SEQUENTIAL SCAN

2) informix.item: INDEX PATH

(1) Index Keys: id score bin (Serial, fragments: ALL)

Lower Index Filter: informix.item.id=informix.component.itemid

NESTED LOOP JOIN

3.4 Sequences

Sequences are an alternative to the serial type for numbering. The serial type does not give as much flexibility in choice of successive values or how values are used. Sequences, as they are provided in IDS, are similar to those provided by the Oracle DBMS.

Example 3-17 demonstrates how to use sequences. You must first create the sequence object using the CREATE SEQUENCE statement. This statement has a number of options that enable you to customize the sequence of numbers and the action taken when the sequence is exhausted.

Because a sequence is a database object, you must grant permission to alter the sequence to anyone who will use it. If not, those users will not be able to generate a value using the nextval method. So just as permissions are granted on tables, they are granted on sequences.

When the sequence is created and the proper users have been granted permission to alter the sequence, use the nextval method to generate numbers and the currval method to retrieve the most recently generated value. The nextval method is used as rows are inserted. This is similar in effect to what would happen if the book_num column were defined as a serial or serial8 data type and we used zero as the value in the INSERT statement.

This is not the next value to be used, it is the last value that was used. The next value will be the current value modified by the rules governing the sequence. Those rules are such as the starting value and increment defined when the sequence was created.

Example 3-17 Basic Sequence Operations

```
CREATE SEQUENCE redbook
  INCREMENT BY 1000
  START WITH 1
  MAXVALUE 99000
  NOCYCLE;
```

Sequence created.

```
GRANT ALTER ON redbook TO informix;
Permission granted.
```

```
CREATE TABLE dual (c1 INTEGER);
Table created.
```

```
CREATE TABLE books (
  book_num      INTEGER,
```

```

        title      VARCHAR(50),
        author     CHAR(15)
    ) LOCK MODE ROW
    ;
    Table created.

    INSERT INTO books VALUES (redbook.nextval, 'IDS V10', 'Anup, et.al.');
```

1 row(s) inserted.

```

    INSERT INTO books VALUES (redbook.nextval, 'Replication', 'Charles,
    et.al.');
```

1 row(s) inserted.

```

    INSERT INTO books VALUES (redbook.nextval, 'Informix Security',
    'Leffler');
```

1 row(s) inserted.

```

    INSERT INTO books VALUES (redbook.nextval, 'The SQL Language',
    'Snoke');
```

1 row(s) inserted.

```

    INSERT INTO dual VALUES (redbook.nextval);
```

1 row(s) inserted.

```

> SELECT * FROM books ;
```

book_num	1
title	IDS V10
author	Anup, et.al.
book_num	1001
title	Replication
author	Charles, et.al.
book_num	2001
title	Informix Security
author	Leffler
book_num	3001
title	The SQL Language
author	Snoke

4 row(s) retrieved.

```

SELECT * FROM dual;
```

c1

4001

1 row(s) retrieved.

> SELECT redbook.currval FROM dual;

currval
4001

1 row(s) retrieved.

> SELECT redbook.currval FROM books;

currval
4001
4001
4001
4001

4 row(s) retrieved.

> SELECT redbook.currval FROM manufact;

currval
4001
4001
4001
4001
4001
4001
4001
4001
4001
4001
4001
4001
4001
4001
4001
4001
4001
4001

17 row(s) retrieved.

Example 3-18 illustrates that you must take care in working with sequences. Before you can select the current value from a sequence, you must have the sequence defined for the session. The only way to do that is to execute the `nextval` method for the sequence object. However, that uses up one value in the sequence. However, there is no way to determine the current value without generating a value. So you might need to do this, and it might leave blanks in the sequence of numbers.

Example 3-18 Sequences and sessions

```
> TRUNCATE dual;
Table truncated.

> INSERT INTO DUAL VALUES (redbook.nextval);
1 row(s) inserted.

> SELECT redbook.currval FROM dual;

          currval
          5001

1 row(s) retrieved.

> SELECT * FROM dual;

          c1
          5001

1 row(s) retrieved.

> DISCONNECT current;
Disconnected.

> SELECT redbook.currval FROM dual;
349: Database not selected yet.
Error in line 1
Near character position 32

> DATABASE stores_demo;
Database selected.

> SELECT redbook.currval FROM dual;

          currval
```

8315: Sequence (root.redbook) CURRVAL is not yet defined in this session.
Error in line 1
Near character position 31

```
> TRUNCATE dual;  
Table truncated.  
  
> INSERT INTO dual VALUES (redbook.nextval);  
1 row(s) inserted.  
  
> SELECT redbook.currval FROM dual;
```

currval
6001

1 row(s) retrieved.

If you know the rules of the sequence, you can compute the next value from the current value. If you do not know the rules, you can retrieve useful information from the syssequences catalog, as illustrated in Example 3-19.

Example 3-19 The syssequences catalog

```
SELECT SUBSTR(a.tabname,1,20) AS table,  
       b.seqid AS sequence_id  
FROM   systables a,  
       syssequences b  
WHERE  a.tabid = b.tabid  
;
```

table	sequence_id
redbook	1

1 row(s) retrieved.

```
> SELECT * FROM syssequences WHERE seqid = 1;
```

seqid	1
tabid	132
start_val	1
inc_val	1000
min_val	1
max_val	99000
cycle	0

```
restart_val
cache      20
order      1

1 row(s) retrieved.
```

3.5 Collection data types

With the built-in data types, sometimes it is not possible to store all the data with homogeneous dependencies and characteristics in a single table without causing data redundancy. The collection data type was introduced to address this issue without causing, or at least minimally causing, data redundancy.

A collection data type is an extended complex data type that is comprised of one or more collection elements, all of the same data type. A collection element can be of any data type (including other complex types) except BYTE, TEXT, SERIAL, or SERIAL8. Figure 3-4 illustrates the hierarchy of the collection data type.

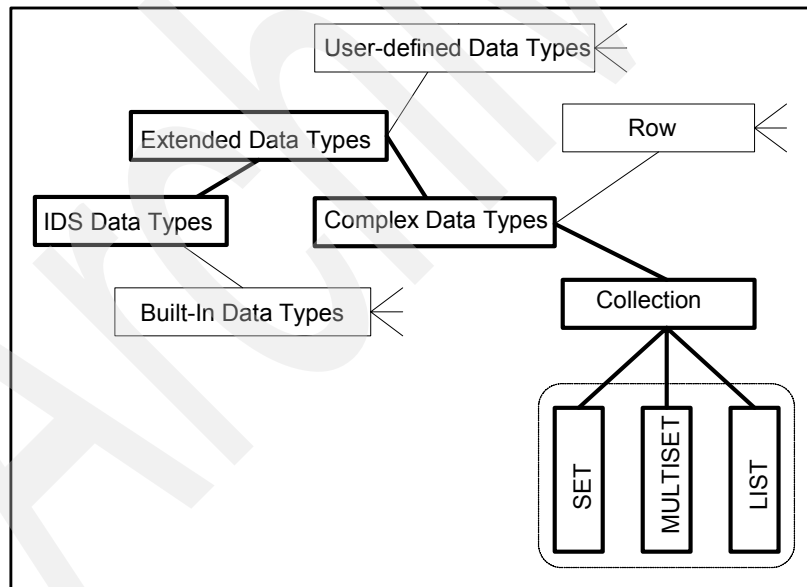


Figure 3-4 IDS collection data types

3.5.1 Validity of collection data types

One of the main restrictions enforced, when a collection type is declared, is that the data value in that collection should all be of the same data type. There is however an exception to this rule. For example, when there is data with character strings of the type VARCHAR (255 or less bytes) but other elements are longer than 255 bytes. In such a case you can assign the string as CHAR or cast the string a LVARCHAR as shown in Example 3-20.

Example 3-20 Casting with LVARCHAR for collection type

```
LIST {  
    'String more than 255 bytes ...',  
    'A small character string',  
    'Another very long string ...'}::LIST (LVARCHAR NOT NULL)
```

The second restriction is that the collection type not be of NULL value. This can be enforced during the schema definition. Example 3-21 illustrates some approaches for implementing a collection schema.

Example 3-21 Collection definition

```
CREATE TABLE figure (  
    id          INTEGER,  
    vertex      SET(INT NOT NULL))  
  
CREATE TABLE numbers (  
    id          INTEGER PRIMARY KEY,  
    primes      SET ( INTEGER NOT NULL ),  
    evens       LIST ( INTEGER NOT NULL ),  
    twin_primes LIST ( SET ( INTEGER NOT NULL ) NOT NULL ))  
  
CREATE TABLE dress (  
    type        INTEGER,  
    location    INTEGER,  
    colors      MULTISET ( ROW ( rgb INT ) NOT NULL ))
```

It is invalid to have NULL values in collection data. As per syntax the NULL values will not be allowed in a collection data type. And if it is attempted, an error is returned.

Example 3-22 illustrates how the syntax definition itself enforces the NOT NULL constraint. In Query 1, we try to declare a collection data type SET in such a way that NULL data can be inserted. That is, however, rejected by the parser indicating an invalid syntax. This is one of the most efficient ways to implement NOT null in a collection data type. Query 2 implements the schema definition for SET collection type using the correct method.

Example 3-22 Implementation of collection type

```
> -- Query 1
> CREATE TABLE figure ( id INTEGER, vertex SET(INT));

201: A syntax error has occurred.
Error in line 1
Near character position 49

> -- Query 2
> CREATE TABLE figure ( id INTEGER, vertex SET(INT NOT NULL));

Table created.
```

In Example 3-23, we try to explore the ways of inserting valid data using the table figure defined in Example 3-22.

Statements 1 and 2 insert regular and valid SET data into the table. Statement 3 tries to insert a NULL tuple in the SET data. Because by definition NULL value data are not allowed in any collection type, the parser gives an error for that insertion. Statement 5 inserts a collection SET with no values. Then Statement 6 addresses an issue that has been presented many times to IBM technical support.

As per the design specification of collection types, it is not possible to have the tuples inside the SET/LIST/MULTISET to be NULL. However, it is possible to have the column representing the collection type, SET/LIST/MULTISET, to be set to NULL. this means that there is no data available for that collection type. Statement 6 illustrates the contents of the table after all the data is inserted. It is possible to eliminate NULL altogether from the collection data type by creating the schema as shown in Statement 7, and supported by statements 8 and 9.

The collection set itself can be NULL, but the elements inside the set cannot be NULL (NULL means unknown). The complete SET itself can be unknown, but a known set cannot have unknown elements in it.

Example 3-23 Inserting data

```
>-- Statement 1
> INSERT INTO figure VALUES ( 1, SET { 1, 2 } );

1 row(s) inserted.

>-- Statement 2
> INSERT INTO figure VALUES ( 2, SET { 11, 12, 13 } );

1 row(s) inserted.

>-- Statement 3
> INSERT INTO figure VALUES ( 4, SET { NULL } );

9978: Insertion of NULLs into a collection disallowed.
Error in line 1
Near character position 44

>-- Statement 4
> INSERT INTO figure VALUES ( 3, NULL );

1 row(s) inserted.

>-- Statement 5
> INSERT INTO figure VALUES ( 4, SET{});

1 row(s) inserted.

>-- Statement 6
> SELECT * FROM figure;
```

id	1			
vertex	SET{1	,2		}
id	2			
vertex	SET{11	,12	,13	}
id	3			
vertex				
id	4			
vertex	SET{}			

5 row(s) retrieved.

```
>-- Statement 7
> CREATE TABLE gofigure (
    id      INTEGER,
    vertex SET(INT NOT NULL) NOT NULL)
;
Table created.
```

```
>-- Statement 8
> INSERT INTO gofigure VALUES ( 3, NULL );

391: Cannot insert a null into column (gofigure.vertex).
Error in line 1
Near character position 38
```

```
>-- Statement 9
> INSERT INTO gofigure VALUES ( 3, SET{NULL});

9978: Insertion of NULLs into a collection disallowed.
Error in line 1
Near character position 42
```

Example 3-24 illustrates valid declarations of the collection data type.

Example 3-24 Valid Collection type declaration

SET {}

LIST {'First Name', 'Middle Name', 'Last Name'}

MULTISET{5, 9, 7, 5}

SET {3, 5, 7, 11, 13, 17, 19}

LIST(SET{3,5}, SET{7,11}, SET{11,13})

3.5.2 LIST, SET, and MULTISET

The Informix Dynamic Server supports three built-in collection types: LIST, SET, and MULTISET. Collection data types are a subset of extended complex data type, as represented in Figure 3-4. The collection type can use any type except

TEXT, BYTE, SERIAL, or SERIAL8. And these collection types can be nested by using elements of a collection type. The following are descriptions and examples of the three built-in collection types.

SET

A SET is an unordered collection of elements, each of which has a unique value. Define a column as a SET data type when you want to store collections whose elements contain no duplicate values and have no associated order.

Example 3-25 shows some valid SET data values that can be entered. Because the order of data is not of significance in SET data type, the data Sets 1 and 2 are identical, so are 3 and 4, and 5 and 6. Care must be taken so that the tuples inside the SET data are not repeated or duplicated.

Example 3-25 SET data

-
1. SET{1, 5, 13}
 2. SET{13, 5, 1}
 3. SET{"Pleasanton", "Dublin", "Fremont", "Livermore"}
 4. SET{"Livermore", "Fremont", "Dublin", "Pleasanton"}
 5. SET{"Anup", "Mehul", "Rishabh"}
 6. SET{"Rishabh", "Anup", "Mehul"}
-

MULTISET

A MULTISET is an unordered collection of elements in which elements can have duplicate values. A column can be defined as a MULTISET collection type when you want to store collections whose elements might not be unique and have no specific order associated with them.

Example 3-26 shows some valid MULTISET data values that can be entered. As with the SET type, with a MULTISET type the order of data is not of significance. Two MULTISET data values are equal if they have the same elements, even if the elements are in different positions within the set. The multiset values 1 and 2 are not equal however if the data set 3 and 4 are equal, even if there are duplicates. Although the MULTISET type is similar to an extension of SET type, unlike the SET type duplicates are allowed in MULTISET type.

Example 3-26 MULTISET data

-
1. MULTISET{"Pleasanton", "Dublin", "Fremont", "Livermore"}
 2. MULTISET{"Pleasanton", "Dublin", "Fremont"}
 3. MULTISET{"Anup", "Mehul", "Rishabh", "Anup"}
 4. MULTISET{"Rishabh", "Anup", "Anup", "Mehul"}
-

LIST

A LIST is an ordered collection of elements that can include duplicate elements. A LIST differs from a MULTISET in that each element in a LIST collection has an ordinal position in the collection. You can define a column as a LIST collection type when you want to store collections whose elements might not be unique but have a specific order associated with them.

Two list values are equal if they have the same elements in the same order. In Example 3-27, all are list values. Data set 1 and 2 are not equal because the values are not in the same order. To be equal, the data sets have to be in the same order, as are 3 and 4.

Example 3-27 LIST data

-
1. LIST{"Pleasanton", "Dublin", "Livermore", "Pleasanton"}
 2. LIST{"Pleasanton", "Dublin", "Fremont", "Dublin"}
 3. LIST{"Pleasanton", "Dublin", "Livermore"}
 4. LIST{"Pleasanton", "Dublin", "Livermore"}
-

Summary of data in collection type constructors

Table 3-1 is a snapshot of the type of data representation for each of the collection data types.

Table 3-1 Snapshot™ of data types

Type Constructor	Are Duplicates Allowed	Is Data Ordered
SET	NO	NO
MULTISET	YES	NO
LIST	YES	YES

Restrictions

The following are some of the restrictions for collection data types.

- ▶ In a collection data type an element cannot have a NULL value.
- ▶ Collection data types are not valid as arguments to functions that are used for functional indexes.
- ▶ As a general rule, collection data types cannot be used as arguments in the following aggregate functions:
 - AVG
 - SUM
 - MIN
 - MAX

- ▶ A user-defined routine (UDR) cannot be made parallelizable if it accepts a collection data types as an argument.
- ▶ You cannot use a CREATE INDEX statement to create an index on collection.
- ▶ You cannot create a functional index for a collection column.
- ▶ User-defined cast cannot be performed on a collection data type.

3.6 Distributed query support

IBM Informix Dynamic Server supports distributed queries. This allows Data Manipulation Language (DML) and Data Definition Language (DDL) queries to access and create data across databases on a single IDS instance, or on multiple IDS instances. These multiple server instances can be on the same machine, or different machines in the network. All databases accessed by distributed queries should have the same database mode: ANSI, logging or non-logging. An example application in this environment is depicted in Figure 3-5.

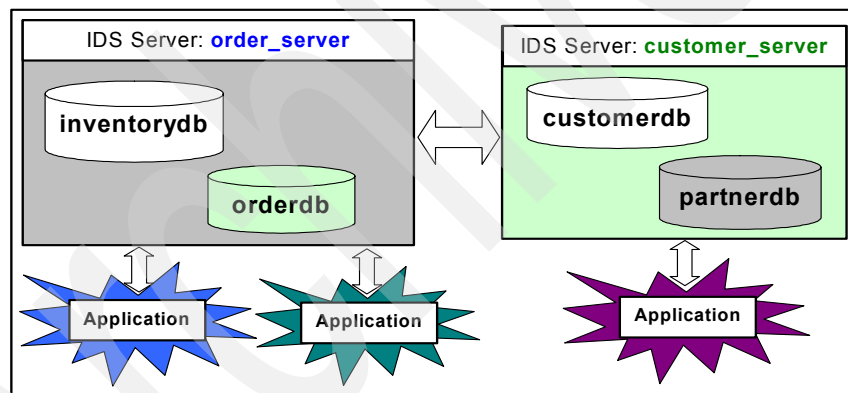


Figure 3-5 Sample application

3.6.1 Types and models of distributed queries

In this section, we define some of the methods used in distributed queries.

Cross database query

Databases that reside on the same server instance are classified as Cross Databases. If a query involves databases residing on the same server instance, the query is called a cross database query. A sample cross database query is depicted in Example 3-28.

Example 3-28 Cross database

```
SELECT *
FROM   orderdb:ordertab o,
       inventorydb:itemtab i
WHERE  o.item_num = i.item_num
```

Cross server query

Databases residing on different server instances are classified as Cross Server databases. If query involves databases residing on different server instances it is called a cross server query. A sample cross database query is depicted in Example 3-29.

Example 3-29 Cross Server

```
SELECT *
FROM   orderdb@order_server:ordertab o,
       customerdb@customer_server:itemtab c
WHERE  o.cust_num = c.cust_num
```

Local database

A local database refers to client-connected database, namely the database to which the client is connected when it starts its operation.

Remote database

A remote database refers to a database which is accessed through queries from the local database. For example, the client connects to a database which becomes the local database for the client. From the client local database the client then issues a query which has a table that does not reside on the local database, but rather resides on some other database. This other database is called as a Remote database. The remote database can further be classified as a Cross Database database or a Cross Server database.

The distributed query model supports the following:

- ▶ DML queries
 - INSERT
 - SELECT
 - UPDATE
 - DELETE
- ▶ DDL statements
 - CREATE VIEW with remote table/view references
 - CREATE SYNONYM with remote table/view references
 - CREATE DATABASE with remote server references

- ▶ Miscellaneous statements
 - DATABASE
 - EXECUTE PROCEDURE / FUNCTION
 - LOAD / UNLOAD
 - LOCK / UNLOCK
 - INFO
- ▶ Data types supported by IDS are depicted in Figure 3-6. These are data types that were already supported by distributed queries prior to IDS V10.

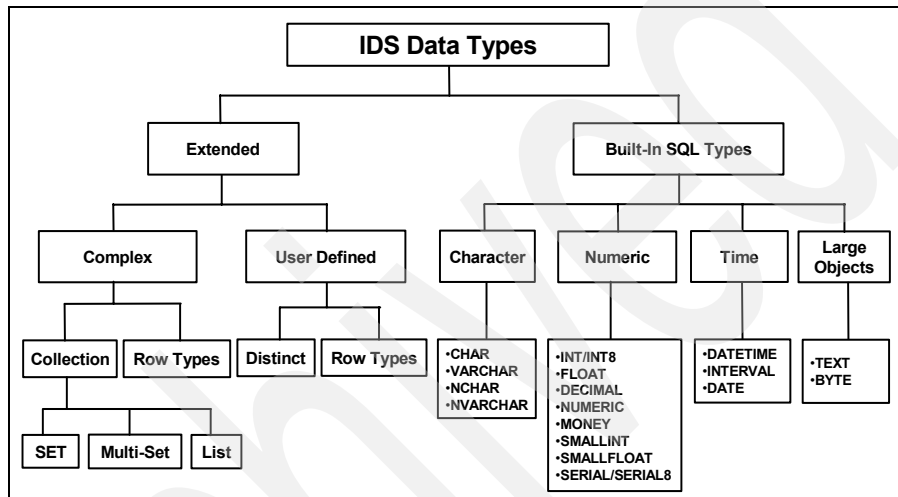


Figure 3-6 Data types supported in distributed queries before V10

3.6.2 The concept

Built-in user-defined types (UDTs) were introduced in IDS V9. Built-in UDT Data type refers to BOOLEAN, LVARCHAR, BLOB, CLOB data types which are implemented by the IBM Informix Server as internal server defined UDTs. You can also define your own UDTs, which are simply referred to as UDTs.

Some of the system catalog tables that use built-in SQL types in the IDS V7 server were re-designed to use built-in UDTs. Typically clients have applications and tools that need to access system catalog tables across databases. Because the IDS V9 server does not allow access to built-in UDT columns across databases, these client applications and tools did not work properly after the migration to IDS V9.

IDS V10 has extended the data type support for distributed queries. In addition to built-in data types, built-in UDTs are now also supported for access across databases in a single server instance (cross database queries). Support has also

been extended to DISTINCT types of built-in data types or built-in UDTs. In addition, UDTs are supported if they have casts to built-in data types or built-in UDTs.

All references to operations or objects across databases, will hereafter, in this discussion, refer to cross databases on a single server instance and not to cross databases across multiple servers. A local database refers to client-connected database, while a remote database refers to databases which are accessed through queries from the local database.

Note: All references to operations or objects across databases, hereafter in this section will refer to cross databases on a single server instance and not to cross databases across multiple servers.

3.6.3 New extended data types support

The following new datatype support was implemented in IDS V10.

- ▶ Built-in UDTs across databases in a single server instance:
 - BOOLEAN
 - LVARCHAR
 - BLOB
 - CLOB
 - SELFUNCARGS
 - IFX_LO_SPEC
 - IFX_LO_STAT
 - STAT
- ▶ Distinct's of built-in data types
- ▶ Distinct's of built-in UDTs
- ▶ Non built-in UDTs (distinct and opaque types) across databases in a single server instance, with explicit cast to built-in data types or built-in UDTs.

Of these new extended data types BOOLEAN, LVARCHAR, BLOB and CLOB are the most commonly used in applications.

3.6.4 DML query support

SELECT, INSERT, UPDATE, and DELETE DML commands are supported by all distributed queries. To illustrate the use of these DML commands we make use of the schema shown in Example 3-30.

Example 3-30 The schema

```
CREATE DATABASE orderdb WITH LOG;

{
http://www.ibm.com/developerworks/db2/zones/informix/library/techarticle/db\_shapes3.html
CREATE USER UDT TYPE myshape
CREATE EXPLICIT CAST OF myshape TO BUILT-IN TYPE VARCHAR AND LVARCHAR
}

CREATE TABLE ordertab (
    order_num      INT,
    item_num       INT,
    order_desc     CLOB,
    item_list      LVARCHAR(300),
    order_route    BLOB,
    order_shelf_life BOOLEAN,
    order_lg_pk    myshape,
    order_sm_pk    myshape
);

INSERT INTO ordertab VALUES (
    "1",
    "1",
    FILETOCLOB('canned.doc', 'client'),
    "Canned Fish, Meat, Veges",
    FILETOBLOB('cust1.map', 'client'),
    "F",
    "box(1,1,2,2)"
);

INSERT INTO ordertab VALUES (
    "2",
    "1",
    FILETOCLOB('fresh.doc', 'client'),
    "Fresh Fruits, Veges",
    FILETOBLOB('cust2.map', 'client'),
    "T",
    "box(2,2,2,2)"
);

CREATE DATABASE inventorydb WITH LOG;

CREATE TABLE itemtab (
```

```

        item_num    INT,
        item_desc   LVARCHAR(300),
        item_spec   CLOB,
        item_pic    BLOB,
        item_sm_pk  myshape,
        item_lg_pk  myshape,
        item_life   BOOLEAN
        item_qty    INT8
    );

    INSERT INTO itemtab VALUES (
        1,
        "High Seafood",
        FILETOCLOB('tuna.doc', 'client'),
        FILETOBLOB('tuna.jpg', 'client'),
        "box(1,1,2,2)",
        "box(6,6,8,8)" ,
        "F");

    INSERT INTO itemtab VALUES (
        2,
        "Magnolia Foods",
        FILETOCLOB('apple.doc', 'client'),
        FILETOBLOB('apple.jpg', 'client'),
        "box(1,1,2,2)",
        "box(6,6,8,8)" ,
        "T"
    );

    CREATE FUNCTION instock(item_no INT, item_d LVARCHAR(300)) RETURNING
    BOOLEAN
        DEFINE count INT8;

        SELECT item_qty
        INTO    count
        FROM    itemtab
        WHERE  item_num = item_no AND item_desc = item_d;

        IF count > 0 THEN RETURN "t";
        ELSE RETURN "f"
        END IF
    END FUNCTION;

    CREATE FUNCTION item_desc(item_no) RETURNING LVARCHAR(300)
        RETURN SELECT item_desc

```

```

        FROM    itemtab
        WHERE   item_num = item_no;
END FUNCTION;

CREATE PROCEDURE mk_perisable(item_no INT, perish BOOLEAN)
    UPDATE itemtab
    SET     item_life = perish
    WHERE   item_num = item_no;
END PROCEDURE;

```

SELECT

All built-in UDTs are supported for cross database select queries. Cross database references of columns in column list, predicates, subqueries, UDR parameters and return types can be built-in UDTs. Direct reference to user created UDTs is not allowed. However you can use a UDT in cross database queries, if it is defined in all participating databases and explicitly cast to built-in types. The cast and the cast functions must exist in participating databases. This is depicted in Example 3-31.

Example 3-31 Simple query on remote table having built-in UDTs

```

DATABASE orderdb;

SELECT item_num,
       item_desc,
       item_specs,
       item_pic,
       item_life
FROM   inventorydb:itemtab
WHERE  item_num = 1;

```

Example 3-32 shows a query on a remote table with a UDT.

Example 3-32 Simple query on remote table having UDTs

```

DATABASE orderdb;

SELECT item_sm_pk::LVARCHAR,
       item_lg_pk::varchar
FROM   inventorydb:itemtab
WHERE  item_num = 2;

```

INSERT

Only BOOLEAN, LVARCHAR, BLOB and CLOB built-in UDTs are supported for Insert statements. Inserting remote table BLOB and CLOB columns will remain the same as inserting into a local table. A Large Object (LO) will be created in a SMART BLOB Space and the corresponding row in table space will contain a pointer to the LO. User created UDTs will be supported through explicit casting, the UDTs and their cast should exist in all databases referenced in the insert statement.

Inserting into a remote table having built-in UDTs is depicted in Example 3-33.

Example 3-33 Insert on Remote table having built-in UDTs

```
DATABASE orderdb;

INSERT INTO inventorydb:itemtab
(
    item_life,
    item_desc,
    item_pic,
    item_spec
)
VALUES
(
    't',
    "fresh large spinach",
    filetoblob('spinach.jpg', 'client'),
    filetoclob('spinach.doc', 'client')
)
```

Example 3-34 show inserting into a remote table when selecting from a local table.

Example 3-34 Insert into remote table Select from local table built-in UDTs

```
DATABASE orderdb;

INSERT INTO inventorydb:itemtab
(
    item_life,
    item_desc,
    item_pic,
    item_spec
)
SELECT shelf_life, order_list, order_route, order_desc
FROM ordertab
WHERE order_num > 1;
```

Example 3-35 shows inserting in a remote table with non built-in UDTs.

Example 3-35 Insert on remote table non built-in UDTs

```
DATABASE orderdb;

INSERT INTO inventorydb:itemtab (item_sm_pk)
  SELECT order_lg_pk::lvarchar
  FROM   ordertab
  WHERE  order_sm_pk::lvarchar = "box(6,6,8,8)";
```

UPDATE

Only BOOLEAN, LVARCHAR, BLOB and CLOB built-in UDTs are supported from the newly extended supported types. User created UDTs will be supported through explicit casting to built-in types or built-in UDTs. These UDTs and their casts should exist in all databases participating in the update statement. Any updates to BLOB/CLOB columns will be reflected immediately in the cross database tables.

Example 3-36 shows updating of a remote table having built-in UDTs.

Example 3-36 Update on remote tables having built-in UDTs

```
DATABASE orderdb;

UPDATE inventorydb:itemtab
SET
(
  item_life,
  item_desc,
  item_pic,
  item_spec
) =
(
  "f",
  "Fresh Organic Broccoli",
  Filetoblob('broccoli.jpg', 'client'),
  Filetoclob('broccoli.doc', 'client')
)
WHERE item_num = 3;
```

Example 3-37 shows updating of a remote table with non built-in UDTs.

Example 3-37 Update on remote tables having non built-in UDTs

```
DATABASE orderdb;

UPDATE inventorydb:itemtab
SET (item_lg_pk) =
((
    SELECT order_lg_pk::lvarchar
    FROM   ordertab
    WHERE  order_sm_pk::lvarchar = "box(2,2,2,2)"
))
WHERE order_shelf_life = "t";
```

DELETE

All of the newly extended supported types work with the delete statement. Non built-in UDTs are also be supported through an explicit casting mechanism. Non built-in UDTs and their cast should exist in all databases participating in the delete statement. A delete from a BLOB/CLOB remains the same as on a local database. Deleting a row containing BLOB/CLOB column deletes the handle to the Large Object (LO). At transaction commit, the IDS server checks and deletes the LO if the reference and open count to it is zero. This example is depicted in Example 3-38.

Example 3-38 Delete from Remote table with built-in UDTs

```
DATABASE orderdb;

DELETE FROM inventorydb:itemtab
WHERE item_life = "t" AND
      item_desc = "organic squash" AND
      item_pic IS NOT NULL AND
      item_spec IS NULL
```

Example 3-39 shows deleting from a remote table with non built-in UDTs.

Example 3-39 Delete from remote table with non built-in UDTs

```
DATABASE orderdb;

DELETE FROM inventorydb:itemtab
WHERE item_lg_pk::lvarchar = "box(9,8,9,8)"
```

3.6.5 DDL queries support

For distributed databases, CREATE VIEW, CREATE SYNONYM and CREATE DATABASE have been modified to handle remote table/view references. The use of CREATE VIEW and CREATE SYNONYM are explained in the following sections.

CREATE VIEW

All of the newly extended types are supported for a create view statement on local database having built-in UDT columns on cross database tables. This means that queries in a create view can now have a remote reference to built-in UDT columns of remote tables. User created UDTs will have to be explicitly cast to built-in UDTs or built-in types for view creation support. All non built-in UDTs and their casts should exist on all databases in the create view statement. The view will only be created on the local database. This example is depicted in Example 3-40.

Example 3-40 CREATE VIEW with built-In UDT columns

```
DATABASE orderdb;

CREATE VIEW new_item_view AS
  SELECT item_life,
         item_desc,
         item_pic,
         item_spec
  FROM inventorydb:itemtab
```

Example 3-41 shows creating a view with non built-in UDT columns.

Example 3-41 CREATE VIEW with non built-in UDT columns

```
DATABASE orderdb;

CREATE VIEW item_pk_view AS
  SELECT item_lg_pk::lvarchar,
         item_sm_pk::varchar
  FROM   inventorydb:itemtab
  WHERE  item_sm_pk::char matches "box(2,2,2,2)*"
```

CREATE SYNONYM

Creation of a synonym on the local database for across database or across server tables was previously only supported for tables having built-in types. Now the create synonym statement is supported for all cross database remote tables

on a single server having built-in UDTs. The synonym created will be in the local database. You cannot create a synonym on the remote database from the local database.

Example 3-42 shows creating a synonym on a remote table with built-in UDTs.

Example 3-42 CREATE SYNONYM on remote table having built-in udt's

```
DATABASE inventorydb;

CREATE TABLE makertab
(
    item_num          INT,
    mk_whole_retail    BOOLEAN,
    mk_name            LVARCHAR
);

DATABASE orderdb;

CREATE SYNONYM makersyn FOR inventorydb:makertab;
```

3.6.6 Miscellaneous query support

Other prominent queries supported are DATABASE, LOAD/UNLOAD, LOCK/UNLOCK, INFO, and CREATE TRIGGER. The use of EXECUTE PROCEDURE / FUNCTION and CREATE TRIGGER are explained in detail in this section.

EXECUTE procedure

The Server now supports implicit and explicit execution of UDRs having built-in UDTs. The UDR might have built-in UDTs as parameters or as return types, and also might either be a function or a stored procedure. The UDR might have UDT return types or parameters and the caller can invoke it with the return types and parameters explicitly cast to built-in SQL type or built-in UDTs. UDRs written in SPL, C and Java will support these extended data types. UDRs having OUT parameters of built-in UDTs will also now be supported.

Consider the following set of statements:

```
CONNECT TO orderdb@order_server;

EXECUTE FUNCTION cardb:gettopcars();

SELECT inventorydb:getinv(a.parts)
FROM   cardb:partstable a
WHERE  a.partid = 12345;
```

The current database, for the execution of the two functions, `gettopcars()` and `getinv()`, will be the database they were created with. That is, `cardb` for `gettopcars()` and `inventorydb` for `getenv()`. All default table references from those routines will come from respective databases.

Implicit execution

When a UDR is used in a projection list or as a predicate of a query, or when it is called to convert a function argument from one data type to another, the execution is called an implicit execution. When an operator function for a built-in UDT is executed, that is also called an implicit execution. The IDS functionality now supports implicit execution of UDRs with built-in UDTs. Non built-in UDTs must be cast to built-in types or built-in UDTs for use in UDT execution.

This implicit execution, for select queries, is shown in Example 3-43.

Example 3-43 Implicit execution of select queries

```
DATABASE orderdb;

SELECT inventorydb:instock(item_num, item_desc)
FROM   ordertab;

SELECT inventorydb:instock(item_num, item_desc)
FROM   ordertab
WHERE  inventorydb:item_desc(item_num) = "organic produce";
```

Example 3-44 shows the implicit execution of update, delete, and insert statements.

Example 3-44 Implicit usage in update, delete, and insert statements

```
DATABASE orderdb;

UPDATE ordertab
SET item_shelf_life = inventorydb:mkperishable(item_num, 't')
WHERE inventorydb:instock(item_num, item_desc) = 't';
```

```

INSERT ordertab
(
    order_shelf_life,
    item_list
)
VALUES
(
    inventorydb:instock(item_num, item_desc),
    inventorydb:item_desc(item_num)
)

DELETE FROM ordertab
WHERE inventorydb:mkperisable(item_num, order_shelf_life) = "f";

```

Example 3-45 shows the implicit execution of a built-in UDT function.

Example 3-45 Implicit execution of built-in UDT function ifx_boolean_equal

```

SELECT *
FROM   ordertab, inventorydb:itemtab et
WHERE  order_shelf_life = et.item_life;

```

Explicit execute

When a function or procedure is executed using an execute function or an execute procedure statement, the type of execution is called explicit execution. Built-in UDTs are now supported for explicit execution of UDRs. They can be in the parameter list or the return list. Non-built in UDTs are supported with explicit casting.

Example 3-46 shows an explicit execution using EXECUTE PROCEDURE.

Example 3-46 Explicit usage in EXECUTE PROCEDURE

```

EXECUTE PROCEDURE inventorydb:mkperisable(2, "t");

```

Example 3-47 shows an explicit execution using EXECUTE FUNCTION.

Example 3-47 Explicit usage in EXECUTE FUNCTION

```

EXECUTE FUNCTION inventorydb:instock(6, "yogurt");

```

Trigger actions

With this added functionality, TRIGGER statements can now refer to built-in UDTs and non built-in UDTs that span across databases. This new data type support can be used in the action clauses as well as in the WHEN clause. The

action statements can be any DML query or an EXECUTE remote procedure statement that refers to an across database built-in UDT. Non built-in UDTs must cast to built-in UDTs or built-in types for this functionality to work.

Example 3-48 shows the statements to insert a TRIGGER statement.

Example 3-48 Simple insert TRIGGER

```
CREATE TRIGGER new_order_audit

INSERT ON ordertab REFERENCING NEW AS post
FOR EACH ROW
(EXECUTE PROCEDURE inventorydb:mkperisable
(new. item_num, new.order_shelf_life))
```

Example 3-49 shows the statements to update a TRIGGER.

Example 3-49 Simple update TRIGGER

```
CREATE TRIGGER change_order_audit

UPDATE OF order_desc on ordertab
REFERENCING OLD AS pre NEW AS post
FOR EACH ROW WHEN ( pre.order_shelf_life = "t" )
(UPDATE inventorydb:itemtab

SET item_life = pre.order_shelf_life)
```

3.6.7 Distinct type query support

All distinct types of built-in UDTs and built-in data types are supported for cross database select queries. Distinct types can be cross database references of columns in column list, predicates, sub queries, UDR parameters and return types. The distinct types, their hierarchies, and their cast should all be the same across all the cross databases in the query. Example 3-50 shows a select for orders weighing more than the item weight for the same item number.

Example 3-50 Select using distinct types

```
DATABASE inventorydb;

CREATE DISTINCT TYPE pound AS FLOAT;

CREATE DISTINCT TYPE kilos AS FLOAT;

CREATE FUNCTION ret_kilo(p pound) RETURNS kilos;
  DEFINE x float;
```

```

    LET x = p::float / 2.2;

    RETURN x::kilos;
END FUNCTION;

CREATE FUNCTION ret_pound(k kilos) RETURNS pound;
    DEFINE x float;
    LET x = k::float * 2.2;
    RETURN x::pound;
END FUNCTION;

CREATE implicit cast (pound as kilos with ret_kilo);

CREATE explicit cast (kilos as pound with ret_pound);

ALTER TABLE itemtab ADD (weight pound);

DATABASE orderdb;

CREATE DISTINCT TYPE pound AS float;

CREATE DISTINCT TYPE kilos AS float;

CREATE FUNCTION ret_kilo(p pound) RETURNS kilos;
    DEFINE x float;
    LET x = p::float / 2.2;
    RETURN x::kilos;
END FUNCTION;

CREATE FUNCTION ret_pound(k kilos) RETURNS pound;
    DEFINE x float;
    LET x = k::float * 2.2;
    RETURN x::pound;
END FUNCTION;

CREATE IMPLICIT CAST (pound AS kilos WITH ret_kilo);

CREATE EXPLICIT CAST (kilos AS pound WITH ret_pound);

ALTER TABLE ordertab ADD (weight kilos);

SELECT x.order_num, x.weight
FROM   orderdb:ordertab x
       inventorydb:itemtab y
WHERE  x.item_num = y.item_num AND
       x.weight > y.weight      -- compares pounds to kilos.
GROUP BY x.weight;

```

3.6.8 In summary

With this data type support extension, built-in UDTs can be treated the same as built-in data types, and can be used in distributed queries where built-in data types are allowed. Also, user created UDTs support enables their use in distributed queries. When you query across multiple databases, the database server opens the new database for the same session, without closing the database you are already connected to. In a single session, keep the number of databases you access within a single database server to 8 or less. However, this limitation does not apply when you go across multiple database servers. Then each server can open up to 8 databases for each connection.

The extended support for built-in UDRs and UDTs can be used in many different ways, and in operations such as subqueries and joins. There are no changes in transaction semantics such as `BEGIN WORK` or `COMMIT WORK`. This support will contribute significantly in creating distributed queries using built-in UDTs.

3.7 External Optimizer Directives

The Optimizer Directives are those keywords that can be used inside DML to partially or fully specify the query plan of the optimizer. Prior to IDS V10, the optimizer directives were embedded in the DML itself. This, at times, could present a problem because when the application executed, the DML the query would always choose the optimizer directive as specified in the DML. This could potentially result in making it inefficient for those data sets that need not follow the directives specified in the DML. And at times it was desirable to influence the selectivity of the optimizer path. To enable these functionality, the concept of external optimizer directives was introduced in IDS V10.

Important: IDS server re-versioning to a lower version is not allowed if there are external directives defined in the server. The external directives must be deleted before reverting to an lower level IDS version.

3.7.1 What are external optimizer directives

If a DML query starts performing poorly, the optimizer path might need to be altered. For small and very simple DML, the query itself can be altered to attain the performance desired. However, for relatively huge or complex DML it might not be possible. There might also be situations where the DBA prefers to have a short-term solution without rewriting the DML. In these situations, having external optimizer directives can be helpful. This feature provides a more flexible way of specifying optimizer directives and optimizer hints.

Unlike embedded optimizer directives, the external optimizer directives are stored as a database object in the sysdirectives system catalog table. This object can be created and stored in the system table by a DBA. Variables such as IFX_EXTDIRECTIVES (environment variable for client) and EXT_DIRECTIVES (ONCONFIG parameter for server) are used to enable or disable the use of the external directive. This is explained in more detail in the following sections. The structure of sysdirectives system catalog table is shown in Table 3-2.

Table 3-2 Sysdirectives system catalog table structure

Column Name	Column Type	Comments
id	SERIAL	Unique code identifying the optimizer directive. There is unique index on this column
query	TEXT	Text of the query as it appears in the application. NULL values are not allowed
directive	TEXT	Text of the optimizer directive, without comments
directivecode	BYTE	Blob of the optimizer directive
active	SMALLINT	Integer code that identifies whether this entry is active (= 1) or test only (= 2)
hashcode	INT	For internal use only

External directives are for occasional use only. The number of directives stored in the sysdirectives catalog should not exceed 50. A typical enterprise will only needs perhaps from 0 to 9 directives.

3.7.2 Parameters for external directives

To enable the external directives, there are two variables to use, namely EXT_DIRECTIVES and IFX_EXTDIRECTIVES.

EXT_DIRECTIVES: The ONCONFIG parameter

To enable or disable use of the external SQL directives you must first set the EXT_DIRECTIVES configuration parameter in the ONCONFIG configuration file. By default, the server considers call external directives as disabled. You must explicitly use the EXT_DIRECTIVES parameter in the ONCONFIG file to make use of the external directives. Table 3-3 shows the values for the EXT_DIRECTIVES parameter.

Table 3-3 *EXT_DIRECTIVES* Values

Value	Comments
0	External Directives is disabled irrespective of client setting. This the default value.
1	Enabled for a session. Must be explicitly requested by client through IFX_EXTDIRECTIVES=1
2	Enabled for all sessions unless explicitly disabled by client through IFX_EXTDIRECTIVES=0

Any other value for EXT_DIRECTIVES in ONCONFIG file other than 1 and 2 are interpreted same as 0 by the server.

Important: The database server must be shutdown and restarted for this configuration parameter to take effect.

IFX_EXTDIRECTIVES: Client environment variable

The environment variable IFX_EXTDIRECTIVES supplements the configuration variable EXT_DIRECTIVES. This variable is to be used on the client-side to enable or disable the use of external optimizer directives. The values for this variable are depicted in Table 3-4.

Table 3-4 *IFX_EXTDIRECTIVES* Values

Value	Comments
0	Disable for this client, irrespective of server setting.
1	Enable, unless server has disabled external directives.

Any other value for the environment variable IFX_EXTDIRECTIVES file other than 1 has the same interpretation of 0.

Table 3-5 gives a snapshot of the inter-relation for different values of EXT_DIRECTIVES and IFX_EXTDIRECTIVES variables.

Table 3-5 Status of external directives as per parameter setting

EXT_DIRECTIVES	IFX_EXTDIRECTIVES (Not Set)	IFX_EXTDIRECTIVES = 0	IFX_EXTDIRECTIVES = 1
(Not Set)	OFF	OFF	OFF
0	OFF	OFF	OFF
1	OFF	OFF	ON
2	ON	OFF	ON

3.7.3 Creating and saving the external directive

Before using any external directive for optimization purpose it must be created and saved in the sysdirectives table. A new SQL statement, SAVE EXTERNAL DIRECTIVES, was introduced for this purpose.

SAVE EXTERNAL DIRECTIVES is used to create and register the optimizer directive in the sysdirectives table. The syntax is as shown in Figure 3-7.

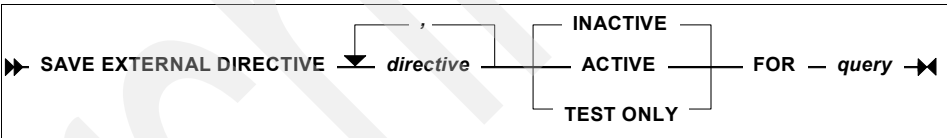


Figure 3-7 SAVE EXTERNAL DIRECTIVES syntax

The keywords: ACTIVE, INACTIVE, TEST ONLY

While defining the directive, at least one of the keywords, ACTIVE, INACTIVE or TEST ONLY must be specified. The execution the external directive depends on these keywords. Table 3-6 shows the interpretation of each of these keywords.

Table 3-6 Interpretation of ACTIVE, INACTIVE, TEST ONLY keywords

Keywords	Description
ACTIVE	Enable: apply directive to any subsequent query matching the directive query sting
INACTIVE	Disable: The directive becomes dormant, with out any effect to queries matching the directive query string in sysdirectives.
TEST ONLY	Restrict Scope: Available for DBA and informix user

Example 3-51 shows sample code for creating an external directive.

Example 3-51 Creating external directive

```
CREATE TABLE monitor (  
    modelnum    INTEGER,  
    modeltype   INTEGER,  
    modelname   CHAR(20)  
);  
  
SAVE EXTERNAL DIRECTIVES {+INDEX(monitor, modelnum) }  
ACTIVE FOR  
    SELECT {+INDEX(monitor, modelnum) } modelnum  
    FROM   monitor  
    WHERE  modelnum<100;
```

The corresponding entry in the sysdirectives table for Example 3-51 is depicted in Example 3-52.

Example 3-52 The sysdirectives entry

```
> SELECT * FROM sysdirectives;
```

id	2
query	SELECT {+INDEX(monitor, modelnum) } modelnum FROM monitor WHERE modelnum<100
directive	INDEX(monitor, modelnum)
directivecode	<BYTE value>
active	1
hashcode	598598048

1 row(s) retrieved.

While using the SAVE EXTERNAL DIRECTIVES command make sure that the value of the parameters EXT_DIRECTIVES and IFX_EXTDIRECTIVES are appropriately set as described in 3.7.2, “Parameters for external directives” on page 126. If the client or the server does not set the proper access values, the server will give an error message as shown in Example 3-53.

Example 3-53 Error on SAVE EXTERNAL DIRECTIVES

```
> SAVE EXTERNAL DIRECTIVES {+INDEX(monitor, modelnum) }  
  ACTIVE FOR  
    SELECT {+INDEX(monitor, modelnum) } modelnum  
    FROM   monitor  
    WHERE  modelnum<100;
```

```
9934: Only DBA is authorized to do this operation.  
Error in line 5  
Near character position 20
```

3.7.4 Disabling or deleting an external directive

When an external optimizer directive is no longer required, you can either disable or delete it.

Disabling the external directive

To disable the directive, set the active column of the sysdirectives table to 0. This action is depicted in Example 3-54. You must have the appropriate privileges on the system table to perform this activity.

Example 3-54 Disabling the external directive

```
> SELECT * FROM sysdirectives WHERE id=2;
```

```
id          2  
query       SELECT {+INDEX(monitor, modelnum) } modelnum  
            FROM   monitor  
            WHERE  modelnum<100  
directive   INDEX(monitor, modelnum)  
directivecode <BYTE value>  
active      1  
hashcode    598598048
```

```
1 row(s) retrieved.
```

```
> UPDATE sysdirectives SET active=0 WHERE id=2;
```

1 row(s) updated.

```
> SELECT * FROM sysdirectives WHERE id=2;
```

id	2
query	SELECT {+INDEX(monitor, modelnum) } modelnum FROM monitor WHERE modelnum<100
directive	INDEX(monitor, modelnum)
directivecode	<BYTE value>
active	0
hashcode	598598048

1 row(s) retrieved.

Deleting an external directive

When the directive is no longer needed, the DELETE SQL statement can be used to purge the record from the sysdirectives. This activity is shown in Example 3-55. You must have appropriate privileges on the system table to perform this activity.

Example 3-55 Deleting the external directive

```
> SELECT * FROM sysdirectives WHERE id=2;
```

id	2
query	SELECT {+INDEX(monitor, modelnum) } modelnum FROM monitor WHERE modelnum<100
directive	INDEX(monitor, modelnum)
directivecode	<BYTE value>
active	0
hashcode	598598048

1 row(s) retrieved.

```
> DELETE FROM sysdirectives WHERE id=2;
```

1 row(s) deleted.

```
> SELECT * FROM sysdirectives WHERE id=2;
```

No rows found.

3.8 SQL performance improvements

The robustness of an application is as good as the time taken by the server to massage the query and fetch the data from the database. The more time it takes to optimize the query the less throughput you will have with the application. In this section we discuss some of the key improvements in the IDS V10 SQL that can reduce the over all processing time.

The key enhancements that we discuss in this section are:

- ▶ Configurable memory allocation: For Hash Join, Aggregates' and Group by
- ▶ View folding: For queries with ANSI outer joins
- ▶ ANSI Join optimization: For distributed queries

3.8.1 Configurable memory allocation

IDS V10 has made considerable improvements in the area of configurable memory allocation for queries that incorporate Hash Join, Aggregate, and Group by, directives.

Hash joins

The IDS optimizer chooses to use a hash join strategy to optimize a query in either of the following conditions:

- ▶ The total count of data (rows fetched) for each individual table in the join strategy is very high. Joining such tables can result in an extremely high data count, and thus incur significant overhead.
- ▶ The tables in the join strategy do not have any index on the join column. Consider for example, that there are two tables, table A and table B, that are going to be joined through an SQL query on a common column named x. During the SQL join it is found that column x has not been indexed or has no reference in any of the indices of either table A or table B, then the hash join strategy will be used.

In such a situation, the server builds a hash table on the join columns based on a hash function and then probes this hash table to complete the join.

Example 3-56 shows the output of SQEXPLAIN for a query using the has join strategy.

Example 3-56 HASH JOIN

```
SELECT *  
FROM   customer, supplier  
WHERE  customer.city = supplier.city
```

```
Estimated Cost: 125  
Estimated # of Rows Returned: 510  
1) informix.supplier: SEQUENTIAL SCAN  
2) informix.customer: SEQUENTIAL SCAN
```

DYNAMIC HASH JOIN

Dynamic Hash Filters: informix.supplier.city = informix.customer.city

It is the server that determines the amount memory needed to allocate and build the hash table, so that the hash table can fit in memory. If PDQPRIORITY is set for the query, the Memory Grant Manager (MGM) uses the following formula to determine the memory requirement:

$$\text{memory\_grant\_basis} = (\text{DS\_TOTAL\_MEMORY} / \text{DS\_MAX\_QUERIES}) * (\text{PDQPRIORITY} / 100) * (\text{MAX\_PDQPRIORITY} / 100)$$

In IDS versions prior to IDS V10, when PDQPRIORITY was not set or was set to zero, MGM always allocated 128 KB of memory for each query. This limitation could result in bottlenecks for those queries that had high selectivity, basically because the resultant hash table would not fit in the memory. To overcome this bottleneck IDS would use the temporary dbspaces or operating system files to create the hash table, which could also impact the query performance.

Figure 3-8 illustrates the logic used for determining the memory allocation strategy used for the hash table.

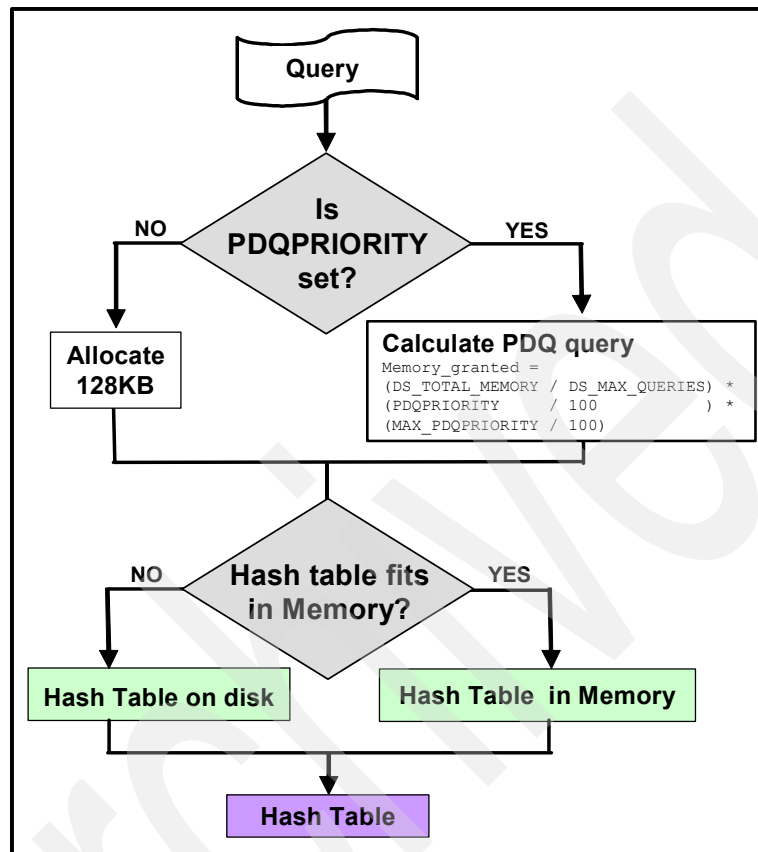


Figure 3-8 Hash table memory allocation strategy

Other memory allocations: ORDER BY and GROUP BY

To evaluate ORDER BY and GROUP BY operations IDS first determines the amount of memory available for each query. As with the hash join, MGM uses the same formula to determine the memory size. If the PDQPRIORITY is not set, the query is allocated the standard 128 KB.

The improvement

Because the server always allocates a maximum of 128 KB of memory per query, irrespective of the size of the hash table to be built, it uses disk to build the hash table. This is also true for those conditions in which queries with large intermediate results need sorting or need to store large temporary results from

the sort. The use of the disk to store the hash table can dramatically impact slow down both hash join and sorting.

To avoid these potential bottlenecks IDS V10 introduced a new configuration variable called `DS_NONPDQ_QUERY_MEM`. This new variable uses a minimum value of 128 KB and maximum of 25% of `DS_TOTAL_MEMORY` as the allocation of memory. Users now have a way of increasing the size of this memory allocation, thus avoiding the use of disk to store hash table results. Thus enabling improved performance.

You can set the `DS_NONPDQ_QUERY_MEM` variable in following ways:

1. A configuration parameter in `ONCONFIG` file. For example:

```
DS_NONPDQ_QUERY_MEM      512    # KB is the unit.
```

2. Using the `onmode` command with the `-wm` or `-wf` options. For example:

- **onmode -wm**: Changes the `DS_NONPDQ_QUERY_MEM` value in the memory. The value set by `-wm` option is lost when the IDS server is shutdown and restarted.

```
onmode -wm DS_NONPDQ_QUERY_MEM=512
```

- **onmode -wf**: Changes the `DS_NONPDQ_QUERY_MEM` value in the memory, along with the value in the `ONCONFIG` file. The value set by the `-wf` option is not lost when the IDS server is shutdown and restarted.

```
onmode -wf DS_NONPDQ_QUERY_MEM=512
```

3. In the **onmonitor** utility: The Non PDQ Query Memory option can be used to set the value for the `DS_NONPDQ_QUERY_MEM` variable. To navigate to this menu, use the **onmonitor** → **Parameters** → **pdQ options**.

When the value of `DS_NONPDQ_QUERY_MEM` is set or changed, you can use the **onstat** utility to verify the amount of memory granted.

Figure 3-57 shows the MGM output displayed by the `onstat` utility.

Example 3-57 MGM output

```
% onstat -g mgm
```

```
IBM Informix Dynamic Server Version 10.00.UC5 -- On-Line -- Up 126 days
00:28:17 -- 1590272 Kbytes
```

Memory Grant Manager (MGM)

```
-----
MAX_PDQPRIORITY:      100
DS_MAX_QUERIES:        20
DS_MAX_SCANS:          1048576
DS_NONPDQ_QUERY_MEM: 16000 KB
```

DS_TOTAL_MEMORY:		100000 KB		
Queries:	Active	Ready	Maximum	
	0	0	20	
Memory:	Total	Free	Quantum	
(KB)	100000	100000	5000	
Scans:	Total	Free	Quantum	
	1048576	1048576	1	
Load Control:	(Memory)	(Scans)	(Priority)	(Max Queries)
(Reinit)				
	Gate 1	Gate 2	Gate 3	Gate 4
Gate 5				
(Queue Length)	0	0	0	0
0				
Active Queries: None				
Ready Queries: None				
Free Resource	Average #		Minimum #	
-----	-----		-----	
Memory	0.0 +- 0.0		12500	
Scans	0.0 +- 0.0		1048576	
Queries	Average #		Maximum #	Total #
-----	-----		-----	-----
Active	0.0 +- 0.0		0	0
Ready	0.0 +- 0.0		0	0
Resource/Lock Cycle Prevention count: 0				

3.8.2 View folding

Normally simple views join one or more tables without using the ANSI join, ORDER BY, DISTINCT, AGGRIGATE and UNION clause. However, complex views joining tables would use at least one of these clauses. If these queries are executed by the IDS server without any optimization processing, they would require higher resource usage and would be less efficient. To avoid this additional overhead, IDS server tries to rewrite these queries during the intermediate rewrite phase without changing the meaning of the query or the result from the query. This technique is known as view folding. For example, by

rewriting the query, the server tries to avoid the creation of temporary tables. This improves the resource usage and improves query performance.

Although the view folding was available in versions prior to IDS V10, it was disabled for the following scenarios:

- ▶ Views that have an aggregate in any of its subqueries.
- ▶ Views that have GROUP BY / ORDER BY / UNION.
- ▶ Views that have DISTINCT in its definition.
- ▶ Views that have Outer Joins (both standard or ANSI).
- ▶ When parent query has UNION or UNION ALL (nested UNIONS).
- ▶ Any distributed query.

In IDS V10, views with UNION ALL in view query and ANSI join in the query using the view can be folded. However, all the other restrictions are still enforced.

Example 3-58 shows the usage of UNION ALL which can use the view folding during query execution.

Example 3-58 UNION ALL view

```
CREATE VIEW topology (vc1, vc2, vc3, vc4) AS
(
    SELECT t1.c1, t1.c2, t2.c1, t2.c2
    FROM   t1, t2
    WHERE  t1.c1 = t2.c1 and ( t1.c2 < 5 )
);
```

Example 3-59 shows the usage of ANSI JOIN query which can also use the view folding during query execution.

Example 3-59 ANSI JOIN Query

```
SELECT    va.vc1, t3.c1
FROM      va
LEFT JOIN t3
ON        va.vc1 = t3.c1;
```

To configure for view folding, set the configuration variable IFX_FOLDVIEW in the ONCONFIG file to enable the view folding. Set to value 1 to enable folding and to value 0 to disable folding.

```
IFX_VIEWFOLD 1
```

Case studies

In this section we discuss the concept of the view folding in several case study scenarios.

Case 1: Multiple table join with ANSI joins case

This case is a comparison between the old and the new query plan when multiple tables are joined with ANSI joins for the following schema and query:

Schema:

```
CREATE VIEW topology (vc1, vc2, vc3, vc4) AS
(
    SELECT t1.c1, t1.c2, t2.c1, t2.c2
    FROM   t1, t2
    WHERE  t1.c1 = t2.c1 and ( t1.c2 < 5 )
);
```

Query:

```
SELECT    va.vc1, t3.c1
FROM      va
LEFT JOIN t3
ON        va.vc1 = t3.c1;
```

- **Old query plan:** A temp table is created for the view with multiple table joins and an ANSI left join in the parent query.

1) (Temp Table For View): SEQUENTIAL SCAN

2) usr1.t3: AUTOINDEX PATH

(1) Index Keys: c1 (Key-Only)

Lower Index Filter: (Temp Table For View).vc1 = usr1.t3.c1

ON-Filters: (Temp Table For View).vc1 = usr1.t3.c1

NESTED LOOP JOIN (LEFT OUTER JOIN)

- **New query plan:** This new plan for multiple table join with ANSI joins shows that the view references are replaced by base tables in the parent query.

1) usr1.t2: SEQUENTIAL SCAN

2) usr1.t1: AUTOINDEX PATH

Filters:

Table Scan Filters: usr1.t1.c2 < 5

(1) Index Keys: c1

Lower Index Filter: usr1.t1.c1 = usr1.t2.c1

ON-Filters:(usr1.t1.c1 = usr1.t2.c1 AND usr1.t1.c2 < 5)
NESTED LOOP JOIN

3) usr1.t3: AUTOINDEX PATH

(1) Index Keys: c1 (Key-Only)
Lower Index Filter: usr1.t1.c1 = usr1.t3.c1

ON-Filters:usr1.t1.c1 = usr1.t3.c1
NESTED LOOP JOIN(LEFT OUTER JOIN)

Here are few more query examples where IDS folds the view query into the parent query:

1. SELECT va.vc1, t3.c1

FROM va
RIGHT JOIN t3 ON va.vc1 = t3.c1;

2. SELECT va.vc1, t3.c1

FROM va
FULL JOIN t3 ON va.vc1 = t3.c1;

3. SELECT t1.c2, va.vc4, t3.c2

FROM t1
RIGHT JOIN (va LEFT JOIN t3 ON va.vc3 = t3.c1)
ON (t1.c1 = va.vc3 AND va.vc3 < 2)
WHERE t1.c1 = 1;

4. SELECT t3.c1

FROM (t3 RIGHT OUTER JOIN (t2 RIGHT OUTER JOIN va ON t2.c1=vc1)
ON t3.c1=t2.c1);

5. SELECT t3.c1

FROM (t3 LEFT OUTER JOIN (t2 left outer join va ON t2.c1=vc1)
ON t3.c1=t2.c1);

6. SELECT vc1, t1.c2

FROM t1
LEFT OUTER JOIN va ON t1.c1=va.vc3

```

WHERE t1.c2 < 3
UNION
  SELECT vc1, t3.c2
  FROM t3
  LEFT OUTER JOIN va
  ON t3.c1=va.vc3
  WHERE t3.c2 < 2;

```

Case 2: Query rewrite for UNION ALL view with ANSI joins

IDS V10 will fold views with UNION ALL into the parent query. As part of this view folding, it will rewrite the parent query into an equivalent UNION ALL query. This new query will incorporate the view with its query and predicates.

Consider the view definition with the union all in Example 3-60.

Example 3-60 View with union all

```

View V1: <Left_query > UNION ALL <Right_query>
Parent query: SELECT * FROM V1 LEFT JOIN tab1 <ON_CLAUSE>
IDS will rewrite this parent query as UNION ALL during execution:
SELECT * FROM <Left_query> LEFT JOIN tab1 <ON_CLAUSE for Left_query>
UNION ALL
SELECT * FROM <Right_query> LEFT JOIN tab1 <ON_CLAUSE for Right_query>

```

IDS rewrites a UNION ALL view query when the parent query includes the following:

- ▶ Regular join
- ▶ Informix join
- ▶ ANSI join
- ▶ ORDER BY

IDS will also create a temp table to get an intermediate result set for the view under the following conditions with UNION ALL view.

- ▶ View has aggregate, GROUP BY, ORDER BY, UNION, DISTINCT, outer joins (ANSI or Informix)
- ▶ Parent query has UNION, UNION ALL
- ▶ Remote query with UNION ALL view references
- ▶ View has UNION clause but not UNION ALL. A UNION ALL result set can have duplicate rows, but UNION requires removal of duplicates.

Example 3-61 shows a UNION ALL case where the view is folded into the parent query.

Example 3-61 UNION ALL with folded view

```
CREATE VIEW v0 ( vc1, vc2, vc3, vc4)
AS
(
SELECT c1, c2, c3, c3-1 FROM t1 WHERE c1 < 5
UNION ALL
SELECT c1, c2, c3, c3-1 FROM t2 WHERE c1 > 5
);
```

Consider a parent query where a view reference (v0) is on the dominant (outer table) side. This is depicted in Example 3-62. Here, the view reference in the parent query is replaced by the left and right parts of the union node. Using the view folding technique, the parent query is executed as a UNION ALL query with all available table filters from view definition.

Example 3-62 View reference on the dominant side

Left join case

```
SELECT v0.vc1, t3.c1 FROM v0 LEFT JOIN t3 ON v0.vc1 = t3.c1
```

1) usr1.t1: INDEX PATH

(1) Index Keys: c1 (Key-Only) (Serial, fragments: ALL)
Upper Index Filter: usr1.t1.c1 < 5

2) usr1.t3: AUTOINDEX PATH

(1) Index Keys: c1 (Key-Only)
Lower Index Filter: usr1.t1.c1 = usr1.t3.c1

ON-Filters:usr1.t1.c1 = usr1.t3.c1
NESTED LOOP JOIN(LEFT OUTER JOIN)

Union Query:

1) usr1.t2: SEQUENTIAL SCAN

Filters: usr1.t2.c1 > 5

2) usr1.t3: AUTOINDEX PATH

(1) Index Keys: c1 (Key-Only)
Lower Index Filter: usr1.t2.c1 = usr1.t3.c1

ON-Filters:usr1.t2.c1 = usr1.t3.c1
NESTED LOOP JOIN(LEFT OUTER JOIN)

Right join case

SELECT t3.c1, v0.vc1 FROM t3 RIGHT JOIN v0 ON v0.vc1 = t3.c1

1) usr1.t1: INDEX PATH

(1) Index Keys: c1 (Key-Only) (Serial, fragments: ALL)
Upper Index Filter: usr1.t1.c1 < 5

2) usr1.t3: AUTOINDEX PATH

(1) Index Keys: c1 (Key-Only)
Lower Index Filter: usr1.t1.c1 = usr1.t3.c1

ON-Filters:usr1.t1.c1 = usr1.t3.c1
NESTED LOOP JOIN(LEFT OUTER JOIN)

Union Query:

1) usr1.t2: SEQUENTIAL SCAN

Filters: usr1.t2.c1 > 5

2) usr1.t3: AUTOINDEX PATH

(1) Index Keys: c1 (Key-Only)
Lower Index Filter: usr1.t2.c1 = usr1.t3.c1

ON-Filters:usr1.t2.c1 = usr1.t3.c1
NESTED LOOP JOIN(LEFT OUTER JOIN)

In Example 3-63 the view on dominant side (v1) is folded where as the view on the subservient side (v2) is materialized. IDS will convert the parent query into a UNION ALL query for v1 view reference.

Example 3-63 Multiple UNION ALL views

CREATE TABLE t5 (c1 INT, c2 INT , c3 INT);


```
CREATE VIEW v1 (vc1) as
SELECT t1.c2 FROM t1 WHERE t1.c2 > 3
UNION ALL
SELECT t2.c2 FROM t2 WHERE t2.c2 < 3 ;
```

```
CREATE VIEW v2 (vc2) as
SELECT t3.c2 FROM t3 WHERE t3.c2 > 3
UNION ALL
SELECT t4.c2 FROM t4 WHERE t4.c2 < 3 ;
```

```
SELECT v2.vc2, v1.vc1
FROM (v1 LEFT JOIN t5 ON v1.vc1 = t5.c1)
     LEFT JOIN v2 ON v1.vc1 = v2.vc2;
```

This SELECT will give the following query plan
LEFT JOIN v2 ON v1.vc1 = v2.vc2;

1) usr1.t1: SEQUENTIAL SCAN
Filters: usr1.t1.c2 > 3

2) (Temp Table For View): AUTOINDEX PATH
(1) Index Keys: c2
Lower Index Filter: usr1.t1.c2 = (Temp Table For View).vc2

ON-Filters:usr1.t1.c2 = (Temp Table For View).vc2
NESTED LOOP JOIN(LEFT OUTER JOIN

3) usr1.t5: AUTOINDEX PATH
(1) Index Keys: c1 (Key-Only)
Lower Index Filter: usr1.t1.c2 = usr1.t5.c1

Union Query:

1) usr1.t2: SEQUENTIAL SCAN
Filters: usr1.t2.c2 < 3

2) (Temp Table For View): AUTOINDEX PATH
(1) Index Keys: c2
Lower Index Filter: usr1.t2.c2 = (Temp Table For View).vc2

ON-Filters:usr1.t2.c2 = (Temp Table For View).vc2
NESTED LOOP JOIN(LEFT OUTER JOIN)

3) usr1.t5: AUTOINDEX PATH

(1) Index Keys: c1 (Key-Only)
Lower Index Filter: usr1.t2.c2 = usr1.t5.c1

ON-Filters:usr1.t2.c2 = usr1.t5.c1
NESTED LOOP JOIN(LEFT OUTER JOIN)

Table 3-7 represents cases where simple and union all views are folded into the parent query. IDS folds union all views when they are referenced as the dominant side. When the same view is referenced on both the dominant and subservient side in the parent query, IDS will fold the view for dominant side and create a temp table for the subservient side. A view reference in a Full Outer Join query will not be folded when the view has union all.

All ANSI join cases with simple views will be folded into parent query.

Table 3-7 Folding Criteria for Simple and UNION ALL Query

Query Type	Main Query LOJ		Main Query ROJ		Main Query FOJ	
View Type	D	S	D	S	D	S
Simple (No UNION ALL)	Yes	Yes	Yes	Yes	Yes	Yes
UNION ALL	Yes	No	Yes	No	No	No

D: View has dominant (outer table) role in main query, for example V1 left join T1
S: View has subservient (inner table) role in main query, for example T1 left join V1
Main query LOJ: main query has Left Outer Join
Main query ROJ: main query has Right Outer Join
Main query FOJ: main query has Full Outer Join

3.8.3 ANSI JOIN optimization for distributed queries

In this section we demonstrate examples of join optimization for distributed queries. Consider the application shown in Figure 3-9. An application connecting to the orderdb@order_server, cannot only query tables in orderdb, but also inventorydb in the same server, and to the payrolldb and partnerdb in the payment_server, and the shippingdb on the shipping_server. In this situation, the order_server is the coordinating IDS server, and payment_server and shipping_server are subordinate IDS servers.

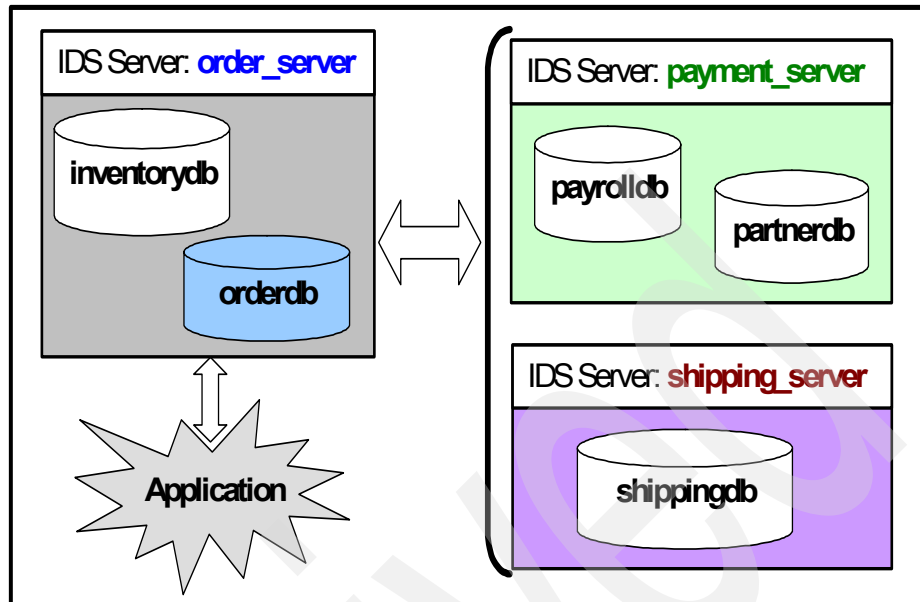


Figure 3-9 A sample application

Querying across databases in the local server has no impact on the optimization of queries. The cost of a table scan, index scan, and page fetch are the same on all objects in the same server. However, all this changes when querying tables in remote servers. The optimizer at the coordinator generates query fragments for each participating server to execute, decides on which predicates to push, and creates the query plan at the coordinator server. The optimizer at respective subordinator server will generate the plans for the query fragments they execute. The coordinating server optimizer has to determine the most cost effective plan to minimize data movement and push the right predicates to enable the optimal plan to be generated at each server.

Now consider the following scenarios:

- ▶ Connect to the server **order_server** and database **orderdb**
`CONNECT TO orderdb@order_server;`
- ▶ Query from **inventorydb** in the same server
`SELECT *`
`FROM inventorydb@order_server:customertab;`

- Query from payrolldb on a remote server: payment_server

```
SELECT *
FROM payrolldb@payment_server:emptab
WHERE empid = 12345;
```

Because all objects referenced in the query comes from one remote server, the complete query will be shipped to remote server. Now consider a distributed query:

- Distributed query for cross server scenario.

The application is connected to orderdb. The query is from the table in inventorydb on the local server and payrolldb on the payment server.

```
SELECT *
FROM inventorydb:customertab a,
      payrolldb@payment_server:emptab b
WHERE a.customer = b.paidcustomer;
```

Example query plan:

```
CONNECT TO orderdb@order_server;
SELECT l.customer_num, l.lname, l.company,
       l.phone, r.call_dtime, r.call_descr
FROM customer l, partnerdb@payment_server:svrc_calls r
WHERE l.customer_num = r.customer_num
      AND r.customer_calldate = '12/25/2005';
```

Estimated Cost: 9
Estimated # of Rows Returned: 7

1) informix.r: REMOTE PATH

Remote SQL Request:

```
select x0.call_dtime,x0.call_descr,x0. customer_num from
partnerdb:"virginia".svrc_calls x0
where x0.customer_calldate = '12/25/2005'
```

2) informix.l: INDEX PATH

(1) Index Keys: customer_num (Serial, fragments: ALL)

Lower Index Filter: informix.l.customer_num =
informix.r.customer_num

NESTED LOOP JOIN

So far so good. IDS optimizes the query to minimize the movement of data and executes the query with lowest possible cost.

IDS versions prior to V10 have had limitations when ANSI joins are introduced. Instead of grouping the tables, predicates and joins into appropriate servers, it retrieves each table separately and joins locally. Obviously, this is not too efficient.

Example:

```
SELECT a1.c1 , a2.c1
FROM payrolldb@payment_server:partstab a1 LEFT JOIN
      partnerdb@payment_server:p_ordertab a2
      ON a1.c1 = a2.c1
```

Estimated Cost: 5
Estimated # of Rows Returned: 1

1) usr1.a1: REMOTE PATH

Remote SQL Request:
select x0.c1 from payrolldb:"usr1".t1 x0

2) usr1.a2: REMOTE PATH

Remote SQL Request:
select x1.c1 from partnerdb:"usr1".t2 x1

ON-Filters:usr1.a1.c1 = usr1.a2.c1
NESTED LOOP JOIN(LEFT OUTER JOIN)

In this plan, IDS chooses to fetch the two tables partstab and p_ordertab from payment_server, and join locally. In this simple query, both tables reside on the same server. However, this plan can be a problem.

The IDS V10 optimizer, recognizes these situations and groups the tables and queries together.

Case 1: LEFT JOIN of two tables from the same remote database

```
SELECT a1.*, a2.*
FROM rdb2@rem1:t1 a1
LEFT JOIN rdb2@rem1:t2 a2
ON a1.t1c1 = a2.t2c1
```

```
SELECT a1.c1 , a2.c1
FROM payrolldb@payment_server:partstab a1 LEFT JOIN
      partnerdb@payment_server:p_ordertab a2
      ON a1.c1 = a2.c1
```

Estimated Cost: 3
Estimated # of Rows Returned: 4

1) usr1.a1, usr1.a2: REMOTE PATH

Remote SQL Request:
SELECT x3.t1c1 ,x3.t1c2 ,x2.t2c1 ,x2.t2c2
FROM (payrolldb:"usr1".t1 x3
LEFT JOIN payrolldb:"usr2".t2 x2
ON (x3.t1c1 = x2.t2c1))

Case 2: RIGHT JOIN of two tables from a remote database

```
SELECT a1.*, a2.*  
FROM rdb2@rem1:t1 a1  
RIGHT JOIN rdb2@rem1:t2 a2  
ON a1.t1c1 = a2.t2c1
```

Estimated Cost: 3
Estimated # of Rows Returned: 3

1) usr1.a2, usr1.a1: REMOTE PATH

Remote SQL Request:
SELECT x1.t1c1 ,x1.t1c2 ,x0.t2c1 ,x0.t2c2
FROM (rdb2:"usr1".t2 x0
LEFT JOIN rdb2:"usr1".t1 x1
ON (x1.t1c1 = x0.t2c1))

Case 3: Views and procedures

```
CREATE VIEW vc1(c1, c2, c3, c4) AS  
SELECT a1.*, a2.* FROM rdb2@rem1:t1 a1 CROSS JOIN rdb2@rem1:t2 a2;
```

QUERY:

SELECT * FROM vc1

Estimated Cost: 5
Estimated # of Rows Returned: 12

1) rdb2@rem1:usr1.t2, rdb2@rem1:usr1.t1: REMOTE PATH

Remote SQL Request:
SELECT x1.t1c1 ,x1.t1c2 ,x0.t2c1 ,x0.t2c2
FROM rdb2:"usr1".t2 x0, rdb2:"usr1".t1 x1

Procedure: usr1.pr2

```
SELECT count(*)
FROM rdb3@rem1:"usr1".r2 x0 ,rdb3@rem1:"usr1".r1 x1
WHERE ((x1.r1c1 = x0.r2c1 ) AND (x1.r1c1 <= 5 ) )
```

QUERY:

Estimated Cost: 2

Estimated # of Rows Returned: 1

1) usr1.a2, usr1.a1: REMOTE PATH

Remote SQL Request:

```
SELECT COUNT(*)
FROM rdb3:"usr1".r2 x2 ,rdb3:"usr1".r1 x3
WHERE (x2.r2c1 <= 5 ) AND
      ((x3.r1c1 = x2.r2c1 ) AND
       (x3.r1c1 <= 5 ))
```

Case 4: Combination of LEFT join and RIGHT joins and multiple IDS servers

```
SELECT a1.t1c2, a2.t2c2, a3.r3c2
FROM (rdb2@rsys2:t1 a1 LEFT JOIN rdb2@rsys2:t2 a2
      ON a1.t1c1 = a2.t2c1) RIGHT JOIN rdb3@rsys3:r3 a3
      ON (a1.t1c1 = a3.r3c1 AND a3.r3c1 < 3) WHERE a1.t1c1 = 1
```

Estimated Cost: 2

Estimated # of Rows Returned: 1

1) usr1.a3: REMOTE PATH

Remote SQL Request:

```
SELECT x0.r3c1 ,x0.r3c2
FROM rdb3:"usr1".r3 x0
```

2) usr1.a1, usr1.a2: REMOTE PATH

Remote SQL Request:

```
SELECT x1.t1c1 ,x1.t1c2 ,x2.t2c1 ,x2.t2c2
FROM (
      rdb2:"usr1".t1 x1
      LEFT JOIN rdb2:"usr1".t2 x2
```

```
ON (x1.t1c1 = x2.t2c1 )  
)  
WHERE ( x1.t1c1 = 1 )
```

```
ON-Filters:(usr1.a1.t1c1 = usr1.a3.r3c1 AND usr1.a3.r3c1 < 3 )
```

```
NESTED LOOP JOIN
```

The examples demonstrated in this section have provided some good information and understanding of the power of IDS V10 as it supports distributed queries.

Extending IDS for business advantages

In the computing world today, it seems that most software products have extensibility features. You can look at operating systems with their device drivers, Web servers and the Common Gateway Interface (CGI) and application programming interfaces (APIs), Microsoft Visual Studio and the Eclipse platforms with their plug-ins, and application servers. All these products allow significant levels of customization. This customization enables products to better fit their environment, providing greater value to their users.

Data Servers are no exception. Stored procedures and triggers can be considered an initial step toward customization. IDS was the first data server to include a comprehensive set of extensibility features above and beyond that initial step. Over the last 10 years, these features have improved and matured to provide a robust and flexible environment for extensibility.

You can take advantage of IDS extensibility by using existing DataBlades, by tailoring example code to fit your needs, or by writing your own. DataBlades are pre-packaged sets of extensibility functions that IBM or third-party vendors offer. We described them briefly in 4.3.1, “DataBlades” on page 162 and discuss them in more detail in Chapter 5, “Functional extensions to IDS” on page 175.

4.1 Why extensibility

The bottom line is that extensibility allows you to adapt the data server to your business environment instead of compromising your design to fit the data server. The examples in the sections that follow illustrate this point. The result is less complexity and better performance. The benefits include, among other things, lower cost due to higher performance, faster time-to-market, and easier maintenance due to less complexity.

What do we mean by reducing complexity? This is easier answered through examples. In the following sections are three examples that describe different areas of extensibility. You can find more examples in the articles that we list in “Related publications” on page 387.

4.1.1 Date manipulation example

Let us start with a simple example, date manipulation. Many companies analyze data based on quarterly activities. Some also have a need to look at their data depending on the week of the year. In addition, these date manipulations might be dependant on an arbitrary year that starts on an arbitrary date. And there are some who have a need to manipulate their date based on multiple different definitions of the year.

You can write a user-defined routine (UDR), also called a user-defined function (UDF), that takes a date as input and returns a week of the year value. This value could include the year and the week and could be implemented as an integer, such as 200601, or as a character string, such as 2006W01. These are implementation choices. It could even be an integer that only represents the week of the year, without consideration for the year. An implementation using this value would, as an example, give IDS the ability to select, group, and order rows based on the week of the year in the date column. The set processing is provided by the IDS data server. The routine only knows how to take a date and return a week of the year. With the new `weekOfYear()` routine, you can answer questions such as:

- ▶ What is the revenue for week 27 in 2005 and 2006?
- ▶ What is the percentage increase or decrease of revenue from 2005 to 2006 for week 32?

Example 4-1 shows an example SQL statement that answers the question in the second bullet item.

Example 4-1 Week of the year

```
SELECT 100 * (SUM(n2.total_price) -  
SUM(p2.total_price)) / SUM(p2.total_price)  
FROM orders p1, orders n1, items p2, items n2  
WHERE p1.order_num = p2.order_num  
AND n1.order_num = n2.order_num  
AND YEAR(p1.order_date) = 2005 - year to use  
AND weekOfYear(p1.order_date) = 32 - week of the year  
AND YEAR(p1.order_date) = (YEAR(n1.order_date) - 1)  
AND weekOfYear(p1.order_date) = weekOfYear(n1.order_date);
```

How could the percentage increase or decrease in revenue be calculated without the `weekOfYear()` routine? It is likely that it would result in two different SQL statements to get the revenue for week 32 of 2005 and 2006 and then use of an application or a stored procedure to calculate the percentage increase or decrease. However, a stored procedure can only use static SQL, which means that it becomes specific to a particular problem.

With the `weekOfYear()` routine, it is all done in one SQL statement, and there is no need for a custom application to get to the final result.

To improve performance, it is also possible to create an index on the result of the routine so the previous SQL statement could use indexes. The index creation code could look like the following:

```
CREATE INDEX orders_week_idx  
ON orders( weekOfYear(order_date) );
```

The result is better performance because of the use of one SQL statement instead of two, and by the fact that the index is very likely to be used to solve the query.

4.1.2 Fabric classification example

Now let us look at an example from the Textile industry, specific to window coverings. When classifying the fabric color of the window covering, it must be precisely identified. The industry uses a standard, called *CILAB*, that uses three numbers to identify a specific color. If there is a need to do a search to find all the fabrics that are in shades within a desired range of a specific color, the search involves varying three separate values. A query would have the form shown in Example 4-2.

Example 4-2 Fabric color example

```
SELECT * FROM fabric
WHERE colorV1 between ? AND ?
AND colorV2 between ? AND ?
AND colorV3 between ? AND ?;
```

This query is likely to always use a table scan because any B-tree index defined on the table would not be able to eliminate enough values to be selected by the optimizer. The general rule-of-thumb is that if an index returns more than 20% of the values, a table scan is more efficient. Because each value is independent from the other, it is unlikely that an index would be able to eliminate enough values.

IDS allows the database developer to define new data types and also to use a new type of index called the R-tree index. The R-tree index is a multidimensional index. It can index multiple values, providing the type of search needed to solve the problem. The capability is easier to explain through an example that uses two dimensions instead of three. Looking at Figure 4-1, you can see that the desired values were identified by placing a circle around a particular two-dimensional value.

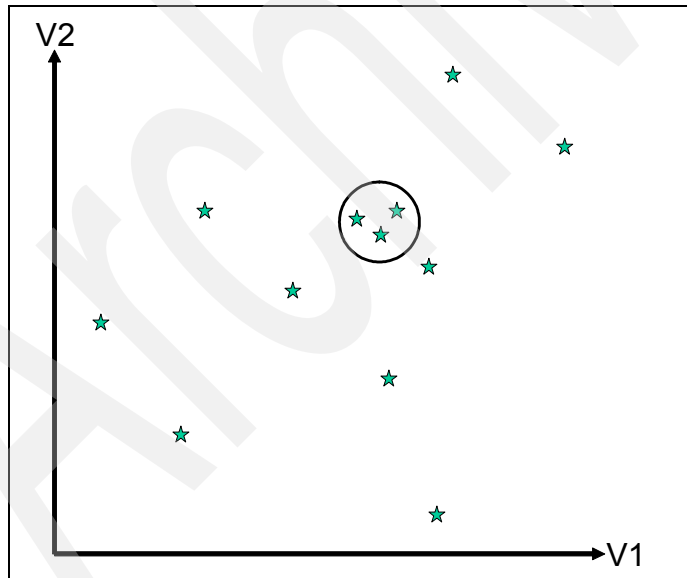


Figure 4-1 Multidimensional index

For a three dimensional situation, as in the problem discussed previously, you can define a sphere around a specific point and locate the colors desired.

Because the index is able to accommodate this type of search, it would eliminate enough values to be selected as the processing path by the optimizer. The general form of such an SQL query would be:

```
SELECT * FROM fabric  
WHERE CONTAINS(color, CIRCLE(?,?));
```

In this case, the arguments of the circle function would be a specific color and the variation on each dimension.

There are many problems that can take advantage of this type of indexing. The most obvious one relates to spatial problems. We describe the spatial and geodetic DataBlades in Chapter 5, “Functional extensions to IDS” on page 175.

4.1.3 Risk calculation example

Imagine an executive that is operating a bank with multiple branches in a number of regions. This bank provides loans to companies in different industries. One important task for this executive is to monitor the amount of risk taken by each branch. If a branch is taking more risk than is acceptable, the executive must do further analysis on that branch to make sure there are acceptable business reasons for this extra level of risk.

The standard way to do this would be to retrieve each loan from a branch, calculate the risk of each loan and calculate an average in the application. For those people using an object oriented approach, it might mean that an object will be instantiated for the corporation, multiple objects for the regions and branches, and one object per loan. Then, keeping with the encapsulation concept, the corporate object must ask for the average balance of each branch to each region. In turn, each region asks each branch and each branch will request information from each loan. Figure 4-2 illustrates this scenario.

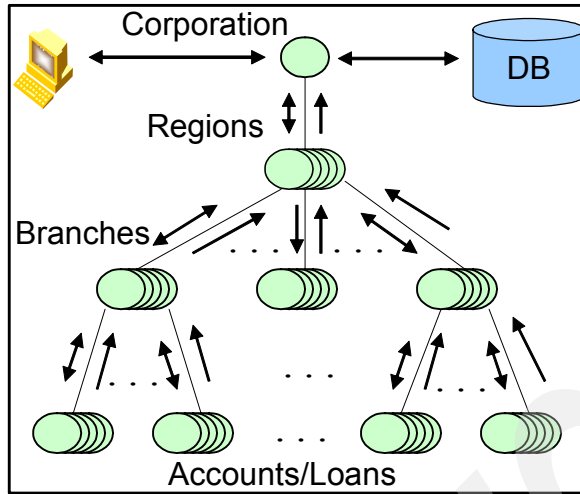


Figure 4-2 Loan example

This approach could easily overwhelm a computer system, possibly forcing an upgrade to a larger machine or requiring the use multiple machines to do the processing. When the implementation must be moved to multiple machines, the level of complexity explodes.

With IDS you can create a user-defined aggregate that can use the algorithm developed for the application to calculate the average risk. A solution could also include the use of a table hierarchy to differentiate between the types of loans. Example 4-3 shows a SQL statement that solves the problem.

Example 4-3 Risk example code

```
SELECT branch_id, AVGRISK(loans)
FROM loans
GROUP BY 1
HAVING AVGRISK(loans) > 1
ORDER BY 2 DESC;
```

Once again, the set processing is done in the data server which simplifies the code needed to solve the problem. The result is a simpler solution that performs much better due in large part to the fact that we limit the data movement.

4.2 IDS extensibility features

You have seen in the extensibility examples some of the IDS extensibility features. There are brief descriptions of those features in the following sections.

4.2.1 Data types

IDS support four types of data types for extensibility:

- ▶ Distinct

A *distinct* type is based on an already existing type, including any types that are available in IDS. A distinct type allows you to implement strong typing in the database. An example of use is the definition of a *U.S. dollar* type and a *euro* type instead of simply use the standard *Money* type. Simple SQL statement errors, such as adding dollars to euros, could be detected and avoided.

- ▶ Opaque

An *opaque* type is a type that is newly defined. The implementation must also include a set of support functions that define how to manipulate this type. The code can also decide at storage time if the type should be stored in-row or in a smart-blob space. This is called multi-representation.

- ▶ Row

A *row* type is similar to a table definition. IDS supports two types of rows, *named* and *unnamed*. A row type can be used as a column type or as a table type. Row types support single inheritance so you can define a row type starting from an existing one. It is also used to support table hierarchies.

- ▶ Collections

A *collection* type can be either a set, multiset, or list. A set is a collection of unordered unique values. A multiset is a collection of unordered collections of non-unique value. A List is an ordered collection of unique values.

4.2.2 Routines

IDS supports user-defined routines (UDRs) and user-defined aggregates (UDAs). A routine can either return a single value or a set of values. In the latter case, it is called an iterator routine. It is similar to stored procedure routines that executes a return *with resume*.

The creation of a UDR is similar to the creation of a stored procedure. Example 4-4 shows an example of the simplified syntax.

Example 4-4 UDR syntax

```
CREATE [DBA] FUNCTION function_name( [parameter_list] )  
RETURNING SQL_type  
  [WITH( [ { HANDLESNULLS | [NOT] VARIANT | ITERATOR | PARALLELIZABLE} ]  
  ]  
EXTERNAL NAME 'path_name'  
[LANGUAGE { C | Java } ]  
END FUNCTION  
;  
  
parameter_list: [{IN|INOUT}] parameter [[, [{OUT|INOUT}] parameter]. .  
.]  
  
parameter: [parameter_name] SQL_type  
[DEFAULT default_value]
```

This syntax is easier to understand with another example. Let us look at the creation of a function named *Quarter* that is implemented in C as shown in Example 4-5.

Example 4-5 Quarter function

```
CREATE FUNCTION quarter(DATE)  
RETURNING VARCHAR(10)  
WITH (NOT VARIANT, PARALLELIZABLE)  
EXTERNAL NAME  
"$INFORMIXDIR/extend/datesfn/datesfn.bld(quarter)"  
LANGUAGE C  
END FUNCTION;
```

This statement creates a function named *Quarter* that takes a date as an argument and it returns a VARCHAR(10). The function includes two modifiers (WITH statement). It always returns the same value for a given argument and does not modify the database (NOT VARIANT). This also means that the result could be indexed through a functional index, as explained in 4.2.3, “Indexing” on page 160.

The modifier PARALLELIZABLE indicates that this C function can run in parallel on the server according to the parallelization rules of IDS.

The statement then provides the location of the shared library that contains the compiled implementation and the name used in the library for the

implementation. This is explained in “Extending IDS with C” on page 169. The CREATE statement is completed by indicating that the language is “C”, and terminate the statement with the END FUNCTION.

A user-defined aggregate (UDA) is similar in concept to any other aggregate function available on the server, such as AVG and SUM. Example 4-6 shows the UDA syntax.

Example 4-6 UDA

```
CREATE AGGREGATE aggregate_name WITH ( modifier_list) ;

modifier_list: modifier [ [, modifier] ...]

modifier: { [INIT=function_name] | ITER=function_name |
COMBINE=function_name | [FINAL=function_name] | HANDLESNULLS }
```

The idea behind the UDA implementation is that it has to fit within the IDS multi-threaded architecture. This means having the capability to merge partial results into a final result.

A UDA consists of up to four functions. An optional function to set some initial values, a mandatory iterator function that is used to calculate a partial result, a mandatory combine function that merges together two partial results, and an optional finalization function that converts the internal final result into the SQL data type chosen for the implementation.

The HANDLESNULLS argument in the modifier definition is used to indicate that the UDA can receive and process a NULL value. If this modifier is not used, NULL values are ignored and do not contribute to the result.

IDS also includes to capability to overload comparison functions and operator. This means that if you define a new data type, you can create the functions that will provide the implementation for such operator as equal (=) and greater than (>). This is done by providing functions with special names. These special names include:

- ▶ concat
- ▶ divide
- ▶ equal
- ▶ greaterthan
- ▶ greaterthanorequal
- ▶ lessthan
- ▶ lessthanorequal
- ▶ minus
- ▶ negative

- ▶ notequal
- ▶ plus
- ▶ positive
- ▶ times

These routines (UDRs, UDAs, and operators) give you the flexibility that you need to implement business rules and business processing in the database when that is part of the design.

4.2.3 Indexing

IDS extensibility features include a new type of indexing, the R-tree index. This is a multidimensional index that can be used in many situations (for example, spatial-type queries). IDS also supports B-tree indexing of any type, as long as you can define a sorting order.

In addition, IDS can index the result of a user-defined routine. This type of index is called a *functional index*. It is important to note that you cannot create a functional index on the result of a built-in function. For example, you cannot create a functional index on the result of the UPPER function. You can work around this limitation easily by creating an SPL routine that serves as a wrapper. The definition of this function would be:

```
CREATE FUNCTION myUpper(in VARCHAR(300))
WITH (NOT VARIANT)
RETURN(UPPER(in))
END FUNCTION;
```

The creation of an index on the result of the myUpper function could speed up the processing of SQL statements such as the following:

```
SELECT * FROM customer
WHERE MyUpper(lastname) = "ROY";
```

4.2.4 Other capabilities

IDS includes more extensibility features. Here are short descriptions of the more major features:

- ▶ Table/Row Type hierarchy: IDS allows you to define single inheritance hierarchies and tables.
- ▶ Trigger introspection: You can retrieve the context of a trigger from within a UDR written in C. See the article *Event-driven fined-grained auditing with Informix Dynamic Server* that we list in “Related publications” on page 387.

- ▶ **Events:** the DataBlade API allows you to register functions that will be called when events such as rollback or commits occur.
- ▶ **Virtual Table Interface:** This is an advance API that allows you to make any data look like a table. An example of that is provided in the article *Informix Flat-File Access* that we list in “Related publications” on page 387.
- ▶ **Virtual Index Interface:** Another advanced interface that allows you to implement new indexing methods.

4.3 Extending IDS

Before looking at how you can use IDS extensibility, we must first define the required task. Here are some quick guidelines that can help you decide what can be done in the data server as opposed to an application:

- ▶ **Eliminate large data transfer:**

Avoiding data transfer can translate into a noticeable performance improvement due to the difference in speed of CPU/memory compared to network speed. This can be important even when using shared memory connections.

- ▶ **Provide new grouping or sorting order:**

The date processing example for weekOfYear in Example 4-1 on page 153 illustrates the grouping benefit.

- ▶ **Implement consistent business rules:**

This could be as simple as implementing a routine that takes a date and returns the business quarter so people do not make mistakes in their queries and miss a day at the end of the quarter. It could go as far as implementing complex risk analysis functions that can be used by multiple applications.

- ▶ **Define new types of relationships:**

An example of this is hierarchical relationships, which can be addressed with a new data type. The *Node* type, available on the IBM developerWorks Web site (<http://www-128.ibm.com/developerworks/>) and described in detail in the book *Open-Source Component for IDS 9.x* (which we list in “Related publications” on page 387), can greatly improve the performance of queries handling hierarchies.

These are high-level suggestions. You must make the final decision based on your particular requirements, and on what makes sense to add to the data server in your environment.

Another suggestion is that when you decide to implement new data types and functions, start small. Determine what is the small functionality that could be implemented that could give you the largest benefit. Look at IDS as a relational framework. In this case, your functions should augment the RDBMS capabilities. That is, your function should augment the IDS capability to sort, group, and perform set processing.

Before getting into any extensibility development, look at what is already available. You have the choice to use DataBlades that come with IDS or DataBlades that are available at a cost, or to use Bladelets or example code.

4.3.1 DataBlades

When it comes to taking advantage of IDS extensibility, the first step should be to look at the available DataBlade modules that come with IDS or are available for a charge.

A DataBlade is a packaging of functionality that solves a specific domain problem. It can include user-defined types (UDTs), user-defined routines (UDRs), user-defined aggregates (UDAs), tables, views, and even a client interface. They are packaged in a way that makes it very easy to register their functionality into a database.

For more information about DataBlades, refer to Chapter 5, “Functional extensions to IDS” on page 175.

Bladelets and example code

The next best thing to using DataBlades is to take advantage of extensions already written. They are known as either *Bladelets* (small DataBlades) or example code. These extensions are available in a similar fashion to open-source code. That is, they come with source code and they are not supported.

With access to the source code, you can study the implementation, and then add to it to better fit it into your business requirements and environment. These extensions can provide some key functionality that could save time, cost, resources, and effort. Here are some examples of these extensions:

- **Node:** The Node type allows you to manipulate hierarchies in a more efficient manner. It includes a set of functions such as `isAncestor()`, `isDescendant()`, `isParent()`, and so on that provides information about the hierarchical relationship of two node types. This can be useful in applications such 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.

- ▶ Genxml: The genxml set of functions give you the capability of generating XML directly from IDS.
- ▶ fvti: This extension uses the virtual table interface to make files outside the IDS engine appear as tables. This can be useful when there is a need to join the content of a file with information already in the database server. This solution then uses the power of the IDS engine to do the join and the conditions attached to it, and simply to get the result you are looking for.
- ▶ idn_mrLvarchar and regexp: These two extensions work together to provide better searching capabilities. The idn_mrLvarchar is a character string type that is stored in-row when it is shorter than 1024 bytes and stored in a character large object (CLOB) when it exceeds that length. This provides a more efficient storage mechanism for character fields such as remarks and descriptions. The regexp set of functions give you expression search capabilities on character fields stored in an idn_mrLvarchar data type.

You can obtain these extensions and many more from multiple sources. Refer to the listings in “Related publications” on page 387 for more information about these resources.

Building your own

SPL, Java, C, or a mix of the three can be used to extend IDS. The choice depends on the particular function desired, the performance requirements, and any personal preferences.

Extending IDS with SPL

SPL, or Stored Procedure Language, is a very simple scripting language that has been available with IDS for a number of years, and can be used in some limited fashion for extensibility. For example, you cannot create an opaque type in SPL and you have limited abilities to write user-defined aggregates.

However, despite these limitations you can still write some interesting user-defined functions. A few examples are the following:

- ▶ Date manipulation: The ability to work on data by grouping it based on such things as the calendar or business quarter, week of the year, day of the year, day of the week and so on.
- ▶ Conversion functions: If there is a need to convert between feet and meters, Fahrenheit and Celsius, gallons and liters, or any other conversions that might be useful in your business.

If you wanted to implement a `quarter()` function based on the calendar year, you could create the following function:

```
CREATE FUNCTION quarter(dt date)
RETURNS integer
WITH(NOT VARIANT)

RETURN (YEAR(dt) * 100) + 1 + (MONTH(dt) - 1) / 3;

END FUNCTION;
```

In this case, the function returns an integer that is the year times 100 and the addition of the quarter value. This means that the representation can easily be sorted by year and quarter. The integer 200603 then represents the third quarter of the year 2006. The type of the return value is an implementation choice.

The `quarter()` function implementation takes advantage of two built-in SQL functions: `YEAR` and `MONTH`. This makes the implementation of `quarter()` trivial. After the `CREATE FUNCTION` statement completes, the new `quarter()` function can be used in SQL just like any other built-in function. For example, to get the total revenue for each quarter of the year 2005, you can write the SQL statement:

```
SELECT quarter(order_date), SUM(amount)
FROM transactions
WHERE quarter(order_date) BETWEEN 200501 AND 200504
GROUP BY 1
ORDER BY 1;
```

If a different quarter function is needed because the business year starts 01 June, you can easily write the following function:

```
CREATE FUNCTION bizquarter(dt date)
RETURNS integer
WITH(NOT VARIANT)

DEFINE yr int;
DEFINE mm int;

LET yr = YEAR(dt);
LET mm = MONTH(dt) + 7; -- June to Jan. is 7 months
IF mm > 12 THEN
    LET yr = yr + 1;
    LET mm = mm - 12;
END IF

RETURN (yr * 100) + 1 + (mm - 1) / 3;

END FUNCTION;
```

You simply add seven months to the input date, because 01 June 2005 is in the first quarter of 2006. The `bizquarter()` function is used the same way as the `quarter()` function. You can create an index on the result of a function. For example:

```
CREATE INDEX trans_quarter_idx
ON transactions( quarter(order_date) );
```

This way, the optimizer could decide to use the index path to solve the SELECT statement above.

For more information about date manipulation functions in SPL, refer to the article *Date processing in Informix Dynamic Server* that we list in “Related publications” on page 387.

SPL is a very simple scripting language. Because of its simplicity, it can be seen as limited. Keep in mind that any function that is defined in the database can be used in an SPL function. This includes all the IDS built-in operators and functions, the IDS defined constants, and any already defined user-defined functions and aggregates. You can find a list of operators, functions, and constants in the *Informix Guide to SQL: Syntax*.

Extending IDS with Java

IDS supports Java as a language to write user-defined functions, aggregates and opaque types. The application programming interface (API) is tailored on the JDBC interface. This makes it easy for Java programmers to learn to write extensions for IDS.

The limitations to Java UDRs are the following:

- ▶ **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.
- ▶ **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.

These types of functions must be written in C.

Before starting to use Java UDRs, you must configure IDS for Java support. In brief, you must:

- ▶ Include a default sbpace (SBSPACENAME onconfig parameter)
- ▶ Create a jvp.properties file in \$INFORMIXDIR/extend/krakatoa
- ▶ Add or modify the Java parameters in your onconfig file
- ▶ Optionally, set some environment variables

These steps are described in more detail in Chapter 11, “IDS delivers services (SOA)” on page 311.

A Java UDR is implemented as a static method within a Java Class. For example, to create a quarter() function in Java, use the code provided in Example 4-7.

Example 4-7 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, simply use the **javac** command from Java Development Kit (JDK™). For Util.java, you can use the following command:

```
javac Util.java
```


If you are using JDBC features, or some special classes such as one that keep tracks of state information in an iterator function, you need to add two jar files in your classpath. The following command illustrated their use:

```
javac -classpath  
$INFORMIXDIR/extend/krakatoa/krakatoa.jar;$INFORMIXDIR/extend/krakat  
oa/jdbc.jar Util.java
```

After you compile the Java code, you must put it in a Java archive file (jar) 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, could be as shown in Example 4-8.

Example 4-8 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, called Util.mf in our example, can be as follows:

```
Manifest-Version: 1.0  
Name: Util.txt  
SQLJDeploymentDescriptor: TRUE
```

With these files, you can create the jar file with the following 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 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 that is 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:

```
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 can be more desirable to use SPL or C, rather than Java. By its very nature, Java requires more resources to run than SPL or C. However, this does not mean it would necessarily be a wrong 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. Because the UDR runs in a Java virtual machine you can also use any class library to do processing, such as parsing 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.

Extending IDS with C

C has the most extensive interface for extensibility, the DataBlade API. This API contains a comprehensive set of functions. The following list subjects that are supported by this API:

- ▶ Byte operations
- ▶ Callbacks
- ▶ Character processing
- ▶ Collections
- ▶ Connecting and disconnecting
- ▶ Converting and copying
- ▶ Conversion
- ▶ Date, datetime, and intervals
- ▶ Decimal operations
- ▶ Exception handling
- ▶ Functions execution
- ▶ Memory management
- ▶ OS file interface
- ▶ Parameters and environment
- ▶ Row processing
- ▶ Smart large objects
- ▶ Thread management
- ▶ Tracing
- ▶ Transactions

It also includes the ability to create a table interface to arbitrary data and the ability to create new indexing methods.

This functionality has allowed the creation of many interesting extensions. Some have been previously mentioned, but you can go as far as generating events out of the database, based on transactions. This is discussed in the article *Event-driven fined-grained auditing with Informix Dynamic Server*, which we list in “Related publications” on page 387.

IDS accesses a C UDR through a shared library. The first time the UDR is called, the library is loaded. From that point on, the UDR is part of the server.

The first step in creating a C UDR is to write the code. Example 4-9 shows the code that implements a `quarter()` function in C.

Example 4-9 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;
}

```

In this example, 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 following line defines the `quarter()` function 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, among other things.

The rest of the function is straightforward. We extract the month, day, and year from the date argument using the ESQ/C `rjulmdy()` function, 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 indicating 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 which 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 4-10 shows a simple makefile to create `quarter.bld`.

Example 4-10 Makefile

```
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
    $(SHLIBOD) $(LINKFLAGS) -o quarter.bld quarter.o
```

To use this makefile on another platform, you simply need to change the include file name on the first line from `makeinc.linux` to `makeinc` with the suffix from another platform.

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 because 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. When the function is defined, it can be used in SQL statement or called directly as we saw 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);
```

DataBlade Development Kit

For those new to IDS extensibility, or if a project requires more than just a few types and functions, IDS provides the DataBlade Development Kit (DBDK) for the Windows platform.

DBDK is a graphical user interface (GUI) that includes the following parts:

- ▶ **BladeSmith:** Helps you manage the project. It also assists in the creation of the function based on the definition of the arguments and return value. It also generates header files, makefiles, functional test files, SQL scripts, messages, and packaging files.
- ▶ **DBDK Visual C++® add-in and lfxQuery:** integrates into Visual C++ to automate many of the debugging tasks and run unit tests.
- ▶ **BladePack:** Can create a simple directory tree that includes files to be installed. The resulting package can be registered easily in a database using BladeManager.
- ▶ **BladeManager:** A tool that is included with IDS on all platforms. It simplifies the registration or de-registration of DataBlades.

Chapter 11, “IDS delivers services (SOA)” on page 311 includes a simple example of the use of BladeSmith and BladePack. Chapter 5, “Functional extensions to IDS” on page 175 shows how to use BladeManager to register DataBlades into a database.

The DataBlade Development Kit is very useful to generate the skeleton of a project. When this is done, the project files can be moved into a different environment including source control and concurrent access control. Anyone new to IDS extensibility should take some time to get familiar with this tool to facilitate the learning curve and improve productivity.

4.4 A case for extensibility

Many companies take the approach that application code should run against any database server without source modifications. This approach implies that applications should use the highest common denominator between all database products. This choice results in an application environment that cannot take

advantage of any vendor added value. It is, at best, an average set of available SQL features.

A data server should be seen as a strategic asset that gives a company a business advantage. Just as improvements in business processes, management techniques or proprietary algorithms can give a company a business advantage, data server features should contribute to making companies more efficient, effective, flexible and agile.

The examples in this chapter provide a glimpse at the advantages of IDS extensibility. Take the time to analyze your environment to determine how you could use extensibility to gain a competitive edge. Sometimes a very simple addition can result in a very noticeable benefit. It is worth the effort.

Functional extensions to IDS

IBM provides plug-in extensions that extend the capabilities of the IDS data server. These extensions, called DataBlade *modules*, can reduce application complexity and improve their performance because they provide solutions for specific problem domains. This way, IDS adapts better to the business environments that must address these problems.

These IDS modules come as built-in DataBlades, free-of-charge DataBlades, and DataBlades available for a fee. In this chapter, we discuss the standard procedure for installing DataBlades and describe the DataBlade modules that are available currently with IDS 10.0. Consult the most current release notice when a DataBlade is desired, to see if new ones have been added recently.

5.1 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. You must first install any other DataBlade modules in the data server. The installation process is described in detail in the manual *DataBlade Modules Installation and Registration Guide*, G251-2276.

In general, the installation requires the following steps:

1. Unload the files into a temporary directory. This might require the use of a utility such as `cpio`, `tar`, or an extraction utility, depending on the platform that you use.
2. Execute the installation command. This command is usually called `install` on UNIX-type platforms (or `rpm` on Linux) and `setup` on Windows platforms.

After the installation, there will be 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.8.20.UC2`.

A DataBlade module must be registered into a database before it is available for use. The registration might create new types, user-defined functions, and even tables and views. The registration process is made easy through the use of the DataBlade Manager utility (`blademgr`). Example 5-1 depicts the process for registering the spatial DataBlade module in a database called *demo*.

Example 5-1 DataBlade registration

```
informix@ibmswg01:~> blademgr
ids10>list demo
There are no modules registered in database demo.
ids10>show modules
4 DataBlade modules installed on server ids10:
          LLD.1.20.UC2          ifxbuiltins.1.1
          ifxrltree.2.00       spatial.8.20.UC2
If a module does not show up, check the prepare log.
ids10>register spatial.8.20.UC2 demo
Register module spatial.8.20.UC2 into database demo? [Y/n]
Registering DataBlade module... (may take a while).
Module spatial.8.20.UC2 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.8.20.UC2 was successfully registered in database
demo.
ids10>ids10>bye
Disconnecting...
informix@ibmswg01:~>
```

In this example, we started by executing `blademgr`. The prompt indicates with which instance we are working. This example uses the `ids10` instance. The first command executed is `list demo`, which looks into the `demo` database to see if any DataBlade modules are already installed. The next command is `show modules`, which provides a list of the DataBlade modules that are installed in the server under the `$INFORMIXDIR/extend` directory. The names correspond to the directory names in the `extend` directory. The DataBlade Manager utility looks into the directories to make sure that they are proper DataBlade directories. Thus, other directories could exist under the `extend` directory but would not be listed by the `show modules` command.

The registration of a DataBlade is done with the `register` command. Upon execution, it looks for dependencies with 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 `blademgr`.

That is all there is to registering a DataBlade module. After the DataBlade module is registered, you can start using it in the specified database.

There is also a graphical user interface version of the DataBlade Manager utility available on Windows. You can see a brief example of its use in 11.3.9, “A simple IDS 10 / gSOAP Web service consumer example” on page 356.

5.2 Built-in DataBlades

The `$INFORMIXDIR/extend` directory includes several sub-directories that relate to managing the registration of DataBlade modules. It also includes the `ifxrltree.2.00` that is used to register the messages related to the R-tree index interface.

IDS currently defines two built-in datablades: large object locator and MQ. Additional built-in DataBlades are expected in future releases of IDS.

5.2.1 The Large Object Locator module

The Large Object Locator DataBlade module appears as LLD.1.20.UC2 in the extend directory. With this module, a user can have a consistent interface if the design of the application calls for storing some of the large objects in the database (such as CLOBs or BLOBs) and others outside the database in the file system.

LLD includes three interfaces:

- ▶ SQL interface: The SQL interface is a set of functions that are used in SQL statements to manipulate the LLD objects. This includes loading an LLD object from a file or writing an LLD object to a file.
- ▶ An API library. This interface can be used when there is a need to manipulate the LLD objects in user-defined functions written in C.
- ▶ An SQL/C library. This library allows ESQL/C client programs to manipulate LLD objects.

With the removal of the four terabyte instance storage limit in IDS 9.40 and above, there is less of a need to store large objects in the file system. It is easier to manage the large objects in the database. For example, objects stored outside the database cannot be rolled back if a transaction fails for any reason. There can still be a need to access a mix of large objects inside and outside the database. When that is the case, the large object locator can make an implementation much easier.

5.2.2 The MQ DataBlade module

The MQ DataBlade module, listed as mqblade.2.0 under the extend directory, is available on IDS 10.0 xC3 and above. The platforms supported in xC3 are AIX, HP-UX, Solaris, and Windows, all in their 32-bit implementation. The xC4 release adds support for AIX and HP-UX 64-bit implementations. At the time of this writing, the current release is xC5 and it increases the platforms that are supported with Linux 32-bit, Linux 64-bit on System p™ machines, and Solaris 64-bit implementations.

The MQ DataBlade offers an interface between the IDS data server and the WebSphere MQ messaging products installed on the same machine as IDS. WebSphere MQ products are key components of the IBM enterprise service bus (ESB) to support the service-oriented architecture (SOA). WebSphere MQ enables the exchange of asynchronous messages in a distributed, heterogeneous environment.

The interface between IDS and WebSphere MQ is transactional and uses the two-phase commit protocol. This means that when a transaction that includes a

message queue operations is rolled back, the message queue operation is also rolled back.

The availability of the MQ DataBlade module makes available a new way to integrate IDS within the enterprise environment. It also provides an easy interface to integrate multiple pieces of information. An application might need to read a row in a database and gather additional information coming from another data source that happens to be available on a message queue. With the MQ DataBlade module, it is possible to complete the operation within the database and take advantage of the joining capabilities of the data server instead of replicating that capability in the application code.

The use of the MQ DataBlade could also be a way to shield developers from having to learn an additional interface. The access to message queues can be kept in the database. It is done through function calls or even through a table interface. An application could read a message queue with a simple SQL statement such as:

```
SELECT * FROM vtiMQ;
```

By the same token, writing to a message queue can be as simple as using an INSERT statement:

```
INSERT INTO vtiMQ(msg) VALUES("<insert message here>");
```

So, if the programmer knows how to execute SQL statements from an application, then message queues can be easily manipulated. The MQ DataBlade module supports the various ways to operate on message queues, such as read, receive, publish, and subscribe.

Table 5-1 lists the functions that are available with the MQSeries® DataBlade, along with brief descriptions of those functions.

Table 5-1 List of MQ functions in IDS

Function	Description
MQSend()	Send a string message to a queue
MQSendClob()	Send CLOB data to a queue
MQRead()	Read a string message in the queue into IDS without removing it from the queue
MWReadClob()	Read a CLOB in the queue into IDS without removing it from the queue
MQReceive()	Receive a string message in the queue into IDS and remove it from the queue

Function	Description
MQReceiveClob()	Receive a CLOB in the queue into IDS and remove it from the queue
MQSubscribe()	Subscribe to a Topic
MQUnSubscribe()	UnSubscribe from a previously subscribed topic
MQPublish()	Publish a message into a topic
MQPublishClob()	Publish a CLOB into a topic
CreateMQVTIRead()	Create a read VTI table and map it to a queue
CreateMQVTIReceive()	Create a receive VTI table and map it to a queue
MQTrace()	Trace the execution of MQ Functions
MQVersion()	Get the version of MQ Functions

For more information about the MQ DataBlade module, consult *Built-In DataBlade Modules User's Guide*, G251-2770.

5.3 Free-of-charge DataBlades

The Spatial DataBlade is currently the only free DataBlade module. It is not included with IDS. You can download separately from the following URL:

<http://www14.software.ibm.com/webapp/download/search.jsp?go=y&rs=ifxsdb-hold&status=Hold&sb=ifxsdb>

5.3.1 The Spatial DataBlade module

The Spatial DataBlade module is available on AIX, HP-UX, and Solaris, and all are available in 32-bit and 64-bit versions. It is also available on SGI, Linux, and Windows in 32-bit versions.

The spatial DataBlade implements a set of data types and functions that allow for the manipulation of spatial data within the database. With it, some business related questions become easier to answer. As examples:

- ▶ Where are my stores located, related 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 to 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 customers locations (among other things)?

Locations and spatial objects are represented with new data types in the database, such as: `ST_Point`, `ST_LineString`, `ST_Polygon`. Functions such as `ST_Contains()`, `ST_Intersects()`, and `ST_Within()` operate on these data types.

With these types and functions, you can answer a question such as: Which hotels are within three miles of my current location?

The following SQL statement illustrates how this could be answered:

```
SELECT Y.Name, Y.phone, Y.address
FROM e_Yellow_Pages Y
WHERE ST_Within(Y.Location,
               ST_Buffer(:GPS_Loc, (3 * 5280) ) )
```

The `WHERE` clause of this query evaluates if `Y.Location` is within a buffer of 3 miles from the location that is passed as argument `(:GPS_Loc)`.

The performance of this query could improve even more if it could take advantage of an index. The spatial DataBlade supports the indexing of spatial types through the use of the multi-dimensional R-Tree index. The creation of an index on the location column would be as follows:

```
CREATE INDEX eYP_loc_idx
ON e_Yellow_pages(Location ST_Geometry_ops)
USING RTREE;
```

The operator class `ST_Geometry_ops` defines the functions that might be able to use the index. They include the functions mentioned previously as well as `ST_Crosses()`, `ST_Equals()`, `SE_EnvelopesIntersect()`, `SE_Nearest()`, `SE_NearestBbox()`, `ST_Overlaps()`, and `ST_Touches()`.

Without this functionality, it becomes more difficult to manage spatial information. It could result in more complex applications and unnecessary data transfer. The spatial DataBlade module can reduce application complexity and greatly improve performance.

The spatial DataBlade module is designed to operate on a flat map model. This is sufficient for a large number of spatial applications. For more specialized application that must have a high level of precision over large distances, the geodetic DataBlade module would likely be a better fit. This DataBlade takes into consideration the curvature of the earth to answer those specialized needs. The

Geodetic DataBlade module is described in 5.4.2, “The Geodetic DataBlade module” on page 183.

5.4 Chargeable DataBlades

At the time of this writing, the following DataBlades modules are supported by IDS 10.0: Excalibur Text Search, Geodetic, TimeSeries, TimeSeries Real Time Loader, and Web. Other DataBlades such as C-ISAM, Image Foundation, and Video might become certified at a later date.

5.4.1 Excalibur Text search

People store text in the database descriptions, remark fields, and a variety of documents using different formats (such as PDF and Word). A support organization is a simple example of this, because they need to store a description of their interactions with customers. This includes the problem description and the solution. Other examples include product descriptions, regulatory filings, news repositories, and so forth.

When an organization has a need to search the content of documents they cannot depend simply on keywords. They need to use more complex search criteria. For example, how could you answer the following question: Find all the documents that discuss using IDS with IDS with WebSphere Application Server Community Edition.

This search causes numerous issues, such as:

- ▶ What if the document says Informix Dynamic Server rather than IDS? Similarly, WebSphere Application Server Community Edition rather than IDS with WebSphere Application Server Community Edition?
- ▶ Are you looking for an exact sentence or just a number of words?
- ▶ What happens if the document describes *the use* of IDS rather than *using* IDS?

These are just a few of the issues that the Excalibur Text DataBlade module is designed to solve. Excalibur Text provides a way to index the content of documents so the entire content can be searched quickly. It provides features such as:

- ▶ Filtering: This strips documents of their proprietary formatting information before indexing.
- ▶ Synonym list: This is where you can determine that IDS and Informix Dynamic Server are the same.

- Stop word list: These are words that are excluded from indexing. This can include words such as: the, and, or, and so forth. These words usually add no meaning to the search so they do not need to be indexed.

When an Excalibur Text index is created, SQL queries can use it in statements such as:

```
SELECT id, description FROM repository
WHERE etx_contains(description,
ROW("Using IDS with WAS CE",
"SEARCH_TYPE = PHRASE_APPROX"));
```

Many more complex searches can be executed using this DataBlade module. Any company that has a need for these types of searches could greatly benefit from Excalibur Text.

5.4.2 The Geodetic DataBlade module

The Geodetic DataBlade module manages Geographical Information System (GIS) data in IDS 10.0. It does so by treating the earth as a globe. This is an important difference from the spatial DataBlade module. Because 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-coordinate and y-coordinate 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.

The Geodetic DataBlade module defines a set of domain-specific data types and functions. It also supports indexing of the GeoObjects. This means that an index could be used to solve queries that include the functions `beyond()`, `inside()`, `intersect()`, `outside()`, and `within()`.

Consider the following SQL statement:

```
SELECT * FROM worldcities
WHERE Intersect(location,
'((37.75,-122.45),2000,any,any)':GeoCircle);
```

This query selects the cities that intersect within a circle of radius 2000 meters, at the longitude and latitude listed. The additional arguments listed as *any* represent additional available information of altitude and time. This query would obviously be very selective considering the size of the circle. An index on the location column would eliminate most of the rows from the table and return much faster than without the index.

The Geodetic DataBlade module is best used for global data sets and applications.

5.4.3 The Timeseries DataBlade module

Some companies have a need to analyze data in order based on the time when the reading occurred. For example, this is the case for the financial industry where they do analysis on the price variation of a stock over time to determine if it is a good investment. This could also be used to analyze computer network traffic for load prediction, or in scientific research to keep track of multiple variables changing over time.

Keeping this type of information in relational tables raises at least two major issues. First, it duplicates information. For example, the information about a stock price must always include which stock is being considered. Each row must have this additional information.

Second, a relational table is a set of unordered rows. Any query must order the timeseries data before it can be processed.

Using a non-relational product is also problematic, because they might have limited capabilities. Also, there is likely to be a need to merge the timeseries data with relational data during the analysis. A non-relational product makes this task difficult.

The Timeseries DataBlade module provides a solution to optimize the storage and processing of data based on time. It includes such features as user-defined timeseries data types, calendars and calendar patterns, regular and irregular timeseries, and a set of functions to manipulate the data.

Timeseries can be manipulated in SQL, Java, and C. For a good introduction to the Timeseries DataBlade, read the article titled *Introduction to the TimeSeries DataBlade* located at the following URL:

<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0510durty2/index.html>

5.4.4 Timeseries Real Time Loader

The Timeseries Real Time Loader is a toolkit that complements the Timeseries DataBlade module by adding the capability to load a large volumes of time-based data and make the data available for analysis in real-time.

The system manages the incoming data through data feeds. A data feed is a custom implementation that is able to manage a specific data source using a

well-defined format. The Timeseries Real Time Loader can manage up to 99 different data feeds.

IDS communicates with a data feed through shared memory segments. When an SQL query accesses data in the server, it can also use the data that is available in memory. This means that the data is available for query as soon as it gets into memory.

If you have a need to handle high volumes of incoming time-based data, the Timeseries Real Time Loader DataBlade module, with the timeseries DataBlade module, might be the answer to your needs.

5.4.5 The Web DataBlade module

The Web DataBlade module provides a way to create Web applications that generate dynamic HTML pages. These pages are created based on information stored in IDS. 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 allows 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 could as easily generate XML documents.

The appPages are interpreted by a user-defined function called `WebExplode()`. This function parses the appPage and dynamically builds and executes the SQL statements and processing instructions embedded in the appPage tags.

The standard way to obtain the resulting document from the execution of an appPage is through the `webdriver()` client application. This application is included in the Web DataBlade module and comes in four implementations:

- ▶ NSAPI: This implementation is written with the Netscape Server API and is used only with Netscape Web Servers.
- ▶ Apache: This implementation is written with the Apache API for Apache Web Servers.
- ▶ SAPI: This implementation is for Microsoft Internet Information Servers.
- ▶ CGI: this implementation uses the standard Common Gateway Interface (CGI) and can be used by all Web servers.

The Web DataBlade module also includes the appPage builder application. This application facilitates the creation and management of appPages through a browser interface.

The Web DataBlade module is a key component for several customers. It provides an easy to use way to create Web sites and minimize the number of different Web pages needed. This helps to better manage the Web.

5.5 Conclusion

The use of DataBlade modules can greatly enhanced the capabilities of IDS, and better serve your business purpose. They can provide benefits such as better performance and scalability, and faster time to market. A DataBlade module addresses requirements of a specific problem-domain. You can also use multiple DataBlade modules together as building blocks to solve your business problems.

Make sure to understand what DataBlade modules are available and use them to give your company a business advantage. For more ways to create business advantages, read Chapter 4, “Extending IDS for business advantages” on page 151.

Development tools and interfaces

In this chapter we provide an overview of all of the development tools and application programming interfaces (APIs) that IBM supports currently and a selection of additional Open Source and third-party tools that you can use in combination with IDS V10.

6.1 IDS V10 software development overview

Software developers have different backgrounds and different requirements to develop database-oriented applications. They basically need choices so that they can choose the best development tool or API for a specific development job. It is this flexibility and availability of a variety of functions and features that enables developers to be creative, and develop powerful and functionally rich applications.

To motivate software developers to use the advanced features of IDS V10 actively, IBM is offering a broad selection of IDS programmer APIs and development tools that complement the IDS V10 database server offering and that provide choice to the IDS developer community. In this chapter, we introduce the new IDS V10 APIs and tools and provide a small programming example for each.

6.1.1 IBM supported APIs and tools for IDS V10

The majority of IBM Informix development APIs have been grouped together into the IBM Informix Client Software Development Kit (CSDK) with an associated runtime deployment component called *IBM Informix Connect*.

In addition, there are quite a few additional, stand-alone APIs/tools that we also discuss throughout this chapter.

Even though we introduce each development API/tool in a separate section, we note all the tools that are part of the CSDK in their section headers for easy identification.

6.1.2 Embedded SQL for C (ESQL/C) - CSDK

ESQL/C allows for the easy integration of SQL statements with C programming language applications. The SQL statement handling is a combination of an ESQL/C language pre-processor which takes the ESQL/C statements and converts them into ESQL/C library function calls in combination with an ESQL/C runtime library.

This approach can be very helpful when dealing with many SQL related activities in a C based application, and needs to focus on the SQL programming more than on knowing how to call any kind of complex call level interface to achieve the same goal. Even though ESQL/C provides a very tight integration with C applications, it still allows you to focus on the actual SQL problem solution.

Informix ESQL/C supports the ANSI standard for an embedded SQL for C which also makes ESQL/C a good technology foundation for any database application migrations to IDS V10.

ESQL/C source code files typically have the extension *.ec*. In the first processing step the ESQL/C pre-processor converts all of the embedded ESQL/C statements into C function calls and generates a *.c* file, which is compiled eventually either to an object file or an executable.

You can embed SQL statements in a C function with one of two formats:

- The EXEC SQL keywords:

```
EXEC SQL SQL_statement;
```

Using EXEC SQL keywords is the ANSI-compliant method to embed an SQL statement.

- The dollar sign (\$) notation:

```
$SQL_statement;
```

It is recommended that you use the EXEC SQL format to create more portable code, when required. To get a better impression of how ESQL/C looks, Example 6-1 shows one of the ESQL/C that ships with the product.

Example 6-1 Example ESQL/C code

```
#include <stdio.h>

EXEC SQL define FNAME_LEN 15;
EXEC SQL define LNAME_LEN 15;

main()
{
EXEC SQL BEGIN DECLARE SECTION;
    char fname[ FNAME_LEN + 1 ];
    char lname[ LNAME_LEN + 1 ];
EXEC SQL END DECLARE SECTION;

    printf( "Sample ESQL Program running.\n\n");
EXEC SQL WHENEVER ERROR STOP;
EXEC SQL connect to 'stores_demo';

EXEC SQL declare democursor cursor for
    select fname, lname
    into :fname, :lname
    from customer
    order by lname;
```

```

EXEC SQL open democursor;
for (;;)
{
EXEC SQL fetch democursor;
if (strncmp(SQLSTATE, "00", 2) != 0)
    break;

printf("%s %s\n", fname, lname);
}

if (strncmp(SQLSTATE, "02", 2) != 0)
    printf("SQLSTATE after fetch is %s\n", SQLSTATE);

EXEC SQL close democursor;
EXEC SQL free democursor;

EXEC SQL disconnect current;
printf("\nSample Program over.\n\n");

return 0;
}

```

6.1.3 The IBM Informix JDBC 3.0 driver - CSDK

Java database connectivity (JDBC) is the Java specification of a standard application programming interface (API) that allows Java programs to access database management systems. The JDBC API consists of a set of interfaces and classes written in the Java programming language. Using these standard interfaces and classes, programmers can write applications that connect to databases, send queries written in structured query language (SQL), and process the results.

The JDBC API defines the Java interfaces and classes that programmers use to connect to databases and send queries. A JDBC driver implements these interfaces and classes for a particular DBMS vendor. There are four types of JDBC drivers:

- ▶ Type 1: JDBC-ODBC bridge plus ODBC driver.
- ▶ Type 2: Native-API, partly Java driver.
- ▶ Type 3: JDBC-Net, pure-Java driver.
- ▶ Type 4: Native-protocol, pure-Java driver.

For more information about this topic, see the *Informix JDBC Driver - Programmer's Guide*, Part No.000-5354.

The Informix JDBC 3.0 driver is an optimized, native-protocol, pure-Java driver (Type 4). A type 4 JDBC driver provides direct connection to the Informix database server, without a middle tier and is typically used on any platform providing a standard Java virtual machine.

The current Informix JDBC 3.0 driver is based on the JDBC 3.0 standard, provides enhanced support for distributed transactions and is optimized to work with IBM WebSphere Application Server. It promotes accessibility to IBM Informix database servers from Java client applications, provides openness through XML support (JAXP), fosters scalability through its connection pool management feature, and supports extensibility with a user-defined data type (UDT) routine manager that simplifies the creation and use of UDTs in IDS V10.

This JDBC 3.0 driver also includes Embedded SQL/J which supports embedded SQL in Java.

The minimum Java runtime/development requirements are JRE or JDK 1.3.1 or higher. And, a JRE/JDK 1.4.2 is recommended. Example 6-2 shows a sample Java method.

Example 6-2 A simple Java method that uses JDBC 3.0 API

```
private void executeQuery()
{
    // The select statement to be used for querying the customer table
    String selectStmt = "SELECT * FROM customer";

    try
    {
        // Create a Statement object and use it to execute the query
        Statement stmt = conn.createStatement();
        queryResults = stmt.executeQuery(selectStmt);

        System.out.println("Query executed...");
    }
    catch (Exception e)
    {
        System.out.println("FAILED: Could not execute query....");
        System.out.println(e.getMessage());
    }
} //end of executeQuery method
```

Example 6-3 shows a simple SQL/J code fragment.

Example 6-3 A simple SQL/J code fragment supported by the Informix JDBC 3.0 driver

```
void runDemo() throws SQLException
{
    drop_db();

    #sql { CREATE DATABASE demo_sqlj WITH LOG MODE ANSI };

    #sql
    {
        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)
        )
    };

    try
    {
        #sql
        {
            INSERT INTO customer VALUES
            ( 101, "Ludwig", "Pauli", "All Sports Supplies",
              "213 Erstwild Court", "", "Sunnyvale", "CA",
              "94086", "408-789-8075"
            )
        };

        #sql
        {
            INSERT INTO customer VALUES
            ( 102, "Carole", "Sadler", "Sports Spot",
              "785 Geary St", "", "San Francisco", "CA",
              "94117", "415-822-1289"
            )
        };
    }
}
```

```

        )
    };
}
catch (SQLException e)
{
    System.out.println("INSERT Exception: " + e + "\n");
    System.out.println("Error Code           : " +
                       e.getErrorCode());
    System.err.println("Error Message       : " +
                       e.getMessage());
}

```

6.1.4 IBM Informix .NET provider - CSDK

.NET is an environment that allows you to build and run managed applications. A managed application is one in which memory allocation and de-allocation are handled by the runtime environment. Another good example for a managed environment is a Java virtual machine (JVM™).

The .NET key components are:

- ▶ Common Language Runtime
- ▶ .NET Framework Class Library, such as ADO.NET and ASP.NET

ADO.NET is a set of classes that provide access to data sources and has been designed to support disconnected data architectures. A DataSet is the major component in that architecture, and is an in-memory cache of the data retrieved from the data source.

ADO.NET differs from ODBC and OLE DB, and each provider exposes their own classes that inherit from a common interface. As examples, IfxConnection, OleDbConnection, and OdbcConnection.

The Informix .NET provider

The IBM Informix .NET Provider is a .NET assembly that lets .NET applications access and manipulate data in IBM Informix databases. It does this by implementing several interfaces in the Microsoft .NET Framework that are used to access data from a database.

Using the IBM Informix .NET Provider is more efficient than accessing an IBM Informix database using either of these two methods:

- ▶ The Microsoft .NET Framework Data Provider for ODBC along with the IBM Informix ODBC Driver
- ▶ The Microsoft .NET Framework Data Provider for OLE DB along with the IBM Informix OLE DB Provider

Any application that can be executed by the Microsoft .NET Framework can use the IBM Informix .NET Provider. Here are some examples of programming languages that create applications that meet this criteria:

- ▶ Visual BASIC .NET
- ▶ Visual C# .NET
- ▶ Visual J# .NET
- ▶ ASP.NET

Figure 6-1 shows how the Informix .NET provider fits into the overall .NET framework.

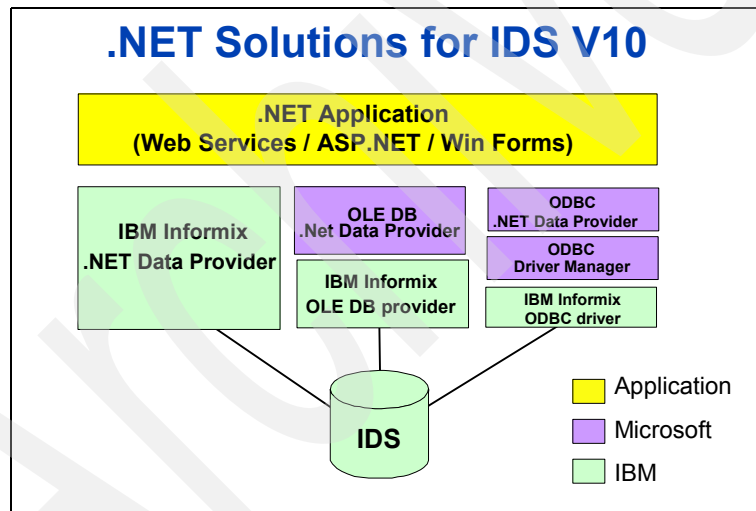


Figure 6-1 The Informix .NET provider and the .NET framework

The IBM Informix .NET Provider runs on all Microsoft Windows platforms that provide full .NET support. You must have the Microsoft .NET Framework SDK, Version 1.1, or later and version 2.90, or later, of the IBM Informix Client SDK, installed on your machine.

Example 6-4 shows a simple .NET code snippet that accesses IDS V10 to select some data from the customer table in the stores_demo database.

Example 6-4 A simple Informix .NET driver based code fragment

```
public void getCustomerList()
{
    try
    {
        // Create an SQL command
        string SQLcom = "select fname,lname from customer";
        IfxCommand SelCmd = new IfxCommand(SQLcom, ifxconn);
        IfxDataReader custDataReader;
        custDataReader = SelCmd.ExecuteReader();
        while (custDataReader.Read())
        {
            Console.WriteLine (" Customer Fname " +
                               custDataReader.GetString(1));
        }
        custDataReader.Close();
        // Close the connection
    }
    catch(Exception e)
    {
        Console.WriteLine("Get exception " + e.Message.ToString());
    }
    Console.Read();
}
```

6.1.5 IBM Informix ODBC 3.0 driver - CSDK

The IBM Informix Open Database Connectivity (ODBC) driver is based on the Microsoft ODBC 3.0 standard, which by itself is based on Call Level Interface specifications developed by X/Open and ISO/IEC. The ODBC standard has been around for a long time and is still widely used in database oriented applications.

The current IBM Informix ODBC driver is available for Windows, Linux and UNIX platforms and supports pluggable authentication modules (PAM) on UNIX and Linux plus LDAP authentication on Windows.

IBM Informix ODBC driver-based applications enable you to perform the following operations:

- ▶ Connect to and disconnect from data sources
- ▶ Retrieve information about data sources

- ▶ Retrieve information about IBM Informix ODBC Driver
- ▶ Set and retrieve IBM Informix ODBC Driver options
- ▶ Prepare and send SQL statements
- ▶ Retrieve SQL results and process the results dynamically
- ▶ Retrieve information about SQL results and process the information dynamically

Figure 6-2 shows a typical execution path of an Informix ODBC 3.0 based application.

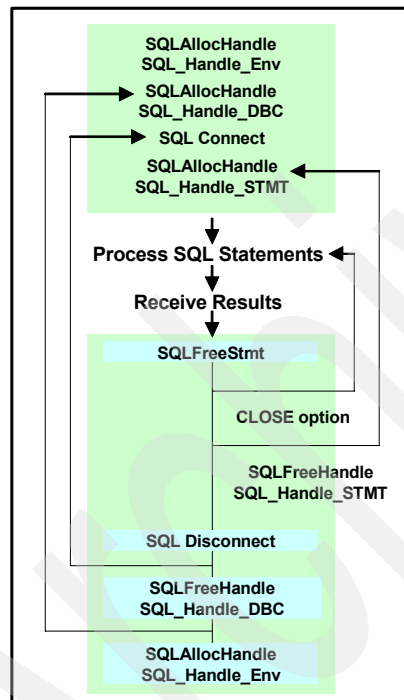


Figure 6-2 Typical execution path

Many third-party applications, such as spreadsheets, word processing, or analytical software, support at least ODBC connectivity to databases. So the Informix ODBC driver might be the best option to connect such applications with IDS V10.

The most recent version of the IBM Informix ODBC driver (V2.9) supports the following features:

- ▶ Data Source Name (DSN) migration
- ▶ Microsoft Transaction Server (MTS)
- ▶ Extended data types, including rows and collections:
 - Collection (LIST, MULTISSET, SET)
 - DISTINCT
 - OPAQUE (fixed, unnamed)
 - Row (named, unnamed)
 - Smart large object (BLOB, CLOB)
 - Client functions to support some of the extended data types
- ▶ Long identifiers
- ▶ Limited support of bookmarks
- ▶ GLS data types
 - NCHAR
 - NVCHAR
- ▶ Extended error detection
 - ISAM
 - XA
- ▶ Unicode support
- ▶ XA support
- ▶ Internet Protocol Version 6 support for internet protocols of 128-bits

6.1.6 IBM Informix OLE DB provider - CSDK

Microsoft OLE DB is a specification for a set of data access interfaces designed to enable a variety of data stores to work together seamlessly. OLE DB components are data providers, data consumers, and service components. Data providers own data and make it available to consumers. Each provider's implementation is different, but they all expose their data in a tabular form through virtual tables. Data consumers use the OLE DB interfaces to access the data.

You can use the IBM Informix OLE DB provider to enable client applications, such as ActiveX® Data Object (ADO) applications and Web pages, to access data on an Informix server.

Due to the popularity of the Microsoft .NET framework, Informix developers on the Microsoft platform typically prefer the .NET database provider and integrating

existing OLE DB based applications through a MS .NET provider for OLE DB. Example 6-5 shows a code sample for the OLEDB provider.

```
int main()
{
    const char *DsnName = "Database@Server";
    const char *UserName = "UserID";
    const char *PassWord = "Password";

    DbConnect MyDb1;
    HRESULT hr = S_OK;
    int tmp = 0;

    WCHAR wSQLcmd[MAX_DATA];

    CoInitialize( NULL );

    // Create DataSource Object and Opent A Database Connection
    if (FAILED(hr = MyDb1.MyOpenDataSource( (REFCLSID) CLSID_IFXOLEDBC,
        DsnName, UserName, PassWord )) )
    {
        printf( "\nMyOpenDataSource() failed");
        return( hr );
    }

    if (FAILED( hr = MyDb1.MyCreateSession() ) )
    {
        printf( "\nMyCreateSession Failed" );
        return( hr );
    }

    if (FAILED( hr = MyDb1.MyCreateCmd() ) )
    {
        printf( "\nMyCreateCmd Failed" );
        return( hr );
    }

    swprintf( wSQLcmd, L"DROP TABLE MyTable" );
    MyDb1.MyExecuteImmediateCommandText( wSQLcmd );

    swprintf( wSQLcmd,
        L"CREATE TABLE MyTable
        (
        \
        \
    )
}
```



```

        AcNum        INTEGER NOT NULL, \
        Name         CHAR(20),         \
        Balance      MONEY(8,2),       \
        PRIMARY KEY  (AcNum)           \
    );" );

    if (FAILED( hr = MyDb1.MyExecuteImmediateCommandText( wSQLcmd ) ) )
    {
        printf( "\nMyExecuteImmediateCommandText Failed" );
        return( hr );
    }

    swprintf( wSQLcmd,
        L"INSERT INTO MyTable  VALUES ( 100, \'John\', 150.75 );" );

    if (FAILED( hr = MyDb1.MyExecuteImmediateCommandText( wSQLcmd ) ) )
    {
        printf( "\nMyExecuteImmediateCommandText Failed" );
        return( hr );
    }

    swprintf( wSQLcmd,
        L"INSERT INTO MyTable  VALUES ( 101, \'Tom\', 225.75 );" );

    if (FAILED( hr = MyDb1.MyExecuteImmediateCommandText( wSQLcmd ) ) )
    {
        printf( "\nMyExecuteImmediateCommandText Failed" );
        return( hr );
    }

    tmp = MyDb1.MyDeleteCmd();
    tmp = MyDb1.MyDeleteSession();
    tmp = MyDb1.MyCloseDataSource();

    CoUninitialize();
    return(0);
}

```

6.1.7 IBM Informix Object Interface for C++ - CSDK

The IBM Informix Object Interface for C++ encapsulates Informix database server features into a class hierarchy.

Operation classes provide access to Informix databases and methods for issuing queries and retrieving results. Operation classes encapsulate database objects such as connections, cursors, and queries. Operation class methods encapsulate tasks such as opening and closing connections, checking and handling errors, executing queries, defining and scrolling cursors through result sets, and reading and writing large objects.

Value interfaces are abstract classes that provide specific application interaction behaviors for objects that represent IBM Informix Dynamic Server database values (value objects). Extensible value objects let you interact with your data.

Built-in value objects support ANSI SQL and C++ base types and complex types such as rows and collections. You can create C++ objects that support complex and opaque data types. Example 6-6 shows a simple object interface.

Example 6-6 A simple Informix Object Interface for C++ application

```
int
main(int, char *)
{
    // Make a connection using defaults
    ITConnection conn;
    conn.Open();

    // Create a query object
    ITQuery query(conn);

    string qtext;

    cout << "> ";

    // Read rows from standard input
    while (getline(cin, qtext))
    {
        if (!query.ExecForIteration(qtext.c_str()))
        {
            cout << "Could not execute query: " << qtext << endl;
        }

        else
        {

```

```

        ITRow *comp;
        int rowcount = 0;
        while ((comp = query.NextRow()) != NULL)
        {
            rowcount++;
            cout << comp->Printable() << endl;

            comp->Release();
        }
        cout << rowcount << " rows received, Command:"
            << query.Command() << endl;
    }
    if (query.Error())
        cout << "Error: " << query.ErrorText() << endl;
        cout << "> ";
    }
    conn.Close();

    cout << endl;
    return 0;
}

```

6.1.8 IBM Informix 4GL

Informix Fourth Generation Language (4GL) has been for a very long time a very successful database (Informix) centric business application development language. Initially developed in the mid-1980s, it became very popular in the late 1980s until the mid 1990s.

The broad success of Informix 4GL among the Informix ISV community is heavily based on its very high productivity for developing customized, character based and SQL focused applications. 4GL supports concepts for creating and maintaining menus, screens and windows in addition to display input/output forms and to be able to generate flexible reports.

As of writing this redbook, the classic 4GL is still being maintained (no new major features) on several current OS platforms and has reached a version number 7.32.

IBM Informix 4GL programs are written with a program editor, that is, a textual line editor. These ASCII text files are written and then compiled. 4GL comes in two dialects:

- ▶ A precompiler version which essentially generates ESQL/C code in a first phase, and in the second phase generates C code that later on compiles into OS-dependant application binaries.
- ▶ An interpretative version, called 4GL RDS (Rapid Development System), generates OS independent pseudo code which requires a special 4GL RDS runtime execution engine. In addition to the core language one can also debug 4GL RDS based application with the optional Interactive Debugger (ID).

From a 4GL language perspective, both versions should behave the same.

Applications written in Informix 4GL can also be easily extended by external C routines to enhance the functional capabilities through the addition of new functions to the 4GL based solution. This feature is supported in both 4GL dialects. Example 6-7 shows a simple 4GL application.

Typical Informix 4GL applications consist out of three file types:

- ▶ *.4gl* files contain the actual Informix 4GL business logic and the driving code for the user interface components (as examples, menus, windows, and screens).
- ▶ *.per* files are the source code versions of 4GL forms and do not include any procedural logic. They are basically a visual representation of how the forms should appear.
- ▶ *message* files can be used to create customized error- and application-messages to, for examples, accommodate different language settings for deployment.

Example 6-7 A simple Informix 4GL program

```
DATABASE stores
```

```
GLOBALS
```

```
    DEFINE p_customer RECORD LIKE customer.*  
END GLOBALS
```

```
MAIN
```

```
    OPEN FORM cust_form FROM "customer"  
    DISPLAY FORM cust_form  
    CALL get_customer()  
    MESSAGE "End program."  
    SLEEP 3
```

```

CLEAR SCREEN
END MAIN

FUNCTION get_customer()
  DEFINE s1, query_1 CHAR(300),
         existSMALLINT,
         answer CHAR(1)

  MESSAGE "Enter search criteria for one or more customers."
  SLEEP 3
  MESSAGE ""
  CONSTRUCT BY NAME query_1 ON customer.*
  LET s1 = "SELECT * FROM customer WHERE ", query_1 CLIPPED
  PREPARE s_1 FROM s1
  DECLARE q_curs CURSOR FOR s_1
  LET exist = 0
  FOREACH q_curs INTO p_customer.*
    LET exist = 1
    DISPLAY p_customer.* TO customer.*
    PROMPT "Do you want to see the next customer (y/n) ? "
    FOR answer
      IF answer = "n" THEN
        EXIT FOREACH
      END IF
    END FOREACH
  IF exist = 0 THEN
    MESSAGE "No rows found."
  ELSE
    IF answer = "y" THEN
      MESSAGE "No more rows satisfy the search criteria."
    END IF
  END IF
  SLEEP 3
END FUNCTION

```

Example 6-8 shows the associated .per 4GL form description file for Example 6-7.

Example 6-8 The associated .per 4GL form description file for Example 6-7

DATABASE stores

```

SCREEN
{

```

CUSTOMER FORM

```

      Number:  [f000      ]

      First Name: [f001      ]      Last Name: [f002      ]

      Company:  [f003      ]

      Address:  [f004      ]
                [f005      ]

      City:    [f006      ]

      State:   [a0]      Zipcode: [f007 ]

      Telephone: [f008      ]
-----
}
END

TABLES
customer

ATTRIBUTES
f000 = customer.customer_num;
f001 = customer.fname;
f002 = customer.lname;
f003 = customer.company;
f004 = customer.address1;
f005 = customer.address2;
f006 = customer.city;
a0 = customer.state;
f007 = customer.zipcode;
f008 = customer.phone;
END

INSTRUCTIONS
SCREEN RECORD sc_cust (customer.fname THRU customer.phone)
END

```

Because the classic Informix 4GL will not receive any further major enhancements, Informix 4GL developers might want to consider a move to the IBM Enterprise Generation Language, which incorporates many of the powerful features of the 4GL language.

6.1.9 IBM Enterprise Generation Language

IBM Enterprise Generation Language (EGL) is a procedural language used for the development of business application programs. The IBM EGL compiler outputs Java/J2SE or Java/J2EE™ code, as needed. With IBM EGL, one can develop business application programs with no user interface, a text user interface, or a multi-tier graphical Web interface.

Additionally, IBM EGL delivers the software re-use, ease of maintenance, and other features normally associated with object oriented programming languages by following the Model-View-Controller (MVC) design pattern. Because IBM EGL outputs Java J2EE code, IBM EGL benefits from and follows the design pattern of MVC and Java/J2EE. EGL will cause the programmer to organize elements of a business application into highly structured, highly reusable, easily maintained, and highly performant program components.

IBM EGL is procedural, but it is also fourth generation. While IBM EGL supports all of the detailed programming capabilities one needs to execute in order to support business (procedural), it also has the higher level constructs that offer higher programmer productivity (fourth generation).

In another sense, IBM EGL is also declarative. There are lists of properties that can be applied to various EGL components, and these properties greatly enhance or configure the capability of these objects. Lastly, the term enterprise, as in Enterprise Generation Language, connotes that EGL can satisfy programming requirements across the entire enterprise. For example, with EGL, one can deliver intranet, extranet, and Internet applications, including Web services.

EGL provides a simplified approach to application development that is based on these simple principles:

- ▶ **Simplifying the specification:** EGL provides an easy to learn programming paradigm that is abstracted to a level that is independent from the underlying technology. Developers are shielded from the complexities of a variety of supported runtime environments. This results in a reduced training costs and a significant improvement in productivity.
- ▶ **Code generation:** High productivity comes from the ability to generate the technology neutral specifications (EGL) or logic into optimized code for the target runtime platform. This results in less code that is written by the business oriented developer and potentially a reduced number of bugs in the application.
- ▶ **EGL Based Debugging:** Source level debugging is provided in the technology neutral specification (EGL) without having to generate the target platform code. This provides complete, end-to-end isolation from the complexity of the underlying technology platform.

Database connectivity with EGL

Accessing data from databases can sometimes be challenging to developers whose primary objective is to provide their users with the information that is optimal for them to make business decisions. To be able to access data, a developer needs to:

- ▶ Connect to a database.
- ▶ Know and use the database schema.
- ▶ Be proficient in SQL in order to get the appropriate data.
- ▶ Provide the primitive functions to perform the basic CRUD (Create, Read, Update, and Delete) database tasks.
- ▶ Provide a test environment to efficiently test your application.

Figure 6-3 on page 206 depicts an example of a simple EGL program that connects to and accesses data from IDS.

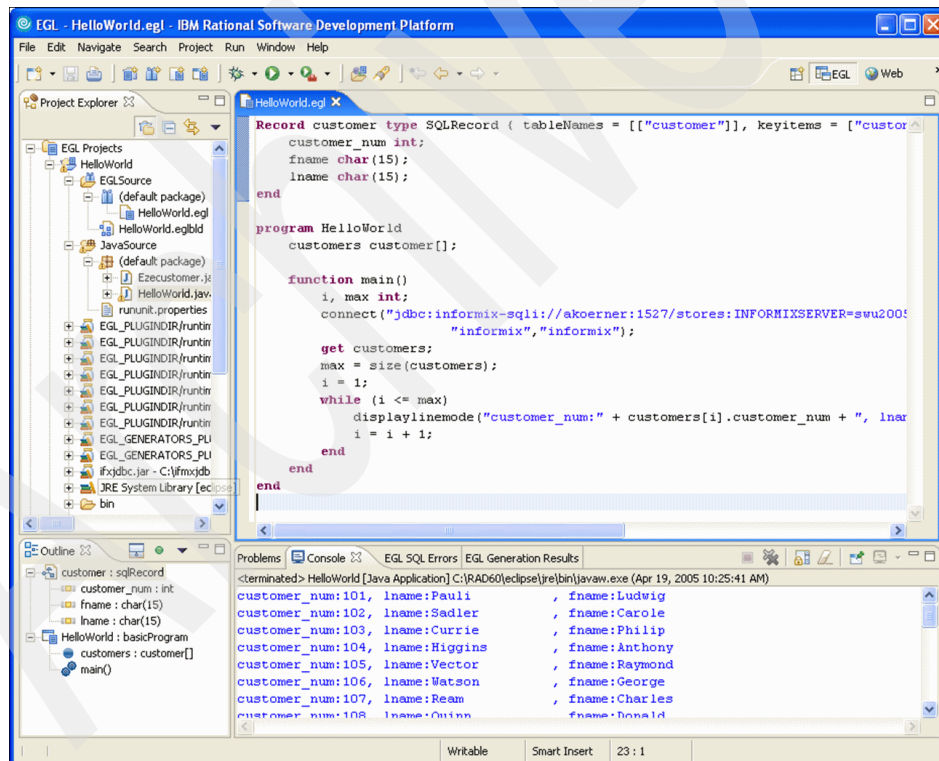


Figure 6-3 A simple EGL program accessing IDS V10

EGL provides capabilities that make this task very easy for that business oriented developer:

- ▶ **Connectivity:** Wizards take developers through a step-by-step process of defining connectivity.
- ▶ **Database Schema:** If you are using an existing database, EGL provides an easy-to-use import capability that makes the schema structure available to your application.
- ▶ **SQL Coding:** EGL provides the generation of SQL statements based on your EGL code. You then have the option to use the SQL that was generated or for those power SQL users. You can alter the generated SQL to suit your needs.
- ▶ **Primitive functions:** The EGL generation engine generates the typical CRUD functions that are the workhorse functions for database driven applications automatically.
- ▶ **Test capabilities:** The IBM Rational development tools have a test environment that eliminates the complexities that are associated with deploying and running your application in complex target platforms.

6.1.10 IBM Informix Embedded SQL for Cobol (ESQL/Cobol)

IBM Informix ESQL/Cobol is an SQL application programming interface (SQL API) that lets you embed SQL statements directly into Cobol code.

It consists of a code preprocessor, data type definitions, and Cobol routines that you can call. And it can use both static and dynamic SQL statements. When static SQL statements are used, the program knows all the components at compile time.

ESQL/Cobol is currently only available on AIX, HP/UX, Linux and Solaris. Example 6-9 shows an ESQL/Cobol snippet.

Example 6-9 An ESQL/Cobol example snippet

```
13 IDENTIFICATION DIVISION.  
14 PROGRAM-ID.  
15 DEMO1.  
16 *  
17 ENVIRONMENT DIVISION.  
18 CONFIGURATION SECTION.  
19 SOURCE-COMPUTER. IFXSUN.  
20 OBJECT-COMPUTER. IFXSUN.  
21 *  
22 DATA DIVISION.  
23 WORKING-STORAGE SECTION.
```

```

24 *
25 *Declare variables.
26 *
27 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
28 77 FNAME PIC X(15).
29 77 LNAME PIC X(20).
30 77 EX-COUNT PIC S9(9) COMP-5.
31 77 COUNTER PIC S9(9) VALUE 1 COMP-5.
32 77 MESS-TEXT PIC X(254).
33 EXEC SQL END DECLARE SECTION END-EXEC.
34 01 WHERE-ERROR PIC X(72).
35 *
36 PROCEDURE DIVISION.
37 RESIDENT SECTION 1.
38 *
39 *Begin Main routine. Open a database, declare a cursor,
40 *open the cursor, fetch the cursor, and close the cursor.
41 *
42 MAIN.
43 DISPLAY ' '.
44 DISPLAY ' '.
45 DISPLAY 'DEM01 SAMPLE ESQL PROGRAM RUNNING.'.
46 DISPLAY ' TEST SIMPLE DECLARE/OPEN/FETCH/LOOP'.
47 DISPLAY ' '.
48
49 PERFORM OPEN-DATABASE.
50
51 PERFORM DECLARE-CURSOR.
52 PERFORM OPEN-CURSOR.
53 PERFORM FETCH-CURSOR
54 UNTIL SQLSTATE IS EQUAL TO "02000".
55 PERFORM CLOSE-CURSOR.
56 EXEC SQL DISCONNECT CURRENT END-EXEC.
57 DISPLAY 'PROGRAM OVER'.
58 STOP RUN.
59 *
60 *Subroutine to open a database.
61 *
62 OPEN-DATABASE.
63 EXEC SQL CONNECT TO 'stores7' END-EXEC.
64 IF SQLSTATE NOT EQUAL TO "00000"
65 MOVE 'EXCEPTION ON DATABASE STORES7' TO WHERE-ERROR
66 PERFORM ERROR-PROCESS.
67 *
68 *Subroutine to declare a cursor.

```

```
69 *
70 DECLARE-CURSOR.
71 EXEC SQL DECLARE DEMOCURSOR CURSOR FOR
72 SELECT FNAME, LNAME
73 INTO :FNAME, :LNAME
74 FROM CUSTOMER
75 WHERE LNAME > 'C'
76 END-EXEC.
77 IF SQLSTATE NOT EQUAL TO "00000"
78 MOVE 'ERROR ON DECLARE CURSOR' TO WHERE-ERROR
79 PERFORM ERROR-PROCESS.
```

6.2 Additional tools and APIs for IDS V10

In addition to the already broad support of IBM supported development tools and APIs for IDS V10, there are plenty of additional options through either the Open Source community or third-party vendors. This section documents a selection of those offerings.¹

6.2.1 IDS V10 and PHP support

Hypertext Preprocessor (PHP) is a powerful server-side scripting language for Web servers. PHP is popular for its ability to process database information and create dynamic Web pages. Server-side refers to the fact that PHP language statements, which are included directly in your Hypertext Markup Language (HTML), are processed by the Web server.

Scripting language means that PHP is not compiled. Because the results of processing PHP language statements is standard HTML, PHP-generated Web pages are quick to display and are compatible with most all Web browsers and platforms. In order to run PHP scripts with your HTTP server, a PHP engine is required. The PHP engine is an open source product or is quite often already included in the HTTP server.

¹ Some material in this section is copyright (c) Jonathan Leffler 1998, 2006, and has been used with permission.

IBM Informix IDS supported drivers for PHP

There are currently three different options for how IDS can be integrated with a PHP environment:

- Unified ODBC (ext/odbc)

The unified ODBC driver has been built into the PHP core and is normally compiled against a generic ODBC driver manager. It can be also compiled against the specific IDS ODBC libraries (you need to run `./configure with-custom-odbc`) for access. Although this driver works with both PHP 4 and PHP 5, it has some drawbacks.

As examples, it always requests scrollable cursors which could lead to some slow query processing, and it might be warning-prone. In addition there is no support for OUT/INOUT stored procedures in IDS V10.

- PHP driver for IDS (Extensions for IDS)

This IDS driver (Informix Extensions) is available from the PHP.NET repository and is also part of the Zend Optimizer core. You can download it from:

<http://us3.php.net/ifx>

It is developed and supported through the PHP community.

The current version of the Informix Extensions has full featured support for IDS 7 but only partial support for IDS versions greater than 9 (including IDS V10). Because it is based on ESQL/C (see also the 6.1.2, “Embedded SQL for C (ESQL/C) - CSDK” on page 188), it provides very performant access to the underlying IDS database. Like the unified ODBC driver, it also works with PHP 4 and PHP 5. Example 6-10 includes a PHP/Informix Extensions code example.

Example 6-10 A PHP code example which uses the Informix PHP extensions

```
<HTML>
<TITLE>Simple Connection to IDS </TITLE>
<HEAD>
<TITLE> PHP Information </TITLE></HEAD>
<BODY>
<?php
$conn_id=ifx_connect("stores_demo","informix","xxx");

$result = ifx_prepare("select fname,lname from customer",$conn_id);
if(!ifx_do($result))
{
    printf("<br>Could not execute the query<br>");
    die();
}
```

```

else
{
    $count = 0;
    $row = ifx_fetch_row($result,"NEXT");
    while(is_array($row))
    {
        for(reset($row); $fieldname=key($row); next($row)) {
            $fieldvalue = $row[$fieldname];
            printf("<br>%s = %s</br>", $fieldname, $fieldvalue);
        }
        $row = ifx_fetch_row($result,"NEXT");
    }
}

?>
</BODY>
</HTML>

```

► PHP Data Objects: PDO_IDS, PDO_ODBC

PDO is a fast, light and pure-C standardized data access interface for PHP 5. It is already integrated in PHP 5.1 or is available as PECL (PHP extension community library) extension to PHP 5.0. Because PDO requires the new OO (Object Oriented) features in PHP 5, it is not available to earlier PHP versions, such as PHP 4. For IDS V10, you can either use the PDO_ODBC or the new PDO_INFORMIX driver.

There are several advantages of using the new PDO_INFORMIX driver that you can download from:

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

Its a native driver, so it provides high performance. It also has been stress tested heavily, which makes it a very good candidate for production environments. Example 6-11 shows a simple PDO example.

Example 6-11 A simple PDO_INFORMIX example

```

<?php
try
{
    $db = new PDO("informix:host=akoerner; service=1527;
    database=stores_demo; server=ol_itso2006; protocol=onsoctcp;
    EnableScrollableCursors=1;", "informix", "informix");
}
$stmt = $db->prepare("select * from customer");
$stmt->execute();

```

```
while($row = $stmt->fetch( ))
{
    $cnum=$row['customer_num'];
    $lname=$row['lname'];
    printf("%d %s\n",$cnum,$lname);
}
$stmt = null;
?>
```

Zend Core for IBM

Zend Core for IBM is the first and only certified PHP development and production environment that includes tight integration with Informix Dynamic Server (IDS) and our greater family of data servers. Certified by both Zend and IBM, Zend Core for IBM delivers a rapid development and production PHP foundation for applications using PHP with IBM data servers

Additional information

For more information about how to develop IDS based PHP applications, refer to *Developing PHP Applications for IBM Data Servers*, SG24-7218.

6.2.2 PERL and DBD::Informix

Perl is a very popular scripting language, which was originally written by Larry Wall. Perl version 1.0 was released in 1987. You can find the current version of Perl is 5.8.8 and all Perl related source code and binaries on the Comprehensive Perl Archive Network (CPAN) at:

<http://www.cpan.org>

The Perl Database Interface (DBI) was designed by Tim Bunce to be the standard database interface for the Perl language.

There are many database drivers for DBI available, including ODBC, DB2, and of course, Informix IDS (plus some others). The current version of DBI as of writing this Redbook is 1.52 and it requires Perl 5.6.1 or later.

DBD::Informix

The DBD::Informix driver for Perl has been around for quite a while. The original versions (up to version 0.24) were written by Alligator Descartes in 1996. Jonathan Leffler (then working for Informix, now working for IBM) took over the guardianship and development of DBD::Informix, creating numerous releases between 1996 (version 0.25) and 2000 (version 0.97).

The current version number is 2005.02. You can download it from CPAN at:

<http://www.cpan.org>

Support for the current DBD::Informix driver can be obtained through the following channels:

- ▶ dbd.informix@gmail.com
- ▶ dbi-users@perl.org

To build the DBD::Informix driver for Perl, you would need the following components:

- ▶ IBM Informix ESQL/C 5.00 or later (ClientSDK)
- ▶ An ANSI C compiler (code uses prototypes)
- ▶ A small test database with DBA privileges
- ▶ Perl version 5.6.0 or later (5.8.8 strongly recommended)
- ▶ DBI version 1.38 or later (1.50 or later strongly recommended)

Example 6-12 shows the use of the DBD::Informix driver in a perl application.

Example 6-12 How to use DBD::Informix in a simple perl application

```
#!/usr/bin/perl -w
use DBI;
$dbh = DBI->connect('DBI:Informix:stores7','', '', {RaiseError => 1,
PrintError=>1});
$stmt = $dbh->prepare(q%SELECT Fname, Lname, Phone
FROM Customer WHERE Customer_num = ? %);
$stmt->execute(106);
$ref = $stmt->fetchall_arrayref();
for $row (@$ref) {
    print "Name: $$row[0] $$row[1], Phone: $$row[2]\n";
}
$dbh->disconnect;
```

The current DBD::Informix driver has a few known limitations:

- ▶ Not fully aware of 9.x collection types or UDTs
- ▶ No support for bind_param_inout method
- ▶ No handling for new DBI features since v1.14

6.2.3 Tcl/Tk and the Informix (isqltcl) extension

Tcl stands for Tool Command Language. Tk is the Graphical Toolkit extension of Tcl, providing a variety of standard GUI interface items to facilitate rapid, high-level application development. Tcl was designed with the specific goals of

extensibility, a shallow learning curve, and ease of embedding. Tk development began in 1989, and the first version was available in 1991. The current version of Tcl/Tk as of writing this redbook is 8.4.13. You can download Tcl/Tk from the following Web site:

<http://www.tcl.tk/>

Tcl/Tk is an interpreted environment. The Tcl interpreter can be extended by adding pre-compiled C functions, which can be called from within the Tcl environment. These extensions can be custom for a specific purpose or generic, and are widely useful.

To access IDS V10 from within a Tcl/Tk application you need to obtain a copy of the *isqltcl* extension for Tcl/Tk from:

<http://isqltcl.sourceforge.net/>

The current version is version 5, released February 2002. Before you can use the *isqltcl* extension you have to compile it into a shared library for your target platform. A Windows DLL seems to be available for download from the *isqltcl* Web site.

To use the *isqltcl* extension you need to preload it with the **load isql.so** or **load isql.dll** command within the Tcl/Tk shells *tclsh* or *wish* first. After that execute the *isqltcl* commands to access IDS V10:

Connect to a given database:

```
sql connect dbase as conn1 user $username password $password
```

Close the database connection:

```
sql disconnect [current|default|all|conn1]
```

Sets the specified connection:

```
sql setconnection [default|conn1]
```

Executable statements:

- ▶ Prepare and executes the statement
- ▶ Optionally takes a number of arguments for placeholders
- ▶ Returns zero on success; non-zero on failure
- ▶ Statements that return no data

As an example:

```
sql run {delete from sometable where pkcol = ?} $pkval
```


Cursor handling in isqltcl (for SELECTs or EXECUTE PROCEDURE):

```
set stmt [sql open {select * from sometable}]
```

This statement does a PREPARE, DECLARE, OPEN and returns a statement number (id) or a negative error and optionally takes arguments for placeholders.

```
set row [sql fetch $stmt 1]
```

This command collects one row of data and creates it as a Tcl list in the variable 'row'. The 1 is optional and means strip trailing blanks and the list is empty if there is no more data.

```
sql reopen $stmt ?arg1? ?arg2?
```

Reopens the statement, with new parameters.

```
sql close $stmt
```

Indicates that you have no further use for the statement and it frees both the cursor and statement!

6.2.4 Python, Informix DB-2.2 and IDS V10

Python is an increasingly popular, general-purpose, object-oriented programming language. Python is free and Open Source, and runs on a variety of platforms, including Linux, UNIX, Windows, and Macintosh. Python has a clean, elegant syntax that many developers find to be a tremendous time-saver. Python also has powerful built-in object types that allow you to express complex ideas in a few lines of easily maintainable code. In addition, Python comes with a standard library of modules that provide extensive functionality and support for such tasks as file handling, network protocols, threads and processes, XML processing, encryption, object serialization, and E-mail and news group message processing.

The DB-API is a standard for Python modules that provide an interface to a DBMS. This standard ensures that Python programs can use a similar syntax to connect to, and access data from, any supported DBMS including IDS V10.

The most current Informix IDS implementation of the Python DB-API 2.0 specifications is called InformixDB-2.2, was developed by Carsten Haese and released in March 2006. You can find more detailed information and downloads related to the Informix Python module at:

<http://informixdb.sourceforge.net/>

Example 6-13 shows a simple Python code example that connects to an IDS database server.

In order to build the InformixDB-2.2 module you need to have a current version of ESQL/C installed on your development machine installed. It also requires a Python version of 2.2 or higher (the current version of Python as of this writing is 2.4.3). You can find Python itself at:

<http://www.python.org/>

Example 6-13 A simple Python application to access IDS V10

```
import informixdb
conn = informixdb.connect("test", "informix", "pw")
cur = conn.cursor()
cur.execute("create table test1(a int, b int)")
for i in range(1,25):
    cur.execute("insert into test1 values(?,?)", (i, i**2))
cur.execute("select * from test1")
for row in cur:
    print "The square of %d is %d." % (row[0], row[1])
```

6.2.5 IDS V10 and the Hibernate Java framework

Hibernate (see also <http://www.hibernate.org>) is a very popular object-relational, Java based persistence and query service. It supports the use of either the Informix IDS specific SQL language or a portable Hibernate SQL extension, called HQL. Informix IDS is one of the Hibernate community supported databases.

In order to use IDS V10 in combination with Hibernate be sure to select the Informix dialect through a property setting in the hibernate.properties file. For IDS, you set the hibernate.dialect property to `org.hibernate.dialect.InformixDialect`.

Hibernate based applications can be either executed stand-alone or in an Java J2EE based application server environment such as IBM WebSphere Application Server.

Tip: During runtime a Hibernate based application typically re-uses only a few SQL statements. To reduce unnecessary network traffic between the Hibernate application and the IDS instance, and to reduce unwanted statement parsing overhead in IDS, consider the use of a connection pool which supports prepared statement caching. One example of such a caching connection pool is the Open Source C3P0 Connection Pool which comes bundled with the Hibernate source code.

When the Hibernate based application is deployed on a Java application server, consider using the connection pool caching which is integrated into the target application server (for example, IBM WebSphere). The JDBC 3.0 standard actually defines a prepared statement cache for connection pooling. At the writing of this redbook, the current IBM Informix JDBC 3.0 driver does not support that feature.

Data encryption

In this chapter, we discuss how to encrypt data in a database. Informix Dynamic Server (IDS) V10 introduces a set of functions to encrypt and decrypt data as it is moved to or from the database. This is a powerful tool, but you do need to understand the costs that are associate with using this tool as well as the mechanics of the tool itself.

This chapter is not intended to provide a comprehensive treatment of data security or system security. It is only a discussion of one feature of the IDS. You should think about the use of data encryption as one part of a larger system and data security program. This feature alone is not usually sufficient.

7.1 Scope of encryption

Data encryption applies separately to each column of each table. The work is done in functions within the SQL statement. An example is shown in Example 7-2 on page 222.

Only character or BLOB data can be encrypted with the current built-in functions. Columns of other types cannot be encrypted unless user-defined functions are provided for that purpose.

You can choose to encrypt some or all of the columns in any row, and you can choose to encrypt some or all of the rows in any column. These functions apply cell-by-cell. So, for example, you might choose to encrypt all the employee addresses (that column in every row) but only the salaries above USD100,000 (that column in only some of the rows).

There is no method for encrypting all of the applicable data in a table or a whole database with just a single command. You have to use views or modify individual SQL statements to encrypt or decrypt the data.

7.2 Data encryption and decryption

The basic technique for data encryption and decryption is straightforward. You use your choice of encryption function to insert or modify data. Example 7-1 illustrates the process. In this example, we alter the cust_calls table of the stores database to add a column. We then populate that column with the encrypted form of the user_id column. After that, we select the plain text, encrypted and decrypted forms of the data, to show how they look.

In this example, we use the SET ENCRYPTION PASSWORD statement to simplify the syntax. This syntax also ensures that the same password is used in every function with no possibility of typographical error.

We can also provide a hint in the SET ENCRYPTION PASSWORD statement, but that hint is optional.

This example uses the ENCRYPT_AES() function. There is another choice, the ENCRYPT_TDES() function. Both have the same arguments, but use different encryption algorithms. Either function can be used for any column.

Example 7-1 Basic data encryption and decryption

```
C:\idsdemos\stores9>dbaccess -e stores9 example1
```

```
Database selected.
```

```
alter table cust_calls add enc_id char(200);
```

```
Table altered.
```

```
set encryption password "erewhon" with hint "gulliver";
```

```
Encryption password set.
```

```
update cust_calls set enc_id = encrypt_aes(user_id);
```

```
7 row(s) updated.
```

```
select  first 2
        user_id as plain_user_id
        , enc_id as enc_id
        , decrypt_char(enc_id) as decrypted_user_id
from    cust_calls
;
```

```
plain_user_id      richc
enc_id
v0AQAAAAEAW4Xcnj8bH292ATuXI6GIiGA8yJJviU9JCUq9ngvJkEP+/BzwFhGSYw==
decrypted_user_id  richc
```

```
plain_user_id      maryj
enc_id
0gH8QAAAAEASv+pWbie/aTjzqckgdHCGhogtLyGH8vGCUq9ngvJkEP+/BzwFhGSYw==
decrypted_user_id  maryj
```

```
2 row(s) retrieved.
```

```
Database closed.
```

The next example is more complicated. We use the `call_type` table in the `stores` database to encrypt multiple columns in a table, as well as a more complex scheme of encryption functions and passwords.

Example 7-2 on page 222 shows this more complex processing. Two columns are added to the table, and the two existing columns are each encrypted. That encryption uses the two encryption algorithms for various rows, and some of the rows have their own unique passwords. The remainder of the rows use the common password from a `SET ENCRYPTION PASSWORD` statement. This

example demonstrates that the password in a function call takes precedence over the previously set password.

Important: *Be very careful if you choose to use this technique. To retrieve the data, you must specify the password that was used to encrypt the row or rows you want. If you use different passwords for the rows of a table, then you will not be able to get them all back with a single SELECT statement because there is no way to include multiple passwords in a single function call. This is demonstrated in the first two SELECT statements in the example. To get multiple rows, you would have to use a UNION to get each row of the result set.*

Example 7-2 Complex encryption and decryption

```
C:\idsdemos\stores9>dbaccess -e stores9 example2
```

Database selected.

```
ALTER TABLE call_type ADD enc_code char(99);
Table altered.
```

```
ALTER TABLE call_type ADD enc_descr char(150);
Table altered.
```

```
SET ENCRYPTION PASSWORD 'erewhon' WITH HINT 'gulliver';
Encryption password set.
```

```
UPDATE call_type SET enc_code =
CASE
    WHEN call_code = 'B' then encrypt_aes('B', "Michael")
    WHEN call_code = 'D' then encrypt_aes('D', "Terrell")
    WHEN call_code = 'I' then encrypt_tdes('I', "Norfolk")
    WHEN call_code = 'L' then encrypt_aes('L')
    WHEN call_code = 'O' then encrypt_aes('O')
END;
6 row(s) updated.
```

```
SET ENCRYPTION PASSWORD 'whosonfirst' WITH HINT 'abbotcostello';
Encryption password set.
```

```
UPDATE call_type SET enc_descr = ENCRYPT_TDES(code_descr);
6 row(s) updated.
```

```
SELECT DECRYPT_CHAR(enc_code, "Michael") AS decrypted_code
FROM   call_type
```



```

WHERE    call_code = 'B'
;

decrypted_code  B

1 row(s) retrieved.

SELECT  DECRYPT_CHAR(enc_code) AS decrypted_code
FROM    call_type
WHERE   call_code = 'B'
;

26008: The internal decryption function failed.
Error in line 27
Near character position 20

SELECT  DECRYPT_CHAR(enc_code, "erewhon") AS decrypted_code
        , DECRYPT_CHAR(enc_descr) AS decrypted_description
FROM    call_type
WHERE   call_code = 'L'
;

decrypted_code      L
decrypted_descrip+  late shipment

1 row(s) retrieved.

```

7.3 Retrieving encrypted data

As previous examples have shown, encrypted data is retrieved using the `decrypt_char()` function. This function knows from the encrypted string how to do the decryption for either encryption function. The returned value has the same size as the original unencrypted data. You do not need to have variables as large as the encrypted data.

BLOB data is encrypted in the same way as character data. However, retrieving the BLOB data is done using the `DECRYPT_BINARY()` function.

Example 7-3 demonstrates the `DECRYPT_BINARY()` function. We alter the catalog table in the stores database.

Example 7-3 Decrypting binary data

```
C:\idsdemos\stores9>dbaccess -e stores9 example9
```

Database selected.

```
ALTER TABLE catalog ADD adstuff BLOB;  
Table altered.
```

```
ALTER TABLE catalog ADD enc_stuff BLOB;  
Table altered.  
UPDATE catalog  
SET adstuff = FILETOBLOB ('c:\informix\gif','server')  
WHERE catalog_num =10031  
;  
1 row(s) updated.
```

```
SET ENCRYPTION PASSWORD "Erasmus";  
Encryption password set.
```

```
UPDATE catalog  
SET enc_stuff = ENCRYPT_AES (adstuff)  
WHERE catalog_num =10031  
;  
1 row(s) updated.
```

```
SELECT  catalog_num  
        , LOTOFILE(DECRYPT_BINARY(enc_stuff),  
        'c:\informix\gif2','server')  
FROM catalog  
WHERE catalog_num = 10031  
;  
  
catalog_num    10031  
(expression)  c:\informix\gif2.0000000044be7076  
  
1 row(s) retrieved.
```

Database closed.

Each use of the encryption functions results in a different encrypted string. You do not get the same result by encrypting the same string more than once, which is a good thing for security. However, that means *one encrypted value cannot be*

correctly compared to another. Before you compare values, decrypt both of them to get accurate comparisons as demonstrated in Example 7-4. You must decrypt a value used in a comparison before doing the comparison.

Example 7-4 Incorrect and correct comparison of encrypted data

```
C:\idsdemos\stores9>dbaccess -e stores9 example5
```

Database selected.

```
SELECT  DECRYPT_CHAR(enc_descr, 'whosonfirst') AS description
FROM    call_type
WHERE   enc_code = ENCRYPT_AES('B', 'Michael')
;
```

No rows found.

```
SELECT  DECRYPT_CHAR(enc_descr, 'whosonfirst') AS description
FROM    call_type
WHERE   DECRYPT_CHAR(enc_code, "Michael") = 'B'
;
```

description billing error

Database closed.

7.4 Indexing encrypted data

There is no effective way to index an encrypted column. If the encrypted values are used in the index, then there is no way to correctly compare those keys to values in SQL statements. The unencrypted values cannot be put in the index because the encryption occurs before the data is stored.

Example 7-5 on page 226 shows another aspect of the problem. For the example, we added a column to the stock table and use it to hold the encrypted value of the status column. The status column is either null or has the value A. There are only two distinct values. Notice that after encryption, there are a number of distinct values equal to the number of rows in which the status column is not null. That happens because each encryption results in a different value, even for the same input value. This makes the index useless for finding all the rows with some specific value in the indexed column.

If indexes are required on encrypted data, then the unencrypted data must also be stored in the database. That usually defeats the purpose of encrypting the data.

However, indexes can be created on encrypted columns. There are no warnings or errors, and the index is created correctly.

Example 7-5 Encrypted indexes

```
kodiak:/home/informix $ dbaccess stores9 -
```

Database selected.

```
> info columns for stock;
```

Column name	Type	Nulls
stock_num	smallint	yes
manu_code	char(3)	yes
...		
status	char(1)	yes
...		
e_manucode	char(36)	yes
e_unit	char(35)	yes
e_desc	char(43)	yes
e_manuitem	char(43)	yes
e_status	char(35)	yes
e_bigger	char(35)	yes

```
>
```

```
> select count(distinct status) from stock;
```

```
(count)
1
```

1 row(s) retrieved.

```
> select count(distinct e_status) from stock;
```

```
(count)
44
```

1 row(s) retrieved.

```
> select count(*) from stock where status is not null;
```

```
(count(*))
```

1 row(s) retrieved.

```
> create index estat on stock(e_status);
```

Index created.

7.5 Hiding encryption with views

Using encryption in an existing database, poses the question of whether to change existing SQL statements to include the encryption and decryption functions. That can be both time-consuming and error-prone.

An alternative is to define views to hide the encryption functions. Example 7-6 shows the technique. In the example, we rework the cust_calls table so that the user_id is encrypted, but the existing SQL statements using that column need not be changed. To accomplish that goal, we use the modified cust_calls table created in Example 7-1 on page 221. We rename the table, create a view with the former table name, and use the DECRYPT_CHAR function to have the decrypted user_id appear in the proper place.

This technique does require a careful use of the encryption password. You still need to assure you have the correct password in place for the table. However, you are limited to the SET ENCRYPTION PASSWORD statement. You cannot use separate passwords for some rows in this method. If you wish to do that, you will need a stored procedure to retrieve the encrypted data so that you can pass the password as a parameter. Example 7-6 demonstrates this technique.

Example 7-6 Views to hide encryption functions

```
C:\idsdemos\stores9>dbaccess -e stores9 example7
```

Database selected.

```
RENAME TABLE cust_calls TO cust_calls_table;
Table renamed.
```

```
CREATE VIEW cust_calls ( customer_num
                        , call_dtime
                        , user_id
                        , call_code
                        , call_descr
```

```

, res_dtime
, res_descr )

```

AS

```

SELECT  customer_num
        , call_dtime
        , DECRYPT_CHAR (enc_id)
        , call_code
        , call_descr
        , res_dtime
        , res_descr
from    cust_calls_table
;

```

View created.

```

SET ENCRYPTION PASSWORD "erewhon";
Encryption password set.

```

```

select * from cust_calls;

```

```

customer_num  119
call_dtime    1998-07-01 15:00
user_id       richc
call_code     B
call_descr    Bill does not reflect credit from previous order
res_dtime     1998-07-02 08:21
res_descr     Spoke with Jane Akant in Finance. She found the error and
is send
              ing new bill to customer

```

```

customer_num  121
call_dtime    1998-07-10 14:05
user_id       maryj
call_code     0
call_descr    Customer likes our merchandise. Requests that we stock
more types
              of infant joggers. Will call back to place order.
res_dtime     1998-07-10 14:06
res_descr     Sent note to marketing group of interest in infant
joggers

```

.... (some rows deleted)

```

customer_num  110
call_dtime    1998-07-07 10:24
user_id       richc

```

```
call_code      L
call_descr    Order placed one month ago (6/7) not received.
res_dtime     1998-07-07 10:30
res_descr     Checked with shipping (Ed Smith). Order sent yesterday-
we were w     aiting for goods from ANZ. Next time will call with delay
if nece       ssary.

7 row(s) retrieved.

Database closed.
```

7.6 Managing passwords

In this section we discuss and describe several approaches for managing passwords.

7.6.1 General password usage

You might use a common password established with the SET ENCRYPTION PASSWORD statement, or you might specify the password separately for each encrypted column. If you specify the password separately for each column, you can use different passwords for each column or group of columns. If you have a SET ENCRYPTION PASSWORD in effect, a password in a SQL statement overrides the more global password. If you provide no password and have no SET ENCRYPTION PASSWORD in effect, you will get an error.

7.6.2 The number and form of passwords

Encryption passwords are a difficult subject, as passwords are for most people. One needs to balance the simplicity of a single password for all the data against the difficulty of remembering multiple passwords and when each of them applies.

First, consult your corporate security standards regarding the form (such as the number of characters and special characters) of passwords, and abide by those rules. IDS has only one requirement and that is a password must be at least six but not more than 128 characters. There are no rules for such things as starting or ending characters, required alphabetic, or numeric or special characters.

At some point, you need to choose some number of passwords. There are many schemes for choosing passwords. We discuss four options here. Study the choices, and then determine the scheme that meets your requirements.

One scheme that we think makes sense is to choose a password for each table in the database that holds encrypted data. That means you have at most a number of passwords equal to the number of tables. Because most databases have a significant number of tables with codes and meanings that are not encrypted, the actual number of tables with encrypted data is likely to be relatively small.

An alternative is to associate a password with the data used in each job role within the organization. If the jobs use disjoint sets of data, this will work nicely. If there is significant overlap in the data that is used, this might not work as well.

Another alternative to consider is to have a password for each encrypted column. That might be a rather large number, but it would be somewhat more secure.

If the database has natural groupings of tables, consider having a password per group of tables. For example, if you have both personnel records and financial accounting data in the same database, perhaps have just two passwords, one for each major category.

7.6.3 Password expiration

IDS does not provide or enforce any password expiration. If your standards require password changes at some interval, you will have to do some work to update your data to the new password. *Without the correct password, the data cannot be retrieved by any means.* So you have to develop a way to make the changes and schedule the updates to change the passwords before they expire.

The SQL to do this sort of password change can be tricky. Because there can be only one password set at any point using the SET ENCRYPTION PASSWORD statement, you can use that for only the old or new password. The other password must be included in each update statement.

Example 7-7 demonstrates the technique. In the example, we have not used the SET ENCRYPTION PASSWORD statement. Rather, we have included all the passwords in the statement. In addition, we have updated only one row.

Example 7-7 Encryption password updating

```
update call_type
  set enc_code =
        encrypt_aes(decrypt_char(enc_code, "Dennis"), "Erasm")
where call_code = 'X';
```

Example 7-7 is not a very practical example because it affects only a single row. A more common operation is to update the common password for all the rows in a table or group of tables. Example 7-8 demonstrates that operation. The example works on the sales_rep table of the stores database.

Example 7-8 Multi-row password updates

```
C:\idsdemos\stores9>dbaccess -e stores9 example10.sql
```

Database selected.

```
ALTER TABLE sales_rep ADD enc_name ROW(fname LVARCHAR, lname LVARCHAR);
Table altered.
```

```
SET ENCRYPTION PASSWORD "Ricardo" WITH HINT "Vancouver";
Encryption password set.
```

```
UPDATE sales_rep
SET enc_name = ROW(ENCRYPT_TDES(name.first), ENCRYPT_TDES(name.last));
2 row(s) updated.
```

```
SELECT * FROM sales_rep;
```

```
rep_num      101
name          ROW('Peter','Jones')
region_num    6
home_office   t
sales         SET{ROW('1997-03',' $47.22'),ROW('1997-04',' $55.22')}
commission    0.03000
enc_name
ROW('11mkQAAAACAxE6fSstEGWCEdN/qDRyR0zZ1tX4Yu0U9go5dXfYD/ZY=', '1C/
      QQAAAACA62TTuNf9q8csJ0hyVB8wQTZ1tX4Yu0U9go5dXfYD/ZY=')
```

```
rep_num      102
name          ROW('John','Franklin')
region_num    6
home_office   t
sales         SET{ROW('1997-03',' $53.22'),ROW('1997-04',' $18.22')}
commission    0.03000
enc_name
ROW('1qVYQAAAACAfYXfz0uyd2CJJkqhpkLVPjZ1tX4Yu0U9go5dXfYD/ZY=', '1hH
      sQAAAAEAz6TfNs9/MsWxEqEz1xMYtNwst7RWg8hzNmW1fhi45T2Cj1ld9gP9lg==')
```

2 row(s) retrieved.

```

SELECT DECRYPT_CHAR(enc_name.fname) || ' ' ||
DECRYPT_CHAR(enc_name.lname) as NAME
FROM   sales_rep

name  Peter Jones
name  John Franklin

2 row(s) retrieved.

SET ENCRYPTION PASSWORD "DenverLad" WITH HINT "Karl";
Encryption password set.

update sales_rep
set enc_name =
ROW(encrypt_tdes(decrypt_char(enc_name.fname, "Ricardo"))
    ,encrypt_tdes(decrypt_char(enc_name.lname, "Ricardo"))
    )
;
2 row(s) updated.

SELECT DECRYPT_CHAR(enc_name.fname) || ' ' ||
DECRYPT_CHAR(enc_name.lname) as NAME
FROM   sales_rep
;
name  Peter Jones
name  John Franklin

2 row(s) retrieved.
Database closed.

```

7.6.4 Where to record the passwords

Consult with your corporate security officer about where and how to record the passwords when they are chosen. If the correct password is not supplied in a `decrypt_char()` function, no data is returned. *Without the correct password, the data cannot be retrieved by any means.* It is important to have a record of the passwords stored in a secure location.

7.6.5 Using password hints

The encryption functions allow a hint to be included to assist in recalling the password. The `GETHINT()` function retrieves the hint. Example 7-9 depicts the

usage. This example selects just one row. If the predicate results in multiple rows being returned, you get the hint for each row even if they are identical. If you know that the hints are identical, then use the `DISTINCT` keyword to reduce the number of returned rows.

If you use the `GETHINT()` function when no hint was set up, then you get no error, just a set of null values.

Example 7-9 The `GETHINT()` function

```
> select * from call_type where call_code = 'X';

call_code    X
code_descr   Test of a lengthy description.
enc_code
enc_descr
1bZIQAAAAIAP4PoYtU4yma3rW0VDFEoDKmHK62JYMt1EJPPR3MGDXXiR/0mj1cMu1DP
8EVcR0PsohMB0ZQaB2w=

1 row(s) retrieved.

> select gethint(enc_descr) from call_type where call_code = 'X';

(expression)

abbotcostello

1 row(s) retrieved.
```

7.7 Making room for encrypted data

In this section we discuss the space considerations for encrypted data.

7.7.1 Determining the size of encrypted data

Naturally, encrypted data requires more room than unencrypted data. How much more is relatively easy to calculate for fixed-length columns. For variable-length columns, you have to make some choices. In either case, the size depends on the size of the password and on the size of the hint, if a hint exists.

Be conservative. If you put encrypted data into a column that is too small, *there is no warning or error*.

The LENGTH function provides a convenient way to calculate the storage requirements of encrypted data directly, as shown in Example 7-10.

Example 7-10 Sizing encrypted data using LENGTH()

```
execute function length(encrypt_tdes('1234567890123456', 'simple
password'));
```

```
(expression)
          55
```

1 row(s) retrieved.

```
execute function length(encrypt_tdes('1234567890123456', 'simple
password', '1234567890123456789012'));
```

```
(expression)
          87
```

1 row(s) retrieved.

```
execute function length(encrypt_aes('1234567890123456', 'simple
password'));
```

```
(expression)
          67
```

1 row(s) retrieved.

```
execute function length(encrypt_aes('1234567890123456', 'simple
password', '1234567890123456789012'));
```

```
(expression)
          99
```

1 row(s) retrieved.

Alternatively, the Guide to SQL: Syntax, G251-2284 manual includes a chart with size estimates for size ranges for each of the encryption methods, with and without hints. The formulae for calculating exact sizes are also provided in the manual.

7.7.2 Errors when the space is too small

If you fail to make the columns for encrypted data large enough, then you will be allowed to insert data into those columns with no warnings or errors. However, any attempt to retrieve the data will fail. The data is truncated during the insert or update operation, but no notice is given. The truncated data is then not sufficient to allow decryption. So the decryption functions will fail. Example 7-11 demonstrates the problem. In the example, a column of 87 bytes is large enough, but a column of 86 bytes is too small. The insert succeeds in both cases, but the select fails when the column is too small.

Example 7-11 Column too small for encrypted data

```
C:\idsdemos\stores9>dbaccess -e stores9 example
```

```
Database selected.
```

```
alter table call_type drop enc_descr;  
Table altered.
```

```
ALTER TABLE call_type ADD enc_descr char(87);  
Table altered.
```

```
SET ENCRYPTION PASSWORD 'whosonfirst' with hint 'abbotcostello';  
Encryption password set.
```

```
update call_type set enc_descr = encrypt_tdes(code_descr) where  
call_code = "X";  
1 row(s) updated.
```

```
select  decrypt_char(enc_code, "erewhon") as decrypted_code  
        , decrypt_char(enc_descr) as decrypted_description  
from    call_type  
where   call_code = 'X'  
;
```

```
decrypted_code  
decrypted_descript+ Test of a lengthy description.
```

```
1 row(s) retrieved.
```

```
alter table call_type drop enc_descr;  
Table altered.  
ALTER TABLE call_type ADD enc_descr char(86);  
Table altered.
```

```
SET ENCRYPTION PASSWORD 'whosonfirst' with hint 'abbotcostello';
Encryption password set.
```

```
update call_type set enc_descr = encrypt_tdes(code_descr) where
call_code = "X";
1 row(s) updated.
```

```
select  decrypt_char(enc_code, "erewhon") as decrypted_code
        , decrypt_char(enc_descr) as decrypted_description
from    call_type
where   call_code = 'X'
;
```

```
26012: The internal base64 decoding function failed.
Error in line 26
Near character position 20
```

```
Database closed.
```

7.8 Processing costs for encryption and decrypting

In addition to the SQL issues, there are processing costs for encrypting and decrypting data. These operations are not free, and they do consume CPU cycles. Because the algorithms are complex, the encryption and decryption routines are executed in separate Virtual Processors (VPs). The CPU costs for the ENCRYPT VPs can be reported either by **onstat -g glo** or by querying the **sysmaster:sysvpprof**.

Any estimate of costs of using encrypted data is usually based on a limited set of test cases. The actual costs for any particular application and workload might be much more or less than any estimate.

In an extreme test, we inserted encrypted and unencrypted values into a temporary table and measured the costs as reported by **onstat -g glo** and by **time**. Example 7-12 on page 237 shows the results.

This example is an extreme case, and the encryption routines dominate the CPU usage. In a more normal workload, many SQL statements will get more than a single encrypted column or no encrypted data. In addition, the mix of SQL statements will include insert, update and delete statements, and each of those might or might not operate on encrypted columns. That mix (how much of the SQL actually uses encrypted data and how often each statement is executed)

will be the most important factor in how much extra CPU power is needed to maintain response times using encrypted data.

Example 7-12 CPU costs of encryption

```
onstat -z && sqlcmd -B -d erasmo_enc -f - < timedecrypt.sql && onstat
-g glo
IBM Informix Dynamic Server Version 10.00.UC3E -- On-Line -- Up
00:18:48 -- 37092 Kbytes
Fri Sep 15 15:52:46 2006
+ CONNECT TO 'erasmo_enc' WITH CONCURRENT TRANSACTIONS
Time: 0.019028
Fri Sep 15 15:52:46 2006
+ SELECT SUBSTRING(manu_code FROM 1 FOR 3) kk from kk WHERE 1 = 0 INTO
TEMP t3;
Time: 0.002010
Fri Sep 15 15:52:46 2006
+ insert into t3
select SUBSTRING(decrypt_char(e_manucode, "gulliver") FROM 1 FOR 3) kk
from kk ;
Time: 22.832941
```

```
IBM Informix Dynamic Server Version 10.00.UC3E -- On-Line -- Up
00:19:12 -- 37092 Kbytes
MT global info:
sessions threads vps lngspins
1 16 9 0
sched calls thread switches yield 0 yield n yield forever
total: 1765261 1764134 1089 42 2373
per sec: 312 312 0 0 156
```

```
Virtual processor summary:
class vps usercpu syscpu total
cpu 1 5.99 4.73 10.72
aio 2 0.07 0.05 0.12
lio 1 0.00 0.02 0.02
pio 1 0.00 0.00 0.00
adm 1 0.00 0.00 0.00
msc 1 0.00 0.00 0.00
encrypt 1 10.51 1.87 12.38
ETX 1 0.00 0.00 0.00
total 9 16.57 6.67 23.24
```

```
Individual virtual processors:
vp pid class usercpu syscpu total
1 14568 cpu 5.99 4.73 10.72
```

2	14569	adm	0.00	0.00	0.00
3	14570	ETX	0.00	0.00	0.00
4	14571	lio	0.00	0.02	0.02
5	14573	pio	0.00	0.00	0.00
6	14575	aio	0.07	0.05	0.12
7	14577	msc	0.00	0.00	0.00
8	14580	aio	0.00	0.00	0.00
9	14983	encrypt	10.51	1.87	12.38
		tot	16.57	6.67	23.24

```
=====
onstat -z && sqlcmd -B -d erasmo_enc -f - < timenormal.sql && onstat
-g glo
IBM Informix Dynamic Server Version 10.00.UC3E -- On-Line -- Up
00:16:55 -- 37092 Kbytes
Fri Sep 15 15:50:53 2006
+ CONNECT TO 'erasmo_enc' WITH CONCURRENT TRANSACTIONS
Time: 0.019973
Fri Sep 15 15:50:53 2006
+ SELECT SUBSTRING(manu_code FROM 1 FOR 3) kk  from kk WHERE 1 = 0 INTO
TEMP t3;
Time: 0.001703
Fri Sep 15 15:50:53 2006
+ insert into t3
select SUBSTRING(manu_code FROM 1 FOR 3) from kk;
Time: 1.220955
```

```
IBM Informix Dynamic Server Version 10.00.UC3E -- On-Line -- Up
00:16:57 -- 37092 Kbytes
MT global info:
sessions threads  vps      lngspins
1         16         9         0
sched calls      thread switches yield 0   yield n   yield forever
total:    2056           1003      1115     6       420
per sec:  478           478        0       0       239
```

```
Virtual processor summary:
class      vps      usercpu   syscpu   total
cpu        1        0.95      0.03     0.98
aio        2        0.01      0.01     0.02
lio        1        0.01      0.01     0.02
pio        1        0.00      0.00     0.00
adm        1        0.00      0.00     0.00
msc        1        0.00      0.00     0.00
encrypt    1        0.00      0.00     0.00
ETX        1        0.00      0.00     0.00
```


total	9	0.97	0.05	1.02

Individual virtual processors:

vp	pid	class	usercpu	syscpu	total
1	14568	cpu	0.95	0.03	0.98
2	14569	adm	0.00	0.00	0.00
3	14570	ETX	0.00	0.00	0.00
4	14571	lio	0.01	0.01	0.02
5	14573	pio	0.00	0.00	0.00
6	14575	aio	0.01	0.01	0.02
7	14577	msc	0.00	0.00	0.00
8	14580	aio	0.00	0.00	0.00
9	14983	encrypt	0.00	0.00	0.00
		tot	0.97	0.05	1.02

Authentication approaches

In this chapter, we discuss how the DBMS authenticates the user identity with a connection is established or rejected. IDS 9.40 introduced some new choices, and IDS V10 adds more choices. Table 8-1 lists the options and their attributes. You can choose any of these methods for each DBSERVERNAME or DBSERVERALIAS, but you can choose only one option for each. If you need for users to connect using multiple methods, you must provide multiple DBSERVERALIAS, one per authentication method.

Table 8-1 Authentication choices

Method	Attributes
OS Userid	No encryption, uses OS password lookup
Password Encryption	As for OS Userid, but with the password encrypted during transmission
Pluggable Authentication Module (PAM)	User-provided authentication methods
LDAP	User-provided access to the LDAP directory

We discuss each of these choices in detail, as well as the use of roles in controlling access to data. However, before you think about DBMS security, you should remember where this all fits from the perspective of a broader security policy.

8.1 Overall security policy

Security of data is essential to the long-term success of any organization, but this is only one aspect of a proper security program. Policies and enforcement defined should exist for access to each computer system, application and database in the organization. The people responsible for these policies are probably not in the DBA group, but are more likely to be corporate officers (the CIO or equivalent). As you consider how to configure DBMS authentication, consult these policies and choose the technique that best fits those requirements.

Secondly, in addition to user authentication, there are other aspects of the DBMS installation and configuration you should consider. In particular, IDS V10 checks a number of files as the DBMS is starting up and puts warning messages in the message log if key files do not have properly secure permissions. You should periodically review the log and correct any problems that have been found.

Also be sure the database objects (such as tables, indexes, and procedures) are each restricted to only those users and uses required for your business needs. Guidelines for setting these permissions should be part of the corporate security policies.

There are other things too. For example, the database backup and log archive files should be stored securely. That means having the most restrictive permissions your business needs can tolerate, and it means storing the media in some secure location.

In this chapter, we do not discuss all these other aspects of security, but you should think about user authentication the same way. Do what is required by policy and restrict things to what your business requires.

8.2 To trust or not to trust, that is the question

Before checking a user ID and password combination, certain lists of trusted users or trusted systems are checked by the DBMS to decide whether or not to accept a connection. Which of these lists are or are not checked is determined by settings in field five of each SQLHOSTS entry. Field 5 can contain several different kinds of specifications, among them **r=something** and **s=something**. The **r=** values are used by the client libraries and ignored by the server. The **s=** values are used by the server and ignored by the client libraries.

For **s=4**, a Pluggable Authentication Module (PAM) is used. PAMs and their configuration are discussed in 8.5, “Using pluggable authentication modules (PAMs)” on page 250.

For **s=6**, the connection is restricted. It can be used only by the replication (ER or HDR) threads for transferring replicated data between IDS instances. No other uses are permitted. Even the administrative commands for replication must use some other connection.

8.2.1 Complete trust

For **s=1** or **s=3**, the DBMS checks to see if the user or host or both are listed in the `/etc/host.equiv` file (Linux or UNIX) or has a trust relationship (Windows). If so, then any connection request is accepted. No password is required or checked. A password can be submitted, but if it is, it must be correct. The connection is refused if the password is incorrect. This technique should be used only rarely because it is so trusting.

It is required for systems performing distributed queries. That is, if a query refers to a database on another system, the two systems must trust each other because the IDS engine does not use a password when connecting to the remote system during query processing.

For the same reason, this is required for instances participating in Enterprise Replication. No ID or password is used for the connections between the instances passing the replication data.

Other uses of this technique should be rare, because a system or user in this list is trusted always.

This is demonstrated in Example 8-1. The database name and server name are provided without any user name or password, and the connection is established. Compare this to Example 8-2 on page 244, where the server name is different. In that case, the connection is rejected because the client is not trusted by the server.

Example 8-1 Connection with **s=3**

```
informix@nile:~> dbaccess stores9@ban_er1 -
```

```
Database selected.
```

```
>
```

8.2.2 Partial trust

For **s=2** or **s=3**, a different list of trusted users and systems is consulted. Each user's home directory can include a file named **.rhosts**. This file is similar to **/etc/hosts.equiv**, but its use is limited to the specific user. Note that for **s=3**, both files are used.

Using **.rhosts**, you can configure things so that each user is trusted from a few specific systems and required to submit a password from all other systems. This is useful when the majority of their work is performed from a fixed location, such as their office desktop system. When they are at that system, they are trusted. When they use any other system, they are not trusted.

The **s=2** setting disallows distributed queries because it enables only the **.rhosts** lookup, not the **/etc/hosts.equiv** lookup.

8.2.3 No trust at all

For **s=0**, a password is required. No checks for trusted systems or users are performed. If this setting is used, then distributed queries are impossible and replication is impossible. Example 8-2 demonstrates the error that is received when no password is supplied, and a correct connection if a password is provided.

Example 8-2 Connecting with s=0

```
informix@nile:~> dbaccess stores9@kod_auth1 -
```

```
956: Client host or user informix@nile.itsosj.sanjose.ibm.com is not
trusted by the server.
```

```
informix@nile:~> dbaccess - -
> connect to 'stores9@kod_auth1' user 'informix';
ENTER PASSWORD:
```

```
Connected.
```

```
> select first 2 * from call_type;
```

```
call_code code_descr
```

```
B      billing error
D      damaged goods
```

```
2 row(s) retrieved.
```

```
>
```

8.2.4 Impersonating a user

For the value `r=1`, a client program can present an identity (user ID and password) other than their own. In this case, the file `.netrc` in the user's home directory is examined. If a set of credentials is listed there for the host where the DBMS exists, then those credentials are used in place of the user's real identity.

However, if the user includes a password in the connect statement, then the `.netrc` file is ignored.

Note that this setting must be in the `sqlhosts` file or registry on the machine where the client is executing. It is ignored in the `sqlhosts` file on the machine where the DBMS is executing.

Important: *This is a very risky thing to do. One or more user IDs and passwords are stored in plain text on a client system. Unless you are sure that system is not accessible by anyone except its intended users, the credentials are at risk. Because those credentials are valid, if they are used from another system, they will be accepted. Do not use this unless you have no other choice.*

The value `r=0` disables the `.netrc` lookup and is recommended to prevent unwanted impersonations.

Example 8-3 shows a sample `.netrc` file. This file is in the home directory of user `informix`. If a connection is made to a server name for which `r=1` is set in the `sqlhosts` entry in the `sqlhost` file on the client machine, then the user ID `richard` and password `96eagles` are used rather than the current user ID.

Example 8-3 Sample .netrc file

```
informix@nile:~> cat .netrc
machine kodiak
    login richard
    password 96eagles
```

8.3 Basic OS password authentication

IDS has always used this basic authentication. This technique requires an OS user ID and password for each user who will connect to the DBMS. The user ID and password are submitted by the user or application program, and the DBMS verifies the password using an OS library function. If the OS function indicates the user ID or password (or both) are not in the OS set of user IDs and

passwords, then the DBMS connection is rejected with error 951 (the user ID) or 952 (the password).

In this scheme use the OS administrative tools to create users and accounts. If using AIX, for example, the tool is SMIT. If you use Linux, use YAST or SAX2 to accomplish these tasks.

An example using SUSE Linux: We created a user ID and password for a new user. The entire process is shown in Example 8-4, and is very simple.

Creating the accounts and passwords must be done by the user root.

Example 8-4 Setting up and using Linux user IDs and passwords

```
useradd -d /home/tom -p 96eagles tom
useradd -d /home/dick -p 72mercury dick
```

Example 8.5 shows the DBA (user informix) granting permission to connect to the DBMS and to certain databases for the other users. Just having an account is not sufficient to connect to a database. The user must also have at least connect permission to some database. DBMS connections will be refused for valid OS accounts if no permissions for that user have been defined in the DBMS. The example also shows the errors that result when an incorrect user name or password is used.

The first connection uses a user ID that does not exist. The other connections use valued user IDs.

It is not necessary for all users to have permission to use all databases. Each user should only have permission to use the databases required by their work. This is where roles are most useful. Roles are discussed further in 8.7, “Roles” on page 252.

Example 8-5 Connect permission and connection attempts

```
connect to 'stores9' user 'mike' using '94tarheels';
951: Incorrect password or user mike@IBM-5AEA059BC79 is not known on
the database

951: Incorrect password or user  is not known on the database server.
Error in line 2
Near character position 1
```

```
connect to 'stores9' user 'harry' using '94tarheels';
387: No connect permission.
```


111: ISAM error: no record found.

Error in line 4

Near character position 1

```
connect to 'stores9' user 'informix' using 'in4mix';
Connected.
```

```
grant resource to harry;
Permission granted.
```

```
grant connect to tom;
Permission granted.
```

```
disconnect current;
Disconnected.
```

```
connect to 'stores9' user 'tom' using '94tarheels';
Connected.
```

```
select first 2 * from call_type;
272: No SELECT permission.
Error in line 18
Near character position 31
```

```
disconnect current;
Disconnected.
```

```
connect to 'stores9' user 'harry' using '94tarheels';
Connected.
```

```
select first 2 * from call_type;
272: No SELECT permission.
Error in line 24
Near character position 31
```

```
select first 2 * from orders;
```

order_num	1001
order_date	05/20/1998
customer_num	104
shipping	ROW('06/01/1998',20.40 ,'\$10.00','express')
backlog	t
po_num	B77836
paid_date	07/22/1998

```
order_num      1002
order_date     05/21/1998
customer_num   101
shipping       ROW('05/26/1998',50.60      ,'$15.30','PO on box; deliver
to back
               door only')
backlog        t
po_num         9270
paid_date      06/03/1998
```

2 row(s) retrieved.

```
disconnect current;
informixDisconnected.
```

```
connect to 'stores9' user 'informix' using 'in4mix';
Connected.
```

```
grant select on call_type to harry;
Permission granted.
```

```
disconnect current;
Disconnected.
```

```
connect to 'stores9' user 'harry' using '94tarheels';
Connected.
```

```
select first 2 * from call_type;
```

```
call_code code_descr
```

```
B      billing error
D      damaged goods
```

2 row(s) retrieved.

8.4 Encrypting passwords during transmission

The use of an OS ID and password takes advantage of the password being encrypted in the files used by the OS. However, the password is sent in plain text

from the application to the DBMS. It might be useful or required to encrypt the passwords as they move between the programs.

To accomplish this encryption, specify the use of the encryption in the SQLHOSTS file and the conccsm.cfg file. Both these files are usually in \$INFORMIXDIR/etc. The conccsm.cfg file can be elsewhere, but then the INFORMIXCONCCSM environment parameter must specify the complete path to the file.

Both the server and client systems have to configure password encryption in their respective SQLHOSTS files or registries and the conccsm.cfg file. If only one system configures encryption, then the password will be either be encrypted when the DBMS is not expecting it, or will not be encrypted when it should be.

Be sure the client applications are using the proper DBSERVERNAME or DBSERVERALIAS. By having multiple names, some connections can have encrypted passwords while others do not.

Example 8-6 shows the sqlhosts file entries and related conccsm.cfg file entry for an IDS instance with a name and two aliases. The first two sqlhosts entries do not use password encryption. The third one does. Note that the name provided in the sqlhosts entry (SPWDCCSM in this case) must match the entry in the conccsm.cfg file.

In the conccsm.cfg file, the parameters following the CSM name vary for the different CSM modules. In this case, the first parameter is the full path of the library for doing the encryption and decryption. The second parameter is not used. The third parameter is null in this example. See the IDS Administrators Guide, G251-2267, for details of the other choices for the third parameter.

There is no way to observe or confirm the use of the encryption libraries, so no example is provided. If the configuration is not correct you will get error 930, "Cannot connect to database server (servername)".

Example 8-6 Sample SQLHOSTS and CONCCSM.CFG for password encryption

sqlhosts entries:

kod_auth1	onsoctcp	kodiak	kod_auth1	s=0
kod_auth2	onsoctcp	kodiak	kod_auth2	r=1
kod_auth3	onsoctcp	koidak	kod_auth3	csm=(SPWDCCSM)

conccsm.cfg entry;

```
SPWDCCSM("/usr/informix/lib/csm/libixspw.so", "", "")
```

8.5 Using pluggable authentication modules (PAMs)

The third option for user authentication is the use of a PAM. You can choose among modules available from third parties, or you can write your own. General information about PAMs is available at:

http://inetsd01.boulder.ibm.com/pseries/en_US/aixbman/security/pam_overview.htm

Or

<http://www.sun.com/software/solaris/pam/>

A description of using a PAM with an Informix ESQL/C program is at:

<http://www-128.ibm.com/developerworks/db2/zones/informix/library/techarticle/0306mathur/0306mathur.html>

8.5.1 Basic configuration

A *PAM* is a set of libraries and configuration files. The libraries contain the routines, and the configuration files instruct programs when to use the various routines.

For IDS, the libraries usually reside in \$INFORMIXDIR/lib and the configuration files in \$INFORMIXDIR/etc. Both are referenced in the conccsm.cfg file.

The use of a PAM is specified in the sqlhosts file entry for the DBSERVERNAME or DBSERVERALIAS being used. The fifth field of the entry specifies the name of the PAM and must match the name of an entry in the \$INFORMIXDIR/etc/conccsm.cfg file. This is exactly the same mechanism as for the password encryption module describe in 8.4, “Encrypting passwords during transmission” on page 248.

Example 8-7 shows an SQLHOSTS entry using a PAM. In this example, the s=4 indicates that the rest of the parameters refer to a PAM. The library containing the PAM functions is called authpam. The other specification is that only a password is required. If this PAM was going to issue a challenge, then entry would read pamauth=(challenge).

Example 8-7 SQLHOSTS entry using a PAM

```
kod_auth3 olsoc tcp kodiak kod_auth3 s=4, pam_serv=(authpam),  
pamauth=(password)
```

8.5.2 Using password authentication

Using a password authentication PAM is similar in concept to the basic IDS authentication using OS accounts. The difference is that some other method is used to check whether or not the user ID and password are acceptable. The application program operates exactly the same way, and a user ID and password are submitted the same way. The only difference is that the OS is not used to check the credentials. That other method might be a third-party method such as Kerberos, or you might choose to write your own method.

8.5.3 Using challenge-response authentication

Another technique for authentication is to return a *challenge* to the application. If the proper response is provided for the challenge, then the connection is granted. The idea is similar to the technique used in some military operations. A sentry calls out a challenge question. If the person wanting to gain access knows the right answer, then they are allowed to enter.

8.5.4 Application considerations for challenge-response

If a challenge-response authentication method is chosen, then each application must be programmed to do something differently. First, the challenge must be accepted. Secondly, the proper response to the challenge must be sent after the challenge has been received.

This technique usually requires a set of callback functions in the application. The details might differ from one PAM to another, so consult the manuals for the product you have chosen.

If some applications cannot be altered to meet the requirements of the PAM, then a different DBSERVERNAME or DBSERVERALIAS must be provided and used.

8.6 Using an LDAP directory

The LDAP support provided in IDS requires some programming. A skeleton module is provided, but it must be completed according to the details of each installation. The IDS Administrators Guide, G251-2267-02 includes instructions on how to complete, compile and configure the LDAP support module.

You can find more general information about LDAP at the following Web sites:

<http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg244986.html?Open>

http://inetsd01.boulder.ibm.com/pseries/ca_ES/aixbman/security/ldap_exploitation.htm

You must have some client-side modules compatible with your LDAP server in order to complete the work. Those modules are not provided with the IDS release but might be provided by the LDAP server provider. Follow the instructions provided there in addition to what is in the *IDS Administrators Guide*, G251-2267.

An almost fully functional sample LDAP module is provide in the \$INFORMIXDR/demo/authentication directory. If you are using Microsoft Windows, then this module might be sufficient for your needs, but it provides password authentication only. Note that *these files must be modified* to include the correct names and addresses for the chosen LDAP server and service.

8.7 Roles

Roles are just sets of privileges. Privileges might be granted to a role rather than to a user. When that is done the role might be granted to a user, and that is usually simpler than granting a set of privileges to a set of users.

In addition, a number of roles might be granted to one user. The user can then choose which role (that is, which set of privileges) should be in effect by using the SET ROLE statement.

Example 8-8 on page 253 shows the technique. Here the following steps are performed:

1. Create two roles.
2. Grant the roles to a user.
3. Connect as that user.
4. Verify privileges.
5. Use the SET ROLE statement to adopt the other role.
6. Show that the privileges have changed.

8.7.1 Default roles

Each user can have a default role assigned. If so, then that role is the set of privileges the user has when connecting to the DBMS. A SET ROLE statement is required if different privileges are needed, and is demonstrated in Example 8-8. The default role has only select privileges. In order to update the table, the more privileged role must be set.

One use of default roles is to restrict what users can do outside certain applications. If the user default role is very restricted, then an application can use the SET ROLE statement to have its required privileges when using the application. Outside the application, the user is restricted. That helps control who can make ad-hoc changes to a database or query for restricted data.

Example 8-8 Using a role

Database selected.

```
create role reporting;  
Role created.
```

```
create role fullpriv;  
Role created.
```

```
grant select on customer_log to reporting;  
Permission granted.
```

```
grant select on items to reporting;  
Permission granted.
```

```
grant select on location_non_us to reporting;  
Permission granted.
```

```
grant select on location_us to reporting;  
Permission granted.
```

```
grant select on manufact to reporting;  
Permission granted.
```

```
grant select on msgs to reporting;  
Permission granted.
```

```
grant select on orders to reporting;  
Permission granted.
```

```
grant select on region to reporting;
```

Permission granted.

grant select on retail_customer to reporting;
Permission granted.

grant select on sales_rep to reporting;
Permission granted.

grant select on state to reporting;
Permission granted.

grant select on stock to reporting;
Permission granted.

grant select on stock_discount to reporting;
Permission granted.

grant select on units to reporting;
Permission granted.

grant select on whlsale_customer to reporting;
Permission granted.

grant select, insert, update, delete on call_type to fullpriv;
Permission granted.

grant select, insert, update, delete on cat_hits_log to fullpriv;
Permission granted.

grant select, insert, update, delete on catalog to fullpriv;
Permission granted.

grant select, insert, update, delete on cust_calls to fullpriv;
Permission granted.

grant select, insert, update, delete on customer to fullpriv;
Permission granted.

grant select, insert, update, delete on customer_log to fullpriv;
Permission granted.

grant select, insert, update, delete on items to fullpriv;
Permission granted.

grant select, insert, update, delete on location_non_us to fullpriv;

Permission granted.

grant select, insert, update, delete on location_us to fullpriv;
Permission granted.

grant select, insert, update, delete on manufact to fullpriv;
Permission granted.

grant select, insert, update, delete on msgs to fullpriv;
Permission granted.

grant select, insert, update, delete on orders to fullpriv;
Permission granted.

grant select, insert, update, delete on region to fullpriv;
Permission granted.

grant select, insert, update, delete on retail_customer to fullpriv;
Permission granted.

grant select, insert, update, delete on sales_rep to fullpriv;
Permission granted.

grant select, insert, update, delete on state to fullpriv;
Permission granted.

grant select, insert, update, delete on stock to fullpriv;
Permission granted.

grant select, insert, update, delete on stock_discount to fullpriv;
Permission granted.

grant select, insert, update, delete on units to fullpriv;
Permission granted.

grant select, insert, update, delete on whlsale_customer to fullpriv;
Permission granted.

grant default role reporting to tom;
Permission granted.

grant fullpriv to tom;
Permission granted.

Database closed.

```

> connect to 'stores9' user 'tom';
  ENTER PASSWORD:

Connected.

> select first 2 order_num, po_num from orders;

      order_num po_num
      1001 B77836
      1002 13

2 row(s) retrieved.

> update orders set po_num = 9270 where order_num = 1002;

273: No UPDATE permission.
Error in line 1
Near character position 15

> set role fullpriv;

Role set.

>
> update orders set po_num = 9270 where order_num = 1002;

1 row(s) updated.

> set role reporting;

Role set.

> update orders set po_num = 9270 where order_num = 1002;

273: No UPDATE permission.
Error in line 1
Near character position 16
>

```

Legendary backup and restore

In every job, there are inevitabilities—tasks that must be done as well as situations that are guaranteed to occur regardless of how much one might wish or hope otherwise. In database server operations, one of the most important required tasks is the creation and management of instance backups. The inevitable is that at some moment in time, these backups will be needed to recover from either a systemic or user problem.

Throughout the technical life of IDS, it has always been preeminent in the backup and restore functionality it provided. With version 10, new technology was added giving administrators greater granularity in restoring data as well as flexibility in executing backups. In this chapter we briefly recap the existing IDS backup and recovery technologies and explain what was added in version 10. Future versions of the database server promise to include even more flexibility and capability.

9.1 IDS backup and restore technologies

As database sizes have grown and data sophistication and complexity has increased, requirements to manage data have changed over time. A key component of the new requirements has been the ability to backup and restore only what was needed or critical in a timely manner, whether it be the entire data store or a focused sub-set of the data. IDS has either led or kept pace with the industry in terms of data backup and restoration, as we explain in this section.

9.1.1 Cold, warm, and hot as well as granularity

So what does temperature have to do with data backup and restoration? Not much unless you consider database server users getting “hot” under the collar if their access is interrupted because of a backup operation or the “cold” stare from management when told a key data set cannot be recovered either at all or in a reasonable amount of time. No, the terms cold, warm and hot refer to the invisibility of backup or restoration operations to user activities.

From the very beginning, IDS has been able to execute full or incremental hot backups without interrupting user operations. In addition, extra steps have never been necessary to isolate a separate static copy of data to execute the backup in order to ensure its consistency. Backup can be executed on production tables and will run in background mode to other operations occurring within the instance. This does not mean the backup operation is starved for resources and requires a long time to execute. Because of the threaded nature of the database server, there is more than sufficient power to handle both kinds of concurrent operations. The limiting factor turns out to be the I/O channel of the backup device.

Where a hot backup is completely invisible to users, instance operations can be halted to execute a cold backup if desired. This might be necessary to migrate a static data image from one instance to another or to create a reference copy of the instance. As a point of information, there is no such thing as a warm backup.

Backup operations can be executed at several levels of granularity and, if the ON-Bar utility suite is used, include either the entire instance or just a subset of the spaces. Both ontape and ON-Bar, the most frequently used utilities, support two levels of incremental backups as well as a full backup, as summarized in Table 9-1.

Table 9-1 IDS backup levels

Backup level	Description
0	Complete instance backup NOTE: Temporary space pages (either standard or smart) are <i>never</i> included in backup operations.
1	All instance pages which have changed since the last level 0 backup
2	All instance pages which have changed since the last level 1 backup

With these incremental levels, administrators can intelligently manage backup media resources as well the time required to execute backup operations based on business needs and requirements. For example, in an environment where a relatively small amount of data changes from week to week and the amount of time required to effect a complete restore is not extremely critical, a level 0 backup can be taken on the first of the month, a level 1 the first of each successive week, with a level 2 every other business day. If, for example, the time to restore is more important and there is sufficient backup media available, then daily level 0 backups should be taken. Finally, to balance media usage and restore time in an environment where there are many data changes, a level 0 could be executed the first of each week with a level 1 every day. There are any number of variations which can be used depending on business requirements and constraints.

As might be expected and as illustrated in Figure 9-1, the nested nature of backup operations affects how restores are executed. The timeline in the figure shows a series of backup operations and the required restore operations. Using the first example from above, if a complete restore is required, the level 0 created at the first of the month, the level 1 from the beginning of the week and the level 2 media from the previous day would need to be restored, followed by the logical logs.

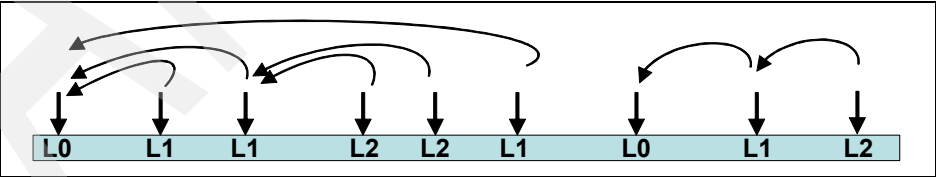


Figure 9-1 Incremental backups and restoration steps

Obviously, the longer the time between backup levels or the amount of changed data will affect the size and time required to create either a level 1 or 2 backup.

From a restore perspective, depending on the restoration needs there are only two temperature options, cold, warm and an interloper called mixed. If any of the critical instance dbspaces are lost, such as the rootdbs or any space containing logical logs or the physical log, the entire instance must be restored through cold restore. If several backup levels exist, each should be restored as part of this cold restore along with the logical logs to bring the recovery point as close as possible to when the instance went down. If a non-critical dspace requires restoration, it can be restored in warm mode with user operations occurring in other parts of the instance. Obviously the spaces being restored are blocked to user access until the restoration has completed. A mixed restore is the combination of a cold restore to recover the entire instance to a specific moment in time followed by one or more warm restores. A mixed restore might be used to pull one or more spaces to a more current moment in time to get around an undetected problem in another space that occurred at an earlier time. Another potential use of a mixed restore could be in the recovery of an instance supporting OLTP and historical analysis. The cold restore would bring back the instance allowing users access to their tables for day-to-day work while the warm restore was used to restore historical tables stored in other spaces.

Both types of restores can either be complete (to the latest recorded logical log record) or, depending on the utility used, to a specific moment in time. In either case, logical consistency of the data is ensured which can have consequences in the restored and other spaces during a warm restore. For example, using ontape the granularity of a warm restore of dbspaces is limited timewise to an individual logical log. After the full / incremental backups have been restored, each log can be restored in sequential order. The logical recovery can be halted at any log but when stopped, the instance is checked for logical consistency. If the data is not consistent, the restored space will be marked down, as shown in Example 9-1.

Example 9-1 Consequences of not completing a logical restore

Noon: a level 0 backup is created, current logical log #17

12:05 pm: table_1 with a primary key is created in dspace_a and loaded with 5 rows of data

12:10 pm: table_2 with a foreign key referencing table_1 is created in dspace_b and loaded with 5 rows of data

12:15 pm: logical log 17 fills is backed up to tape, current log #18

12:20 pm: 5 more rows of data are inserted into table_1 and table_2

12:21 pm: a user drops a table in dspace_a requiring the space to be restored

12:30 pm: Warm restore begins on dbspace_a
instance backup restored
logical log 17 restored but not 18 since it contains the dropped
table operation

12:35 pm: restore operation completes but dbspace_a is off-line

At the end of this restore operation, table_2 had 10 rows of data while table_1 only had 5 rows. The logical consistency check failed because of orphaned child rows in a foreign key table and dbspace_a was marked offline.

With the ON-Bar utility suite, restores, either cold or warm, can be executed to a specific second in time and, like ontape, can be focused to one or more specific dbspaces. Restoring to the granularity of a second assumes that full and complete logical log backups exist to cover the time after the latest full / incremental instance backup has been restored. As with ontape, when the restore operation is completed, or stopped after a specific logical log, logical consistency is checked.

With IDS V10 there is new technology that permits recovering just the dropped table from dbspace_a without affecting the rest of the tables in the dbspace. We discuss this functionality in 9.3.2, “Table Level Point-in-Time Restore (TLR)” on page 275.

As evidenced by references in this section, IDS has at least two utilities for backing up and recovering an instance and its logical log records. These will be discussed next.

9.1.2 Ontape

Ontape, in its current form, is the latest version of the backup utility which has been included in all versions of Informix database servers from the very beginning. It is a work-horse type of product, and a significant number of customers still use it today even though more advanced functionality exists. While ontape has seen functional enhancements such as that described in 9.3.1, “Ontape to STDIO” on page 274, it still remains more of a hands-on, administrator-driven utility.

Designed to be used with up to two locally-connected backup devices (one for instance and the other for logical log operations), invoking a backup or restore operation requires responding to several administrative prompts. When the operation begins, it is executed serially beginning at chunk 0 and proceeding through to the last chunk created. If only one backup device is available, separate media must be used to backup logical log records. There is no intelligent tape handling to read the tape and advance past existing data stored

on the media; the process begins reading or writing at the beginning of the media. Where two devices are available, one can be configured within the instance for logical log backups and `ontape` can be invoked to automatically back up logical logs to this device as they fill. This is a foreground operation though requiring a dedicated terminal or terminal window to be open and active for the operation to continue.

While the assumption is that instance and logical log backups created with `ontape` are being output to tape, it is possible to backup to disk. It is important to note though that the utility simply opens the configured target device and begins to writing to it. If the instance backup device is configured to point to `/opt/backups/informix/prodinst_backup`, that disk file needs to exist and have the correct write permissions. In addition, the next time an instance backup is executed, regardless of the level, it will overwrite the existing file. As a result, it is critical that a process exist to copy and rename the existing backup file prior to invoking a new backup. The same applies to logical log records backed up to disk. If multiple logs are being backed up at the same time through an `ontape -a` command, all the logs will be written out in one file. However if the logs are backed up as they fill through the `ontape -c` foreground process described earlier, each log backed up will over write the existing file requiring the same kind of file management process.

9.1.3 ON-Bar utility suite

ON-Bar is referred to as a utility suite because it actually has two components, the ON-Bar API and the Informix Storage Manager (ISM), a basic tape management system which can be used in conjunction with the ON-Bar API. Unlike `ontape`, all user interaction prompts are expected to be handled by the tape management system making ON-Bar the preferred utility for unattended backups as well integration into an existing enterprise backup and recovery plan.

The ON-Bar API is the Informix implementation of the client component of the Open Systems Backup Services Data Movement (XBSA) Application Programming Interface (API) defined by the X/Open organization. The X/Open XBSA API itself was created as a standard through which applications could communicate to create and restore backups regardless of the platform. It was designed to provide five basic functions:

- ▶ Concurrent services: The ability to concurrently back up or restore defined data sets of any type in a heterogeneous environment.
- ▶ Object searches: The ability to search for a given backup object in order to initiate a restore operation.
- ▶ Scalability: The ability to reliably store and retrieve a data set of any size.

- ▶ System configuration independence: The ability to support a wide range of heterogeneous computing platforms and network configurations.
- ▶ Integrity: The ability to configure and run backup and restore operations as independent events so that the actions of one operation do not adversely affect other operations.

The interface contains four basic operational components as illustrated in Figure 9-2.

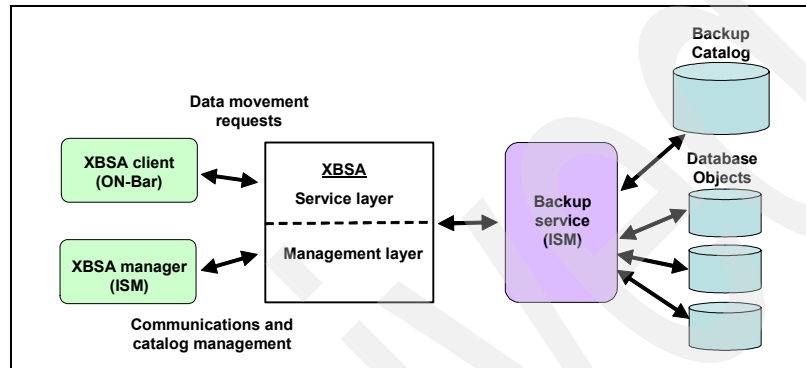


Figure 9-2 Components of the ON-Bar API

These components include the:

- ▶ XBSA Client: Software that sits next to the data that is to be backed up and responds to requests from the XBSA Manager to provide or receive data.
- ▶ XBSA Manager: Handles the communication and administrative overhead between the actual backup software and the client component. This component provides an interface into the image database catalog maintained by the Backup Service.
- ▶ Backup Service: The software that actually performs the read and write operations to the storage media. This software also builds and maintains a database that tracks the backup images that are available for use on an “as needed” basis.
- ▶ XBSA application: A user of the API, this is a generic term that can refer either to an XBSA client or an XBSA Manager.

The backup service component can be fulfilled by the ISM or any other third-party tape management system, such as Tivoli Storage Manager, Veritas NetBackup, and other products being used within the business to backup UNIX, Windows and other servers and their file systems.

The ON-Bar utility provides greater power, flexibility and granularity when executing backup and restore operations. While it has the same backup levels as ontape, when executing a backup it can be restricted to a specific subset of spaces instead of the entire instance. In this way, if a database is built such that static tables, whether reference or historical are stored in one set of spaces while volatile tables are stored in another set, only the dbspaces with the changing tables can be backed up. As previously mentioned, restore operations cannot only be limited to one or more spaces but to a specific moment in time with one second increments.

Because the ON-Bar API is not bound by a limit of physical backup devices (within reason), operations can be executed in parallel and or concurrently. Unlike ontape which uses a single I/O thread to read data from chunk 0 to N, ON-Bar will fork as many I/O threads as requested by the storage management software to provide a data stream for the available devices. For example, if there is a 5 device tape jukebox available, a backup operation can be defined in the storage management software to invoke 5 threads to stream data to all devices. When the job is executed, the instance will allocate the threads to the first 5 spaces to be backed up. When one thread finishes its space, the instance will point the thread to the next space to be backed up until all requested spaces have been backed up.

More than one operation can be executing concurrently as well. For example, with the same 5 device jukebox, two backup jobs can be defined to backup different spaces, one using N devices, the other 5 - N devices. Both jobs can then be executed at the same time if desired.

An ON-Bar operation can also be configured to execute serially if desired. Backup and restore operations cannot be mixed however, a serial backup can only be restored with a serial restore and vice versa.

There is a small amount of overhead in using ON-Bar regardless of the storage management software. A database is created in the rootdbs to manage configuration parameters and the instance's record of operations. This database is very tiny consisting of just a few tables. While it will contain a record of backup objects and operations, most storage management applications have a utility which communicates through the ON-Bar API to this database to purge entries as save sets and or media expires.

The ISM

Shortly after Informix released the ON-Bar API, a large number of customers responded that while they would like to use the functionality and power it provided, they did not want or could not afford to purchase a full storage management system. They just wanted to backup their instances and only needed a basic, limited functionality storage manager. Being an extremely

customer-focused company, Informix responded by bundling the ISM as part of the database server.

The ISM supports the configuration of up to four locally connected devices which can be used for sequential or parallelized operations. While device support is more limited than would be available in commercial releases of storage management software, the list of devices contains all of the most commonly used media types. Automated devices such as jukeboxes are NOT supported however. The ISM supports configuring devices to output to disk file though each file is limited to being 2 GB in size. Unlike ontape, when backing up to disk the ISM provides intelligent filename management automatically to avoid file overwrite conditions. These file names are maintained and automatically used as appropriate for a restore operation. No administrator intervention is required to rename or move disk files.

The ISM supports media pools and cloning as well as save set and media expiration so media can be cycled appropriately. Unlike commercial releases, each piece of media must be manually formatted and labeled (media labeling, not a physical sticker on the tape cartridge) before it can be assigned to a media pool. The ISM can interface with the ON-Bar database to purge expired save set and media records so the database size does not grow.

The ISM does not support job scheduling like commercial releases of management software but jobs can be automated by invoking the operation with a command line call by the OS crontab or other scheduling utility. Because automated devices are not supported, after the media is loaded (unless going to disk) and the job has been invoked, the ISM handles all the remaining tasks.

Overall, the ISM provides a fairly broad range of functionality and works well. While it might not have all the features of a commercial product, as a free bundle included with the database server, it is a great deal financially.

9.1.4 External backups

External backups and their associated recovery operations are primarily performed outside of either the ontape or ON-Bar utilities. Ontape or ON-Bar is only used to backup and restore the logical logs when the external backup/restore operation has been completed. In some respects, an IDS external backup is how many competing products required backups to be done until just recently. That is, to segment a copy of the entire instance and then back it up using OS utilities.

Having an external backup can, in certain situations, enable a faster restore. The concept behind an external backup is that a logically consistent and static copy of the instance is somehow copied using OS utilities. This could happen within a

disk farm by segregating a set of disks which have been acting as mirrors and remirroring with another set of disks. The partitions on the disks segregated can then be mounted and used by the OS as new devices. Some disk vendors provide this option as part of their configuration—a mirror set actually creates two sets of copies, not one, so the extra set can be removed as needed. This was done to facilitate other competing database servers creating a pseudo-hot backup because they could not maintain logical consistency during backups.

Another option if a third mirror set of disks is not available is to segment the mirror and use utilities such as **dd**, **tar**, **cp** and so on to copy the contents of the disks to OS files or tape devices. After the copy has completed, the disks are reinserted as mirrors and brought into consistency with the primary disks. Finally, if no mirrors are available, almost all instance operations are blocked while the contents of the production chunks are copied to OS files or tape devices as just mentioned.

If a failure occurs that requires a restore, rather than using Informix created backups, it might be faster to copy the data from the OS copies or the new devices that were segmented from a double mirror disk farm over the production devices. This is followed by a brief logical restore of the logical logs taken with **ontape** or **ON-Bar**.

Because external backup and restore operations occur outside of the instance, there are not any specific utilities for them, only flag options to administrative and **ontape** or **ON-Bar** utilities to block and unblock instance processing and to signal instance control structures that an external restore has occurred requiring a logical log roll forward operation.

9.2 Executing backup and restore operations

As shown so far, IDS has a broad and varied set of options to back up and restore instance data. While this is good, if these options are complicated to use they will not be executed as often as they should leaving the instance vulnerable to failures. In this section we briefly discuss the syntax to execute backup and restore operations with **ontape** and **ON-Bar**. Because external backup and restore operations only require using database server utilities to back up the logical logs, we will not discuss these operations.

This should not be considered a comprehensive overview because it does not contain any configuration guidance on setting tunables such as storage devices for **ontape**, memory and threads for **ON-Bar** API operations nor device, save set, expiration, media, and other parameters within the ISM.

9.2.1 Creating a backup

Backups of instances and logical logs are treated separately from a media management perspective requiring two sets of media that cannot be intermingled. It follows then that a discussion of how to execute a backup should differentiate these two processes.

The commands to backup an instance with `ontape` are very simple and straightforward. The `ontape` syntax tree is:

```
ontape -s -L [0/1/2] [-F]
```

where `-L` is followed by the desired backup level. If the `-L` flag is omitted, the `ontape` utility will prompt for the level. When this backup is created, the backup timestamp information is updated in the reserved pages of the rootdbs. In most cases, this is fine, but there might be a need to capture a copy of the instance without updating this information, such as when the backup media is being sent offsite and will never be available to restore from. In this case, a fake can be made by adding the `-F` flag. There is no difference in terms of what is output by `ontape` in a regular or fake backup.

There are additional flags to the `ontape` backup command that can be used to change the logging mode of one or more databases in the instance. This might be required to execute a series of very large maintenance transactions without the overhead of logging for example. The options are:

```
-A list  
-B list  
-N list  
-U list
```

where the databases included in *list* are converted from their current log state to mode ANSI, buffered, no log or unbuffered logging respectively. This change occurs after the backup operation has occurred so the backup will capture the current state of the instance not the newly changed state. If more than one database needs to have its logging state changed, they should be listed with a single space as a separator without any other punctuation.

There is one more option available for `ontape` but it, and the possibilities with its use, will be discussed in 9.3.1, “`Ontape to STDIO`” on page 274.

The backup syntax for an ON-Bar instance backup is a little more complex but not difficult. The ON-Bar syntax tree is:

```
onbar -b [-L 0/1/2] [-f filename / space(s)] [-0]
```

Or

```
onbar -b - w [-L 0/1/2] [-0]
```

Or

```
onbar -b -F
```

While the syntax looks somewhat similar to that of `ontape`, there are a number of subtle yet significant differences. The first is the backup level. It is not strictly required to use the `-L` followed by a level. If not used, the default is to back up all spaces and logs.

The next difference is fairly significant. As mentioned in 9.1.3, “ON-Bar utility suite” on page 262, ON-Bar operations can either be serialized like `ontape` or parallelized. This is controlled by the `-w` flag. If set, the backup and associated restore operation executes almost exactly as an `ontape` operation. As such, it has limited functionality, just the choice of a backup level.

If the `-w` flag is not used, the operation executes in parallel with as many output streams as requested by the storage management system. With this parallelism comes greater flexibility in what is backed up. The backup operation can be limited to a subset of the instance spaces with the `-f` flag followed either by a pathed filename containing the spaces (one to a line with no additional punctuation), or a simple list of the spaces separated by a single whitespace.

When an ON-Bar operation begins, the first thing that happens is that the status of each storage space is checked to make sure it is online and available to participate in the operation if needed. If one or more are not available, the operation will indicate the error and stop. This can be overridden with the `-0` flag.

Finally, a fake backup can be created with just the `-b` and `-F` (uppercase) flags. Unlike `ontape` though, nothing is output from the instance, overhead flags are set on newly added spaces or chunks and other objects that a backup has occurred and the objects should be marked available for use. This is actually a better choice to use than that provided by `ontape` because the `TAPEDEV` parameter does not need to be set to `/dev/null` to throw away the output. Obviously executing either type of a fake backup should only be executed when the administrator is positive a real backup is not needed.

In considering the backup of logical logs, there are only two choices—continually as they fill or as requested by the administrator. There are advantages and disadvantages to both approaches. Choosing continuous mode with `ontape`

requires a dedicated terminal/terminal window as well as a dedicated device. With ON-Bar and a management system, depending on the devices available, the number of other concurrent operations by the management system and the frequency with which the logs fill, either one or more devices will be busy satisfying log backup requests or these requests will be slowed in the backup job queue and not get written to tape as quickly as they should. Backing them up upon administrative request will reduce the demand for an available backup device but leaves the instance at greater risk of not being able to recover to as close to a failure condition as possible. The best balance is to use ON-Bar with continuous logical log backups but size the logs so that they fill every 45 to 60 minutes of normal daily activity. With this, the demand for backups is relatively light permitting other operations to occur during the day but if there are spikes in activity there is enough time to absorb them and still get the logs backed up in a reasonable amount of time.

The syntax to backup the logical logs through ontape is:

```
ontape [-a / -c]
```

with the **-a** flag to backup all filled logs in one operation and the **-c** flag for continuous backups. The ON-Bar syntax tree is similar to:

```
onbar -b -l [-c / -C / -s] [-0]
```

where the **-C** (uppercase) flag indicates a continuous operation while the **-c** (lowercase) flag only backs up the filled logs.

In the event of a catastrophic system failure involving the disks, it is possible the disk media containing the logical logs might still be available and readable. Prior to replacing the damaged hardware and starting a restore, it would be wise to salvage as much of the logical log information as possible from the media. This is accomplished with the **-s** flag to ON-Bar. The **-0** option overrides errors as when executing an instance backup operation.

9.2.2 Verifying backups

Until a few years ago, instance backups were a matter of faith. Administrators created them regularly but never knew if they would work to restore an instance. Unless a separate machine and storage environment was available and could be configured identically to the original and the restore tried there, verification came when it was time to use the backup for real. As you might imagine, finding out part way through restoration of a down instance or dbspace that the backup is corrupted is not a pleasant experience for the administrator or the business that is depending on that data. The archecker utility was developed to solve that problem. As discussed in 9.3.2, “Table Level Point-in-Time Restore (TLR)” on

page 275, new and exciting functionality has been added to archecker giving administrators even more restoration options.

Archecker was originally designed to do two things—read through media used for a backup to verify the control information and format of the backup and make a copy of the backup to another set of media for redundancy. While it could not verify that the actual user data recorded to tape was correct, the utility could verify that all the structures surrounding the user data were complete and the user data could be read and translated into valid data types for use inside a table. Reliance on backups could now move from faith to more comfort based on knowledge.

Using archecker is not difficult and only requires a minimum of setup. The AC_CONFIG environment variable needs to point to the archecker configuration file in \$INFORMIXDIR/etc. An example, called ac_config.std is available for copying and modification like the sqlhosts and onconfig default files. This configuration file only has a few parameters that determine whether the utility will execute in verbose mode, which directory will hold its temporary and log files and so on. Not a lot of disk space is used for these files, perhaps 50 MB maximum and, provided the check completes successfully, the files are removed at the end of the operation. If the verification fails, the files are left in place to diagnose the problem.

When the file and environment variable is set, the utility is invoked through an ON-Bar restore command. The syntax is:

```
onbar -v [-w] [-p] [-t "time"] [-f filename / spaces]
```

where the **-w** flag indicates the backup to verify was created serially. The **-t** flag followed by a time value in quoted *yyyy-mm-dd hh:mm:ss* format will verify data pages marked with that timestamp or earlier. If the backup to verify was created using parallel mode, verification can test a subset of the spaces backed up using the **-f** flag followed either by a pathed filename containing the spaces (one to a line with no additional punctuation), or a simple list of the spaces separated by a single whitespace.

The archecker verification process does have a few limitations, specifically with BLOBs. It cannot verify the 56 byte BLOB descriptors that join standard data rows and simple BLOBs stored in a simple BLOBspace. This can only be accomplished by executing an **oncheck -cD** command. With the more advanced smart BLOB technology, archecker can only verify the extents of the smart large objects and not that the data can be accurately reconstructed. Given the tremendous range of data types that can be stored in smart BLOBspaces, including user-defined types, this is not surprising. A more complete check of smart BLOBspace integrity can be done by executing an **oncheck -cS** command.

During the verification process, archecker verifies the following conditions:

- ▶ Data page corruption
- ▶ Corrupt control information
- ▶ Missing pages that have been added since the last level-0 backup
- ▶ Retrieval of the wrong backup objects

If a backup fails verification, do not attempt to restore from it. The results will be unpredictable and range from corruption of the database to a failed restore because ON-Bar cannot find the correct backup object on the storage manager. If a restore is attempted, it might appear to be successful but hide a multitude of data problems.

Resolving corruption issues found through verification depends on what was found.

- ▶ If the pages are corrupt, the problem is with the databases rather than with the backup or the media. Run **oncheck -cd** on any tables that produce errors and then redo the backup and validation. To check extents and reserved pages, run **oncheck -ce** and **oncheck -cr**.
- ▶ If the control information is corrupted, it could cause problems with the restore. Contact Informix Technical Support for assistance. Unless otherwise indicated, the data pages should be fine.
- ▶ If data is missing, it might not be recoverable. After a data loss, try to restore from an older backup and then restore the current logical logs.
- ▶ There are cases where archecker returns success to ON-Bar but shows failure in the archecker message logs. This situation occurs when archecker verifies that ON-Bar backed up the data correctly, but the database server data was invalid or inconsistent when it was backed up. A new backup should be created and verified. If this backup fails, the instance should be placed in single-user administrative mode with an **onmode -j** or **oninit -j** command to shut down all user activity and another backup created and verified.

9.2.3 Restoring from a backup

The difference in functionality between ontape and ON-Bar restore operations is similar to the differences in creating the backups used for restoration. Regardless of the utility used, there are three types of restore operations:

- ▶ Physical: Restores data from backups of all or selected storage spaces
- ▶ Logical: Restores transactions from logical log records
- ▶ Full: The automatic combination of a physical and logical restore

Each has its purpose depending on what needs to be done. A full restore is generally used to recover an instance to a desired moment in time with

transactional integrity. A physical restore operation restores data from a set of level 0, 1, and 2 backups (if created) and could be used to recover one or more spaces on a failed drive that had to be replaced. In order to bring the tables on that drive to consistency with the rest of the instance, a logical restore using logical log records would then be executed.

The ontape syntax tree for restore operations looks like:

```
ontape [-r / -p] [-D DBspace_list]
[-rename {-f filename / -p old_path -o old_offset -n new_path -o
new_offset...}]
```

The **-r** or **-p** flags indicate whether the restore will be full or physical only respectively. The **-D** flag is used to execute a warm restore of one or more spaces listed with a single whitespace separator.

A recent functional enhancement to both ontape and ON-Bar restore operations is the ability to redirect spaces during the restore. Previous to this feature, chunks had to be restored to the exact location they were in when backed up. In most cases, this was not a problem. Most administrators use symbolic links pointing to the real device when defining chunks. If a storage location needs to change as part of a restore operation, the symbolic link is recreated to point to the new location, the restore occurs and the instance was none the wiser. In some cases though, administrators use real device paths in defining chunks and have problems if the device is changed and either its parameters are different (the new drive had less usable space due to bad blocks) or the OS creates a different device path. They can no longer restore because either the device no longer exists or insufficient usable space is available to create all the chunks.

With the **-rename** flag, device paths to chunks (with their associated offsets) can be changed and data restored to the new locations. If only one or two chunks need to be removed, they can be listed in the command itself. For example:

```
ontape -r -rename -p /chunk1 -o 0 -n /chunk1N -o 20000
-rename -p /chunk2 -o 10000 -n /chunk2N -o 0
```

If many chunk paths need to be changed, the **-f** flag followed with the fully pathed location of a flat file containing all the parameters can be used. The file contents should be one set of chunk relocation information per line, separated by a single whitespace and no additional punctuation. For example:

```
/chunk1 0 /chunk1N 20000
/chunk2 10000 /chunk2N 0
```

To execute a logical restore, the syntax is very simple:

```
ontape -l
```

Regardless of the restore operation executed, a prompt to salvage existing logical log records will occur. This might or might not be necessary depending on the reasons causing the restore. Generally, it is good operating practice to salvage these logs by using a new tape. With this log information, recovery up to the point of the failure is possible. A manual logical log salvage operation can occur prior to a restoration attempt by executing an **ontape -S** command.

ON-Bar restore operations include full, physical-only, logical-only and relocated chunks as **ontape** but ON-Bar also supports imported and restartable restore operations. An imported operation is one in which a full level 0 backup is taken on one physical server and restored to another physical server to provide a redundant or test copy of the instance. Imported restores can also be used to initialize High Availability Data Replication (HDR) or for database server version upgrades. There are a number of restrictions on imported restores, specifically:

- ▶ The storage manager supports imported restores.
- ▶ Whole-system backups (with the **-w** flag) must include all storage spaces; logical logs are optional.
- ▶ Backups created in parallel must include all storage spaces and logical logs.
- ▶ The database server name can be changed in an imported restore.
- ▶ The version of the IDS database server is identical on both physical servers.
- ▶ The two physical servers have identical operating systems and hardware—particularly disks and paths to the disks. Chunk paths cannot be renamed in an imported restore.
- ▶ Both physical servers are on the same LAN or WAN.
- ▶ Both servers are using identical storage manager versions with compatible XBSA libraries.

The process of executing an imported restore involves copying the ON-Bar emergency boot file (**ixbar.num**), the **oncfg** files (**oncfg_servername.servernum**), instance ONCONFIG, and any storage management configuration files from the source server to the target server and modifying them as necessary for the new server. This would include renumbering the **ixbar** file, changing the **DBSERVERNAME** and **DBSERVERALIAS** parameters, changing the **oncfg** parameters to match the new instance name and **ixbar** number and any other changes required within the storage management files. When ready, an **onbar -r [-w]** command is executed on the target server to restore the data.

The restartable restore option is only available for cold or warm restores. If, during the restore operation, a failure occurs with the database server, media, or ON-Bar, the restore can be restarted from the place it failed. This assumes there is not a crippling error or corruption in the backup media. The ability to use ON-Bar's restartable restore functionality is controlled by the

RESTARTABLE_RESTORE \$ONCONFIG parameter. By default it is set to on. To restart a failed ON-Bar restore operation, simply execute an **onbar -RESTART** command.

To execute other ON-Bar restore operations, the syntax tree is:

```
onbar -r [-w] [-0] [-f filename / spaces] [-t timestamp / -n  
lognumber] [-rename {-f filename / -p old_path -o old_offset -n  
new_path -o new_offset...}]
```

The **-w** flag indicates the backup was created serially while the **-0** flag functions identically to the **ontape** flag. An ON-Bar restore can be restricted to a subset of the chunks backed up with the **-f** flag followed either by a list of dbspaces separated by a single whitespace or the fully pathed location of a flat file containing the dbspace names to be restored. The file contents should be one dbspace name per line with no additional punctuation.

Somewhat similar to **ontape**, an ON-Bar operation can restore to a particular moment in time using the **-t** flag. ON-Bar provides two options though for determining when to stop the restore operation—by a logical log number or a time value to the second in quoted *yyyy-mm--dd hh:mm:ss* format. ON-Bar also supports the ability to redirect chunks during a restore with the **-rename** flag. Its parameters are identical to those for **ontape**.

In executing a logical restore, ON-Bar supports optional functionality to stop the restore based on time or logical log number. If not used, all available logical log records will be restored. The syntax for an ON-Bar logical restore is:

```
onbar -r -l [-t timestamp / -n lognumber]
```

9.3 New functionality

As indicated earlier in this chapter, Informix developers have continued to enhance the IDS backup and recovery functionality to meet customer needs. Version 10 of the database server includes two exciting new features alluded to earlier. In this section we discuss these new features.

9.3.1 Ontape to STDIO

In introducing the **ontape** utility in 9.1.2, “Ontape” on page 261, reference was made to the fact that it is an administrative-driven utility requiring responses to prompts for backup and restore operations. Its functionality, while certainly adequate, is restricted to one or two locally connected devices and, if configured to disk, does not provide any intelligent file handling. With IDS V10, **ontape**

operations can now be directed to or from *standard in* or *standard out* (STDIO), providing support for a much broader range of options and functional uses.

Executing operations to STDIO can be driven two ways, the **TAPEDEV** or **LTAPEDEV** \$ONCONFIG parameters can be set to **STDIO** or the **-t STDIO** flag can be used in the command string to override the configured tape device for the operation. When an operation is executed using STDIO, there are no administrative prompts, the operation begins immediately.

Executing a backup operation to STDIO begins streaming data into the OS shared memory where, unless the intent is fill the system's memory, it should be redirected through an OS pipe to another utility or device. The stream could be directed through a compression utility then to disk / tape or directly to a disk file but with an intelligent name so file management is not needed. For example:

```
ontape -s -L 0 -t STDIO | tar cvf /ids_backups/042406_lev0.tar
```

Another very handy use for a backup to STDIO is to initialize an HDR secondary instance. The first step in creating an HDR secondary is to restore a full instance backup from the primary followed by a logical recovery of the logical logs also from the primary across the network connection. Until now, this required creating a backup to media on the primary then transferring the media to the mirror for restoration. Depending on the distance between the two servers and the media used (tape or disk file), there could be an overnight or longer delay between the backup and restore operations making initialization very difficult. With STDIO functionality, this can be done in real time with a OS-based pipe connection between the servers:

```
ontape -s -L 0 -F |rsh mirror_server "ontape -p"
```

9.3.2 Table Level Point-in-Time Restore (TLR)

One of the most sought after features in the Informix user community has been the ability to restore data from a single table instead of an entire dbspace. With IDS V10, the archecker utility functionality was enhanced to support a feature called the Table Level Point-in-Time Restore (TLR). While language usage experts cringed (points exist in geometric space, there are individual and discrete moments in time), IDS administrators cheered because the new functionality provided the ability to filter the data restored and even restore data to an instance supported by a server running a different operating system!

What happens in the software is not complicated. Either an ontape or ON-Bar level 0 backup stream is read by the archecker utility. The utility uses a schema file created by the administrator to determine the data to be extracted. The extracted data is converted to ASCII then inserted into the target specified in the schema file. It is this conversion to ASCII which enables a TLR operation to pull

data from a AIX-based instance and insert it into a Linux-based instance for example.

As might be expected, the syntax tree for a TLR is pretty simple:

```
archecker -X [-b / -t] [-f schemafile] [-d / -D] [-v] [-s] [-l phys  
/ stage / apply]
```

The **-b** and **-t** flags indicate whether the stream to be processed was created by ON-Bar or ontape respectively. The TLR schema file determining the actions to take can be defined in one of two places, with the **AC_SCHEMA** parameter in the **ac_config** file or through the **-f** flag followed by the fully pathed location of the schema file. This flag can also be used to override any value set in the **ac_config** file.

When archecker performs a TLR, a set of working tables are created in the sysutils database as well as some working flat files on disk. With the **-d** flag, archecker first removes any existing TLR restore files (except for the TLR log) before proceeding with the restore operation. The **-D** option removes TLR files (except for the log) plus any working tables in the sysutils database then the utility exits.

The **-v** and **-s** flags determine how verbose the utility will be while it works. The **-v** flag turns verbose “on” and **-s** turns on status messages to the session window through which the utility was invoked.

A TLR operation can perform a physical-only, logical-only or full restore. The default is a full restore but the **-l** flag can be used to restrict the operation to a physical restore or one or more parts of a logical restore. Any or all of the three options can be specified if separated by a comma and no other space. For example, to specify a full restore with this flag the command would include **-l phys,stage,apply**. The three options do the following:

phys	Starts a restore of the system, but stops after physical recovery is complete
stage	After physical recovery is complete, extracts the logical logs from the storage manager, stages them in their corresponding tables and starts the stager
apply	Starts the log applier. The applier takes the transactions stored in the stage tables and converts them to SQL and replays the operations to bring the table(s) data to a more current moment of consistency

With this granularity, a physical restore can be done and users can get at base data sooner. Later, during a maintenance period or less busy time, the log records can be extracted or applied to bring the data as current as desired so it

can be put back where it originated from. Because the intent of the TLR is to recovery dropped tables, in replaying the logical logs the TLR operation ignores the **drop table** SQL command which created the need to execute a TLR in the first place.

The schema command file

The parameters for a TLR operation are contained in the schema file defined in **ac_config** or by the **-f** flag. The file specifies the source tables, destination tables, table schemas, databases, external tables if any, the moment in time the table is to be restored to as well as other options. Standard SQL syntax is used to define the various options with standard SQL comment syntax included for documentation purposes as well. Comments are ignored during the execution of a TLR operation.

Opening a database

This sets the current database for the operation and uses the following syntax:

```
database name [mode ANSI]
```

Multiple **database** statements can be used within a file though all table names referenced following this statement are associated with the current database. With this syntax, data can be restored from one database to a table in another database as shown in one of the example sets in Example 9-2.

Example 9-2 The database syntax as used in a TLR command file

```
SET #1
database my_database;
create table source (...);
create table target (...);
insert into target select * from source;

SET #2
database local_db;
create table source (...) in dbspace1;

database remote_db;
create table target (...) in dbspace2;

insert into target select * from local_db:source;
```

Creating tables

Source and destination tables are defined with the **create table** statement. Archecker supports the full syntax of this command, including the **external** key word to create an external table with its parameters. The schema for the source

table in this file must be identical to the schema of the source table at the time the backup was created; if not, unpredictable results will occur. The target table can include all or a subset of the source table attributes as well as other attributes. A TLR operation can even be used to create a new table which has never existed in the target database and populate this new table with data from one or more existing tables.

The source table cannot be a synonym or view. The schema of the source table only needs the column list and storage options. Other attributes such as extent sizes, lock modes, and so on are ignored. For an ON-Bar backup, archecker uses the list of storage spaces for the source table to create its list of objects to retrieve from the storage manager. If the source table is fragmented, all dbspaces containing table data must be listed. The archecker utility only extracts data from the dbspaces listed in the schema command file.

If the source table contains constraints, indexes or triggers, they are disabled during the TLR operation. After the TLR operation completes, the constraints, indexes, and triggers are enabled.

The schema of the target table is also created in the command file. If the target table does not exist at the time the restore is performed, it is created using the schema provided. If the target table already exists, its schema must match the schema specified in the command file. Data is then appended to the existing table.

Several sets of table definition statements are included in Example 9-3.

Example 9-3 Multiple examples of the create table syntax in a TLR command file

SET #1

```
create table source (col1 integer, ...) in dbspace1;
```

```
create table target (col1 integer, ...)
  fragment by expression
  mod(col1, 3) = 0 in dbspace3),
  mod(col1, 3) = 1 in dbspace4),
  mod(col1, 3) = 2 in dbspace5);
insert into target select * from source;
```

SET #2

```
create table source
  (col1 integer,
   col2 integer,
   col3 integer,
   col4 integer);
```



```

create table target
  (col1 integer,
   col2 integer);
insert into target (col1, col2) select (col3, col4) from source;

SET #3
create table source_a
  ( columns ) in dbspace1;

create table target_a ( columns );

create table source_b
  ( columns ) in dbspace1;

create table target_b ( columns );

insert into target_a select * from source_a;
insert into target_b select * from source_b;

```

In Set 1, the full table was restored but the fragmentation scheme was changed while in Set 2, only a few of the source table attributes were restored. In the last set, several tables were restored in one operation.

When a TLR operation includes a logical restore, two additional work columns and an index are added to the destination table. These columns contain the original rowid and original part number for each restored row. These columns provide a unique key which identifies the location of the row as recorded in the original source backup. Controlling where this index is stored can be modified with the **set workspace** option discussed in the “Additional options” section on page 280. The default behavior is to store the index in the same space as the table.

Inserting data

As shown in Example 9-3, the standard SQL **insert** syntax is used to select and insert data during a TLR operation. Archecker supports many of the filtering options in the **select** and **insert** portions of the command. However, archecker does not support:

- ▶ aggregates (such as **sum** and **mod**)
- ▶ function and procedure calls
- ▶ subscripts
- ▶ subqueries
- ▶ views
- ▶ joins within the **select** operation

Filters are only be applied during physical restore portion of the TLR operation.

Specifying a moment in time

An optional command, the **restore to** option, can be used to indicate the moment in time the tables specified in the command file should be restored to. The default is to restore to as current as possible from the logical logs. The syntax for using this functionality is:

```
restore to [current / timestamp] [with no log]
```

Only one **restore to** statement can be specified in a command file. If the **with no log** option is used, only a physical restore is performed. In addition, the two extra columns and the index used during the logical restore are not added to the destination table.

Additional options

The **set** keyword is used to determine how many rows are processed before a **commit work** SQL command is automatically executed as well as the dbspace to be used for TLR objects such as the extra index created on tables during the logical restore portion of the operation. The syntax is:

```
set [commit to number] [workspace to dbspace(s)]
```

If these options are not set, the default is to commit every 1000 rows and to build the objects in the rootdbs. Given the critical nature of the rootdbs, it is not good operating practice to fill it with indexes and other objects. As such it should be considered mandatory to define one or more working spaces with this option. Only standard dbspaces can be used for TLR workspaces. More than one can be listed provided they are comma separated as shown in Example 9-4.

Example 9-4 Setting TLR options

```
set commit to 20000;  
set workspace to dbspace1;  
  
set workspace to dbspace1,dbspace4
```

Really easy administration

The longevity of any software tool or object directly depends on the way it is used, because maintenance is the key factor in the determination of the health of the tool. Databases and application software are similar and also follow the same pattern. With the passing of time, if the databases are not serviced or maintained properly the robustness and the performance can start to deteriorate. This is especially true for OLTP applications where the data and the environment dynamics are frequently changing.

To keep pace with the dynamics, the software providers must continue to deliver new features and tools. Thereafter it becomes the users responsibility to make sure that the tools and features are implemented and fine tuned to best meet their business requirements.

In this chapter we discuss a number of tools and enhancements which have been introduced in IDS V10 to ease the administrative work in maintaining databases.

10.1 Flexible fragmentation strategies

IBM Informix Dynamic Server (IDS) V10 provides a new fragmentation strategy to enable you to manage large number of dbspaces. In this section, we highlight some of the fragmentation strategies.

10.1.1 Introduction to fragmentation

Consider an application where the data fragmentation is based on a date expression, with a separate fragment for sales for each day. Then, daily you will need to add a new dbspace for the date expression. If the application requires a large number of ranges in different fragments, you will have to administer multiple dbspaces. This dbspace management, including dbspace creation and deletion, adds to the tasks required, typically of the DBA. The fragmentation strategy of releases prior to IDS V10 do not allow tables to be fragmented with multiple fragments in one dbspace, and so each dataset range needed to be a separate dbspace.

Example 10-1 shows a customer table where fragmentation by expression is based on the STATE column. Each table fragment is associated with separate dbspace, so when a new state needs to be added, you must add a new dbspace.

Example 10-1 Fragment by expression

```
CREATE TABLE customer
(
    id      INT,
    state   CHAR(2)
)
FRAGMENT BY EXPRESSION
    (state = "AZ") IN dbspace1,
    (state = "CA") IN dbspace2,
    (state = "WA") IN dbspace3,
    (state = "NY") IN dbspace4,
    REMAINDER      IN dbspace5;
```

By enabling multiple fragments in a single dbspace, you can manage a large number of data fragments without the maintenance effort of finding for free space in the file system and selecting a new name for the dbspace. You can simply add or attach a new expression and specifies the new fragment name.

Fragmentation Background

IDS V10 supports table and index fragmentation (also called partitioning) which allows you to store a table on multiple disk devices. A proper fragmentation strategy can significantly reduce the I/O contention and increase manageability.

The Informix fragmentation strategy consists of two parts:

- ▶ A distribution scheme that specifies how to group rows into fragments. You specify the distribution scheme in the `FRAGMENT BY` clause of the `CREATE TABLE`, `CREATE INDEX`, or `ALTER FRAGMENT` statements.
- ▶ The set of dbspaces in which you locate the fragments are specified by the `IN` clause (storage option) of these SQL statements.

A fragmentation strategy can be built on a table, an index or both. Fragmentation distribution schemes can be either expression based or round robin based:

- ▶ Expression based fragmentation enables distributing the rows into multiple fragments based on a fragment expression (for example, `state = "AZ"`) as shown in Example 10-1. Each fragment expression isolates rows and aids in narrowing the search space for queries. You can define range rules or arbitrary rules that indicate to the database server how rows are to be distributed.
- ▶ Round-robin fragmentation strategy distributes the rows so the number of rows in each fragment remains approximately the same.

For a common understanding, here are a few definitions of terms used:

- ▶ A table fragment (partition) refers to zero or more rows that are grouped together and stored in a dbspace that you specify when you create the fragment. Each table fragment has its own tablespace with a unique `tblspace_id` or `fragment_id`.
- ▶ A dbspace includes one or more chunks. You typically monitor the dbspace usage and add chunks as necessary.
- ▶ A chunk is a contiguous section of disk space available for a database server.

10.1.2 Fragmentation strategies

Pre IDS V10 releases, for example IDS V9.4, supported two strategies, fragment by expression and fragment by round robin. Although this is good, it requires each table fragment to be stored in distinct dbspaces. One dbspace can contain fragments from multiple tables, but a single table cannot have more than one fragment in a single dbspace. This can result in manageability overhead, requiring the creation and monitoring of a large number of dbspaces. In addition, there is a limitation on the number of chunks per dbspace, and a fixed page size for all dbspaces.

All of the above mentioned limitations are addressed in IDS V10 by using multiple fragments in a single dbspace, large chunk and non-default page size support.

In IDS V10, you can consolidate tables or indexes on multiple dbspaces into a single dbspace. New tables and indexes can be created with one or more fragments from one or more dbspaces. And, you can manage a large number of table or index fragments with a manageable number of dbspaces.

Also in IDS V10, each fragment distribution scheme is associated with a partition. So the existing fragmentation strategy can easily be converted into a multiple fragment strategy by using partition syntax in the alter fragment command. Fragmented tables or indexes can be created using old and new fragmentation syntax. And, the multiple fragment strategy does not impact PDQ. Parallel threads are executed the same as with the old fragmentation strategy, and the old fragmentation strategy is still supported in IDS V10.

New fragment strategy

In the new schema, as shown in Example 10-2, the table uses a single dbspace but keeps original fragment expression on state column. The partitions in the new fragmentation strategy are az_part, ca_part, wa_part and ny_part.

Example 10-2 Expression based fragments in a single dbspace

```
CREATE TABLE customer
(
    id      INT,
    state   CHAR(2)
)
FRAGMENT BY EXPRESSION
    PARTITION az_part (state = "AZ") IN dbspace1,
    PARTITION ca_part (state = "CA") IN dbspace1,
    PARTITION wa_part (state = "WA") IN dbspace1,
    PARTITION ny_part (state = "NY") IN dbspace1,
    PARTITION remainder_part REMAINDER IN dbspace1;
```

Figure 10-1 shows a sample data representation for Example 10-2.

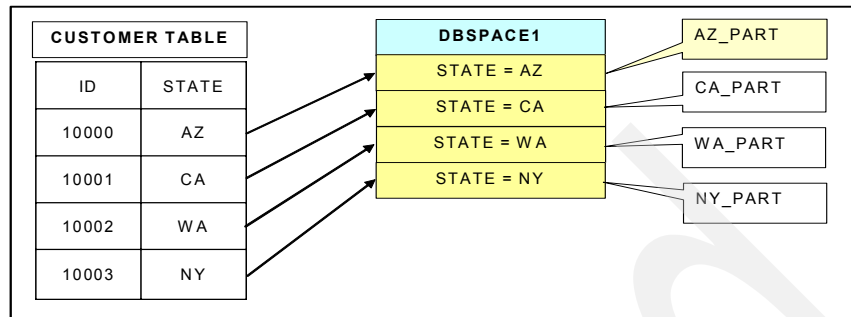


Figure 10-1 Expression based fragments in single dbspace

10.1.3 Table and index creation

A new table or an index can be created with multiple fragments in single dbspace. The partition keyword has been introduced to define the fragment strategy.

Example 10-3 illustrates the index created on the customer table using a similar fragment strategy as with the table definition.

Example 10-3 Creating Index with fragmentation strategy

```
CREATE TABLE customer
(
    id      INT,
    state   CHAR(2)
)
FRAGMENT BY EXPRESSION
PARTITION az_part (state = "AZ") IN dbspace1,
PARTITION ca_part (state = "CA") IN dbspace1,
PARTITION wa_part (state = "WA") IN dbspace1,
PARTITION ny_part (state = "NY") IN dbspace1,
REMAINDER IN dbspace1;

CREATE INDEX state_ind ON customer (state)
FRAGMENT BY EXPRESSION
PARTITION az_part (state = "AZ") IN dbspace2,
PARTITION ca_part (state = "CA") IN dbspace2,
PARTITION wa_part (state = "WA") IN dbspace2,
PARTITION ny_part (state = "NY") IN dbspace2,
REMAINDER IN dbspace2;
```

The round-robin fragmentation method can be applied to a table using a single dbspace fragment strategy. We illustrated that using the customer table in Example 10-4.

Example 10-4 Round-robin fragmentation

```
CREATE TABLE customer
(
    id      INT,
    state   CHAR(2)
)
FRAGMENT BY ROUND ROBIN
    PARTITION az_part IN dbspace1,
    PARTITION ca_part IN dbspace1,
    PARTITION wa_part IN dbspace1,
    PARTITION ny_part IN dbspace1;
```

You can select a fragment expression to be stored in a single dbspace, based on the requirements. Example 10-5 illustrates how you can combine the old and new fragment strategies for a table or index creation.

Example 10-5 Mixed fragmentation strategy

```
CREATE TABLE customer
(
    id      INT,
    state   CHAR(2)
)
FRAGMENT BY EXPRESSION
    PARTITION az_part (state = "AZ") IN dbspace1,
    PARTITION ca_part (state = "CA") IN dbspace1,
    (state = "WA") IN dbspace2,
    (state = "NY") IN dbspace3,
    REMAINDER IN dbspace4;
```

NEW METHOD

OLD METHOD


```
CREATE INDEX state_ind ON customer (state)
FRAGMENT BY EXPRESSION
    PARTITION az_part (state = "AZ") IN dbspace2,
    PARTITION ca_part (state = "CA") IN dbspace2,
    PARTITION wa_part (state = "WA") IN dbspace2,
    (state = "NY") IN dbspace3,
    REMAINDER IN dbspace4;
```

NEW METHOD

OLD METHOD

The server also supports `PARTITION BY EXPRESSION` instead of `FRAGMENT BY EXPRESSION` and `PARTITION BY ROUND ROBIN` instead of `FRAGMENT BY ROUND ROBIN` in all statements with single dbspace fragment strategy as shown in Example 10-6.

Example 10-6 Using PARTITION BY

```
CREATE TABLE customer
(
    id    INT,
    state CHAR (2)
)
PARTITION BY EXPRESSION
PARTITION az_part (state = "AZ") IN dbspace1,
PARTITION ca_part (state = "CA") IN dbspace1,
PARTITION wa_part (state = "WA") IN dbspace1,
PARTITION ny_part (state = "NY") IN dbspace1,
REMAINDER IN dbspace1;
```

```
CREATE TABLE customer
(
    id    INT,
    state CHAR (2)
)
PARTITION BY ROUND ROBIN
PARTITION az_part IN dbspace1,
PARTITION ca_part IN dbspace2,
PARTITION wa_part IN dbspace3,
PARTITION ny_part IN dbspace4;
```

10.1.4 Alter fragment examples

You can easily update existing fragmentation methods using the `alter fragment` command. These modification can be applied to both table and index fragmentation.

Alter fragment on table

To modify existing fragment strategies, use the alter fragment options. These options can help to create mixed fragment strategies based on requirements. Example 10-7 shows the use of all table specific options, which are add, drop, attach, detach, modify, and init.

Example 10-7 Alter fragment on table

```
CREATE TABLE customer
(
    id      INT,
    state CHAR(2)
)
FRAGMENT BY EXPRESSION
    PARTITION az_part state = "AZ" IN dbspace1,
    PARTITION ca_part state = "CA" IN dbspace1,
    PARTITION wa_part state = "WA" IN dbspace1,
    PARTITION ny_part state = "NY" IN dbspace1,
    REMAINDER IN dbspace2;

ALTER FRAGMENT ON TABLE customer
ADD PARTITION part_or (state = "OR")
IN dbspace1 BEFORE ca_part;

ALTER FRAGMENT ON TABLE customer
DROP PARTITION part_or;

ALTER FRAGMENT ON TABLE customer
ATTACH customer_or
AS PARTITION part_3 (state = "OR");

ALTER FRAGMENT ON TABLE customer
DETACH PARTITION part_3 customer_or;

ALTER FRAGMENT ON TABLE customer
MODIFY PARTITION az_part
TO PARTITION part_az (state = "AZ")
IN dbspace2;

ALTER FRAGMENT ON TABLE customer
INIT FRAGMENT BY EXPRESSION
    PARTITION az_part (state = "AZ") IN dbspace2,
    PARTITION ca_part (state = "CA") IN dbspace2,
    PARTITION wa_part (state = "WA") IN dbspace3,
    PARTITION ny_part (state = "NY") IN dbspace3,
    REMAINDER IN dbspace3;
```

The partition syntax can be used in a round robin fragmentation, as shown in Example 10-8.

Example 10-8 Partition in round-robin fragmentation

```
ALTER FRAGMENT ON TABLE customer
ADD PARTITION part_or IN dbspace1;

ALTER FRAGMENT ON TABLE customer
DROP PARTITION part_or;

ALTER FRAGMENT ON TABLE customer
ATTACH customer_or AS partition part_3;

ALTER FRAGMENT ON TABLE customer
DETACH PARTITION part_3 customer_or;

ALTER FRAGMENT ON TABLE customer
INIT FRAGMENT BY ROUND ROBIN
    PARTITION az_part IN dbspace2,
    PARTITION ca_part IN dbspace2,
    PARTITION wa_part IN dbspace3,
    PARTITION ny_part IN dbspace3;
```

Alter fragment on index

Alter fragment on index cases are similar to table syntax. Example 10-9 illustrates the use of alter fragment.

Example 10-9 Alter fragment on index

```
CREATE INDEX state_ind ON customer (state)
FRAGMENT BY EXPRESSION
    PARTITION az_part state = "AZ" IN dbspace2,
    PARTITION ca_part state = "CA" IN dbspace2,
    PARTITION wa_part state = "WA" IN dbspace2,
    PARTITION ny_part state = "NY" IN dbspace2,
    REMAINDER IN dbspace3;

ALTER FRAGMENT ON INDEX state_ind
ADD PARTITION part_or (state = "OR")
IN dbspace2
BEFORE ca_part;

ALTER FRAGMENT ON INDEX state_ind
DROP PARTITION part_or;
```

```

ALTER FRAGMENT ON INDEX state_ind
MODIFY PARTITION az_part
TO PARTITION part_az (state = "AZ")
IN dbspace3;

ALTER FRAGMENT ON INDEX state_ind
INIT FRAGMENT BY EXPRESSION
PARTITION az_part (state = "AZ") IN dbspace2,
PARTITION ca_part (state = "CA") IN dbspace2,
PARTITION wa_part (state = "WA") IN dbspace3,
PARTITION ny_part (state = "NY") IN dbspace3,
REMAINDER IN dbspace3;

```

10.1.5 System catalog information for fragments

System catalog, `sysfragments`, has a partition column `varchar(128,0)` which provides the partition name for a given fragment. If you create a fragmented table with partitions, each row in the `sysfragments` system catalog contains a partition name in the partition column. If you create a fragmented table without partitions, the name of the `dbspace` appears in the partition column. The server will continue to use the `dbspace varchar(128,0)` field for storing `dbspace` names. Example 10-10 shows the `sysfragments` catalog entry for `STATE = "AZ"` fragment expression.

Example 10-10 Sysfragments catalog entry

```
> SELECT * FROM sysfragments WHERE partition ="az_part";
```

```

fragtype      T
tabid         102
indexname
colno         0
partn         2097159
strategy      E
location      L
servername
evalpos       0
exprtext      (state = 'AZ' )
exprbin       <BYTE value>
exprarr       <BYTE value>
flags         0
dbspace       dbspace1
levels        0

```

```
npused      0
nrows       0
clust        0
partition    az_part
```

```
1 row(s) retrieved.
```

10.1.6 SQEXPLAIN output

SQEXPLAIN output provides new syntax changes, as shown in Example 10-11. All the fragment numbers in the output reference correspond to partitions from a single dbspace.

For example, Fragment 0 refers to the `part_1` partition from `dbspace1`. The fragment expression for fragment 0 can be obtained by using the `dbschema` or `sysfragments` catalog. IDS does not change the query plan for the PDQ environment, as described in 10.1.1, “Introduction to fragmentation” on page 282. The query plans in a PDQ environment will have similar output along with secondary thread details.

Example 10-11 Information in SQEXPLAIN output

```
CREATE TABLE t1
(
  c1 INT ,
  c2 CHAR(20)
)
FRAGMENT BY EXPRESSION
  PARTITION part_1 (c1 = 10) IN dbspacel,
  PARTITION part_2 (c1 = 20) IN dbspacel,
  PARTITION part_3 (c1 = 30) IN dbspacel,
  PARTITION part_4 (c1 = 40) IN dbspacel,
  PARTITION part_5 (c1 = 50) IN dbspacel;
```

QUERY:

```
SELECT COUNT(*) FROM t1 WHERE c1 = 10
```

Estimated Cost: 1

Estimated # of Rows Returned: 1

1) informix.t1: SEQUENTIAL SCAN (Serial, fragments: 0)

Filters: informix.t1.c1 = 10

QUERY:

```
SELECT * FROM t1 WHERE c1 > 30
```

Estimated Cost: 2

Estimated # of Rows Returned: 26

1) informix.t1: SEQUENTIAL SCAN (Serial, fragments: 3, 4)

Filters: informix.t1.c1 > 30

10.1.7 Applying new fragment methods after database conversion

You can use new fragment strategy after a database conversion is complete. During the conversion process the dbspace column from sysfragments is copied to a partition column.

The Alter Fragment table (or Index) will convert the old fragment strategy to a new one on the existing tables and index, as shown in Example 10-12.

Example 10-12 Table schema on old database before conversion process

```
CREATE TABLE customer
(
    id    INT,
    state CHAR (2)
)
FRAGMENT BY EXPRESSION
(state = "AZ") IN dbspace1,
(state = "CA") IN dbspace2,
(state = "WA") IN dbspace3,
(state = "NY") IN dbspace4,
REMAINDER IN dbspace5;
```

Apply the alter fragment table statement to use a new fragment strategy after database conversion. Both the init and modify options will alter the fragment strategy in this case, as shown in Example 10-13.

Example 10-13 Applying new fragment strategy after conversion

```
ALTER FRAGMENT ON TABLE customer
MODIFY
dbspace1 TO PARTITION az_part (state = "AZ") IN dbspace1,
dbspace2 TO PARTITION ca_part (state = "CA") IN dbspace1,
dbspace3 TO PARTITION wa_part (state = "WA") IN dbspace1,
```

```
dbspace4 TO PARTITION ny_part (state = "NY") IN dbspace1,  
dbspace5 TO REMAINDER IN dbspace1;
```

```
ALTER FRAGMENT ON TABLE customer  
INIT FRAGMENT BY EXPRESSION  
PARTITION az_part (state = "AZ") IN dbspace1,  
PARTITION ca_part (state = "CA") IN dbspace1,  
PARTITION wa_part (state = "WA") IN dbspace1,  
PARTITION ny_part (state = "NY") IN dbspace1,  
REMAINDER IN dbspace1;
```

You can verify the new fragment strategy from the `partition` and `exprtext` columns of `sysfragments`, as shown in Example 10-14.

Example 10-14 Fragment strategy from system tables

```
> SELECT partition, exprtext FROM sysfragments;
```

```
partition  az_part  
exprtext   (state = 'AZ' )
```

```
partition  ca_part  
exprtext   (state = 'CA' )
```

```
partition  wa_part  
exprtext   (state = 'WA' )
```

```
partition  ny_part  
exprtext   (state = 'NY' )
```

```
partition  dbspace1  
exprtext   remainder
```

```
5 row(s) retrieved.
```

A similar process can be followed for an index fragment strategy.

10.1.8 Oncheck utility output

Details about the new fragment method are provided in the `oncheck` utility. An additional partition name has been added to all of the `oncheck` options. The new `oncheck` output includes Table fragment partition `<partition name>` in `DBspace <dbspace name>` format.

When the new fragment method is not used, oncheck displays <dbspace name> for table or index fragment partition.

Oncheck output for the **-cD**, **-cd**, **-cI**, **-ci**, **-pD**, **-pd**, **-pT**, and **-pt** options shows partition <partition name> along with dbspace name, as shown in Example 10-15.

Example 10-15 Table schema

```
CREATE TABLE customer
(
    id INT,
    state CHAR (2)
)
FRAGMENT BY EXPRESSION
PARTITION az_part (state = "AZ") IN dbspace1,
PARTITION ca_part (state = "CA") IN dbspace1,
PARTITION wa_part (state = "WA") IN dbspace1,
PARTITION ny_part (state = "NY") IN dbspace1,
REMAINDER IN dbspace1;
```

For a table schema with the new fragment syntax as shown in Example 10-15, **oncheck -cD testdb:customer** shows output as shown in Example 10-16.

Example 10-16 oncheck -cD output

```
TBLspace data check for testdb:informix.customer

Table fragment partition az_part in DBspace dbspace1
Table fragment partition ca_part in DBspace dbspace1
Table fragment partition wa_part in DBspace dbspace1
Table fragment partition ny_part in DBspace dbspace1
Table fragment partition dbspace1 in DBspace dbspace1
```

Example 10-17 shows the index specific options for oncheck, which displays similar output.

Example 10-17 Index schema

```
CREATE INDEX state_ind ON customer (state)
FRAGMENT BY EXPRESSION
PARTITION az_part (state = "AZ") IN dbspace2,
PARTITION ca_part (state = "CA") IN dbspace2,
PARTITION wa_part (state = "WA") IN dbspace2,
PARTITION ny_part (state = "NY") IN dbspace2,
REMAINDER IN dbspace2;
```

For an index schema with the new fragment syntax, the **oncheck -cI testdb:customer** output is shown Example 10-18.

Example 10-18 The oncheck -cl output

```
Validating indexes for testdb:informix.customer...
  Index state_ind
    Index fragment partition az_part in DBspace dbspace2
    Index fragment partition ca_part in DBspace dbspace2
    Index fragment partition wa_part in DBspace dbspace2
    Index fragment partition ny_part in DBspace dbspace2
    Index fragment partition dbspace2 in DBspace dbspace2
```

10.1.9 Fragmentation strategy guidelines

Before trying to determine the best fragmentation strategy for your organization, a close consideration should be given to the following features of the IDS server:

- ▶ **PDQ:** Delivers maximum performance benefits when the data queried in fragmented tables. Each decision-support query has a primary thread. The database server can start additional threads to perform tasks for the query (for example, scans and sorts). Depending on the number of tables or fragments that a query must search and the resources that are available for a decision-support query, the database server assigns different components of a query to different threads.
- ▶ **Disk I/O:** Configuration of dbspaces should be determined based on available disk arrays. A resource is said to be critical to performance when it becomes overused or when its utilization is disproportionate. You can place a table with high I/O activity on a dedicated disk device and thus reduce the contention. When disk drives have different performance levels, you can put the tables with the highest use on the fastest drives. To isolate a high used table on its own disk device, assign the device to a chunk, assign that chunk to a dbspace, and then place the table in the dbspace that you created.
- ▶ **Large Chunk:** IDS can hold large number of partitions for the given dbspace by using Large Chunk support. Large Chunk support can be used for increasing number of data fragment expressions in a single dbspace. The size of chunks for dbspaces is 4 terabytes for a 2-kilobyte page. Chunks can reside anywhere in a 64-bit address space.

The **onmode -BC** (backward-compatible) commands are useful if you have converted from Dynamic Server 9.40 (small chunk mode) to IDS V10.

The **onmode -BC 1** command enables support of large chunks and large offsets that are greater than 2 GB, and allows more than 2047 chunks per dbspace. The **onmode -BC 2** command allows large-chunk-only mode for all

dbspaces. When IDS V10 is first initialized (with the **oninit -iyv** command), by default it comes online with large chunk mode already fully enabled.

- ▶ Non default page size: The root dbspace is the default page size – 4 KB on Windows and AIX, and 2 K on others. The dbspace created must use a multiple of the default pagesize and cannot exceed 16 KB.

You can specify a page size for a standard or temporary dbspace.

Performance advantages of a larger page size include:

- Reduced depth of b-tree indexes, even for smaller index keys.
- Decreased checkpoint time, which typically occurs with larger page sizes.

Additional performance advantages occur because you can:

- Group on the same page long rows that currently span multiple pages of the default page size.
 - Define a different page size for temporary tables, so the temporary tables have a separate buffer pool.
- ▶ Space and page issues: When you plan a fragmentation strategy, be aware of these space and page issues:
 - Although a 4-terabyte chunk can be on a 2-kilobyte page, only 32 gigabytes can be utilized in a dbspace because of a row ID format limitation.
 - For a fragmented table, all fragments must use the same page size.
 - For a fragmented index, all fragments must use the same page size.
 - Dbspaces used for table and indices on it can use different page size.

Using multiple fragments in single dbspace, you can easily monitor large number of table/index fragments in the given application thus reducing the administrative effort and time.

10.2 Shared memory management

One of the key components that impacts the performance of the IDS server is the way the shared memory and the disk space are arranged, configured and managed. In this section we provide a snapshot of the various parameters and attributes that need to be configured for best performance. Some of these configuration changes might be dependent on the OS, and those have been noted separately.

Refer to *IBM Informix Dynamic Server Administrator's Reference*, G251-2268, for more in depth coverage of each parameter

10.2.1 Database shared memory

Shared memory is an operating system feature that allows the database server threads and processes to share data by sharing access to pools of memory. Some of the main advantages and usefulness of these are:

- ▶ **Memory:** Reduces overall memory usage by letting the virtual processes and utilities access shared data instead of keeping their own private copies.
- ▶ **Disk I/O:** Reduces disk I/O and execution time by storing the most frequently used data in the cache memory common pool, thus reducing the disk I/O during data access.
- ▶ **IPC:** Allows an efficient and robust Inter-Process Communication (IPC) between the virtual processes, enabling the message read/write to happen at the speed of memory.

Figure 10-2 illustrates an generic view of an IDS memory scheme.

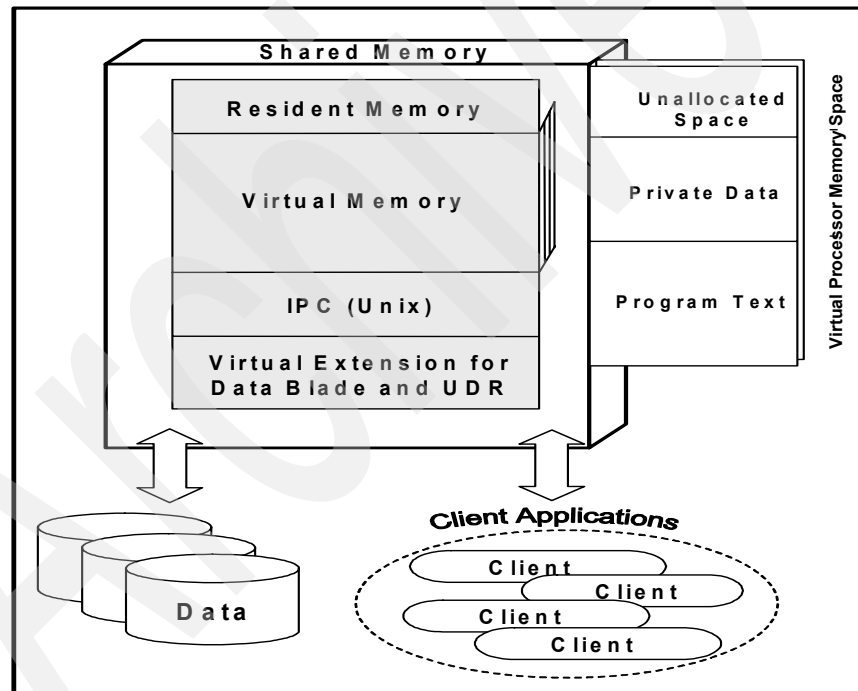


Figure 10-2 Shared memory scheme for IDS

10.2.2 Managing shared memory

One of the factors that determine how efficient the database server functions is the way the shared memory is configured and managed. Managing a shared memory includes the following tasks:

- ▶ Setting up shared memory and changing shared-memory configuration parameter values.
- ▶ Using the SQL statement cache to reduce memory and time for queries.
- ▶ Changing forced residency.
- ▶ Adding segments to the virtual portion of shared memory.
- ▶ Monitoring shared memory.

In the following sections, we discuss the shared memory management issues in more detail.

Setting the shared memory configuration parameters

Setting up a shared memory for a system involves setting it at the operating system level as well as the database level.

Operating system shared memory

Setting up the shared memory configuration parameters differs from one operating system to other. The different UNIX operating systems manage the same shared memory configuration through its on unique proprietary mechanism. However, irrespective of the operating system, you might need to tune the following functional parameters:

- ▶ Maximum operating system shared-memory segment size
- ▶ Minimum shared memory segment size, expressed in bytes
- ▶ Maximum number of shared memory identifiers
- ▶ Lower-boundary address for shared memory
- ▶ Maximum number of attached shared memory segments per process
- ▶ Maximum amount of system wide shared memory
- ▶ Maximum number of semaphore identifiers (UNIX)
- ▶ Maximum number of semaphores (UNIX)
- ▶ Maximum number of semaphores per identifier (UNIX)

Example 10-19 shows a sample Solaris operating system shared memory configuration parameter from /etc/system on a IDS V10 test database server.

Example 10-19 Solaris 5.8 /etc/system entry

```
* Set shared memory
set shmsys:shminfo_shmmax=2048000000
set shmsys:shminfo_shmmin=128
```

```
set shmsys:shminfo_shmmni=500
set shmsys:shminfo_shmseg=64
```

```
* Set Semaphores
set semsys:seminfo_semmni=4096
set semsys:seminfo_semmns=4096
set semsys:seminfo_semmnu=4096
set semsys:seminfo_semume=64
set semsys:seminfo_semmap=256
```

Refer to the specific operating system manual for further information.

Database server system shared memory configuration parameters

These parameters can be generally classified into 3 broad categories, which can be set through an editor or the ON-Monitor utility. These parameters are usually set in the ONCONFIG database configuration file:

► Resident shared memory parameters

These parameters effect the resident portion of the memory buffer pool. The database server must be shutdown and restarted for the parameters to take effect.

Table 10-1 Resident shared memory configuration parameters

Parameter	Description
BUFFERPOOL	Specifies the default values for buffers and LRU queues in a buffer pool for both the default page size buffer pool and for any non-default pages size buffer pools. BUFFERPOOL also encapsulates the values of BUFFERS, LRUS, LRU_MAX_DIRTY, and LRU_MIN_DIRTY which were earlier specified separately. The format of BUFFERPOOL is: default, lrus=num_lrus, buffers=num_buffers, lru_min_dirty=percent_min, lru_max_dirty=percent_max_dirty size=sizeK, buffers=num_buffers, lrus=num_lrus, lru_min_dirty=percent_min, lru_max_dirty=percent_max_dirty
LOCKS	Specifies the initial size of the lock table. The lock table holds an entry for each lock that a session uses. If the number of locks that sessions allocate exceeds the value of LOCKS, the database server increases the size of the lock table.
LOGBUFF	Specifies the size in kilobytes for the three logical-log buffers in shared memory.
PHYSBUFF	Specifies the size in kilobytes of the two physical-log buffers in shared memory.
RESIDENT	Specifies whether resident and virtual segments of shared memory remain resident in operating-system memory.
SERVERNUM	Specifies a relative location of the server in shared memory. If multiple servers are active on the same machine then this number has to be unique per server.

Parameter	Description
SHMTOTAL	Specifies the total amount of shared memory to be used by the database server for all memory allocations.

► Virtual shared memory parameters

Table 10-2 lists the parameters that effect the virtual portion of the memory buffer pool. You must shutdown and restart the database server for the parameters to take effect.

Table 10-2 Virtual shared memory configuration parameters

Parameter	Description
DS_HASHSIZE	Specifies the number of hash buckets in the data-distribution cache that the database server uses to store and access column statistics that the UPDATE STATISTICS statement generates in the MEDIUM or HIGH mode.
DS_POOLSIZE	Specifies the maximum number of entries in each hash bucket in the data-distribution cache that the database server uses to store and access column statistics that the UPDATE STATISTICS statement generates in the MEDIUM or HIGH mode.
PC_HASHSIZE	Specifies the number of hash buckets in the caches that the database server uses. Applies to UDR cache only.
PC_POOLSIZE	Specifies the maximum number of UDRs stored in the UDR cache.
SHMADD	Specifies the size of a segment that is dynamically added to the virtual portion of shared memory.
EXTSHMADD	Specifies the size of extension virtual segments that you add. Other virtual segment additions are based on the size that is specified in the SHMADD configuration parameter.
SHMTOTAL	Specifies the total amount of shared memory to be used by the database server for all memory allocations.
SHMVIRTSIZE	Specifies the initial size of a virtual shared-memory segment.
STACKSIZE	Specifies the stack size for the database server user threads. The value of STACKSIZE does not have an upper limit, but setting a value that is too large wastes virtual memory space and can cause swap-space problems.

► Performance parameters

Table 10-3 lists the ONCONFIG parameters that set shared-memory performance options. The database server must be shutdown and restarted for the parameters to take effect.

Table 10-3 Performance parameters

Parameter	Description
CKPTINTVL	Specifies, in seconds, the frequency at which the database server checks to determine whether a checkpoint is needed. When a full checkpoint occurs, all pages in the shared-memory buffer pool are written to disk.
CLEANERS	Specifies the number of page-cleaner threads available during the database server operation.
RA_PAGES	Specifies the number of disk pages to attempt to read ahead 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. It is used in conjunction with RA_PAGES.

10.2.3 Setting SQL statement cache parameters

Table 10-4 lists the ONCONFIG parameters that set SQL statement cache parameters that effect the performance. For the parameters to take effect, you need to shutdown and restart the database server.

Table 10-4 SQL statement cache configuration parameters

Parameter	Description
STMT_CACHE	Determines whether the database server uses the SQL statement cache.
STMT_CACHE_HITS	Specifies the number of references to a statement before it is fully inserted in the SQL statement cache.
STMT_CACHE_NOLIMIT	Controls whether to insert qualified statements into the SQL statement cache after its size is greater than the STMT_CACHE_SIZE value.
STMT_CACHE_NUMPOOL	Specifies the number of memory pools for the SQL statement cache
STMT_CACHE_SIZE	Specifies, in kilobytes, the size of the SQL statement cache.

Changing forced residency

To change the usage status of the resident portion of the shared memory we can either use the **onmode** utility or change the RESIDENT parameter of the ONCONFIG file. The residency status can be changed either temporarily or permanently.

Temporarily change the residency: Online mode

Use the **onmode** utility to change the residency status. You must have DBA authority to perform this action. You can perform this action while the server is in the online mode:

- ▶ To turn on residency, execute the following command: **onmode -r**
- ▶ To turn off residency, execute the following command: **onmode -n**

The changes are temporary till the time either an onmode command is issued to revert back the setting or till the time the server is restarted, at which time the status is set to the value as specified in the RESIDENT parameter in the ONCONFIG file. The ONCONFIG value of RESIDENT is not changed through this method.

Change the residency: Offline mode

You can change the RESIDENT parameter value in the ONCONFIG file to change the status of the residency. The status becomes effective when the server is started.

Adding segments to the virtual portion of shared memory

This is usually not necessary as the server will automatically allocate additional shared memory on a need-by basis, except in very rare cases when the operating system enforces some internal restrictions for the number of segments a process can allocate. In such cases you can use the **-a** option of the onmode utility to allocate additional shared memory segments.

Monitoring shared memory

In most cases the **onstat** utility can be used to monitor the status of the database shared memory. The **onstat** utility shows a snapshot of the memory at a given point of time during which it is issued. Table 10-5 illustrates some of the **onstat** sub-commands that can be used to monitor the shared memory.

Table 10-5 Monitoring shared memory

Onstat parameter	What is monitored
-g seg	Shared-Memory Segments
-s	Shared-Memory Latches
-p	Shared-Memory Profile Buffers: through the statistic (bufwaits) that indicates the number of times that sessions had to wait for a buffer. Buffer Pool Activity: through the statistic (ovbuff) that indicates the number of times the database server attempted to exceed the maximum number of shared buffers specified by buffers value in the BUFFERPOOL configuration parameter.
-B	Buffers: all buffers currently in use.
-b	Buffers: Details of each buffer in use.
-X	Buffers: complete list of all threads that are waiting for buffers in addition to the information provided by -b.
-R	Buffers: information about buffer pools.
-F	Buffer Pool Activity: Statistics of count by write type (foreground, LRU, chunk) of the writes performed.
-R	Buffer Pool Activity: number of buffers in each LRU queue and the number and percentage of the buffers that are modified or free.

10.3 Configurable page size and buffer pools

In this section, we discuss the following topics related to the configurable page size feature:

- ▶ Why configurable page size
- ▶ Advantages of using the configurable page size feature.
- ▶ Specifying page size

10.3.1 Why configurable page size

The following is a brief description of two primary reasons for page size:

- ▶ Long data rows split over multiple pages

If you have data rows of size greater than 2 KB in the table, they split over multiple pages. So every time you access these rows of size greater than 2 KB, IDS has to read the home page of the row to know which page to read next. This increases the disk activity.

- ▶ Need for longer index keys

This has been a particular issue with UNICODE data, which causes an increase in the maximum length of key values due to the use of multiple bytes for each character.

10.3.2 Advantages of using the configurable page size feature

To understand the advantages of configurable page size, you must first understand page overhead. Figure 10-3 depicts an example of page overhead.

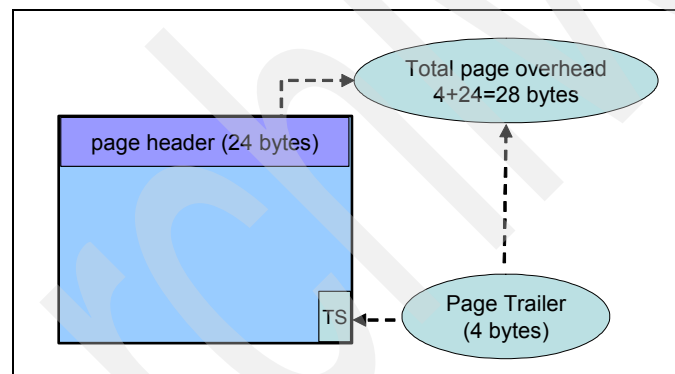


Figure 10-3 Page overhead

The advantages of using the configurable page size are:

- **Space efficiency**

Consider the example of thirty rows of 1200 bytes each. One row can fit in a 2 K page size and 3 rows can fit in a 4 K page size. Table 10-6 provides the space requirement and percent of space saved for various page sizes.

Table 10-6 Space requirement for 30 rows of 1200 bytes each

Page size	Number of pages required	Total space required	Saving percent
2 K	30	60 K	
4 K	10	40 K	33%
6 K	6	36 K	40%

- **Increased maximum key size**

As you increase the key size fewer key entries can be accommodated in the page, which causes the B tree to become deeper, thus making it less efficient. By increasing the page size you can include more index keys on one page, making the B tree less deep and thus making it more efficient.

- **Access efficiency**

You can increase the efficiency of the operating system I/O operation by putting large rows on one page. This results in fewer page operations per row.

10.3.3 Specifying page size

You can specify the page size while creating the dbspace. Figure 10-4 shows the example of using the **onspace** command to specify the page size.

```
$ userid informix onspaces -c -d dbspace1 -k 16 -p /ATM/MAIN950/dbspaces/dbsp -o 0 -s 50000
Verifying physical disk space, please wait ...
Space successfully added.

** WARNING ** A level 0 archive of Root DBSpace will need to be done.
$ onstat -d

IBM Informix Dynamic Server Version 10.00.UC5      -- On-Line -- Up 19:09:33 -- 38912 Kbytes

Dbspaces
address number  flags      fchunk  nchunks  pgsz  flags  owner  name
b7317e8  1      0x60001    1        1    2048   N B   informix rootdbs
c1134f0  2      0x60001    2        1   16384   N B   informix dbspace1
2 active, 2047 maximum
```

Figure 10-4 The onspace command to specify the page size

All critical dbspaces, such as rootdbs, containing logical logs and dbspaces containing physical logs must use the basic page size.

Table 10-7 shows the things changed and not changed regarding a page concept with the configurable page size feature.

Table 10-7 Configurable page size changes

Not Changed	Changed
Maximum number of pages per partition still 16,777,216	Home data pages
Maximum number of rows per data page remains at 255	Remainder Pages
Maximum number of parts per key remains 16	Partition freemap pages
Text or byte types in a blobspace are not affected	Partition partition pages
Smartblobs are not affected	Chunk free list pages
User-defined external spaces are not affected	Index pages

Not Changed	Changed
R-tree indexes must be stored in a dbspace with the default page size	Partition BLOB pages - partition BLOBs will use the same page size as the large page size defined for the dbspace they reside in
The rootdbs must be the default page size	
The physical and logical log dbspaces must be the default page size	
Dynamically created logs must be in a default page size dbspace	

Buffer pools

The buffer pool in the resident portion of shared memory contains buffers that store dbspace pages read from disk. The pool of buffers comprise the largest allocation of the resident portion of shared memory. The number of buffer pools is dependent upon the default page size. The maximum number of buffer pools on a system with default page size of 2 k is 8 and with default page size of 4 k is 4. You use a BUFFERPOOL configuration parameter to specify information about buffer pool, including the number of buffers in a buffer pool.

Note: The deprecated configuration parameters are:
BUFFERS, LRUS, LRU_MIN_DIRTY, LRU_MAX_DIRTY
Instead of these configuration parameters, use BUFFERPOOL.

When you create a dbspace with a non-default page size, and if no buffer pool of this page size exists, a new buffer pool is created using the default buffer pool configuration. If there is no BUFFERPOOL default value in \$ONCONFIG, the value from the onconfig.std file is used. Example 10-20 shows the value set of BUFFERPOOL configuration parameter in the onconfig file.

Example 10-20 Bufferpool

```
BUFFERPOOL
size=16K,buffers=1000,lrus=4,lru_min_dirty=50.000000,lru_max_dirty=60.000000
BUFFERPOOL
size=2K,buffers=2000,lrus=8,lru_min_dirty=50.000000,lru_max_dirty=60.000000
```

10.4 Dynamic OPTCOMPIND

The OPTCOMPIND configuration parameter helps the optimizer choose an appropriate access method for the application. When the optimizer examines join plans, OPTCOMPIND indicates the preferred method for performing the join operation for an ordered pair of tables. Till now the value of OPTCOMPIND can be set in the ONCONFIG file at the server level and in the environment. With this feature in IDS V10 you can change the value of OPTCOMPIND within a session and control the type of execution plan generated depending on the type of query being executed. The OPTCOMPIND environment variable/onconfig parameter can be set to values 0, 1, 2 having the following meanings:

- 0 A nested-loop join is preferred, where possible, over a sort-merge join or a hash join.
- 1 When the transaction isolation mode is not Repeatable Read, the optimizer behaves as in setting 2; otherwise, the optimizer behaves as in setting 0.
- 2 Nested-loop joins are not necessarily preferred. The optimizer bases its decision purely on costs, regardless of transaction isolation mode.

If you set the value of OPTCOMPIND using the new command, then that value will take precedence over both the environment setting (if specified) and the ONCONFIG setting. The value of OPTCOMPIND will not change even if the application switches to another database. Within a session, OPTCOMPIND can now be set using the command:

```
SET ENVIRONMENT OPTCOMPIND <'value'>; -- value {'0','1','2', DEFAULT}
```

Consider a database dbs1 having the following tables and indexes as defined in Example 10-21.

Example 10-21 Table definitions

```
CREATE TABLE resident (id INT, name CHAR (20));  
CREATE INDEX uqidx ON resident (id);  
CREATE TABLE chapters ( owner_id INT, topic CHAR (12));  
CREATE INDEX dupidx ON chapters(owner_id);
```

Now within a session you can influence the access path chosen by setting appropriate values for OPTCOMPIND. If a nested loop join is preferred then you can set the value of OPTCOMPIND to either 0, or set it to 1 and the current transaction isolation level should be set to Repeatable Read.

```
SET ENVIRONMENT OPTCOMPIND '0';
```

Or

```
SET ENVIRONMENT OPTCOMPIND '1';  
SET ISOLATION TO REPEATABLE READ;  
SELECT * FROM residents, chapter WHERE resident.id = chapters.owner_id;
```

The query produces the explain output as shown in Example 10-22.

Example 10-22 Explain output

```
SELECT * FROM resident, chapters WHERE resident.id = chapters.owner_id
```

```
Estimated Cost: 3
```

```
Estimated # of Rows Returned: 1
```

```
1) informix.resident: SEQUENTIAL SCAN
```

```
2) informix.chapters: INDEX PATH
```

```
(1) Index Keys: owner_id (Serial, fragments: ALL)
```

```
Lower Index Filter: informix.resident.id =
```

```
informix.chapters.owner_id
```

```
NESTED LOOP JOIN
```

Now, if you want the optimizer to base its decision purely on cost then the value of OPTCOMPIND can be set to 2 or can be set to 1 and the current isolation level should not be Repeatable Read.

```
SET ENVIRONMENT OPTCOMPIND '2';
```

Or

```
SET ENVIRONMENT OPTCOMPIND '1';
```

The same select statement produces the explain output as shown in Example 10-23.

Example 10-23 Explain output

```
select * from resident, chapters where resident.id = chapters.owner_id
```

Estimated Cost: 2

Estimated # of Rows Returned: 1

1) informix.chapters: SEQUENTIAL SCAN

2) informix.resident: SEQUENTIAL SCAN

DYNAMIC HASH JOIN

Dynamic Hash Filters: informix.resident.id =
informix.chapters.owner_id

IDS delivers services (SOA)

In this chapter, we introduce the service-oriented architecture (SOA) and document how IDS can easily be integrated into such an environment by:

- ▶ Providing Web services on top of IDS through the WOF framework and exposing IDS database operations through EGL (Enterprise Generation Language) and Java based Web services.
- ▶ Consuming Web services by using the Apache Axis Java framework in combination with IDS J/Foundation and the Open Source gSOAP framework to consume Web services through UDR written in the C programming language.

11.1 An introduction to SOA

At a very simplistic level, an *SOA* is a collection of services on a network that communicate with one another in order to carry out business processes. The communication can either be data passing or can trigger several services that implement some activity. The services are loosely coupled, have platform independent interfaces, and are fully reusable.

SOA is a business-centric, IT architectural approach that supports integrating your business as linked, repeatable business tasks or *services*. SOA helps users build composite applications, which are applications that draw upon functionality from multiple sources within and beyond the enterprise to support horizontal business processes.

An SOA helps hide the IT complexity that is inherent in even seemingly simple interactions. One key technical foundation of SOA are *Web services*, which we will introduce later in this chapter.

SOA is the architectural style whose goal is to achieve loose coupling among interacting software agents. A *service* is a unit of work done by a service provider to achieve desired end results for a *service consumer*. Both provider and consumer are roles played by software agents on behalf of their owners.

Note: SOA is not really new, but the availability of more and improved Web services applications is making SOA much more powerful and easier to implement today.

11.1.1 An SOA example: an Internet bookstore

To better illustrate the usage of an SOA, let us take a look behind the scenes of a real-world scenario: the ordering process of an Internet bookstore.

The customer point of view

A book is ordered online, and a few days later it is delivered to your home.

What happens behind the scenes

Behind the scenes of your book order, the follow process happens:

- Your identity is authenticated.

At the time you register a new account with the Internet bookstore, it needs to verify your billing address to make sure that you are a valid customer. To do this, it is very likely using a Web service that is provided by a third-party provider such as a telecommunications company, a bank, or the government.

- ▶ Your charge card is validated.

Because the Internet bookstore normally does not issue a charge card (credit card, debit card, or similar) it uses a validation Web service that is provided by a central credit card validation system or by a bank directly to verify that you are using a valid payment card.

- ▶ Your order is acknowledged by e-mail.

This service might be directly provided by the Internet bookstore, but it can be easily seen as another internal service call in the overall ordering process.

- ▶ The order needs to be sent to a distributor.

The Internet bookstore might have its own warehouses to stock some of the most requested books. In the case of a book which it does not stock, it needs to forward the order request to a distributor, who in return might send out the requested book directly to Internet bookstore customer. In that case the distributor is offering a Web service to its order entry system.

- ▶ The book is located and boxed for shipping.

Either at the bookstore's warehouse or the distributor (see above) the book is packaged eventually. As soon as the book has been packaged the new status is reflected on the order status page that is associated with your account. Depending on where the book has been packaged the status change is documented by either an internal or an external service call.

- ▶ The bookstore hands off the packaged books to the shipper (at which point the shipper's supply chain management system tracks the movement of the purchase).

Finally the Internet bookstore ships the books to your home and, because it is not likely that the bookstore is in the shipping business itself, the bookstore hands over that task to a shipping company. As soon as the shipper has taken over the package its generating a shipment tracking number and also offers typically a Web service to allow the tracking of the package in transit to your home.

- ▶ The shipment is acknowledged by e-mail.

As a customer you definitely would like to know what is the status of my order and therefore most serious Internet merchants are sending out shipping confirmation e-mails. This likely an internal service of the Internet bookstore.

- ▶ The books are delivered and acknowledgement of receipt.

Much like you as the customer who wants to know what the status of your Internet order is, the Internet bookstore also would like to know that your order has been correctly delivered to your house. For that purpose they rely on the delivery notification Web service provided by the shipping company.

How SOA comes into play

Each of the applications in the process performs a service that is orchestrated increasingly by SOA.

11.1.2 What are Web services

A Web service is a set of related application functions that can be programmatically invoked over the Internet. Businesses can mix and match Web services dynamically 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:

- ▶ **Web services are self-contained:** On the client side, a programming language with XML and HTTP client support is enough to get you 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.
- ▶ **Web services are self-describing:** The client and server need to recognize only the format and content of request and response messages. The definition of the message format travels with the message; no external metadata repositories or code generation tools are required.
- ▶ **Web services are 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 might be anything. Some examples, theatre review articles, weather reports, credit checks, stock quotations, travel advisories, or 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 inter operability allows businesses to dynamically publish, discover, and bind a range of Web services through the Internet.

Web services best practices

Although one could describe Web services simply as XML-protocol based remote function calls, you should avoid treating them as such for certain kinds of applications. Based on some real world projects which have been implemented at customer sites, most often without any IBM constancy, we already have learned quite a few lessons. So, you need to watch out for the following issues if

you're planning to implement Web services on top of your IBM Informix infrastructure:

- ▶ Do not use Web services between layers of an application or, for example, within a Java application server. The parsing of every Web service message is very costly and will slow down your application.
- ▶ Do not use Web services if you're not exposing external interfaces, for example, for inter operability or if you don't use an XML document based workflow.
- ▶ Use Web services on the edge of your application server to expose external APIs or if you need to execute remote calls through a firewall.
- ▶ If you have a need to execute function calls between Java application servers you might want to consider other protocols, for example, RMI/IIOP.

11.2 IDS 10 as a Web service provider

There are multiple options to use IDS as a Web service provider. Those options heavily depend on your development environment, programming language preferences and deployment platforms.

In the following sections, we discuss the most common Web services approaches for IDS developers and users.

11.2.1 IDS Web services based on Enterprise Java Beans (EJBs)

Creating an IDS Web service based on EJBs is a very straightforward process. In order to access an IDS based entity bean, you create a stateless session bean first and then use the Web services wizard in the Rational SDP to generate the necessary code for accessing the session bean.

Because these kind of Web services are more or less database independent due to the intermediate abstraction layer (session and entity beans), we did not include any example in this book, but instead, we refer you to another IBM Redbook that covers this topic in great detail: *Self-Study Guide: WebSphere Studio Application Developer and Web Services*, SG24-6407.

11.2.2 IDS and simple Java Beans Web services

Using Java beans for IDS Web services is a very flexible and simple approach. The Java bean could contain either Informix JDBC calls to the database, data access bean code from IBM (a different abstraction layer to a pure JDBC application), calls to the SQLToXML and XMLToSQL class libraries, or even

Java bean code which has been generated by any other third-party Java development environment.

11.2.3 IDS 10 and EGL Web services

Those developers who have the need to develop robust Web services on top of IDS, but who do not want to use pure Java technology to achieve that goal, should take a serious look at IBM Enterprise Generation Language and its very powerful Web services support.

Since version 6.0.1 of the Rational Software Development Platform (SDP), EGL offers a simple but powerful way of supporting Web services. You can write EGL applications which can provide Web services or consume Web services.

The current Web services support in EGL relies on the WebSphere Application Server Web services runtime framework, but future releases of EGL will likely support all standard J2EE application servers in combination with standard Web services runtime frameworks (for example, Apache's Axis).

The combination of easy data server access in EGL and the included Web services support makes EGL a very interesting alternative to coding SOA applications in Java, especially for developers who have a non-Java background.

Because EGL is an already established conversion path for VisualAge® Generator and Informix 4GL applications, the recently added EGL Web services support allows those customers to easily integrate their existing applications into a modern SOA framework by just converting their existing applications into EGL.

In the next section we will show how easily EGL can be utilized to provide Web services on top of IDS.

11.2.4 EGL Web service providing

Before we start on the details on how to develop an EGL based Web service, one should mention that EGL actually supports two kind of services:

- ▶ EGL Service: A type of service for applications written entirely in EGL
- ▶ EGL Web service: A type of service that is created using EGL, but can be accessed from both EGL and non-EGL clients

This section focuses only on EGL Web services which can be also called from any standards compliant Web service client.

The high-level steps to create an EGL Web service are:

1. Configure your project for Web services: In this first step you set project and Workspace properties to enable Web Service development and testing.
 - a. Specify Workspace Capabilities to support Web services development
 - b. Specify Web service-specific Build Options
2. Define the Web Service EGL Part: Here you will create a new EGL file of type Service
3. Code the Web Service Business Logic: In this new Service EGL file, you will add EGL functions and variables (statements), that perform the service (for example, business logic) required by your application
4. Generate the Web Service: After you have finished coding, you will save and generate Java for your Service
5. Optional: Test the Web Service interactively: You can then test your Web service interactively using the Rational SDP Web service testing tools. If there are logic issues, you can return to step 3 and fix them.
6. Optional: Create the Web service Binding Library: When you are satisfied with your logic/testing results, you will create a Web Service Binding Library, which contains entries that allow you to deploy and call your new Web service from an EGL client (a Pagehandler file, program, library or some other Service Part)
7. Optional: Create a JSP™ page to test the Web service as a client: Finally, you can create a page and EGL Pagehandler to call the Web service from an Internet application.

A simple EGL Web service example

To better understand EGL Web services, let us take a look at the simple example in Example 11-1.

The task is to provide a Web service called `getAllCustomers` which returns a list of all customers from the customer table in the underlying Informix stores demo database. The service has one parameter which is actually being used to return the customer record entries and one return value which contains the actual number of customer records returned.

Example 11-1 A simple EGL Web services example

```
// EGL Web service
```

```
package EGLWebServices;
```

```
record Customer type SQLRecord
```

```

                                {tableNames = [{"customer"}], keyItems =
["customer_num"]}
    customer_num int;
    lname char(15);
    fname char(15);
    company char(20);
    city char(15);
end

Service CustomerService

    function getAllCustomers(customers Customer[]) returns(int)
        get customers;
        return (size(customers));
    end

end

```

After coding and generating the EGL service part (Example 11-1), you notice that the EGL code generator already generated the necessary WSDL file for the newly developed Web service into the WebContent/WEB-INF/wsdl folder.

In order to test the new EGL Web service within the Rational SDP, make sure that the integrated WebSphere application server test environment is up and running. Then, follow these steps:

1. Navigate to the Web Services/Services/CustomerServiceService folder.
2. Right-click the WSDL file and select Test with Web Services Explorer.

After selecting the getAllCustomers Web service operation, you see a result such as depicted in Figure 11-1.

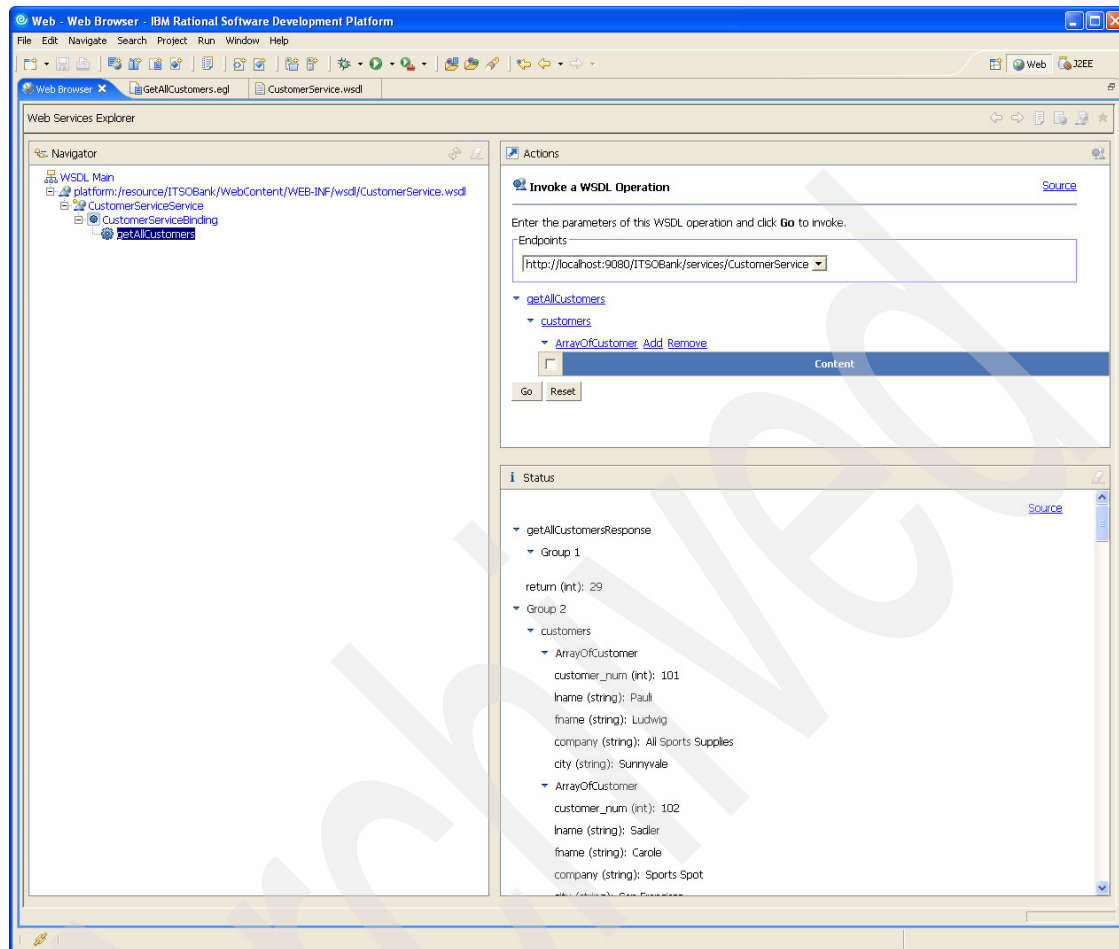


Figure 11-1 Using the Web Services Explorer to test the EGL Web service

11.2.5 IDS 10 and WORF (DADX Web services)

The document access definition extension (DADX) Web services had been originally developed with IBM DB2 and its XML Extender in mind. It allows you to easily wrap IBM DB2 XML Extender or regular SQL statements inside a Web service.

Fortunately, the non XML Extender related operations also work without any problems when using IBM Informix IDS¹. The supported DADX functions for IDS are:

- ▶ Query
- ▶ Insert
- ▶ Update
- ▶ Delete
- ▶ Call Stored Procedures (limited support for IDS 7²)

The runtime component of DADX Web services is called Web Services Object Runtime Framework (WORF). WORF uses the SOAP protocol and the DADX files and provides the following features:

- ▶ Resource based deployment and invocation
- ▶ Automatic service redeployment, at development time, when defining resource changes
- ▶ HTTP GET and POST bindings, in addition to SOAP
- ▶ Automatic WSDL and XSD generation, including support for UDDI Best Practices

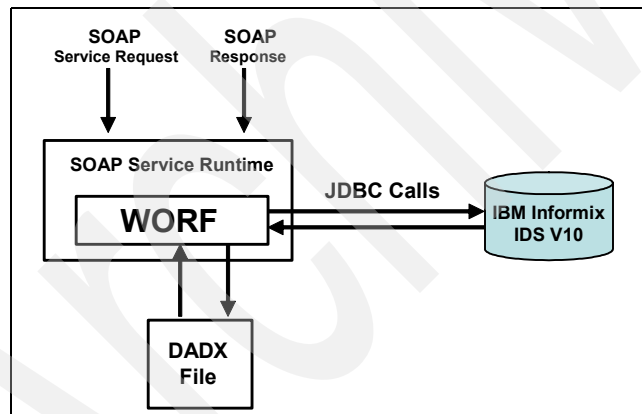


Figure 11-2 How WORF and IDS work together

So how does WORF handle a Web service request in combination with IDS?

1. WORF receives an HTTP SOAP GET or POST service request.

The URL of the request specifies a DADX or DTD file, and the requested action, which can be a DADX operation or a command, such as TEST,

¹ WORF has been already certified against IDS 10.x

² In IDS 7, you can only call stored procedures which do not return any results.

WSDL, or XSD. A DADX operation can also contain input parameters. WOF performs the following steps in response to a Web service request:

2. Loads the DADX file specified in the request.
3. Generates a response, based on the request.

For operations:

- Replaces parameters with requested values
- Connects to IDS and runs any SQL statements, including UDR calls
- Formats the result into XML, converting types as necessary

For commands:

- Generates necessary files, test pages, or other responses required.

4. Returns the response to the service requestor.

Because it is so easy to implement an IDS based Web service with DADX Web services, in the next section we look at how to develop such a service.

How to build a DADX Web service with IDS 10

In order to build a DADX Web service, we need to have some SQL statements on which the service should be based. In this section we are also assuming that the reader knows how to use the Rational development tools. For the following examples we are using the Rational Application Developer 6.0.1.1 (RAD).

Before we can actually build the DADX Web service we need to create a new Dynamic Web Project and define some SQL Statements. To keep the example simple, we will only define two statements, one SELECT statement and one INSERT statement.

Creating a Web project and a connection to the database

To create a Web project and a connection to the database:

1. Start RAD, and then define a new workspace (or choose an existing one)
2. Create a new Web Project by selecting **File** → **New** → **Dynamic Web Project**.

Give the new Web project a name and optionally choose an associated page template while navigating through the project wizard.

In our example, we name the project *InformixSOADemo*. In the Project Explorer window, you should see two project folders, one for the actual Web project (located in the Dynamic Web Projects folder) and a related Enterprise Archive (EAR) project with a similar name in the Enterprise Applications folder.

3. Define a connection to the standard IDS stores demo database and create a simple SELECT statement to select all customer data from the demo database.
4. Switch to the RAD Data view by selecting **Window** → **Open Perspective** → **Data**.
5. Right-click in the Database Explorer window and select **New Connection**.
6. On the first screen on the New Database Connection wizard, choose a database manager and JDBC driver and provide a connection name, for example *StoresDemo*.
7. In the New Database Connection, complete the correct connection properties to connect to the stores_demo database.

Choose the appropriate Informix Dynamic Server version in the Select a database manager field. For IDS 10.x you should use Informix Dynamic Server, V9.4 and the JDBC driver Class location should point to a JDBC driver jar file of version 3.30.JC1 or higher.

Figure 11-3 shows the New Database Connection window.

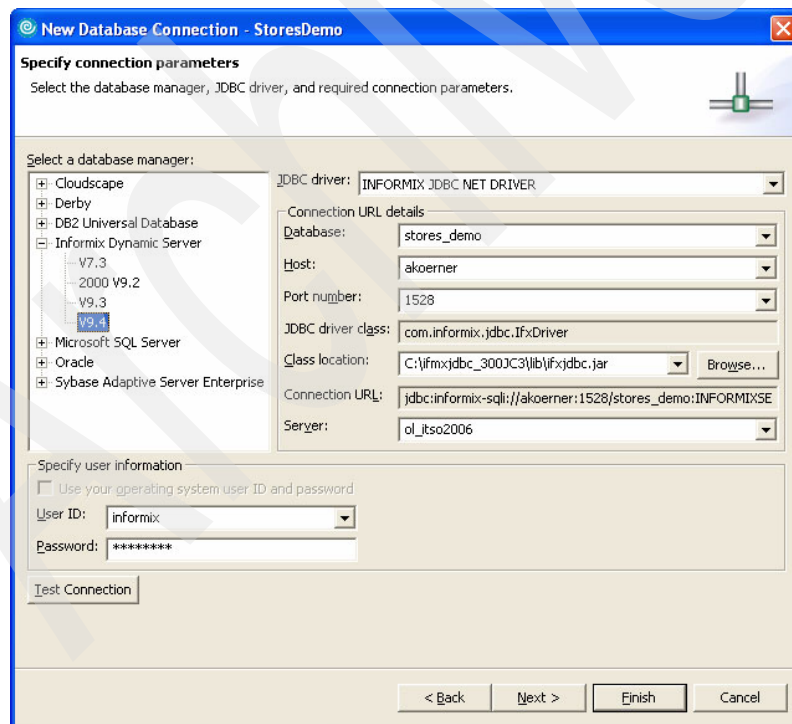


Figure 11-3 The New Database Connection wizard window with IDS settings

Before you proceed, it's recommended to use the Test Connection button to verify that all connection details are correct. On the following wizard screens, you have some options include and exclude some tables from underlying database schema.

Tip: If your database and its tables are owned by the user `informix`, you might want to deselect the *Schema NOT LIKE `informix`* option on the second wizard screen. Otherwise, you will not be able to import the schema information of all the tables which belong to user `informix`.

Towards the end of the schema import from the IDS database, you are asked if you want to copy the schema information into an existing project. After answering that question with *yes*, select the newly created Dynamic Web project (InformixSOADemo).

Defining a SELECT statement by utilizing the Query Builder

In the Data Definition window, follow these steps:

1. Navigate to the **InformixSOADemo** → **Web Content** → **WEB-INF** → **InformixSOADemo** → **stores_demo** → **Statements** folder.
2. Right-click the **Statements** folder and choose **New** → **Select Statement**.
3. Name the Statement, for example, *selectOneCustomer*, and click **OK**.
4. Now, you should see the interactive query builder. In the tables window, you need to select (by right-clicking) the tables that you want to include in the query.

In our demo, we select only the table `informix.customer` because we want to show the complete customer information. Because we want to include all attributes from the customer table, select all customer attributes in the table attribute check boxes.

5. We want to select only one customer, therefore we need to define a WHERE condition. To do this, select the Conditions tab in the lower window of the query builder. Select **informix.customer.customer_num** as the column, and choose equal (=) as the operator.
6. We also need to provide a host variable, which acts as a placeholder for different `customer_num` values later in the process. Let us name the host variable `:customernum`. (The colon is important!)
7. Now, save your statement into the Web project by selecting **File** → **Save stores_demo - selectOneCustomer**.

Defining an INSERT statement for the demo DADX Web service

To define an INSERT statement, follow these steps:

1. Switch to the Data perspective by selecting **Window → Open Perspective → Data**.
2. In the Data Definition window, open the InformixSOADemo/Web Content/InformixSOADemo/stores_demo folder. Right-click the Statements folder and select **New → Insert Statement**.
3. Name the new statement InsertOneCustomer and click **OK**.
4. In the interactive SQL builder window, right-click in the Tables window. Select **Add Table** and then from the tables selection menu, select the **informix.customer** table.
5. Within the informix.customer table, select all attributes for the INSERT statement. In the window below the Tables window, define the host variables as placeholders for the later inserts, which should be executed against the customer table.
6. To make it simple, name all host variables by using the column name with a colon (:) in front. To do this, click the **Value** column for each table attribute and enter the host variable name, for example, :fname for the fname attribute.

Important: Because the customer_num attribute is defined as a SERIAL data type in the database, we set the insert value to zero to generate a new customer_num value automatically during each insert! So, eventually, the SQL builder window should look like in Figure 11-4. As soon as you have defined the INSERT statement, save it into the demonstration Web project.

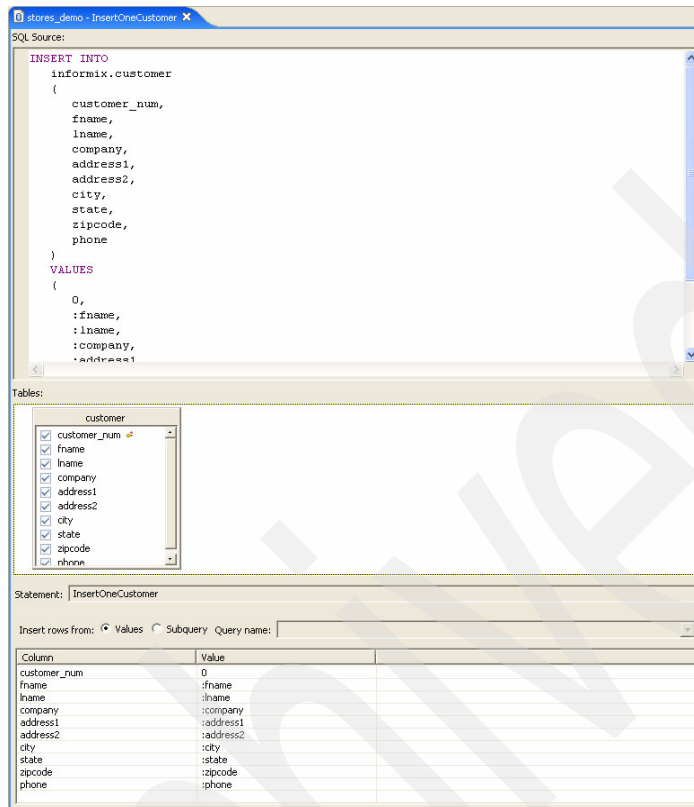


Figure 11-4 The InsertOneCustomer Statement is being constructed in the SQL Builder

Creating a DADX group and define its properties

In preparation for the to be generated DADX file, we need first to create a DADX group, which combines one or more SQL statement into one logical DADX Web service. So it could make sense, for example, to group all operations on the customer table into one group, while operations on the account table will be grouped into another DADX group.

Each DADX group also maintains its own database connection properties, so one could also use different DADX groups to connect to different databases or even different database servers (vendors).

To create a new DADX group:

1. Open the **Web Perspective** by selecting **Window → Open Perspective → Web**. Then select the Project Explorer window.

2. Select **File** → **New** → **Other** → **Web Services** → **Web Service DADX Group Configuration**. Click **Next**.
3. In the next window, select the **InformixSOADemo** folder and then click **Add group**.
4. For the group name, enter **ITSOCustomerService**. Click **OK**.
5. While still being in the same window, now select the **InformixSOADemo/ITSOCustomerService** folder and then click **Group properties**.
6. In the DADX Group Properties window, complete the following information:
DB driver: `com.informix.jdbc.IfxDriver`
DB URL:
`jdbc:informix-sqli://akoerner:1528/stores_demo:INFORMIXSERVER=ol_its
o2006;user=informix;password=informix`
7. Leave the other fields as-is. Click **OK**.
8. In the DADX Group Configuration window, click **Finish**.

Generating the DADX file

Now, we can generate the DADX file for the two SQL Statements. To do this:

1. Select **File** → **New** → **Other** → **Web Services** → **DADX File**. Click **Next**.
2. In the Create DADX window select **InformixSOADemo** as the project and the **ITSOCustomerService DADX Group**. As a file name, enter **ITSOCustomerService.dadx**. Also select the **Generate a DADX file from a list of SQL queries or Stored Procedures**. Click **Next**.
3. In the Select SQL Statements and Stored Procedures window, open the **InformixSOADemo/Web Content/InformixSOADemo/stores_demo/Statements** folder.
4. Because we would like to select both SQL statements (**insertOneCustomer**, **selectOneCustomer**) we need to select the **insertOneCustomer** statement first and then control-click the **selectOneCustomer** also. Now both statements should be selected (highlighted). Click **Next**.
5. Just click **Next** in the Select DAD Files window because DAD files are not yet supported with IBM Informix IDS.
6. In the DADX operations window, click **Finish**.

The generated ITSOCustomerService.dadx file should look the one in Example 11-2. Notice the XML compliant format and the specific DADX keywords.

Example 11-2 The ITSOCustomerService.dadx file

```
<?xml version="1.0" encoding="UTF-8"?>
<dadx:DADX xmlns:dadx="http://schemas.ibm.com/db2/dxx/dadx"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xsi:schemaLocation="http://schemas.ibm.com/db2/dxx/dadx
dadx.xsd">
  <dadx:operation name="selectOneCustomer">
    <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
      <![CDATA[
      ]]>
    </dadx:documentation>
    <dadx:query>
      <dadx:SQL_query>
        <![CDATA[
          SELECT * FROM informix.customer WHERE
informix.customer.customer_num = :customernum
        ]]>
      </dadx:SQL_query>
      <dadx:parameter name="customernum" type="xsd:int"/>
    </dadx:query>
  </dadx:operation>
  <dadx:operation name="InsertOneCustomer">
    <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
      <![CDATA[
      ]]>
    </dadx:documentation>
    <dadx:update>
      <dadx:SQL_update>
        <![CDATA[
          INSERT INTO informix.customer ( customer_num, fname,
lname, company, address1, address2, city, state, zipcode, phone )
VALUES ( 0, :fname, :lname, :company, :address1, :address2, :city,
:state, :zipcode, :phone )
        ]]>
      </dadx:SQL_update>
      <dadx:parameter name="fname" type="xsd:string"/>
      <dadx:parameter name="lname" type="xsd:string"/>
      <dadx:parameter name="company" type="xsd:string"/>
    </dadx:update>
  </dadx:operation>
</dadx:DADX>
```

```

        <dadx:parameter name="address1" type="xsd:string"/>
        <dadx:parameter name="address2" type="xsd:string"/>
        <dadx:parameter name="city" type="xsd:string"/>
        <dadx:parameter name="state" type="xsd:string"/>
        <dadx:parameter name="zipcode" type="xsd:string"/>
        <dadx:parameter name="phone" type="xsd:string"/>
    </dadx:update>
</dadx:operation>
</dadx:DADX>

```

Because the interactive query builder in RAD only supports SELECT, INSERT, UPDATE and DELETE statements you might have to edit the generated DADX file manually if you want to add support for IDS UDR.

Creating a DADX Web service based on the generated DADX file

Now let us generate the necessary files for a DADX Web service based on the DADX file we generated in the previous section. First, we need to prepare the InformixSOADemo Web project for Informix Dynamic Server database access in combination with the DADX Web service:

1. In the Project Explorer, right-click the InformixSOADemo project folder and then select **Properties**.
2. In the Properties window, select **Java Build Path** and then the **Libraries** tab.
3. Now add the Informix JDBC driver to the Class Path entries by clicking **Add External JARs**. In the file browser select the correct **ifxjdbc.jar** file and the **ifxjdbcx.jar** file and click **Open**.
4. Close the Properties window by clicking **OK**.

Now we build the Web service itself:

1. Open the Web perspective and in the InformixSOADemo Web project click the file Java Resources/JavaSource/groups.ITSOCustomerService/ITSOCustomerService.dadx.
2. Select **File** → **New** → **Other** → **Web Services** → **Web Service**. Click **Next**.
3. In the Web Services window select as the **Web service type**: DADX Web Services. In addition, select the **Start Web service in Web project** option and the options, **Overwrite files without warning** and **Create folders when necessary**. Click **Next**.
4. In the Object Selection Page verify that have selected the correct DADX file. In our example it is **/InformixSOADemo/JavaSource/groups/ITSOCustomerService/ITSOCustomerService.dadx**. Click **Next**.

5. In the **Service Deployment Configuration** leave the default values and as the **Service project** choose **InformixSOADemo** and as the EAR project choose **InformixSOADemoEAR**. Click **Next**.
6. In the Web Services DADX Group properties verify the database connection information. Click **Next**.
7. In the Web Service Publication window just **click Finish**.

Let us test the newly created Web service with the built-in RAD Web services Test client:

1. In the Project Explorer Window, in the InformixSOADemo Web project folder, locate the WebContent/wsdl/ITSOCustomerService/ITSOCustomerService.wsdl file. Right-click this file and the select **Web Services** → **Test with Web Services Explorer**.
2. While in the Web Services Explorer, select **theService** → **theSOAPBinding** → **selectOneCustomer**.
3. In the Actions window, enter a valid value (such as, 104) for the customer_num value and click **GO**. You see the result in Figure 11-5.

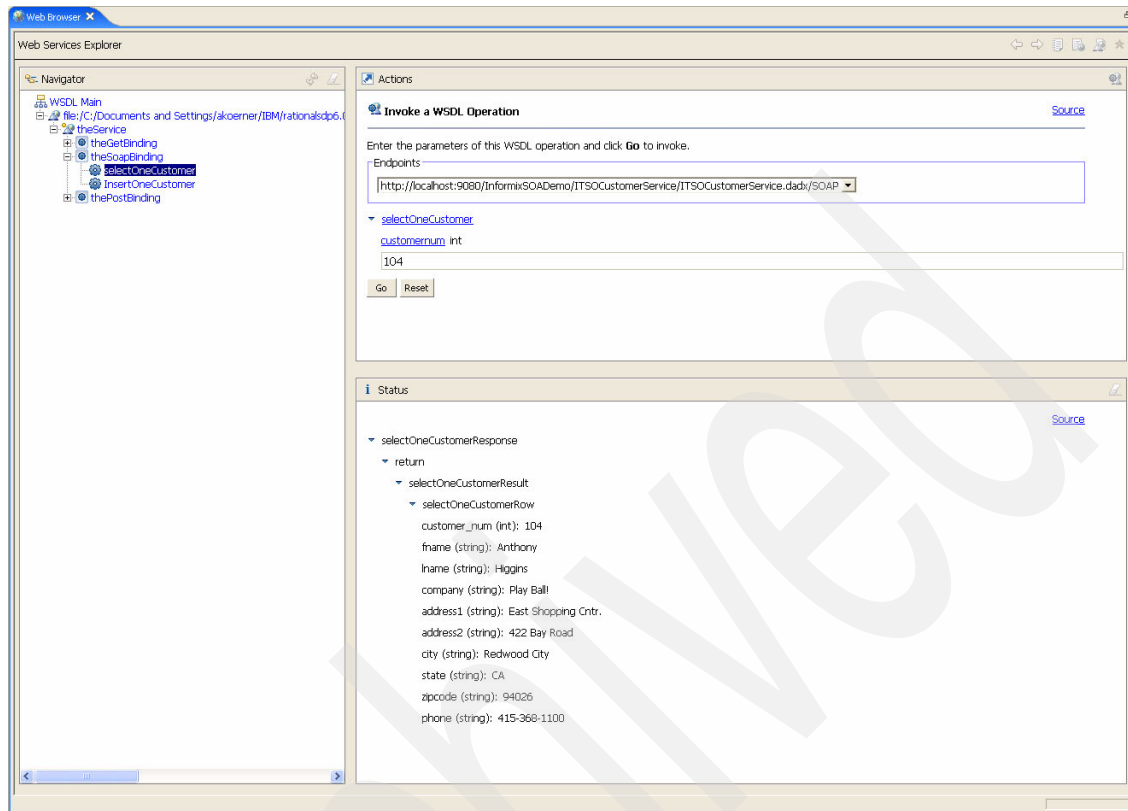


Figure 11-5 RAD Web Services Explorer to test the selectOneCustomer DADX service

4. Now you can also try the InsertOneCustomer Web service. In this case you need to provide some values for the customer fields (fname, lname, and so forth). The Web services explorer is shown in Figure 11-5.

DADX support for UDR and stored procedures

In addition to standard SQL statements like SELECT, INSERT, DELETE and UPDATE, the WOF framework also supports the execution of UDR or stored procedures in IDS. In order to do this, the framework utilizes (internally) the JDBC CallableStatement class which is a very portable way of calling stored procedures and functions in database servers.

Because this feature is unfortunately not supported through the interactive SQL builder in RAD, we need to either create a new DADX file or modify an existing one.

Before we go ahead with an example, let us take a look at the DADX file syntax for stored procedures/functions based on the XML schema for DADX files (Example 11-3).

Example 11-3 DADX call operation (XML schema definition)

```
<element name="call">

  <annotation>
    <documentation>
      Calls a stored procedure.
      The call statement contains in, out, and in/out parameters using host
variable syntax.
      The parameters are defined by a list of parameter elements that are uniquely
named
      within the operation.
    </documentation>
  </annotation>

  <complexType>
    <sequence>
      <element name="SQL_call" type="string"/>
      <element ref="dadx:parameter" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="dadx:result_set" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>

  <unique name="callParameterNames">
    <selector xpath="dadx:parameter"/>
    <field xpath="@name"/>
  </unique>

  <unique name="callResultSetNames">
    <selector xpath="dadx:result_set"/>
    <field xpath="@name"/>
  </unique>

</element>
```

As mentioned earlier, the WORF framework utilizes the JDBC `java.sql.CallableStatement` interface for the execution of IDS UDR. Therefore the syntax for calling routines in IDS this way should follow the JDBC guidelines. For a simple example in DADX syntax how to call a UDR which does not return any results, see Example 11-4.

Example 11-4 Simple UDR call in DADX syntax

```
<dadx:operation name="createOneCustomerSimple">
  <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
    </dadx:documentation>
  <dadx:call>
    <dadx:SQL_call>
      <![CDATA[
        { call create_customer_simple (:fname, :lname, :company,
        :address1, :address2, :city, :zipcode, :state, :phone)}
      ]]>
    </dadx:SQL_call>
    <dadx:parameter name="fname" type="xsd:string" kind="in"/>
    <dadx:parameter name="lname" type="xsd:string" kind="in"/>
    <dadx:parameter name="company" type="xsd:string" kind="in"/>
    <dadx:parameter name="address1" type="xsd:string" kind="in"/>
    <dadx:parameter name="address2" type="xsd:string" kind="in"/>
    <dadx:parameter name="city" type="xsd:string" kind="in"/>
    <dadx:parameter name="zipcode" type="xsd:string" kind="in"/>
    <dadx:parameter name="state" type="xsd:string" kind="in"/>
    <dadx:parameter name="phone" type="xsd:string" kind="in"/>
  </dadx:call>
</dadx:operation>
```

If you need to return results back to the DADX Web service consumer you might have different options:

- Starting with IDS 9 you can utilize multiple out parameters in the UDR parameter list. To use these out parameters in combination with DADX you need to declare them as in/out parameters and the Web service caller might have to supply dummy values (for example, zero for integer types) to make it work. This behavior seems to be IDS specific and does not apply to other databases. The UDR `create_customer_out` in Example 11-5 shows a simple SPL UDR which uses one out parameter (`newcustomernum`).

Example 11-5 IDS UDR with an out parameter (in SPL)

```
create procedure create_customer_out (fname lvvarchar, lname lvvarchar,
    company lvvarchar, address1 lvvarchar, address2 lvvarchar,
    city lvvarchar, zipcode lvvarchar, state lvvarchar,
```

```

        phone lvarchar, OUT customernum int)

define new_customernum int;
insert into customer values (0, fname, lname, company, address1,
                           address2, city, state, zipcode,
phone);
let new_customernum = dbinfo('sqlca.sqlerrd1');
let customernum = new_customernum;

end procedure;

```

Example 11-6 shows the correct DADX syntax for calling such a UDR. Notice the in/out option for the newcustomernum parameter.

Example 11-6 DADX syntax fragment for the IDS UDR from Example 11-5

```

<dadx:operation name="createOneCustomer">
  <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
    </dadx:documentation>
  <dadx:call>
    <dadx:SQL_call>
      <![CDATA[
        { call create_customer_out (:fname, :lname, :company,
:address1, :address2, :city, :zipcode, :state, :phone,
:newcustomernum)}
      ]]>
      </dadx:SQL_call>
      <dadx:parameter name="fname" type="xsd:string" kind="in"/>
      <dadx:parameter name="lname" type="xsd:string" kind="in"/>
      <dadx:parameter name="company" type="xsd:string" kind="in"/>
      <dadx:parameter name="address1" type="xsd:string" kind="in"/>
      <dadx:parameter name="address2" type="xsd:string" kind="in"/>
      <dadx:parameter name="city" type="xsd:string" kind="in"/>
      <dadx:parameter name="zipcode" type="xsd:string" kind="in"/>
      <dadx:parameter name="state" type="xsd:string" kind="in"/>
      <dadx:parameter name="phone" type="xsd:string" kind="in"/>
      <dadx:parameter name="newcustomernum" type="xsd:int"
kind="in/out"/>
    </dadx:call>
  </dadx:operation>

```

Tip: This restriction seems to be specific to the DADX/IDS combination, because a similar restriction had been already removed because the IBM Informix JDBC 2.21.JC4 driver and is no longer valid. Callers need to use `registerOUTparameter()` only and they do not need to use `setXXX()` method on OUT parameters. A future version of DADX will very likely address this change in the Informix JDBC driver.

- You could simply return a result for a UDR or even complete results sets. See the following important tip regarding the support in IDS for that feature.

Tip: DS 10 supports a feature which allows that the columns of a result set for a UDR can have display labels. The WOF framework requires the usage of those labels in IDS or one could not use UDRs with result sets.

To give you already the information about how the DADX syntax for an UDR what a result set should look like, take a look at the SPL UDR in Example 11-7 and the associated DADX syntax in Example 11-8. Notice the display label syntax in the stored procedure (returning ... as ...) and also the `result_set` definition and usage in the DADX file fragment.

Example 11-7 IDS stored procedure with display labels for the result set

```
create procedure read_address (lastname char(15))
    returning char(15) as pfname, char(15) as pname,
           char(20) as paddress1, char(15) as pcity,
           char(2)  as pstate, char(5) as pzipcode;
define p_fname, p_city char(15);
define p_add char(20);
define p_state char(2);
define p_zip char(5);
select fname, address1, city, state, zipcode
into p_fname, p_add, p_city, p_state, p_zip
from customer
where lname = lastname;
return p_fname, lastname, p_add, p_city, p_state, p_zip;
end procedure;
```

Example 11-8 shows the DADX syntax associated to Example 11-7.

Example 11-8 DADX syntax fragment for the UDR in Example 11-7

```
<dadx:result_set_metadata name="customerAddress" rowName="customer1">
  <dadx:column name="pfname" type="VARCHAR" nullable="true" />
  <dadx:column name="pname" type="VARCHAR" nullable="true" />
  <dadx:column name="paddress1" type="VARCHAR" nullable="true" />
  <dadx:column name="pcity" type="VARCHAR" nullable="true" />
  <dadx:column name="pstate" type="VARCHAR" nullable="true" />
  <dadx:column name="pzipcode" type="VARCHAR" nullable="true" />
</dadx:result_set_metadata>

<dadx:operation name="readOneCustomer">
  <dadx:documentation xmlns="http://www.w3.org/1999/xhtml">
  </dadx:documentation>
  <dadx:call>
    <dadx:SQL_call>
<![CDATA[
      { call read_address (:lname) }
]]>
    </dadx:SQL_call>
    <dadx:parameter name="lname" type="xsd:string" kind="in"/>
    <dadx:result_set name="customer1" metadata="customerAddress" />
  </dadx:call>
</dadx:operation>
```

Important: The result set metadata definitions (<dadx:result_set_metadata> </dadx:result_set_metadata> tag) are global to the DADX and must precede all of the operation definition elements in a DADX file

11.2.6 IDS 10 and other Web services environments (.NET, PHP)

Developers have options on choosing the development environment to use for their Web services development. Some might prefer the Java environment, while others prefer .NET or other powerful frameworks like PHP.

The IBM Informix database development APIs support all major development environments including .NET and PHP database access.

For more details on the IBM Informix development APIs which enable Web services development on top of IDS 10 in a .NET or PHP environment, also refer to 6.1.4, “IBM Informix .NET provider - CSDK” on page 193 and 6.2.1, “IDS V10 and PHP support” on page 209.

11.3 IDS 10 as a Web service consumer

In the previous sections, we described in detail how to use the different tools to enable IBM Informix IDS as a Web service provider. Now we would like to focus on IDS as a Web service consumer.

This section is intended as a how-to guide to use IDS 10 as a Web service consumer. It requires either a basic knowledge of the Java language (for example, you should know how to edit and compile a Java program) or you should know the basics of the C programming language. You should also have a basic understanding of the IDS 10 extensibility features.

Why IDS as a Web service consumer

In addition to provide Web services, it can be very interesting for an application developer to integrate existing Web services. Those Web services could be either special business-to-business scenarios or public accessible services like currency conversion, stock ticker information, news, weather forecasts, search engines, and many more. Would not it be great to have dynamic access to an official currency conversion service on a database level if the application needs to deal with this information? Or if an application wants to relate actual business data stored in an IDS database against news from news agencies?

Sources for public accessible Web services are, for example:

<http://www.webservicelist.com>
<http://www.xmethods.net>

Web services rely on very simple open standards like XML and SOAP and be accessed through any kind of client application. Typically those applications are written in Java, C++, or C#. For somebody who already has an existing application which is based on an SQL database and also already utilizes business logic in the database server through UDR, developers might want to integrate access to Web services on the SQL level.

Some of the advantages of having Web services accessible from SQL would include easy access through the SQL language and standardized APIs (for example, ODBC, JDBC), moving the Web service results closer to the data processing in the database server which could speed up applications, and providing Web service access to the non Java or C++ developers.

What are the Web service consumer requirements for IDS

In order to be able to call a Web service from within IDS, you need to be able to:

- ▶ Construct a SOAP message based on a given Web service description
- ▶ Send this SOAP message to the Web service provider through the required protocol (typically HTTP)
- ▶ Be able to receive the Web service response, parse it, and handle the results on an SQL level

All of this needs to be executed from the IDS SQL layer to achieve the required portability.

Why IDS 10 and not IDS 7

Although IDS 7 supports stored procedures with an already very powerful stored procedure language (SPL), it is somewhat limited if there is a need, for example, to access external networks or include external libraries.

IDS10 through its very powerful DataBlade technology allows the easy integration of external routines written in C or Java into so called UDRs. Those UDRs can also be written in SPL. So one can say that UDRs are the generalized description of SPL, C, and Java stored procedures. In addition to the very flexible options of writing UDRs, IDS 9 also supports new data types and user-defined types (UDTs).

Having these extensibility technologies available in IDS 10 in combination with the underlying, proven, high-end OLTP architecture of IDS 7 makes it a perfect choice to develop some database extensions which will provide access to Web services across standard network protocols.

Because you have the choice as an IDS 10 developer to either use C or Java for the development of Web service consumer routines, you could either include, for example, a C based SOAP framework or a Java based SOAP framework in your final solution.

To better demonstrate the flexibility of IDS 10 and to give you the choice on which programming language to choose for a Web service consumer implementation, we are documenting the use of the Apache AXIS Java framework (11.3.1, “Utilizing IDS and Apache’s AXIS for Web service consumption”) and the Open Source gSOAP C/C++ framework (11.3.6, “Consume Web services with IDS and the gSOAP C/C++ toolkit”) for the development of IDS 10 Web services consumer routines.

11.3.1 Utilizing IDS and Apache's AXIS for Web service consumption

In this section we describe how to use IDS and Apache AXIS for Web service consumption.

IDS 10 and J/Foundation

IDS 9 with J/Foundation enables database developer's to write server-side business logic using the Java language. Java UDRs have complete access to the leading extensible database features of the IDS 10 database. Making IDS 10 the ideal platform for Java database development.

In addition to Java UDRs, IDS conforms to the SQLJ standard for Java-stored procedures, enabling the use of the standard Java packages that are included in the Java Development Kit (JDK). Writing UDRs in Java delivers far more flexible applications that can be developed faster than C, and more powerful and manageable than stored procedure languages.

IDS with J/Foundation provides these advantages over other Java based solutions:

- ▶ Better performance and scalability
- ▶ Fully certified and optimized standard JVMs for each supported platform
- ▶ Simpler application integration
- ▶ Easy migration of existing Java applications
- ▶ Transaction control through stored data

J/Foundation is provided with IDS on many of the supported IDS 10 platforms.

Technology

IDS 10 provides the infrastructure to support Java UDRs. The database server binds SQL UDR signatures to Java executables and provides mapping between SQL data values and Java objects so that the database server can pass parameters and retrieve returned results. IDS 10 also provides support for data type extensibility and sophisticated error handling.

Java UDRs execute on specialized virtual processors called Java virtual processors (JVPs). IDS 10 embeds a Java virtual machine (JVM) in the code of each JVP. The JVPs are responsible for executing all server-based Java UDRs and applications.

Although the JVPs are mainly used for Java-related computation, they have the same capabilities as a CPU VP, and they can process all types of SQL queries. This eliminates the need to ship Java-related queries back and forth between CPU VPs and JVPs.

For more technical details of J/Foundation, refer to the IBM Informix J/Foundation Developer's Guide.

The Apache AXIS framework

So what is the Apache AXIS framework?

The Axis framework is a Java-based, open source implementation of the latest SOAP specification, SOAP 1.2, and SOAP with Attachments specification from the Apache Group. The following are the key features of this AXIS framework:

- ▶ **Flexible messaging framework:** Axis provides a flexible messaging framework that includes handlers, chain, serializers, and deserializers. A handler is an object processing request, response, and fault flow. A handler can be grouped together into chains and the order of these handlers can be configured using a flexible deployment descriptor.
- ▶ **Flexible transport framework:** Axis provides a transport framework that helps you create your own pluggable transport senders and transport listeners.
- ▶ **Data encoding support:** Axis provides automatic serialization of a wide variety of data types as per the XML Schema specifications and provides a facility to use your own customized Serializer and Deserializer.
- ▶ **Additional features:** Axis provides full support for WSDL as well as Logging, Error, and Fault Handling mechanisms.

Axis also provides a simple tool set to easily generate Java classes based on given Web service description files (WSDL) and has tools to monitor Web services.

The latest Axis distribution and more detailed information about Axis can be obtained at:

<http://ws.apache.org/axis>

11.3.2 Configuring IDS 10 and AXIS 1.3 for the examples

In the previous sections we have described the using of IDS with Apache AXIS. In this section we describe how to configure the two products for use together, along with providing examples for clarification.

Tip: All of the configuration and installation information in this section is based on Windows XP but can be easily also applied to other platforms such as Linux or UNIX.

Installing and preparing AXIS 1.3

First, you need to download the AXIS release from the following Web site:

<http://ws.apache.org/axis/java/releases.html>

The release that we use for the examples below is based on AXIS 1, version 1.3 Final. After downloading the release, extract the AXIS distribution into a directory of your choice (for example, directly into the C:\ directory). Make sure that you also extract the folder structure.

If you are finished, you should have an `<install_dir>\axis-1_3` directory.

In addition to AXIS we also need a JAXP 1.1 XML compliant parser. The recommended one is the Apache Xerces. Just download the latest stable version from the following Web site (for example, Xerces-J-bin.2.5.0.zip):

<http://xml.apache.org/dist/xerces-j>

Extract it into a local directory (for example, C:\). Eventually you should have an `<install_dir>\xerces-2_5_0` directory.

For more advanced Axis SOAP handling (for example, SOAP attachments), you also might want to download the following Java packages:

- ▶ jaf-1_0_2-upd2 (JavaBeans™ Activation Framework, <http://java.sun.com/products/javabeans/glasgow/jaf.html>)
- ▶ javamail-1_3_3_01 (Java Mail, <http://java.sun.com/products/javamail/>)

All the classpath settings in our examples below include the necessary activation.jar and mail.jar files out of the optional Java packages for completeness.

IDS 10 with J/Foundation configuration for AXIS

Because the AXIS Framework is Java based, we need to configure IDS 10 for Java UDRs. Before we go ahead, make sure that you're using an IDS 10 with J/Foundation. You can verify this by checking the \$INFORMIXDIR/extend directory for the existence of a krakatoa subdirectory. If this directory is missing, you do not have the correct version of IDS 10.

First, you need to enable J/Foundation for your IDS 10 instance:

1. Create an sbpace to hold the Java JAR files. The database server stores Java JAR files as smart large objects in the system default sbpace. If you do not already have a default sbpace, you must create one. After you create the sbpace, set the SBSPACENAME configuration parameter in the ONCONFIG file to the name that you gave to the sbpace.

2. Add (or modify) the Java configuration parameters in the ONCONFIG configuration file. The ONCONFIG configuration file (\$INFORMIXDIR/etc/\$ONCONFIG) includes the following configuration parameters that affect Java code:

- JDKVERSION
- JVPPROFILE
- JVMTHREAD
- JVPCLASSPATH
- JVPHOME
- JVPJAVAILIB
- JVPJAVAVM
- JVPLOGFILE
- JVPARGS
- VPCCLASS

Make sure that these parameters exist or are not un-commented. For an example ONCONFIG file fragment, see Example 11-9.

Example 11-9 J/Foundation settings for the AXIS framework in the IDS ONCONFIG file

```
VPCCLASS          jvp,num=1  # Number of JVPs to start with

JVPJAVAHOMEC:\informix\extend\krakatoa\jre# JDK installation root
directory
JVPHOMEC:\informix\extend\krakatoa# Krakatoa installation directory

JVPLOGFILEC:\informix\extend\krakatoa\ol_itso2006_jvp.log# VP log file
JVPPROFILEC:\informix\extend\krakatoa\jvpprops_ol_itso2006# JVP
property file

JDKVERSION        1.4          # JDK version supported by this server

# The path to the JRE libraries relative to JVPJAVAHOME
JVPJAVAILIB        \bin\

JVPJAVAVM          jsig;dbgmalloc;hpi;jvm;java;net;zip;jpeg

# Classpath to use upon Java VM start-up (use _g version for debugging)
#JVPCLASSPATH
C:\informix\extend\krakatoa\krakatoa.jar;C:\informix\extend\krakatoa\jdb
c.jar
JVPCLASSPATHfile:C:\informix\extend\krakatoa\jvp_classpath

#JVPARGS
-Djava.security.policy=C:\informix\extend\krakatoa\informix.policy
```

In Example 11-9, we also define the JVPCLASSPATH to point to a file in the krakatoa directory. Having an external file to contain the JVP classpath information gives us more flexibility regarding the maximal length of the JVPCLASSPATH because the length in the ONCONFIG file is otherwise limited to 256 characters. See Example 11-10 for an AXIS compliant classpath file.

Tip: In our examples, we are copying the AXIS class libraries directly into the \$INFORMIXDIR\extend\krakatoa directory to avoid any changes to the informix.policy file. I would be probably a cleaner approach to keep the AXIS files in their original directories and adjust the informix.policy file to allow access for the J/Foundation class loader.

Example 11-10 The jvp_classpath file for the AXIS integration

```
C:\informix\extend\krakatoa\krakatoa.jar;C:\informix\extend\krakatoa\jdb
bc.jar;C:\informix\extend\krakatoa\axis.jar;C:\informix\extend\krakatoa
\jaxrpc.jar;C:\informix\extend\krakatoa\saa.jar;C:\informix\extend\kra
katoa\commons-logging-1.0.4.jar;C:\informix\extend\krakatoa\commons-dis
covery-0.2.jar;C:\informix\extend\krakatoa\wsdl4j-1.5.1.jar;C:\informix
\extend\krakatoa\xercesImpl.jar;C:\informix\extend\krakatoa\xmlParserAP
Is.jar;C:\informix\extend\krakatoa\axis-ant.jar;C:\informix\extend\krak
atoa\log4j-1.2.8.jar;
```

In addition, we also need to modify the default security settings for the Java VM.

The default security settings for J/Foundation can be defined in the JVPHOME/informix.policy file. The necessary entries to support the AXIS framework with J/Foundation are listed in Example 11-11.

Example 11-11 The Informix.policy file with AXIS support

```
grant codeBase "file:/C:/informix/extend/krakatoa/-" {
    permission java.security.AllPermission;
};

grant {
    permission java.io.SerializablePermission "enableSubstitution";
    permission java.lang.RuntimePermission "shutdownHooks";
    permission java.lang.RuntimePermission "setContextClassLoader";
    permission java.lang.RuntimePermission "reflectionFactoryAccess";
    permission java.lang.RuntimePermission "unsafeAccess";
    permission java.net.NetPermission "specifyStreamHandler";
    permission java.lang.reflect.ReflectPermission
"suppressAccessChecks";
```



```

    permission java.util.PropertyPermission "user.language","write";
    permission java.util.PropertyPermission "user.dir","write";
    permission java.security.SecurityPermission "getPolicy";
    permission java.util.PropertyPermission
"java.naming.factory.initial","write";
    permission java.util.PropertyPermission
"java.naming.provider.url","write";
};

grant {
    permission java.util.PropertyPermission
"java.protocol.handler.pkgs","write";
};

```

3. Create the JVP properties file (optional). It is optional to define the JVP properties, but they are often used for debugging Java UDRs. You will find a template file in the \$INFORMIXDIR\extend\krakatoa directory.
4. Set environment variables. You do not need any extra environment variables to execute UDRs written in Java code. However, because we are developing Java UDRs, you must include JVPHOME/krakatoa.jar in your CLASSPATH environment variable so that JDK can compile the Java source files that use Informix Java packages. For a complete description of the CLASSPATH settings for AXIS UDR development, refer to “Java classpath settings for AXIS UDR development” on page 343.
5. Now, copy all Java class libraries from the AXIS distribution (for example, c:\axis-1_3\lib) into the \$INFORMIXDIR\extend\krakatoa directory.
6. Finally, copy the **xercesImpl.jar** and the xmlParserAPIs.jar class library from the Xerces distribution (for example, C:\xerces-2_5_0) also into the \$INFORMIXDIR\extend\krakatoa directory.

Java classpath settings for AXIS UDR development

Example 11-12 shows the Java classpath for developing the AXIS based UDRs.

Example 11-12 Classpath settings for AXIS UDR development

```

C:\axis-1_3\lib\axis.jar;C:\axis-1_3\lib\jaxrpc.jar;C:\axis-1_3\lib\saa
j.jar;c:\axis-1_3\lib\commons-logging-1.0.4.jar;C:\axis-1_3\lib\commons
-discovery-0.2.jar;C:\axis-1_3\lib\wsdl4j-1.5.1.jar;C:\xerces-2_5_0\xer
cesImpl.jar;C:\xerces-2_5_0\xmlParserAPIs.jar;C:\informix\extend\krakat
oa\krakatoa.jar;C:\jaf-1.0.2\activation.jar;C:\javamail-1.3.3_01\mail.j
ar;.

```

11.3.3 The IDS 10 / AXIS Web service consumer development steps

Before we start to access some Web services from IDS 10, let us consider the required steps:

1. Obtain access to the WSDL file for the desired Web service, either by downloading it to the local server or have access to it through the http protocol.
2. Use the AXIS WSDL2Java tool to generate the Web service Java class files.
3. Compile the class files from step 2 (no coding needed!)
4. Write a small Java UDR wrapper to access the generated AXIS classes. You can take the Java UDR wrappers from the examples below as templates for your own projects.
5. Create a Java jar file which should contain the generated AXIS class files and your Java UDR wrapper class.
6. Write a simple SQL script to register your Java UDR in the IDS database of your choice.
7. Register your Java UDR in the database of your choice with the SQL script from step 6.
8. Run and test your Java UDRs to access the Web services.

11.3.4 The AXIS WSDL2Java tool

The WSDL2Java tool, which is part of the `org.apache.axis.wsdl.WSDL2Java` class, is the starting point to generate Java classes from a given WSDL file.

This tool is executed by the following command line:

```
java org.apache.axis.wsdl.WSDL2Java <WSDL-file-URL>
```

Tip: To make the execution of this tool easier for you throughout the examples in the following sections, we suggest that you create a small batch file similar to the one shown in Example 11-13. Call this file (in a Windows environment) `wsdl2java.bat`.

Example 11-13 The `wsdl2java.bat` file (for Windows platforms)

```
SET CLASSPATH=.  
SET CLASSPATH=%CLASSPATH%;C:\axis-1_3\lib\axis.jar  
SET CLASSPATH=%CLASSPATH%;C:\axis-1_3\lib\jaxrpc.jar  
SET CLASSPATH=%CLASSPATH%;C:\axis-1_3\lib\saaj.jar  
SET CLASSPATH=%CLASSPATH%;C:\axis-1_3\lib\commons-logging-1.0.4.jar  
SET CLASSPATH=%CLASSPATH%;C:\axis-1_3\lib\commons-discovery-0.2.jar
```

```

SET CLASSPATH=%CLASSPATH%;C:\axis-1_3\lib\wsdl4j-1.5.1.jar
SET CLASSPATH=%CLASSPATH%;C:\xerces-2_5_0\xercesImpl.jar
SET CLASSPATH=%CLASSPATH%;C:\xerces-2_5_0\xmlParserAPIs.jar
SET CLASSPATH=%CLASSPATH%;C:\axis-1_3\lib\axis-ant.jar
SET CLASSPATH=%CLASSPATH%;C:\axis-1_3\lib\log4j-1.2.8.jar
SET CLASSPATH=%CLASSPATH%;C:\jaf-1.0.2\activation.jar
SET CLASSPATH=%CLASSPATH%;C:\javamail-1.3.3_01\mail.jar
echo -----
echo --- Classpath has been set for AXIS needs ---
echo -----
java org.apache.axis.wsdl.WSDL2Java -p %2 -v %1
SET CLASSPATH=%TMPCLASSPATH%

```

The wsdl2java.bat script file has two parameters: the WSDL file URL and a package name. The package name becomes also a local subdirectory to the directory in which you are executing the wsdl2java.bat file.

The WSDL file URL can be either a local file name or a URL on the Internet (for example, <http://www.someserver.com/webserviceinfo/myservice.wsdl>).

11.3.5 A simple IDS 10 / AXIS Web service consumer example

So let us start with our example project, the currency exchange Web service from <http://www.xmethods.net>. This Web service allows the currency conversion between different foreign currencies. You only have to provide the source currency country name and then the target currency country name.

Now follow the development steps we have outlined in 11.3.3, “The IDS 10 / AXIS Web service consumer development steps” on page 344:

1. Obtain a copy of the Web service WSDL file:

The WSDL file for this Web service can be obtained from:

<http://www.xmethods.net/sd/2001/CurrencyExchangeService.wsdl>

You can either download the WSDL file to your local disk or use the above URL directly as input to the WSDL2Java tool. For your convenience we have also included the WSDL file in Example 11-14.

Example 11-14 The CurrencyExchange WSDL file

```

<?xml version="1.0"?>
<definitions name="CurrencyExchangeService"
targetNamespace="http://www.xmethods.net/sd/CurrencyExchangeService.wsdl"
xmlns:tns="http://www.xmethods.net/sd/CurrencyExchangeService.wsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"

```

```

xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <message name="getRateRequest">
    <part name="country1" type="xsd:string"/>
    <part name="country2" type="xsd:string"/>
  </message>
  <message name="getRateResponse">
    <part name="Result" type="xsd:float"/>
  </message>
  <portType name="CurrencyExchangePortType">
    <operation name="getRate">
      <input message="tns:getRateRequest" name="getRate"/>
      <output message="tns:getRateResponse" name="getRateResponse"/>
    </operation>
  </portType>
  <binding name="CurrencyExchangeBinding"
type="tns:CurrencyExchangePortType">
    <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getRate">
      <soap:operation soapAction=""/>
      <input name="getRate">
        <soap:body use="encoded"
namespace="urn:xmethods-CurrencyExchange"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
      </input>
      <output name="getRateResponse">
        <soap:body use="encoded"
namespace="urn:xmethods-CurrencyExchange"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
      </output>
    </operation>
  </binding>
  <service name="CurrencyExchangeService">
    <documentation>Returns the exchange rate between the two
currencies</documentation>
    <port name="CurrencyExchangePort"
binding="tns:CurrencyExchangeBinding">
      <soap:address
location="http://services.xmethods.net:80/soap"/>
    </port>
  </service>
</definitions>

```

While looking at the WSDL file, you might have already noticed that the two input parameter (country1 and country2) are of type String and the result is of type float.

2. Now we need to generate the AXIS Java classes for our Web service.

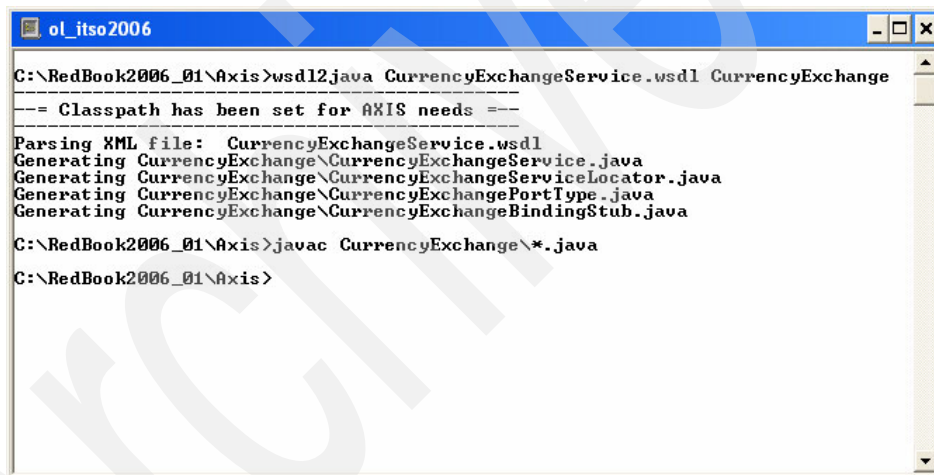
To do this, create a directory of your choice (for example, C:\Redbook2006_01\Axis) and copy the WSDL file into this directory.

From a command line window, run the prepared wsd12java.bat scrip file with the following parameters:

```
wsd12java CurrencyExchangeService.wsdl CurrencyExchange
```

This script generates a subdirectory called CurrencyExchange, and this subdirectory should include the following files:

CurrencyExchangeBindingStub.java, CurrencyExchangePortType.java, CurrencyExchangeService.java, CurrencyExchangeServiceLocator.java (Figure 11-6).



```
ol_itso2006
C:\RedBook2006_01\Axis>wsd12java CurrencyExchangeService.wsdl CurrencyExchange
---= Classpath has been set for AXIS needs =---
Parsing XML file: CurrencyExchangeService.wsdl
Generating CurrencyExchange\CurrencyExchangeService.java
Generating CurrencyExchange\CurrencyExchangeServiceLocator.java
Generating CurrencyExchange\CurrencyExchangePortType.java
Generating CurrencyExchange\CurrencyExchangeBindingStub.java
C:\RedBook2006_01\Axis>javac CurrencyExchange\*.java
C:\RedBook2006_01\Axis>
```

Figure 11-6 Generating and compiling the CurrencyExchange AXIS classes

3. Now you need to compile the generated Java classes from step 2 by simply executing:

```
javac CurrencyExchange\*.java
```

Before you execute the Java compiler, make sure that you have set the CLASSPATH environment variable correctly (Example 11-13) and also that you have the Java compiler in your PATH environment variable (for example, C:\j2sdk1.4.2_06\bin).

4. In order to use the generated AXIS class files for the CurrencyExchange Web service we need to write a simple Java wrapper UDR to call the required methods.

So first take a look at the final code in Example 11-15.

Example 11-15 CurrencyExchangeUDRs..java

```
import CurrencyExchange.*;

public class CurrencyExchangeUDRs
{
    public static double currencyExchange( String country1, String
country2)
                                                throws
Exception
    {
        double  RetVal;

        CurrencyExchange.CurrencyExchangeService service =
new
CurrencyExchange.CurrencyExchangeServiceLocator();

        CurrencyExchange.CurrencyExchangePortType port =
service.getCurrencyExchangePort();

        RetVal = port.getRate(country1, country2);

        return RetVal;
    }
};
```

The `CurrencyExchange` method implements the Web service API by accepting the two country descriptions as Java strings and returns a Java double type.

First, we need to create a service instance of type `CurrencyExchangeService` which can be achieved by creating a new `CurrencyExchangeServiceLocator` object.

Then we need to obtain the port object of type `CurrencyExchangePortType` from the service object.

And finally we need to call the `getRate(String, String)` method to generate the SOAP message which is then being sent to the Web service provider.

The `getRate()` method is defined in the `CurrencyExchangeBindingStub.java` file.

Save the Java code from Example 11-15 into your example directory (for example, `C:\RedBook2006_01\Axis`) as `CurrencyExchangeUDRs.java`.

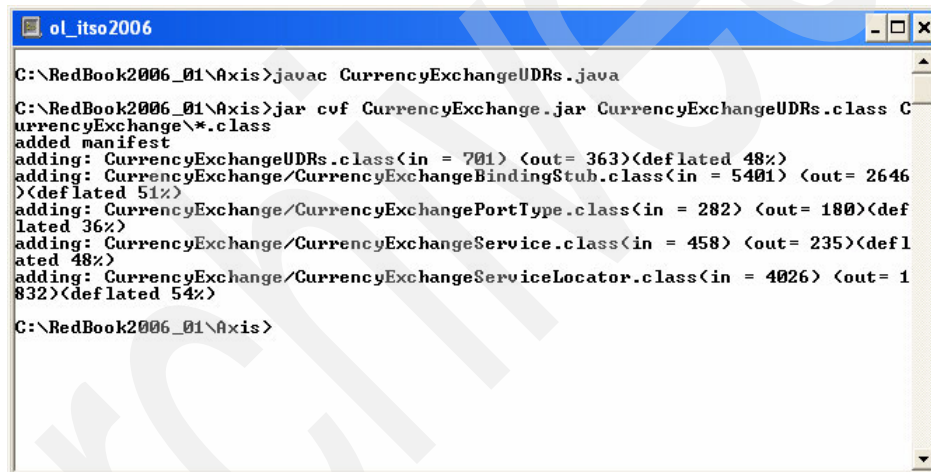
Now compile the `CurrencyExchangeUDRs.java` file:

```
javac CurrencyExchangeUDRs.java
```

5. In preparation for the registration in your IDS 9 database we need to pack all of our classes (generated AXIS classes plus the UDR wrapper) into a Java jar file. To do this, execute this command:

```
jar cvf CurrencyExchange.jar CurrencyExchangeUDRs.class  
CurrencyExchange\*.class
```

(Also see Figure 11-7.)



```
ol_itso2006  
C:\RedBook2006_01\Axis>javac CurrencyExchangeUDRs.java  
C:\RedBook2006_01\Axis>jar cvf CurrencyExchange.jar CurrencyExchangeUDRs.class C  
urrencyExchange\*.class  
added manifest  
adding: CurrencyExchangeUDRs.class(in = 701) (out= 363)(deflated 48%)  
adding: CurrencyExchange/CurrencyExchangeBindingStub.class(in = 5401) (out= 2646  
) (deflated 51%)  
adding: CurrencyExchange/CurrencyExchangePortType.class(in = 282) (out= 180)(def  
lated 36%)  
adding: CurrencyExchange/CurrencyExchangeService.class(in = 458) (out= 235)(defl  
ated 48%)  
adding: CurrencyExchange/CurrencyExchangeServiceLocator.class(in = 4026) (out= 1  
832)(deflated 54%)  
C:\RedBook2006_01\Axis>
```

Figure 11-7 Compile the UDR wrapper and create the jar file

6. Now we need to create a simple SQL script to first store our `CurrencyExchange.jar` file which contains the UDR wrapper plus the generated AXIS classes into the database and then connect the Java classes with the SQL layer by defining a Java UDR with the `CREATE FUNCTION` SQL statement.

You can use the SQL script from Example 11-16 as a template for similar Java UDRs in the future. So on the SQL level, we name our UDR simply *CurrencyExchange*. This routine takes two LVARCHARs as parameters and returns a SQL FLOAT data type which matches the Java double type.

Example 11-16 The register_CurrencyExchange.sql script

```
execute procedure
install_jar('file:C:/RedBook2006_01/Axis/CurrencyExchange.jar','Currenc
yExchange');

execute procedure ifx_allow_newline('t');

begin work;

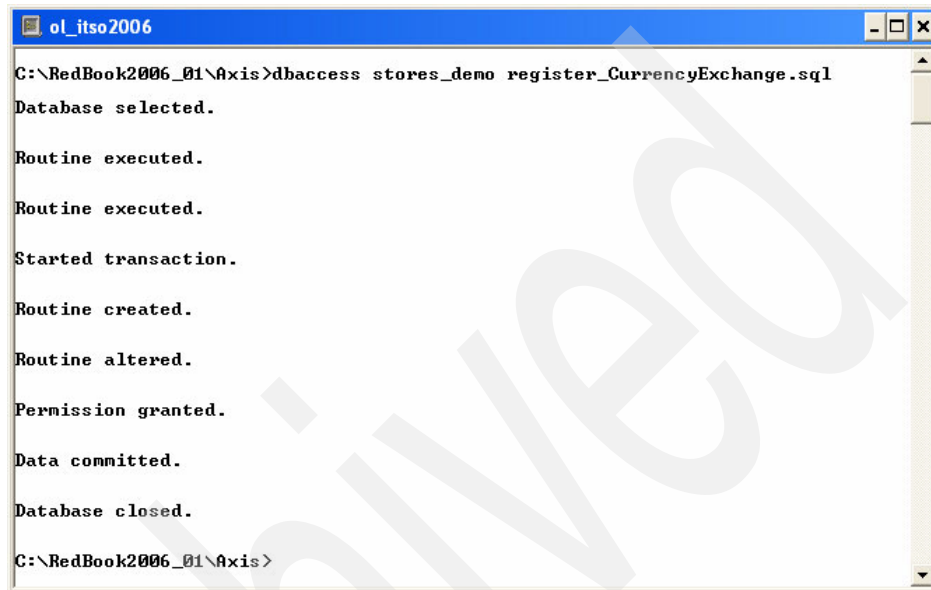
create function CurrencyExchange (lvarchar, lvarchar)
returns float as exchange_rate
external name
'CurrencyExchange:CurrencyExchangeUDRs.currencyExchange(java.lang.Strin
g, java.lang.String)'
language java;

alter function CurrencyExchange (lvarchar, lvarchar)
with (add parallelizable);

grant execute on function CurrencyExchange (lvarchar, lvarchar) to
public;

commit work;
```

The `install_jar` procedure stores the `CurrencyExchange.jar` into a smart blob in the default smart blob space in the IDS 10 instance and gives it the symbolic name `CurrencyExchange`. which can be used in the `create function` statement to reference the jar file. See Figure 11-8.



```
ol_itso2006
C:\RedBook2006_01\Axis>dbaccess stores_demo register_CurrencyExchange.sql
Database selected.
Routine executed.
Routine executed.
Started transaction.
Routine created.
Routine altered.
Permission granted.
Data committed.
Database closed.
C:\RedBook2006_01\Axis>
```

Figure 11-8 Register the Java Wrapper UDR with the `stores_demo` database

The `create function` finally registers the Java UDR with the database and makes it available to any SQL compliant application.

7. In order to register your `CurrencyExchange` UDR you should have a database with logging enabled. Assuming you might want to register your UDR with the IDS `stores_demo` database you only have to run the SQL script by executing:

```
dbaccess stores_demo register_CurrencyExchange.sql
```

See also Figure 11-8.

8. Now we are ready to test the Java UDR to call the Web service. Before you can test the UDR make sure that you're connected to the Internet. Then, for example, start `dbaccess` to connect to the `stores` database and execute the `CurrencyExchange` function. Because we are using SQL and because SQL does not differentiate between lowercase and uppercase letters we just simply type:

```
execute function currencyexchange("<country1>", "<country2>")
```

For valid values for the country parameters, consult the CurrencyExchange Web service description on the Web site:

<http://www.xmethods.net>



```
ol_itso2006 - dbaccess stores_demo -
C:\informix>dbaccess stores_demo -
Database selected.
> execute function currencyexchange('euro', 'usa');

exchange_rate
1.237200021744
1 row(s) retrieved.
> execute function currencyexchange('euro', 'brazil');

exchange_rate
2.622699975967
1 row(s) retrieved.
>
```

Figure 11-9 Test of the CurrencyExchange() UDR from within dbaccess

Tip: If you are behind a firewall, then you might have to set additional properties for the J/Foundation Java VM through the JVPARGS variable.

So, if you typically use a SOCKS compliant proxy, replace the JVPARGS value in the ONCONFIG file with the following line:

```
-Djava.security.policy=C:\informix\extend\krakatoa\informix.policy;-
DsocksProxyHost=<SocksProxyhostname>;-DsocksProxyPort=<socksProxyPort
value>
```

If you are using a standard HTTP proxy, you might have to use the following value for JVPARGS instead:

```
-Djava.security.policy=C:\informix\extend\krakatoa\informix.policy;-
Dhttp.proxyHost=<httpProxyhostname>;-Dhttp.proxyPort=<httpProxyPortv
alue>
```

For more details about proxy support in the Java VM, consult the following Web site:

<http://java.sun.com/j2se/1.4.2/docs/guide/net/properties.html>

11.3.6 Consume Web services with IDS and the gSOAP C/C++ toolkit

In the previous sections, we described on how to consume Web services with IDS by using a Java based Web services framework in combination with the powerful IDS J/Foundation capabilities.

Because some IDS customers and developers might not be too much interested in using a portable, Java based approach, but instead might be looking for a more performant, but platform specific C based solution, we document in this section the integration of the Open Source gSOAP toolkit with IDS leading database extensibility.

Web service calls involve lots of parsing of XML formatted messages. This parsing is very costly CPU-wise and therefore the usage of a platform specific and optimized C toolkit like gSOAP can be very beneficial for an IDS based Web service consumption.

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.

gSOAP is capable to generate both, the Web service client or the Web service server code. In addition gSOAP is self contained, so no additional libraries or products are required which in return allows 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:

- ▶ <http://sourceforge.net/projects/gsoap2>
- ▶ <http://www.cs.fsu.edu/~engelen/soap.html>

11.3.7 Configuration of IDS 10 and gSOAP for the examples

Compared to the installation and configuration of the Axis Java framework, the setup of gSOAP is relatively simple. Just follow the few steps below.

gSOAP installation and configuration

In the first step you should download a recent version of gSOAP for your desired development platform (UNIX, Linux, or Windows) from the following URL:

http://sourceforge.net/project/showfiles.php?group_id=52781

Throughout this project, we have been using version 2.7.8c of gSOAP for Windows (Win32®). All of the examples below have been developed and tested on Windows XP SP2, but we also did a few tests on SLES 9 (SUSE Linux) with an earlier version of gSOAP for Linux to match the installed Linux system and C compiler libraries with the same positive results.

After downloading the gSOAP toolkit, extract the compressed file into a folder of your choice (for example, C:\RedBook2006_02\gsoap-win32-2.7). In the sections that follow, we refer to this gSOAP installation location as the GSOAP_DIR.

Because we need to compile C source code files, make also sure that you have a C-compiler installed on your development platform. For the examples below we have been using Microsoft Visual C++ .NET.

IDS 10 configuration for gSOAP

Because we are going to use a C language based UDR, the configuration of the IDS 10 instance is very easy. The only configuration we need to apply is to add an additional virtual processor (VP) class, called *soapvp*, to the IDS instance ONCONFIG file. By adding an dedicated virtual processor class to the IDS configuration, we can separate the execution of the blocking network calls of the Web service consumer DataBlade from the overall IDS query processing.

To enable at least one dedicated VP class for that purpose, add the following line to the ONCONFIG file of your IDS 10 instance:

```
VPCLASS soapvp,num=1
```

After restarting or starting the IDS instance you should see the additional VP listed after executing the `onstat -g glo` command (Figure 11-10).

```

ol_itso2006
C:\informix\etc>onstat -g glo

IBM Informix Dynamic Server Version 10.00.IC4      -- On-Line -- Up 00:18:23 -- 2
1568 Kbytes

MT global info:
sessions threads vps lngspins
0 14 10 1

total: sched calls thread switches yield 0 yield n yield forever
per sec: 0 6212240 0 269034 0 5943233 0 2183 0 132882

Virtual processor summary:
class vps usercpu syscpu total
cpu 1 2.23 0.60 2.83
aio 1 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
soc 2 0.00 0.00 0.00
msc 1 0.00 0.00 0.00
jvp 1 0.00 0.00 0.00
soapvp 1 0.00 0.00 0.00
total 10 2.23 0.60 2.83

Individual virtual processors:
up pid class usercpu syscpu total
1 2076 cpu 2.23 0.60 2.83
2 2960 adm 0.00 0.00 0.00
3 3224 jvp 0.00 0.00 0.00
4 5040 soapvp 0.00 0.00 0.00
5 4700 lio 0.00 0.00 0.00
6 3296 pio 0.00 0.00 0.00
7 2508 aio 0.00 0.00 0.00
8 5816 msc 0.00 0.00 0.00
9 1308 soc 0.00 0.00 0.00
10 5980 soc 0.00 0.00 0.00
tot 2.23 0.60 2.83

C:\informix\etc>

```

Figure 11-10 The newly configured VP (soapvp) has been started

In addition to ONCONFIG modification, make sure that you also installed the DataBlade Developers Kit (DBDK) which comes bundled with the Windows version of IDS 10. This concludes the installation and configuration section for IDS10/gSOAP.

11.3.8 The IDS 10 / gSOAP Web service consumer development steps

To get a better overview on how to develop Web service access with gSOAP and IDS 10, take a look at the following development steps:

1. Obtain access to the WSDL file for the desired Web service, either by downloading it to the local development machine or have access to it through the http protocol.
2. Use the gSOAP wsdl2h tool to obtain the gSOAP header (.h) file specification of a Web service from an existing WSDL document.
3. Invoke the gSOAP stub and skeleton compiler on the .h file from step 2 to create the necessary Web service client routines (.c and .h files) which can be later integrated into an IDS 10 UDR.

4. Write a small C UDR wrapper to access the generated gSOAP functions. To get you easily started with that step, we would recommend to use the Informix DBDK for that purpose.
5. Compile the C language files from step 3 and step 4 into a shared library which becomes a DataBlade for IDS 10 which contains the Web service access UDRs.
6. Write a simple SQL script or use an automatically DBDK generated one to register your Web service access UDR in the IDS 10 database of your choice.
7. Run and test your gSOAP based C UDRs to access the Web services.

11.3.9 A simple IDS 10 / gSOAP Web service consumer example

As discussed in 11.3.5, “A simple IDS 10 / AXIS Web service consumer example” on page 345, we implement the access to a public Web service provided by www.xmethods.net. However, this time we choose the DelayedStockQuote service:

1. Download the DelayedStockQuote WSDL file from the following Web site:
<http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl>
2. Copy the WSDL file into a local folder on your development machine and rename the file to *quotes.wsdl* to make the handling of the following steps easier.

3. From a command line window run the following gSOAP command on the WSDL file:

```
%GSOAP_DIR%\bin\wsdl2h -c -t %GSOAP_DIR%\typemap.dat quotes.wsdl
```

This step should generate a file called *quotes.h* in your current working directory. The **-c** option is important to generate the required C language template instead of the default C++ template.

4. From the same command line window, generate the necessary C stubs and skeletons (.c and .h files), which you need to integrate into an IDS 10 UDR by executing the following gSOAP command:

```
%GSOAP_DIR%\bin\soapcpp2 -c -C -I%GSOAP_DIR%\import quotes.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.

Before we go ahead with the UDR development, we can quickly test the generated gSOAP code by calling it from a simple, stand-alone test application. For that purpose create a new file called test.c and copy the C language code from Example 11-17 into it.

Example 11-17 A simple, stand-alone test program to test the StockQuote service

```
#include "soapH.h"
#include
"net_x002exmethods_x002eservices_x002estockquote_x002eStockQuoteBinding
.nsmap"

main()
{
    struct soap soap;

    float result;

    soap_init(&soap);

    soap_call_ns1__getQuote(&soap, NULL, NULL, "IBM", &result);

    printf("Delayed quote: %5.2f\n", result);
}
```

To compile the simple test program above on Windows execute the following Visual C++ command from a Visual C++ command line window:

```
cl -I %GSOAP_DIR%\import -I %GSOAP_DIR% -o test.exe test.c soapC.c
soapClient.c %GSOAP_DIR%\stdsoap2.c wsock32.lib
```

After compiling the test.c program you should be able to execute it and if your are connected to the internet, you should see the delayed stock quote for the symbol IBM displayed. See also Example 11-11.

If everything works so far, we can continue with the next step and write our Web service consumer C UDR with the help of the DBDK.

```

C:\RedBook2006_02\StockQuoteBlade>cl -I %GSOAP_DIR%\import -I %GSOAP_DIR% -o test.exe test.c soapC.c soapClient.c %GSOAP_DIR%\stdsoap2.c wsock32.lib
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 13.00.9466 for 80x86
Copyright (C) Microsoft Corporation 1984-2001. All rights reserved.

test.c
soapC.c
soapClient.c
stdsoap2.c
Generating Code...
Microsoft (R) Incremental Linker Version 7.00.9466
Copyright (C) Microsoft Corporation. All rights reserved.

/out:test.exe
/out:test.exe
test.obj
soapC.obj
soapClient.obj
stdsoap2.obj
wsock32.lib

C:\RedBook2006_02\StockQuoteBlade>test
Delayed quote: 78.54

C:\RedBook2006_02\StockQuoteBlade>

```

Figure 11-11 Compile and run the simple, stand-alone test application

5. Create a simple C UDR that defines a SQL callable function and internally calls the gSOAP generated `soap_call_ns1_getQuote()` function. For this purpose, we use the Informix DataBlade Developers Kit (DBDK) that we introduced in Chapter 4, “Extending IDS for business advantages” on page 151.

The DelayedStockQuote service accepts one string parameter for the stock symbol and returns a float data type as a result. Thus, we need to define a UDR which accepts `LVARCHAR` as the parameter and returns `SMALLFLOAT` (Informix SQL equivalent to a C float type). Let us name that UDR `get_stockquote()`.

- a. Start the DBDK BladeSmith and create a new Project by selecting **Project** → **New**.
- b. In the New Project Wizard: page 1, enter the DataBlade name `StockQuotes` in the DataBlade name field and click **Next**.
- c. As the Unique Vendor ID on page 2 enter `IBM`. Leave the remaining fields as is for now. Click **Next**.
- d. On page 3 just select **Finish**.
You should see a file browser like window which shows the `StockQuotes.1.0` project.
- e. From the main menu of BladeSmith select **Edit** → **Insert** → **Routine**.
- f. In the New Routine Wizard: page 1 window, enter `get_stockquote` in the Routine Name field. Click **Next**.

- g. On page 2 choose **C** as the implementation language. Click **Next**.
- h. On page 3 select the correct return type for the routine, which is **SMALLFLOAT**. Do not select the Last argument passed to function returns a value option. Click **Next**.
- i. On wizard page 4 define the one parameter of the `get_stockquote()` UDR by entering the argument name `symbol` first, then selecting the argument type **LVARCHAR** second and finally clicking on **Add**. The wizard screen should look like Example 11-12. Click **Next**.

This page lets you add, remove, or re-order arguments that are passed to the routine you are defining.

A routine can have from 0 to 20 arguments.

Argument Name	Argument Type	Default Value
symbol	lvvarchar	

Argument Name: Argument Type: Default value:

Figure 11-12 Defining the UDR parameter in BladeSmith

- j. On page 5 select the first two options (Routine does not accept NULL arguments and Routine may return inconsistent results) and do not select the remaining ones. Click **Next**.
- k. You can skip page 6 by clicking **Next** and advancing to page 7.
- l. This wizard page (page 7) is important because you need to define on what kind of virtual processor our UDR should run. Because our Web service calls include some blocking network system calls, we need to define the routine as “poorly behaved” and define a virtual processor class. So select the Routine is poorly behaved option and also enter `soapvp` in the Name of user-defined virtual processor class field.

The name of the VP class is the same as the one we configured in “IDS 10 configuration for gSOAP” on page 354.

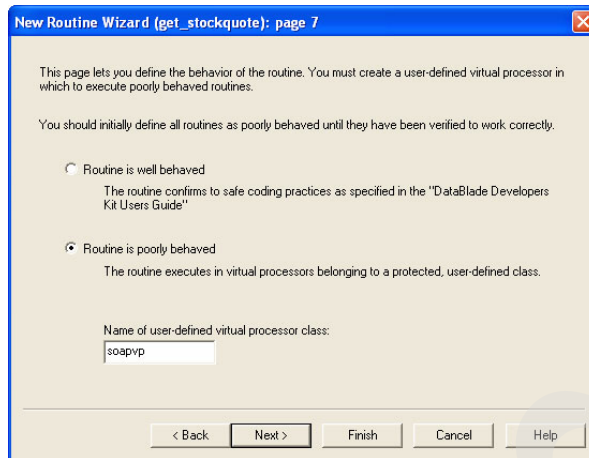


Figure 11-13 Define the custom VP class *soapvp* for the *get_stockquote()* UDR

- m. Now forward through the wizard screens 8 to 11 by clicking **Next**. There is currently no need to change anything there.
- n. While on screen 11 click **Finish**.

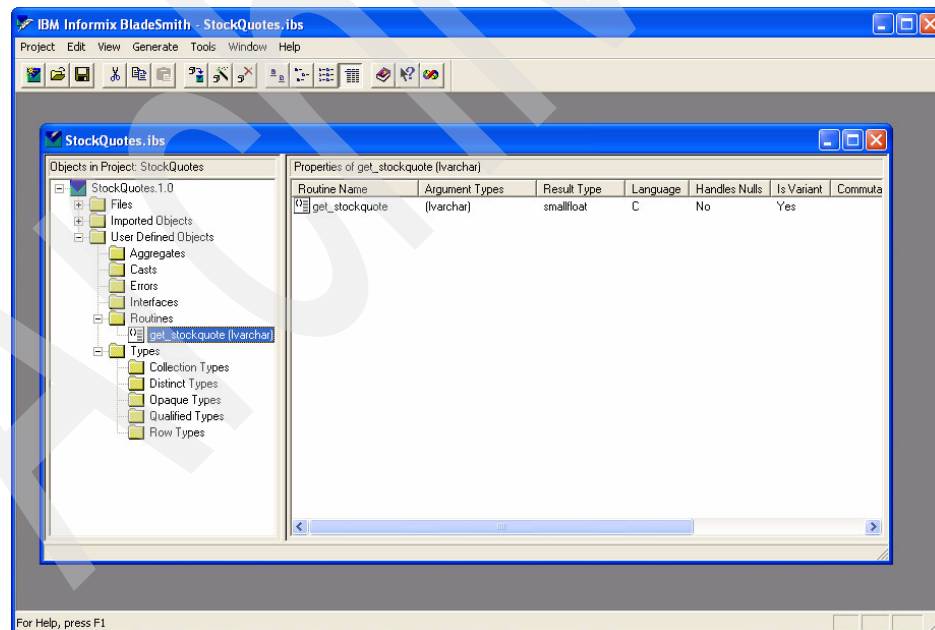


Figure 11-14 The newly defined C UDR *get_stockquote()*

6. We now should save the BladeSmith project into a folder of choice by using the name StockQuotes. The BladeSmith windows should now look like Figure 11-14.
 - a. Up to now, you have defined only the new C UDR, but you have not generated the associated .c, .h and makefiles. To do this, select from the BladeSmith main menu **Generate** → **DataBlade**, and on the Generate DataBlade wizard screen, click **Generate DataBlade**. This step takes a few seconds to finish.

The files and folders have been generated into the same folder where the BladeSmith project has been saved. You can now finish BladeSmith.

Because BladeSmith only generated the necessary C skeleton and SQL registration files, we need to now actually write our C UDR which calls the SOAP function. To do this locate the file `udr.c` in the `src\c` folder, which itself is located in the folder where your BladeSmith project resides.
 - b. Now open the `udr.c` file with an editor of your choice or, for example, within the Visual C++ workbench and apply the following changes:
 - c. Add the required include files (`soapH.h` and the `*.nsmap` file) to the first `ADDITIONAL_CODE/Preserve` Section as shown in Example 11-18.

Example 11-18 Add the include files to `udr.c`

```

/* {{ADDITIONAL_CODE(b9c0d30d-1dc4-11D3-8a74-00c04f79b326)
(PreserveSection) */

/* This area will be preserved when files merge. */
/* Code outside merging blocks will be removed when merging files. */

#include <soapH.h>
#include
"net_x002exmethods_x002eservices_x002estockquote_x002eStockQuoteBinding
.nsmap"

/* }}ADDITIONAL_CODE (#0000) */

```

- d. Put the two variable declarations (struct `soap *` and a float) that you need for the SOAP call into the `Your_Declarations (get_stockquote)` (`PreserveSection`) in `udr.c` as shown in Example 11-19.

Example 11-19 The variable declarations for the SOAP call in `udr.c`

```

/* --- {{Your_Declarations(get_stockquote) (PreserveSection) BEGIN ---
*/

struct soap *soap;

```

```
float result = (float)0.0;

/* --- }}Your_Declarations (#0000) END --- */
```

- e. Remove the lines of code shown in Example 11-20 from the generated udr.c file. Those lines would raise an error message to Informix client application (for example, dbaccess) that the function has not yet been implemented.

Example 11-20 Code to be deleted from the udr.c file

```
/*
** TO DO: Remove this comment and call to
**      mi_db_error_raise after implementing
**      this function.
*/
mi_db_error_raise( Gen_Con, MI_EXCEPTION,
                  "Function get_stockquote has not been implemented." );
```

- f. Now replace the generated C code between the
 /* ---{{Your_Code(get_stockquote)(PreserveSection) BEGIN --- */
 and the /* --- }}Your_Code (#A8DG) END --- */ with the C code from
 Example 11-21.

Example 11-21 The code sequence which call the gSOAP function

```
/* ---{{Your_Code(get_stockquote)(PreserveSection) BEGIN --- */

soap = (struct soap*)mi_alloc(sizeof(struct soap));
if( soap == 0)
{
    DBDK_TRACE_ERROR( "get_stockquote", ERRORMESG2, 10 );

    /* not reached */
}

soap_init(soap);

/*
** Allocate the return value. It must be
** allocated if it is a UDT or type whose
** size is greater than 4 bytes.
*/
Gen_RetVal = (mi_real *)mi_alloc( sizeof( mi_real ) );
if( Gen_RetVal == 0)
{
```

```

/*
** Memory allocation has failed so issue
** the following message and quit.
**
** "Memory allocation has failed in get_stockquote."
*/
DBDK_TRACE_ERROR( "get_stockquote", ERRORMESG2, 10 );

/* not reached */
}

if (soap_call_ns1__getQuote(soap, NULL, NULL,
                           mi_lvarchar_to_string(symbol), &result) == SOAP_OK)
    *Gen_RetVal = result;
else
    mi_db_error_raise(Gen_Con, MI_EXCEPTION, "SOAP Fault");

/* --- }}Your_Code (#A8DG) END --- */

```

g. After applying those changes, save the modified **udr.c** file. You can find a complete version of the modified **udr.c** file Appendix A, “IDS Web service consumer code examples” on page 369.

7. Let us quickly recap what we have done thus far:

- We have created the necessary SOAP skeleton and stub files with the gSOAP tools, based on the given WSDL Web service description.
- Through DBDK’s BladeSmith we defined and generated a simple `get_stockquote()` template C UDR.
- We modified the generated UDR to include our gSOAP generated function call `soap_call_ns1__getQuote()` to execute the remote Web service.

Because we all necessary source code files in place, we only need to compile those files into a shared object (Windows: DLL file) aka DataBlade.

If you have the Microsoft Visual C++ environment installed, the easiest approach is to:

- a. Open the BladeSmith generated `StockQuotes.dsw` Visual C++ Workspace.
- b. Add to the already existing files in the **Source Files** folder in the `StockQuotes` project the following required .c files
 - `soapClient.c` (generated by the `soapcpp2` tool)
 - `soapC.c` (generated by the `soapcpp2` tool)
 - `stdsoap2.c` (located in the `GSOAP_DIR`)

- c. To make sure that the required include files can be found, add the following folders to the StockQuotes C/C++ properties as Additional Include Directories:
 - <GSOAP_DIR>\import
 - <GSOAP_DIR>
 - <DIRECTORY_where_the_gSOAP_files_have_been_generated>
- d. Finally you also need to add the Windows wsock32.lib to the Linker/Input Additional Dependencies property for the StockQuotes project.
- e. Now you can build the DataBlade by using the **Build** → **Build Solution** menu entry from the Visual C++ main menu.

If you want to compile your new DataBlade on a Linux or UNIX platform, then simply modify the BladeSmith generated StockQuotesU.mak makefile to include the files and directories mentioned in the Windows section above, except for the wsock32.lib library, which is only required on a Windows platform.

Tip: If you would like to compile the StockQuotes DataBlade on Linux or UNIX, take a look at an already modified StockQuotesU.mak makefile in Appendix A, “IDS Web service consumer code examples” on page 369. In order to use that makefile, make sure that the environment variable \$TARGET has been set and points to \$INFORMIXDIR/incl/dbdk/makeinc.linux (for the Linux operating system) or to \$INFORMIXDIR/incl/dbdk/makeinc.solaris on the Sun Solaris OS. After setting \$TARGET, simply start the make process by executing **make -f StockQuotes.mak**.

Also, you need to have a copy of gSOAP installed on your Linux or UNIX build machine before you can start the build process.

In any case the generated shared object (the DataBlade) will be named (in our example) StockQuotes.bld.

8. Before we can run and test the compiled DataBlade/UDR within IDS 10 we need to either write a simple SQL script to register the new get_stockquote() routine with the database server or utilize DBDK's BladePack to generate an installation package for the DataBlade.

IDS 10 expects that normally a DataBlade reside within a subdirectory below the dedicated directory for all DataBlades called \$INFORMIXDIR/extend. In our example the assumed location for the StockQuote DataBlade is \$INFORMIXDIR/extend/StockQuotes.1.0. Based on that assumption a simple script to register (activate) the UDR for a given database could look like the one in Example 11-22.

Example 11-22 Simple SQL script to register the get_stockquote() C UDR

```
begin work;  
  
create function get_stockquote (lvarchar) returns smallfloat  
with (class="soapvp")  
external name  
"$INFORMIXDIR/extend/StockQuotes.1.0/StockQuotes.bld(get_stockquote)"  
language c;  
  
grant execute on function get_stockquote (lvarchar) to public;  
  
commit work;
```

As mentioned previously, we could also use DBDK's BladePack to generate a complete DataBlade installation directory. BladeSmith already created the necessary BladePack project file in <DATABLADE_DIR>\Install directory, called StockQuotes.prd. To use BladePack open the StockQuotes.prd file by double clicking it. BladePack will start and the main window should look like Figure 11-15.

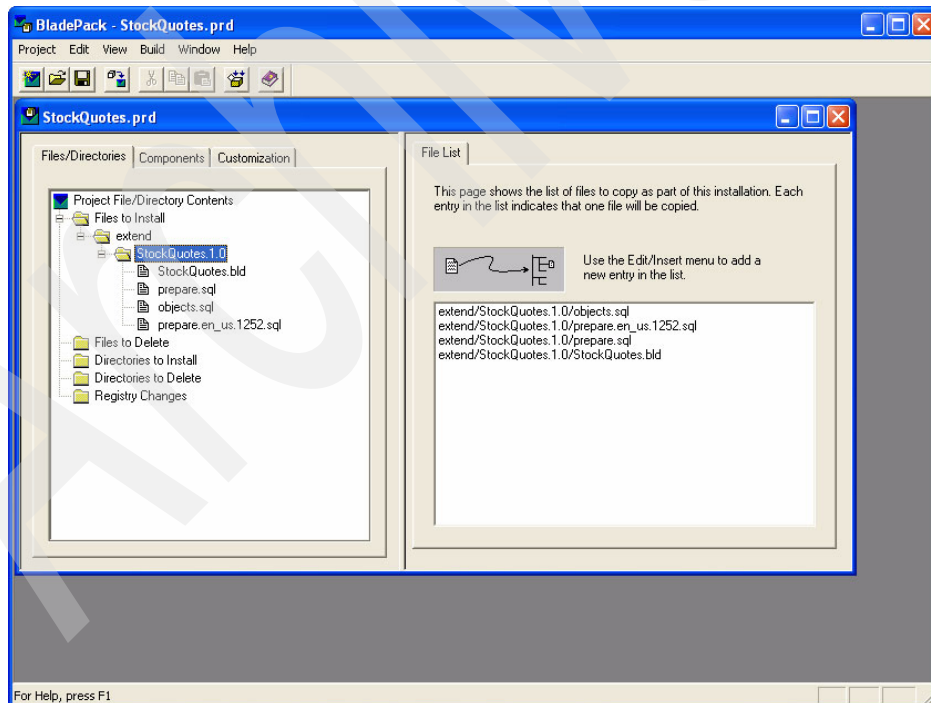


Figure 11-15 DBDK's BladePack with the StockQuotes.prd project open

After starting BladePack with the StockQuotes.prd project file opened, you have the choice what kind of installation package to create and for which deployment platform. It can be anything from a simple folder containing the required files to a very sophisticated installation package which might contain additional files like DataBlade documentation or usage examples.

For our StockQuotes example we will just create a simple, temporary installation folder in the C:\tmp directory. To achieve this, do the following:

- a. With the StockQuotes.prd file opened in BladePack, select **Build** → **Build Installation**. Acknowledge the project file save message with **OK**.
- b. On the Installation Wizard: page 1 select option 2: **Copy files to target directory** because we do not need any installation script or Setup.exe to be generated for our test. Click **Next**.
- c. The Installation Wizard: page 2 lets you select the target operating system for which the installation should be built. For Windows we choose **WinNT/i386** (see also Figure 11-16).

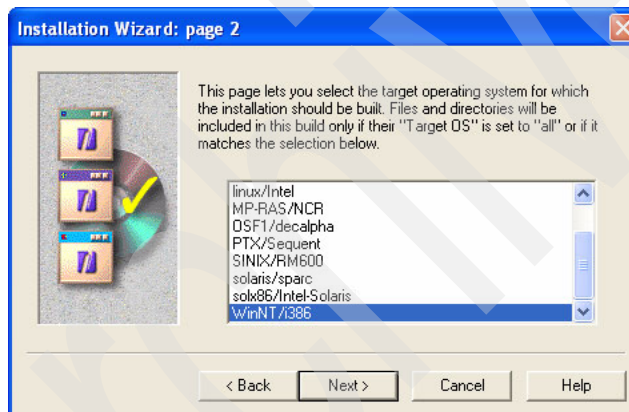


Figure 11-16 BladePack: Installation target OS choices

- d. On Installation Wizard: page 3, you can choose different categories to customize your installation. We skip this wizard by clicking **Next**.
- e. Finally you need to select a staging directory to store the generated installation folder in. Through the Browse option, select the C:\tmp directory (or any other appropriate folder for that purpose). Click **Next**.
- f. Installation Wizard: page 5 allows to bundle multiple BladePack projects together into a single installation. Because we do not use that option for our example, advance to the next page by clicking **Finish**.
- g. On the final page of the Build Installation task, we can now select an installation project and start the actual build by selecting the Build option.

- h. As soon as the build has been completed you should see an Installation Build Output window much like the one in Figure 11-17.

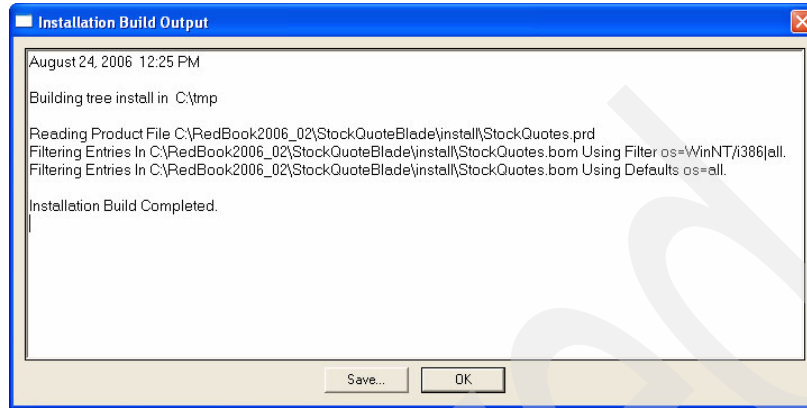


Figure 11-17 BladePack: Installation Build Output window

- i. Now you are ready to register and test run our new `get_stockquote()` UDR in IDS 10.
9. The easiest way to register and to run the new UDR is to copy the BladePack generated `StockQuotes.1.0` folder (located in `C:\tmp\extend` based on the BladePack example above) into the `$INFORMIXDIR\extend` folder.

As soon as the `StockQuotes.1.0` blade folder has been copied, run `BladeManager`, either the command line or the GUI version to register the `DataBlade` with the database of your choice.

Let us take a quick look at the `BladeManager` GUI version which is only currently available on the Windows platform. Assuming that the `StockQuotes.1.0` `DataBlade` has already been copied into the `$INFORMIXDIR\extend` folder and that we want to enable that blade for the `stores_demo` database, just follow the these steps:

- Make sure that your IDS 10 is up and running and that you can connect to your database `stores_demo`.
- Start `BladeManager` on Windows and in the Databases tabbed window select your IDS 10 instance and then the `stores_demo` database. As soon as you select the `stores_demo` database, you should see a list of Available and maybe also some Registered `DataBlades` listed in the `DataBlades` Modules section.
- From the list of available `DataBlades` select the new **StockQuotes.1.0** `DataBlade` and click **Add** and click **Apply** to start the blade registration

process. If everything worked out well BladeManager should look like Figure 11-18.

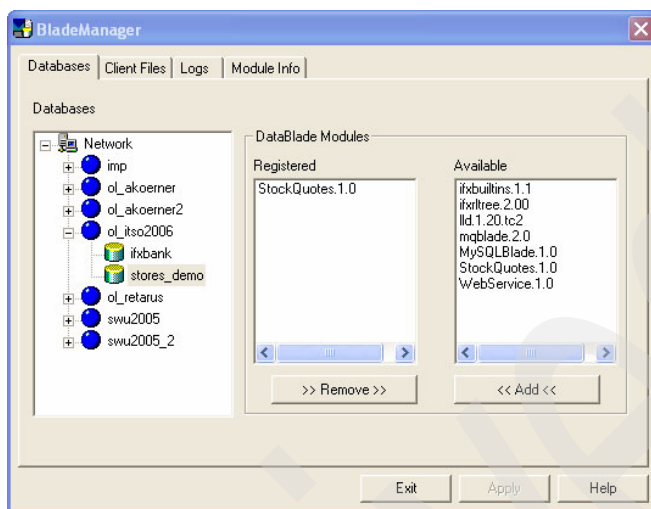


Figure 11-18 BladeManager (GUI version): successfully added the new DataBlade

- d. Click **Exit** to finish and close BladeManager. Now we are ready to test run our StockQuote DataBlade.

To test the `get_stockquote()` UDR, use `dbaccess` to connect to the `stores_demo` database and execute the `get_stockquote()` function a few times with some known stock symbols such as IBM (see Figure 11-19).



Figure 11-19 Test of the `get_stockquote()` UDR in `dbaccess`

IDS Web service consumer code examples

This appendix includes some code samples for the Web service consumer examples in the Chapter 11, “IDS delivers services (SOA)” on page 311.

- ▶ IDS10 / gSOAP Web service consumer:
 - The complete udr.c (Windows and Linux)
 - A complete makefile to generate the example DataBlade on Linux

IDS10 / gSOAP Web service consumer: udr.c

Example: A-1 The complete udr.c file

```
/* {{COMMENT_BLOCK(b9c0d30d-1dc4-11D3-8a74-00c04f79b326) (PreserveSection) */

/*
** Title:          udr.c
** SCCSid:         %W% %E% %U%
** CCid:          %W% %E% %U%
** Author:
** Created:       08/23/2006 14:36
** Description:   This is a generated source file for the StockQuotes DataBlade
module.
** Comments:      Generated for project StockQuotes.1.0
*/

/* }}COMMENT_BLOCK (#0000) */

/* {{WHAT(a9c0d30d-1dc4-11D3-8a74-00c04f79b326) (PreserveSection) */

/*
** The following string is used to enable "what" functionality.
** For more details, see "man what".
*/
static char WhatStr[] = "@(#)StockQuotes.1.0/udr.c(08/23/2006 14:36)";

/* }}WHAT (#COE6) */

/*
** The following is placed here to insure
** that name "mangling" does not occur.
*/
#ifdef __cplusplus

extern "C"
{

#endif

/* Standard library includes. */
#include <ctype.h>
#include <stdio.h>
#include <string.h>
#include <stddef.h>
```

```

/* Used by Informix GLS routines. */
#include <ifxgls.h>

/* Include when accessing the Informix API. */
#include <mi.h>

/* This is the project include file. */
#include "StockQuotes.h"

/* {{ADDITIONAL_CODE(b9c0d30d-1dc4-11D3-8a74-00c04f79b326) (PreserveSection) */

/* This area will be preserved when files merge. */
/* Code outside merging blocks will be removed when merging files. */

#include <soapH.h>
#include
"net_x002exmethods_x002eservices_x002estockquote_x002eStockQuoteBinding.nsmap"

/* }}ADDITIONAL_CODE (#0000) */

/* {{FUNCTION(0f3de26c-42c5-4524-b4c5-0df4f560572a) (MergeSection) */

/* {{COMMENT_BLOCK(get_stockquote)(PreserveSection) Start */

/*****
**
** Function name:
**
** get_stockquote
**
** Description:
**
** Special Comments:
**
** Entrypoint for the SQL routine get_stockquote (lvarchar) returns smallfloat.
**
** A stack size of 32,767 bytes has been requested for
** the routine. Normally, this is sufficient memory for most
** invocations of your UDR. If you intend, however, to call
** this routine recursively or other routines that use large
** or unknown stack sizes, you should use mi_call(). mi_call
** checks to insure that sufficient stack space is available.
** For more details regarding this function, look in:

```

```

** The DataBlade API Programmer's Manual (see Stack Space
** Allocation in Chapter 11).
**
** Parameters:
**
** mi_lvarchar *          symbol
** MI_FPARAM *           Gen_fparam Standard info - see DBDK docs.
**
** Return value:
**
** mi_real
**
** History:
**
** 08/23/2006 - Generated by BladeSmith Version 4.00.TC.
**
** Identification:
**
** NOTE:
**
** BladeSmith will add and remove parameters from the function
** prototype, and will generate tracing calls. Only edit code
** in blocks marked Your_<section>. You can also edit within
** the COMMENT_BLOCK. Any other modifications will require
** manual mergein.
**
*****
*/

/* }}COMMENT_BLOCK (#0000) End */

UDREXPORT mi_real *get_stockquote
(
mi_lvarchar *          symbol,
MI_FPARAM *           Gen_fparam      /* Standard info - see DBDK docs.
*/
)

{
mi_real *             Gen_RetVal;      /* The return value.
*/
MI_CONNECTION *       Gen_Con;         /* The connection handle.
*/

```

```

/* --- {{Your_Declarations(get_stockquote)(PreserveSection) BEGIN --- */

struct soap *soap;
float result = (float)0.0;

/* --- }}Your_Declarations (#0000) END --- */

/* Use the NULL connection. */
Gen_Con = NULL;

/* ---{{Your_Code(get_stockquote)(PreserveSection) BEGIN --- */

soap = (struct soap*)mi_alloc(sizeof(struct soap));
if( soap == 0)
{
    DBDK_TRACE_ERROR( "get_stockquote", ERRORMESG2, 10 );

    /* not reached */
}

soap_init(soap);

/*
** Allocate the return value. It must be
** allocated if it is a UDT or type whose
** size is greater than 4 bytes.
*/
Gen_RetVal = (mi_real *)mi_alloc( sizeof( mi_real ) );
if( Gen_RetVal == 0)
{
    /*
    ** Memory allocation has failed so issue
    ** the following message and quit.
    **
    ** "Memory allocation has failed in get_stockquote."
    */
    DBDK_TRACE_ERROR( "get_stockquote", ERRORMESG2, 10 );

    /* not reached */
}

```

```

        if (soap_call_ns1__getQuote(soap, NULL, NULL, mi_lvarchar_to_string(symbol),
&result) == SOAP_OK)
            *Gen_RetVal = result;
        else
            mi_db_error_raise(Gen_Con, MI_EXCEPTION, "SOAP Fault");

/* --- }}Your_Code (#A8DG) END --- */

/* Return the function's return value. */
return Gen_RetVal;
}
/* }}FUNCTION (#4802) */

#ifdef __cplusplus
}

#endif

```

A makefile to create the StockQuote example on Linux

Example: A-2 The StockQuote DataBlade makefile for Linux

```

# This Makefile builds the StockQuotes DataBlade.
# TARGET must be set to the location/filename
# of the platform-specific make include file.
include $(TARGET)

# This make file assumes a directory structure that is similar to
# the directory structure in which the source files were originally
# generated by BladeSmith. This is:
#
#                               src <- the makefile goes here
#                               /\
#                               ActiveX c
#
# Because the directory structure is the same, files can be copied
# from NT to UNIX (and back) by first NFS mounting your UNIX file
# system and then using Windows NT Explorer to copy the files.
# =====

# {{FUNCTION(a4ad226d-1dcd-11D3-8a74-00c04f79b326) (PreserveSection)

```



```

# This is the project title.
PROJECT_TITLE = StockQuotes

# The linked DataBlade module is placed here.
BINDIR = $(OS_NAME)-$(PLATFORM)

# Platform independent code goes here.
# The following code was generated by BladeSmith.

# GSOAP_DIR points to the gSOAP installation directory
GSOAP_DIR = /home/informix/gsoap-linux-2.7.8c

# STOCKQ points to the folder in which contains the gSOAP generated
# stub and skeleton files
STOCKQ      = /home/informix/RedBook2006/StockQuoteBlade

MI_INCL     = $(INFORMIXDIR)/incl

CFLAGS      = -DMI_SERVBUILD $(CC_PIC) -I$(MI_INCL)/public -I$(MI_INCL)/esql
             -I$(MI_INCL) -I$(GSOAP_DIR)/import -I$(GSOAP_DIR) -I$(STOCKQ) $(COPTS)

LINKFLAGS = $(SHLIBFLAG) $(SYMFLAG)
LIBS      =

# This is a list of the C object files.
PROJECTC_OBJS = \
    $(BINDIR)/support.$(OBJSUFF) \
    $(BINDIR)/udr.$(OBJSUFF) \
    $(BINDIR)/stdsoap2.$(OBJSUFF) \
    $(BINDIR)/soapC.$(OBJSUFF) \
    $(BINDIR)/soapClient.$(OBJSUFF)

# This is a list of the ActiveX server object files.
PROJECTX_OBJS = \

PROJECT_LIBS = $(BINDIR)/$(PROJECT_TITLE).$(BLDLIB_SUFFIX)

all : $(BINDIR)
    if test "$(OS_NAME)" = "hpux" ;\
    then $(MAKE) $(MKFLAGS) -f $(PROJECT_TITLE)U.mak server $(BUILD_TARGET) ; \
    else $(MAKE) $(MAKEFLAGS) -f $(PROJECT_TITLE)U.mak server $(BUILD_TARGET) ; \
    fi

# Construct each object file.

```

```

$(BINDIR)/support.$(OBJSUFF) : c/support.c
    $(CC) $(CFLAGS) -o $@ -c $?

$(BINDIR)/udr.$(OBJSUFF) : c/udr.c
    $(CC) $(CFLAGS) -o $@ -c $?

$(BINDIR)/stdsoap2.$(OBJSUFF) : $(GSOAP_DIR)/stdsoap2.c
    $(CC) $(CFLAGS) -o $@ -c $?

$(BINDIR)/soapC.$(OBJSUFF) : $(STOCKQ)/soapC.c
    $(CC) $(CFLAGS) -o $@ -c $?

$(BINDIR)/soapClient.$(OBJSUFF) : $(STOCKQ)/soapClient.c
    $(CC) $(CFLAGS) -o $@ -c $?

c/udr.c : c/$(PROJECT_TITLE).h

$(STOCKQ)/soapC.c : $(STOCKQ)/soapH.h

$(STOCKQ)/soapClient.c : $(STOCKQ)/soapH.h

$(GSOAP_DIR)/stdsoap2.c : $(GSOAP_DIR)/stdsoap2.h

# Construct the shared library.
# Do *NOT* link with client side libraries. You will see many
# undefined symbols during linking. This is normal since those
# symbols are resolved when the server loads your shared object.
#
# ATTENTION:
# The ld "Symbol referencing errors" warning is normal. These
# unresolved symbols are resolved when the server loads the shared
# object. This list should be examined, however, for symbol names
# that may have been inadvertently misspelled. Misspelled symbol
# names will not be resolved here or at load time. If a version
# 9.20 Informix Server is installed, these symbols are filtered
# by the filtersym.sh script.
#
$(PROJECT_LIBS) : $(PROJECTC_OBJS) $(PROJECTX_OBJS)
    $(SHLIBLOD) $(LINKFLAGS) -o $(PROJECT_LIBS) \
    $(PROJECTC_OBJS) $(PROJECTX_OBJS) $(LIBS) \
    $(DATABLADE_LIBS) 2> link.errs
    if test -x $(INFORMIXDIR)/bin/filtersym.sh ;\
    then $(INFORMIXDIR)/bin/filtersym.sh link.errs ;\

```

```
    else cat link.errs ;
    fi

server :$(PROJECT_LIBS)

clean :
    $(RM) $(RMFLAGS) $(PROJECT_LIBS) $(PROJECTC_OBJS) $(PROJECTX_OBJS)

$(BINDIR) :
    -mkdir $(BINDIR)

# }}FUNCTION (#TG06)
```

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. Pre-calculated and pre-stored 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. 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 some 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. Refer to Enterprise Replication

DDL (data definition language). 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.

DML (data manipulation language). An INSERT, UPDATE, DELETE, or SELECT SQL statement.

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. This is typically to improve the quality, consistency, and usability of the data.

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 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 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.

Debugger. A facility on the Message Flows view in the Control Center that enables message flows to be visually debugged.

Deploy. Make operational the configuration and topology of the broker domain.

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.

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 some 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 database servers.

Extenders. These are 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. It can use native access or 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. An application programming interface that has the same characteristics as ODBC, but is specifically designed for use by Java database applications.

Java Development Kit. Software package used to write, compile, debug, and run Java applets and applications.

Java Message Service. An application programming interface that provides Java language functions for handling messages.

Java Runtime Environment. A subset of the Java Development Kit 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. It is 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 group. Group of one or more database partitions.

Node. An instance of a database or database partition.

ODS. (1) Operational data store: 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.

OLAP. Online analytical processing. Multidimensional data analysis, performed in real time. Not dependent on an underlying data schema.

Open Database Connectivity. A standard application programming interface 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.

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 one that is executed at a remote server.

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. Also see 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	DCE	distributed computing environment
ADK	Archive Development Kit	DCM	Dynamic Coserver Management
API	application programming interface	DCOM	Distributed Component Object Model
AQR	automatic query rewrite	DDL	data definition language
AR	access register	DES	Data Encryption Standard
ARM	automatic restart manager	DIMID	Dimension Identifier
ART	access register translation	DLL	dynamic link library
ASCII	American Standard Code for Information Interchange	DML	data manipulation language
AST	application summary table	DMS	database managed space
BLOB	binary large object	DPF	data partitioning facility
BW	Business Information Warehouse (SAP)	DRDA®	Distributed Relational Database Architecture™
CCMS	Computing Center Management System	DSA	Dynamic Scalable Architecture
CDR	Continuous Data Replication	DSN	data source name
CFG	Configuration	DSS	decision support system
CLI	call-level interface	EAI	Enterprise Application Integration
CLOB	character large object	EBCDIC	Extended Binary Coded Decimal Interchange Code
CLP	command line processor	EDA	enterprise data architecture
CORBA	Common Object Request Broker Architecture	EDU	engine dispatchable unit
CPU	central processing unit	EGM	Enterprise Gateway Manager
CS	Cursor Stability	EJB™	Enterprise Java Beans
DAS	DB2 Administration Server	ER	Enterprise Replication
DB	database	ERP	Enterprise Resource Planning
DB2 II	DB2 Information Integrator	ESE	Enterprise Server Edition
DB2 UDB	DB2 Universal Database™	ETL	Extract, Transform, and Load
DBA	database administrator	FP	fix pack
DBM	database manager	FTP	File Transfer Protocol
DBMS	database management system	Gb	gigabits

GB	gigabytes	LPAR	logical partition
GUI	graphical user interface	LV	logical volume
HADR	High Availability Disaster Recovery	Mb	megabits
		MB	megabytes
HDR	High Availability Data Replication	MDC	multidimensional clustering
HPL	High Performance Loader	MPP	massively parallel processing
I/O	input/output	MQI	message queuing interface
IBM	International Business Machines Corporation	MQT	materialized query table
ID	identifier	MRM	message repository manager
IDE	Integrated Development Environment	MTK	DB2 Migration Toolkit for Informix
IDS	Informix Dynamic Server	NPI	non-partitioning index
II	Information Integrator	ODBC	Open Database Connectivity
IMS™	Information Management System	ODS	operational data store
		OLAP	online analytical processing
ISAM	Indexed Sequential Access Method	OLE	object linking and embedding
		OLTP	online transaction processing
ISM	Informix Storage Manager	ORDBMS	Object Relational Database Management System
ISV	independent software vendor	OS	operating system
IT	information technology	PDS	partitioned data set
ITR	internal throughput rate	PIB	parallel index build
ITSO	International Technical Support Organization	PSA	persistent staging area
IX	index	RBA	relative byte address
J2EE	Java 2 Platform Enterprise Edition	RBW	red brick warehouse
		RDBMS	Relational Database Management System
JAR	Java Archive	RID	record identifier
JDBC	Java Database Connectivity	RR	repeatable read
JDK	Java Development Kit	RS	read stability
JE	Java Edition	SCB	session control block
JMS	Java Message Service	SDK	Software Developers Kit
JRE	Java Runtime Environment	SID	surrogate identifier
JVM	Java virtual machine	SMIT	Systems Management Interface Tool
KB	kilobyte (1024 bytes)	SMP	symmetric multiprocessing
LDAP	Lightweight Directory Access Protocol	SMS	System Managed Space

SOA	service-oriented architecture
SPL	Stored Procedure Language
SQL	structured query
TCB	thread control block
TMU	table management utility
TS	table space
UDB	Universal Database
UDF	user-defined function
UDR	user-defined routine
URL	Uniform Resource Locator
VG	volume group (RAID disk terminology).
VLDB	very large database
VP	virtual processor
VSAM	virtual sequential access method
VTI	virtual table interface
WSDL	Web Services Definition Language
WWW	World Wide Web
XBSA	X-Open Backup and Restore APIs
XML	Extensible Markup Language
XPS	Informix Extended Parallel Server

Related publications

We consider the publications that we list in this section particularly suitable for a more detailed discussion of the topics that we cover in this IBM Redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 389. Note that some of the documents referenced here might be available in softcopy only.

- ▶ *Using Informix Dynamic Server with WebSphere*, SG24-6948
- ▶ *IBM Informix: Integration Through Data Federation*, SG24-7032
- ▶ *Database Transition: Informix Dynamic Server to DB2 Universal Database*, SG24-6367

Other publications

These publications are also relevant as further information sources:

- ▶ *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
- ▶ *J/Foundation Developer's Guide*, G251-2291
- ▶ *R-Tree Index User's Guide*, G251-2297
- ▶ *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
- ▶ *Built-In DataBlade Modules User's Guide*, G251-2770
- ▶ *DataBlade Module Installation and Registration Guide*, G251-2276
- ▶ *Spatial DataBlade Module User's Guide, Version 8.20*, G251-1289

- ▶ *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
- ▶ *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
- ▶ *Read Me First Informix Data Director for Web, Version 2.0*, G251-0512
- ▶ *Geodetic DataBlade Module User's Guide, Version 3.11*, G251-0610
- ▶ *Image Foundation DataBlade Module User's Guide, Version 2.0*, G251-0572
- ▶ *TimeSeries DataBlade Module User's Guide, Version 4.0*, G251-0575
- ▶ *TimeSeries Real-Time Loader User's Guide, Version 1.10*, G251-1300
- ▶ *Guide to SQL: Syntax*, G251-2284
- ▶ *IDS Administrators Guide*, G251-2267
- ▶ *Internals of DBD::Informix, IDUG North America 2006 Education Seminar W13*, Jonathan Leffler

Online resources

These Web sites are also relevant as further information sources:

- ▶ Informix Flat-File Access
http://www-128.ibm.com/developerworks/db2/zones/informix/library/demo/ids_ffvti.html
- ▶ Generating XML from IDS 9.x
<http://www-128.ibm.com/developerworks/db2/zones/informix/library/techarticle/0302roy/0302roy2.html>
- ▶ Using GUIDs with IDS 9.x
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0401roy/index.html>
- ▶ Event-driven fined-grained auditing with Informix Dynamic Server
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0410roy/>

- ▶ Date processing in Informix Dynamic Server
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0510roy/>
- ▶ DataBlade Developer's corner
http://www-128.ibm.com/developerworks/db2/zones/informix/corner_dd.html
- ▶ Downloadable demos
http://www-128.ibm.com/developerworks/db2/zones/informix/library/samples/db_downloads.html
- ▶ Object-relational database extensibility, including datablades
http://www.iug.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-128.ibm.com/developerworks/db2/zones/informix/library/techarticle/0207harrison/0207harrison.html>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Numerics

4GL 316

A

access methods 14, 16, 19, 21
access plans 7–8, 53
ACTIVE 128, 130
ActiveX 197, 374–375
administration 2, 14, 22–23, 26, 35, 37, 39, 64, 67, 69, 281
ADO 193, 197
AIX 26, 30, 178, 180, 207, 246, 276, 296
aliases 249
Alter Fragment 292
Alter fragment 287, 289
Apache 311, 316, 338–339, 344–345
Apache Axis 339
Apache AXIS framework 339
 Data Encoding support 339
 Flexible Messaging Framework 339
 Flexible Transport Framework 339
application programming interface (API) 348
application server 316, 318
architecture 1–3, 10, 17, 25, 39, 56, 159, 178, 193, 311, 337
authentication 27, 30, 65, 243
AXIS 311, 316, 339–340
AXIS WSDL2Java tool 344

B

backup 2, 21–25, 32, 39–40, 47, 50, 52, 58–61, 66–67, 242, 257–271, 273–275, 278–279
backup and recovery 61, 257, 262, 274
backup utility 61, 261
best practices 320
binding 317, 346
Bladelets 21, 162
BladeManager 172, 367–368
BladePack 172, 364, 366–367
BladeSmith 172, 358–359, 361, 363, 365, 372, 374–375
BLOB 36, 111–114, 116–118, 168, 197, 220, 223,

270, 307, 351
B-tree 19, 154, 160
buffer pool 7, 51, 68, 296, 299–301, 307
buffer pools 299, 303, 307
Business logic 317, 336, 338
business-to-business 336

C

C 16, 74, 158, 178, 188, 221, 250, 269, 311, 370
cache 8, 54, 82, 85, 102, 193, 217, 297–298, 300–301
CallableStatement interface 332
CASE Clause 74
CASE clause 74, 76–77
CASE expression 74
case study 138
cataloging 21
challenge-response authentication 251
character strings 103
chunk 59, 66, 261, 264, 272, 283, 295–296, 303, 306
class libraries 315, 342–343
class location 322
CLASSPATH 340–343
client application 336
CLOB 36, 111–114, 116–118, 163, 179–180, 197
Cobol 207
collection 16, 102, 104, 106, 108, 197
collection data type 102–104, 106, 108
collection type 15, 103–104, 106–108, 157
column name 324, 335
Commit 350
configuration 1, 27–32, 34, 39–40, 46, 54, 56, 63, 68, 126–127, 135, 137, 168, 242–243, 249–250, 263–266, 270, 273, 296, 298–300, 303, 307, 339, 341, 353–355
configuration file 63, 126, 270, 341
configuration parameters 298, 307, 341
connection 321–322, 325, 329
connection name 322
cooked files 66
CopperEye Indexing DataBlade 19
cost-based optimizer 7, 13

- create a database 17
- CREATE INDEX statement 109
- CREATE PROCEDURE 334
- CREATE SYNONYM 110, 119
- CREATE TRIGGER 120, 123
- CREATE VIEW 85, 110, 119, 137–138, 141, 143, 148, 227
- Create, Read, Update, Delete (CRUD) 206–207
- customer table 317, 323–325
 - banking customers 317

D

- DAD 326
- DADX 319–321, 324–326, 328, 330–332, 334–335
 - column name 335
 - document access definition extension 319
 - build a DADX Web service 321
 - DADX functions 320
 - generated DADX file 328
 - runtime component 320
 - support for stored procedures 330
 - support for UDR 330
 - documentation xmlns 327, 332–333, 335
 - operation name 333, 335
 - parameter name 327, 332–333
- DADX file 320–321, 325–326, 328, 331
 - operation definition elements 335
- DADX Group
 - Properties pop-up window 326
- DADX group 325–326, 329
- DADX syntax 332–334
- DADX Web service 320–321, 328
 - necessary files 328
 - Web service 321, 324–325, 328, 332
- Data Definition Language 14, 109
 - See DDL
- Data Encryption 221
- data encryption 2, 219
- data movement 41–42, 145, 156
- data perspective 324
- data replication 34
- data type 1, 3, 9, 14–16, 18–19, 36, 44, 57, 68, 74, 102, 106, 108, 111–112, 120, 123, 125, 154, 157, 162, 180–181, 183–184, 197, 200, 270, 324, 337–339, 350
 - user-defined 109
- data type casting 17
- data type support 111, 122, 125

- database administration 69
- database connection 322, 329
- database explorer 322
- database manager 322
- database objects 200, 242
- database operations 3, 6–7, 11, 70, 311
- database server 29, 299, 336, 338, 340
 - data processing 336
- database shared memory 303
- databases 193, 206, 325, 332
- DataBlade 19–21, 55, 161–162, 169–170, 172, 175–186, 337, 354–356, 358, 361, 363–364, 366–370, 372, 374–375
 - DataBlade API 21, 170
- DataBlade Manager 176
- DataBlade modules 19, 21, 175–177, 186
- DataBlades 19–21, 151, 155, 162, 172, 175–177, 180, 182, 353, 364, 367
- DB2 XML Extender 319
- DBD 212–213
- DBI 212–213
- DBMS 14, 74, 77, 97, 190, 215, 241–243, 245–246, 249, 253
- DBSERVERNAME 241, 249–251, 273
- dbspace 26–27, 32–33, 44, 46, 58, 60, 65, 260–261, 269, 274–275, 280, 282–286, 290, 292–293, 295–296, 306–307
- DDL 14, 31, 42–44, 46, 109–110, 119
- default buffer pool 307
- default roles 253
- default value 329
- degree of parallelism 11
- DELETE 78, 110, 112, 118, 122, 131, 328, 330
- Denial of Service 30–31
- deployment 329
- deployment descriptor 339
- development APIs 188, 335
- development tool 207
- Disabling external directive 130
- disk 6–12, 22, 24–25, 29, 32, 43, 46, 51–54, 56, 58–59, 66, 68, 134–135, 262, 265–266, 269–270, 274–276, 283, 295–296, 301, 304, 307, 345
- disk allocation 52
- distinct data type 16
- distinct type 123, 157
- distributed queries 109–112, 125, 132, 137, 144, 146, 150, 243–244
- DML 109–110, 112, 123, 125
- DROP 33–34, 78, 167, 172, 198, 288–289

DROP TABLE 78
DTD 320
dynamic hash join 93
Dynamic Scalable Architecture 1, 3, 51
dynamic SQL 207
dynamic Web project 321, 323

E

EAR 321, 329
Eclipse 151
EGL application 316
EGL code 207, 318
EGL program 206
EGL Web service 316–318
element name 331
element ref 331
embedded SQL 191, 207
ENCRYPTION 220–222, 224, 227, 229–230, 232, 235
encryption with views 227
Enterprise Edition 22–23, 25, 34
Enterprise Generation Language (EGL) 204–207, 311, 316–319
Enterprise Java Beans 315
Enterprise Replication 22, 26, 34, 38, 79, 243
Entity Bean 315
ENVIRONMENT 206–207, 315–316, 335, 343
environment variables 62, 166, 343
error handling 338
ESQL/C 170, 178, 188–189, 202, 210, 213, 216
expression-based 9
extended data type 15, 17, 74, 112
extensibility 14, 25, 151–152, 157, 160–163, 168, 172, 191, 214, 337, 353
 triggers 151
extent 43, 59, 278
external backups 265
External Optimizer Directives 27–28
external routines 26, 29

F

fan-in parallelism 4–5
fan-out parallelism 5, 10
field xpath 331
firewall 352
FIRST 85–96
flexibility 342
Forced Residency 302

fragmentation 8–9, 40, 43, 46, 53, 279, 282–287, 295–296
 Expression 283

G

Geodetic DataBlade 20, 182–183
Graphical Toolkit 213
GROUP BY 12, 27, 75, 124, 134, 137, 140, 156, 164, 168
GUI 172, 213, 367–368
Guidelines 332

H

Hash Joins 132
HDR 22, 25–26, 32, 34, 60, 68, 79, 243, 273, 275
Hibernate 216–217
high availability 34, 79, 273
High Availability Data Replication
 See HDR
High Performance Loader 10, 46
host variable (HV) 323–324, 331
Host variable name 324
HP-UX 30, 178, 180
HTML 185, 209–210
HTTP 209, 314, 320, 327, 332, 335, 337
HTTP protocol 344, 355
HTTP proxy 352
HTTP server 209
Hypertext Preprocessor 209
Hypertext Transfer Protocol (HTTP) 327, 333

I

IBM DB2 319
IBM EGL 205
 compiler 205
IBM Informix 193
 database 193–194
 database development tool 335
 J/Foundation Developer 339
 JDBC 2.21.JC4 driver 334
 ODBC Driver 194
 OLE DB Provider 194
IBM Tivoli 25
IBM WebSphere Application Server 191
IDS 1–4, 6–15, 17–27, 29–31, 33–43, 46, 50, 52–58, 60–66, 69, 73, 76–77, 85, 89, 92, 95, 97–98, 109, 111–112, 118, 121, 125, 132–137, 139–140,

- 142, 144, 146–147, 149–151, 154, 156–163, 165, 168–169, 171–172, 175, 177–180, 182–183, 185–189, 191, 195–196, 200, 206, 209–210, 212, 214–217, 219, 229–230, 238, 241–243, 245, 249–252, 257–258, 261, 265–266, 273–275, 281–284, 291, 295–298, 304, 308, 311, 315–316, 320–323, 326, 328, 330, 332–333, 335, 337–338, 340–341, 344–345, 349, 351, 353–356, 364, 367
 - as a Web service consumer 336
 - DADX Web service 321
 - extensibility 336, 338
 - stores_demo database 322
 - Web service consumer 336–337
 - Web service consumer example 345
 - Web service provider 315
 - with J/Foundation 338
- IDS 10 320, 322, 336–337, 339
 - correct version 340
 - Web services 337, 344
- IDS and WOF 319
- IDS Workgroup Edition 22–23
- IDS-Express 23–24
- INACTIVE 128
- incremental backup 260
- index scans 8
- Indexes 9, 58, 226
- indexes 8–9, 12–14, 19, 21, 24, 33, 47, 51, 54, 58, 68, 92, 108, 153, 226, 242, 278, 280, 284, 295–296, 307–308
- Indexing encrypted data 225
- Informix 1, 3, 8, 15, 17–23, 25–26, 36, 38–41, 47, 50–52, 57–58, 61–63, 65, 67, 69, 98, 106, 109, 111, 135, 140, 160, 165, 169, 182, 188–190, 192–197, 200–202, 204, 207, 210, 212–213, 215–217, 219, 237–238, 250, 261–262, 266, 271, 274–275, 282–283, 296, 315–317, 328, 334–335, 339, 343, 356, 358, 362, 371, 376
 - 4GL 201
- Informix .NET Provider 193
- Informix 4GL 201–202, 204
- Informix C-ISAM 21
- Informix Client Software Development Kit 188
- Informix Connect 188
- Informix Dynamic Server
 - See IDS
- Informix Excalibur Text DataBlade 21
- Informix High-Availability Data Replication 25
- Informix IDS 320, 326
- Informix Image Foundation DataBlade 21

- Informix JDBC 191
- Informix ODBC 194–196
- Informix OLE DB 194, 197
- Informix Open Database Connectivity 195
- Informix Spatial DataBlade 20
- Informix TimeSeries Real-Time Loader 20
- INFORMIXDIR 340–343
- INFORMIXSERVER 62–63, 326
- InformixSOADemo Web project
 - folder 329
- informix-sqli 326
- input message 346
- input parameter 321, 347
- INSERT 79, 97–98, 100–101, 105, 110, 112–114, 116–117, 122–123, 179, 192–193, 199, 324, 327–328, 330
- Insert statement 321, 324
- instance 6–9, 23–26, 30, 34–35, 38–43, 47, 50–56, 59–70, 72, 109, 111–112, 177–178, 217, 249, 257–262, 264–269, 271–273, 275, 327, 340, 348, 351, 354, 367
 - configuration 40, 354
 - creation 52
 - operation 50, 261, 264
 - shared memory 7, 56, 65, 68
- integer variable 87
- Integration 337
- ISOLATION 205
- iterator function 159, 167

J

- J/Foundation 311, 338, 340
- J2EE 205, 216, 316
 - Benefits 205
 - Perspective 325
- jar 322, 328, 340, 342
- JAR file 328, 344, 349, 351
- Java 16, 18–19, 21, 55, 64, 70, 120, 158, 163, 165–168, 184, 190–191, 193, 205, 216–217, 311, 315–317, 328, 335–338, 340, 342–344, 347, 349, 351–353
 - Java classpath 343
 - Java virtual machine (JVM) 338
 - Java Virtual Processors 338
 - Java wrapper UDR 348
- Java bean 315–316
- Java class 347, 349
- Java class files 344

Java classes 339, 344, 347
Java Development Kit (JDK) 338
Java packages 338, 340, 343
Java UDR
 wrapper 344, 349, 356
 wrapper class 344
Java virtual machine (JVM) 338
java.sql 332
Java/J2EE 205
JavaMail 340, 343, 345
JavaSource 328
JAXP 191, 340
JDBC 26, 36, 42, 165, 167, 190, 192, 217, 315,
322, 326, 330, 332, 334, 336
JDBC driver 322, 328, 334
JDK 338, 341, 343
join operation 308
JOIN optimization 144
joins 11, 94, 96, 125, 132, 138, 140, 147, 149, 279,
308
JSP 317
JVM 338, 341

K

Key 339
krakatoa directory 342–343

L

LANG 342, 350
Large Object Locator 178
latency 10, 54
LDAP 65, 195, 241, 251–252
LDAP directory 241
LDAP support 251
LEFT OUTER JOIN 138–139, 141–144, 147
Library 317, 343
LIMIT 82, 85
Linux 2, 6, 23, 30, 41, 62–67, 176, 178, 180, 195,
207, 215, 243, 246, 276, 339, 353, 364, 369, 374
LIST 103–104, 106, 108, 197
load 6, 10, 20, 24, 39, 42–46, 49–50, 61, 67, 184,
214, 376
lock modes 278
log buffers 299
logical logs 32, 39, 47, 54, 59, 61, 67–68, 259–260,
262, 265–269, 271, 273, 275–276, 280, 306

M

memory 3, 7–8, 10–11, 21, 23–27, 32, 34, 39, 51,
55, 62, 65, 68, 92, 132–135, 161, 185, 193, 266,
275, 296–301, 303, 307, 371
memory allocation 132, 134–135
Memory Grant Manager 10–11, 133, 135
memory management 298
memory pools 34
memory usage 56, 297
merge join 308
Metadata 335
metadata 20–21, 25, 314
Microsoft .NET 193
migration process 41–42
Model 205
model 15, 20, 24, 54, 56–57, 110, 181
MQ DataBlade 178–179
MTK 41–42
multiple devices 46
MULTISET 103–104, 106–108, 197
MVC 205

N

Namespace 346
nested loop join 94, 309
NULL values 103, 126, 159

O

object oriented programming languages 205
object types 215
object-oriented 215
object-relational 14, 22, 216
ODBC 190, 193–196, 210, 212, 336
OLAP 11, 57
OLE DB 193, 197
OLTP 1, 7, 10, 56, 260, 281, 337
ON-Bar 24, 261–262, 264–266, 268–270,
272–276, 278
 components 262
oncheck 40, 68, 270–271, 293–294
ONCONFIG 28, 30, 32, 34, 40, 56, 60, 62–63,
65–66, 126–127, 135, 137, 166, 168, 270, 273, 275,
299, 301–302, 307, 341–342, 354–355
ONCONFIG file 30, 126, 135, 302, 308, 340–341,
352, 354
Online Index Build 33
onmode 31, 34, 40, 68, 135, 271, 295, 302
onstat 69, 135, 236, 238, 303, 354

- ontape 27, 32, 261, 265, 267, 274
- opaque type 157, 163, 165
- open standards 336
- operation name 327, 332, 346
- optimization 3, 14, 94–96, 128, 132, 136, 144–145
- optimizer 7–9, 11, 13–14, 28–29, 33–34, 39, 53, 58, 67, 93, 95–96, 125–128, 130, 132, 145, 147, 154, 165, 308–309
 - directives 14, 27–28, 125
- Oracle 42, 97
 - datetime 42
- ORDER BY 27, 75–76, 82, 84–85, 90–93, 95, 134, 136–137, 140, 156, 164, 168
- outer join 139
- output message 346

P

- packages 317, 338, 340, 343, 345
- page 27, 68–69, 82, 84–85, 145, 185, 271, 283, 295–296, 299, 301, 303–307, 313, 317, 321, 358, 366
- page size 27, 284, 296, 303–305, 307
- page template 321
- pagination 81, 83–85, 92
- PAM 27, 30, 195, 241, 243, 250–251
- parallel insert 12
- parallel scan 10–12
- parallelism 4–5, 8, 11–13, 52, 56, 268
- part name 346
- partial trust 244
- partitioning 9, 11–12, 26, 283
- password 221, 227, 229–230, 232, 245–246, 248–249, 326
- password authentication 245, 251–252
- password encryption 249–250
- PATH 316, 328, 341, 347
- PDQ 10–11, 24, 26–27, 68, 135, 284, 291, 295
- PDQ priority level 11
- performance tuning 39
- PERL 212
- perspective 322, 324–325, 328
- PHP 209–210, 212, 335
- Pluggable Authentication Module 30, 241
- Port 346, 348
- PortType 346
- primary key 192, 260
- privileges 32, 64, 130, 213, 252
- processes 3–4, 6, 73, 168, 173, 215, 267, 297, 312,

- 314
- processing costs 236
- programming languages 194, 205
 - IBM EGL 205
 - Java/J2EE 205
 - structured 205
- project 317, 321, 323, 325
- Python 215–216

Q

- query
 - fragments 33, 60, 145, 295
 - optimizer 155
 - rewrite 136
- Query Builder 323, 328
- query fragments 145
- query plan 125, 138, 143, 145–146, 291
- queues 55, 179, 299
- quotations 314

R

- Rational 207, 315, 317, 321
 - Application Developer 321
- Rational Application Developer (RAD) 321, 328
- Rational SDP 318
- Rational Software Development Platform 316
- raw devices 66
- raw disk 10
- read-ahead 8, 12, 301
- recovery 2–3, 21, 47, 50, 59, 61, 257, 260, 262, 265, 273–276
- Redbooks Web site 389
 - Contact us xvi
- referential integrity 10, 43
- reorganizing 32
- REPEATABLE READ 309
- Repeatable Read 28, 308–309
- Replicate Resync 26, 35
- replication 26, 34, 40, 243–244
- restore 22, 24, 26, 32–33, 39–41, 58–61, 257–276, 278–280
- restricting registration 26, 29
- result set 334
- RIGHT OUTER JOIN 139
- Roles 252
- roles 241, 246, 252, 312
- rollback 161
- round-robin fragmentation 9, 53

row type 16, 157
R-tree 19–20, 154, 160, 177
runtime environment 205
runtime framework 316, 320

S

sbspace 340
scalability 1, 3, 8, 22, 24, 54, 186, 191, 262, 338
schema 42–43, 103–104, 112, 138, 206–207, 275–278, 284, 292, 294, 323, 327, 331, 346
scrollable cursors 210
search 82, 84, 182, 336
secure environment check 30
security 2, 22–23, 26, 29, 41, 65, 219, 224, 229, 232, 241–242, 341–343, 352
 permissions 242
SELECT 34, 74–77, 79–81, 85–96, 98, 100–101, 105, 110, 112, 114, 116, 118–119, 121, 124, 129–131, 133, 137–143, 145–149, 153, 155–156, 160, 164, 168, 179, 181, 183, 191, 203, 209, 213, 222–225, 228, 231–232, 237–238, 247, 290–291, 293, 309, 321–323, 327–328, 330
selector xpath 331
Sequences 97
serial data type 324
server 1, 3, 8, 22–23, 25–26, 29, 36, 47, 51, 61–63, 65, 67, 69, 106, 109–111, 120, 135, 160, 165, 169, 182, 185, 191, 197–198, 200, 209, 212, 219, 237–238, 282, 295–296, 299, 316, 318, 322, 376
 configuration 340
 connection 322
service provider 315, 336–337
session bean 315
sessions 100
SET 28, 77, 94, 103–108, 115, 117–118, 121, 123, 130, 197, 220–222, 224, 227, 229–231, 235, 252, 277–279, 308–309, 344
SET ISOLATION 309
setXXX 334
shared library 18, 158, 169–171, 214, 356, 376
shared memory 7, 297–300, 303
size of encrypted data 233
SKIP 81, 85, 87–93, 95
SMART BLOB 116
smart large objects 169, 340
SMP 3, 10
snapshot 84, 108, 128, 296, 303
SOA 178, 311–312, 314, 316

SOA application 316
SOAP 320, 336–337, 339–340, 346, 353, 361, 363, 374
SOAP message 348
Software Development Platform (SDP) 316–318
Spatial DataBlade 20, 180
SPL routines 18
SQEXPLAIN 133, 291
SQL 5, 7–9, 11–14, 17–19, 21, 23–24, 26, 28–29, 31, 39, 43, 47, 54, 56, 69, 73–74, 77, 81–82, 84, 98, 111, 120, 126, 128, 131–132, 146–149, 153, 155–157, 159–160, 164–165, 167–168, 170, 172–173, 178–179, 181, 183–185, 188–192, 195–196, 200–201, 206–208, 216–217, 220, 225, 227, 229–230, 234, 236, 276–277, 279–280, 283, 298, 301, 319, 321, 324–326, 330, 336, 338, 344, 349, 351, 358, 361, 364, 371
SQL builder
 window 324
SQL data 338
SQL level 336–337, 350
 Web services 336
SQL script 344, 349, 351, 356
SQL statement 325, 349
 DADX file 326
sqlhosts file 30, 65, 245, 249–250
SQLJ 338
SQLSTATE 190, 208
SQLtoXML 315
statistical information 13–14
statistics 33, 40, 67, 69, 80, 165, 300
STDIO 27, 32, 274–275
storage manager 25, 262
stored procedure (SP) 18, 44, 55, 163, 210, 320, 326, 330–331, 334, 337–338
Stored Procedure Language (SPL) 332
stores_demo database 87, 195, 322, 351, 367–368
striping 53
Sun Solaris 364
Sybase 42
system catalog 18, 111, 126, 290

T

table space 116
tables 7–10, 13–14, 16, 18–19, 21, 24, 28, 33–35, 40, 42–44, 46, 51, 54, 56–61, 67, 78–79, 82, 92–93, 95, 97, 111, 117–119, 132, 136, 138, 144–145, 147–148, 160, 162, 176, 184, 197, 230–231, 242,

- 258, 260–261, 264, 271, 276–277, 279–280, 282–284, 292–293, 295–296, 308, 323
- target platform 205
- Tcl/Tk 213
- temporary tables 137, 296
- test environment 318
- TEST ONLY 128
- Text User Interface 205
- thread 4, 6–7, 11, 30, 54, 237–238, 264, 291, 295
- Timeseries Real Time Loader 184–185
- Tool Command Language 213
- Toolkit 41, 213
- triggers 40, 47, 79, 278
- TRUNCATE 77–81, 100–101
- TRUNCATE TABLE 78
- Truncation 77
- tuning 3, 6, 39, 50, 55, 62, 67, 72
- two-phase commit 178

U

- UDA 18–19, 156, 159
- UDDI 320
- UDF 17, 152, 163, 165, 176, 178, 220
- UDR 18, 44, 55, 57, 109, 115, 120, 123, 152, 157–158, 160, 162, 165, 167–169, 300, 311, 321, 328, 330, 332–334, 336–338, 343–344, 348–349, 351–352, 354–361, 363–364, 367–368, 371
- UDT 111, 113, 115, 119–120, 122–123, 191, 362, 373
- UNION ALL 137, 140–144
- UNION ALL views 142
- unique name 331
- UNIX 2, 6, 298, 339, 374
- UPDATE 13, 77, 80, 110, 112, 115, 117–118, 121, 123, 130, 222, 224, 231, 256, 300, 328, 330
- UPDATE STATISTICS 80, 300
- URL 320, 326, 343–345
- user authentication 242, 250
- user IDs 245–246
- user interface 205
 - business application programs 205
- user-defined aggregate
 - See UDA
- user-defined function
 - See UDF
- user-defined routine
 - See UDR
- user-defined types 17, 19, 162, 270

UTF-8 327

V

- VARCHAR 335
- verifying backups 269
- video 2, 21
- View Folding 136
- views 136–137, 140, 142, 144, 148, 162, 176, 220, 227, 279
- Virtual Index Interface 19, 161
- virtual processor 338
- virtual processors 4–8, 10, 12, 236–237, 239, 338
- Virtual Table Interface 19, 161
- VisualAge Generator 316

W

- WAS 207
- Web DataBlade 185
- Web Perspective 328
- WEB project 321, 323–324, 328
- Web service 168, 205, 311–312, 314–321, 326, 328–330, 332, 335–337, 339, 344–345, 347–348, 351, 353–356, 359, 363, 369–370
 - consumer requirements for IDS 337
 - IDS as a consumer 336
 - standards 316, 336
- Web services Object Runtime Framework (WORF) 320
- WSDL 318, 329
- WebContent 318, 329
- WebSphere 178, 182, 191, 216–217
- WebSphere Application Server 316, 318
- Windows 2, 6, 10, 23, 26, 30, 41, 62–66, 172, 176–178, 180, 194–195, 214–215, 243, 252, 263, 296, 339, 344, 353, 355, 357, 363, 366–367, 369, 374
 - SQLHOSTS 65
- Wizard 315, 321–323
- WORF 311, 319–320
- Workspace 317, 321
- wrappers 344
- WSDL 318, 320–321, 329, 339, 344–345, 347
- WSDL file 344–345, 347, 355
- URL 345

X

- Xerces 340, 343, 345

XHTML 327, 332
XML 2, 163, 168, 185, 191, 215, 314–315,
319–321, 327, 331, 336, 339–340, 353
 Schema 331, 339
XML Extender 319
XML Schema 331
xml version 327, 345
XMLtoSQL 315
XPath 331
XSD 320–321, 327, 332

Z

Zend Core for IBM 212



Informix Dynamic Server V10 . . . Extended Functionality for Modern Business

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



Redbooks

Informix Dynamic Server V10 ... Extended Functionality for Modern Business

**Enable easy
application
development and
flexible SOA
integration**

**Simplify and
automate IDS
administration and
deployment**

**Realize blazing fast
OLTP performance**

This IBM Redbook provides an overview of the Informix Dynamic Server (IDS), Version 10. IDS provides the reliability, flexibility, and ease of maintenance that can enable you to adapt to new customer requirements. It is well known for its blazing online transaction processing (OLTP) performance, legendary reliability, and nearly hands-free administration for businesses of all sizes—all while simplifying and automating enterprise database deployment.

Version 10 offers significant improvements in performance, availability, security, and manageability, including patent-pending technology that virtually eliminates downtime and automates many of the tasks that are associated with deploying mission-critical enterprise systems. New features speed application development, enable more robust enterprise data replication, and enable improved programmer productivity through support of IBM Rational development tools, JDBC 3.0, and Microsoft .NET as examples. Version 10 provides a robust foundation for e-business infrastructures with optimized Java support, IBM WebSphere certification, and XML and Web services support.

Ready for service-oriented architecture (SOA)? This IBM Redbook also includes descriptions and demonstrations of support that are specific to IDS for an SOA.

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