Information Management software

IBM

Informix Dynamic Server 11: Advanced Functionality for Modern Business

- **Agile: Adaptable, flexible, and with blazing fast speed**
- **Invisible: Small, embeddable, and an administration free zone**
- **Resilient: Reliable, highly available, and secure**

Carlton Doe
Ajay Gupta
Jacques Roy
Keshava Murthy
Alexander Koerner
Chuck Ballard
Randy House
Vijay Lolabattu
Karl Schimmel
Richard Snoke

ibm.com/redbooks

International Technical Support Organization

Informix Dynamic Server 11: Advanced Functionality for Modern Business

October 2007

SG24-7465-00

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

Second Edition (October 2007)

This edition applies to Version 11.10 of the IBM Informix Dynamic Server.

Contents

Notices ix
Trademarks x

Preface xi
The team that wrote this book xii
Become a published author xvi
Comments welcome xvi

Chapter 1. IBM Informix Dynamic Server 11 essentials 1
1.1 Informix data 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 Leveraging the strengths of DSA 10
1.1.5 An introduction to IDS extensibility 15
1.2 Informix Dynamic Server editions and functionality 23
1.2.1 Informix Dynamic Server Express Edition (IDS-Express) 25
1.2.2 Informix Dynamic Server Workgroup Edition (IDS-WGE) 25
1.2.3 Informix Dynamic Server Enterprise Edition (IDS) 27
1.3 New features in Informix Dynamic Server 11 28

Chapter 2. Fast implementation 37
2.1 How to get to IDS 11 38
2.1.1 In-place upgrades 38
2.1.2 Migrations 41
2.2 The installation process 41
2.2.1 Bundles, products, and components 41
2.2.2 Manifest files and reinstallation 42
2.2.3 Installation modes 42
2.2.4 Preliminary installation steps 43
2.2.5 Choosing components 44
2.2.6 Installing with the graphical installer for Windows 45
2.2.7 Installing with the graphical installer on UNIX or Linux 50
2.2.8 Installing in console mode on Linux or UNIX 51
2.2.9 Silent installation 54
2.2.10 Using the product installation scripts 61
2.2.11 Using the JAR file on Linux or UNIX 61

2.3 Adding or removing components 62
2.4 First steps after installation 64
2.4.1 The environment and user profiles 64
2.4.2 The configuration file 64
2.4.3 The SQLHOSTS file 65
2.4.4 Starting the IDS instance 65
2.5 Administration and monitoring utilities 66
2.5.1 Command-line utilities 66
2.5.2 The OpenAdmin Tool for IDS 68
2.5.3 The Server Studio Java Edition (SSJE) 69

Chapter 3. Configuration 73
3.1 Reasons for changes 74
3.1.1 Performance considerations 74
3.1.2 Administration considerations 75
3.2 Recovery time objective 75
3.2.1 Recovery 76
3.2.2 Configuring for performance 78
3.3 Checkpoints 79
3.3.1 Traditional checkpoints 79
3.3.2 Fuzzy checkpoints 80
3.3.3 Current issues 81
3.3.4 Changes to checkpoints 81
3.3.5 RTO checkpoints 81
3.3.6 Managing checkpoints 81
3.4 CPU Virtual Processors 82
3.4.1 An overview 82
3.4.2 Traditional methods of managing CPU VPs 83
3.4.3 Making changes 84
3.5 Dynamic logging 85
3.5.1 Managing logical logs 86
3.5.2 Traditional method for managing logical logs 87
3.5.3 Dynamic management of logical logs 88
3.6 AUTO_* Parameters 89
3.6.1 AUTO_LRU_TUNING 89
3.6.2 AUTO_AIOVPS 90
3.6.3 AUTO_CKPTS 90
3.6.4 AUTO_REPREPARE 90
3.7 Distributed Relational Database Architecture 91
3.7.1 An overview 91
3.7.2 Configuring DRDA 92
3.7.3 Managing a DRDA environment 92

3.8 New parameters or variables 93
3.8.1 Onconfig parameters. 93
3.8.2 Environment variables. 98
3.8.3 Deprecated parameters 99

Chapter 4. Enhancements in high availability 101
4.1 A review of existing IDS availability solutions 103
4.1.1 Backup and restore 103
4.1.2 Disk mirroring 103
4.1.3 High Availability Data Replication 104
4.1.4 Enterprise Replication 105
4.2 New high availability solutions in IDS 11 106
4.2.1 Remote Standalone Secondary server 106
4.2.2 Shared Disk Secondary servers 108
4.2.3 Continuous Log Restore 109
4.3 Failover of primary server and recoverable groups 110
4.3.1 Recoverable Groups 111
4.3.2 Recoverable Groups based on the SDS server 113
4.3.3 Recoverable Groups with multi-layer HA solutions on IDS 11 117
4.4 Combining IDS availability options 119
4.5 Conclusion. 121

Chapter 5. Data privacy. 123
5.1 Authentication approaches 124
5.1.1 Connection authentication choices 124
5.1.2 Using Pluggable Authentication Modules (PAMs). 130
5.1.3 Using an LDAP directory. 131
5.2 Authorization 132
5.2.1 Role-Based Access Control 132
5.2.2 Label-Based Access Control. 136
5.3 ENCRYPTION. 148
5.3.1 Scope of encryption 149
5.3.2 Data encryption and decryption 149
5.3.3 Retrieving encrypted data 152
5.3.4 Indexing encrypted data 153
5.3.5 Hiding encryption with views 155
5.3.6 Password usage 157
5.3.7 Encrypting passwords during transmission. 157
5.3.8 The number and form of passwords 158
5.3.9 Password expiration 159
5.3.10 Where to record the passwords 161
5.3.11 Using password hints 161
5.3.12 Determining the size of encrypted data. 162

5.3.13 Errors when the space is too small . . . 163
5.4 Overall security policy . . . 164

Chapter 6. Easy administration . . . 167
6.1 Backup and recovery changes . . . 168
6.1.1 Backup and restore history . . . 168
6.1.2 Backup and restore strategies . . . 174
6.1.3 New features for ontape . . . 178
6.2 Session properties . . . 179
6.2.1 Define session properties . . . 180
6.2.2 Configure session properties . . . 181
6.2.3 Modifying session properties . . . 182
6.3 Improvements in statistics collection . . . 183
6.3.1 Create index distribution implementation . . . 184
6.3.2 Improved sampling size . . . 185
6.3.3 Improving SET EXPLAIN . . . 186
6.3.4 Update statistics improved tracking . . . 187
6.3.5 Temp table statistics improvements . . . 187
6.4 ONMODE utility . . . 188
6.4.1 Description of using onmode options . . . 189
6.4.2 Modifying config parameters with the engine offline . . . 190
6.4.3 Modifying config parameters with engine online . . . 190
6.4.4 Examples of using onmode -wm and onmode -wf . . . 191

Chapter 7. The administrative API . . . 193
7.1 The sysadmin database . . . 194
7.1.1 Database administration in previous IDS versions . . . 195
7.1.2 Remote administration of databases . . . 196
7.1.3 Examples of task() and admin() functions . . . 196
7.2 The Scheduler . . . 199
7.2.1 Tasks . . . 200
7.2.2 Sensors . . . 200
7.2.3 Startup tasks . . . 202
7.2.4 Startup sensors . . . 202
7.2.5 Examples of task and sensor . . . 204
7.3 Command line interface for database administration . . . 206
7.3.1 Database administration by using the command line . . . 206
7.3.2 Database administration by using SQL and stored procedures . . . 208
7.3.3 Syntax examples . . . 213

Chapter 8. SQL Query Drill-Down . . . 217
8.1 Understanding the SQL Query Drill-Down feature . . . 218
8.1.1 SQLTRACE . . . 219
8.1.2 SQL trace using the administrative API functions . . . 222

8.2 How to display and analyze the trace information 227
8.2.1 onstat -g his 227
8.2.2 Using syssqltrace 233
8.2.3 Using syssqltrace_info 237
8.2.4 Using syssqltrace_iter 238
8.2.5 SQL Query Drill-Down from the IDSAdmin console 239
8.3 Summary 240

Chapter 9. SQL Language 241
9.1 Hierarchical data type (Node DataBlade) 243
9.2 Basic Text Search index 246
9.3 Binary data types 250
9.4 WebSphere MQ messaging in IDS applications 252
9.5 XML publishing and XPath functions 257
9.6 Enhanced concurrency 267
9.7 Improved concurrency 270
9.8 Automatic repreparation of SQL statements 270
9.9 Named Parameters support for JDBC 273
9.10 Full support for subqueries 275
9.11 Enhancements to distributed queries 276
9.12 Trigger enhancements 280
9.13 Index self-join access method 283
9.14 Optimizer directives 286
9.15 Statistics collection and query explain 287
9.16 Stored procedure language (SPL) enhancements 291
9.17 SYSDBOPEN() and SYSDBCLOSE() procedures 294
9.18 Disabling logging for temporary tables 296
9.19 New functions and expressions 296

Chapter 10. Functional extensions to IDS 301
10.1 Installation and registration 303
10.2 Built-in DataBlades 304
10.2.1 Indexable binary data types 305
10.2.2 Basic text search 305
10.2.3 Large Object locator 306
10.2.4 The MQ DataBlade 306
10.2.5 Node 308
10.2.6 Web Feature Service 309
10.3 Free-of-charge DataBlades 310
10.3.1 Spatial 310
10.4 Chargeable DataBlade modules 312
10.4.1 Excalibur text search 312
10.4.2 Geodetic 313

10.4.3 TimeSeries . . . 314
10.4.4 TimeSeries Real Time Loader . . . 314
10.4.5 Web . . . 315
10.5 Conclusion . . . 316

Chapter 11. Development tools, interfaces, and SOA integration . . . 317
11.1 IDS 11 software development overview . . . 318
11.1.1 IBM supported APIs and tools for IDS 11 . . . 318
11.1.2 Embedded SQL for C (ESQL/C): CSDK . . . 318
11.1.3 The IBM Informix JDBC 3.0 Driver: CSDK . . . 320
11.1.4 IBM Informix .NET Provider: CSDK . . . 323
11.1.5 IBM Informix ODBC 3.0 Driver: CSDK . . . 325
11.1.6 IBM Informix OLE DB Provider: CSDK . . . 327
11.1.7 IBM Informix Object Interface for C++: CSDK . . . 329
11.1.8 IBM Informix 4GL . . . 330
11.1.9 IBM Enterprise Generation Language (EGL) . . . 334
11.1.10 IBM Informix Embedded SQL for COBOL (ESQL/COBOL) . . . 336
11.2 Additional tools and APIs for IDS 11 . . . 338
11.2.1 IDS 11 and PHP support . . . 338
11.2.2 PERL and DBD::Informix . . . 341
11.2.3 Tcl/Tk and the Informix (isqltcl) extension . . . 342
11.2.4 Python, InformixDB-2.2, and IDS 11 . . . 344
11.2.5 IDS 11 and the Hibernate Java framework . . . 345
11.2.6 IDS 11 and WAS CE . . . 345
11.2.7 IDS 11 support for Ruby and Rails (Ruby on Rails) . . . 346
11.2.8 IDS 11 software development direction . . . 347
11.3 IDS 11 and SOA integration . . . 347
11.3.1 SOA foundation technologies in IDS 11 . . . 348
11.3.2 Service providing with IDS 11 . . . 348
11.3.3 Service consumption with IDS 11 . . . 349

Glossary . . . 353

Abbreviations and acronyms . . . 359

Related publications . . . 363
IBM Redbooks publications . . . 363
Other publications . . . 363
How to get IBM Redbooks publications . . . 364
Help from IBM . . . 364

Index . . . 365

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)  ®	Architecture™	MQSeries®
developerWorks®	DB2 Universal Database™	MVS™
pureXML™	DB2®	Rational®
pSeries®	DRDA®	Redbooks®
AIX®	Illustra™	System p™
BladeCenter®	Informix®	Tivoli®
DataBlade™	IBM®	UC2™
Distributed Relational Database	IMS™	WebSphere®

The following terms are trademarks of other companies:

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

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

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

ActiveX, Microsoft, Visual Basic, Visual C#, Visual J#, Visual Studio, Windows NT, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

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

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

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

Preface

In this IBM® Redbooks® publication, we provide an overview of Informix® Dynamic Server (IDS) 11. We briefly introduce the technological architecture, which supports all versions of IDS, and then describe in more detail several of the new features available in IDS 11. Most of these features are unique in the industry today, enabling clients to use information in new and more efficient ways to create business advantage.

IDS is designed to help businesses better leverage their existing information assets as they move into an on demand business environment. In this type of environment, mission-critical database management applications typically require a combination of online transaction processing (OLTP) and batch and decision-support operations, including online analytical processing (OLAP). And, IDS offers capabilities to minimize downtime and to enable a fast and full recovery if an outage occurs.

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

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

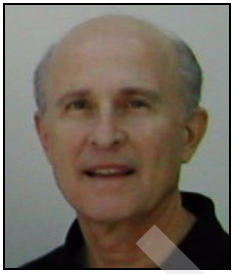
Built on the IBM Informix Dynamic Scalable Architecture (DSA), it provides one of the most effective solutions available, including a next-generation parallel data server architecture that delivers mainframe-caliber scalability; manageability and performance; minimal operating system overhead; automatic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

IDS delivers proven technology that efficiently integrates new and complex data directly into the database. It handles time-series, spatial, geodetic, Extensible Markup Language (XML), video, image, and other user-defined data side by side with traditional data to meet today's most rigorous data and business demands. It also helps businesses lower their total cost of ownership (TCO) by leveraging 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 team that wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center. The team members are depicted below, along with a short biographical sketch of each:



Chuck Ballard is a Project Manager at the International Technical Support organization, 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 had over 10 years of Informix experience as a DBA, 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). He served as IIUG President, Advocacy Director, and sat on the IIUG Board of Directors for several years. Carlton is best known for having written the books *Informix-OnLine Dynamic Server Handbook* and *Administering Informix Dynamic Server on Windows NT*, both of which are currently undergoing revisions for a second edition. His most recent book, *IBM Informix Dynamic Server 11, The Next Generation in OLTP Data Server Technology*, was just published by IBM. He is also the author of numerous IBM white papers and technical articles and co-authored several other IBM Redbooks publications. Carlton is currently a Senior Certified Consulting IT Specialist working as the Informix Technical Specialist for the Americas Geographies. He lives in Dallas, TX.

Ajay Gupta is the Software Architect for Backup/Restore, High Availability, and Data Replication Technology. He has held several technical leadership positions in the Development and QA organizations working on all major components of the IBM Informix Dynamic Server. Before joining IBM, he worked in the Operating Systems Group of the National Center for Software Technology (NCST), a premier R&D organization in India. Ajay holds a Masters of Technology (M.Tech) Degree in Computer Science from the Indian Institute of Technology (IIT), Bombay, India.

Randy House is an Advanced Support Engineer specializing in the Informix Dynamic Server and has worked with Informix for over 10 years. Randy holds a Bachelors degree and a Masters degree in Computer Science and a Ph.D. in Engineering Management from the University of Missouri at Rolla. He is located in Lenexa, KS.

Alexander Koerner is a Certified Senior IT-Specialist in the IBM German Channel Technical Sales organization, based in Munich, Germany. Joining Informix in October 1989, he was instrumental in starting and leading the SAP/R3 on Informix project and has contributed to the success of many strategic projects across the region. Alexander currently focuses on Informix and SOA integration, but also actively supports IDS, XML, and DataBlade™ technology. His activities include contributions to several IBM Redbooks publications, presentations at conferences and events, such as the IBM Information OnDemand 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.

Vijay Lolabattu is an Advisory Software Engineer in Advanced Informix support, currently working as Premium Support Analyst for SAP/Informix clients and a major banking client, and is located in Menlo Park, CA. He joined Informix in 1997 and has 13 years of industry experience. Vijay has a Bachelors degree in Commerce from India and a Masters degree in Computer Sciences from Jackson State University in Mississippi.

Keshava Murthy is the Architect of the IBM Informix Dynamic Server SQL and Optimizer components. He has worked on Sybase, Illustra™, and Informix database servers. Keshav has developed features in the SQL, RTREE, distributed queries, heterogeneous transaction management, and extensibility components of IDS. Keshav has advised and enabled IBM clients and partners - big and small - to design, develop, and deploy enterprise and embedded applications using IDS. He received an outstanding technical achievement award for enabling Manhattan Associates' logistics application on IDS. Keshav holds a Bachelors degree in Computer Science and Engineering from the University of Mysore, India.

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 user group meetings.

Karl Schimmel is a Certified IT Specialist in the US ChannelWorks group and is located at the Informix Lab in Lenexa, Kansas. After graduating from Wichita State University with a Bachelor of Science degree in Computer Science in 1993, he started work for Informix in technical support. He spent the majority of his time in the Down System and Diagnostics group of Advanced Support for the core database engine. Karl transitioned into a pre-sales engineering role eight years ago, where he predominately works with partners from small SMB to large OEM accounts.

Richard Snoke is a Senior Certified IT Specialist in the ChannelWorks group in the United 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.

Other Contributors:

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

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

From IBM Locations Worldwide

Lynette Adayilamuriyil - Software Engineer, IDS Kernel Development, Lenexa, KS.

Srini Bhagavan - Architect/Manager, IM Application Enablement, Lenexa, KS.

Judy Burkhart - Software Engineer, IDS Information Development, San Francisco, CA.

David Desautels - Senior Software Engineer, Informix Database Security, Lenexa, KS.

Cindy Fung - Software Engineer, IDS Product Management, Menlo Park, CA.

James Gebhard - Software Development Engineer, IDS DataBlades Development, Schaumburg, IL.

Fred Ho - Software Development Engineer, IDS Product Management, Menlo Park, CA.

Jonathan Leffler - IDS Security Architect, IBM Informix Database Engineering, Menlo Park, CA.

John Miller III - Senior Technical Staff Member, Worldwide Informix Support, Menlo Park, CA.

Pat Moffatt - Program Manager, Education Planning and Development, Markham, ON Canada.

Bingjie Miao - Software Development Engineer, IDS Development, Portland, OR.

Manoj Mohan - Software Engineer, Informix Database Security, Lenexa, KS.

Vinayak Shenoi - Software Engineer, IDS Development, Menlo Park, CA.

Sitaram Vemulapalli - Software Development Engineer, IDS Development, Menlo Park, CA.

From the International Technical Support Organization, 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 Redbooks publication dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

- Use the online **Contact us** review IBM Redbooks publication 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

1

IBM Informix Dynamic Server 11 essentials

IBM Informix Dynamic Server 11 (IDS 11) continues a long-standing tradition within IBM and Informix of delivering first-in-class data servers. It combines the robustness, high performance, availability, and scalability needed in modern business today.

Complex, mission-critical database management applications typically require a combination of online transaction processing (OLTP) and batch and decision-support operations, including online analytical processing (OLAP). Meeting these needs is contingent upon a data server that can scale in performance as well as in functionality. It must dynamically adjust as requirements change from accommodating larger amounts of data, to changes in query operations, to increasing numbers of concurrent users. The technology must be designed to efficiently use all the capabilities of the existing hardware and software configuration, including single and multiprocessor architectures. Finally, the data server must satisfy users' 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 data server architecture that delivers mainframe-caliber scalability, manageability, and performance; minimal operating system overhead; automatic distribution of

workload; and the capability to extend the server to handle new types of data. With Version 11, IDS increases its lead over the data server landscape with even faster performance, a new suite of business availability functionality, greater flexibility and performance in backing up and restoring an instance, automated statistical and performance metric gathering, improvements in administration, reducing the cost to operate the data server, and more.

IDS delivers proven technology that efficiently integrates new and complex data directly into the database. It handles time-series, spatial, geodetic, Extensible Markup Language (XML), video, image, and other user-defined data side by side with traditional data to meet today's most rigorous data and business demands. IDS helps businesses to lower their total cost of ownership (TCO) by leveraging 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 many years of widespread use in critical business operations, which attests to its stability, performance, and usability. IDS 11 moves this already highly successful enterprise relational data server to a new level.

In this IBM Redbooks publication, we briefly introduce the technological architecture that supports all versions of IDS, and then we describe in greater detail many of the new features available in IDS 11. Most of these features are unique in the industry. With Version 11, IDS continues to maintain and accelerate its lead over other data servers on the market today, enabling clients to use information in new and more efficient ways to create business advantage.

1.1 Informix data 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. It is available on many hardware platforms and, because the underlying platform is transparent to applications, the data server 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.

Data server architecture is a significant differentiator and contributor to the server's performance, scalability, and ability to support new data types and processing requirements. Almost all data servers available today use an older technological design that requires each database operation for an individual user (for example, 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 data 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 data 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. Each component of DSA is discussed in the following sections of this chapter.

1.1.1 DSA components: Processor

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

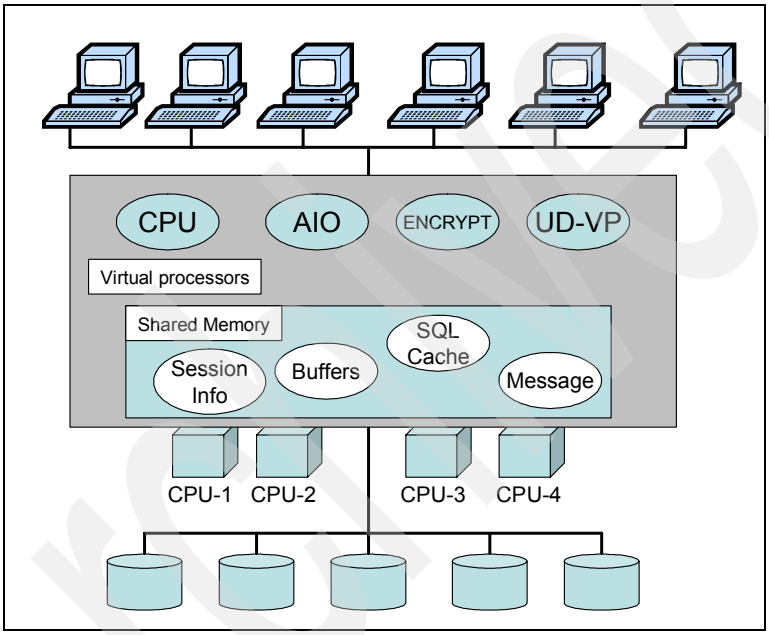


Figure 1-1 IDS pool of VPs for parallel task execution

A *thread* represents a discrete task within a data 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 occurs and the Virtual Processor immediately works 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

enables an efficient use of hardware resources and is referred to as *fan-in parallelism*. It is illustrated in Figure 1-2.

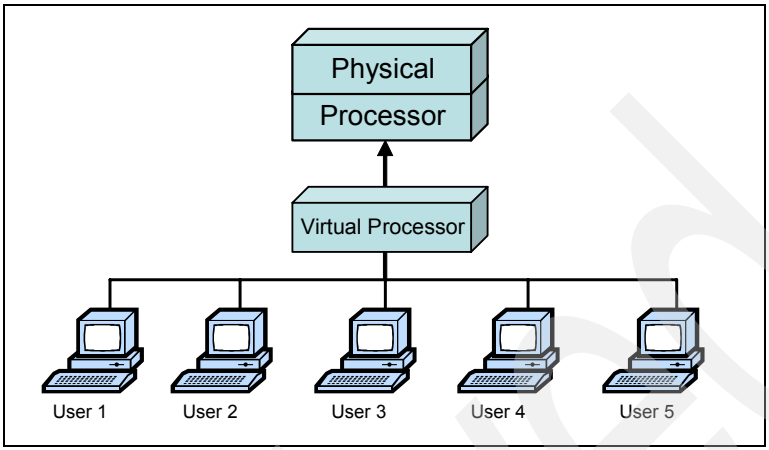


Figure 1-2 Fan-in parallelism

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 data server divides the task into multiple subtasks and then spreads these subtasks across all available Virtual Processors. Fan-out parallelism uses many VPs to process a single SQL operation. 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.

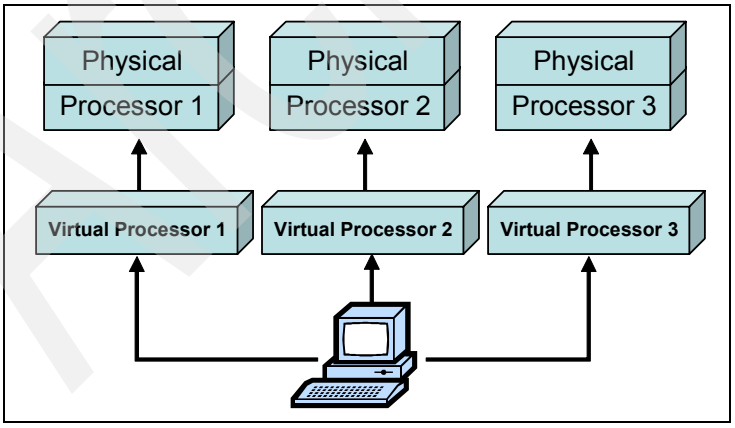


Figure 1-3 Fan-out parallelism

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

Dynamic load balancing occurs within 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 of which is optimized for a particular function, such as CPU operations, disk I/O, communications, and administrative tasks, as illustrated in Figure 1-4.

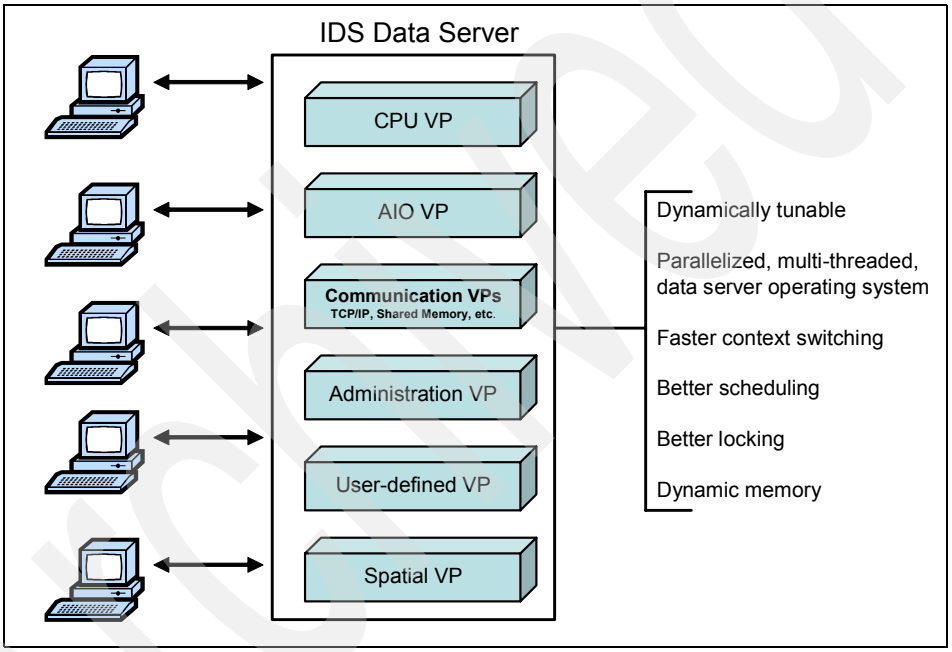


Figure 1-4 VPs are grouped into classes and optimized by function

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. With IDS 11, the data server in certain cases automatically adjusts the mix of VPs to meet the current workload.

In UNIX and Linux systems, the use of multi-threaded Virtual Processors significantly reduces the number of UNIX and 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 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, the IDS data server does not need as much hardware power to achieve comparable to better performance than other data servers. In fact, real-world tests and client experiences indicate IDS only needs between 25% - 40% of the hardware resources to meet or exceed the performance characteristics of single-threaded or process-based data servers.

1.1.2 DSA components: Dynamic shared memory

All memory used by 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 queried and modified data, sort, join and group tables, and lock pointers, as examples. What is unique to IDS is that if database operations require more (or less) shared memory, additional segments are dynamically added and dropped 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 *bufferpool* is used to hold data from the data disk supply during processing. When users request data, the data server first attempts to locate the data in the instance's bufferpool to avoid unnecessary disk I/Os. Depending on the characteristics of the instance workload, increasing the size of the bufferpool 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 bufferpool 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 bufferpool 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 first-in, first-out (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, although with slightly different variable conditions, such as customer number and invoice number.

Each time an SQL operation executes, the data 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 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 through which to look. When joining data from several tables, the optimizer evaluates which table provides the data to which the others join 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 where data I/O plans can be stored for reuse 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 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 data 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 (VPs) are available to service I/O requests:

- Logical log I/O
- Physical log I/O

- Asynchronous I/O
- 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. With this release, the IDS data server also dynamically creates and removes certain types of I/O VPs to facilitate instance operations by reducing the amount of administrative monitoring.

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 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 is required, it completes quickly because the partitions are 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 certain 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 that are 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 (LOBs), 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 need to be fragmented into simple blobspaces. However, because they are black-box 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 does not need to 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 improve 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 and index partitions where the data is known to reside or must be placed. The benefit is that multiple operations can execute simultaneously on the same table, each in its unique partition, resulting in greater system performance than typical data servers provide.

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. They also allow IDS to use direct memory access when writing data. On Windows-based platforms, this access is not possible so standard file system files are used.

With IDS 11, if the Linux or UNIX operating system permits, the data server bypasses the system's I/O buffers and writes directly to *cooked* or operating system flat files that support chunks and dbspaces. With this feature enabled, you can approach the same I/O performance levels as with raw space. Given that all operating systems (OSs) fragment files on disk based on block size, the data pages within a chunk are not contiguous as with raw space. For many clients, that is an acceptable trade for the ability to manage IDS storage space with normal OS disk tools.

1.1.4 Leveraging the strengths of DSA

With an architecture as robust and efficient as 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 (such as tapes, disk files, pipes, or other

tables, as examples) 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. You can configure an HPL job 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. You can also use the HPL 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 that is manipulated and the database design. While many simple OLTP operations, such as single row inserts, updates, and deletes can be executed without straining the system, a properly designed database can leverage 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 online analytical processing (OLAP)-oriented, than operational, or online transaction processing (OLTP)-oriented. With PDQ enabled, not only is a complex SQL operation divided into a number of subtasks, but the subtasks are given higher or lower priority for execution within the data server'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 data server 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 executes 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 data operations, to scan in parallel or have multiple operations executing simultaneously across the table or index can provide a significant performance boost.

With full, integrated parallelism, IDS can simultaneously execute several tasks that are required to satisfy an SQL operation. As data is 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.

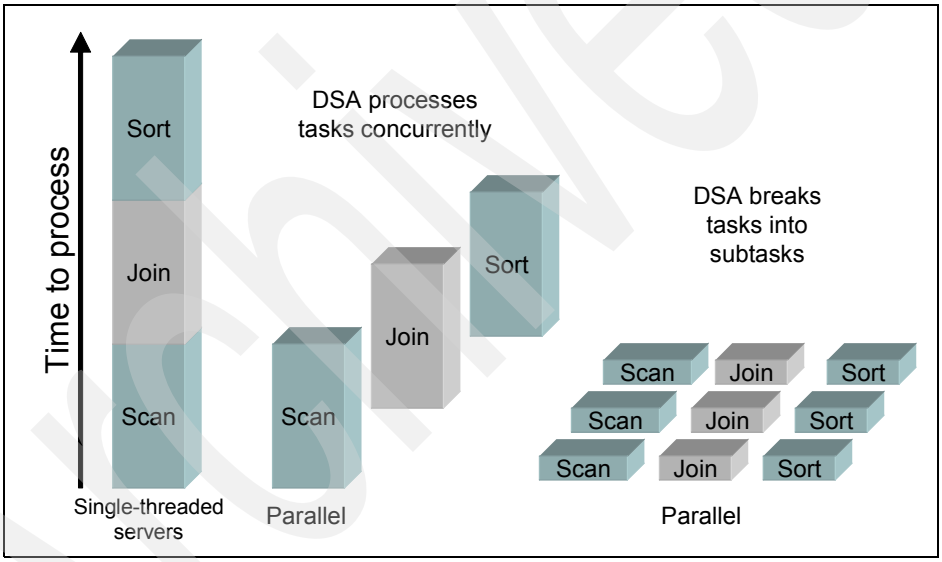


Figure 1-5 IDS integrated parallelism

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 completes.

Like 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 data servers, index building can be a time-consuming process. 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.

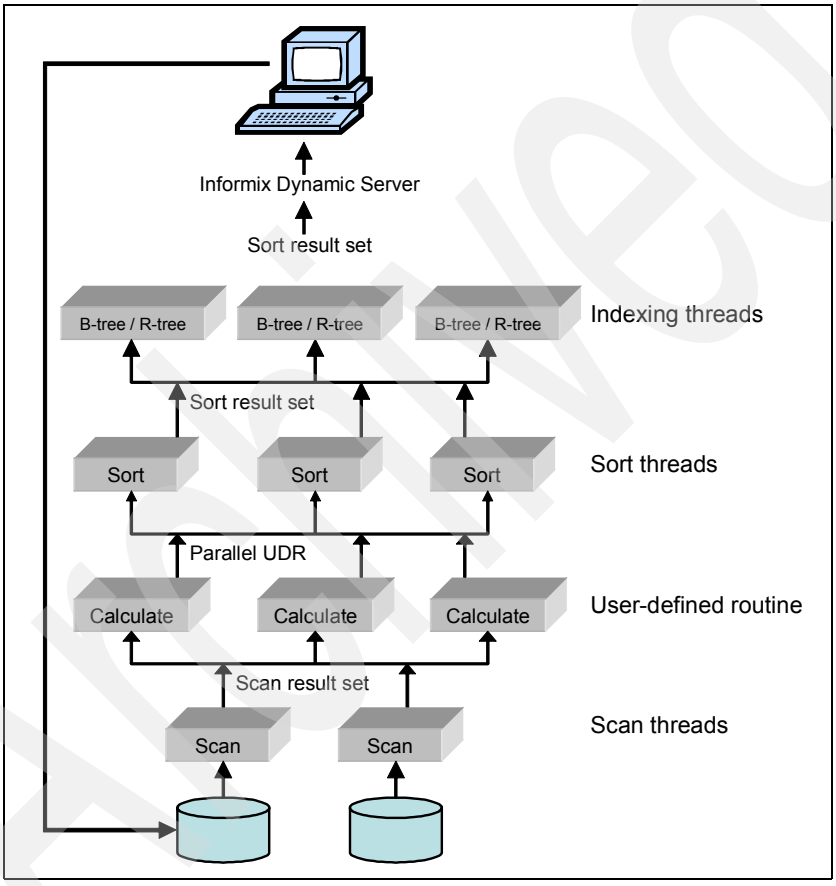


Figure 1-6 IDS parallelism reduces index build and maintenance time

An additional benefit in IDS 11 is that as part of the index build process, statistical information about the index and its lead attributes is gathered and fed to the query optimizer. After the index build completes, the optimizer can efficiently use it rather than wait for an administrator to execute an additional preparatory step.

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

IDS uses a cost-based optimizer to determine the fastest way to retrieve data from database tables and 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 provides 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 that developers and engine administrators can evaluate the effectiveness of their application and 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 stops 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 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. You can set optimizer directives 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. You can also set an optimizer directive to optimize a query to retrieve only the N rows of the possible result set.

This release of the data server adds additional optimizer statistical functionality. In addition to the automatic gathering of statistics as indexes are created, administrators can now more precisely control the amount of data scanned to produce the statistics. They can either specify the number of rows or a percentage of the table to read. Statistical information about explicitly generated temporary tables, including their indexes, is automatically captured as the temporary tables are created and used. From an application development

perspective, more statistical information is available for SQL operations. Rather than an overview of the entire operation, iterator-level diagnostics are available to help precisely locate performance problems in the operation.

1.1.5 An introduction to IDS extensibility

IDS provides a complete set of features to extend the data server, including support for new data types, routines, aggregates, and access methods. This technology recognizes and stores standard character and numeric-based information. Also with this technology, the data server 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 certain types can be table-like in and of themselves, the data 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 (DBMSs) 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, where the object-relational technology is built into the DSA core and can be used, or not, at will within the context of a single data environment.

Data types

IDS uses a wide range of data types to store and retrieve data as illustrated in Figure 1-7 on page 16. The breadth and depth of the data types available to the database administrator and application developer are 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.

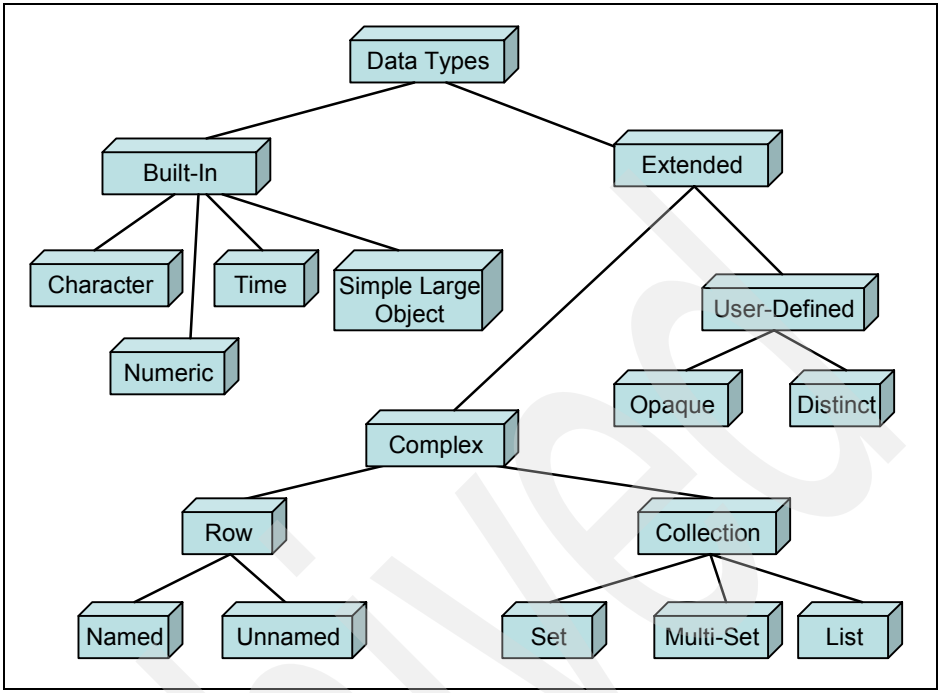


Figure 1-7 The IDS data type tree

Certain 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 *lvarchar*.

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 IDS data 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 normally require multiple rows or redundant columns in one or more tables in a traditional relational-only data server. The three collection types enforce rules about whether duplicate values are significant or data order is significant. Collection data types can be nested and contain almost any type, built-in or extended.

With *row* data types, you can build a new data type that is comprised 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 a data type. When defined, row types can be used as columns within a table or as a table. With certain restrictions, a row type can be dynamically defined as a table is created or can be inherited into other tables as illustrated in Figure 1-8.

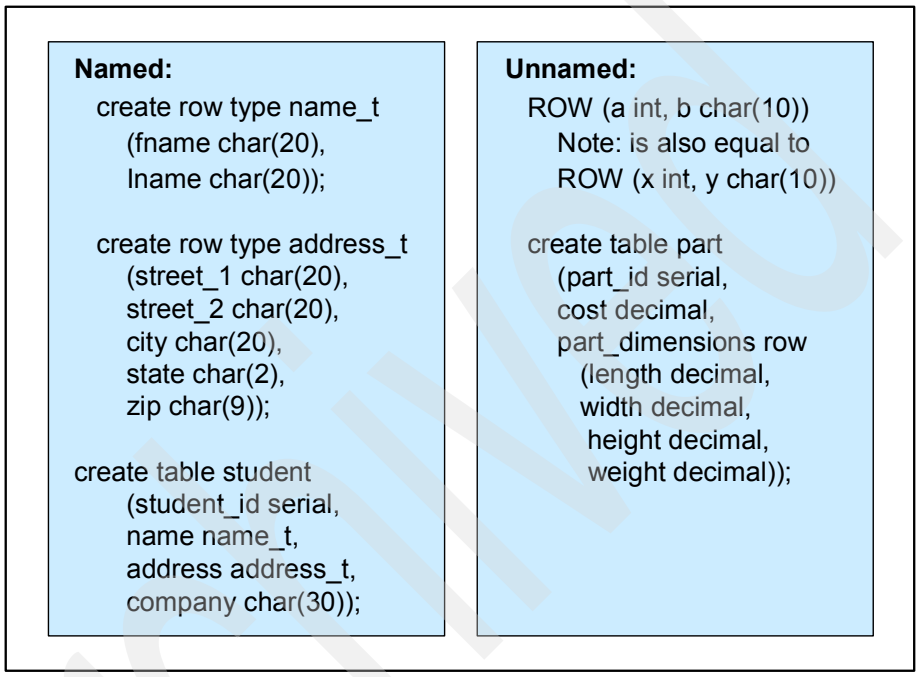


Figure 1-8 Examples of named and unnamed row types

A distinct data type is an alias for an existing data type. A newly defined distinct data type inherits all of the properties of its parent type (for example, a type defined using a *float* parent inherits the elements of precision before and after the decimal point). However, 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 data 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 data 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 data 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 are built into the DSA, 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 IDS data 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 (UDFs).

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 a while; however, they have been hidden in the data 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 data 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 certain 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 that 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 data server only had stored procedures with which to work. 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 data 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 data 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 data 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.

You can create an extension of the SUM aggregate 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, requires creating (and registering) a user-defined function that overloads the PLUS function and takes 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 these functions:

- 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 that are created and registered for UDAs can be written in SPL, C, or Java. Like built-in aggregates, the data server 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. You can create additional user-defined access methods to access other data sources. IDS has methods that provide SQL access to data in either a heterogeneous database table or an external sequential file or to other nonstandard data stored anywhere in the network. You can define secondary access methods to index any data, as well as alternative strategies to access SLOs. You can create these access methods by 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 data 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 and routines. With IBM Informix DataBlade modules, you can manage almost any kind of information as a data type within the data server.

With IDS 11, several DataBlades are bundled as part of the data server enabling application developers to quickly and easily enrich their applications. These Blades include:

- Binary DataBlade: The Binary DataBlade provides the ability to use two new *indexable* extensible data types: *binary18* and *binaryvar*. The binary18 data type is a fixed-length data type that holds 18-bytes. Because this data type is fixed in length, unused space is right-padded with zeros until the column length reaches 18. The binaryvar data type is a variable-length type that can hold up to 255 bytes of information. Built-in functions permit the execution of bit-wise AND, OR, XOR, and COMPLEMENT (also known as NOT) operations, as well as standard COUNT, DISTINCT, MAX, and MIN operations. However, character-oriented operations, such as LIKE or MATCHES, are not supported.
- Basic Text Search DataBlade: This DataBlade expands the text string matching capabilities of the data server through the creation of a specialized index that supports proximity (for example, are two words within N words of each other) and fuzzy text searches. More robust text search functionality is available through the IBM Informix Excalibur Text DataBlade.
- Node: The Node DataBlade is a fully supported version of technology that has been available for download from several IDS-oriented Web sites. It permits the accurate and native modeling of hierarchical data, such as networks, biological or botanical kingdoms, or your company's organizational chart. With it, you can address operations that require complicated and expensive set processing or recursive SQL operations easily.

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

The current list of available IBM Informix DataBlade technologies is available at:

`http://www.ibm.com/informix`

This is a partial list of available IBM Informix DataBlade technologies:

- IBM Informix TimeSeries DataBlade: This DataBlade provides a better way to organize and manipulate any form of real-time, time-stamped data. Use this DataBlade for applications that use large amounts of time-stamped data, such as network analysis, manufacturing throughput monitoring, or financial tick data analysis. You can achieve measurably better performance and reduced data storage requirements with this DataBlade than by using traditional relational database design, storage, and manipulation technologies.

- IBM Informix TimeSeries Real-Time Loader: A companion component 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: These DataBlades 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 an airline, cruise, or military environment. The IBM Informix Spatial DataBlade is a set of routines that are compliant with open geographic information system (GIS) standards, which take a *flat-earth* perspective to mapping geospatial data points. This DataBlade is based on routines, utilities, and ESRI technology. For more information about ESRI technology, go to:

  `http://www.esri.com`

  For this reason, 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, which results 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 all properly licensed users of IDS.
- IBM Informix Excalibur Text DataBlade: This 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: This 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: The 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: This DataBlade provides two separate pieces of functionality to the storage and use of Indexed Sequential Access Method (ISAM)-based data. In environments where data is stored in its native flat-file format, the DataBlade provides data server-based SQL access to the data. From a user or application developer perspective, it is as though the data resides in standard database tables. The second element of functionality enables the storage and retrieval of ISAM data in or 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 continues 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-based, C-based, and SPL-based DataBlades and the DataBlade application programming interface. The DataBlade API is a server-side C API for adding functionality to the data server, as well as for managing data connections, server events, errors, memory, and processing query results. Additional support for DataBlade module developers includes the IBM Informix Developer Zone, which is 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 download 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 data server technology as dynamic and flexible as DSA, it is only natural to assume that clients can buy just the level of IDS functionality that they need. IBM has packaged IDS into three editions, each tailored from a price and functionality perspective to a specific market segment. Regardless of which edition you purchase, each edition comes with the full implementation of DSA and its unmatched performance, reliability, ease of use, and availability, depending on bundle-driven hardware, connection, and scalability restrictions. Table 1-1 on page 24 contains a brief comparison of the three editions and their feature sets.

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, mid-sized companies.	Large enterprises
Function	This is a full-function, object-relational data server that includes important capabilities such as high reliability, security, usability, manageability, and performance. It includes self-healing manageability features and near-zero administration. It allows integration into Rational® Application Developer and Microsoft Visual Studio®, support for transparent silent installation, and support for a wide array of development paradigms. It has minimal disk space requirements and a simplified installation.	This edition includes all of the features of IDS Express plus features to handle high data loads, including Parallel Data Query, Parallel backup and restore, and 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) and High Availability Data Replication (HDR).
Customizable	Installation sets common defaults.	Installation offers greater flexibility.	This edition supports the greatest flexibility to allow tailoring the product to meet the most demanding environments.

	Express Edition	Workgroup Edition	Enterprise Edition
Scalable	2 CPUs/4 GB RAM maximum.	For V10: 4 CPU, 8 GB memory maximum. For V7.31 and 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 Marketing Representative, IBM Business Partner, or this Web site for the full details:

http://www-306.ibm.com/software/data/informix/ids/ids-ed-choice/

1.2.1 Informix Dynamic Server Express Edition (IDS-Express)

Targeted toward 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 two 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 full support for a wide range of development languages and interfaces. It cannot be used, however, to support Internet-based application connections. For those people with little data server skill, the installation process can be invoked to not only install the data server but also to configure an operational instance with a set of default parameters.

1.2.2 Informix Dynamic Server Workgroup Edition (IDS-WGE)

Informix Dynamic Server Workgroup Edition (IDS-WGE) is for any size business needing additional power to process SQL operations, efficiently manage extremely large databases or build a robust, fail-over system to ensure database continuation in the event of natural or human-caused 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. However, its ability to process more complicated SQL operations on larger databases is enhanced because of the PDQ and MGM components that we discussed in the "Parallel Data Query and Memory Grant Manager" on page 11 are available for use. With PDQ and the MGM, data server resources can be pre-reserved and 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, which further increases performance.

Managing large databases is not just an 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. You can use the HPL that we discuss in the "The High Performance Loader" on page 10 to execute bulk data load and unload operations. It uses the DSA 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, which increases system uptime.

Backing up (or restoring) large databases or a specific subset of the data requires very powerful and flexible backup and restore functionality. IDS-WGE includes the OnBar utility suite with its ability to perform partial or full instance backups and restores. In certain 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. You can send the output to multiple backup devices to reduce the amount of time that is required to create a backup or to perform a restore.

Through the OnBar utility suite and the appropriate third-party tape management software, instance or logical log backups can be incorporated into the management software that handles all the other backups occurring across the enterprise. Using this management software, you can use 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. 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, which relieves the IDS administrator of another task.

For data centers that 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 four locally connected backup devices to be used by the OnBar 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, you can execute full or partial restores as necessary.

Clients that want to provide continuation of database services in the event of a natural or human-made outage can purchase and use the Informix High Availability Data Replication (HDR) option 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 standby server. When in standby 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 number of reporting applications, offloading their execution to the mirror server can provide a measurable performance improvement to day-to-day operations.

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

HDR is extremely simple to set up and use. Consider HDR as required technology for operations that need robust data availability.

For clients using IDS Enterprise Replication (ER) technology to distribute and consolidate data throughout their environment, IDS-WGE servers can be leaf or target nodes of replication events. They cannot, however, publish data out to other participants in the replication cluster.

1.2.3 Informix Dynamic Server Enterprise Edition (IDS)

Informix Dynamic Server Enterprise Edition (IDS) includes the full feature set of the data server. IDS can be deployed in any size environment that requires the richest set of functionality that is 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 on

which it is installed. With its renowned stability and extensibility, IDS is the perfect data 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 have received and guarantees delivery of data even if the replication target is temporarily unavailable. Data flow can be either unidirectional 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 7 instance on Windows can be replicated to an IDS 11 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, which gives a business the ability to protect itself from outages as well as automatically migrate data either for application partitioning or distribution and consolidation purposes.

This edition also supports the newly released Continuous Availability add-on module as well and the bundled enhancements to HDR. Using this technology, you can build an availability fabric that can seamlessly withstand disk, server, and network failures.

1.3 New features in Informix Dynamic Server 11

In this section, we briefly introduce the new features in IDS 11 that provide the extended functionality needed for modern business. IDS 11 provides significant advances in data server security, performance, availability, replication, administration, and applications. We briefly summarize a few of these features in Table 1-2 on page 29 and then discuss them.

Table 1-2 New features in IDS 11

Security	Performance	Administration	Application development	Replication and high availability
Label-based Access Control	Interval checkpoints	Recovery time objective	XML publishing	ER enhancements
Common criteria certification	Direct I/O to cooked space	OpenAdmin Tool for IDS	Concurrent optimization	HDR enhancements for continuous availability
Sysdbopen()/ sysdbclose() functions	Index self-join	Automated statistics gathering	Multiple triggers on events	N/A
Backup/restore filters	N/A	Instance administration mode	Derived tables	N/A

Replication and high availability

In many respects, the functionality included here is the crown jewel of this release. While the past few versions of IDS have included enhancements to ER technology, HDR technology in IDS 11 has been enhanced with the ability to create multiple layers of secondary copies of the primary server. These layers can be added or removed depending on network, server, or disk environments and the level of protection that you want to build into your business continuity plans.

The first of the new server types is not really a part of HDR technology, rather it is an extension to the ontape utility and the OnBar suite. But we include it here, because it forms part of the availability fabric. Called a Continuous Log Restore (CLR) server, it supports the intermittent application of completed logical log records from the primary to create a near-line copy. You can use this technology in environments where network connectivity is not constant or the available throughput is too slow to support any other kind of replication technology.

Remote Standby Secondary (RSS) servers are full copies of the primary, but they are maintained asynchronously as opposed to the synchronous nature of communication between the primary and the HDR secondary regardless of its replication mode. This feature is not just 1 to N HDR secondary, though. An RSS instance behaves slightly differently from the typical HDR secondary instance. First, you can have as many RSS instances as you like. An RSS instance cannot be promoted to become an HDR primary. It can, however, become an HDR secondary after which it can be promoted to primary if needed. Finally, RSS instances do not recognize the value of the DRAUTO parameter. As a result,

RSS servers must be considered disaster recovery instances, not high availability (HA) instances.

RSS instances are deployed in order to expand the real-time failover capability of an HDR environment. In this case, the HDR secondary might be adjacent to the primary to mitigate against transient or temporary primary failure. One or more RSS instances can be deployed in more remote locations to guard against a complete primary site failure. Another use is to provide promotable redundancy. In the event of a primary failure, the HDR secondary assumes the primary role but is vulnerable to a failure. An RSS instance can be promoted to the HDR secondary to protect the new primary instance. A third benefit to an RSS instance is where the one and only failover server must be located geographically distant from the primary and the network latency or throughput is too great to support normal HDR replication.

The last of the expanded HDR server types is called the Shared Disk Secondary (SDS) server. As the name implies, one or more instances use the same network-mounted disk logical unit numbers (LUNs) as illustrated in Figure 1-9.

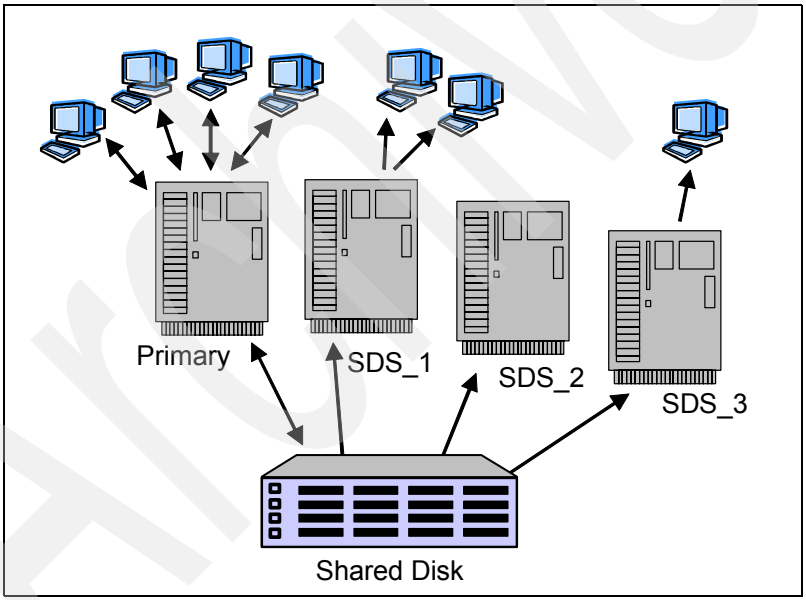


Figure 1-9 A conceptual view of an SDS environment

SDS instances provide redundancy and failover options if the disk LUNs are network-mounted, because they can be anywhere in the network. The weakness to SDS instances is that they do not protect against disk-related failures. Therefore, you either have to manage this within the disk array or with a combination of HDR secondary and RSS instances.

All of these instances, except for CLR, can be used for read-only operations while the availability cluster is operational. The instances are not required to be isolated from each other, which competitive technologies require.

In addition to providing availability and disaster recovery, the instances can also participate in Enterprise Replication, further expanding your options for building a completely failure-resistant environment.

There are several functional enhancements to the ER technology. Previously, modification of certain configuration parameters required a service interruption; that is no longer necessary. Operations, such as TRUNCATE TABLE and the renaming of database objects, is now supported, as well as the ability to indicate whether local triggers fire on changes made as result of replication.

Performance

A number of features were built into IDS 11 to increase its OLTP throughput. Not all of the features are listed in this or other sections of this chapter. Together, however, they contribute to significant double-digit increases in performance based on TPC-C-type tests.

Direct I/O to cooked spaces has already been discussed, but this release incorporates a new checkpoint algorithm, the *interval checkpoint*. *Checkpoints* are operations that are conducted by the data server to insure the logical consistency of the data. Periodically, the contents of instance memory that contain new or modified data are flushed to disk. In the past, this operation has been triggered by several conditions, generally speaking, the passage of N minutes as defined by the IDS administrator.

In earlier versions of IDS, when a checkpoint occurred, DML operations were suspended for the duration of the checkpoint for the sorted disk writes to occur. In certain environments, this interruption, though transient, took longer than desired so administrators tried to work around the interruption by forcing the memory buffers to flush to disk outside of checkpoints. While this approach reduced checkpoint duration, it had an impact on instance processing.

With interval checkpoints, the service interruption as been virtually eliminated. Now, memory writes occur in the background allowing user transactions to continue. Because interval checkpoints no longer interfere with DML operations, administrators can reduce the amount of forced memory flush operations allowing those resources to support user operations. The net result is a significant improvement in the number of transactions that can be processed over time.

Another performance improvement occurred in the SQL optimizer and its ability to rewrite SQL operations. Many times, indexes are created on tables that have a

significant number of duplicate values. In cases such as these, as the optimizer evaluates query plans, indexes are not used, because they do not provide enough discrimination between values; the entire index has to be scanned so the optimizer opts for a light sequential scan of the table.

In IDS 11, the optimizer now recognizes this condition and rewrites the SQL predicate to join the table to itself by using the index to create the data discrimination that it needs. Using this approach, sequential or full index scans are avoided and results are returned more quickly.

Security

In the area of security enhancements, there were several improvements to help businesses, such as yours, meet any regulatory requirements. The first is the addition of Label-Base Access Control (LBAC), which permits you to design and implement multi-level data access and control labels and policies. These labels and policies are enforced regardless of the method that is used to access the data. Policies can be as simple (public and private) or as complicated as needed for your environment.

With this functionality, a new data server role, DBSECADM, was created. Membership in this role permits the creation and modification of labels and policies, therefore, it must be tightly controlled. The SET SESSION AUTHORIZATION SQL function, which enables a user to temporarily assume another user ID, has been restricted as well.

Another enhancement is the ability of the instance to automatically invoke a specific UDR whenever a session connects or disconnects from the instance. While IDS 10 permitted the IDS administrator to declare default roles for a user whenever that ID connected to the instance, that was the extent of what was enforced. All other actions, such as setting PDQPRIORITY and other session-related parameters, had to occur within the application.

You can now write a series of UDRs that are executed when a user session connects and disconnects from the instance. These routines can perform any functionality, including setting roles, specifying session operating parameters, such as PDQPRIORITY, setting the isolation level or the output location of optimizer reports (also known as sqexplain.out files), turning on (or off) monitoring functionality, and sending an alert, in short, whatever you need to do. Unlike other UDRs, you cannot call another function, so all required functionality must be completely contained in the open and close function.

The final security enhancement that is discussed here is the ability to use filters in backup and restore operations. Typically, data sent to backups is executed by the *informix* user ID, which permits full access to all data, including automatic decryption as part of the backup operation. This is not acceptable in

environments where all data server data, regardless of its location, must be protected.

IDS 11 provides the ability to use a filter in-line with the backup or restore operation. This filter can re-encrypt the data before it leaves the instance for the storage medium or perform other operations, such as compression, if the hardware device does not support that functionality.

Administration

It is extremely hard to improve on the already low cost of ownership and ease of administration of IDS, but there are several significant enhancements in this area. The first enhancement is the introduction of a new administrative API that allows almost 80 instance operations to be executed through an SQL statement rather than one of the IDS command-line utilities. Operations executed through this interface are stored in a new system table for historic purposes.

Another enhancement is actually a combination of several other changes that affect how the instance manages operations. The combination of all of them enables an administrator to set a *recovery time objective*, or the amount of time that the instance takes to restart and come fully online after shutting down either normally or abnormally. In order to meet this objective, the instance now manages when checkpoints and buffer flushes occur, how least recently used (LRU) queues are managed, as well as the VPs associated with logical and physical log writes and other operations.

With IDS 10, DBAs were able to create or drop indexes without interrupting user operations on the tables involved. Unfortunately, the indexes were not of use to the SQL optimizer until statistical information was generated on the indexed columns. This additional statistics gathering process required manual execution and sometimes interfered with operations. With IDS 11, when an index is created, metadata from the sort portion of the index build is provided to the SQL optimizer, enabling it to accurately use the index in building access plans as soon as the index build completes. In addition, statistical information is gathered on explicitly generated temporary tables and the indexes created on them.

The last major enhancement that we discuss here is a new graphical DBA tool called *OpenAdmin Tool for IDS*. Designed to help DBAs answer the most commonly asked questions, it is written in Hypertext Preprocessor (PHP) and works in conjunction with an Apache (or similar) Web server to manage all instances in your environment without having to load anything on the target servers. The utility contains tasks and sensors to gather information and execute operations. The type of the information that can be gathered includes historical performance data that you can use to create histograms to monitor and improve instance operations.

Application development

There are a large number of enhancements targeted to application developers, far too many to list in this chapter. Of them, perhaps the most notable are the ability to publish well-formed XML as the result of an SQL operation, concurrent optimization, derived tables, and the ability to define multiple triggers for the same triggering event.

There are two approaches to XML and its use: document-centric as implemented in DB2® 9 with the *pureXML*™ technology and data-centric being implemented in IDS. In IDS 11, the first stage of XML functionality is included, which is the ability to publish the results of SQL operations in well-formed XML out of the data server. This release also supports the ability to parse XML data, serialize it, or even operate on it to determine whether specific components exist. Using the included Extensible Stylesheet Transformation (XSLT) functions, you can transform XML documents into other formats if necessary.

Until now, the number and types of triggers that you can create on a table have had limitations. Although you can create multiple SELECT and UPDATE triggers (provided the UPDATE triggers referenced different columns), you can have only one INSERT and DELETE trigger. In IDS 11, you now have the ability to define multiple triggers for all triggering actions, including INSTEAD OF triggers on views. When multiple triggers are defined for the same triggering event, all BEFORE actions of the appropriate triggers are executed first, followed by FOR EACH ROW actions. Last, the AFTER actions are executed. This process ensures there are no out-of-sequence execution issues if you need to drop, modify, and re-create one or more triggers.

With concurrent optimization, the behavior of the SQL operations in COMMITTED READ or DIRTY READ isolation levels changes to prevent deadlocks from occurring. Previously, if one operation was modifying rows in a table, other operations were either blocked until the transaction committed or rolled back or received non-committed values that can change based on the success or failure of the transaction. With the new LAST COMMITTED option, the previously committed value of the data is returned to the requesting operations.

Finally, IDS now supports the ANSI standard syntax to use the results of an encapsulated SELECT statement in the FROM clause, as depicted in Example 1-1 on page 35.

Example 1-1 IDS support for encapsulated SELECT statements

```
select sum(col_1) as sum_col1, col2
   from
      (select col_a, col_b from my_tab)
      as virt_tab(virt_col1, virt_col2)
   group by virt_col2;
```

With this functionality, your application developers might no longer need to create and populate temporary tables. The encapsulated SELECT statement can execute almost everything that a normal SELECT statement can, including aggregation, sorting, grouping, and so on. And, you can create simple or more complex JOIN conditions as part of the encapsulated select, including UNION and OUTER.

2

Fast implementation

In this chapter, we discuss topics related to the installation of the IBM Informix Dynamic Server (IDS) software. This is not an exhaustive discussion, which is beyond the scope of this book. It includes only those things that are new in IDS 11 and critical to getting your copy of IDS up and running.

We briefly discuss upgrades and migration, because many of the installations are for existing IDS implementations or new installations that are used to replace other existing DBMS products. We discuss upgrades and migrations only briefly, because they are not the primary topics of this chapter.

The primary topics in this chapter are:

- The ability to choose what software products or components to install
- Using the bundle and product installers
- Using the installer to add or remove products or components

Note: Most of the figures in this chapter are only portions of the full window in order to make the areas of the figures that we describe more legible. Therefore, they do not depict exactly how the windows appear in an actual implementation.

2.1 How to get to IDS 11

IDS 11 is the path forward for all previous versions of the Informix Dynamic Server. Many installations of IDS 11 will be upgrades to earlier versions of IDS, other versions of Informix database servers, or even other database servers. 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 upgrade occurs in place with the old instance shut down, the new binaries loaded, and the instance restarted. The upgrade is complete. The unspoken assumption is that installation and testing occurred first on a test or development server.

Important: That assumption is extremely important. Never install any software upgrade into production without first testing the process and application thoroughly.

Depending on the current version of the database server, moving to IDS 11 can occur through an in-place upgrade. Other versions or even other Informix servers require an interim step. The paths to IDS 11 are:

- IDS 7.31, 9.21, 9.30, 9.40, or 10.00: It is a direct upgrade. You can install IDS 11, then shut down the instance, reset the environment to point to the IDS 11 software, and restart the instance. IDS 11 recognizes the older data and adjusts things as necessary.
- IDS 7.30 or 7.24 without Enterprise Replication: You must move to either IDS 9.40 or 10.00 before moving to IDS 11. Between these earlier versions and more current versions, a large number of structural changes occurred, not all of which are visible to an administrator. Make these changes to instance and database structures prior to moving to IDS 11.
- IDS 7.24 with replication: You must move to IDS 7.31 first.
- IDS 9.14 or 9.20: You must move to IDS 9.21 or 9.30 before moving to IDS 11.
- XPS 8.*x* except 8.21U: You must move to IDS 7.24 before moving to IDS 11.
- XPS 8.21U: You must move to IDS 7.24 without replication before moving to IDS 11.
- OnLine 5.*x*: You must move to IDS 7.31 first.

It *is* possible to upgrade from OnLine v. 5.1*x* by performing an incremental upgrade to IDS v.7.31 first. In following this path, the OnLine administrator needs to get as much education as possible on IDS prior to going through the upgrade process. As described in Chapter 1, "IBM Informix Dynamic Server 11 essentials" on page 1, the IDS architecture is radically different and, as a result, so are the administration, performance tuning, and the day-to-day maintenance activities of the environment. IDS requires new skills and insight.

Explicit directions for executing an in-place upgrade are contained in the *IBM Informix Migration Guide,* G251-2293, for each version of IDS. One important but undocumented step involves the installation of the new binaries. You must install the new binaries in a different directory than the current $INFORMIXDIR (the directory where the software is installed). Simply change $INFORMIXDIR and $PATH to reflect the new location, so that the correct binaries are used when restarting the instance. This facilitates easier reversion if necessary. We summarize the general process of an in-place upgrade 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 must 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 need to 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 must be multiple full instance backups, as well as other data exports if time and space permit. These backups must be verified with the *archecker* utility before the upgrade begins. System backups and copies of configuration files must 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 the largest user load that you can create.
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 about 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 the instance to quiescent mode, and take the appropriate actions if using replication. *Create one or more level 0 backups*. Verify them with the *archecker* utility.
Install and configure.	Install the new binaries preferably in another directory than the current $INFORMIXDIR. Copy or 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, *do not attempt to connect to the instance until the log indicates the databases have been rebuilt; otherwise, the instance will become corrupted.*
Perform post-migration activities.	Drop the update statistics, create a new level 0 backup, verify data integrity with the *oncheck* utility, re-enable or restart replication, and restore user access.

If problems arise after the upgrade, there are at least two options to revert back to the earlier version. The options are using the IDS bundled reversion utility or restoring the level 0 backup created at the beginning of the upgrade process and resetting the environment parameters to point to the original $INFORMIXDIR and related structures. Both options work, although there are conditions that preclude using the reversion utility. These conditions include 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 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. The process is to 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 message log (declared in the MSGPATH parameter in the configuration file) for all reversion messages, including those about the "sys" databases. After the process completes, verify the data integrity with the *oncheck* utility, create a level 0 backup, and 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 can originate from an Informix Standard Engine (SE) instance or another server from the IBM, or a competitor's portfolio of products. The migration process is not included in this chapter. Compared to an upgrade or initial installation, migration is much more labor-intensive and requires more careful preparation and planning, as well as more time to execute.

You can find all the details about how to migrate to IDS 11 in the *IBM Informix Migration Guide,* G251-2293. And, you can download that manual from the IBM Web site.

IBM has a tool to assist and even perform part of the work to convert database structures and to move data from a source server to IDS. The IBM Migration Toolkit (MTK) is a free downloadable utility with a Windows and UNIX or Linux port. At the time of the writing of this book, the current Web site to access the MTK is:

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

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

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

2.2 The installation process

Now we return to the main topic of this chapter, installing IDS 11. With IDS 11, IBM has introduced new packaging and processes for installing several or all of the products or parts of products, and we discuss package and process.

2.2.1 Bundles, products, and components

IDS ships in a package that includes the DBMS, the Client Software Development Kit (CSDK), Informix Connect (I-Connect), and the JDBC™ Driver. This entire collection is called the IDS *Bundle*.

The pieces that comprise the bundle are called *products*. The Dynamic Server, the CSDK, and I-Connect are the products in the IDS bundle.

The pieces or features of each product are called *components*. The IDS product has a number of components that might be included in or excluded from the installation. You can, for example, include or exclude the OnBar component for

backup and restore. Similarly, you might exclude the locales that you do not expect to use. For more information, see 2.2.5, “Choosing components” on page 44, which lists the components of the IDS product.

Similarly, the CSDK has ESQL, the ODBC Driver, and other components. You can choose which components to install or not install.

On a UNIX or Linux system, there is a master list in *$INFORMIXDIR/etc/IIFfiles* of the contents of each component of IDS. The file describes the location, permissions, and component of each file in the IDS product. This file contains the master list of all of the files, where they reside if they are installed, and what the owner, group, and permissions must be if they are installed. No equivalent file exists on Windows systems.

2.2.2 Manifest files and reinstallation

The installer for IDS 11 is able to both install and reinstall either the bundle, individual products, or individual components of products. This installation keeps a record of what is currently installed. The record is in the *manifest file*. The file is *$INFORMIXDIR/etc/manifest.inf*. This is a simple text file that you can view with any viewer or editor.

Important: Do not modify the manifest file. Modifying the manifest file makes any future changes very difficult.

Just execute the installer again to change the components that have been installed (to add more locales, for example). As we will show you later, you can change the list of installed components. The installer then adds or removes the correct files so that the new manifest matches what was chosen.

This means that you only install the programs that you use, which minimizes the disk space needed and prevents users from accessing the wrong programs.

2.2.3 Installation modes

There are three modes for installing the IDS bundle:

- Console
- Graphical
- Silent

The default method on UNIX or Linux is a text-based set of questions and answers suitable for use on a system console. This is called *console mode*. A console mode installation has all the same options and functions as the graphical

installer, but it is more cumbersome to use, because more keystrokes are required. This method works on any UNIX or Linux system regardless of the windowing or graphics support that is used.

The graphical installer is the easiest, most intuitive method to install IDS. It is a Java program that presents a series of windows in which choose what you want to do. The graphical installer can record what you chose so that you can use those choices later in silent installations. This is the default for Windows systems.

The Java Virtual Machine (JVM™) is included in the IDS bundle, so you do not need a specific version of Java to use the graphical installer.

Last, you can choose a silent installation. This technique requires a file that declares what you want. The default files are *bundle.ini* for the IDS bundle and *server.ini* for the product. A silent installation is invoked from the command prompt and there is no further interaction to complete the installation.

Each of these installation methods is discussed in the subsequent sections of this chapter.

2.2.4 Preliminary installation steps

You must sign on as user root in Linux or UNIX or as a user with administrative privileges on Windows to install IDS, the CSDK, or Informix-Connect (I-Connect). If you do not have these privileges, you must get help with the installation, because you cannot perform any of these installations as an unprivileged ordinary user.

The first step in any installation is to uncompress the delivery media, if that is required. This is usually required if you downloaded the bundle or product, because the download is compressed to make it smaller. If you are installing from a DVD or CD, you probably do not need to uncompress anything.

Whether you use a DVD or uncompress a downloaded file, the result is a directory with the software ready to install. In later sections, we refer to this as the *source* directory.

After that, you decide where to place the software on your system. You must assign the directory where the software will be located as the value of an environment variable called *INFORMIXDIR*. That variable (*$INFORMIXDIR*) is used from now on to designate the directory where the software was or is to be installed.

2.2.5 Choosing components

The next step of the installation is to decide which components of IDS to include and which components to exclude. You can change the selections if necessary, but it is best if you have already decided which components you want before you begin.

You need to consider a few major choices, which include:

- Will ontape or OnBar be used for backups? If you will use ontape, then choose to exclude OnBar from the installation. Ontape is always installed, even if it is never used.
- Is bulk unloading or loading of data planned, and if so, which utility will you use for that purpose? If you do not plan to use any of the utilities, exclude the whole set (onload, onunload, dbload, dbunload, and the High-Performance Loader). If you only use one of the utilities, you can exclude the other utilities.
- Which languages are used? Include only those languages that you want and exclude the other languages.

The full set of IDS components and their subcomponents includes:

- The base DBMS
- Server extensions:
  - J/Foundation (for executing Java routines)
  - Datablade modules (the built-in modules)
  - Conversion and reversion utilities
- Backup and restore:
  - Archecker
  - OnBar
- Informix Storage Manager (ISM)
- The interface for the Tivoli Storage Manager (TSM)
- Demo
- Data loading utilities:
  - ON-Load
  - DB-Load
  - HPL
- Enterprise Replication
- Administrative utilities:
  - Performance monitoring
  - Monitoring
  - Auditing
  - Database Import and Export Utility
- Global Language Support (GLS):
  - Western Europe
  - Eastern Europe

- Chinese
- Korean
- Japanese
- Other

For an installation, you can select or exclude each component or subcomponent separately.

2.2.6 Installing with the graphical installer for Windows

Invoke the installer by executing the **setup.exe** file in the source directory. Following a splash window, the next window that you see has a frame in the middle of it, as depicted in Figure 2-1 on page 46.

This window is where you choose the products that you want to install. Notice that the list of products is different from those products that are available on Linux or UNIX. The Datablade Developers Kit, the BladeManager, and the Cluster Setup Utility are all programs that run only on Windows systems.

For this example, we have chosen not to install the Cluster Setup Utility, the Datablade Developers Kit, or the BladeManager. Instead, we focus on the DBMS and the software to connect to it (CSDK or I-Connect). I-Connect is usually not required if you install the CSDK, because the CSDK includes all the libraries and functions of I-Connect. We have chosen to install only the CSDK.

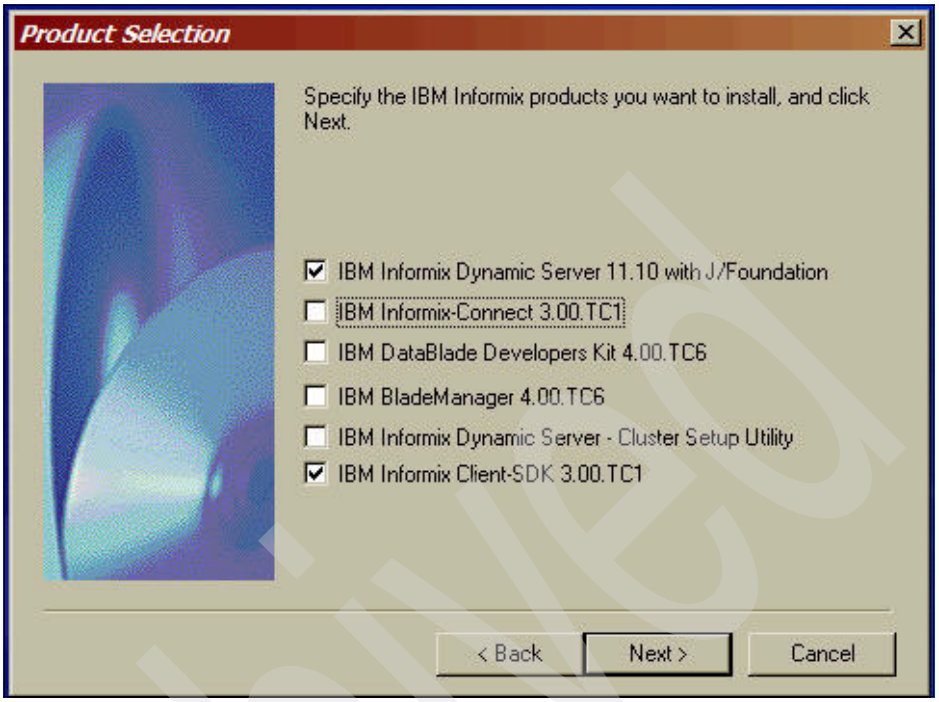


Figure 2-1 Product selection for Windows

When you have chosen the products that you want to install, select **Next**. After a splash window, a welcome window, and you accept the contractual terms, the next window gives you the choice between a *typical* installation or a *custom* installation.

A *typical* installation offers very few choices about what to do. You cannot choose the components to install, and the instance is initialized automatically. Use this choice to install all the software and initialize the instance. Note that the existing databases are destroyed in the process.

A *custom* installation allows many more choices about what to do. You can choose the components that you want to include or exclude. And, you can choose whether to initialize the instance. You also can make other choices about role separation, the instance name and number, and the port number for communications. Use the custom installation to control any of those options.

For this example, we chose a custom installation.

After the selections are made and you click **Next**, a window similar to Figure 2-2 on page 47 appears that shows the default location to install the software. You have the option to change to a different location.

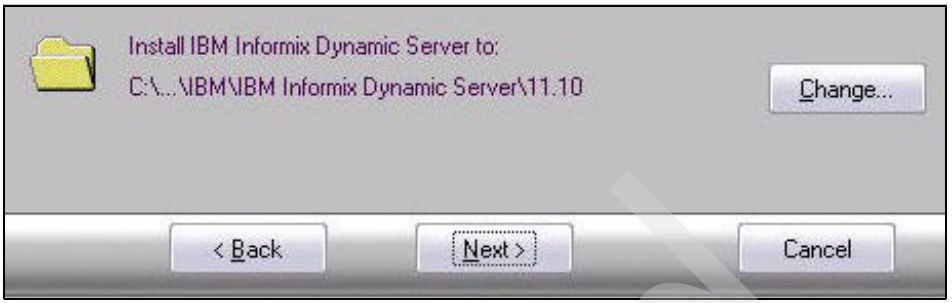


Figure 2-2 Windows installation path

When you select **Next**, you see a window that contains the information shown in Figure 2-3 on page 48.

This is the new part of the custom installation process and is not part of the typical installation process. Here, the installer asks you which features or components of the IDS product to include or exclude. Installing only the base server and a single set of locales reduces the required disk space to approximately 106 MB. Installing everything requires approximately 210 MB of disk space.

Choose the components that you need based on the intended use of the product. For example, if you do not plan to use any User Defined Routines written in Java, you do not need the J/Foundation component. Also, refer to 2.2.5, “Choosing components” on page 44.

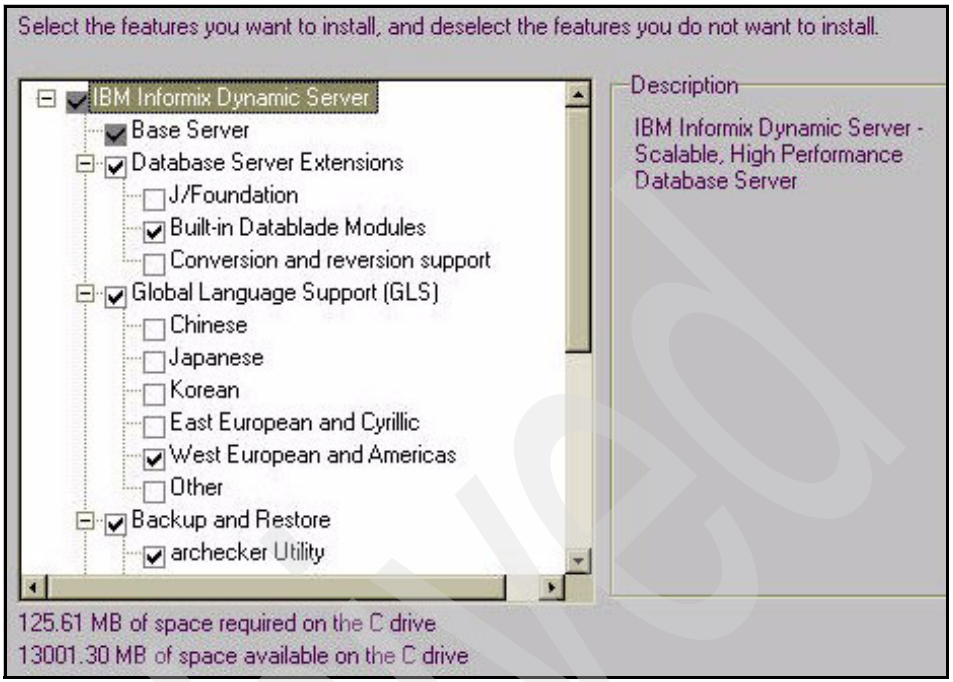


Figure 2-3 Component selection

After you select the components to install, a window containing information similar to Figure 2-4 on page 49 appears. Here, you can select the instance name, number, and also the port for network communications.

Note that the default port number has changed to 9088 from 1526 in order to avoid conflicts with other products that typically use port 1526.

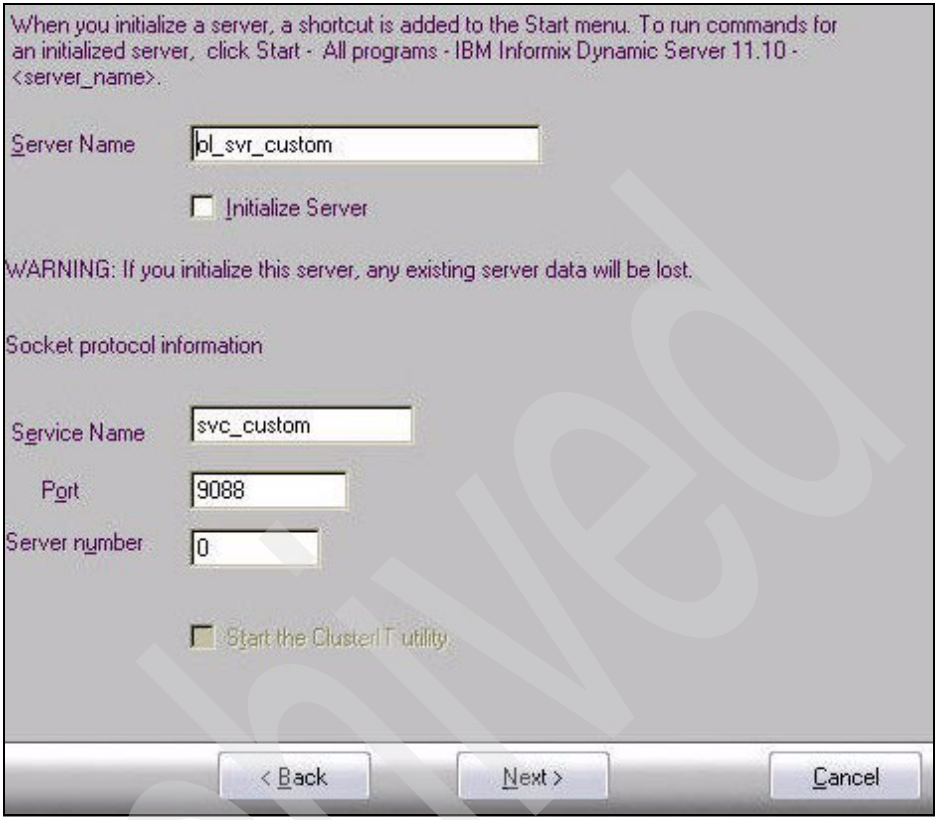


Figure 2-4 Instance name, number, and port

Select **Next** to see a window similar to Figure 2-5 on page 50. This window summarizes the choices and gives you the opportunity to go back to change any of those choices before installing the software. Note that the list of specific components that you selected are *not* listed on this window.

When you select **Next** from this window, then the actual software installation begins.

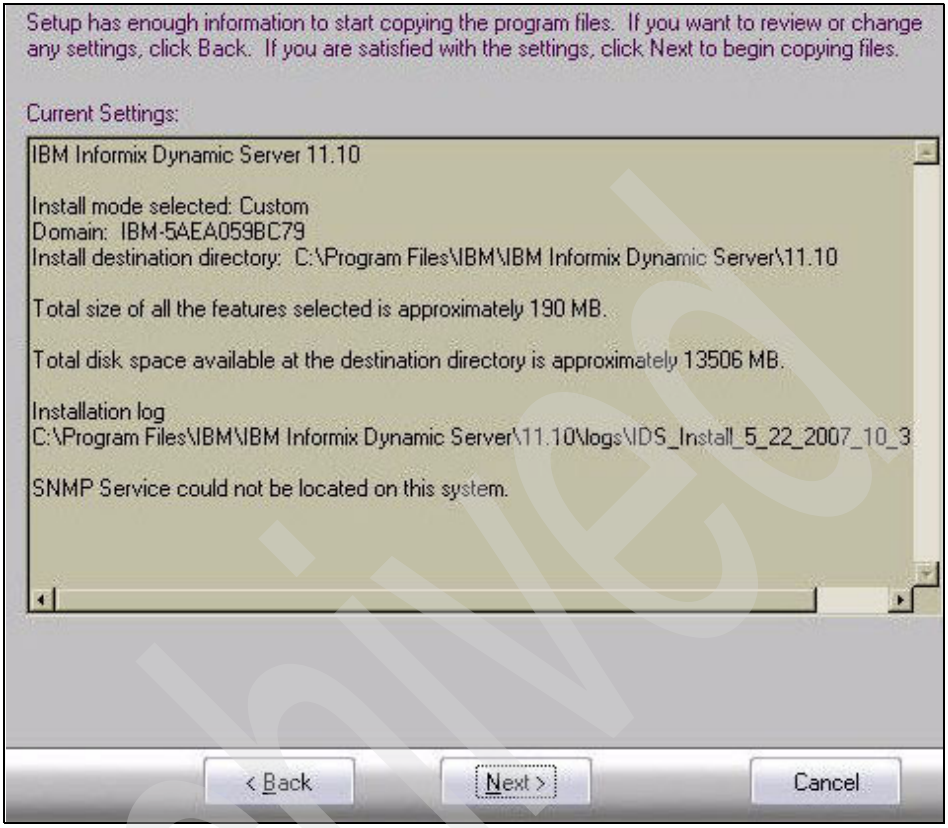


Figure 2-5 Ready to install

2.2.7 Installing with the graphical installer on UNIX or Linux

In this section, we discuss installing the IDS bundle with the graphical installer on Linux and UNIX. You invoke the graphical installer by using the **`ids_install`** command found in the source directory. The command is:

```
ids_install -gui
```

The **`-gui`** switch is required, because the default is console mode. You must specifically ask for the graphical installer.

The windows are the same as described in 2.2.6, “Installing with the graphical installer for Windows” on page 45.

A few things to note about the installation process:

- The default directory is now */opt/IBM/informix*. This is a change from the previous default value of */usr/informix* in order to conform to the IBM standard for installing software products.
- If you set *$INFORMIXDIR* before starting the installer, the installer uses that setting as the default. That is one way to change the default.
- The installation process can create the `stores` demonstration database. This is a new feature in the installation. Previous installation processes did not automatically create the demonstration database. If you choose to create the demonstration database, additional windows solicit the path and size for the dbspace to use for the database.
- In addition, if the demonstration database is created, the IDS instance is also initialized. If you are upgrading or want to start the instance later, do not choose the demonstration database. You can create the demonstration database later by using the **dbaccessdemo** or **dbaccessdemo_ud** utilities.
- When the installation process is complete, there is a final confirmation of what was done and what to do next. The next steps are discussed in 2.4, “First steps after installation” on page 64.

2.2.8 Installing in console mode on Linux or UNIX

To install IDS in console mode is nearly the same process as installing with the graphical installer. Example 2-1 shows the command and first few steps. All the same functions and choices are available in console mode as are available with the graphical installer.

Notice a number of differences from the graphical installation:

- You might have a default action if you just press the Enter key. Read the directions carefully, because the actions and the steps differ.
- Installation is much more laborious. For example, to change from a typical to a custom installation for IDS, you select the product, declare that you want to change the mode, and then select *custom* as the mode. All of this is done with numbers rather than words.
- You can still move forward and backward as you can between the windows of the graphical installation, but again, it is a step-by-step process.

Example 2-1 Console mode installation

```
[root@server_1 expand]# ids_install

          Initializing InstallShield Wizard........
          Launching InstallShield Wizard........
```

```
   1. Release Notes
   2. Installation Guide
   3. Launch Information Center
   4. Begin Installation

Please select one of these options [4]

Beginning installation...

Press 1 for Next, 3 to Cancel or 4 to Redisplay [1]

Welcome to the InstallShield Wizard for IBM Informix Dynamic Server Bundle

The InstallShield Wizard will install IBM Informix Dynamic Server Bundle on
your computer.
To continue, choose Next.

Press 1 for Next, 2 for Previous, 3 to Cancel or 4 to Redisplay [1]

Software Licensing Agreement
Press Enter to display the license agreement on your screen. Please
read the agreement carefully before installing the Program. After
 reading the agreement, you will be given the opportunity to accept it
 or decline it. If you choose to decline the agreement, installation
 will not be completed and you will not be able to use the Program.

1
     International License Agreement for Early Release of
     Programs

Part 1 - General Terms

THIS INTERNATIONAL LICENSE AGREEMENT FOR EARLY RELEASE OF
      PROGRAMS ("AGREEMENT") IS A LEGAL AGREEMENT BETWEEN YOU AND
      IBM. BY DOWNLOADING, INSTALLING, COPYING, ACCESSING, OR
      USING THE PROGRAM YOU AGREE TO THE TERMS OF THIS AGREEMENT.
      IF YOU ARE ACCEPTING THESE TERMS ON BEHALF OF ANOTHER
      PERSON OR A COMPANY OR OTHER LEGAL ENTITY, YOU REPRESENT
      AND WARRANT THAT YOU HAVE FULL AUTHORITY TO BIND THAT
      PERSON, COMPANY, OR LEGAL ENTITY TO THESE TERMS.

Press Enter to continue viewing the license agreement, or, Enter 1 to
accept the agreement, 2 to decline it or 99 to go back to the previous
 screen.

1

Press 1 for Next, 3 to Cancel or 4 to Redisplay [1]
```

```
IBM Informix Dynamic Server Bundle Install Location

Please specify a directory or press Enter to accept the default directory.

Directory Name: [/opt/IBM/informix/11]

Press 1 for Next, 3 to Cancel or 4 to Redisplay [1] 1

Searching for products available for install: this may take a few minutes.

Select the products you would like to install:

   To select/deselect a product or to change its setup type, type its number:

   Product                                          Setup Type
   ------------------------------------------------ -----------------
    1. [ ] IBM Informix IConnect
    2. [x] IBM Informix Client-SDK                   Typical
    3. [x] IBM Informix Dynamic Server               Typical
    4. [x] IBM Informix JDBC Driver

   Other options:

    0. Continue installing

   Enter command [0] 3

    1. Deselect 'IBM Informix Dynamic Server'
    2. Change 'IBM Informix Dynamic Server' setup type

   Enter command [1] 2

   Select a setup type for 'IBM Informix Dynamic Server':

    1. Typical
    2. Custom

   Select a setup type [1] 2

Select the products you would like to install:

   To select/deselect a product or to change its setup type, type its number:

   Product                                        Setup Type
   ---------------------------------------------- ----------------------
    1. [ ] IBM Informix IConnect
    2. [x] IBM Informix Client-SDK                 Typical
    3. [x] IBM Informix Dynamic Server             Custom
```

```
 4. [x] IBM Informix JDBC Driver

Other options:

 0. Continue installing

Enter command [0]
```

2.2.9 Silent installation

Silent installation is a different process than those we have discussed so far. This installation is driven by a file that declares the desired actions. All of the same choices are available as in the graphical or console installations, but they are declared differently.

On Linux or UNIX, the source directory includes a file named *bundle.ini*. That file can be used as a template for a customized silent installation. There is no bundle.ini file on Windows.

A file named server.ini is in the SERVER subdirectory on Linux or UNIX. The server.ini file is in the IIF subdirectory on Windows.

The windows server.ini file is quite cryptic. Do not try to copy that file and edit it to create a new response file without extremely careful testing. A better method is to redo the graphical installation to create a new response file, changing options as desired in the new installation. In this case, be sure to uninstall everything before starting the new installation.

Important: Do not modify either bundle.ini or server.ini. Copy those files to other files and modify the copies.

On Linux and UNIX systems, both these files are structured the same way. Part of the Linux bundle.ini file is shown in Example 2-2 on page 55. The Windows server.ini file
is shown in Example 2-3 on page 56.

In both files, each choice in the console or graphical installation is presented. Certain items allow a data value. The installLocation item is an example. The components are included by setting their active attribute to true, and components are excluded by setting their active attribute to false.

Example 2-2 The bundle.ini file

```
############################################################################
# InstallShield Options File Template
#
# Wizard name: Install
# Wizard source: suite.jar
# Created on: Mon Aug 09 15:59:20 CDT 2006
# Created by: InstallShield Options File Generator
#
# This file contains values that were specified during a recent execution of
# Install. It can be used to configure Install with the options specified below
# when the wizard is run with the "-options" command line option. Read each
# setting's documentation for information on how to change its value.
#
# A common use of an options file is to run the wizard in silent mode. This
lets
# the options file author specify wizard settings without having to run the
# wizard in graphical or console mode. To use this options file for silent mode
# execution, use the following command line arguments when running the wizard:
#
#    -options "bundle.ini" -silent
#
############################################################################

############################################################################
# IBM Informix Dynamic Server Bundle Install Location
#
# The install location of the product. Specify a valid directory into which the
# product should be installed. If the directory contains spaces, enclose it in
# double-quotes. For example, to install the product to /usr/informix use
#
#    -P installLocation="/usr/informix"
#

-P installLocation="/usr/informix"

############################################################################
# IBM Informix Client-SDK
#
# The selection state of product "IBM Informix Client-SDK". Legal values are:
#
#    true  - Indicates that the product is selected for installation
#    false - Indicates that the product is not selected for installation
#
# For example, to select "IBM Informix Client-SDK" for installation, use
#
#    -P csdk.active=true
#
```

```
# Also, only either IConnect OR the full Client-SDK can be installed at once.
# They cannot both be installed into the same location
#

-P csdk.active=false

###############################################################################
#
# Setup Type
#
# The setup type to be used when installing the product. Legal values are:
#
#    typical - Typical: The program will be installed with the suggested
#              configuration. Recommended for most users.
#    custom  - Custom: The program will be installed with the features you
#              choose. Recommended for advanced users.
#
# For example, to specify that the "Typical" setup type is selected, use
#
#    -W setupTypes.selectedSetupTypeId=typical
#
# You may also set the setup type to nothing by using
#
#    -W setupTypes.selectedSetypTypeId=
#
# This clears the current setup type and prevents any changes to the set of
# selected features. Use this option whenever you set feature active states in
# this options file. If you do not clear the selected setup type, the setup
type# panel will override any changes you make to feature active states using
this
# file.

-SW CSDK/UNIX/csdk.jar setupTypes.selectedSetupTypeId=typical
```

The Windows server.ini file is shown in Example 2-3.

Example 2-3 The Windows server.ini file

```
[InstallShield Silent]
Version=v7.00
File=Response File
[File Transfer]
OverwrittenReadOnly=NoToAll
[{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-DlgOrder]
Dlg0={0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-SdWelcome-0
Count=10
Dlg1={0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-Software License Agreement-0
Dlg2={0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-SetupType2-0
Dlg3={0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-SdAskDestPath2-0
```

```
Dlg4={0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-SdComponentTree-0
Dlg5={0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-Informix user setup-0
Dlg6={0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-Custom server configuration setup-0
Dlg7={0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-Server DBSpace SBSpace Setup-0
Dlg8={0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-SdStartCopy-0
Dlg9={0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-SdFinish-0
[{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-SdWelcome-0]
Result=1
[{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-Software License Agreement-0]
Accept License Agreement=1
Result=1
[{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-SetupType2-0]
Result=303
[{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-SdAskDestPath2-0]
szDir=C:\Program Files\IBM\IBM Informix Dynamic Server\11.10
Result=1
[{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-SdComponentTree-0]
szDir=C:\Program Files\IBM\IBM Informix Dynamic Server\11.10
IBM Informix Dynamic Server\Engine-type=string
IBM Informix Dynamic Server\Engine-count=3
IBM Informix Dynamic Server\Engine-0=IBM Informix Dynamic
Server\Engine\Krakatoa
IBM Informix Dynamic Server\Engine-1=IBM Informix Dynamic
Server\Engine\DataBlades
IBM Informix Dynamic Server\Engine-2=IBM Informix Dynamic
Server\Engine\Conversion Reversion
IBM Informix Dynamic Server\GLS-type=string
IBM Informix Dynamic Server\GLS-count=6
IBM Informix Dynamic Server\GLS-0=IBM Informix Dynamic Server\GLS\Chinese
IBM Informix Dynamic Server\GLS-1=IBM Informix Dynamic Server\GLS\Japanese
IBM Informix Dynamic Server\GLS-2=IBM Informix Dynamic Server\GLS\Korean
IBM Informix Dynamic Server\GLS-3=IBM Informix Dynamic Server\GLS\EastEurope
IBM Informix Dynamic Server\GLS-4=IBM Informix Dynamic Server\GLS\WestEurope
IBM Informix Dynamic Server\GLS-5=IBM Informix Dynamic Server\GLS\Other
IBM Informix Dynamic Server\Backup Restore-type=string
IBM Informix Dynamic Server\Backup Restore-count=4
IBM Informix Dynamic Server\Backup Restore-0=IBM Informix Dynamic Server\Backup
Restore\Archecker Utility
IBM Informix Dynamic Server\Backup Restore-1=IBM Informix Dynamic Server\Backup
Restore\OnBar Utility
IBM Informix Dynamic Server\Backup Restore-2=IBM Informix Dynamic Server\Backup
Restore\OnBar with ISM support
IBM Informix Dynamic Server\Backup Restore-3=IBM Informix Dynamic Server\Backup
Restore\OnBar with TSM support
IBM Informix Dynamic Server\Data Load-type=string
IBM Informix Dynamic Server\Data Load-count=3
IBM Informix Dynamic Server\Data Load-0=IBM Informix Dynamic Server\Data
Load\OnLoad Utility
```

```
IBM Informix Dynamic Server\Data Load-1=IBM Informix Dynamic Server\Data
Load\DBLoad Utility
IBM Informix Dynamic Server\Data Load-2=IBM Informix Dynamic Server\Data
Load\High Performance Loader
IBM Informix Dynamic Server\Utilities-type=string
IBM Informix Dynamic Server\Utilities-count=3
IBM Informix Dynamic Server\Utilities-0=IBM Informix Dynamic
Server\Utilities\Server
IBM Informix Dynamic Server\Utilities-1=IBM Informix Dynamic
Server\Utilities\Audit
IBM Informix Dynamic Server\Utilities-2=IBM Informix Dynamic
Server\Utilities\DBA Tools
IBM Informix Dynamic Server-type=string
IBM Informix Dynamic Server-count=8
IBM Informix Dynamic Server-0=IBM Informix Dynamic Server\Core
IBM Informix Dynamic Server-1=IBM Informix Dynamic Server\Engine
IBM Informix Dynamic Server-2=IBM Informix Dynamic Server\GLS
IBM Informix Dynamic Server-3=IBM Informix Dynamic Server\Backup Restore
IBM Informix Dynamic Server-4=IBM Informix Dynamic Server\Demo
IBM Informix Dynamic Server-5=IBM Informix Dynamic Server\Data Load
IBM Informix Dynamic Server-6=IBM Informix Dynamic Server\Enterprise
Replication
IBM Informix Dynamic Server-7=IBM Informix Dynamic Server\Utilities
HIBM Informix Dynamic Server-type=string
HIBM Informix Dynamic Server-count=1
HIBM Informix Dynamic Server-0=HIBM Informix Dynamic Server\HCore
Component-type=string
Component-count=2
Component-0=IBM Informix Dynamic Server
Component-1=HIBM Informix Dynamic Server
Result=1
[{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-Informix user setup-0]
User=<Machine>\informix
Install in Domain=0
Password=<Informix Password>
Confirm Password=<Confirm Password>
Enable Role Separation=0
Result=1
[{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-Role separation user setup-0]
DBSA Group Account=Informix-Admin
DBSSO Group Account=ix_dbsso
DBSSO User Account=DBSSO
DBSSO Password=<DBSSO Password>
DBSSO Confirm Password=<Confirm DBSSO Password>
AAO Group Account=ix_aao
AAO User Account=AAO
AAO Password=<AAO Password>
AAO Confirm Password=<Confirm AAO Password>
IDS Users Group Account=ix_users
```

```
Result=1
[{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-Custom server configuration setup-0]
Server Name=ol_svr_custom
Service Name=svc_custom
Port=9088
Server number=2
Initialize Server=1
Result=1
[{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-Server DBSpace SBSpace Setup-0]
DBSpace Name=ol_svr_custom
Primary Data Location=C:
Mirror Data Location (optional)=' '
Size (MB)=100
SBSpace Name=sbspace
SBSpace Primary Data Location=C:
SBSpae Mirror Data Location (optional)=' '
SBSpace Size (MB)=100
Page Size (in pages)=1
Result=1
[{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-SdStartCopy-0]
Result=1
[Application]
Name=IBM Informix Dynamic Server
Version=11.10
Company=IBM
Lang=0009
[{0F0B0E22-611B-48D7-A6D0-138DCBE1354C}-SdFinish-0]
Result=1
bOpt1=0
bOpt2=0
[{8BB748AE-8CEB-44F4-A1FF-BE74FF70ABBE}-DlgOrder]
Count=0
```

On any platform, a silent installation can be performed by using the default values in server.ini and bundle.ini. However, custom files can be used for silent installations. As we just mentioned, *do not modify the files in the distribution*, copy them to other files before changing anything. After you copy the files, you can use any editor to modify the copies.

On Windows, the command for a silent installation is different from the command on Linux or UNIX. For example, the silent installation corresponding to the one just given for Linux is:

```
setup.exe -s -f1 "c:\TEMP\responsefile.ini" -f2 "c:\TEMP\install.log"
```

Read the installation guide carefully to understand all of the syntax.

Alternatively, the graphical installers can create a customized silent installation file by recording the choices made during the graphical installation. You do this by a switch on the command to launch the graphical installation.

On Linux, for example, the command to capture responses from a graphical installation might be:

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

The *responsefile.ini* file is created and contains the specifications that match what you do during the graphical or console installation. Note that there is no other way to declare that you want to record. You must use this command line command to launch the graphical installer and instruct it to record the choices.

After creating the response file, use the file in the command to perform the silent installation. If you used the previous command to capture your choices, the silent installation command is similar to:

```
ids_install -silent -options responsefile.ini -log /tmp/install.log
```

This command reads the *responsefile.ini* to determine what to do, and it records the results in */tmp/install.log*.

There is little to show about a silent installation, because it actually is silent. There is nothing to show the progress or that it has completed. The only way that you can discover that it has completed is that the command prompt returns. To learn what happened, you must examine the log file. Example 2-4 shows the results of the silent installation command.

Example 2-4 Log of a silent installation

```
$TMP not set, defaulting to /tmp
Log file created by ISMP installer @Tue May 22 06:06:05 CDT 2007
INSTALLER_PATH=/ifmx_data/expand/./ids_install
Verifying /usr/java/default/ using the JVM file
/tmp/istemp4889142060605/genericjvmfile
Verifying... /usr/java/default//bin/java
$FULLCURRJVM is "1.6.0"
$CURRJVMNOQ is 1.6.0
$CURRJVMNO_ is 1.6.0
$CURRJVMNODOT is 160
$CURRJVM is 160
Version of /usr/java/default//bin/java is 160
Comparing to required version of 142
Version "1.6.0" has been verified
Verification passed for /usr/java/default/ using the JVM file
/tmp/istemp4889142060605/genericjvmfile.
command line switch -javahome is specifed. Verifying the JVM with the JVM files
specifed with the launcher.
```

```
Verifying /usr/java/default/ using the JVM file
/tmp/istemp4889142060605/genericjvmfile
Verifying... /usr/java/default//bin/java
$FULLCURRJVM is "1.6.0"
$CURRJVMNOQ is 1.6.0
$CURRJVMNO_ is 1.6.0
$CURRJVMNODOT is 160
$CURRJVM is 160
Version of /usr/java/default//bin/java is 160
Comparing to required version of 142
Version "1.6.0" has been verified
Verification passed for /usr/java/default/ using the JVM file
/tmp/istemp4889142060605/genericjvmfile.
LAUNCH_COMMAND=/usr/java/default//bin/java  -cp
.::::::"suite.jar":"SERVER/IIF.jar":"SERVER/IDS2000.jar":"ICONNECT/UNIX/conn.jar
":"
SDK/UNIX/csdk.jar":"JDBC/setup.jar":"suite.jar":::::: -Dtemp.dir="/tmp"
-Dis.jvm.home="/usr/java/default/" -Dis.jvm.temp="0" -Dis.media.home=""
-Dis.launcher.file="/ifmx_data/expand/./ids_install"
-Dis.jvm.file="/tmp/istemp4889142060605/genericjvmfile" -Dis.exter
al.home="/ifmx_data/expand/."   run -G overwriteJVM=yes  -silent -options
example.ini 1>>/tmp/install.log 2>>/tmp/istemp48891420606
5/APP_STDERR
```

2.2.10 Using the product installation scripts

Rather than start with the bundle installer (**`ids_install`** on Linux or UNIX, or **`setp.exe`** on Windows), you can install each product separately by using their individual installation scripts. For example, to install IDS, use the *installserver* script in the SERVER subdirectory of the source directory. All the same options are available, but a separate command is required for each product. The bundle installer is usually an easier starting point.

2.2.11 Using the JAR file on Linux or UNIX

Rather than using any of the previous options, you can directly execute the installation program JAR file. There are separate files for the base DBMS, the full IDS suite, the CSDK, and I-Connect. You must use each file separately if you choose this technique. For example, to install just the base DBMS, the command is either:

`java -cp IIF.jar run` for a console installation or

`java -cp IIF.jar run -swing` for the graphical installer

2.3 Adding or removing components

After installing IDS 11, you might need to alter the installation by adding or removing IDS components. For example, you might need to add locales for new users or choose to begin using OnBar rather than ontape for backups.

To add or remove components, use the same installer that you used for the previous installation. However, because the installer finds the product already installed, there are different choices. After launching the installer and choosing the product that you want to affect (on the second window), you see the choices shown in Figure 2-6.

You can remove or uninstall components or the entire product; you can reinstall damaged files; or you can change what is installed by adding or removing (or both) components.

For the example depicted in Figure 2-6, we chose to **Modify** the installation.

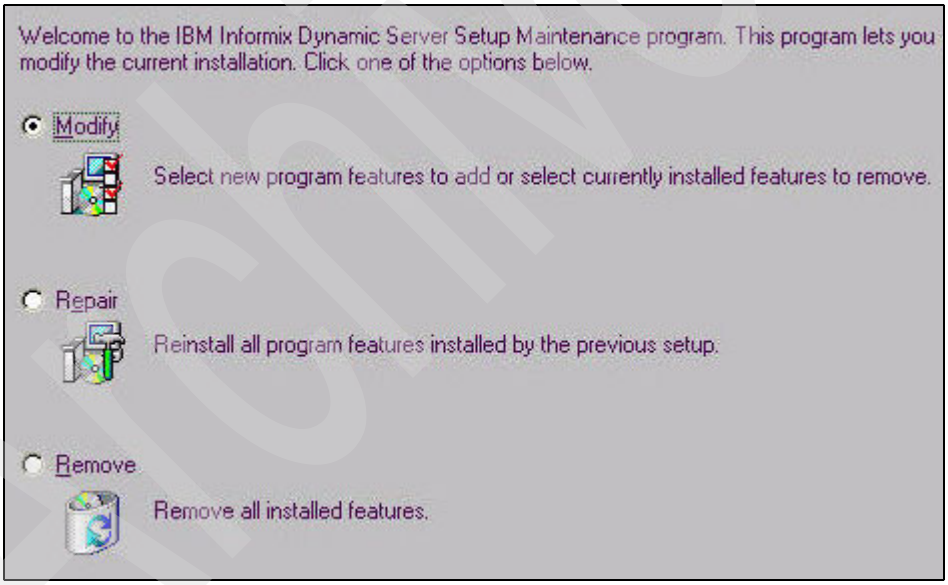


Figure 2-6 IDS setup maintenance program

When you click **Next**, the window shown in Figure 2-7 on page 63 appears. Compare this to Figure 2-3 on page 48. The same choices are available in both cases.

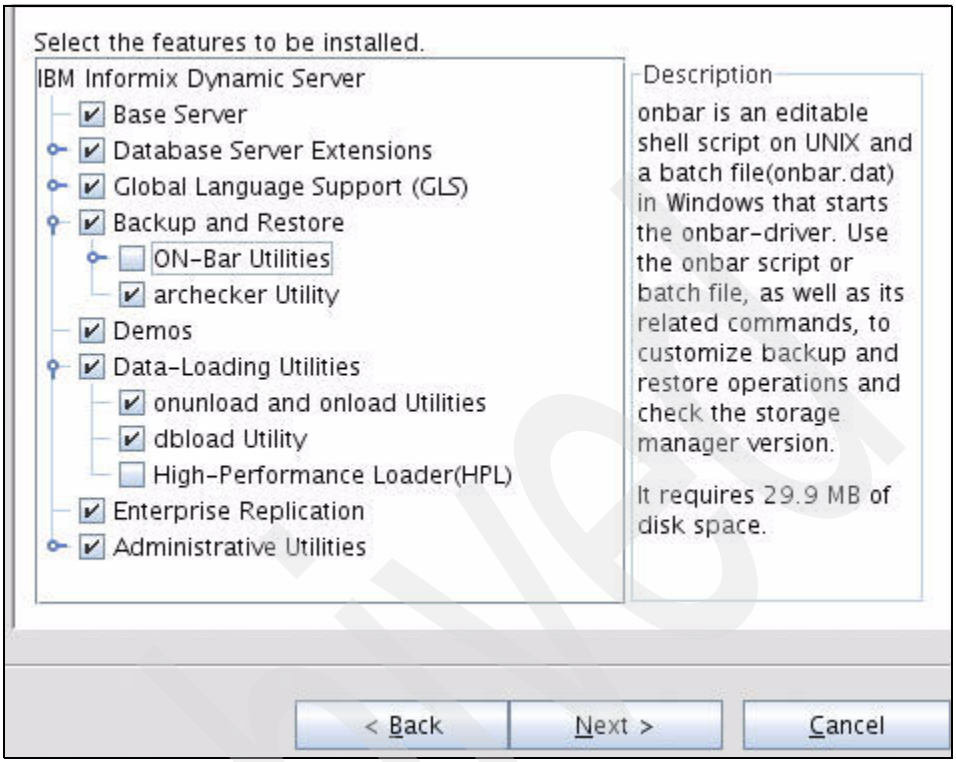


Figure 2-7 Component selection on reinstallation

You can change the selections by adding or removing them. For example, you can add the High-Performance Loader. When you click **Next** again, the installer makes the changes that you want. Therefore, the source directory must be available to add components. You might want to copy that directory to a CD or DVD so that it does not take disk space on your system. Then, the reinstallation can be performed directly from the CD or the DVD.

Note: The installer uses the manifest files to know what is present or not, which means that $INFORMIXDIR must be set to the directory to be modified, and you must not change the manifest files outside of the installer. If $INFORMIXDIR is not set correctly or if the manifest files are modified, the installer might not perform as expected.

2.4 First steps after installation

After the installer has put the software in its proper location, you must perform additional configuration work based on your intended use for the IDS server.

2.4.1 The environment and user profiles

You must set the $INFORMIXDIR, $INFORMIXSERVER, and $PATH environment variables (or %INFORMIXDIR% and so forth on Windows) for each user who will use the IDS instance. You can set these environment variables in a number of ways, but you must set them.

On Linux for example, you might put the appropriate entries in the *.profile* or equivalent file for each user. Or, you might put them in */etc/profile* so that they apply to all users. Or, you might put them in the scripts or .rc files associated with the applications that use the IDS instance. There is no correct or preferred choice. Use whichever method is most convenient, meets your needs, and complies with your company's policies.

2.4.2 The configuration file

In this section, we discuss the configuration file items that you must change in order for the DBMS to function properly; however, we do not discuss the configuration file in great detail. We provide the minimum required information in order for you to successfully start the DBMS.

The items that you need to change include:

- ROOTPATH: The location of the instance catalogs and metadata. This is the *full path* to the file, logical volume, or device that contains the root dbspace.
- MSGPATH: Where to write the message log file. This is the full path to a file where IDS writes all its messages.
- DBSERVERNAME and DBSERVERALIASES: The instance name or names. These are the names that client programs or utilities use to connect to the instance. There must be a separate name for each protocol that is used. For example, if connections are made using shared memory and sockets, at least two names are needed. If the server has more than one IP address, separate names are needed for each IP address. Users can only connect using the names that have been defined and their associated protocols.
- DBSERVERNUM: The instance number. This must be unique among all the instances on the machine. This number is used as part of the shared memory key value.

- NETTYPE: Threads for managing connections.
- TAPEDEV and LTAPEDEV: Where to put database or log backups. These can be the path to an actual tape drive, the path to a directory, or /dev/null (on Linux or UNIX) or NUL (on Windows). On Linux or UNIX, they might be Standard Input/Output (STDIO) as well. Set the value based on where you plan to store your database and log archive files.

2.4.3 The SQLHOSTS file

In this section, we discuss the SQLHOSTS file, /etc/services, and /etc/hosts. Entries must be created in the $INFORMIXDIR/etc/sqlhosts file for each DBSERVERNAME or DBSERVERALIAS in the configuration file. Those entries usually include host names and service names from the /etc/hosts and /etc/services files. See the *Informix Dynamic Server Administrator's Guide,* G251-2681, for the details of the entries in the sqlhosts file.

2.4.4 Starting the IDS instance

With the software installed, the configuration file correctly changed, the root dbspace file or device created, and the environment set correctly, you can start the IDS server.

The program that starts the DBMS can be either $INFORMIXDIR/bin/oninit or %INFORMIXDIR%\bin\oninit.exe. The following examples assume that $INFORMIXDIR/bin or %INFORMIXDIR%\bin are in the $PATH or %PATH%.

Starting IDS for the first time

Starting IDS the first time is different from any other time, because the DBMS must create the first copy of all its metadata. You only run this command one time, because if you do it a second time, you destroy whatever you had.

The command to start the DBMS and create the first metadata is:

```
oninit -i or oninit -iv or oninit -ivy
```

The first form prompts to verify that you really want to initialize things. The second form prompts you and provides verbose messages about the tasks of starting the DBMS. The third form assumes that you answered `yes` to the prompt and provides the verbose output. You can use **`oninit -iy`** to assume the `yes` answer without the verbose output.

Getting the verbose output the first time allows you to more easily determine what is wrong if there is a problem.

Starting IDS any other time

After the first time, you want to start the DBMS so that it uses the data that existed when the DBMS was shut down.

The commands in this case are the same, but without the *-i.* This is extremely important, so *use the -i only if you wish to destroy all your existing data.*

Verifying correct startup

To be sure nothing is missing or incorrect, check the IDS message log. This is a text file located by the MSGPATH parameter in the configuration file. Check the messages there to be sure that all the functions and threads of the IDS instance started properly.

You can check the message log by either method:

- Use the command **`onstat -m.`**
- View the file with any text editor or text file viewer.

Viewing the file might be more useful, because the **`onstat`** utility displays only a limited number of the last lines of the file.

2.5 Administration and monitoring utilities

Each edition of Informix Dynamic Server includes a complete set of utilities to administer and monitor the instance. Most of the utilities are command-line-oriented, which fits the database server's UNIX or Linux heritage, but there are graphical utilities as well.

2.5.1 Command-line utilities

We briefly describe the command-line utilities in Table 2-2 on page 67. The *IBM Informix Dynamic Server Administrator's Guide,* G251-2681, has a complete explanation of each command-line utility.

Table 2-2 IDS command-line utilities

Utility	Description
oncheck	Depending on the options used, *oncheck* can check specified disk structures for inconsistencies, repair indexes that are found to contain inconsistencies, display disk structure information, and 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** flag, it initializes or reinitializes an instance. Depending on other flags, *oninit* can bring the instance from off-line 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 or disk.
onmode	Depending on the options used, *onmode* can change the instance operating mode after it is 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 High Availability Data Replication (HDR) functionality, dynamically change certain configuration, PDQ, **set explain**, and other parameters.
onparams	Used to add or delete logical logs, move and resize the physical log, and add a bufferpool to support standard dbspaces created with a larger than default page size.
onspaces	Used to manage the creation and deletion of chunks and dbspaces or 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 (also called the *sysmaster database*) also provides information about the instance and can be queried directly by using any SQL-oriented tool.

2.5.2 The OpenAdmin Tool for IDS

IBM provides a set of Hypertext Preprocessor (PHP) pages that is collectively called the *OpenAdmin Tool for IDS*. The OpenAdmin Tool for IDS provides a browser-based tool to perform many of the tasks involved in administering an IDS 11 instance. We have included the source code for these pages so that you can modify the tool to suit your needs.

Figure 2-8 on page 69 is one example of this tool. This example shows a query plan and the details of the query execution. There are many pages in this tool. Other pages include creating and monitoring storage space (chunks and dbspaces), as well as most of the instance monitoring that you can perform with the onstat utility.

This tool requires a browser, PHP, and a Web server. See the IBM developerWorks Web site for articles about setting up PHP and a Web server to work with IDS. Two very useful articles are at:

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

http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0607bombardier/

Figure 2-8 The OpenAdmin Tool

2.5.3 The Server Studio Java Edition (SSJE)

The second utility, *ServerStudio Java Edition (SSJE)*, was originally designed to be the complementary database administrator's 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. You can add more extensive instance administration and monitoring and trending functionality modules by purchasing those modules from an authorized IBM seller or reseller or directly from AGS, Ltd.

Figure 2-9 on page 70 depicts an example of the SSJE window; you do not need to be able to read everything.

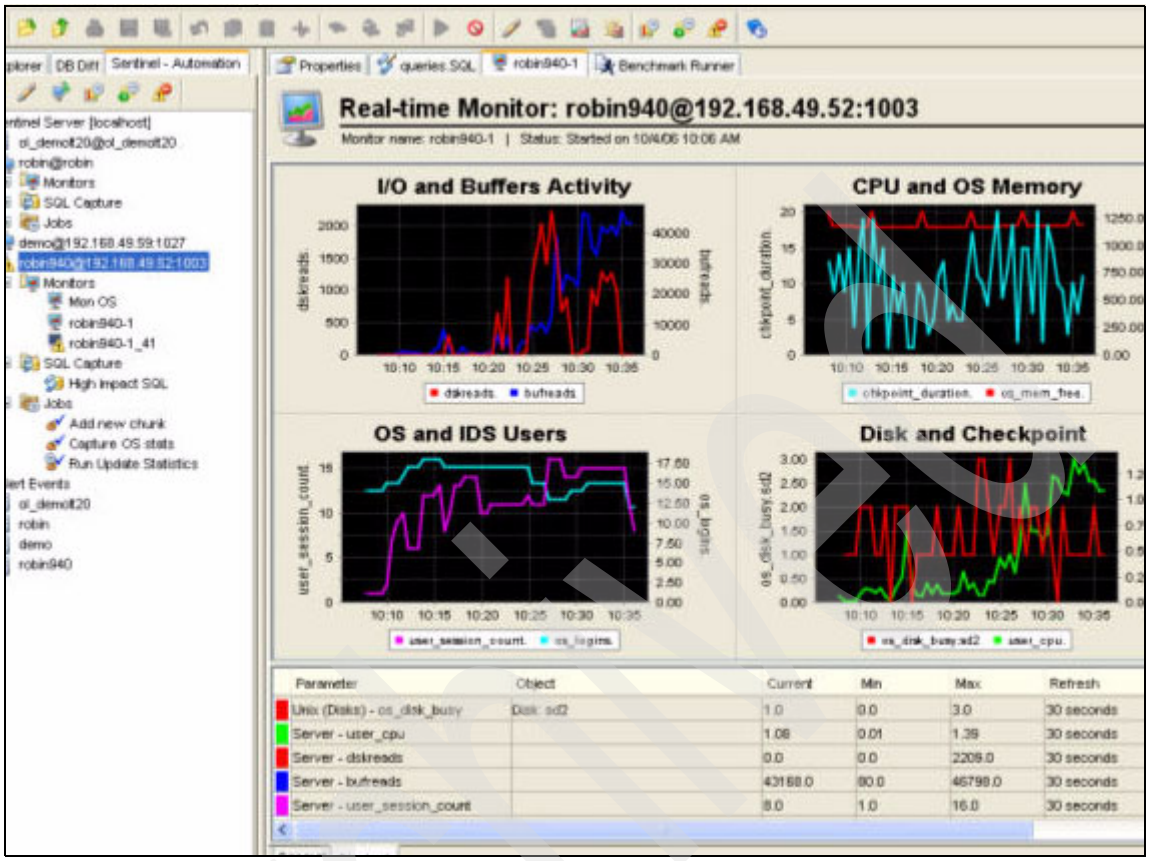


Figure 2-9 The SSJE real-time instance monitoring module

Figure 2-10 on page 71 is an example window that shows the SSJE database Entity Relationship diagram. Again, this is just for familiarity. We do not expect you to be able to read everything on this detailed window; the details come with the installation of the product.

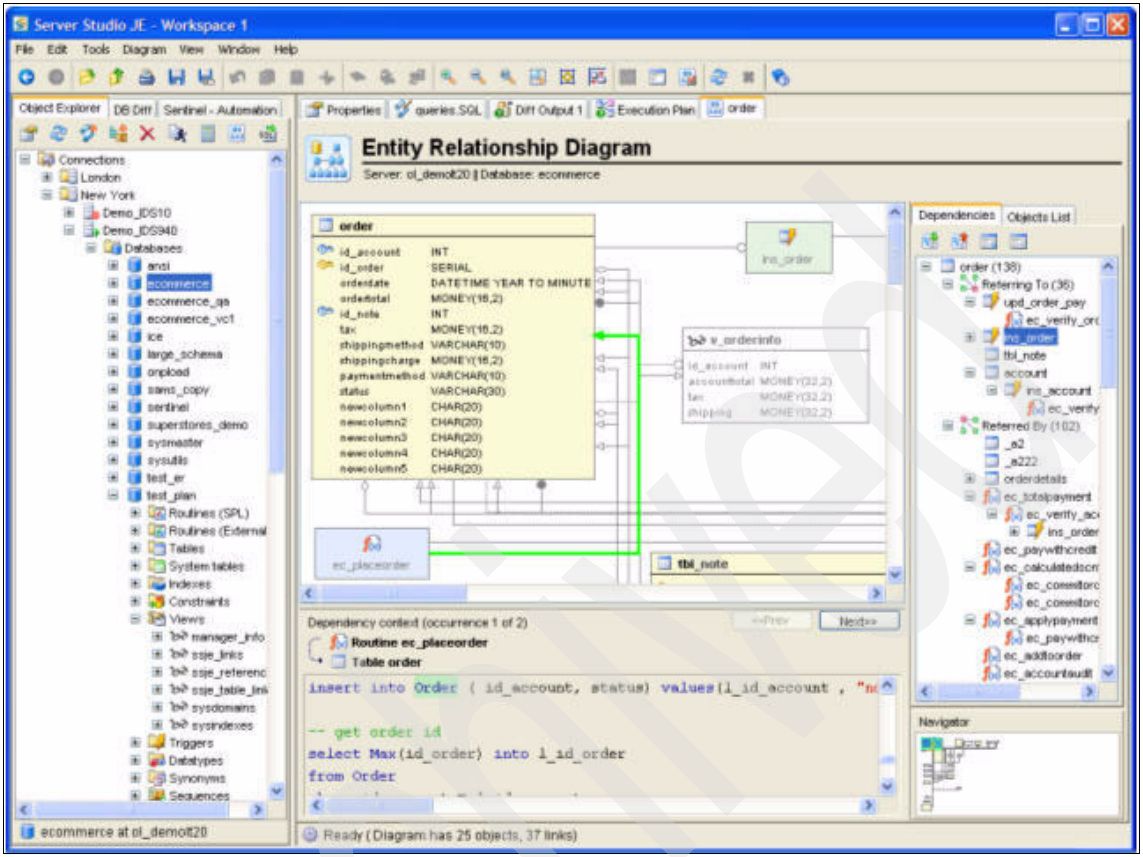


Figure 2-10 The SSJE database ER diagram

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

3

Configuration

In this chapter, we list and discuss the new configuration parameters that are available in IDS 11. We describe the parameters and provide examples for each parameter.

With each release of IDS, parameters are introduced that make managing a database instance easier. Many of the changes to the configuration parameters are intended to allow the DBA more control of the instance and to provide more of a *hands-free* approach. A hands-free approach means that the DBA does not always have be directly involved with required actions, IDS makes decisions and takes actions based on the configuration parameters.

For example, there are several automatic parameters (if enabled) that allow the DBA to have the instance self-manage certain constraints. If a DBA still wants to maintain control over all of the instance, the DBA can disable the automatic parameters.

The parameters that have been introduced in IDS 11 also allow a DBA to tune the instance in new ways. We discuss those options later in this chapter.

3.1 Reasons for changes

Each major release of IDS introduces new configuration parameters to enhance performance or administration or to make IDS more competitive.

We recommend that you read the Administration Guide and the Administration Reference manual for each version of the product if you are interested in learning more about the configuration parameters available in each release. We also recommend that you read the Performance and Tuning guides for IDS to avoid tuning an instance badly. The performance guides provide approaches to help you with overall tuning.

Each release, even minor releases, can have changes to the configuration parameters. We advise you to examine the release notes every time that you install the product. The IBM Web page contains the release notes of all supported versions. The link to this page is:

http://www-306.ibm.com/software/data/informix/pubs/library/notes/relnotes/docnotes.html

There can also be changes to the configuration parameters that are not documented in the release notes. If you have questions about any configuration parameters, we advise you to call the IBM Informix Technical Support team or to check Informix news groups.

We advise you to set up a test instance that is independent of your production environment. In this test instance, you can test changes to the configuration parameters without impacting the production environment. This testing allows the DBA an opportunity to learn more about the outcome of making changes to particular parameters.

3.1.1 Performance considerations

Many of the changes that have been made in Version 11 were to improve the overall performance of the engine.

Trying to eliminate the amount of time that the engine spends in a blocked mode during a checkpoint and also allowing the DBA to configure the engine to help reduce the amount of time spent in crash recovery are two of the primary performance enhancements made with this release.

Allowing the DBA to configure the amount of cache memory for a CPU Virtual Processor is another performance enhancement.

3.1.2 Administration considerations

The RTO_SERVER_RESTART configuration parameter is an example of both a performance and an administrative enhancement. The DBA can configure the amount of time that the engine spends in crash recovery with this parameter.

The dynamic log feature was introduced in Version 10 of IDS and allows the DBA to configure how the engine must respond if the logical logs become full.

The changes are designed to allow the DBA more flexibility and a greater understanding of what can and does happen within the instance.

The AUTO_ config parameters that we describe in this chapter are designed to allow for more automatic control of the instance. The DBA can turn these features off if desired.

3.2 Recovery time objective

Having an instance go offline unexpectedly can be an interesting experience for a DBA. The cause for an instance going offline can be hardware-related, software-related, or a combination of the two situations. When the instance crashes, it is no longer accessible. To make data in the instance accessible again, the instance must be brought online. The process that the instance has to go through to get back online is called *crash recovery*. At the end of crash recovery, the instance is at a point of consistency and data can be accessed again. One of the advantages of IDS is that the DBA can configure the instance in whatever fashion is best for that particular environment. If the DBA wants the instance to rarely perform checkpoints, the DBA can configure the instance that way. If the DBA wants the instance to perform checkpoints frequently, the DBA can configure the instance to do that. However, the choice of either style of configuration has trade-offs. One of the trade-offs can be implicit, that is, the amount of time that the instance potentially spends in crash recovery if the instance has to go through that process.

In previous versions of IDS, the amount of time that the engine spent in crash recovery was unpredictable and not explicitly configured. If an instance happened to crash at the wrong moment, the amount of time that the engine spent getting back to a known point of consistency was potentially great. With Recovery Time Objective (RTO), the DBA can configure the instance to try to control the amount of time that the instance might need if crash recovery is required. Also, the objective is to not negatively impact transaction performance of the instance while it is online.

3.2.1 Recovery

Crash recovery is the process that the instance must go through whenever the engine is not taken offline by design. Crash recovery can happen due to events, such as a power failure, hardware failure, or disk failure, and usually, is an unexpected event.

The engine must be brought back online using the crash recovery process. The starting point for crash recovery is stored in the reserve pages on the checkpoint and logical log reserve pages. To examine this value, execute an **oncheck -pr**.

The crash recovery process in IDS is similar to *fast recovery*. Fast recovery is an automatic procedure that restores the database server to a consistent state after it goes offline under uncontrolled conditions. Fast recovery also rolls forward all committed transactions since the last checkpoint and rolls back any uncommitted transactions.

The difference between the two recover processes is the method by which the engine came offline. With crash recovery, it is assumed the engine came offline unexpectedly.

The engine will go through fast recovery, which consists of two parts: physical recovery and logical recovery. The first part, physical recovery, consists of the engine going through any pages that are in the physical log and replacing them on disk. At each checkpoint, one of the steps is to empty the physical log. With a crash recovery, there are pages in the physical log to replay. An engine that came offline cleanly with an **onmode -uky** has no pages in the physical log to replay. After all the pages in the physical log have been replaced on disk, physical recovery is complete.

At this point, logical recovery starts. Logical recovery consists of two parts: roll-forward and roll-back. During the roll-forward phase, the logical logs are replayed from the starting position listed in the reserve page until the end of the logical logs. After all the log information has been applied, the roll-forward phase is complete. Then, the engine takes any open transactions and rolls the open transactions back. When the roll-back phase is complete, any temporary tables are dropped and then the engine is placed in online mode. See Example 3-1 for details of how the online log looks during crash recovery.

Example 3-1 Crash recovery

```
13:02:25  IBM Informix Dynamic Server Started.

Tue May 15 13:02:25 2007

13:02:25  Event alarms enabled.  ALARMPROG =
'/usr/informix/etc/alarmprogram.sh'
```

```
13:02:25  Dynamically allocated new virtual shared memory segment (size 8416KB)
13:02:25  Memory sizes:resident:13148 KB, virtual:16608 KB, no SHMTOTAL limit
13:02:25  Booting Language <c> from module <>
13:02:25  Loading Module <CNULL>
13:02:25  Booting Language <builtin> from module <>
13:02:25  Loading Module <BUILTINNULL>
13:02:30  DR: DRAUTO is 0 (Off)
13:02:30  DR: ENCRYPT_HDR is 0 (HDR encryption Disabled)
13:02:30  Dynamically allocated new message shared memory segment (size 548KB)
13:02:30  Memory sizes:resident:13148 KB, virtual:17156 KB, no SHMTOTAL limit
13:02:30  Fast poll /dev/poll enabled.
13:02:30  IBM Informix Dynamic Server Version 11.10.FB6TL Software Serial
Number
 AAA#B000000
13:02:31  IBM Informix Dynamic Server Initialized -- Shared Memory Initialized.

13:02:31  Physical Recovery Started at Page (1:791).
13:02:31  Physical Recovery Complete: 180 Pages Examined, 180 Pages Restored.
13:02:31  Logical Recovery Started.
13:02:31  10 recovery worker threads will be started.
13:02:31  Dynamically allocated new virtual shared memory segment (size 8192KB)

13:02:31  Fast Recovery Switching to Log 11
13:02:31  Process exited with return code 127: /bin/sh /bin/sh -c
/usr/informix/
etc/alarmprogram.sh 3 7 "Dynamic Server Initialization failure." "WARNING!
Physi
cal Log
13:02:34  Logical Recovery has reached the transaction cleanup phase.
13:02:34  Logical Recovery Complete.
          236 Committed, 0 Rolled Back, 0 Open, 0 Bad Locks

13:02:34  Dataskip is now OFF for all dbspaces
13:02:34  Checkpoint Completed:  duration was 0 seconds.
13:02:34  Checkpoint loguniq 11, logpos 0x12d018, timestamp: 0x33811 Interval:
9
2

13:02:34  Maximum server connections 0
13:02:34  Checkpoint Statistics - Avg. Txn Block Time 0.000, # Txns blocked 0,
P
log used 295, Llog used 1

13:02:34  On-Line Mode
```

3.2.2 Configuring for performance

To monitor if the instance has been configured for optimal performance, onstat **-g ckp** can provide assistance. For more details, examine Example 3-2.

This feature in **onstat** is new to Version 11. It presents a summary of checkpoint-related information. In previous versions, you were required to run required different **onstat** commands, combine the information, and also include information that was not included in any **onstats** in previous versions.

Example 3-2 Using onstat -g ckp to monitor the engine

```
$ onstat -g ckp

IBM Informix Dynamic Server Version 11.10.FB6TL -- On-Line -- Up 00:11:23 --
39496 Kbytes

AUTO_CKPTS=Off   RTO_SERVER_RESTART=Off

Critical Sections                Physical Log    Logical Log
          Clock                                  Total Flush Block #      Ckpt
Wait  Long  # Dirty   Dskflu  Total    Avg    Total    Avg
Interval   Time      Trigger    LSN                 Time  Time  Time  Waits  Time
Time  Time  Buffers   /Sec    Pages    /Sec   Pages    /Sec
111        09:01:56  Startup    10:0x28f018       0.3   0.0   0.0   0      0.0
0.0   0.0   0         0       0        0      1        0
112        09:06:58  CKPTINTVL  10:0x374018       0.1   0.1   0.0   0      0.0
0.0   0.0   77        77      61       0      229      0

Max Plog       Max Llog       Max Dskflush   Avg Dskflush   Avg Dirty
Blocked
pages/sec      pages/sec      Time           pages/sec      pages/sec      Time
200            200            0              77             0              0

Based on the current workload, the physical log might be too small
to accommodate the time it takes to flush the bufferpool during
checkpoint processing.  The server might block transactions during checkpoints.
If the server blocks transactions, increase the physical log size to
at least 14000 KB.
$
```

Monitoring the engine can ensure that checkpoints are performed efficiently. The **onstat -g ckp** monitors the instance for the previous 20 checkpoints.

From a diagnostic standpoint, collecting information iteratively assists you in determining if a given approach is working. Having information about the previous 20 checkpoints available makes problem determination easier when you try to diagnose performance problems.

This also helps the DBA and Informix Technical Support to determine whether a performance problem with checkpoints is a one time occurrence.

3.3 Checkpoints

Checkpoints are where the information that is in shared memory is reflected on disk. At the exact instance that a checkpoint completes, the information in shared memory is exactly what is on disk. So, if the engine crashed at the exact instant that a checkpoint completed, the engine in theory has no work to do when coming back online.

Checkpoints are also considered known points of consistency within the engine. When an engine is brought offline and then brought back online, the starting place for fast recovery is the last known checkpoint.

Because checkpoints reflect the work that has been done in an engine, an extremely busy engine can have many buffers, or pages, to copy from memory to disk. The actual writing of the information to disk is an extremely expensive operation in terms of computing cycles. With a busy engine, the number of buffers to copy to disk can be quite large and the time to perform the copy can be long.

Traditional checkpoints also blocked new buffers or pages from being modified when a checkpoint was in process, so that the time that the engine takes to copy the current modified or dirty buffers to disk can impact the performance of users trying to modify buffers or pages in the engine. In an online transaction processing (OLTP) environment, the delay of waiting for a checkpoint to complete can result in a performance issue for application users.

Traditionally, there have been four events that can trigger a checkpoint:

- Administrative events can trigger a checkpoint.
- A physical log becoming 75% full triggers a checkpoint.
- A checkpoint in the logical log space. IDS cannot overwrite the logical log containing the most recent checkpoint so it has to trigger a checkpoint prior to moving into that logical log.
- ONCONFIG parameter CKPTINTVL triggers a checkpoint.

3.3.1 Traditional checkpoints

The traditional checkpoints are also known as *blocking checkpoints*, because transactional updates are blocked for a short time, usually less than a second,

while a checkpoint is started. The transactions are then free to perform updates while the bufferpool containing all the transactional updates is flushed to disk. The steps of a traditional checkpoint are:

1. The checkpoint request is issued.
2. The critical section is blocked.
3. There is a wait for sessions to exit the critical section.
4. The physical log buffer is flushed to disk.
5. The dirty buffers are flushed to disk.
6. A checkpoint record is written to the logical log buffer.
7. The logical log buffer is flushed to disk.
8. The block on the critical section is released.

In step 2, the instance allows no new user to enter the critical section of the instance until step 8 is complete. Because the time required for the completion of step 3 might not be predictable, the entire time an instance can be in a checkpoint can be difficult to predict. If an application or user enters a critical section and for whatever reason does not exit the critical section, the instance can essentially be stopped or *hung*. That is, processing stops and cannot resume until completion of a specific event. It is extremely important to configure the instance and the applications to detect if any user or application is staying in the critical section and does not exit.

3.3.2 Fuzzy checkpoints

Introduced in Version 9.20 of IDS, *fuzzy checkpoints* were designed to reduce the amount of time that the engine spent in a blocked state. Fuzzy checkpoints were intended to be a multiple phase implementation, with the first phase consisting of index operations.

With fuzzy checkpoints, the entire list of modified or dirty buffers was not flushed to disk during a checkpoint. A dirty page table (DPT) was created, which consisted of the modified buffers that were not flushed to disk. An algorithm determined when enough events had taken place so that the buffers in the DPT were eventually flushed to disk. One of the issues with fuzzy checkpoints is that while performance with the engine online might be improved, the cost of that performance came during the logical recovery phase of fast recovery. The thought was that the amount of extra time that the engine spent in recovery was worth the performance increase gained when the engine was online and performing checkpoints.

However with the advent of faster and less expensive resources and non-blocking checkpoint technology, there was no longer a requirement for fuzzy checkpoints.

3.3.3 Current issues

The time that the engine spends in a blocked mode waiting for a checkpoint to complete is more important now. With the number of OLTP environments where having the engine blocked for a few seconds causes performance or usability issues, it is more important that the engine can be tuned to reduce the blocking time as much as possible.

Furthermore, when an engine is offline for any reason, extending the time that the engine needs to come online, even for a few seconds, is also a critical factor.

3.3.4 Changes to checkpoints

With IDS 11, changes to checkpoints now include the ability for a checkpoint to essentially be non-blocking. By not blocking during the flushing of a bufferpool, the least recently used (LRU) flushing can be tuned to occur less often, which, in turn, provides more CPU time to an application to handle transactions. Because checkpoints do not block updates, checkpoints can be done more frequently, which helps meet RTO tuning objectives.

In previous versions of IDS, tuning the LRU queues minimum and maximum values was required to improve checkpoint duration. However, tuning the LRUs to a favorable value for checkpoints impacted the time of user transactions. With the new algorithm, there is less impact on transactions and checkpoint durations.

3.3.5 RTO checkpoints

A new configuration parameter, RTO_SERVER_RESTART, allows the DBA to configure the engine so that the time spent in fast recovery is controlled. The trade-off is that more frequent checkpoints might occur.

3.3.6 Managing checkpoints

With previous versions of the IDS engine, a DBA attempted to minimize the amount of time that the engine spent in a blocked state, but there was not a direct parameter to help accomplish this goal. The checkpoint interval was controlled, but there are situations when enough resources have been consumed that a checkpoint request is automatically issued. In previous versions, the only direct configuration option available to the DBA was to tune the LRU queues. The primary objective of tuning the LRU queues is quicker checkpoints. However, a trade-off is made of impacting transaction work by flushing the buffers to disk between checkpoints.

3.4 CPU Virtual Processors

CPU Virtual Processors (VPs) are dedicated to CPU tasks. You can configure the number of CPU VPs by both the onconfig file and through the **onmode** command. Having too few or too many CPU VPs configured on an instance can have performance implications.

In previous versions of IDS, other than configuring the number of CPU VPs and processor affinity, there was not much else to tune in terms of the CPU VPs. One of the new features in IDS 11 allows you to allocate private memory cache for CPU VPs. This increased capability allows the DBA greater flexibility to tune the instance.

Previously when working with memory, the server performed a bitmap search for memory blocks. However, this was limiting, because mutexes/latches are required, which reduces concurrency and can slow large multiprocessor machines with large instance memory. Memory blocks are of a requested length and threads can back up on a single bitmap search. So, there was really no way to tune the memory search.

With the new IDS 11 feature, this tuning can be done both dynamically and by a static method. The **onstat** command has also been modified to allow this memory to be monitored. The **onstat** command is demonstrated in Example 3-7 on page 84.

3.4.1 An overview

A CPU VP can:

- Run all session threads and certain system threads
- Run thread for kernel asynchronous I/O (KAIO) where available
- Run a single poll thread, depending on configuration

When the engine is started, the CPU VP 1 is always the first VP. To examine the other Virtual Processors, use the following command:

```
onstat -g sch
```

This command displays the VPs. The command output is depicted in Example 3-3 on page 83.

Example 3-3 onstat -g sch output

```
C:\Program Files\IBM\IBM Informix Dynamic Server\11.10>onstat -g sch

IBM Informix Dynamic Server Version 11.10.TB6TL   -- On-Line -- Up 00:11:53 --
1696 Kbytes

VP Scheduler Statistics:
 vp    pid       class       semops    busy waits  spins/wait
 1     6412      cpu         1972      0           0
 2     7344      adm         0         0           0
 3     6960      lio         2         0           0
 4     1852      pio         2         0           0
 5     3420      aio         826       0           0
 6     6624      msc         37        0           0
 7     4548      soc         7         0           0
 8     540       soc         1         0           0

Thread Migration Statistics:
 vp  pid      class steal-at steal-sc idlvp-at idlvp-sc inl-polls Q-ln
 1   6412     cpu   0        0        0        0        0         0
 2   7344     adm   0        0        0        0        0         0
 3   6960     lio   0        0        0        0        0         0
 4   1852     pio   0        0        0        0        0         0
 5   3420     aio   0        0        0        0        0         0
 6   6624     msc   0        0        0        0        0         0
 7   4548     soc   0        0        0        0        0         0
 8   540      soc   0        0        0        0        0         0
```

3.4.2 Traditional methods of managing CPU VPs

When the engine is online, the CPU VPs can be increased or decreased by using the **onmode** command. Example 3-4 shows how you can reduce the number of CPU VPs by 2.

Example 3-4 Reducing the number of CPU VPs by 2

```
% onmode -p -2 cpu
```

Another method is to edit the onconfig and then cycle the instance. The parameter in the onconfig file is NUMCPUVPS. In previous versions of IDS, there was no other method to manage or control CPU VPs.

3.4.3 Making changes

One of the changes in IDS 11 is to the memory cache available for CPU VPs. VP_MEMORY_CACHE_KB is an onconfig parameter and also can be modified online with the **onmode -wm** or **onmode -wf** option. If an environment exists where there are multiple CPUs available with large amounts of memory, performance can be improved. There are expectations that with as few as four CPUs, a performance benefit can be realized. Example 3-5 shows a static method for setting the VP memory cache.

Example 3-5 Static method for setting VP_MEMORY_CACHE_KB

```
Onconfig Parameter - VP_MEMORY_CACHE_KB
0 to turn off
800 is the default minimum value (800 KB per CPU VP)
Maximum value is VP_MEMORY_CACHE_KB * number of CPU VPs
Total amount used should not exceed 40% of SHMTOTAL
```

The method that has been chosen for Private Memory Cache on CPU VPs has the following considerations:

- List of blocks memory is 1 to 32 blocks in length per list.
- Block size is 4096.
- This is associated with each CPU VP allocated.
- This method speeds up access and performance of the CPU VPs.
- There is much less latching.

In Example 3-6, we depict a dynamic method for setting the VP memory cache.

Example 3-6 Dynamic method for setting VP_MEMORY_CACHE_KB

```
Onmode –wm VP_MEMORY_CACHE_KB= value (KB)
Onmode –wf VP_MEMORY_CACHE_KB= value (KB)
Setting value to 0 disables feature and empties caches
-wm option – valid for current running instance only Message written to message
log file and command line
-wf option – rewrites onconfig setting Message written to message log file
```

The output from the **onstat -g vpcache** command is shown in Example 3-7. The column descriptions listed in the output are also defined in the lower portion of the output.

Example 3-7 onstat -g vpcache

```
onstat -g  vpcache

IBM Informix Dynamic Server Version 11.10.TB6TL   -- On-Line -- Up 00:01:37 --
21696 Kbytes
```

```
CPU VP memory block cache statistics - 4096 byte blocks

Number of 4096 byte memory blocks requested for each CPU VP:0

vpid    pid         Blocks held  Hit percentage   Free cache
1       6412        0              0.0 %             0.0 %

Current VP total allocations from cache:        0
    size  cur blks   alloc      miss       free      drain     release
    1     0          0          0          0         0         0
    2     0          0          0          0         0         0
    3     0          0          0          0         0         0
    4     0          0          0          0         0         0
    5     0          0          0          0         0         0
    6     0          0          0          0         0         0
    7     0          0          0          0         0         0
    8     0          0          0          0         0         0
    9     0          0          0          0         0         0
    10    0          0          0          0         0         0
-----deleted-----

    31    0          0          0          0         0         0
    32    0          0          0          0         0         0

size: size of memory blocks in 4096 byte blocks
cur blks:current number of 4096 blocks - multiple of size field
alloc:number of times we gave a requestor a block of this size
miss:number of times block was requested but none were available
free:number of times we placed a memory block into the cache
drain:number of time we forced an aged block out to make room
release:number of times the size was full - couldn't insert
```

3.5 Dynamic logging

Logical logs are where the transaction work is stored, are configured when an instance is created, and are reused after they have been backed up. The size and number of the logical logs are tunable. The minimum number of logical logs in an instance is three. To determine an optimal number of logical logs, consult the *Informix Administration Reference Guide*, G251-2268-01.

Logical logs allow the instance to roll forward during a restore to a later point-in-time than when the backup was performed. To roll forward through logical logs during a restore requires the logical logs to be maintained by the

instance and backed up through one of the supported mechanisms. If the logical logs are not backed up, a roll-forward from a restore is not possible.

There are performance implications to having too few or too many logical logs configured in an instance. The **onparams** command allows the DBA the option to add or remove logical logs to an instance. This is a manual command that requires the DBA to enter the command with its options.

Replication either through High Availability Data Replication (HDR) or Enterprise Replication (ER) is achieved through the logical logs. The logical log configuration can impact the performance of replication.

Information that is written to the logical log is under the control of the instance. The logging style of a database controls the amount and type of information that particular database sends to the logical logs. It is not an IDS requirement that each database within an instance has the same style of logical logging.

Introduced in Version 9.40 of IDS, the dynamic logging feature allows the DBA to configure the instance to automatically add logical logs if needed. The assumption is that available space exists in the instance to add a logical log. The instance can be configured to add the logs automatically or to notify the DBA and wait for the DBA to add the needed log. It is also possible to turn this feature off and have the instance behave as in older versions of the product.

With the proper tuning and configuration of the instance, dynamic logging can help prevent the instance from being inaccessible through long transactions.

3.5.1 Managing logical logs

There exists many different strategies for managing logical logs. Depending upon the situation, the strategies can have different implications on an instance. Setting the LTAPEDEV to either /dev/null or /NUL means that the logical logs are not backed up when filled. This can result in faster performance but can also have an impact depending upon the backup strategy employed. It usually results in data loss to have an instance configured this way if the need to perform a restore is ever encountered. We recommend that you thoroughly understand both the positive and negative implications before modifying this parameter.

Logical logs do take disk space through the chunks that are part of dbspaces. The number of logical logs and their location can be controlled by the DBA. The amount of information that is written to the logical logs is a function of the type of logging chosen for a database and the amount and type of work that a particular database has performed against it.

Logical logs are used in a circular fashion, which means that the space is reused automatically. The logs need to be backed up to be reused, and IDS does not reuse a logical log until it is backed up. You can set the LTAPEDEV configuration parameter to a null setting to instruct the instance to not back up used logical logs.

Important: It is important that you thoroughly understand the implications of setting LTAPEDEV to /dev/null or /NUL before actually changing LTAPEDEV.

The method chosen for the backup of the logical logs and the number and size of the logical logs determine the actual performance of using the logical logs. If a form of replication is involved, the configuration of the logical logs can dramatically impact the performance of replication, because the IDS replication styles are logical log-based.

To automatically back up logical logs, edit the alarmprogram script and set BACKUPLOGS=Y.

3.5.2 Traditional method for managing logical logs

When the instance is initialized, the number and size of the logical logs are configured by the configuration file and are placed in the root dbspace. To manage adding or removing logical logs after the instance has been initialized, the command-line interface *onparams* is used. The location of the logical logs can be controlled by **`onparams`**. To move logical logs to a new dbspace, the new logs can be created in the desired new dbspace and the original logical logs can then be deleted. However, logical logs cannot be added or created in a temporary dbspace. The utility also lets you manipulate and remove logical logs as well.

Restriction: An IDS instance requires at least three logical logs.

There are performance reasons to move the location of the logical logs to a different dbspace than the root dbspace. When a restore of an instance is required, the dbspaces containing the logical logs are considered critical dbspaces. A critical dbspace cannot be warm-restored. You can read more details about backup and restore strategies in the *Informix Backup and Restore Guide*, G251-2269.

After logical logs are added or removed, it is a good idea to perform a level 0 backup of the instance, although it is not required.

Adding logical logs is not always required to maintain an instance, because the logical logs are reused after they have been backed up.

Limitations of the traditional method

The DBA is required to monitor the instance closely to determine if a situation has been encountered that requires the addition of logical logs. The script log_full.sh (on UNIX or Linux systems) or log_full.bat (on Windows) can be modified to notify the DBA of the frequency with which the logical logs are filling up. But, adding logical logs requires DBA intervention to run the actual command.

Starting in Version 9.40, you can configure IDS to add logical logs dynamically. This can be helpful, because no DBA intervention is required. If a user has created a situation where a long transaction exists, you can avoid the problem of filling the logical logs and hanging the engine with dynamic logical logs. If a long transaction is detected and sufficient space exists in permanent non-BLOB or non-smartBLOB chunks, the engine can automatically add an additional logical log. This allows the DBA an opportunity to monitor the engine to try to determine what needs to be done with the long transaction operation and also to allow other users to continue to process work normally.

Sometimes, the instance is in a state that the logical logs can only be added by Informix Technical Support. There are also limitations that can prevent the Support Engineer from being able to add the logical log. If you have questions about these situations, contact Informix Technical Support.

3.5.3 Dynamic management of logical logs

When configuring for dynamic management of logical logs, there is one onconfig parameter, DYNAMIC_LOGS, which is depicted in Example 3-8.

Example 3-8 Valid values for DYNAMIC_LOGS

```
# DYNAMIC_LOGS:
#    2 : server automatically add a new logical log when necessary. (ON)
#    1 : notify DBA to add new logical logs when necessary. (ON)
#    0 : cannot add logical log on the fly. (OFF)
```

The DBA does have the flexibility to either add logical logs manually or to be notified by the engine that DYNAMIC_LOGS is set to 1. With DYNAMIC_LOGS set to 2, the engine automatically adds the logical logs. With the parameter set to 0, the dynamic logs feature is disabled.

Example 3-9 on page 89 shows what the online log contains when DYNAMIC_LOGS are set to 1 and the logical logs are full.

Example 3-9 DYNAMIC_LOGS set to 1

```
16:32:21  Btree scanners disabled.
16:32:22  Btree scanners enabled.
16:32:24  Logical Log Files are Full -- Backup is Needed
```

3.6 AUTO_* Parameters

Another feature of IDS 11 is the ability to have the engine assist the DBA in automatically helping to tune the engine performance. There are four configuration parameters that are designed to assist in this function. They are:

- AUTO_LRU_TUNING
- AUTO_AIOVPS
- AUTO_CKPTS
- AUTO_REPREPARE

You can turn off all of the automatic configuration parameters if you want. The goal is to allow more DBA control over how much of the instance is managed and tuned automatically. By using these parameters, a DBA can focus on other activities.

As with any other configuration parameters, we suggest that a DBA create a test environment to determine through the DBA's own testing what the optimal configuration is for the DBA's particular environment.

3.6.1 AUTO_LRU_TUNING

The AUTO_LRU_TUNING configuration parameter is used to enable or disable automatic LRU tuning when the database server starts.

To turn off automatic LRU tuning for a particular session, issue the command:

```
onmode –wm AUTO_LRU_TUNING0
```

To turn on automatic LRU tuning after turning it off during a session, issue the command:

```
onmode –wm AUTO_LRU_TUNING1
```

Automatic LRU tuning changes affect all bufferpools and the adjust lru_min_dirty and lru_max_dirty values in the BUFFERPOOL configuration parameter.

For more information about LRU tuning, see the *IBM Informix Dynamic Server Performance Guide*, G251-2296.

3.6.2 AUTO_AIOVPS

The AUTO_AIOVPS configuration parameter enables or disables the database server to automatically increase the number of AIO VPS and flusher threads when the server detects that AIO VPs are not keeping up with the I/O workload. Disable this parameter if you want to manually adjust the number. For details about setting this parameter, see the *IBM Informix Administration Reference Guide*, G251-2268-01.

You can use the AUTO_AIOVPS configuration parameter to enable the database server to automatically increase the number of AIO Virtual Processors and page-cleaner threads when the server detects that AIO Virtual Processors are not keeping up with the I/O workload.

3.6.3 AUTO_CKPTS

Automatic checkpoints cause the database server to trigger more frequent checkpoints to avoid transaction blocking. They attempt to monitor system activity and resource usage (physical and logical log usage along with how dirty the bufferpools are) to trigger checkpoints in a timely manner so that the processing of the checkpoint can complete before the physical or logical log is depleted.

The database server generates at least one automatic checkpoint for each span of the logical-log space. This guarantees the existence of a checkpoint where fast recovery can begin.

Use the AUTO_CKPTS configuration parameter to enable or disable automatic checkpoints when the database server starts. You can dynamically enable or disable automatic checkpoints by using **`onmode -wm`** or **`onmode -wf`** commands.

3.6.4 AUTO_REPREPARE

AUTO_REPREPARE controls whether a Dynamic Server feature is in effect that automatically re-optimizes Stored Procedures Language (SPL) routines and re-prepares prepared objects after the schema of a table referenced by the SPL routine or by the prepared object has been changed.

When AUTO_REPREPARE is disabled, if Data Definition Language (DDL) statements such as CREATE INDEX, DROP INDEX, DROP COLUMN, and RENAME COLUMN are executed, users of prepared objects that reference the modified tables or SPL routines that reference the modified tables indirectly receive -710 errors the next time that they execute the prepared object or the SPL routine. Enabling AUTO_REPREPARE can avoid many -710 errors and can

reduce the number of reprepare and reoptimize operations that users must perform manually after the schema of a table is modified.

3.7 Distributed Relational Database Architecture

Distributed Relational Database Architecture™ (DRDA) is a protocol that enables communication between Dynamic Server applications and IBM Data Server Clients. DRDA is for open development of applications that allow access of distributed data.

DRDA allows you to access data in a distributed relational database environment by using SQL statements in your applications. The architecture has been designed to allow distributed data access for systems in like and unlike operating environments, which means that your applications can access data residing on homogeneous or heterogeneous platforms.

DRDA is based on these IBM and non-IBM architectures:

- SNA Logical Unit Type 6.2 (LU 6.2)
- TCP/IP Socket Interface
- Distributed Data Management (DDM) architecture
- Formatted Data Object Content Architecture (FD:OCA)
- Character Data Representation Architecture (CDRA)

SQL has become the most common data access language for relational databases in the industry. It was chosen as part of DRDA because of its high degree of standardization and portability. In a distributed environment, where you want to access data at remote locations, the SQL requests are routed to the remote systems and they are executed remotely. Prior to sending the remote SQL request, a DRDA application must establish a connection with the remote relational database where the data is located. This is the purpose of the CONNECT SQL statement provided by DRDA.

3.7.1 An overview

You can configure IDS to respond to requests from IDS ANSI databases to use IBM Data Server Clients, such as Java Common Client (JCC), DB2 CLI Driver, .NET, and OLEDB. These clients all use the DRDA protocol to communicate with database servers, and IDS understands this protocol as well as its native SQLI (SQL Interface) protocol, which is used by tools, such as the Informix Client SDK.

If you plan to communicate using DRDA, you must have installed an IBM Data Server Client and the applicable driver (for example, the DB2 Run-Time Client or

DB2 Universal JDBC Driver). For information about how to install client software, see an installation guide for the software such as the *IBM DB2 Universal Database: Quick Beginnings for DB2 Clients*, GC09-4832.

Important: You cannot have encrypted data in communications with DRDA®.

3.7.2 Configuring DRDA

In this section, we explain how to connect IDS to IBM Data Server Clients.

The prerequisites include an installed IBM Data Server Client and the applicable drivers.

To enable IDS to connect to an IBM Data Server Client:

1. Configure a new server alias in the SQLHOSTS file or Windows registry that uses either the *drtlitcp* or *drsoctcp* connection protocol. See Example 3-10 for more sample DRDA entries.
2. Be sure that the ONCONFIG file lists the DRDA connection as one of the server aliases. The alias must not be the DBSERVERNAME.
3. Use the DRDA Version 3 protocol to connect to IDS.

Important: Each SQLHOSTS file entry or SQLHOSTS REGISTRY key must have a unique service name that is equivalent to a TCP/IP port number. DRDA clients must use separate ports from the SQLI clients, because the DRDA listener thread does not understand any connection attempt by an SQLI client, nor does an SQLI listener thread understand any connection attempt by a DRDA client.

Example 3-10 SQLHOSTS entries for DRDA

```
servicename_drda drtlitcp username_machine name_drda_socketnum
servicename_drda drsoctcp username_machine name_drda_socketnum
```

3.7.3 Managing a DRDA environment

When using DRDA, all IDS SQL DDL statements are supported. The SQL statements supported by IDS have a slightly different syntax from the equivalents in DB2. In IDS DRDA queries, statements that are present in DB2 but not present in IDS are not supported.

Use the following **onstat** and **onmode** commands to display information that includes the DRDA thread name and an indicator that distinguishes SQLI and DRDA sessions:

- **onstat -g ses**
- **onstat -g sql**
- **onstat -g ath**
- **onstat -g stk**
- **onstat -u**
- **onstat -x**
- **onstat -G**
- **onstat -g ddr**
- **onstat -g env**
- **onstat -g stm**
- **onstat -g ssc**
- **onmode -D**
- **onmode -Z**

3.8 New parameters or variables

With each new version of IDS comes new parameters, which can be changes to the onconfig file or environment variables. We discuss those new parameters in this section and also list known config parameters that are being deprecated.

3.8.1 Onconfig parameters

In IDS 10.00, a new parameter was introduced to the oninit startup options. It was *oninit -j*. The j allowed the engine to come to an online mode, but was considered an administrative mode. This allowed the DBA who connected as user *informix* to perform administrative tasks to the instance without having other users able to connect. In IDS 11, an addition to this option is introduced in the onconfig file.

The parameter ADMIN_MODE_USERS is a list that is comma-separated user names that can connect to the instance other than user *informix* when the engine is in administrative mode. If the DBA is not interested in having additional users other than informix, this parameter can be ignored. There is no impact on the performance of the instance regardless of the setting of this parameter. As an example, consider the following:

```
# ADMIN_MODE_USERS
# This is a comma separated list of those users who can connect to
theIDS server while it is in single user mode. This list of users is
in addition to user informix.
```

There are three types of shared memory segments within IDS: Resident, virtual, and messaging. The virtual segment is controlled by the SHMVIRTSIZE configuration parameter. If the amount of memory in the virtual segment is consumed and more memory is needed, assuming the SHMTOTAL is not exceeded, another virtual segment is allocated. Another new parameter, SHMVIRT_ALLOCSEG, allows the DBA to configure when the new virtual segment is allocated. This feature can be turned off by setting the parameter to 0. The advantage of using this parameter is that the new segment can be allocated before the virtual segments are exhausted. There is a performance increase on busy systems that require virtual segments to be allocated. And, there is an ability with this parameter to also adjust the alarm level. The alarm level is what is used in the ALARMPROGRAM if the DBA has configured it for use. The ALARMPROGRAM can be used to notify the DBA or others that certain events have happened within the instance. For further details about configuring the ALARMPROGRAM, see the *Informix Administration Reference Guide*, G251-2268-01. The following is an example of that parameter:

```
SHMVIRT_ALLOCSEG 0
Values between 0 and .99 are %, values > 1 are KB - when this much
virtual memory is used we try to get a new segment. 0 means "off".
The 2nd parameter is the alarm level.
```

The Shared Disk Secondary (SDS) parameters deal with the Multi-instance Active Cluster for High Availability (MACH 11) features. They are listed here for reference. See Chapter 4, “Enhancements in high availability” on page 101, for further details. An example of the configuration items is depicted in Example 3-11.

Example 3-11 SDS configuration items

```
SDS_ENABLE        0 # SDS Activation 0 - no, 1 - yes
SDS_TIMEOUT      20 # Time Delay that Primary will wait for any ACK
                    # while perform page flushing before making
                    # SDS Server as down
SDS_TEMPDBS        # Temporary dbspaces used by the SDS Server
                    # <dbspace Name>,<path>,<pagesize>,<offset>,<size>
...
SDS_PAGING         # Paging File Location
                       # <paging File 1 path>,<paging File 2 path>
```

The IFX_FOLDVIEW parameter is designed for performance improvements with joins. The default setting is 0, which disables view folding.

The following types of queries can take advantage of view folding:

- Views that contain a UNION ALL clause and the parent query has a regular join, Informix join, ANSI join, or an ORDER BY clause.

- Views with multiple table joins where the main query contains Informix or ANSI-type outer joins.

A temporary table is created and view folding is not performed for the following types of queries that perform a UNION ALL operation involving a view:

- The view has one of the following clauses: AGGREGATE, GROUP BY, ORDER BY, UNION, DISTINCT, or OUTER JOIN (either Informix or ANSI-type).
- The parent query has a UNION or UNION ALL clause.
- The parent query has multiple views.

In addition, IFX_FOLDVIEW is disabled and a temporary table is created when IFX_FOLDVIEW is set to 1 and a query containing a UNION ALL clause and multiple views is run. As an example:

```
IFX_FOLDVIEW    0
```

The EXPLAIN_STAT configuration parameter enables or disables the inclusion of a Query Statistics section in the explain output file. You can generate the output file by using either the SET EXPLAIN statement of SQL or the **onmode -Y sessionid** command. When enabled, by setting the parameter to 1, the Query Statistics section shows the Query Plan's estimated number of rows and the actual number of returned rows:

```
EXPLAIN_STAT   1        # Enable Query Statistics in EXPLAIN file.
```

The RTO_SERVER_RESTART configuration parameter is explained in the 3.2, "Recovery time objective" on page 75. This parameter can be configured for performance reasons to try to reduce the amount of time that the instance spends in crash recovery. It can also be turned off. This parameter is a performance-related parameter, and tuning can be expected to improve the performance of the instance. As an example:

```
 RTO_SERVER_RESTART 0
```

The RAS_PLOG_SPEED is a recoverability parameter. It is auto-updated and must not be manually changed at this time:

```
RAS_PLOG_SPEED 0
```

The RAS_LLOG_SPEED is a RAS parameter. This parameter is auto-updated and must not be manually changed at this time:

```
RAS_LLOG_SPEED 0
```

The AUTO_CKPTS parameter is described in detail in 3.6.3, “AUTO_CKPTS” on page 90. As an example:

```
AUTO_CKPTS
```

The AUTO_LRU_TUNING parameter is described in 3.6.1, “AUTO_LRU_TUNING” on page 89. As an example:

```
AUTO_LRU_TUNING
```

The AUTO_AIOVPS parameter is described in 3.6.2, “AUTO_AIOVPS” on page 90. As an example:

```
AUTO_AIOVPS
```

The AUTO_REPREPARE parameter is described in 3.6.4, “AUTO_REPREPARE” on page 90. As an example:

```
AUTO_REPREPARE
```

The USELASTCOMMITTED parameter has been introduced to allow the DBA to have users have access to the last committed version of the data. Locks can prevent sessions from reading data until after a concurrent transaction is committed or rolled back. For databases created with transaction logging, you can use the USELASTCOMMITTED configuration parameter in the ONCONFIG file to specify whether the database server uses the last committed version of the data. The last committed version of the data is the version of the data that existed before any updates occurred.

The value set with the USELASTCOMMITTED configuration parameter overrides the isolation level that is specified in the SET ISOLATION TO COMMITTED READ statement of SQL. The values allowed for USELASTCOMMITTED are depicted in Example 3-12.

Example 3-12 Values for USELASTCOMMITTED

```
None = No isolation level identified
Committed Read = All transactions from a Committed Read isolation level
Dirty Read = All transactions from a Dirty Read isolation level
All = Both Committed Read and Dirty Read isolation levels
```

USELASTCOMMITTED specifies the isolation level for which the LAST COMMITTED feature of the COMMITTED READ isolation level is implicitly in effect. The LAST COMMITTED feature can reduce the risk of locking conflicts between concurrent transactions on tables that have exclusive row locks. USELASTCOMMITTED can also enable LAST COMMITTED semantics for READ COMMITTED and READ UNCOMMITTED isolation levels of the SET TRANSACTION statement.

You can dynamically change the values of USELASTCOMMITTED through the SET ISOLATION statement or by using **onmode -wf** or **onmode -wm**. As an example of the feature:

```
USELASTCOMMITTED "COMMITTED READ"
```

A performance-related problem that has been an issue for many years deals with cooked files that are used for chunks. A list of the problems is:

- Cooked file performance can be orders of magnitude slower than raw devices.
- IDS needs to use other I/O options as available.
- Most operating systems have added options for a program to bypass the File System cache, and use Direct I/O (DIO) with the device. This has been done mainly for DBMS systems to run with cooked files with performance close to that of raw devices, thus achieving ease of use without sacrificing performance.

A solution to this set of problems is to use direct I/O (DIO) or concurrent I/O where available. Consider:

- DIO is to be implemented on Linux, AIX, and Solaris™ platforms.
- DIO generally allows the use of kernel asynchronous I/O (kaio), providing a further boost in performance.
- Concurrent I/O (CIO), which is available on AIX, improves performance by bypassing file system inode locking.

Which method do you use for IDS? For regular files used for dbspace chunks, IDS, if available:

- Uses direct I/O (and concurrent I/O) by default.
- Uses kaio with direct I/O by default.
- Does not use direct I/O if the onconfig parameter DIRECT_IO is set to zero.
- Does not use kaio with direct I/O if environment variable `KAIOOFF` is set.
- Does not use direct I/O or set the D_SYNC flag for temporary dbspaces.
- For TEMP dbspaces, use of file system buffering for regular files is probably faster than using direct I/O, because there is no need to synchronize writes to ensure writes are committed to disk, unlike non-TEMP dbspaces. Therefore for regular files used for temporary dbspaces, we do not use direct I/O, even if direct I/O is enabled, nor do we synchronize writes to the file.

For a white paper discussion of DIO and CIO, refer to:

`http://www.ibm.com/servers/aix/whitepapers/db_perf_aix.pdf`

The DIRECT_IO configuration parameter is used to control the use of direct I/O for regular files used for dbspace chunks. As an example:

```
DIRECT_IO 0
```

Another parameter, LOG_INDEX_BUILDS, is used to enable or disable index page logging. If the parameter is enabled, logical log file space consumption increases, depending on the size of the indexes. This might mean logical log backups need to occur more frequently. Messages are written to the online.log when index page logging status changes. The value of `0` disables the parameter. The value of `1` enables the parameter.

```
LOG_INDEX_BUILDS 0
```

3.8.2 Environment variables

In this section, we discuss new environment variables that are pertinent to IDS 11.

The **`onstat - g env`** command lists the active environment settings.

Tip: You can include the environment variable $INFORMIXDIR in the ONCONFIG file. It must be the first path name value in the path name specifications.

When TAPEDEV and LTAPEDEV specify directories, use the IFX_ONTAPE_FILE_PREFIX environment variable to specify a prefix for backup file names that replaces the hostname_servernum format. If no value is set, file names are hostname_servernum_L*n* for levels and hostname_servernum_Log*nnnnnnnnn* for log files.

If you set the value of IFX_ONTAPE_FILE_PREFIX to My_Backup, the backup file names have the following names:

- My_Backup_L0
- My_Backup_L1
- My_Backup_L2
- My_Backup_Log0000000001
- My_Backup_Log0000000002

Set the value using the following command where string is the prefix to use for the names of backup files:

```
setenv IFX_ONTAPE_FILE_PREFIX string
```

The IFX_NO_LRU_TUNING environment variable lets you configure the network buffers to the optimum size. It specifies the size of all network buffers in the free (unused) global pool and the private network bufferpool for each database server session. Set it using the following command where size is the integer size (in bytes) for one network buffer:

```
setenv IFX_NETBUF_SIZE size
```

The default size is 4 kilobytes (4,096 bytes). The maximum size is 64 kilobytes (65,536 bytes) and the minimum size is 512 bytes. You can specify the value in hexadecimal or decimal form.

Note: You cannot set a different size for each session.

3.8.3 Deprecated parameters

In this section, we list any environment variables and configuration parameters that have been deprecated.

Onconfig parameters

Example 3-13 shows onconfig parameters that are deprecated in IDS 11 and must no longer be used.

Example 3-13 List of deprecated onconfig parameters

```
AFF_NPROCS
AFF_SPROC
BUFFERS
FAST_RESTART_CKPT_FUZZYLOG
FAST_RESTART_PHYSLOG
LBU_PRESERVE
LOGSMAX
LRU_MAX_DIRTY
LRU_MIN_DIRTY
LRUS
NOAGE
NUMAIOVPS
NUMCPUVPS
```

Instead of BUFFERS, LRUS, LRU_MIN_DIRTY, and LRU_MAX_DIRTY in the configuration file, use the BUFFERPOOL parameter instead.

You need to use the VPCLASS parameter instead of the NUMCPUVPS, NUMAIOVPS, NOAGE, AFF_SPROC, and AFF_NPROCS parameters.

The FAST_RESTART_CKPT_FUZZYLOG parameter and the FAST_RESTART_PHYSLOG parameter dealt with fuzzy checkpoints during the roll-forward (log replay) phase of recovery. Because fuzzy checkpoints are being deprecated, these parameters are being deprecated, too.

Information that was specified with the LRU_MAX_DIRTY and LRU_MIN_DIRTY configuration parameters is now specified using the BUFFERPOOL configuration parameter.

IDS no longer supports the LBU_PRESERVE parameter, which reserves the last logical log for ON–Archive use. ON–Archive, which has been discontinued, was the only utility that required free log space to back up a logical log.

IDS also no longer supports the LOGSMAX parameter.

Environment Variable

There is only one environment variable in IDS 11 that is being deprecated, and the name of the variable is DEFAULT_ATTACH.

The DEFAULT_ATTACH environment variable supports the traditional behavior of Version 7.*x* of IDS, which required that only non-fragmented B-tree indexes on non-fragmented tables can be attached.

4

Enhancements in high availability

Traditionally, IDS has provided multiple robust options for supporting high availability data replication solutions. Unquestionably, data replication is a highly reliable and flexible option for high availability solutions. Previous releases of IDS have supported two replication technologies: Enterprise Replication (ER) and High Availability Data Replication (HDR). Together, these technologies have been meeting client solution requirements. Both replication technologies can be integrated with each other and coexist with other availability solutions, such as disk mirroring.

While HDR and ER have been features of IDS for many years and have proven to be highly reliable and low-maintenance technologies, IDS 11 adds support for two new types of secondary servers:

- *Shared Disk Secondary (SDS) servers* provide increased availability by allowing one or more instances of the IDS server to attach to the same disk subsystem.
- *Remote Standalone Secondary (RSS) servers* provide multiple geographically remote backup servers.

Both SDS and RSS servers provide clients a way to obtain increased capacity by distributing workload across multiple servers.

Although each of the IDS 11 solutions works well on its own, all of the types of secondary servers can be combined together, and with ER as well, to provide customized availability solutions. IDS 11 configurations are simple to set up and maintain and are highly scalable.

4.1 A review of existing IDS availability solutions

IDS has always provided robust data replication features, facilitating an extremely high degree of availability both during online operation and in case of a failure. Historically, availability solutions for IDS clients have included:

- Backup and restore
- Disk mirroring
- High Availability Data Replication (HDR)
- Enterprise Replication (ER)

4.1.1 Backup and restore

IDS was one of the first database servers to introduce online backup. With backup and restore, the client periodically performs an online backup of the system onto an external device, such as tape media. The media is then taken off-site to a secure location. In the event of a catastrophic system loss, the external backup is used to restore the system to the point at which the backup was taken.

The logical logs, which contain a list of changes made to the database, are also backed up. By applying the logical log backup to the system being restored, the database can be recovered up to the point at which the last backup of the logical logs was made.

There are many improvements made in backup and restore capabilities in IDS 11 as well. For additional information, refer to 6.1, “Backup and recovery changes” on page 168.

4.1.2 Disk mirroring

Disk mirroring is commonly performed using software or hardware to mirror the database *chunks* (chunks are physical storage locations of data in IDS) on another storage device for backup and restore purposes.

IDS provides built-in support for dual or mirrored chunks. When using mirrored chunks, the chunks are stored in two separate files: the primary and its mirror. Writes occur to both the primary and to the mirror chunk. Then, if the primary chunk becomes unavailable while the server is active, the server will automatically switch to using the mirror chunk.

In recent years, clients have increasingly chosen to use hardware disk mirroring rather than mirrored chunks. With hardware disk mirroring, the entire disk complex is mirrored, usually through a form of RAID or other disk volume

manager. The advent of Storage Area Network (SAN) and Network Attached Storage (NAS) solutions has made it possible to separate the physical storage media from the systems that use that storage. In effect, instead of the disk media being attached to the server, the server is attached to the disk. This means that it is possible to have a separate idle standby system with IDS installed that can be used to provide availability in the event of the loss of the primary server. Because it is possible to locate the mirrored disk a certain distance away from the primary, a high degree of availability is possible.

4.1.3 High Availability Data Replication

As early as IDS Version 7, Informix adopted HDR technology, which is fully integrated within the data server. HDR is extremely easy to set up and administer and does not require any additional hardware or software for automatically handling server or disk failures.

HDR maintains two identical IDS server instances on servers with similar configurations and operating systems, as depicted in Figure 4-1. HDR employs a log record shipping technique to transfer the logical log records from the primary server to the secondary server. The secondary server is in perpetual roll-forward mode so that data on the secondary server remains current with data on the primary server. The secondary server supports read access to data, allowing database administrators to spread workload among multiple servers.

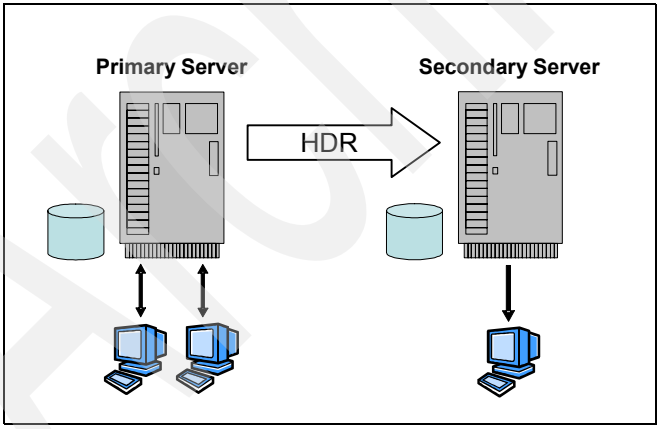


Figure 4-1 High Availability Data Replication

The secondary server can be configured to operate in SYNC (synchronous) or ASYNC (asynchronous) mode. In SYNC mode, HDR guarantees that when a transaction is committed on the primary server, its logs have been transmitted to the HDR secondary server. In ASYNC mode, transaction commitment on the

primary and transmission of updates to the secondary are independent. This mode can provide better performance, but brings with it the risk of possibly losing transactions.

HDR provides automatic failover (using the DRAUTO configuration parameter) and automatic client redirection. With DRAUTO set, if the primary server fails, the HDR secondary server automatically takes over and switches to a standard or primary server (based on the DRAUTO value). When the original primary server becomes available, it is synchronized when HDR is restarted.

Current HDR replication technology also supports the automatic client redirection feature, which makes failover transparent to the application. To activate automatic client redirection, the primary and secondary servers must be defined as a group in the SQLHOSTS file. Clients use the group name to connect to the IDS server. The network layer and the client server protocol ensure that the client is always connected to the primary server in the group. If the primary server fails and the secondary server becomes the new primary server, clients connected to the group will be automatically connected to the new primary server.

Refer to the IBM Redbooks publication*Informix Dynamic Server V10: Superior Data Replication for Availability and Distribution*, SG24-731919, which contains additional detail about existing HDR features in IDS 10. A softcopy of that book is available for download at:

`http://www.redbooks.ibm.com/abstracts/sg247319.html?Open`

4.1.4 Enterprise Replication

Enterprise Replication (ER) provides replication of data across multiple independent IDS servers and has the ability to support both “active-passive” and “active-active” replication. That is, you can write to any of the servers participating in ER. Conflicts between servers are resolved in ER by reconciling transactions. ER can also be used to replicate individual tables or subsets of tables rather than the entire database (as is the case with HDR). ER is designed to support multiple servers with complex topologies, such as that depicted in Figure 4-2 on page 106.

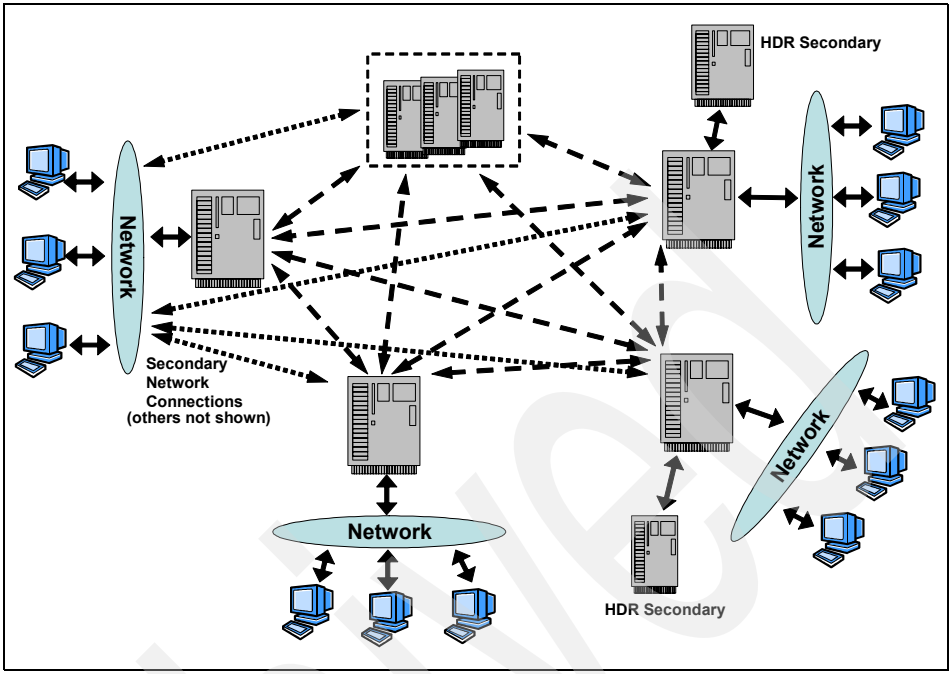


Figure 4-2 Enterprise Replication

4.2 New high availability solutions in IDS 11

With IDS 11, new replication features provide two additional types of continuously available server configurations that can be used in conjunction with existing HDR configurations: Remote Standalone Secondary (RSS) servers and Shared Disk Secondary (SDS) servers. Additionally, a new Continuous Log Restore (CLR) feature makes it possible to manually maintain a backup system.

4.2.1 Remote Standalone Secondary server

Similar to HDR, RSS servers can provide geographically remote, application-accessible copies of the database. Logical logs are continuously transmitted from the primary server and applied to the database on the RSS server, as depicted in Figure 4-3 on page 107.

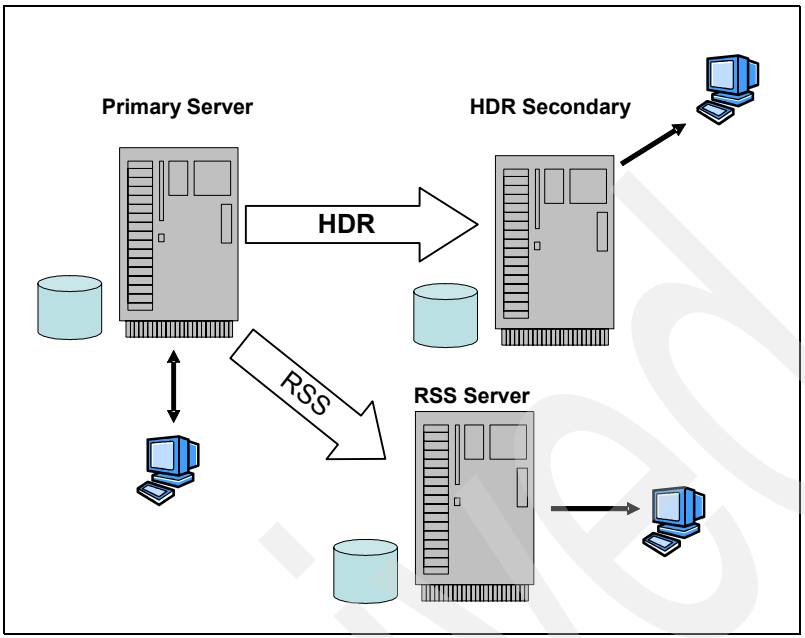


Figure 4-3 Remote Standalone Secondary server

The RSS servers use a fully duplexed communication protocol, allowing the primary server to send data to the RSS servers without waiting for an acknowledgement that the data was received. Using fully duplexed communication means that RSS servers have extremely little impact on the primary server performance.

An RSS server can be switched to become an HDR secondary server if the HDR secondary server becomes unavailable, or if an existing secondary server is promoted to a primary server. An RSS server can become a standard server if the primary server and the current HDR secondary server become unavailable.

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

As depicted in Figure 4-4 on page 108, remote application servers can access local database servers to minimize latency and improve performance.

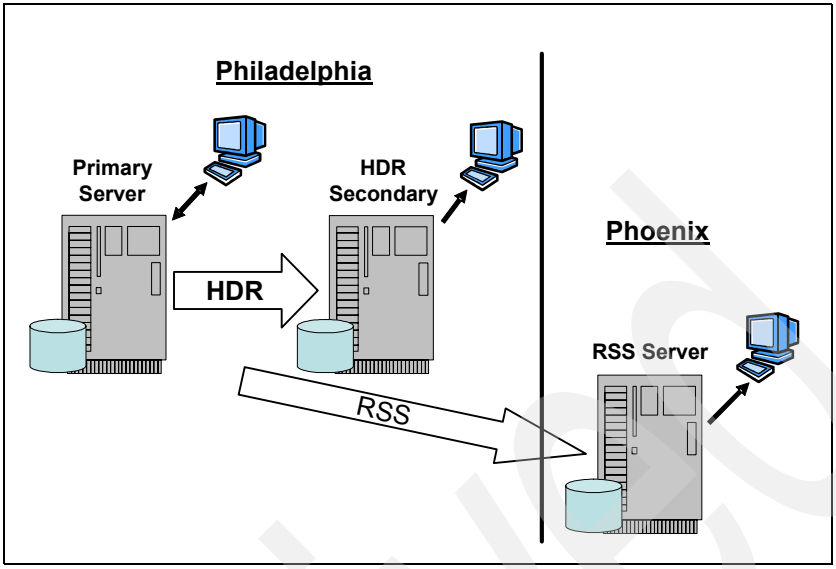


Figure 4-4 Physically remote standalone secondary server

4.2.2 Shared Disk Secondary servers

SDS servers access the same physical disk as the primary server. They provide increased availability and scalability without the need to maintain multiple copies of the database as depicted in Figure 4-5 on page 109.

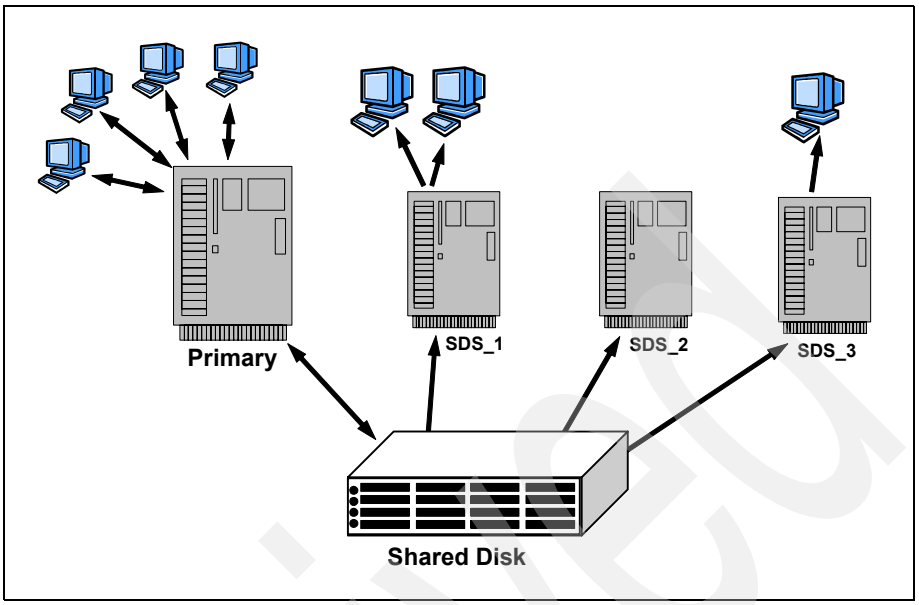


Figure 4-5 Shared Disk Secondary Servers

An SDS server can be made available very quickly. After its configuration, an SDS server joins an existing system and is ready for immediate use. Because SDS servers also use fully duplexed communications with the primary server, having multiple SDS servers has little affect on the performance of the primary server. SDS servers are completely compatible with both hardware-based and software-based disk mirroring.

If the primary server becomes unavailable, failover to an SDS server is easily accomplished. The specified SDS server becomes the new primary server and all other SDS servers automatically connect to the new primary server.

Multiple SDS servers also provide the opportunity to offload reporting, backup, and other functionality from the primary server.

4.2.3 Continuous Log Restore

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

CLR is a robust way to set up a hot backup of a database server. The hot backup of the primary IDS server is maintained on the backup server, which contains similar hardware and an identical version of IDS. To configure a backup server using CLR, a physical backup of the primary server is created and the backup copy is transported to the backup server. The backup is then restored on the backup server. After the restore is complete, the backup server is ready for a logical recovery. In the event that a logical log on the primary server becomes full, it is backed up and then transported to the backup server where logical recovery (log roll-forward) is performed. The CLR operation is depicted in Figure 4-6.

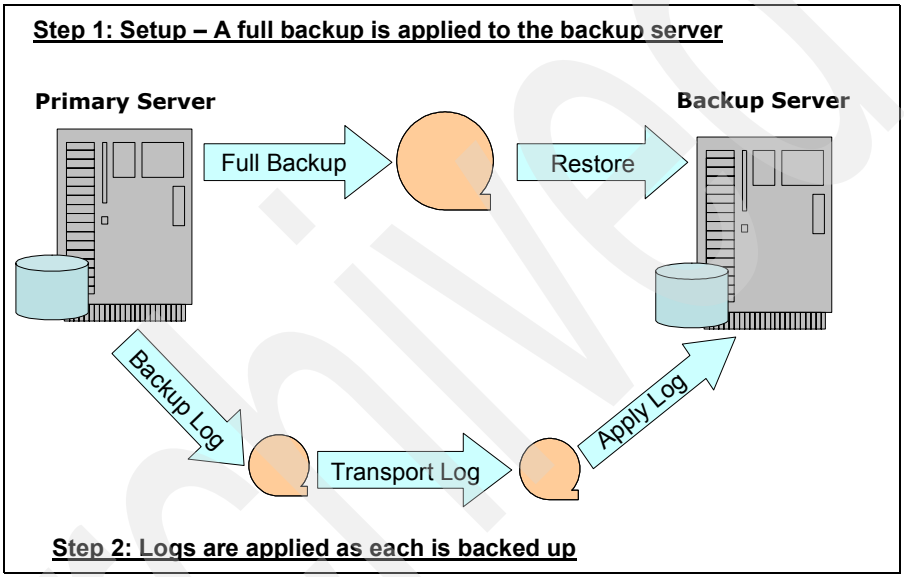


Figure 4-6 Continuous Log Restore

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

CLR can be combined easily with other high availability solutions, such as IDS 11, or with hardware solutions, such as cluster failover.

4.3 Failover of primary server and recoverable groups

In previous versions of IDS, the only failover option available was that of failover of the primary server to an HDR secondary server. IDS 11 extends these options with the addition of two new failover server configuration options that are useful not only in failure or disaster situations, but also for performing maintenance

tasks on the database server. Think of the IDS 11 system as a group of servers that act as a recoverable unit. This recoverable unit consists of a single primary server and one or more secondary servers.

IDS 11 consists of a single primary server and one or more secondary servers. The secondary server can include any combination of SD, RS, and HDR secondary servers.

4.3.1 Recoverable Groups

In this section, we discuss Recoverable Groups based on the HDR and Remote Standalone secondary servers. Until now, the HDR secondary server has been the traditional means of providing high availability with IDS. Over the years, we have been aware that clients want additional functionality. Several recommendations that have been mentioned are:

- Have additional HDR secondary servers.
- Impact the primary as little as possible with the workload on the secondaries.

Remote Standalone Secondary servers (RSS servers), as shown in Figure 4-7 on page 112, directly address these concerns. In either case, additional secondary servers can provide the extra workload relief that clients want. In addition, because the RSS server operates in a fully duplexed communications protocol, the primary server does not have to wait for the RSS server to acknowledge each packet. This capability allows the RSS server to be able to support a much higher rate of data transfer with the primary server even when there is a large latency on the network between the two servers. It also greatly improves the performance of the primary server, because it does not have to wait for an acknowledgement (ACK) for each packet. The downside of this approach is that an RSS server cannot be directly updated to be a primary server. It must first assume the role of the HDR secondary server before it can become a primary server.

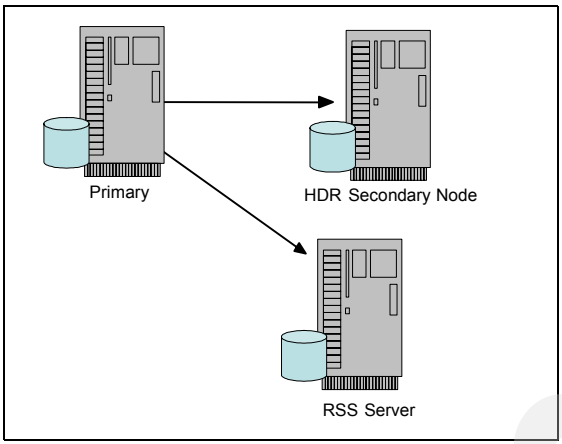


Figure 4-7 HDR secondary server with RSS in normal operation

If at any time the primary fails, the HDR secondary server takes over control by switching to read/write mode. The HDR secondary server is made the primary server and starts sending logs to the RSS server, as depicted in Figure 4-8.

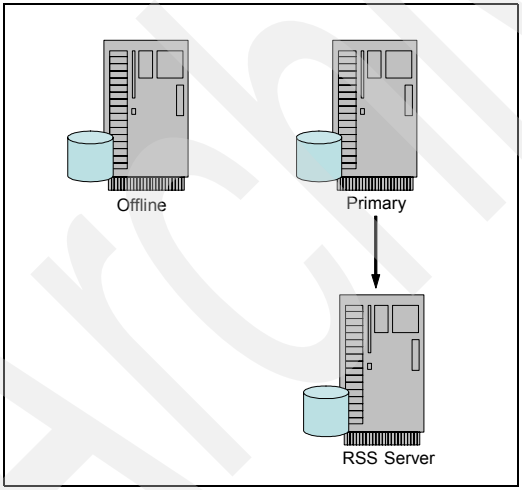


Figure 4-8 Upgrading an HDR secondary server to primary server

If for any reason, the HDR secondary server fails and it appears that the HDR secondary server will be offline for an extended period of time, the RSS server can be converted into the HDR secondary server, as depicted in Figure 4-9 on page 113.

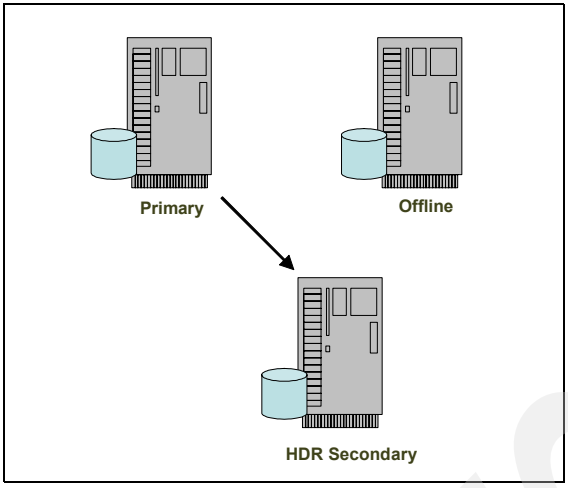


Figure 4-9 Converting RSS server into HDR secondary server

Then, when the original HDR secondary server comes back online, it can be converted to an RSS server and reenter the replication domain, as shown in Figure 4-10.

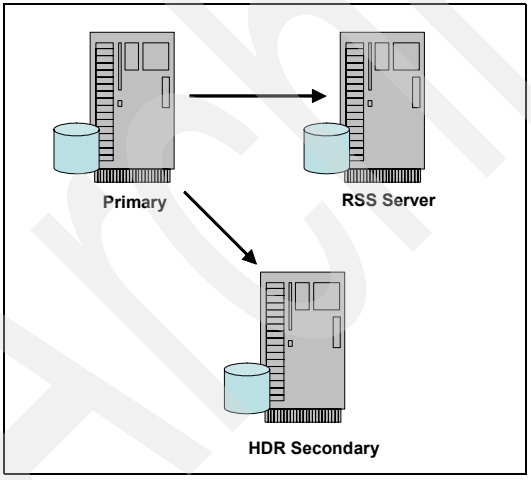


Figure 4-10 Original HDR secondary server becomes the RSS server

4.3.2 Recoverable Groups based on the SDS server

Figure 4-11 on page 114 shows an example of a primary server configured with two SDS servers. In this case, the primary server role can be transferred to either of the two SDS servers. This includes the case where the primary needs to be

taken out of service for a planned outage or because of a failure of the primary server.

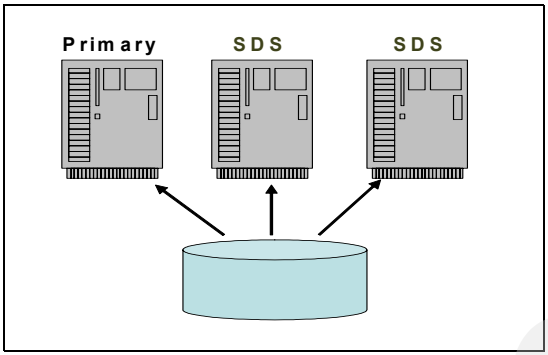


Figure 4-11 Primary server with two SDS servers

Because both of the SDS servers are reading the same disk subsystem, it makes no difference which of the two SDS servers becomes the primary server; they are equally able to assume the role of the primary server if they are similarly sized servers. This is illustrated in Figure 4-12.

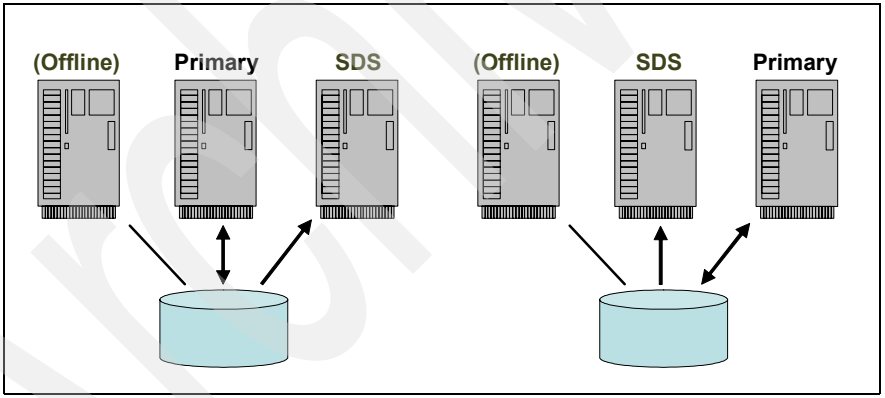


Figure 4-12 Failover of primary server in SDS cluster

There are also a number of recovery options to handle the loss of the shared disk itself. The most common recovery options are to have the shared disk use either RAID technology (such as RAID 5) or to use disks based on SAN technology, which might include a form of remote disk mirroring as depicted in Figure 4-13 on page 115. Because the disk and its mirror can be located in different areas, this provides a high degree of availability for both planned and unplanned outages of either the servers or of the disk subsystems.

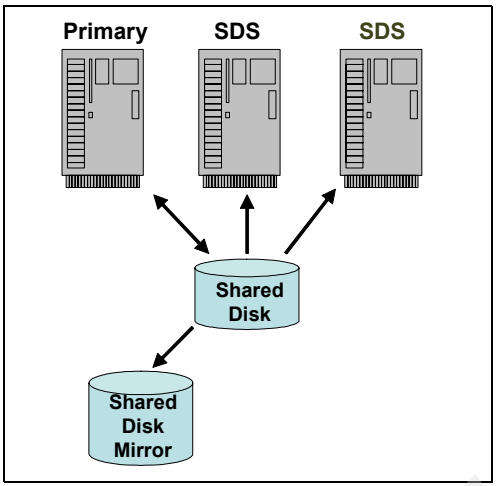


Figure 4-13 Normal operation with disk mirroring

If the primary disk fails, the primary and all SDS clones can be redirected to the mirrored disk, as shown in Figure 4-14.

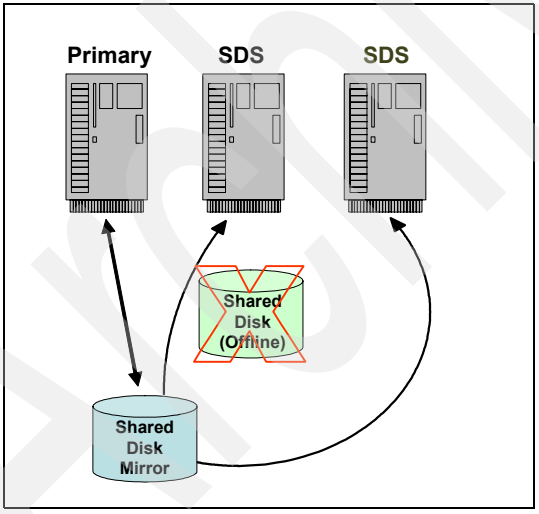


Figure 4-14 SDS with primary shared disk offline

In addition to configuring a mirrored disk subsystem, you might want to have extra redundancy with additional servers. For example, you might want to have the primary and two SDS servers shown in Figure 4-14 contained within a single blade server enclosure. The configuration, depicted in Figure 4-15 on page 116,

shows that it is an attractive solution when you need to periodically increase the processor read capability, such as when performing large reporting tasks.

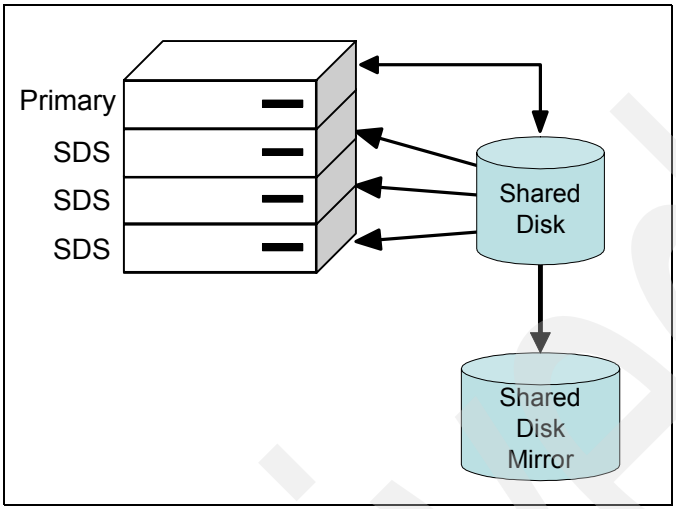


Figure 4-15 Single blade server housing the primary server and three SDS servers

In this type of configuration, you might decide to avoid the possible failure of a single blade server by using multiple blade servers, as depicted in Figure 4-16.

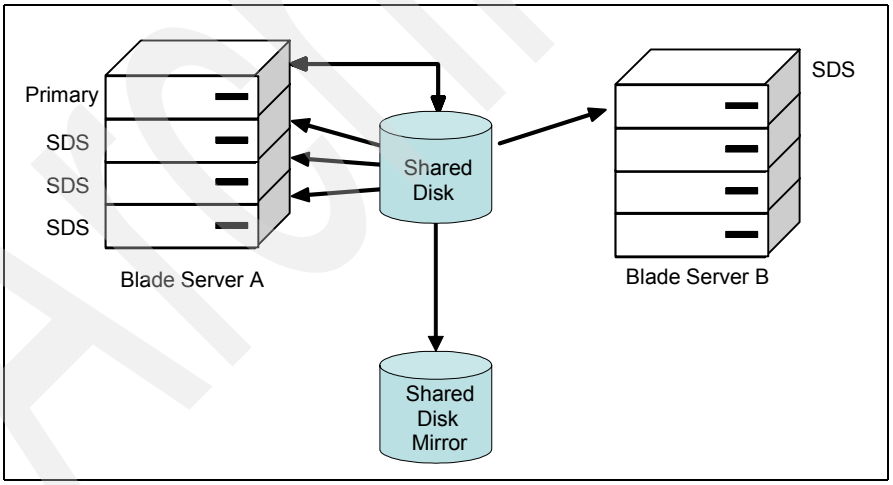


Figure 4-16 Two blade servers used to house SDS servers

However, if `Blade Server A` fails, it is possible to transfer the primary server role to the SDS server on `Blade Server B`. Because it is possible to bring additional

SDS servers online quickly, you can dynamically add SDS servers to `Blade Server B`, as shown in Figure 4-17.

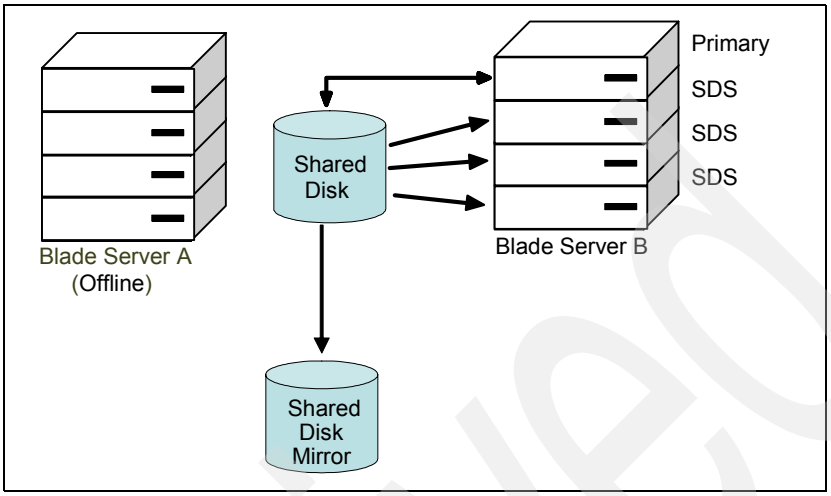


Figure 4-17 Transferring the primary server role

4.3.3 Recoverable Groups with multi-layer HA solutions on IDS 11

Because of limits on the distance that disk mirroring can support, you might need an alternative to using shared disks and shared disk mirroring to provide availability. For example, if you prefer that there is a significant distance between the two copies of the disk subsystem, you might choose to use either an HDR secondary server or an RSS server to maintain the secondary copy of the disk subsystem. Figure 4-18 shows an example of an HDR secondary server in a blade server configuration.

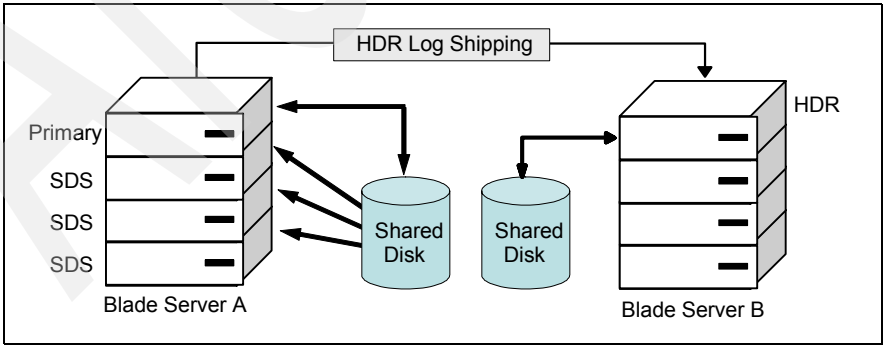


Figure 4-18 Using an HDR secondary server in conjunction with SDS servers

In this configuration, if the primary server fails, but the shared disks are intact and the blade server is still functional, it is possible to transfer the primary server role from `Blade Server A` to another server in the same blade server as depicted in Figure 4-19. Changing the primary server causes the source of the remote HDR secondary server to automatically reroute to the new primary server, as illustrated in Figure 4-19.

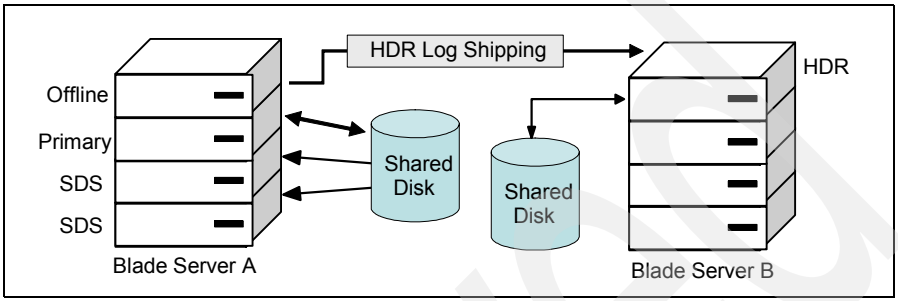


Figure 4-19 Transfer of the primary server to an SDS server

Suppose, however, that the failure described was not simply a blade within the blade server, but the entire blade server or the entire site. In this case, you might have to fail over to the HDR secondary server, as depicted in Figure 4-20.

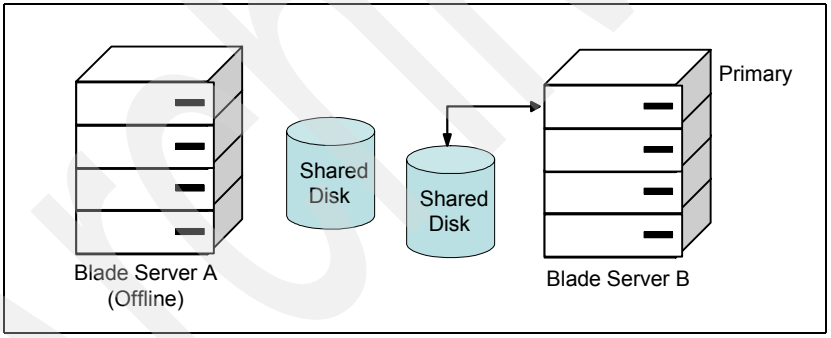


Figure 4-20 Transferring the primary server to the HDR secondary server

Because only one instance of IDS was running on `Blade Server B` as the HDR secondary server, only one instance of the IDS (primary server) was started on `Blade Server B`. But you can easily add SDS servers. After the primary server has been transferred to `Blade Server B`, it becomes possible to start SDS servers on `Blade Server B` as well, as shown in the Figure 4-21 on page 119.

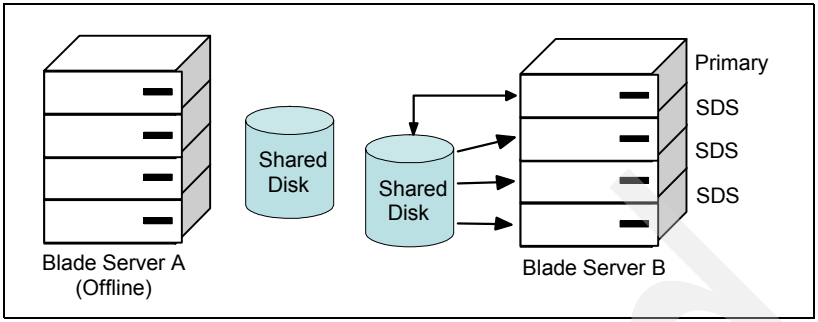


Figure 4-21 Adding SDS servers in Blade Server B

4.4 Combining IDS availability options

IDS provides many fundamental technologies for high availability. You can combine these technologies to suit a variety of business situations. Here are several examples of combinations that you can use to address different scenarios.

Table 4-1 Availability requirements and recommended solution

Application requirements	Recommended solution
You periodically want to increase reporting capacity.	Use SDS or RSS servers. If the amount of data is very large and making multiple copies is difficult, SDS servers are more useful. Alternatively, RSS servers can reduce the total cost of ownership.
You use SAN devices, which provide ample disk hardware availability, but you are concerned about server failures.	Use SDS servers.
You use SAN devices, which provide ample disk hardware mirroring, but you also want a second set of servers that you can bring online if the primary operation fails (and the limitations of mirrored disks are not a problem).	Consider using two BladeCenters running SDS servers at the two sites.
You want to have a backup site a moderate distance away, but you cannot tolerate any loss of data during failover.	Consider using two BladeCenters with SDS servers on the primary BladeCenter® and an HDR secondary in SYNC mode on the remote server.
You want to have a highly available system in which no transaction is ever lost, but you must also have a remote system on the other side of the world.	Consider using an HDR secondary server located nearby running SYNC mode and an RSS server on the other side of the world.
You want to have a high availability solution, but because of the networks in your region, there is a large latency.	Consider using an RSS server.
You want a backup site, but you do not have any direct communication with the backup site.	Consider using Continuous Log Restore with backup and recovery.
You can tolerate a delay in the delivery of data as long as the data arrives eventually; however, you need to have quick failover in any case.	Consider using SDS servers with hardware disk mirroring in conjunction with ER.
You need additional write processing power, can tolerate delay in the delivery of those writes, need something highly available, and can partition the workload.	Consider using ER with SDS servers.

4.5 Conclusion

IDS provides many new innovative features to support high availability of data. The replication technology in earlier versions of IDS is widely used by clients and meets their requirements. With the release of IDS 11, the replication technology is enhanced further with new features, such as RSS, SDS, and CLR. What makes it more interesting is that all the technologies can coexist and they work together. It makes IDS a very powerful server to meet any challenging database requirements. Permutations of HDR, ER, Storage Solution, SDS, RSS, and CLR can provide many combinations to satisfy an extremely wide range of solutions for database requirements. What makes the IDS solution unique is that all the features are built within the server and they interact with each other. This modular approach makes each feature simple to implement and use. The interaction between the features enables you to solve a complex problem, and that makes IDS a very powerful solution enabler. At the same time, the simplicity of IDS is maintained, because any issues that might arise due to the combination of technologies used can be resolved inside the server and thus can minimize DBA activity and the overall cost of ownership.

5

Data privacy

The need to protect the privacy of information from access is of critical importance in the current computing world. Failure to protect data access from accidental, malicious, or unauthorized use can result in a public relations crisis, loss of client loyalty, breach of contract, or even a violation of the law.

Protecting the privacy of an individual's data involves validating identity, controlling access to the data, and maintaining a secure computing infrastructure. The features involved in implementing that scope of work are:

- Authentication:
  - Connection
  - Pluggable Authentication Module (PAM)
  - Lightweight Directory Access Protocol (LDAP)
- Authorization:
  - Role-Based Access Control (RBAC)
  - Label-Based Access Control (LBAC)
- Encryption

5.1 Authentication approaches

In these sections, we discuss how the database management system (DBMS) authenticates the user identity as a connection is established or rejected. The options and their attributes are listed in Table 5-1. You can choose any of these methods for each DBSERVERNAME or DBSERVERALIAS, but you can choose only one option for each. If you need users to connect using multiple methods, you must provide multiple DBSERVERALIASes, one per authentication method.

Table 5-1 Authentication choices

Method	Attributes
OS user ID	No encryption, uses OS password lookup
Password encryption	Same as for OS user ID, but with the password encrypted during transmission
Pluggable Authentication Module (PAM)	User-provided authentication methods
LDAP	User-provided access to the LDAP directory

5.1.1 Connection authentication choices

Before checking a user ID and password combination, certain lists of trusted users or trusted systems are checked by the DBMS to decide whether to accept a connection. Which of these lists are checked is determined by settings in field five of each SQLHOSTS entry. Field five might 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 5.1.2, "Using Pluggable Authentication Modules (PAMs)" on page 130.

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 a another connection.

Complete trust

For `s=1` or `s=3`, the DBMS checks to see if the user, host, or both are listed in the /etc/host.equiv file (Linux or UNIX) or has a trust relationship (Windows). If so,

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 must be used only rarely, because it is so trusting.

Complete trust 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, complete trust is required for instances participating in Enterprise Replication. No user ID or password is used for the connections between the instances passing the replication data.

Other uses of the complete trust technique need to be rare, because a system or user in this list is trusted always.

This technique is demonstrated in Example 5-1. The database name and server name are provided without any username or password, and the connection is established. Compare this to example Example 5-2 on page 126 where the servername is different. In that case, the connection is rejected, because the client is not trusted by the server.

Example 5-1 Connection with s=3

```
informix@nile:~> dbaccess stores9@ban_er1 -

Database selected.
>
```

Partial trust

For `s=2` or `s=3`, a different list of trusted users and systems is consulted. Each user's home directory can contain 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 set up the configuration 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.

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 5-2 demonstrates the error received when no password is supplied and a correct connection if a password is provided.

Example 5-2 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.
>
```

Impersonating a user

For the value r=1, a client program might 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 executes. It is ignored in the sqlhosts file on the machine where the DBMS executes.

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 inaccessible by anyone except its intended users, the credentials are at risk. Because those credentials are valid, if they are used from another system, they are accepted. Do not use this technique unless you have no other choice.

The value `r=0` disables the .netrc lookup and is recommended to prevent unwanted impersonations.

Example 5-3 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 `dick` and password `96eagles` are used rather than the current user ID.

Example 5-3 Sample .netrc file

```
informix@nile:~> cat .netrc
machine kodiak
        login dick
        password 96eagles
```

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 that the user ID or password (or both) are not in the OS set of user IDs and passwords, 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.

For example, we created a user ID and password for a new user using SuSe Linux. The entire process is shown in Example 5-4 and is simple.

The user root must create the accounts and passwords.

Example 5-4 Setting up and using Linux user IDs and passwords

```
useradd -d /home/tom -p 96eagles tom
useradd -d /home/dick -p 72mercury dick
```

Example 5-5 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 insufficient to connect to a database. The user must also have at least connect permission to a database. DBMS connections are 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 username or password is used.

The first connection uses a user ID that does not exist. The other connections use valid user IDs.

It is unnecessary for all users to have permission to use all databases. Each user needs to only have permission to use the databases required by their work. This is where roles are most useful. Roles are discussed further in 5.2.1, “Role-Based Access Control” on page 132.

Example 5-5 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.
```

5.1.2 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-party vendors, or you can write your own module. General information about PAMs is available at:

http://inetsd01.boulder.ibm.com/pseries/en_US/aixbman/security/pam_overview.htm

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

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 reside in $INFORMIXDIR/etc. Both the libraries and the configuration files are referenced in the concsm.cfg file.

The use of a PAM is specified in the sqlhosts file entry for the DBSERVERNAME or DBSERVERALIAS that is 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/concsm.cfg file. This is exactly the same mechanism that is used for the password encryption module that is described in 5.3.7, “Encrypting passwords during transmission” on page 157.

Example 5-6 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 is going to issue a challenge, the entry reads "pamauth=(challenge)" instead.

Example 5-6 SQLHOSTS entry using a PAM

```
kod_auth3 olsoctcp kodiak kod_auth3 s=4, pam_serv=(authpam), pamauth=(password)
```

Using password authentication

Using a password authentication PAM is similar in concept to the basic IDS authentication using OS accounts. The difference is that another method is used to check whether 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 can be a third-party method, such as Kerberos, or you might choose to write your own method.

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, the connection is granted. The idea is similar to the technique that is used in military operations. A sentry calls out a challenge question. If the person wanting to gain access knows the right answer, the person is allowed to enter.

Application considerations for challenge-response

If you choose a challenge-response authentication method, each application must be programmed to do certain things 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 that you have chosen.

If applications cannot be altered to meet the requirements of the PAM, you must provide and use a different DBSERVERNAME or DBSERVERALIAS.

5.1.3 Using an LDAP directory

The LDAP support provided in IDS requires 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 for how to complete, compile, and configure the LDAP support module.

More general information about LDAP is at:

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 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 by the LDAP server provider in addition to the instructions in *The IDS Administrators Guide*, G251-2267-02.

An almost fully functional sample LDAP module is provided in the $INFORMIXDR/demo/authentication directory. If you use Microsoft Windows, 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.

5.2 Authorization

After the connection to the relational database management system (RDBMS) is authenticated, the database server controls the level of access privileges during SQL operations. Privilege-based authorizations manage access control using the specific privileges and permissions of a user.

IDS V10 implemented Role-Based Access Control (RBAC) as a solution to restricting system access to authorized users. To address requirements where multiple levels of security are required, such as at the Department of Defense (DoD), Label-Based Access Control (LBAC) has been added to IDS 11.

5.2.1 Role-Based Access Control

Roles are just sets of privileges. Privileges may be granted to a role rather than to a user. After that is done, the role can 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 can be granted to one user. The user can then choose which role (that is, which set of privileges) are in effect by using the SET ROLE statement.

Example 5-7 shows the technique. These are the steps that 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.

Default roles

Each user can have an assigned default role. If so, that role is the set of privileges that the user has when connecting to the DBMS. A SET ROLE statement is required if different privileges are needed and is demonstrated in Example 5-7. 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 of certain applications. If the user default role is extremely restricted, an application can use the SET ROLE statement to have its required privileges when using the application. Outside the application, the user is restricted. Restricting what users can do outside of certain applications helps control who can make unplanned changes to a database or query for restricted data.

Example 5-7 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
>
```

5.2.2 Label-Based Access Control

Material in this section was taken in large part from an IBM White Paper titled *Label-Based Access Control*, by Dave Desautels and Lynette Adayilamuriyil.

Label-Based Access Control (LBAC) is a form of Mandatory Access Control that allows control over who can access data in individual rows and columns of a

table. In its simplest form, a label is assigned to the data and a label is assigned to a user, and there is a relationship between the labels. If a user's label dominates the data's label, the user is allowed access to the data.

However, it is important to be clear that LBAC is not the only access control within IDS. IDS also provides the SQL standard select, update, insert, and delete (SUID) permissions on tables, as well as control of other database objects, such as indexes and user-defined routines. LBAC does not supersede those permissions; it is an additional and optional means of access control. It differs from the other methods in offering access control at the lower level of rows and columns as opposed to tables.

The starting point for LBAC is to define a security policy. A *security policy* is made up of security components and describes the criteria that are used to control access to sensitive data. There are three distinct types of security components: array, set, and tree. Each component type has a predefined rule set that helps determine label dominance. To protect data, tables are associated with a security policy. The data might be protected by row, by column, or both. Security labels are created for the security policy and attached to individual rows and columns in the table. Users are granted security labels in order to access the protected data. Users can read and write only those rows or columns where their label dominates the row or column label.

A new database security administrator (DBSECADM) role has been introduced to perform all security-related administration, including setup of the LBAC security infrastructure.

Security policies, components, and labels

A security policy is composed of one or more security components. There are three types of security components: arrays, sets, and trees. A security component is made up of elements. A security label is a list of elements (possibly empty) for each component in the policy.

The array security component type

An array security component is an ordered list of elements, for example, `Top Secret`, `Secret`, `Confidential`, and `Public`. In this example, `Top Secret` is the highest level and dominates all the other elements. `Secret` dominates `Confidential` and `Public`, but not `Top Secret`. There can be a maximum of 64 unique elements in the array.

For an array-type component, a user is allowed to read data at or below the user's level and write data only at the user's level. A user, whose label value for this component is `Public`, can only see data that is also labeled `Public`. While a user, whose label value is `Top Secret`, can see all the data in the table. A user,

whose label value is `Top Secret`, can write `Top Secret` data, but cannot write `Secret` data.

You can create an array security component using this command:

```
CREATE SECURITY LABEL COMPONENT level ARRAY ['Top Secret', 'Secret',
'Confidential', 'Public'];
```

The set security component type

A *set security component* is an unordered list of elements, for example, `Marketing`, `Product Development`, and `Quality Assurance`. There can be a maximum of 64 unique elements in the set.

For a set-type component, a user is allowed to read or write data as long as the user's label contains all the elements in the data label. A user, whose label contained `Marketing`, can only see data labeled `Marketing`. A user, whose label contained `Product Development` and `Quality Assurance`, can access data labeled `Product Development`, `Quality Assurance`, or both. If the data is labeled with both `Product Development` and `Quality Assurance`, only a user with both of these elements in their label can access the data.

You can create a set security component this way:

```
CREATE SECURITY LABEL COMPONENT department SET {'Marketing',
'Product Development', 'Quality Assurance'};
```

The tree security component type

A tree security component is a hierarchical set of elements. There can be a maximum of 64 unique elements in the tree. An example is the organizational chart shown in Figure 5-1.

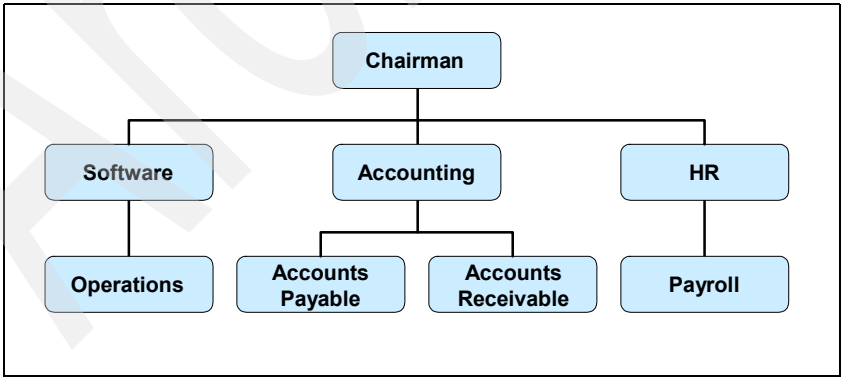


Figure 5-1 Organizational chart

For a tree-type component, a user is allowed to read or write data if the user's label contains any of the elements in the data label or the ancestor of one of these elements. A user, whose label contains `Chairman`, can access all the data. A user, whose label contains `Accounting`, can access data labeled `Accounting`, `Accounts Payable`, and `Accounts Receivable`. A user, whose label contains `Accounts Payable`, can only access data labeled `Accounts Payable`. A user, whose label contains `Operations` and `Payroll`, can access data labeled either `Operations` or `Payroll`.

You can create a tree security component this way:

```
CREATE SECURITY LABEL COMPONENT organization TREE ('Chairman' Root,
'Software' under 'Chairman', 'Accounting' under 'Chairman', 'HR'
under 'Chairman', 'Operations' under 'Software', 'Payroll' under
'HR', 'Accounts Payable' under 'Accounting', 'Accounts Receivable'
under 'Accounting');
```

Security policy definition

A *security policy* is made up of one or more (up to 16) security components and can be created this way:

```
CREATE SECURITY POLICY company COMPONENTS level, department,
organization WITH IDSLBACRULES;
```

This defines the policy as well as the composition of the security labels that belong to this policy. That is, the labels have values for the three components: level, department, and organization.

The *WITH IDSLBACRULES* clause defines the rules for reading and writing data. IDS 11 has only one set of rules, and those rules are described in the document located at:

`http://publib.boulder.ibm.com/infocenter/idshelp/v111/index.jsp?topic=/com.ibm.sqls.doc/sqls287.htm`

IDSLBACRULES is the default value also; therefore, this clause is not strictly necessary in the SQL statement.

Security label definition

A *security label* is part of exactly one security policy and includes one value for each component in that security policy. A value in the context of a component is a list of zero or more of the elements allowed by that component. Values for array-type components can contain zero or one element; values for set and tree type components can have zero or more elements.

You can create a security label this way:

```
CREATE SECURITY LABEL company.director COMPONENT level 'Secret',
COMPONENT department 'Product Development', 'Quality Assurance',
COMPONENT organization 'Software';
```

Assigning labels

Security labels are applied to data in order to protect the data. They are granted to users to allow them to access protected data. A user can be granted, at the most, two security labels: one for read and the other label for write. A user can also have the same label for both read and write access. When a user tries to access protected data, the user's security label is compared to the security label that protects the data. The protecting security label blocks certain security labels and does not block other security labels per the LBAC access rules that are defined in the next section. If a user's security label is blocked, the user cannot access the data.

This is how you grant a security label to a user:

```
GRANT SECURITY LABEL company.director TO 'sam' FOR READ ACCESS;
GRANT SECURITY LABEL company.president TO 'sam' FOR WRITE ACCESS;
GRANT SECURITY LABEL company.director TO 'susan' FOR ALL ACCESS;
```

LBAC rule set

The IDS LBAC rule set for comparison of security labels is known as IDSLBACRULES. When the values of two security labels are compared, one or more of the rules in this predefined rule set are used to determine if one value dominates another value.

Read access rules

The read access rules are applied when data is read (SELECT, UPDATE, and DELETE operations):

- IDSLBACREADARRAY: Each array component of the user's security label must be greater than or equal to the array component of the data security label; that is, the user can only read data at or below the user's level.
- IDSLBACREADSET: Each set component of the user's security label must include the set component of the data security label.
- IDSLBACREADTREE: Each tree component of the user's security label must include at least one of the elements in the tree component of the data security label (or the ancestor of one of these elements).

Write access rules

Write access rules are applied when data is written (INSERT, UPDATE, and DELETE operations):

- IDSLBACWRITEARRAY: Each array component of the user's security label must be equal to the array component of the data security label; that is, the user can write data only at the user's level.
- IDSLBACWRITESET: Each set component of the user's security label must include the set component of the data security label.
- IDSLBACWRITETREE: Each tree component of the user's security label must include at least one of the elements in the tree component of the data security label (or the ancestor of one of these elements).

Summary of access rules

We provide a summary of the access rules in Table 5-2.

Table 5-2 Summary of Access Rules

Rule name	Used when comparing the values of this type of component	Used when attempting this type of access	Access is blocked when this condition is met
IDSLBACREADARRAY	ARRAY	READ	The user's value is lower than the protecting value.
IDSLBACREADSET	SET	READ	There are one or more protecting values that the user does not hold.
IDSLBACREADTREE	TREE	READ	None of the user's values is equal to or an ancestor of one of the protecting values.
IDSLBACWRITEARRAY	ARRAY	WRITE	The user's value is higher than the protecting value or lower than the protecting value.
IDSLBACWRITESET	SET	WRITE	There are one or more protecting values that the user does not hold.

Rule name	Used when comparing the values of this type of component	Used when attempting this type of access	Access is blocked when this condition is met
IDSLBACWRITE TREE	TREE	WRITE	None of the user's values is equal to or an ancestor of one of the protecting values.

LBAC rule exemption

The IDS LBAC implementation provides a way to allow users to bypass one or more access rules for a security policy by granting the user an exemption on one or more of the IDSLBACRULES.

If a user holds an exemption on a particular rule of a particular security policy, that rule is not enforced when the user tries to access data that is protected by that security policy. A user can hold multiple exemptions. If the user holds an exemption to every rule that is used by a security policy, the user has complete access to all data protected by that security policy.

This is the way to grant an exemption to a user:

```
GRANT EXEMPTION ON RULE IDSLBACREADSET FOR company TO 'sam';
GRANT EXEMPTION ON RULE ALL FOR company TO 'sam';
```

Exemptions are useful when a user must load data that has a variety of security levels. The exemption bypasses the usual checks, so that the limitation on writing data only at a single level is avoided. Bulk data loading is one example of when an exemption might be required.

An exemption might also be required in order to change the security label on rows. Data is written only with the security label of the user doing the insert or update. To change to another label, the modification must be done by another user or by a user who is exempt from the checks.

This means that the security policy and choice of label for each row or column needs to be carefully designed, and only appropriate users must make changes. This might also mean that certain users do not have any write access in order to assure that they do not make inadvertent changes to any security labels.

How are security labels compared

When a user tries to access data protected by LBAC, the user's LBAC credentials are compared to one or more security labels to see if the access is

blocked. A user's LBAC credentials are any security labels that user holds plus any exemptions that the user holds.

Security labels are compared component by component. If a security label does not have a value for one of the components, an empty value is assumed. As each component is examined, the appropriate rules of the LBAC rule set are used to decide if the elements in the user's label for that component must be blocked by the elements for the same component in the protecting label (row or column label). If any of the user's component values in the label are blocked by the protecting security label, access is denied.

If the user holds an exemption for the rule that is used to compare two values, that comparison is not done and the protecting value is assumed not to block the value in the user' security label.

Data protection

A table is said to be *protected* when it is associated with a security policy and either rows, columns, or both are protected by security labels. Data in a table can only be protected by security labels that are part of the security policy protecting the table. Data protection, including adding a security policy, can be done when creating the table or later by altering the table. A table can be associated with only one security policy.

When a protected table is accessed, IDS enforces two levels of access control. The first level is the traditional Discretionary Access Control (DAC). That is, IDS verifies whether the user attempting to access the table has been granted the required privileges to perform the requested operation on that table. The second level of access control is LBAC, which can be at the row level, column level, or both.

Row protection

A table can be protected with row level protection granularity by attaching a security policy to the table and by specifying the column where the row security label is stored. IDS has introduced a new data type, IDSSECURITYLABEL, for storing row security labels. Security labels are associated with rows (stored in the IDSSECURITYLABEL type column) when rows are inserted into the table or during alter, where the default security label specified is applied to all existing rows in the table. Row security labels can be modified by updating the row and specifying a new security label for the IDSSECURITYLABEL type column. A table can have at the most one column of type IDSSECURITYLABEL.

To create a row-protected table:

```
CREATE TABLE T1 (C1  IDSSECURITYLABEL, C2  int) SECURITY POLICY
company;
```

```
ALTER TABLE T2 ADD (LBL  IDSSECURITYLABEL DEFAULT 'president'), ADD
SECURITY POLICY company;
```

Note: Adding a security label to a table that contains data is different from adding a label to an empty table. For a table with data, a default value *must* be included, but for an empty table, a default value is not required.

Note: The ALTER TABLE statement in the example performs two actions: adding the label column and adding the security policy. Performing these two actions in individual SQL statements fails. A label cannot be added without a policy, and a policy requires a label column.

Column protection

A database can be protected with column-level protection granularity by attaching a security policy to a table and by attaching a security label to one or more columns of that table. When a column is associated with a security label, that column is referred to as a *protected column*. Column security labels cannot be modified. The existing security label for the column has to be disassociated and a new security label attached if the column security label is to be modified. A table can have any number of protected columns.

To create a column-protected table:

```
CREATE TABLE T1 (C1  int, C2  char (10) COLUMN SECURED WITH director)
       SECURITY POLICY company;
ALTER TABLE T2 MODIFY (C1 int COLUMN SECURED WITH president),
      ADD SECURITY POLICY company;
```

Data access

The SQL for working with secured data is no different from the SQL for unsecured data. Rows are inserted, updated, deleted, and selected in exactly the same way in either case. The only difference is that the security label checks are applied to new data, and less privileged users do not see data that is inserted by more privileged users.

Database security administrator (DBSECADM)

The database security administrator (DBSECADM) is a new built-in role introduced to perform all security-related administration, including setup of the LBAC security infrastructure. It is a server level role and can be granted only by the database server administrator (DBSA).

You might find it useful to create a unique user ID for the database security administrator and to have the people who hold the position use that user ID to

perform all the security-related operations, such as creating labels and components.

The LBAC security infrastructure setup includes setting up security components, security policies, security labels, granting security labels and exemptions to users, and protecting tables by attaching policies.

To grant the DBSECADM role to users:

```
GRANT DBSECADM TO 'sam';
```

Built-in security label functions

IDS provides three built-in SQL functions for working with security labels:

- SECLABEL_BY_COMP: This function builds a security label (internal encoded representation) by specifying a security policy and values for each of the components in the label. The return value has a data type of IDSSECURITYLABEL. This function is used in insert and update operations to provide the row security label for a data row.

  Here are two examples:

  ```
  INSERT INTO T1 VALUES (SECLABEL_BY_COMP('company', 'Secret:(Product
  Development,Quality Assurance):Software', 1);

  UPDATE T1 SET C1 = SECLABEL_BY_COMP('company',
  'Public:Marketing:Software') WHERE C2 = 2;
  ```

- SECLABEL_BY_NAME: This function builds a security label (internal encoded representation) by specifying a security policy and the name of a security label that is part of that security policy. The return value has a data type of IDSSECURITYLABEL. This function is used in insert and update operations to provide the row security label for a data row.

  Here are examples:

  ```
  INSERT INTO T1 VALUES (SECLABEL_BY_COMP('company', 'director', 1);

  UPDATE T1 SET C1 = SECLABEL_BY_COMP('company', 'president') WHERE C2
  = 2;
  ```

- SECLABEL_TO_CHAR: This function returns a string representation of the values that make up a security label. This function is used in select operations to retrieve the row security label column. It can be thought of as the reverse of SECLABEL_BY_COMP.

  Here is an example:

  ```
  SELECT SECLABEL_TO_CHAR('company', C1), C2 FROM T1;
  ```

  The command results are depicted in Table 5-3 on page 146.

Table 5-3 Command results

C1	C2
'Secret:(Product Development,Quality Assurance):Software'	1
'Public:Marketing:Software'	2

An LBAC example

Because there is not a default database security administrator (DBSECADM), you must grant that role to a user. In this example, we simply used user *informix*. But in a commercial deployment, it is prudent to assign the security role to known users.

With the database selected, we grant the example security administrator role with the following command:

```
GRANT DBSECADM TO 'Mary';
```

This example focuses on a software company with employees who need the privacy of their personal data assured. Because it is a small company, they have chosen to have three levels of assigned risk, and therefore, security: `HIGH`, `MEDIUM`, and `LOW`. Furthermore, they are currently flat in perspective of management levels and have a `President`, `Human Resources`, and other staff. They also have guests who come on-site for training and other activities for whom they need to have emergency contact information as a prudent precaution. Finally, the company products span from reselling hardware to the software that runs on it.

In order to encompass future requirements, we define three areas of security to encompass those different aspects, as follows:

```
CREATE SECURITY LABEL COMPONENT levels
ARRAY ['HIGH', 'MEDIUM', 'LOW'];
CREATE SECURITY LABEL COMPONENT compartments
SET {'President', 'HR', 'Sales', 'Tech'};
CREATE SECURITY LABEL COMPONENT groups
TREE ('Company' ROOT, 'Hardware' UNDER 'Company', 'Software' UNDER
'Company', 'Storage' UNDER 'Hardware', 'Disk' UNDER 'Hardware');
```

Note: Traditional discretionary access control needs to be granted to the users accessing this database (at least for *CONNECT* privileges) and executing the security procedures shown here (DBSECADM).

With the components defined, we now define a policy that allows both row and column control if we create or alter the table to support that level of control.

```
CREATE SECURITY POLICY mypolicy COMPONENTS levels, compartments, groups
WITH IDSLBACRULES;
```

Next, the label that Mary, our human resource manager, uses:

```
GRANT SECURITY LABEL mypolicy.HR to 'Mary' FOR ALL ACCESS;

CREATE TABLE employees (seclabel IDSSECURITYLABEL, employeeId int, phone
char(12), SSN char(9) COLUMN SECURED WITH HR) SECURITY POLICY mypolicy;
```

At this point, we have created a table to enforce a security label for each row, and *seclabel* is the column name that was used. Because social security numbers are extremely private, we have enforced a column level access also using the HR label of the policy. We want to maintain good standards in the standard database privileges also, so we define the following permissions:

```
REVOKE ALL ON employees FROM public;
Permission revoked.
GRANT ALL ON employees to 'Mary','Fred';
Permission granted.
```

At this moment, the user performing the DBSECADM role cannot insert or read from this table, only Mary can. Because there is a lot of work to do, Mary has hired an assistant, Fred, to help her enter the employee data. As you can see, we have already granted Fred permission to access the table. But, what you did not see was the *grant connect* to the database for Fred.

The following is what happens when there is an attempt to access the data:

```
SELECT * FROM employees;
8245: User cannot perform READ access to the protected column (ssn).
```

This is the output of an access attempt by the DBSECADM user who also happens to have DBA rights. Fred has the same trouble, so we allow him to help Mary by issuing:

```
GRANT SECURITY LABEL mypolicy.HR to 'Fred' FOR WRITE ACCESS;
Security label granted.
```

Note that because Fred is an outside contractor and only a temporary assistant, we do not want to allow him too much access; therefore, we used the FOR WRITE clause on the GRANT. Also, note that an error occurs if the DBSECADM tries to grant access through a label to the DBSECADM.

We now have an operating security policy established. Although it is a simple policy, it allows Fred to insert rows that have the HIGH label of the level's array, the HR label of the department's set, and the root label, COMPANY, of the group's tree. Fred cannot see any of the data that he inserts, but Mary can see all of the work done on the table by any user. Additional policies can be created

to grant to Fred to allow him to see his inserts, but not Mary's, while still allowing Mary to see all rows.

In Fred's context, or environment, as shown in the following example, he cannot access the complete employee record, but he can access specific columns of that record:

```
select * from employees;
8245: User cannot perform READ access to the protected column (ssn).
SELECT phone FROM employees;
```

Conclusion

With Label-Based Access Control (LBAC), administrators can control the read and write access of a user to a table at the column and row levels. LBAC is the most appropriate solution for clients who want to protect sensitive data and control access to this data.

5.3 ENCRYPTION

IDS V10 introduced a set of functions to encrypt and decrypt data as it moves to or from the database. This encryption is a physical layer of security, preventing a user who has obtained access to the physical data outside of the RDBMS system the ability to read or interpret the information. IDS 11 adds the ability to encrypt communication between an HDR pair and to secure the transmission of data over unsecured networks, including the Internet. You use new configuration parameters to enable encryption between the HDR servers and specify encryption options. After you enable encryption, the HDR primary database server encrypts the data before sending it to the secondary database server. The secondary database server decrypts the data. HDR encryption works in conjunction with Enterprise Replication encryption and operates whether Enterprise Replication encryption is enabled or not.

IDS 11 has not changed the encryption algorithms or cipher modes. To conform to the requirements of the Federal Information Processing Standards (FIPS), the internal cryptography engine changed from using the OpenSSL library to using the IBM standard cryptography library (the ICC library), which is an internal library developed and maintained by IBM. This change was required for FIPS compliance.

5.3.1 Scope of encryption

Data encryption applies separately to each column of each table. The work is done in functions within the SQL statement, as shown in Example 5-9 on page 151.

Only character or binary large object (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 part or all of the columns in any row, and you can choose to encrypt part or all of the rows in any column. These functions apply cell by cell. For example, you might choose to encrypt all the employee addresses (that column in every row) but only the salaries above $100,000 (that column in only a few 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 must use views or modify individual SQL statements to encrypt or decrypt the data.

5.3.2 Data encryption and decryption

The basic technique is straightforward. You use your choice of encryption function to insert or modify data.

We use an example here to illustrate the process. In Example 5-8, 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 Example 5-8, we use the SET ENCRYPTION PASSWORD statement to simplify the syntax. This also assures 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 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. You can use either function for any column.

Example 5-8 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  vOAQAAAAEAW4Xcnj8bH292ATuXI6GIiGA8yJJviU9JCUq9ngvJkEP+/BzwFhGSYw==
decrypted_user_id  richc

plain_user_id      maryj
enc_id  OgH8QAAAAEASv+pWbie/aTjzqckgdHCGhogtLyGH8vGCUq9ngvJkEP+/BzwFhGSYw==
decrypted_user_id  maryj

2 row(s) retrieved.

Database closed.
```

Example 5-9 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 5-9 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 a few of the rows have their own unique passwords. The rest of the rows use the common password from a SET ENCRYPTION PASSWORD statement. This demonstrates that the password in a function call takes precedence over the previously set password.

Be extremely 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 cannot get them all back with a single SELECT statement, because there is no way to include multiple passwords in a single function call. That is demonstrated in the first two SELECT statements in Example 5-9. To get multiple rows, you have to use a UNION to get each row of the result set.

Example 5-9 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.
```

5.3.3 Retrieving encrypted data

As Example 5-8 on page 149 and Example 5-9 on page 151 show, 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, you retrieve the BLOB data using the DECRYPT_BINARY() function.

Example 5-10 demonstrates the DECRYPT_BINARY() function. We alter the `catalog` table in the `stores` database.

Example 5-10 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 "Erasmo";
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 one time. This enhances security.

But, that means *one encrypted value cannot be correctly compared to another encrypted value*. Before you compare values, decrypt both of them to get an accurate comparison, which we demonstrate in Example 5-11. You must decrypt the values to use in a comparison before you compare them.

Example 5-11 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.
```

5.3.4 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 5-12 on page 154 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 a specific value in the indexed column.

If indexes are required on encrypted data, 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 5-12 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(*))

              44

1 row(s) retrieved.

> create index estat on stock(e_status);

Index created.
```

5.3.5 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 approach is to define views to hide the encryption functions. Example 5-13 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 do not need to be changed. To accomplish that goal, we use the modified `cust_calls` table created in Example 5-8 on page 149. 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 ensure that 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 certain rows in this method. If you want to do that, you need a stored procedure to retrieve the encrypted data so that you can pass the password as a parameter. Example 5-13 demonstrates this technique.

Example 5-13 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     O
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.
```

5.3.6 Password usage

You can use a common password established with the SET ENCRYPTION PASSWORD statement, or you can 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 an SQL statement overrides the more global password.

If you provide no password and have no SET ENCRYPTION PASSWORD in effect, you get an error.

5.3.7 Encrypting passwords during transmission

The use of an OS user ID and password takes advantage of the password being encrypted in the files that are 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 concsm.cfg file. Both of these files are usually in $INFORMIXDIR/etc. The concsm.cfg file can be elsewhere, but then the INFORMIXCONCSM 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 conscm.cfg file. If only one system configures encryption, the password is either encrypted when the DBMS is not expecting it or not be encrypted when it needs to be encrypted.

Be sure that the client applications use the proper DBSERVERNAME or DBSERVERALIAS. By having multiple names, certain connections can have encrypted passwords while other connections do not.

Example 5-14 on page 158 shows the sqlhosts file entries and related concsm.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 sqlhosts entry does use password encryption. Note that the name provided in the sqlhosts entry, SPWDCSM, in this case, must match the entry in the concsm.cfg file.

In the concsm.cfg file, the parameters following the CSM name vary for different CSM modules. In this case, the first parameter is the full path of the library for performing the encryption and decryption. The second parameter is unused. The third parameter is null in Example 5-14. See *The IDS Administrators Guide*, G251-2267, for details about 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 incorrect, you get error -930, "`Cannot connect to database server (`*`servername`*`)`."

Example 5-14 Sample SQLHOSTS and CONCSM.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=(SPWDCSM)

concsm.cfg entry;
SPWDCSM("/usr/informix/lib/csm/libixspw.so", "", "")
```

5.3.8 The number and form of passwords

Encryption passwords are a difficult subject, as passwords are for most people. You need to balance the simplicity of a single password for all the data against the difficulty of remembering multiple passwords and when each password 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: A password must be at least six but not more than 128 characters. There are no rules for starting or ending characters, required alphabetic, numeric, or special characters.

At a certain point, a number of passwords must be chosen. 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, which 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 approach is to associate a password with the data that is used in each job role within the organization. If the jobs use disjointed sets of data, this works nicely. If there is significant overlap in the data that is used, this might not work as well.

Another alternative approach to consider is to have a password for each encrypted column. That might be a rather large number, but it is somewhat more secure.

If the database has natural groupings of tables, consider having a password for each 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 password for each major category.

5.3.9 Password expiration

IDS does not provide or enforce any password expiration. If your standards require password changes at a certain interval, you have to develop a way to update your data to the new password. *Without the correct password, the data cannot be retrieved by any means*. Therefore, you have to develop a way to make the changes and schedule the updates to change the passwords before they expire.

The SQL to perform this type 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 5-15 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 5-15 Encryption password updating

```
update call_type
   set enc_code =
            encrypt_aes(decrypt_char(enc_code, "Dennis"), "Erasmo")
where call_code = 'X';
```

Example 5-15 is not a practical example, because it affects only a single row. A more typical operation is to update the common password for all the rows in a table or group of tables. Example Example 5-16 on page 160 demonstrates that operation. The example works on the `sales_rep` table of the `stores` database.

Example 5-16 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('1lmkQAAAACAxE6fSstEGWCEdN/qDRyROzZltX4YuOU9go5dXfYD/ZY=','1C/
             QQAAAACA62TTuNf9q8csJOhyVB8wQTZltX4YuOU9go5dXfYD/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('1qVYQAAAACAfyXfzOuyd2CJJKqhpkLVPjZltX4YuOU9go5dXfYD/ZY=','1hH
             sQAAAAEAz6TfNs9/MsWxEqEzlxMYtNwst7RWg8hzNmW1fhi45T2Cjl1d9gP9lg==')

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.
```

5.3.10 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.

5.3.11 Using password hints

The encryption functions allow you to include a hint to assist the user in recalling the password. The GETHINT() function retrieves the hint. Example 5-17 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 5-17 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   1bZIQAAAAIAP4PoYtU4yma3rWOVDFEoDKmHK62JYMtlEJPPR3MGDXXiR/OmjlcMulDP
            8EVcROPsohMBOZQaB2w=

1 row(s) retrieved.

> select gethint(enc_descr) from call_type where call_code = 'X';

(expression)

abbotcostello

1 row(s) retrieved.
```

5.3.12 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 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 5-18.

Example 5-18 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',
'12345678901234567890'));

(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',
'123456789012345678901'));

(expression)
          99

1 row(s) retrieved.
```

Alternatively, the *Guide to SQL: Syntax*, G251-2284, manual contains a chart with size estimates for size ranges for each of the encryption methods, with and without hints. The manual also provides the formulas for calculating exact sizes.

5.3.13 Errors when the space is too small

If you fail to make the columns for encrypted data large enough, you are allowed to insert data into those columns with no warnings or errors. However, any attempt to retrieve the data fails. The data is truncated during the insert or update operation, but no notice is given. The truncated data is then insufficient to allow decryption. Therefore, the decryption functions fail. Example 5-19 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 5-19 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_descrip+  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.
```

5.4 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 defined enforcement must 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 they are more likely to be corporate officers, such as 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 that you must consider. In particular, IDS 11 checks a number of files as the DBMS starts up and puts warning messages in the message log if key files do not have properly secure permissions. You need to periodically review the log to correct any problems that you discover.

Also, be sure that 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 must be part of the corporate security policies.

There are other considerations, too. For example, the database backup and log archive files must be stored securely, which means having the most restrictive permissions that your business needs can tolerate and storing the media in a secure location.

In this chapter, we do not discuss all of these other aspects of security, but you must do what is required by policy and restrict information based on the requirements of your organization.

6

Easy administration

In this chapter, we discuss the new IDS 11 capabilities for making the job of a database administrator (DBA) easier. For example, the new capabilities allow a DBA to have more control over backup and restore images using ontape and improve the ability to collect and manage statistics. Also, we now have the mechanisms to monitor and manage an instance remotely.

IDS 11 delivers what we call an "*administration free zone*" based on:

- Extended business continuity with multiple high availability data replication shared-disk secondary servers and remote standalone secondary servers
- Database administration tasks easily integrated with the new SQL Application Programming Interface (API)
- Standard tasks scheduled automatically or conditionally
- Improved performance of Enterprise Replication
- Updated SQL tracing for monitoring performance

The changes allow a DBA more control over the instance with the ability to automate many tasks that currently require intervention. And, the ability to script many tasks by using SQL and stored procedures makes those tasks platform independent and allows a DBA to consolidate many of the tasks into a single environment or to monitor many environments.

6.1 Backup and recovery changes

An important part of any activity that a DBA performs is the backup and recovery strategy. There exists two main strategies within IDS: an internal strategy, ontape and OnBar, and an external strategy that gives a DBA greater flexibility. The choice of which strategy to pursue is up to the DBA with each strategy having its specific strengths and weaknesses. For example, with IDS 11, many important changes have been made to ontape.

As database sizes have grown and data sophistication and complexity has increased over time, requirements to manage data have changed. A key component of these new requirements has been the ability to back up and restore only what was needed or critical in a timely manner, either the entire instance or a focused subset of the data. IDS has always either led or kept pace with the industry in terms of data backup and restoration.

Another feature of the internal strategy is the ability to take a backup and roll forward through logical logs to a time after the backup was completed. There are different levels within the internal strategy that allow the DBA the ability to back up the entire instance (level 0) and then to back up only the changes that have occurred in the instance since the last level 0 (level 1 or level 2). The levels of backup that only capture the changes since the last level 0 allow those backups to take less storage space and typically to complete in a shorter period of time. And, the ability to apply logical logs still holds when restoring the level 1 or level 2 backup.

6.1.1 Backup and restore history

IDS has always 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. Backups can be executed on production tables and 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 types 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 you want. 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 OnBar utility suite is used, include either the entire instance or just a subset of the spaces. Both ontape and OnBar, the most frequently used utilities, support two levels of incremental backups, as well as a full backup, as summarized in Table 6-1 on page 170.

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 must be taken. Finally, to balance media usage and restore time in an environment where there are many data changes, a level 0 can 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 6-1, 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 that we just discussed, 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 need to be restored, followed by the logical logs.

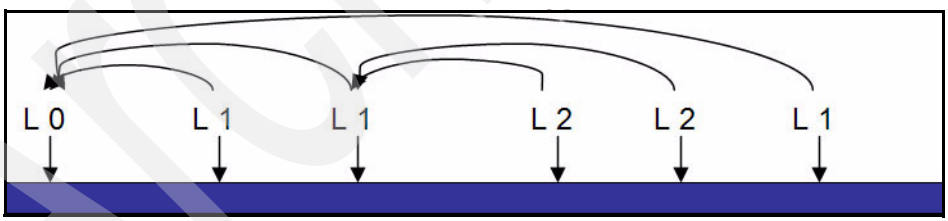


Figure 6-1 Incremental backup and restore steps

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 backup level must be restored as part of this cold restore along with the logical logs to bring the recovery point as close as possible to when there was a failure of the instance. If a non-critical dbspace 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 can be in the recovery of an instance supporting online transaction processing (OLTP) and historical analysis. The cold restore brings 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.

The ontape backup strategy

The backup strategy that has been included in the instance since its beginning is ontape. It is a sequential method to back up the instance. This strategy in previous versions has been to write the backup images to either tape or disk. Control over the backup can be controlled by performing different levels of backups.

A level 0 backup takes all the instance pages to either disk or tape. This usually takes the longest amount of time to perform. The various levels and a description of each level are listed in Table 6-1.

Table 6-1 IDS backup levels

Backup level	Description
0	Complete instance backup. NOTE: Temporary space pages (either standard or smart) are never 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

The OnBar backup strategy

The necessity to have a backup strategy that allowed for parallel backup of the chunks came as the size of databases increased. Another reason for OnBar was to allow for other storage managers to control disk or tape usage. There are many storage managers on the market and OnBar can perform with many of them. The setup of the storage manager is dependent upon the specific vendor from which it was purchased. The storage managers allow for control of when a device can be reused.

OnBar is referred to as a utility suite, because it actually has two components, the OnBar API and the Informix Storage Manager (ISM), a basic tape management system that can be used in conjunction with the OnBar API. Unlike ontape, all user interaction prompts are expected to be handled by the tape management system making OnBar the preferred utility for unattended backups, as well as integration into an existing enterprise backup and recovery plan. The OnBar 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 can 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.

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: The 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 back up UNIX, Windows, and other servers and their file systems.

The OnBar utility provides greater power, flexibility, and granularity when executing backup and restore operations than ontape. 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 OnBar API is not bound by a limit of physical backup devices (within reason), operations can be executed in parallel and concurrently. Unlike ontape, which uses a single I/O thread to read data from chunk 0 to N, OnBar forks 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 five device tape jukebox available, a backup operation can be defined in the storage management software to invoke five threads to stream data to all devices. When the job is executed, the instance allocates the threads to the first five spaces to be backed up. When one thread finishes its space, the instance points the thread to the next space to be backed up until all requested spaces have been backed up.

More than one operation can execute concurrently as well. For example, with the same five device jukebox, two backup jobs can be defined to back up different spaces, one using N devices, the other 5 - N devices. Both jobs can then be executed at the same time if desired.

An OnBar 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 the opposite is true also. The command to perform a serial level 0 archive with OnBar is:

```
onbar -b -w -L 0
```

The option that indicates a serial backup is the -w.

There is a small amount of overhead in using OnBar 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 extremely tiny, consisting of just a few tables. While it contains a record of backup objects and operations, most storage management applications have a utility that communicates through the OnBar API to this database to purge entries as save sets and media expires.

External backup and restore

External backups and their associated recovery operations are primarily performed outside of either the ontape or OnBar utilities. Ontape or OnBar is only

used to back up and restore the logical logs when the external backup and restore operation has completed. In certain 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 can happen within a disk farm by segregating a set of disks that have been acting as mirrors and remirroring with another set of disks. The partitions on the segregated disks can then be mounted and used by the OS as new devices. Certain 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 cannot maintain logical consistency during backups.

Another option if a third mirror set of disks is unavailable 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 OnBar.

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 OnBar 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.

With external backup and restore, clients have the control over how they want to perform the backup. The steps for an external backup are depicted in Example 6-1 on page 174.

Several clients use hardware mirroring to accomplish the external backup. Other clients might use commands that allow copying or ftp the chunks to a separate directory or machine. The control over the methodology is given to the DBA or system administrator.

The control of the methodology and the overall speed are two of the main reasons clients choose this strategy.

Example 6-1 External backup steps

```
onmode -c block
<perform the external backup>
onmode -c unblock
```

6.1.2 Backup and restore strategies

The changes to ontape and also OnBar in Version 10 of IDS included the ability to set the configuration parameter TAPEDEV to Standard Input/Output (STDIO). Also, the ability to perform a table level restore was introduced. With STDIO for TAPEDEV, the DBA has the ability to send the contents of the restore directly to a file. In addition, the DBA can send the contents to a compression program, such as compress or gzip. This allows the DBA to store the contents of the backup directly into a smaller footprint.

Using ontape

In Version 10 of IDS, the ability for the backup to be sent to STDIO was introduced. The ability of the DBA to validate a backup also exists. If TAPEDEV is set to STDIO, the output of the backup is sent to the terminal, so a receptacle must be named. The syntax of the **ontape** command remains the same. Example 6-2 illustrates the concept.

Example 6-2 Using STDIO for TAPEDEV

```
TAPEDEV STDIO (from onconfig file)
$ ontape -s -L 0 > stdio_test
```

The size of the result of the backup is the same whether the backup is sent to STDIO or to another device. If the DBA wants to compress the image of the backup image, it can be compressed while the backup takes place. Example 6-3 shows an example of this process.

Example 6-3 Using STDIO and compression for TAPEDEV

```
TAPEDEV STDIO (from onconfig file)
$ ontape -s -L 0 | compress > stdio_test_1.Z
```

Table level restore

The functionality of archecker was extended in Version 10 of IDS to allow for a table level restore. This permits a DBA to restore a single table as opposed to

having to restore an entire instance. The AC_CONFIG is demonstrated to use the stores_demo database for a test and to restore the customer table.

The sample for an AC_CONFIG is shown in Example 6-4.

Example 6-4 AC_CONFIG with parameters included

```
AC_MSGPATH      /testing/prods/1110FB6/ac_msg.log          # archecker message
log
AC_STORAGE      /testing/prods/1110FB6/ # Directory used for temp storage
AC_VERBOSE      1                           # 1 verbose messages 0 terse
messages
AC_TAPEBLOCK    32
AC_SCHEMA       /testing/prods/1110FB6/helpme_cmd
```

The commands that the archecker actually uses to create the file for the data are in Example 6-5.

Example 6-5 Command file for archecker to read for table level restore

```
database stores_demo;
create table customer
  (
    customer_num serial not null ,
    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)
  ) ;
create external table customer_unl
  (
    customer_num serial not null ,
    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)
  ) using("./customer.unl", delimited);
```

```
insert into customer_unl select * from customer;
restore to current with no log;
```

The syntax to get archecker to extract the contents of the `customer` table using an ontape backup is:

```
archecker -tXvs
```

The syntax changes if OnBar is the chosen method to perform backups. The primary change is that the “t” option changes to a “b”. Consult the *Informix Backup and Recovery Guide* for complete details.

The results of the archecker are written to the AC_MSGPATH file and look similar to Example 6-6 in a successful restore situation. A file called customer.unl is created that is an ASCII-delimited file with the rows of data in it.

Example 6-6 AC_MSGPATH file with successful table level restore

```
2007-05-29 12:18:15
----------------------------------------
STATUS: IBM Informix Dynamic Server Version 11.10.FB6TL
Program Name:   archecker
Version:        8.0
Released:       2007-04-19 22:57:09
CSDK:           IBM Informix CSDK Version 2.91
ESQL:           IBM Informix-ESQL Version 2.91.FN219
Compiled:       04/19/07 22:58  on HP-UX B.11.11 U

STATUS: Arguments [-tXvs]
STATUS: AC_STORAGE              /testing/prods/1110FB6
STATUS: AC_MSGPATH              /testing/prods/1110FB6/ac_msg.log
STATUS: AC_VERBOSE              on
STATUS: AC_TAPEDEV              /testing/prods/1110FB6/bkup/
STATUS: AC_TAPEBLOCK            32 KB
STATUS: AC_LTAPEDEV             /dev/null
STATUS: AC_LTAPEBLOCK           32 KB
STATUS: AC_SCHEMA               /testing/prods/1110FB6/helpme_cmd
TIME: [2007-05-29 12:18:15] All old validation files removed.
STATUS: Dropping old log control tables
STATUS: Restore only physical image of archive
STATUS: Extracting table stores_demo:customer into stores_demo:customer_unl
WARNING: Source table [customer] has no storage option
ERROR: No storage space to scan
STATUS: Archive file /testing/prods/1110FB6/bkup/odgreen_86_L0
STATUS: Tape type:      Archive Backup Tape
STATUS: OnLine version: IBM Informix Dynamic Server Version 11.10.FB6TL
STATUS: Archive date:   Tue May 29 12:14:14 2007
STATUS: Archive level:  0
```

```
STATUS: Tape blocksize:  32768
STATUS: Tape size:  2147483647
STATUS: Tape number in series:  1
TIME: [2007-05-29 12:18:15] Phys Tape 1 started
STATUS: starting to scan dbspace 1 created on 2007-05-29 12:14:14.
STATUS: Archive timestamp 0X0002212B.
TIME: [2007-05-29 12:18:15] Found Partition customer in space rootdbs
(0x0010016
1).
TIME: [2007-05-29 12:18:15] Tape 1 completed
STATUS: Scan PASSED
STATUS: Control page checks PASSED
STATUS: Table checks PASSED
STATUS: Table extraction commands 1
STATUS: Tables found on archive 1
STATUS: LOADED: stores_demo:customer_unl produced 28 rows.
TIME: [2007-05-29 12:18:15] Physical Extraction Completed
TIME: [2007-05-29 12:18:15] Unload Completed

STATUS: archecker completed Physical Restore pid = 12133 exit code: 3
```

External backup and restore

An external backup is when the backup methodology is outside the control of the IDS instance. One method to perform an external backup is to have hardware mirroring in place. To take a backup with this method, an example set of steps is listed in Example 6-7 on page 178.

The **onmode -c block** forces the instance to block any work that allows any modification through an insert, update, or delete operation. The step to break the mirror is a specific command that the particular hardware vendor provides. The **onmode -c unblock** then allows the instance to operate as normal. This sequence can usually be accomplished very quickly. The major advantage to the external backup strategy is the overall speed.

Important: Remember to issue the **onmode -c block** before performing the external backup. Otherwise, the consistency of the external backup cannot be guaranteed.

Remember to issue the **onmode -c block** when performing the backup. If the backup is performed without issuing the onmode command, it can mean that updates might occur on the instance. With the instance in a blocked mode, updates are suspended and data consistency can be assured. Without the data consistency, the possibility of corruption exists.

Example 6-7 External backup

```
onmode -c block
<perform the command that actually does the backup>
onmode -c unblock
```

To perform an external restore, typically the instance is brought offline and the image from the external backup is applied to the instance. The instance can then be brought online and be functional.

Because the external backup is outside the scope of control of the instance, the granularity of a level 1 or level 2 restore does not exist. Also, performing a warm restore using an external backup is not possible.

The external restore is at the instance level, and data consistency with any further granularity cannot be assured.

6.1.3 New features for ontape

The ontape utility has been modified for IDS 11 to allow for backup and restore using directories instead of a tape device or a specific file.

One of the new features allows for ontape to use a directory, such as the TAPEDEV. An advantage to this is that multiple backups can be performed to the same directory, space permitting. If the TAPEDEV is set to a file, successive backups overwrite the information. The directory allows for multiple backups to the same directory and the engine takes the responsibility of renaming the existing files so that they do not get overwritten.

The **ontape** syntax for performing a level 0 backup remains unchanged if a directory is used for TAPEDEV, which is shown in Example 6-8.

Example 6-8 Using ontape to back up to a directory

```
TAPEDEV <path to a directory>
ontape -s -L 0
```

The **ontape** syntax to perform a restore remains unchanged from previous versions also, which is shown in Example 6-9 on page 178. If multiple backups are in the directory, the restore command uses the most recent backup as the restore information.

Example 6-9 Using ontape to restore from a directory

```
TAPEDEV <path to a directory>
```

```
ontape -r
```

Advantages

The major advantage in using a directory for the TAPEDEV is that the engine takes care of renaming the resulting file. If a subsequent backup is performed, the previous backup in that directory is renamed, which is shown in Example 6-10.

Example 6-10 Examining the file where the backups have been placed

```
odgreen_86_20070513_110922_L0
odgreen_86_L0
<machine name>_<SERVERNUM>_L<level of archive>
```

Disadvantages

Because all the backup images will reside in the same directory, it is up to the DBA or systems administrator (SA) to monitor the directory for adequate space. Also, the DBA or SA is responsible for adequate safeguards for the directory. Making sure that the files are not deleted accidentally is a responsibility of the DBA.

6.2 Session properties

In this section, we describe how you can change the session properties of an application without actually changing the application code. You may want to do this, for example, to run an application with a different PDQ setting or a different isolation level. Typically to do this you would have to change the application code to include the appropriate commands. And, if you are working with a purchased application, making changes is typically not even feasible.

However, with IDS you can now change the session properties outside of the application. You do this by creating specific types of stored procedure that contain the appropriate commands. These specific stored procedures execute either when the database is opening or when it is closing. And, they must be given the specific names *sysdbopen* and *sysdbclose*.

They can include any SET, SET ENVIRONMENT, SQL, or SPL statements that are appropriate for execution when a database is being opened or being closed. The general restrictions on SQL statements that are valid in SPL procedures also apply to these procedures. The procedures will be executed in the database, when it is opening and/or when it is closing. After their execution, control is returned to the user.

A user having DBA privileges can also create qualified stored procedure names for a particular user (by defined *USERNAME*) or for the *PUBLIC* group, for a particular database. For example, the qualified names for the procedures could be as follows:

- username.sysdbopen
- public.sysdbopen
- username.sysdbclose
- public.sysdbclose

where *USERNAME* is the OS username and *PUBLIC* is a keyword.

If these procedure names exist, each time the user (with the defined *USERNAME*) connects to a database using either a DATABASE or CONNECT TO statement, IDS will automatically execute the *USERNAME*.sysdbopen procedure. If that procedure name does not exist, the procedure *PUBLIC*.sysdbopen will be executed if it exists. If neither exists, the database opens with the default settings.

Similarly, each time this user explicitly disconnects from a database, using either a CLOSE DATABASE or DISCONNECT statement, or implicitly by terminating the application without formally closing the session, *USERNAME*.sysdbclose will be automatically executed when closing the database. If this procedure does not exist, the *PUBLIC*.sysdbclose procedure will be executed. If neither procedure exists, the database closes with the default settings.

As an example, for *USER1*, you can define procedures that contain SET PDQPRIORITY, SET ISOLATION LEVEL, SET LOCK MODE, SET ROLE, or SET EXPLAIN ON statements that will execute whenever *USER1* opens the database with a DATABASE or CONNECT TO statement.

The SET PDQPRIORITY and SET ENVIRONMENT OPTCOMPIND statements, which are not persistent for regular procedures, are persistent when contained in the sysdbopen procedure. This feature allows the DBA to set the initial server properties for a database. But, it does not prevent users from overriding them within the application, after connecting to the database.

6.2.1 Define session properties

To set the initial environment for one or more sessions, create and install the sysdbopen( ) Stored Procedures Language (SPL) procedure. The typical effect of this procedure is to initialize properties of a session without requiring the properties to be explicitly defined within the session.

These procedures are exceptions to the general rule that IDS ignores the name of the owner of a UDR when a routine is invoked in a database that is not

ANSI-compliant. For UDRs other than sysdbopen( ) and sysdbclose( ), multiple versions of UDRs that have the same SQL identifier but that have different owner names cannot be registered in the same database unless the CREATE DATABASE statement that created the database also included the WITH LOG MODE ANSI keywords.

Note: When a user that is connected to the local database opens a different database using database:object or performs a distributed DML operation that references a remote database object using database@server:object notation, no sysdbopen procedure is invoked in the remote database.

6.2.2 Configure session properties

Restriction: Only a DBA or user *informix* can create or alter sysdbopen( ) or sysdbclose( ) in the ALTER PROCEDURE, ALTER ROUTINE, CREATE PROCEDURE, CREATE PROCEDURE FROM, CREATE ROUTINE FROM, DROP PROCEDURE, or DROP ROUTINE statements of SQL.

To set up a sysdbopen() or sysdbclose() procedure to configure session properties:

1. Set the IFX_NODBPROC environment variable to any value, including 0, to cause the database server to bypass and prevent the execution of the sysdbopen( ) or sysdbclose( ) procedure.
2. Write the CREATE PROCEDURE or CREATE PROCEDURE FROM statement to define the procedure for a particular user or the PUBLIC group.
3. Test the procedure, for example, by using sysdbclose( ) in an EXECUTE PROCEDURE statement.
4. Unset the IFX_NODBPROC environment variable to enable the database server to run the sysdbopen( ) or sysdbclose( ) procedure.

Example 6-11 sets the role and the PDQ priority for the PUBLIC group.

Example 6-11 PUBLIC group example of sysdbopen

```
create procedure public.sysdbopen()
set role to others;
set pdqpriority 1;
end procedure
```

6.2.3 Modifying session properties

To modify session properties upon the opening of a database, the syntax to create the stored procedure is depicted in Example 6-12. This procedure changes the configuration parameter of OPTCOMPIND to 1 and is made for all users. If a specific user only needs to have the configuration changed, for example, the user name is *myuser*, the syntax to change the OPTCOMPIND to 2 is shown in Example 6-14 on page 183.

Example 6-12 Syntax to create a stored procedure for sysdbopen

```
CREATE PROCEDURE public.sysdbopen()
SET ISOLATION TO REPEATABLE READ;
SET ENVIRONMENT OPTCOMPIND '1';
END PROCEDURE;
```

The value of OPTCOMPIND is set to 0 in the configuration file. The results of executing the stored procedure can be seen by using **onstat -g ses <*session id*>** and are shown in Example 6-13,. The optcompind is successfully set to 1.

Example 6-13 Result of sysdbopen stored procedure

```
onstat -g ses 22

IBM Informix Dynamic Server Version 11.10.FB6TL -- On-Line -- Up 00:03:52 --
39496 Kbytes

session                                    #RSAM    total      used
dynamic
id       user     tty      pid      hostname threads  memory     memory
explain
22       informix tc       20795    odgreen  1        90112      76152      off

tid      name     rstcb            flags    curstk   status
45       sqlexec  c000000006318118 Y--P---  61904    cond wait(sm_read)

Memory pools    count 1
name         class addr              totalsize  freesize   #allocfrag #freefrag
22           V     c000000007495040 90112      13960      96         11

name            free       used            name            free       used
overhead        0          3280            scb             0          144
opentable       0          2760            filetable       0          544
log             0          12096           temprec         0          16248
keys            0          632             ralloc          0          8736
gentcb          0          1584            ostcb           0          3416
sqscb           0          19496           sql             0          72
rdahead         0          160             hashfiletab     0          552
```

```
osenv          0          3016           sqtcb          0          3064
fragman        0          352

sqscb info
scb              sqscb            optofc   pdqpriority sqlstats optcompind
directives
c000000007508338 c000000007496028 0       0          0       1          1

Sess  SQL            Current            Iso Lock       SQL  ISAM F.E.
Id    Stmt type      Database           Lvl Mode       ERR  ERR  Vers Explain
22    -              stores_demo        RR  Not Wait   0    0    9.24 Off

Last parsed SQL statement :
  Database 'stores_demo'
```

The ability to change a session property for only a specific user when the user connects to or opens the database requires a different syntax than the public.sysdbopen() syntax. The <*username*>.sysdbopen() is the syntax to limit the command to a certain user. This syntax is shown in Example 6-14.

Example 6-14 Syntax of sysdbopen for a specific user

```
CREATE PROCEDURE myuser.sysdbopen()
SET ISOLATION TO REPEATABLE READ;
SET ENVIRONMENT OPTCOMPIND '2';
END PROCEDURE;
```

6.3 Improvements in statistics collection

In previous versions of IDS, there were restrictions on UPDATE STATISTICS when the operations were performed concerning when the new statistics were applied and used.

Situations existed where a DBA might have performed the proper tasks and the engine still might not have chosen the optimization path desired or expected. This can be caused by the statistics not being updated when the DBA thought they might be.

Creating a new index in previous versions did not automatically generate statistics for the index. So when a new index was created, it might not be used in the access plan.

6.3.1 Create index distribution implementation

With version 11 of IDS, CREATE INDEX automatically creates distributions and statistics for the leading column of the index:

```
UPDATE STATISTICS HIGH/MEDIUM
UPDATE STATISTICS LOW
```

When you upgrade to a new version of the database server, you might need to drop distributions to remove the old distribution structure in the sysdistrib system catalog table. UPDATE STATISTICS distributions are created automatically when an index is created either implicitly or explicitly. Sampling size is the data seen during the static phase of the online index build, and catch-up data is ignored.

The UPDATE STATISTICS feature is enabled by default, and there are no documented ONCONFIG parameters to switch this feature off. This feature performs these actions:

- It leverages the sorted data produced by create index.
- Each sort stream creates mini-distribution bins.
- It ships the mini-bin by using a queue to a mini-bin collector thread.
- The mini-bin collector thread sorts mini-bins.
- It merges the mini-bins into a final distribution bin.

The feature is disabled when the:

- Lead of the index is a User-defined type (UDT), built-in or non-built-in, because this forces a top-down index build.
- The index is of type Functional.
- The index is a Virtual Index Interface (VII).
- There are fewer than two rows in the table.

If the feature is disabled when the UPDATE STATISTICS is executed, certain system catalogs are updated:

- CREATE INDEX of type B-tree, Functional, or VII index force.
- UPDATE STATISTICS LOW equivalent information to be updated in the following system catalogs:
  - Systables
  - Sysfragments
  - Sysindexes
  - Syscolumns

The following SQL statements create auto-distributions and statistics:

```
CREATE INDEX idx_1 ON foo (col1);
ALTER FRAGMENT FOR TABLE foo INIT ….
```

```
ALTER FRAGMENT FOR INDEX idx_1 INIT …
ALTER TABLE ADD UNIQUE CONSTRAINT …
```

6.3.2 Improved sampling size

The UPDATE STATISTICS MEDIUM syntax is enhanced to support a user-configured sampling size.

Here is a list of the enhancements to UPDATE STATISTICS MEDIUM:

- UPDATE STATISTICS MEDIUM SAMPLING SIZE <*number*> number <= 1.0 is interpreted as a percent of the number of rows in the table to be sampled. A number > 1.0 is interpreted as the number of rows to be sampled.
- The SAMPLING SIZE configuration is stored in a new sysdistrib column called *smplsize*.
- The user sampling size might be above the preset sampling size of at resolution of 2.5 and confidence of 80.
- The actual number of rows sampled for UPDATE STATISTICS MEDIUM gets recorded in sysdistrib.rowssmpld.
- SAMPLING is a new keyword and table name SAMPLING cannot be used in the UPDATE STATISTICS statement.

When you use data-distribution statistics for the first time, try to update statistics in MEDIUM mode for all your tables and then update statistics in HIGH mode for all columns that head indexes. This strategy produces statistical query estimates for the columns that you specify. These estimates, on average, have a margin of error less than percent of the total number of rows in the table, where percent is the value that you specify in the RESOLUTION clause in the MEDIUM mode. The default percent value for MEDIUM mode is 2.5 percent. (For columns with HIGH mode distributions, the default resolution is 0.5 percent.)

In Example 6-15, the SAMPLING SIZE 0.7 examines 70% percent of the rows in the table.

Example 6-15 SAMPLING SIZE examining percentage of rows

```
create table test (col1 integer)
Insert 100 rows
UPDATE STATISTICS MEDIUM FOR TABLE test (col1) SAMPLING SIZE 0.7;
```

Example 6-16 on page 186 with the SAMPLING SIZE 40.0 examines 40 rows in the table.

Example 6-16 SAMPLING SIZE examining the number of rows

```
create table test (col1 integer)
Insert 100 rows
UPDATE STATISTICS MEDIUM FOR TABLE test (col1) SAMPLING SIZE 40.0;
```

6.3.3 Improving SET EXPLAIN

To understand the execution path of an SQL query, the SET EXPLAIN command exists. Before you change a query, study its query plan to determine the type and amount of resources it requires. The query plan shows what parallel scans are used, the maximum number of threads required, the indexes used, and so on.

- SET EXPLAIN STATISTICS, undocumented 10.0 feature, is now part of SET EXPLAIN ON by default:
  - Query Statistics are also generated for PDQ-enabled queries.
  - Statistics are aggregated and printed at iterator level.
  - Statistics are only available after the query completes.
- SET EXPLAIN STATISTICS is available for backward compatibility but is superseded by the new SET EXPLAIN ON.
- Query Statistics output is ON:
  - The output can be switched off by setting the new ONCONFIG parameter EXPLAIN_STAT = 0.
  - This is dynamically configurable using onmode –wf and onmode –wm: **`onmode –wf EXPLAIN_STAT=0`**.

The onmode –Y dynamic explain feature is enhanced so that by default when enabled, it prints query statistics information for a session:

- 0 # turn off dynamic explain.
- 1 # turn on dynamic explain, query plan, and statistics.
- 2 # turn on dynamic explain and query plan only.

To dynamically enable set explain for session 32 for both the query plan and statistics, use the command **`onmode -Y 32 1.`**

The SET EXPLAIN FILE TO command now allows users to control where the explain file is generated:

- SET EXPLAIN FILE TO '/work/sqexpl.out';

  Sets explain on and explain file to /work/sqexpl.out.

- SET EXPLAIN FILE TO 'sqex.out'

  Sets explain on and explain file to sqex.out in CWD.

If the file exists, data is appended to the file.

6.3.4 Update statistics improved tracking

Where the update information is stored:

- The time that UPDATE STATISTICS LOW was run is recorded in systables.ustlowts.
- The time that UPDATE STATSTICS MEDIUM or HIGH was executed is stored in date sysdistrib.constr_time.
- User-specified sample size is stored in sysdistrib.smplsize.
- The number of rows sampled during the distribution build is stored in sysdistrib.rowssmpld.

DBExport and DBSchema have been enhanced to dump out sampling size syntax and value for displaying distributions. All of these functions allow a DBA to create SQL queries that can obtain these results and monitor the instance with greater detail. This capability allows a DBA to know exactly when update statistics last executed, and this information can be helpful in solving performance-related issues.

6.3.5 Temp table statistics improvements

Users are no longer required to run UPDATE STATISTICS LOW on temp tables. The number of rows and the number of pages are updated every time we access the temp table data dictionary entry.

Adding indexes to temp tables automatically creates distributions and statistics for the temp table. We retain temp table statistics and distribution information every time that we reset the temp table dictionary information after the Data Definition Language (DDL).

6.4 ONMODE utility

The onmode utility is used for changing the mode of the engine. The various modes that you can change are:

- Change the database server operating mode.
- Force a checkpoint.
- Control the B-tree scanner.
- Change residency of the resident and virtual portions of shared memory.
- Switch the logical-log file.
- Terminate a database server session.
- Add a shared-memory segment to the virtual shared-memory portion.
- Add or remove Virtual Processors.
- Regenerate a .infos file.
- Set decision-support parameters.
- Free unused memory segments.
- Override the WAIT mode of the ONDBSPACEDOWN configuration parameter.
- Enable large chunks and chunk offsets to a maximum size of 4 TB and allow up to 32,766 total chunks.
- Revert data to an earlier database server format. For information about migrating from or reverting to earlier versions of the database server, see the *IBM Informix Migration Guide*, G251-2293.
- Set data replication options.
- Replicate an index with data replication.
- Set SQL statement cache options.
- Dynamically set the value of the SET EXPLAIN statement
- Dynamically update the value of certain connection, PDQ, and memory configuration parameters.

If no options are used when running **onmode**, the database server returns a usage statement.

On UNIX, you must be user root or user *informix* to execute **onmode**. On Windows, you must be a member of the Informix-Admin group. The **onmode** options that we describe have equivalent SQL API commands.

The two new options are **onmode -wm** and **onmode -wf**. They allow the DBA to change certain onconfig parameters while the engine is online. The changes, if desired, can be either only for the duration of the instance being online or written to the onconfig file to be available the next time that the engine is brought online. This gives the DBA the option of controlling the instance without having to restart the instance to change a few of the configuration parameters.

6.4.1 Description of using onmode options

Changing the values of onconfig parameters is a task that the DBA normally performs. Usually the changes are made to the onconfig file associated with the instance. This file is located in $INFORMIXDIR/etc/$ONCONFIG. The file can be modified with any text editor that is installed on the machine. The changes to the file are then saved when the editing session is concluded.

With certain parameters, it is necessary to cycle the informix instance. The newest version of the engine provides a DBA the option of using **onmode** to make the changes to the onconfig file. The two options in **onmode** that allow this to happen are **onmode -wm** and **onmode -wf**. The -wm option only changes the value in memory and not the onconfig file. The -wf option changes both the value in memory and writes the change to the onconfig file.

Example 6-17 shows the list of onconfig parameters on which you can use **onmode -wm** and **onmode -wf**.

Example 6-17 List of config parameters that can be modified with onmode -wm or -wf

```
LISTEN_TIMEOUT
MAX_INCOMPLETE_CONNECTIONS
MAX_PDQPRIORITY
RESIDENT
DS_TOTAL_MEMORY
DS_MAX_QUERIES
DS_MAX_SCANS
DS_NONPDQ_QUERY_MEM
ONLIDX_MAXMEM
USELASTCOMMITTED
IFX_EXTEND_ROLE
LOG_INDEX_BUILDS
VP_MEMORY_CACHE_KB
RTO_SERVER_RESTART
RAS_PLOG_SPEED
RAS_LLOG_SPEED
AUTO_LRU_TUNING
AUTO_CKPTS
AUTO_AIOVPS
INDEX_SELFJOIN
```

```
USE_BATCHEDREAD
USE_KOBATCHEDREAD
TEMPTAB_NOLOG
EXPLAIN_STAT
SORT_MERGE_SIZE
SDS_ENABLE
SDS_TIMEOUT
```

6.4.2 Modifying config parameters with the engine offline

Using a text editor is an easy way to modify the config parameters with the engine offline. The next time that the instance is brought online, any changes to the onconfig file are reflected in the online log. If **oncheck -pr** is executed before the engine has been brought online but after the changes to the onconfig file have been made, the differences are noted in the output of the **oncheck -pr**.

If the number of LOCKS was increased from 2000 to 8000 with the engine offline, the **oncheck -pr** is similar to Example 6-18.

Example 6-18 Results of oncheck -pr with changes made to offline onconfig parameters

```
LOCKS                          2000
ONCONFIG config file error on element LOCKS.
    Value in reserved page:  2000
    Value in config file:    8000
```

With these changes in place to the config parameter, the instance is then brought online. The information in Example 6-19 appears in the online log. After the instance is online, the changes are now applied to the running instance.

Example 6-19 Change to config parameter when instance is offline

```
...
12:55:12  Onconfig parameter LOCKS modified from 2000 to 8000.
...
```

6.4.3 Modifying config parameters with engine online

When the engine is online and the changes are made to the onconfig file by using a text editor and saved, for certain parameters, the engine reflects the changed values. TAPEDEV and LTAPEDEV are two parameters that reflect the latest version of the onconfig values, even if the changes have been made since the instance was brought online.

If TAPEDEV were changed while the instance was online, the **oncheck -pr** output is similar to that in Example 6-20.

Example 6-20 Change to TAPEDEV while instance is online

```
TAPEDEV                         /usr/informix/bkup
ONCONFIG config file error on element TAPEDEV.
    Value in reserved page:  /usr/informix/bkup
    Value in config file:   /dev/null
```

If a backup is now attempted with the changes from Example 6-20 in place, the result is similar to that in Example 6-21.

Example 6-21 Change to TAPEDEV and result to ontape

```
$ ontape -s -L 0
Archive to tape device '/dev/null' is complete.
```

If the instance is taken offline and then back online, the changes are also reflected in the online log as shown in Example 6-22.

Example 6-22 Online log after instance is brought back online

```
...
13:05:22  Onconfig parameter TAPEDEV modified from /usr/informix/bkup to
/dev/null.
...
```

The instance now uses the modified config parameter for TAPEDEV.

Another method to change onconfig parameters has been introduced in the engine with the **onmode -wm** and **onmode -wf** parameters. This allows changes to be made to the configuration parameters with the engine online. The **onmode -wm** option only makes the changes to memory and not to the onconfig file. When the instance is taken offline and then back online, the parameters in the onconfig file are used. To make the changes so that when the engine is taken offline and back online, the changes are still in effect, use **onmode -wf**. These changes are written to memory and the onconfig file.

6.4.4 Examples of using onmode -wm and onmode -wf

For a parameter only to be in effect until the next time that the instance is taken offline, use the **onmode -wm**. Example 6-23 on page 192 shows the **onmode -wm** feature.

Example 6-23 example of onmode -wm

```
onmode -wm RESIDENT=0
onstat -m
...
16:33:05  Value of RESIDENT has been changed to 0.
...
```

The -wf option makes the change to the configuration file at the same time that the value in memory is updated. Example 6-24 shows the **onmode -wf** feature.

Example 6-24 example of onmode -wf

```
onmode -wf RESIDENT=1
onstat -m
...
16:51:35  Value of RESIDENT has been changed to 1.
...
```

If a value passed to either **onmode -wm** or **onmode -wf** is outside of the allowed range for that parameter, an error message prints and the parameter is not changed, as shown in Example 6-25.

Example 6-25 Value outside allowable parameter range

```
onmode -wm RESIDENT=599

13:00:28  Illegal value for RESIDENT, must be between -1 and 99.

onmode: Parameter value out of range.
```

7

The administrative API

Allowing a DBA more capability to maintain instances remotely is a goal of IDS 11. In previous versions of IDS, monitoring multiple instances required multiple active connections. A single connection to an instance can now permit a DBA to monitor and maintain multiple instances. The instances can be local, on the same machine as the DBA, or in remote locations.

7.1 The sysadmin database

The sysadmin database is created in new instances by default in IDS 11. If upgrading to Version 11 from a previous version of IDS, the sysadmin database is created automatically as part of the conversion process. To verify that the database is built successfully, check the online log when performing the upgrade.

There are two major user-defined routines (UDRs) that are introduced with the sysadmin database. These routines, task() and admin(), allow the DBA to perform administrative work through SQL or stored procedures.

There are nineteen tables that are part of the sysadmin database. The list of the table names is given in Example 7-1.

Example 7-1 List of table names in sysadmin database

```
command_history
mon_table_names
ph_task
mon_checkpoint
mon_table_profile
ph_threshold
mon_config
mon_users
ph_version
mon_memory_system
mon_vps
mon_onconfig
ph_alert
mon_prof
ph_alerts
mon_profile
ph_group
mon_sysenv
ph_run
```

Important: The only default user that can connect to the sysadmin database is user *informix*.

The table command_history keeps track of the information from the task() and admin() functions. The UDRs automatically populate the command_history table with their results. No configuration changes are necessary for this to happen.

Important: Do not drop or attempt to alter the sysadmin database, because it is used by several other database server components.

7.1.1 Database administration in previous IDS versions

In previous versions of IDS, administration of the database instance was usually accomplished through a command-line interface. Another option was ISA, which was Web-based. With the command-line interface, it was easy to administer a local instance, one that was on the same machine or one that an administrator can log on to, but difficult for any remote instance management. To administer to different instances, it was necessary to physically connect to the machine containing the instance.

This made it very difficult for a DBA to administer remote databases. The DBA had to open many windows to perform the required administrative tasks. Many of the same administrative tasks also needed to be run on multiple instances. Several of the administrative tasks needed to be run on a schedule, minimizing the impact of the tasks on the general users on the instance. The DBA had to keep track of the number of times that a command has been executed, the results that might be returned from certain administrative commands, and also when the commands had been executed.

Figure 7-1 on page 196 is a graphical example of the desired work environment for a DBA. Having the ability to write SQL or stored procedures that can be hardware independent gives the DBA much greater flexibility to manage an instance. Through one dbaccess window, a DBA can now manage all of the Version 11 IDS instances. And that makes the DBA's job easier.

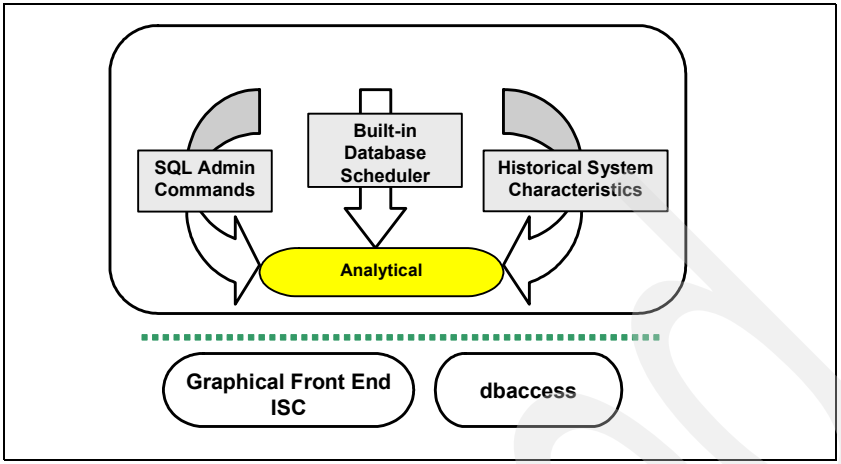


Figure 7-1 Overview of DBA system

7.1.2 Remote administration of databases

SQL can be executed across different databases and instances. Because the sysadmin database is a database, other instances with the proper connect privileges can connect to this database. The commands that are executed against the database are SQL, so remote administration can be accomplished quite easily. Example 7-2 shows the necessary steps for remote administration.

Example 7-2 Remote connection of database

```
connect to sysadmin@testremote;
execute function task (“check extents”);
```

7.1.3 Examples of task() and admin() functions

To check the extents of the instance, similar to **`oncheck -ce`**, see Example 7-3 for the necessary SQL.

Example 7-3 Using task() to simulate oncheck -ce

```
database sysadmin;
execute function task ("check extents");
```

Execution of the commands in Example 7-3 on page 196 results in Example 7-4.

Example 7-4 Results of executing task() example

```
$ dbaccess - sample_task

Database selected.

(expression)  Validating extents for Space 'rootdbs' ...
              Validation of extents for Space 'rootdbs' succeeded

1 row(s) retrieved.

Database closed.
```

The admin() function performs the same as the task() function, but the results returned are slightly different. The change to the SQL script is shown in Example 7-5.

Example 7-5 Using admin() to simulate oncheck -ce

```
database sysadmin;
execute function admin ("check extents");
```

The results of executing the script in Example 7-5 are shown in Example 7-6.

Example 7-6 Results of executing admin() example

```
$ dbaccess - sample_admin

Database selected.

(expression)

         101

1 row(s) retrieved.

Database closed.
```

A table with onmode command line options and the resulting syntax to include in the API to get an equivalent result is shown in Table 7-1 on page 198. A more complete list is shown in Table 7-4 on page 212. The list of commands that the API can execute is shown in Table 7-3 on page 208.

Table 7-1 onmode equivalent options

onmode option	API parameter
onmode -a	ADD MEMORY
onmode -c	CHECKPOINT
onmode -k	SHUTDOWN
onmode -ku	SHUTDOWN { IMMEDIATE }
onmode -s	QUIESCENT
onmode -su	QUIESCENT { IMMEDIATE }

The result of the admin() function is an integer value; the result of the task() function is a character string. The integer value that admin() returns is a link to the command_history table.

To determine the result of the admin() function, a query against the command_history table is required as depicted in Example 7-7. The cmd_number changes to match the integer value returned from the specific admin() function.

Example 7-7 Query against the command_history table

```
database sysadmin;
select * from command_history
where cmd_number=101;
```

The information contained in the command_history table for the cmd_number is returned as depicted in Example 7-8.

Example 7-8 Query results from command_history table

```
$ dbaccess - sample_command_history

Database selected.

cmd_number      101
cmd_exec_time   2007-05-17 14:03:01
cmd_user        informix
cmd_hostname    odgreen
cmd_executed    check extents
cmd_ret_status  0
cmd_ret_msg     Validating extents for Space 'rootdbs' ...
                Validation of extents for Space 'rootdbs' succeeded

1 row(s) retrieved.
```

```
Database closed.
```

7.2 The Scheduler

The Scheduler is a new feature that is intended to allow the DBA the ability to schedule when a SQL, Stored Procedure, or UDR can be executed.

This gives the DBA more flexibility when running administrative tasks. After a task has been created, it can be executed on different platforms. Because the tasks are a form of instruction recognized by the instance, the same task can be executed on different hardware platforms.

The Scheduler manages and executes scheduled maintenance, monitoring, and administrative tasks. This tool enables you to monitor activities (for example, space management or automatically backing up any new log data at timed intervals since the last log backup) and create corrective actions that run automatically.

The Scheduler manages:

- Tasks, which provide the means for running a specific job at a specific time or interval
- Sensors, which collect and save information
- Startup tasks, which run only once when the database server starts
- Startup sensors, which run only once when the database starts

A set of task properties, which define what needs to be collected or executed, control the Scheduler. The task properties are stored in the ph_task table in the sysadmin database. Each row in this table is a separate task, and each column is a task property. The *task properties* indicate to the system when to run an SQL statement, stored procedure, or function and how to handle the task. For example, you can define tasks to check free log space every hour from 9:00:00 to 19:00:00 daily.

Only task properties, not configuration parameters, define what the Scheduler collects and executes. The Scheduler executes tasks at predefined times or as determined internally as required by the database server.

7.2.1 Tasks

A *task* is an operation that can be performed as often as needed. The creation of a task is demonstrated in Example 7-9.

This task is created to monitor the amount of data in the command_history table and is executed once a day at 2 AM. The table ph_task is where the tasks for a given instance are stored. Because it is a table, if the task no longer is needed, it can be deleted from the ph_task table.

Example 7-9 Creation of a task

```
INSERT INTO ph_task
( tk_name, tk_type, tk_group, tk_description, tk_execute,
tk_start_time, tk_stop_time, tk_frequency )
VALUES
("mon_command_history",
"TASK",
"TABLES",
"Monitor how much data is kept in the command history table",
"delete from command_history where cmd_exec_time < (
        select current - value::INTERVAL DAY to SECOND
        from ph_threshold
        where name = 'COMMAND HISTORY RETENTION' ) ",
DATETIME(02:00:00) HOUR TO SECOND,
NULL,
INTERVAL ( 1 ) DAY TO DAY);
```

7.2.2 Sensors

A *sensor* is created to be executed as a simple way of collecting information. Again, the ph_task table is used to create a sensor. A sample sensor is created in Example 7-10.

Example 7-10 Creation of a sensor

```
INSERT INTO ph_task
(tk_name, tk_type, tk_group, tk_description, tk_result_table,
tk_create,tk_execute, tk_stop_time, tk_start_time, tk_frequency, tk_delete )
VALUES
("mon_memory_system",
"SENSOR",
"MEMORY",
"Server memory consumption",
"mon_memory_system",
"create table mon_memory_system (ID integer, class smallint, size int8, used
int8, free int8 )",
```

```
"insert into mon_memory_system select $DATA_SEQ_ID, seg_class, seg_size,
seg_blkused, seg_blkfree FROM sysmaster:sysseglst",
NULL,
NULL,
INTERVAL ( 30 ) MINUTE TO MINUTE,
INTERVAL ( 30 ) DAY TO DAY);
```

The tk_type is what determines if a entry into the ph_task table is a sensor or task.

This sensor in Example 7-10 on page 200 is used to monitor the amount of memory that is used on a system, and that information is stored in a table mon_memory_system. If the table does not exist, the sensor creates it. The information is collected every 30 minutes.

Several sensors have been created and ship with IDS 11. Table 7-2 shows the name of the sensors and the description of the work that the sensor performs.

Table 7-2 Sensors shipped with IDS 11

Sensor	Description
mon_command_history	Purges the command history table.
mon_config	Saves any difference in the onconfig file.
mon_config_startup	Save the onconfig file on every server startup.
mon_profile	Save the server profile information.
mon_vps	Collects the Virtual Processor timings.
mon_checkpoint	Save information about checkpoints.
mon_table_profile	Save table profile information including UDI counters.
mon_table_names	Save the table names along with their creation time.
mon_users	Save profile information about each user.
check_backup	Check to ensure backups have been done.

Refer to the *IBM Informix Administration Guide*, G251-2267, for more details about the sensors that we described.

7.2.3 Startup tasks

A startup task is performed each time that the instance is started and checks the extents, such as oncheck -ce. The syntax to create this startup task is in shown in Example 7-11.

Example 7-11 SQL to create startup task

```
INSERT INTO ph_task
( tk_name, tk_type, tk_group, tk_description, tk_execute,
tk_start_time, tk_stop_time, tk_frequency )
VALUES
("mon_disk_history",
"STARTUP TASK",
"TABLES",
"Monitor data via the oncheck -ce command",
"execute function task('check extents') ",
NULL,
NULL,
INTERVAL ( 1 ) DAY TO DAY);
```

7.2.4 Startup sensors

A startup sensor is executed when the instance is started. A startup sensor that tracks the environment is detailed in Example 7-12. The results of the execution of the sensor are stored in a table called mon_sysenv.

Example 7-12 SQL to create a startup sensor

```
INSERT INTO ph_task (
tk_name,
tk_type,
tk_group,
tk_description,
tk_result_table,
tk_create,
tk_execute,
tk_stop_time,
tk_start_time,
tk_frequency,
tk_delete )

VALUES (
"mon_sysenv",
"STARTUP SENSOR",
"SERVER",
"Tracks the database servers startup environment.",
```

```
"mon_sysenv",
"create table mon_sysenv (ID integer, name varchar(250), value
lvarchar(1024))",
"insert into mon_sysenv select $DATA_SEQ_ID, env_name, env_value FROM
sysmaster:s ysenv",
NULL,
NULL,
"0 0:01:00",
"60 0:00:00" );
```

The values returned from ph_task after creating the example startup sensor are shown in Example 7-13. The tk_type is STARTUP SENSOR. This startup sensor executes every time that the instance is started. The resulting output is stored in the table mon_sysenv.

Example 7-13 Values in table ph_task for startup sensor

```
tk_id 7
tk_name mon_sysenv
tk_description Tracks the database servers startup environment.
tk_type STARTUP SENSOR
tk_executing_sid 0
tk_sequence 0
tk_result_table mon_sysenv
tk_create create table mon_sysenv (ID integer, name varchar(250), value
lvarchar(1024))
tk_execute insert into mon_sysenv select $DATA_SEQ_ID, env_name, env_value FROM
sysmaster:sysenv
tk_delete 60 00:00:00
tk_start_time
tk_stop_time
tk_frequency 0 00:01:00
tk_next_execution 2006-07-24 17:09:50
tk_total_executio+ 0
tk_total_time 0.00
tk_monday t
tk_tuesday t
tk_wednesday t
tk_thursday t
tk_friday t
tk_saturday t
tk_sunday t
tk_attributes 0
tk_group SERVER
tk_enable t
tk_priority 0
```

7.2.5 Examples of task and sensor

The startup task created in Example 7-11 on page 202 is executed on startup of the instance. The task updates a table called mon_diskhistory. To examine the data in that table, use the sample SQL shown in Example 7-14.

Example 7-14 Query of ph_task

```
database sysadmin;
select * from ph_task where tk_name='mon_disk_history';
```

The information returned by executing the commands in Example 7-14 is shown in Example 7-15.

Example 7-15 Output of mon_diskhistory in ph_task

```
tk_id               15
tk_name             mon_disk_history
tk_description      Monitor how much data is kept in the oncheck -ce
tk_type             STARTUP TASK
tk_sequence         1
tk_result_table
tk_create
tk_dbs              sysadmin
tk_execute          execute function task('check extents')
tk_delete            0 01:00:00
tk_start_time
tk_stop_time
tk_frequency          1 00:00:00
tk_next_execution   2007-06-02 19:00:09
tk_total_executio+  1
tk_total_time       0.103155
tk_monday           t
tk_tuesday          t
tk_wednesday        t
tk_thursday         t
tk_friday           t
tk_saturday         t
tk_sunday           t
tk_attributes       0
tk_group            TABLES
tk_enable           t
tk_priority         0
```

This shows the task has been created successfully. This task does not create a result table, but if desired, the DBA can modify the task to create a result table and then store the results of the command being executed. There is a table

ph_run shown in Example 7-16. The run_retcode column contains 0, which indicates the startup task executed successfully.

Example 7-16 Ph_run table containing result of startup task

```
run_id        205
run_task_id   15
run_task_seq  1
run_retcode   0
run_time      2007-06-01 18:59:17
run_duration  0.103155
run_ztime     1180734134
run_btime     1180734134
run_mttime    1180742357
```

The startup sensor that is created by Example 7-13 on page 203 is executed upon startup of the instance. To examine the information that is collected by the sensor, the query in Example 7-17 is constructed, and the output from the query is shown in Example 7-18.

Example 7-17 Query of mon_sysenvr

```
database sysadmin;
select * from mon_sysenv;
```

The sample output in Example 7-18 shows that each time that the instance is started, the column named `id` is incremented by one.

Example 7-18 Sample output from table mon_sysenv

```
id     1
name   DBDELIMITER
value  |

id     1
name   DBPATH
value  .

----repeated for each environment variable----

id     2
name   DBDELIMITER
value  |

id     2
name   DBPATH
value  .
```

```
----repeated for each environment variable---
```

7.3 Command line interface for database administration

The command line versions of onspaces, onparams, and other utilities supplied with IDS 11 still work. What this feature can provide to a DBA is the ability to run the same command on different machines. The SQL is machine independent, so if the DBA wants to add a chunk using the same path on every machine that the DBA administers, this feature allows that SQL to be developed one time and used many times, while running the command through the same window.

7.3.1 Database administration by using the command line

To find options with onspaces, the command **onspaces --** lists the available options from that version of the product. This is depicted in Example 7-19. To execute the **onspaces** command from the command line, it is necessary to run the command on the machine on which the instance is located. This can be a limiting factor, because a DBA might control instances that are on many different machines. Through shell scripting or batch files, the DBA can run similar commands on different machines. This requires the DBA to maintain an active knowledge of scripting languages.

Example 7-19 Output of onspaces --

```
$ onspaces --
Usage:
 onspaces { -a <spacename> -p <path> -o <offset> -s <size> [-m <path> <offset>]
                { { [-Mo <mdoffset>] [-Ms <mdsize>] } | -U }
                } |

          { -c { -d <DBspace> [-k <pagesize>] [-t]
                    -p <path> -o <offset> -s <size> [-m <path> <offset>] } |
                { -d <DBspace> [-k <pagesize>]
                    -p <path> -o <offset> -s <size> [-m <path> <offset>]
                    [-ef <first_extent_size>] [-en <next_extent_size>] } |
                { -b <BLOBspace> -g <pagesize>
                    -p <path> -o <offset> -s <size> [-m <path> <offset>] } |
                { -S <SBLOBspace> [-t]
                    -p <path> -o <offset> -s <size> [-m <path> <offset>]
                    [-Mo <mdoffset>] [-Ms <mdsize>] [-Df <default-list>] } |
                { -x <Extspace> -l <Location> } } |

          { -d <spacename> [-p <path> -o <offset>] [-f] [-y] } |
```

```
        { -f[y] off [<DBspace-list>] | on [<DBspace-list>] } |

        { -m <spacename> {-p <path> -o <offset> -m <path> <offset> [-y] |
                         -f <filename>} } |

        { -r <spacename> [-y] } |

        { -s <spacename> -p <path> -o <offset> {-O | -D} [-y] } |

        { -ch <sbspacename> -Df <default-list> } |

        { -cl <sbspacename> } |

        { -ren <spacename> -n <newname> }

    -a  - Add a chunk to a DBspace, BLOBspace or SBLOBspace
    -c  - Create a DBspace, BLOBspace, SBLOBspace or Extspace
    -d  - Drop a DBspace, BLOBspace, SBLOBspace, Extspace, or chunk
    -f  - Change dataskip default for specified DBspaces
    -m  - Add mirroring to an existing DBspace, BLOBspace or SBLOBspace
    -r  - Turn mirroring off for a DBspace, BLOBspace or SBLOBspace
    -s  - Change the status of a chunk
    -ch - Change default list for smart large object space
    -cl - garbage collect smart large objects that are not referenced
    default-list = {[LOGGING = {ON|OFF}]  [,ACCESSTIME = {ON|OFF}]
            [,AVG_LO_SIZE = {1 - 2097152}] }
    -ren - Rename a DBspace, BLOBspace, SBLOBspace or Extspace
```

Most of the other utilities supplied with IDS 11 have the same general feature. If supplied with two dashes, they print the available options for that utility.

Executing these commands requires a connection to the machine and either the ability to open a command window or run remote shell commands. An extensive knowledge of each operating system is required to understand how to execute the commands on each platform.

You use the command shown in Example 7-20 to add a logical log to an existing system in the root dbspace using the default size of the logical log.

Example 7-20 Using onparams to add a logical log to the root dbspace

```
$ onparams -a -d rootdbs -i
Log operation started. To monitor progress, use the onstat -l command.
Logical log successfully added.
```

7.3.2 Database administration by using SQL and stored procedures

Having the ability to administer a database instance through SQL or stored procedures makes a DBA's job easier. Because SQL or stored procedures can be executed in dbaccess to remote instances, a DBA can administer many instances from a single command line window.

The SQL API allows the DBA to create the SQL or stored procedure to administer the instance. A list of the API commands, their syntax, and the effect of the command is shown in Table 7-3.

Table 7-3 List of SQL API commands

Administrative API command	Effect of the command	Utility or configuration parameter that produces a similar result
ADD BUFFERPOOL	Creates a new bufferpool.	onparams -b or BUFFERPOOL configuration parameter
ADD CHUNK	Adds a chunk to a dbspace or blobspace.	onspaces -a space
ADD LOG	Adds a log to a dbspace.	onparams -a -d
ADD MEMORY	Adds memory to the virtual portion of shared memory.	onmode -a or SHMADD configuration parameter
ADD MIRROR	Adds a mirror chunk to a dbspace.	onspaces -m
ALTER CHUNK OFFLINE	Changes the status of a chunk from online to offline.	onspaces -s
ALTER CHUNK ONLINE	Changes the status of a chunk from offline to online.	onspaces -s
ALTER LOGMODE	Changes the logging mode of the database to unbuffered, buffered, ANSI, or non-logging. (But unlike ondblog or ontape, the database remains accessible; and no L0 backup is required.) This command fails if any other session is active.	ondblog or ontape

Administrative API command	Effect of the command	Utility or configuration parameter that produces a similar result
ALTER PLOG.	Changes the size of the physical log.	onparams
ARCHIVE FAKE	Archives a fake backup operation.	ontape -F
CHECK DATA	Checks the data portion of a table or the part number of a table or fragment.	oncheck -cd dbname:table
CHECK EXTENTS	Checks dbspace extents.	oncheck -ce
CHECK PARTITION	Checks the data portion of the table.	oncheck -ct dbname:table
CHECKPOINT	Forces a checkpoint.	onmode -c
CLEAN SBSPACE	Releases from the sbspace any unreferenced BLOB or CLOB objects.	onspaces -cl sbspace
CREATE BLOBSPACE	Creates a blobspace.	onspaces -c -b blobspace
CREATE CHUNK	Adds a chunk to a dbspace or to a blobspace.	onspaces -a space
CREATE DBSPACE	Creates a dbspace.	onspaces -c -d dbspace
CREATE SBSPACE	Creates an sbspace.	onspaces -c -S sbspace
CREATE TEMPDBSPACE	Creates a temporary dbspace.	onspaces -c -d -t dbspace
CREATE BLOBSPACE	Creates a blobspace.	onspaces -c -b blobspace
DROP BLOBSPACE	Drops a specified blobspace.	onspaces -d blobspace
DROP CHUNK	Drops a chunk in a specified dbspace, blobspace, or sbspace.	onspaces -d space
DROP DBSPACE	Drops a specified dbspace.	onspaces -d dbspace
DROP LOG	Drops a specified logical log.	onparams -d -l dbspace
DROP SBSPACE	Drops a specified sbspace.	onspaces -d sbspace

Administrative API command	Effect of the command	Utility or configuration parameter that produces a similar result
DROP TEMPDBSPACE	Drops a temporary dbspace.	onspaces -d dbspace
PRINT ERROR	Prints the error message associated with an error number.	finderr err_number
PRINT PARTITION	Prints the partition headers of the table.	oncheck -pt table
QUIESCENT	Gracefully puts the database server into quiescent mode.	onmode -s
QUIESCENT IMMEDIATE	Immediately puts the database server into quiescent mode.	onmode -su
RENAME SPACE	Renames a dbspace, blobspace, sbspace, or extspace.	onspaces -ren space -n new_name
SET CHUNK OFFLINE	Changes the status of a mirrored dbspace, blobspace, or sbspace to offline.	onspaces -s space -p pathname -o offset-D
SET CHUNK ONLINE	Changes the status of a mirrored dbspace, blobspace, or sbspace to online.	onspaces -s space -p pathname -o offset-O
SET DATASKIP ON	Specifies that the database server can skip designated dbspaces.	DATASKIP configuration parameter or SQL statement
SET DATASKIP OFF	Disables the DATASKIP setting.	DATASKIP configuration parameter or SQL statement
SET SBSPACE ACCESSTIME ON	Tracks the time of access to all smart large objects stored in the sbspace.	onspaces -Df

Administrative API command	Effect of the command	Utility or configuration parameter that produces a similar result
SET SBSPACE ACCESSTIME OFF	Disables the tracking of the time of access to all smart large objects stored in the sbspace.	onspaces -Df
SET SBSPACE AVG_LO_SIZE	Specifies an expected average sbspace size.	onspaces -Df
SET SBSPACE LOGGING ON	Specifies that the database server logs changes to the user data area of the sbspace.	onspaces -Df
SET SBSPACE LOGGING OFF	Specifies that the database server does not log changes to the user data area of the sbspace.	onspaces -Df
SET SQL TRACING	Enables global SQL tracing with default values.	No comparable utility commands.
SET SQL TRACING OFF	Disables global SQL tracing.	No comparable utility commands.
SET SQL TRACING ON	Sets global SQL tracing to ON.	No comparable utility commands.
SET SQL TRACING RESIZE	Changes the size of the buffers used for SQL tracing.	No comparable utility commands.
SET SQL USER TRACING	Enables SQL tracing for a user for the current session, even if global SQL tracing is disabled.	No comparable utility commands.
SET SQL USER TRACING CLEAR	Clears SQL tracing flags for a user, so the global SQL tracing setting applies to the user.	No comparable utility commands.
SET SQL USER TRACING OFF	Disables SQL tracing for a user for the current session, even if global SQL tracing is enabled.	No comparable utility commands

Administrative API command	Effect of the command	Utility or configuration parameter that produces a similar result
SHUTDOWN	Gracefully shuts down the database server.	onmode -k
SHUTDOWN IMMEDIATE	Shuts down the database server, without removing shared memory segments or doing a checkpoint.	onmode -ku
START MIRRORING space	Starts mirroring the specified dbspace, blobspace, or sbspace.	onspaces -m
STOP MIRRORING	Stops mirroring the specified dbspace.	onspaces -r

The SQL API allows a DBA to pass the following options to the **onmode** command as shown in Table 7-4.

Table 7-4 The onmode options in SQL API

API call format	Effect of command
'ONMODE', 'a', '<*kilobytes*>'	Add memory.
'ONMODE', 'BC', '{ 1 | 2 }'	Change large chunk mode.
'ONMODE', 'c', '{ block | unblock }'	Do a checkpoint, either blocking or unblocking the server.
'ONMODE', 'C'	Tune B-tree scanner.
'ONMODE', 'd'	HDR.
'ONMODE', 'D'	Set PDQ.
'ONMODE', 'e', '<*keyword*>'	Configure or flush the shared statement cache.
'ONMODE', 'F'	Free unused memory.
'ONMODE', 'j'	Switch to single-user mode.
'ONMODE', 'k'	Shutdown gracefully.
'ONMODE', 'k'u	Shutdown immediately.
'ONMODE', 'l'	Switch to next logical log.

API call format	Effect of command
'ONMODE', 'm'	Switch to multi-user mode.
'ONMODE', 'M', '<*kilobytes*>'	Decision support memory.
'ONMODE', 'n'	Unlock resident memory.
'ONMODE', 'O'	Override space down blocking a checkpoint.
'ONMODE', 'p', '{ + | - | # }', '<*class*>'	Add or remove a Virtual Processor of a specified class.
'ONMODE', 'Q'	Set a maximum number for decision-support queries.
'ONMODE', 'r'	Lock memory resident.
'ONMODE', 'R'	Rebuild .infos file.
'ONMODE', 's'	Switch to quiescent mode gracefully.
'ONMODE', 'su'	Switch to quiescent mode immediately.
'ONMODE', 'S'	Specify a maximum number of decision support scans.
'ONMODE', 'W'	Reset statement cache attributes.
'ONMODE', 'wf', '<*parameter = value*>'	Update a value in the configuration file.
'ONMODE', wm', '<*parameter = value*>'	Update onconfig only in memory.
'ONMODE', 'Y' '{ 0 | 1 }'	Set dynamic explain on or off.
'ONMODE', 'z' , '<*session_id*>'	Terminate a user session.
'ONMODE', 'Z', '<*address*>'	Heuristically complete a transaction.

7.3.3 Syntax examples

The SQL to execute PRINT ERROR is shown in Example 7-21. This is equivalent to running the **`finderr 101`** command.

Example 7-21 API call to return information about error number 101

```
database sysadmin;
execute function task ("PRINT ERROR", "101");
```

The result of the PRINT ERROR command is shown in Example 7-22 on page 214.

Example 7-22 Result of PRINT ERROR API call

```
Database selected.
(expression)  ISAM error:  file is not open.
1 row(s) retrieved.
Database closed.
```

To confirm that the command executed properly, see the result in Example 7-23.

Example 7-23 Command line finderr

```
$ finderr 101
-101    ISAM error: file is not open.

The program attempted to use an unopened file, table, partition,
tablespace, or other storage object, or one of these whose access
mode did not support the requested operation (for example, an
attempt to write to a file that was opened in read-only mode).

If the error recurs, refer to the information on trapping errors in
your Administrator's Guide or Reference for additional diagnostics.
Contact Technical Support at tsmail@us.ibm.com with the diagnostic information.
```

To see how **onmode -c** works with two different calls through the SQL API, see Example 7-24.

Example 7-24 Forcing checkpoint through API

```
database sysadmin;
execute function task('CHECKPOINT');
execute function task('ONMODE', 'c');
```

The output from the API commands in Example 7-24 is the same as evidenced in Example 7-25.

Example 7-25 Result of running API

```
$ dbaccess - onmode
Your evaluation license will expire on 2007-11-06 23:00:00
Database selected.
(expression)  Checkpoint Completed
1 row(s) retrieved.
(expression)  Checkpoint Completed
1 row(s) retrieved.
Database closed.
```

The ability to execute the commands through either a command-line interface and then through SQL or stored procedures allows a DBA to create many methods to monitor instances. The SQL API commands allow the DBA to have greater flexibility in this aspect of their job.

8

SQL Query Drill-Down

In the world of databases, everyone understands the importance of a good performing database server to the success of any application. Yet, many times the databases are not optimally configured to take full advantage of the system resources. When performance seems to lag, organizations tend to look at costly hardware enhancements, such as adding more processors or more RAM as a quick fix remedy. However, many times they might find a less costly and better solution by doing performance troubleshooting.

Troubleshooting the performance of a database server can seem to be a daunting task, because of the lack of tools to pinpoint the source of problems. For a DBA, it is very important to be able to identify which application or user is consuming most of the database resources, such as memory, disk I/O, CPU, and locks. Using this information, DBAs can analyze the system performance, make changes in the system configuration, and make suggestions to developers for changing the application logic to enable improvement in the performance of the database system.

In this chapter, we discuss ways to help you identify the performance bottlenecks. This is primarily by using the SQL Query Drill-Down feature in IDS 11 to gather statistical information about each SQL statement executed on the system and how to view and analyze statement history. This is accomplished by using the new SQLTRACE configuration parameter to gather performance information. In the next section, we provide more detail about how this is done.

8.1 Understanding the SQL Query Drill-Down feature

A typical DBA pain point is the lack of having proper tools to identify performance bottlenecks in SQL statements to enable corrective actions to be taken. Prior to IDS 11, the I-SPY product was the only tool available to do query monitoring. But it is more geared toward the data warehouse environment, and it is a separate, standalone product that requires installation and configuration steps to work with IDS. The SET EXPLAIN option, which is a built-in feature, helps to a certain extent, but by the time it is turned on, the problem might have already occurred.

These solutions lack simplicity in implementation and depth and timeliness in statistics collection. The new SQL Query Drill-Down feature was developed to address these shortcomings and give more control to DBAs in monitoring the system usage and adjusting the system configuration accordingly.

The SQL Query Drill-Down feature helps you answer questions such as:

- How long do SQL statements take?
- How many resources are individual statements using?
- How long did statement execution take?
- How much time was involved in waiting for each resource?
- What was the query plan?

Information is gathered by turning on the SQLTRACE parameter. That information is then stored in a circular buffer, which is an in-memory table called *syssqltrace*. It is actually a pseudo table that appears to the user as a regular table on disk. However, the data is not written to disk as a regular table, but is maintained in memory while the database is online. This pseudo table, syssqltrace, can be accessed from the *sysmaster* database.

Because the information is stored in system shared memory, you need to be cautious while configuring the sqltrace buffer size and the number of traces. Improper settings can result in poor utilization of the shared memory. By multiplying the number of buffers times the number of traces, you can determine the total memory usage for the sqltrace buffer. For example, the SQLTRACE settings illustrated in Example 8-1 on page 221 consume 4 MB (2000 x 2 KB) of shared memory space. If you only use a percentage of the allocated individual buffer space, the amount of memory that is allocated for each buffer is still two KB (assuming the size is set to 2 KB). It is a good practice to tune these values as needed to optimize the usage of the memory buffers.

By default this feature is turned off, but you can turn it on for all users or for a specific set of users. When this feature is enabled with its default configuration, the database server tracks the last 1,000 SQL statements that ran, along with the profile statistics for those statements.

You can enable and disable the tracing at any point in time, and you can change the number and size of the trace buffers while the database server is running. If you resize the trace buffer, the database server attempts to maintain the content of the buffer. If the parameters are increased, data is not truncated. However, if the number or the size of the buffers is reduced, the data in the trace buffers might be truncated or lost.

The number of buffers determines how many SQL statements are traced. If that number is exceeded, the oldest traces make way for new incoming traces. Each buffer contains the information for a single SQL statement. An individual trace buffer size is determined by the SQLTRACE parameter called *size,* which can be set at the server startup time and set or changed using the administrative API functions task() or admin() dynamically while server is online. If the text information that is stored in the buffer exceeds the size of the trace buffer currently set, the data is truncated to fit in the currently allocated buffer.

So far we have discussed briefly what the SQL Query Drill-Down feature is and what it does. In the next sections, we discuss how to use this feature, how to configure it based on your specific needs, and how to view and analyze the information in detail.

There are two ways to enable and configure the SQL trace. The first way is to configure it by setting SQLTRACE in the $INFORMIXDIR/etc/$ONCONFIG file. The second way is by calling the administrative API function task() or admin(). These options are explained further in 8.1.1, “SQLTRACE” on page 219 and 8.1.2, “SQL trace using the administrative API functions” on page 222 along with number of examples.

8.1.1 SQLTRACE

Use the SQLTRACE configuration parameter to enable and control the default tracing behavior when the database server starts. The information that you set includes the number of SQL statements to trace, tracing mode, trace level, and trace size.

Any user who can modify the $INFORMIXDIR/etc/$ONCONFIG file can modify the value of the SQLTRACE configuration parameter and affect the startup configuration.

This is the syntax that can be specified with SQLTRACE:

```
SQLTRACE [Level=off|low|med|high],[Ntraces=number of traces],
[Size=size of each trace buffer],[Mode=global|user]
```

The range of parameters and their values are provided in the following list:

- Level: Determines the amount of information traced:
  - Off: This specifies no SQL tracing occurs and is the default.
  - Low: At this level, we capture statement statistics, statement text, and statement iterators.
  - Medium: With this level, we capture all of the information included in low-level tracing, plus table names, the database name, and stored procedure stacks.
  - High: Here, all of the information included in medium-level tracing is captured, plus host variables.
- Ntraces: The number of SQL statements to trace before reusing the resources. The range is from 500 to 2147483647.
- Size: This specifies the number of kilobytes for the size of the trace buffer. If this buffer size is exceeded, the database server discards saved data. The range is 1 KB to 100 KB.
- Mode: Specifies the type of tracing performed:
  - Global: This is for all users on the system.
  - User: Use this for users who have tracing enabled by an Administration API task() function. Specify this if you want to get a sample of the SQL that a small set of users is running.

Enabling tracing in a global mode

There are two modes of SQL tracing: global and user. In global mode, the tracing is enabled for all the sessions that are running on the system. The user mode enables tracing only for the users that you have asked the system to trace. First, let us look at how to set the global mode, which is also the default mode.

You might need global tracing when you want to do a quick comparative analysis on resource usage of all the sessions that are running. Global tracing is also useful when you are unsure of which specific user or users to trace. After you identify those sessions, you can narrow the tracing to session level. If there are many users on the system who are running many SQL applications the tracing buffers can quickly be filled. If the number of traces exceeds your setting, you can lose trace information. Therefore, make your trace setting large enough.

Example 8-1 on page 221 specifies that the database server to gather low-level information about up to 2000 SQL statements executed by all users on the system and allocates approximately four MB of memory (2000 x 2 KB).

Example 8-1 SQLTRACE global

```
SQLTRACE level=LOW,ntraces=2000,size=2,mode=global
```

When the IDS engine is brought up with SQLTRACE turned on, the database server prints information to that effect in the message log. Example 8-2 shows the portion of *online.log*. The last line in the example is what you see when you bring up the IDS engine with SQLTRACE settings as shown in Example 8-1.

Example 8-2 Online.log excerpt showing SQLTRAACE information

```
Mon May 21 20:36:52 2007

20:36:52  Event alarms enabled.  ALARMPROG =
'/usr/informix/etc/alarmprogram.sh'
20:36:52  Booting Language <c> from module <>
20:36:52  Loading Module <CNULL>
20:36:52  Booting Language <builtin> from module <>
20:36:52  Loading Module <BUILTINNULL>
20:36:52  Dynamically allocated new virtual shared memory segment (size 8192KB)
20:36:52  Memory sizes:resident:12288 KB, virtual:16384 KB, no SHMTOTAL limit
20:36:52  SQLTRACE: SQL History Tracing set to 2000 SQL statements.
.......
```

Enabling tracing in a user mode

User mode provides more granularity for DBAs to narrow tracing to specific set of users or just one user. Because the tracing data is stored in memory buffers, setting the tracing only to required sessions gives you more control of the tracing memory usage.

Example 8-3 shows how to set tracing at the user level, but this setting needs to be accompanied by the administrative API task() or admin() to specify on which users that you want tracing, which is discussed in 8.1.2, “SQL trace using the administrative API functions” on page 222.

Example 8-3 Tracing at a user level

```
SQLTRACE level=HIGH,ntraces=500,size=50,mode=user
```

Setting up the SQL tracing parameters using the SQLTRACE configuration variable is good if you want to have default tracing done any time when the engine is online. But if you want to make any changes, use the administrative API functions discussed in 8.1.2, “SQL trace using the administrative API functions” on page 222.

8.1.2 SQL trace using the administrative API functions

The administrative API functions provide more flexibility in configuring the SQL tracing parameters. If you do not want to set the SQLTRACE configuration parameter to turn the tracing on at server startup time or you decide to make changes to the initial settings, the administrative API functions provide you with those capabilities.

The built-in administrative API functions, task() and admin(), from the sysadmin database provide the same functionality as SQLTRACE. However, setting or changing the tracing values using the API functions does not require you to restart the server. Only user *informix* can execute these API functions. With tracing enabled or disabled, using these API functions is effective only until the engine is restarted. After the engine is restarted, the SQLTRACE setting from the configuration file is used.

Enabling tracing in global mode using the API function

When SQLTRACE is turned off in the $ONCONFIG file and the administrative API is used to turn on tracing in global mode, the tracing is turned on with default values. As shown in Example 8-4, the default values for tracing are 1,000 buffers with size 1 KB each in low mode.

Example 8-4 also displays usage about how global tracing can be turned on using the administrative API task().

Example 8-4 Enabling tracing with API task()

```
informix@ramsay[144] dbaccess sysadmin -

Database selected.
> execute function task("set sql tracing on");

(expression)  Global Tracing ON Number of Traces 1000 Trace Size 1000 Mode Low

1 row(s) retrieved.
```

If you do not want to use the default values, you can change the values by passing them as parameters to the API function, as shown in Example 8-5 on page 223. Here the number of traces (buffers) is increased to 3,000, the buffer size is increased to 2 KB and the tracing level is also changed to medium instead of the default low.

Example 8-5 Changing trace values with API task()

```
execute function task("set sql tracing on", 3000, 2, "med", "global");

(expression)  Global Tracing ON Number of Traces 3000 Trace Size 2024 Mode Med

1 row(s) retrieved.
```

Assume that the engine was brought up with SQLTRACE mode set to *user* in the $ONCONFIG file, and now you want to change that mode to *global*. Simply running the command shown in Example 8-4 on page 222 cannot change it. You must either change the SQLTRACE mode value to global and restart the engine or run the API command task() as shown in Example 8-6.

Example 8-6 Steps to change trace mode

```
assuming your $ONCONFIG values for SQLTRACE were as below:

SQLTRACE        level=HIGH,ntraces=1000,size=1,mode=user

you can run the following API task() command to keep same values, you can also
change the values if you want to:

informix@ramsay[181] dbaccess sysadmin -

Database selected.


execute function task("set sql tracing on", 1000, 1, "high", "global");


(expression)  Global Tracing ON Number of Traces 1000 Trace Size 1000 Mode High

1 row(s) retrieved.
```

Note: Be sure to give the parameter values in the same order as the SQLTRACE syntax (as described in 8.1.1, “SQLTRACE” on page 219), or the tracing is turned on with unintended values, such as those shown in Example 8-7.

Here in Example 8-7 on page 224, the intended *ntraces* and *size* parameter values are switched by mistake. However, the API function does not reject the command even though the values are out of bounds for those parameter values’ ranges. Instead, it takes the minimum and maximum values allowed for those two parameters. What was intended was to have 1,000 buffers of 2 KB each, but the command executes without knowing that the values were switched by mistake. The result is that you get 500 buffers, because 2 is out of range for

ntraces. Therefore , it defaults to the minimum, which is 500, and 102376 bytes for buffer size, which is the maximum allowed. It is extremely important to follow the order of parameter values for both task() and admin() functions to get the setting that was intended.

Example 8-7 Parameters set in an incorrect order

```
> execute function task("set sql tracing on", 2, 1000, "med", "global");

(expression)  Global Tracing ON Number of Traces 500 Trace Size 102376 Mode Med

1 row(s) retrieved.
```

Enabling tracing for a specific user session

To enable the tracing at the user level, changing the mode in SQLTRACE in the $ONCONFIG file does not suffice. In order to specify the user or users for which you want to turn on the tracing, after you specify user as the mode in the SQLTRACE configuration parameter, you must also execute an administrative API task() or admin() function to turn SQL history tracing on for the user.

Example 8-8 shows the usage of the administrative API task to enable tracing for a specific user session. In this case, tracing for session 19 is enabled. SQLTRACE must have been configured to mode *user* prior to executing the task function.

Example 8-8 Enabling tracing for specific user

```
informix@ramsay[136] dbaccess sysadmin -
> execute function task("set sql user tracing on", 19);

(expression)  SQL user tracing on for sid(19).

1 row(s) retrieved.
```

If SQLTRACE is not set in the $ONCONFIG at all, you can turn the tracing on while the engine is running by using the API function. For instance, to enable low mode SQL history tracing for all users except *root* or *informix*, you can execute a task() or admin() function, as shown in Example 8-9 on page 225.

Example 8-9 Enabling user mode tracing with new trace values

```
dbaccess sysadmin -

execute function task("set sql tracing on", 1000, 1,"low","user");
 select task("set sql user tracing on", session_id)
        FROM sysmaster:syssessions
  WHERE username not in ("root","informix");
```

If you know which user sessions you want to monitor, we advise you to set the mode to user instead of global. The user level setting not only saves the memory usage but also saves time in analyzing the trace output to identify the problem, because you do not have to eliminate all the traces that are of no interest to get to the traces that you need.

Disabling SQL history tracing globally or for a session

Even if the mode specified in the SQLTRACE configuration parameter is global or user, you can disable SQL history tracing at any time if you want to completely turn off tracing and deallocate resources that are currently in use by the SQL tracing feature.

For example, global SQL tracing can be disabled by executing the statement shown in Example 8-10.

Example 8-10 Disabling tracing using task()

```
execute function task('set sql tracing off');
(expression) SQL tracing off.

1 row(s) retrieved.
```

You can also disable tracing for a particular user by using the administrative API. To disable SQL tracing for a particular session, execute a task() or admin() function as shown in Example 8-11. In this example, tracing is turned off for session 17.

Example 8-11 Disabling tracing for a specific user

```
execute function task(“set sql user tracing off",17);

(expression)  SQL user tracing off for sid(17).

1 row(s) retrieved.
```

Example 8-12 on page 226 displays how you can turn off tracing for all users currently logged in, except for user *informix*. This is convenient when you have many sessions on the systems and you know exactly which sessions you want to

track. You can save time by not having to turn off tracing for each session individually.

This level of control is very good when you know which session is the bottleneck, but you do not know which SQL within that session is creating the bottleneck. You can avoid allocating this extra memory to trace all the sessions, and you can trace only the session or sessions that you know are most likely the culprits.

Example 8-12 Turning off tracing for more than one session

```
select task("set sql user tracing off", sid)
FROM sysmaster:syssessions
WHERE username not in ("informix");

(expression)  SQL user tracing off for sid(20).

1 row(s) retrieved.

As you can see from the following "onstat -g ses" output, currently there are
only two session one in vijayl(20) and the other is informix(19) and the above
SQL turned off tracing for session 20 which is vijayl.

root@ramsay[159] onstat -g ses

IBM Informix Dynamic Server Version 11.10.UB4     -- On-Line -- Up 04:03:22 --
331776 Kbytes

session                                          #RSAM    total      used dynamic
id       user     tty      pid      hostname threads  memory     memory explain
21       informix -        0        -        0        12288      8392       off
20       vijayl   28       12194    ramsay   1        45056      39568      off
19       informix 22       12163    ramsay   1        188416     147112     off
16       informix -        0        -        1        339968     273848     off
15       informix -        0        -        1        270336     202592     off
2        informix -        0        -        0        12288      8392       off
......
```

This concludes our discussion about how to configure the tracing feature to collect the information needed for further analysis. We intentionally do not discuss the tracing parameter *level* (low, med, high) in this section, because it is more relevant to the trace display in 8.2.1, "onstat -g his" on page 227; therefore, we discuss it there.

8.2 How to display and analyze the trace information

In this section, we discuss how to display and analyze the information gathered by turning on the tracing. There are numerous ways to display the collected tracing information. You can either use the IDS administrative tool *onstat* or run SQL to select information from the sqltrace pseudo tables. The easiest way is by using **onstat -g his.**

8.2.1 onstat -g his

In IDS 11, a new *onstat* option has been introduced called **onstat -g his.** This option prints the information that has been collected by the SQLTRACE feature and displays the information in formatted fashion. At this time, **onstat -g his** can only print all the tracing information together as though it is a single trace. There is currently no option to print only for a specific SQL statement or a user session. However, you can capture the output into a file and search that file for specific information.

To understand the individual traces better, we have divided them into three categories: trace profile, statement text, and statement statistics. Be aware that these categories are conceptual only for the purpose of this example. You do not see them categorized in the actual **onstat -g his** output. In the actual output, you only see the output as one continuous trace. The categories are:

- Trace profile or settings: The first few lines of the output describe the SQLTRACE settings that are currently active, such as the trace level, trace mode, number of traces, trace buffer size, and the duration of the buffer in memory. In Example 8-13, you can see the portion of the output that is described as trace settings.

Example 8-13 Trace settings portion of onstat output

```
Statement history:

Trace Level                 High
Trace Mode                Global
Number of traces            3000
Current Stmt ID                0
Trace Buffer size           2024
Duration of buffer           280 Seconds
Trace Flags            0x00007F11
Control Block          b272018

Statement # 977:     @ b2982d0
```

- Statement text and iterators: The next few lines of the output describe the SQL statement that is being traced and the iterators and query plan information. The statement text portion of the output somewhat depends on the trace level that has been chosen in the settings. If the trace level is `low`, the output simply prints the SQL statement that is being traced and a hex value for the database in use. If the tracing level is `medium`, you see the database name instead of the hex value, the SQL statement, the table names that are involved in the SQL statement, and the stored procedure stack if it applies. If the tracing level is set to `high`, you see all the information for the level medium, and in addition, the host variables, if any are used in the SQL statement. The iterator information has no dependency on the level of tracing; it is similar on all the levels. Example 8-14 depicts the onstat output for the same SQL statement, but at the three levels of settings (low, medium, and high) so that you can see the differences that were just described.

Example 8-14 Trace output comparison for all three trace levels

```
Low:

Database:        0x100161
 Statement text:
  insert into customer values(?,?,?,?,?,?,?,?,?,?)

Med:

Database:        stores_demo
 Statement text:
  insert into customer values(?,?,?,?,?,?,?,?,?,?)

  INSERT using tables [ customer ]

HIGH:

Database:        stores_demo
 Statement text:
  insert into customer values(?,?,?,?,?,?,?,?,?,?)

  INSERT using tables [ customer ]

 Iterator/Explain
 ================
    ID   Left  Right   Est Cost   Est Rows   Num Rows     Type
     1      0      0          1          1          1     Insert

 Host Variables
 ==============
 1 char
```

```
 2 char
 3 char
 4 char
 5 char
 6 char
 7 char
 8 char
 9 char
10 char
```

- Statement information and statistics: This portion of the output contains the most important information of the SQL statement and the performance statistics. This portion can be further divided into three categories:
  - Statement information:
    - Session_id: Session that is executing the statement
    - User_id: Operating system user ID
    - Stmt Type: Statement type that is being executed
    - Finish Time: The time that the statement finished execution
    - Run Time: Total run time of the statement
  - RSAM statistics:
    - Buffer reads and writes
    - Page reads and writes
    - Memory sorts and disk sorts
    - Lock requests and waits
    - Logical log records
  - SQL statistics:
    - Estimated # of rows
    - Estimated cost
    - # of rows returned
    - SQL/ISAM errors, if any
    - Database isolation level
    - Memory used by SQL in bytes

Example 8-15 on page 230 illustrates the portion of **`onstat -g his`** output that has printed the statement information and statistics. Often, most of the analysis work is dependent on the information captured under *Statement Statistics*. Here, you find the information related to memory usage, I/O, lock usage and contention, CPU usage, the number of sorts, and index usage. This is definitely the most important tracing information in the entire output that is generated by onstat.

Example 8-15 Statement statistics part of onstat output

```
Statement information:
  Sess_id  User_id  Stmt Type          Finish Time    Run Time
  19       200      INSERT             21:27:29       0.0019

 Statement Statistics:
  Page       Buffer     Read       Buffer     Page       Buffer     Write
  Read       Read       % Cache    IDX Read   Write      Write      % Cache
  0          90         100.00     0          0          90         100.00

  Lock       Lock       LK Wait    Log        Num        Disk       Memory
  Requests   Waits      Time (S)   Space      Sorts      Sorts      Sorts
  197        0          0.0000     9.79 KB    0          0          0

  Total      Total      Avg        Max        Avg        I/O Wait   Avg Rows
  Executions Time (S)   Time (S)   Time (S)   IO Wait    Time (S)   Per Sec
  1          0.0019     0.0019     0.0019     0.000000   0.000000   539.1708
Estimated  Estimated  Actual     SQL        ISAM       Isolation  SQL
  Cost       Rows       Rows       Error      Error      Level      Memory
  0          0          0          0          0          CR         7888
```

Buffer overflow

The information collected for each SQL statement uses exactly one buffer to store that information. Hence, the size of the buffer matters when it comes to collecting the trace. A buffer overflow occurs when the buffer size that has been set using the trace parameter, *size*, cannot accommodate the entire trace information for that SQL statement. In this situation, the statement text section of the trace is truncated to fit in the buffer and a warning is printed in the trace output. Fortunately, the statement statistics are not affected by the buffer size, which is the most valuable information. Example 8-16 illustrates the **onstat** output that demonstrates a buffer overflow. You can correct this situation by using the administrative API function task() or admin() to reconfigure the tracing parameter and the database server dynamically adjusts the values.

Example 8-16 Buffer overflow

```
Statement # 847:     @ b293190

  WARNING: Data has been truncated, trace buffer not large enough.
 Database:
 Statement text:
  create dba procedure informix.systdist (table_id int, column_no int)
       returning int, datetime year to fraction (5), char(1),
                 smallfloat, smallfloat, float , stat, char(1);
```

```
    define v_tabauth      char(8);
    define v_colauth      char(3);
    define is_allowed     int;
    define search_columns int;
    define v_colno        smallint;
    define v_seqno        int;
    define v_constr_time  datetime year to fraction(5);
    define v_mode         char(1);
    define v_resolution   smallfloat;
    define v_confidence   smallfloat;
    define v_encdat       stat;
    define v_owner        char(8);
    define user           procedure;
    define v_stattype         char(1);
    define v_smplsize         float;

-- First verify that the current user has select privileges on this
column
```

SQL trace on distributed query

A *distributed query* is one that involves two or more IDS servers, typically running on separate machines. If you want to enable tracing for distributed queries, the tracing needs to be enabled on both the coordinator and participant servers. The *coordinator* is where the query is run and the *participant* is the branch server that executes a part of the query.

If the SQL tracing is turned on only for the coordinator but not for the participant, the query iterators and explain information are gathered for both the servers but the statement statistics are collected only for the coordinator. In order to collect the statement statistics on the participant server, the tracing needs to be turned on for that server as well.

In Example 8-17 on page 232, you can see the trace information collected on both the coordinator and the participant for a distributed query. Notice that under the Iterators/Explain section of the coordinator trace, the *RemScan* is in fact the participant server query plan. In the participant server trace, notice the statement text that has been sent from the coordinator. It is relatively easy to determine that it is not a local query.

Example 8-17 Distributed query trace

Coordinator:

```
Statement # 226:     @ bac1d70

 Database:        stores_demo
 Statement text:
  select l.customer_num, l.lname, l.company,
              l.phone, r.call_dtime, r.call_descr
            from customer l,  stores_demo@vj_acme_tcp:cust_calls r
             where l.customer_num = r.customer_num

  SELECT using tables [ customer cust_calls ]

 Iterator/Explain
 ================
    ID   Left  Right   Est Cost   Est Rows   Num Rows     Type
     2      0      0          4          7          7     RemScan
     3      0      0          1         28          1     Index Scan
     1      2      3         10          7          7     Nested Join

 Statement information:
  Sess_id  User_id  Stmt Type          Finish Time    Run Time
  18       200      SELECT             21:27:05       0.1350



 Statement Statistics:
  Page       Buffer     Read       Buffer     Page       Buffer     Write
  Read       Read       % Cache    IDX Read   Write      Write      % Cache
  5          34         85.29      0          0          0          0.00

  Lock       Lock       LK Wait    Log        Num        Disk       Memory
  Requests   Waits      Time (S)   Space      Sorts      Sorts      Sorts
  0          0          0.0000     0.000 B    0          0          0

  Total      Total      Avg        Max        Avg        I/O Wait   Avg Rows
  Executions Time (S)   Time (S)   Time (S)   IO Wait    Time (S)   Per Sec
  1          0.1358     0.1358     0.1350     0.000092   0.000275   51.8333

  Estimated  Estimated  Actual     SQL        ISAM       Isolation  SQL
  Cost       Rows       Rows       Error      Error      Level      Memory
  10         7          7          0          0          NL         18784
```

Participant:

```
Statement # 226:     @ bac0d70
```

```
Database:        stores_demo
Statement text:
 select x0.call_dtime ,x0.call_descr ,x0.customer_num from
   stores_demo:"informix".cust_calls x0

 SELECT using table [ cust_calls ]

Iterator/Explain
================
   ID   Left  Right   Est Cost   Est Rows   Num Rows     Type
    1      0      0          4          7          7     Seq Scan

Statement information:
 Sess_id  User_id  Stmt Type          Finish Time    Run Time
 18       200      SELECT             23:30:50       0.0079

Statement Statistics:
 Page       Buffer     Read       Buffer     Page       Buffer     Write
 Read       Read       % Cache    IDX Read   Write      Write      % Cache
 4          11         63.64      0          0          0          0.00

 Lock       Lock       LK Wait    Log        Num        Disk       Memory
 Requests   Waits      Time (S)   Space      Sorts      Sorts      Sorts
 0          0          0.0000     0.000 B    0          0          0

 Total      Total      Avg        Max        Avg        I/O Wait   Avg Rows
 Executions Time (S)   Time (S)   Time (S)   IO Wait    Time (S)   Per Sec
 1          0.0158     0.0158     0.0079     0.000079   0.000158   887.8404

 Estimated  Estimated  Actual     SQL        ISAM       Isolation  SQL
 Cost       Rows       Rows       Error      Error      Level      Memory
 4          7          7          0          0          NL         6688
```

8.2.2 Using syssqltrace

The *syssqltrace table* is one of the 3 in-memory pseudo tables, which are constantly updated by the IDS engine while the tracing is activated, to store the trace information. The syssqltrace table provides detailed tracing information about a single SQL statement in each of its rows. The size of this table (memory buffer) is what is determined by the tracing configuration values (ntraces and size) that you provide. The size is dynamically adjusted whenever these values are reconfigured. Because it is using the database's shared memory, you must apply caution in configuring its size. For example, if you trace a small number of SQL statements, but you set your ntraces and size to high values, the unused tracing space is allocated in the memory but not used, so it is not available for other processes. But, if the values are set too low and the tracing information

exceeds the allocated space, part of the traced data can be lost, because the system truncated it to fit into the allocated buffer.

The schema of syssqltrace is stored in the IDS system monitoring database called *sysmaster*. Table 8-1displays the schema of this table. Notice the similarities between the **`onstat -g his`** output and this schema. The **`onstat`** tool reads the SQL statement information and statistics portion of its output from this table to display it.

Table 8-1 The syssqltrace schema

Column	Type	Description
sql_id	int8	Unique SQL execution ID
sql_address	int8	Address of the statement in the code block.
sql_sid	int	Database session ID of the user running the SQL statement.
sql_uid	int	User ID of the statement running the SQL.
sql_stmttype	int	Statement type.
sql_stmtname	char(40)	Statement type displayed as a word.
sql_finishtime	int	Time this statement completed (UNIX).
sql_begintxtime	int	Time this transaction started.
sql_runtime	float	Statement execution time.
sql_pgreads	int	Number of disk reads for this SQL statement.
sql_bfreads	int	Number of buffer reads for this SQL statement.
sql_rdcache	float	Percentage of time the page was read from the bufferpool.
sql_bfidxreads	int	Number of index page buffer reads.
sql_pgwrites	int	Number of pages written to disk.
sql_bfwrites	int	Number of pages modified and returned to the bufferpool.
sql_wrcache	float	Percentage of time a page was written to the bufferpool but not to disk.
sql_lockreq	int	Total number of locks required by this SQL statement.

Column	Type	Description
sql_lockwaits	int	Number of times the SQL statement waited on locks.
sql_lockwttime	float	Time the system waited for locks during the SQL statement.
sql_logspace	int	Amount of space the SQL statement used in the logical log.
sql_sorttotal	int	Number of sorts that ran for the statement.
sql_sortdisk	int	Number of sorts that ran on disk.
sql_sortmem	int	Number of sorts that ran in memory.
sql_executions	int	Number of times the SQL statement ran.
sql_totaltime	float	Total amount of time spent running the statement.
sql_avgtime	float	Average amount of time spent running the statement.
sql_maxtime	float	Maximum amount of time spent executing the SQL statement.
sql_numiowaits	int	Number of times an I/O operation had to wait.
sql_avgiowaits	float	Average amount of time that the SQL statement had to wait.
sql_totaliowaits	float	Total amount of time that the SQL statement had to wait for I/O. This excludes any asynchronous I/O.
sql_rowspersec	float	Average number of rows (per second) produced.
sql_estcost	int	Cost associated with the SQL statement.
sql_estrows	int	Estimated number of rows returned for the SQL statement as predicted by the optimizer.
sql_actualrows	int	Number of rows returned for the SQL statement.
sql_sqlerror	int	SQL error number.
sql_isamerror	int	RSAM/ISAM error number.
sql_isollevel	int	Isolation level of the SQL statement.
sql_sqlmemory	int	Number of bytes needed to execute the SQL statement.
sql_numiterators	int	Number of iterators used by the statement.

Column	Type	Description
sql_database	char(128)	Database name.
sql_numtables	int	Number of tables used in executing the SQL statement.
sql_tablelist	char(4096)	List of table names directly referenced in the SQL statement. If the SQL statement fires triggers that execute statements against other tables, the other tables are not listed.
sql_statement	char(1600)	SQL statement that ran.

In order to display the SQL trace information (other than by using **`onstat -g his`**), you can directly run SQL queries on this table from the sysmaster database and get the same information. This is convenient when you want to look at a particular SQL statement, all the SQL statements that are run by a session, or any other filter that you want to use on this schema. Example 8-18, for instance, displays all the SQL statements that are traced for session 25. The onstat output prints trace information of all the SQL statements that are executed. In order to look at the specific SQL statement in which you are interested, you must look through the entire output, or you can run selects on this table.

Example 8-18 Querying on the syssqltrace table

```
> select * from syssqltrace where sql_sid = 25;

sql_id          3
sql_address     194531856
sql_sid         25
sql_uid         200
sql_stmttype    2
sql_stmtname    SELECT
sql_finishtime  1180420443
sql_begintxtime 1180420263
sql_runtime     1.997457400000
sql_pgreads     0
sql_bfreads     1284
sql_rdcache     100.0000000000
sql_bfidxreads  0
sql_pgwrites    0
sql_bfwrites    0
sql_wrcache     0.00
sql_lockreq     636
sql_lockwaits   0
sql_lockwttime  0.00
sql_logspace    0
sql_sorttotal   0
```

```
sql_sortdisk      0
sql_sortmem       0
sql_executions    1
sql_totaltime     3.544297100000
sql_avgtime       3.544297100000
sql_maxtime       1.997457400000
sql_numiowaits    0
sql_avgiowaits    0.00
sql_totaliowaits  0.00
sql_rowspersec    106.1349293357
sql_estcost       105
sql_estrows       211
sql_actualrows    212
sql_sqlerror      0
sql_isamerror     0
sql_isollevel     2
sql_sqlmemory     19248
sql_numiterators  3
sql_database      <None>
sql_numtables     0
sql_tablelist     None
sql_statement     select l.tabname, r.created from sysmaster:systables l,
sysmaster@vj_acme_tcp:systables r where l.tabname = r.tabname
```

8.2.3 Using syssqltrace_info

The table *syssqltrace_info* is another pseudo table similar to the *syssqltrace* table. The syssqltrace_info table stores the tracing profile information while syssqltrace stores the statement statistics information. The profile information consists of the trace settings, such as ntraces, size, memory used, and trace starting time. Table 8-2 on page 238 contains the schema of this table. The first few lines displayed by **onstat -g his** output are actually the information displayed from this pseudo table. However, you can also run SQL queries on this table just as you can with the syssqltrace table.

Table 8-2 The syssqltrace_info schema

Column	Type	Description
flags	integer	SQL trace flags
ntraces	integer	Number of items to trace
tracesize	integer	Size of the text to store for each SQL trace item
duration	integer	Trace buffer (in seconds)
sqlseen	int8	Number of SQL items traced since start or resizing
starttime	integer	Time tracing was enabled
memoryused	int8	Number of bytes of memory used by SQL tracing

8.2.4 Using syssqltrace_iter

The *syssqltrace_iter* table is yet another pseudo table that sqltracing creates and uses for the tracing mechanism. The syssqltrace_iter tables stores the SQL query plan and iterators information that is also used by **`onstat -g his`** to display the iterators and explain portion of the onstat output. This table also allows you to run SQL statements to query the table data. This is extremely useful if you just want to know the iterator plan and explain information for a specific SQL, which you need to obtain by running a query using the sql_id on this table instead of using **`onstat`**, which prints all of the tracing information together. Table 8-3 on page 239 illustrates the schema of syssqltrace_iter table.

Table 8-3 The syssqltrace_iter schema

Column	Type	Description
sql_id	int8	SQL execution ID
sql_address	int8	Address of the SQL statement block
sql_itr_address	int	int8 Address of the iterator
sql_itr_id	int	Iterator ID
sql_itr_left	int	Iterator ID to the left
sql_itr_right	int	Iterator ID to the right
sql_itr_cost	int	Iterator cost
sql_itr_estrows	int	Iterator estimated rows
sql_itr_numrows	int	Iterator actual rows processed
sql_itr_type	int	Iterator type
sql_itr_misc	int	Iterator miscellaneous flags
sql_it_info	char(256)	Iterator miscellaneous flags displayed as text

8.2.5 SQL Query Drill-Down from the IDSAdmin console

IDSAdmin, a Hypertext Preprocessor (PHP)-based administration console, is available with IDS 11. This administrative tool also provides an option to view the SQL Query Drill-Down tracing information. This is a good tool to view the information in a nicely formatted GUI at a click of the mouse. Figure 8-1 on page 240 illustrates a screen capture of an SQL trace from IDSAdmin console.

Figure 8-1 phpIDSAdmin SQL trace

8.3 Summary

With the advent of the SQL Query Drill-Down feature, IDS 11 introduces a powerful performance monitoring tool, along with many other great new features. This feature is easy to understand, simple in implementation, and easy to use. It certainly makes the Informix database administration job even easier than it currently is already. The easiest way to understand this feature is by using it. Try it on a test system. For example, try configuring the SQLTRACE in different ways. Use the administrative API functions to change the trace level, trace mode, and size to see how the **`onstat -g his`** output differs with those changes. Practicing gives you a good understanding of the feature and it enables you to better use the correct settings for your production system. It is important not only to use the SQLTRACE effectively, but also optimally.

9

SQL Language

The Structured Query Language (SQL) is the language most widely used to model, create, and query the data in relational databases. And, there is an SQL language standard committee that continues to refine and expand the SQL standard. All relational database management system (RDBMS) vendors support various levels of the SQL standard and proprietary enhancements.

IDS 11 enhances compliance with the SQL standard and enables easier application development, integration, and maintenance.

In this chapter, we discuss those features, which include support for:

- Subqueries in the FROM clause
- Named parameter support for JDBC
- Multiple triggers for the same event
- Stored procedure language enhancements
- New data types for hierarchical data modeling and access methods for text search and messaging:
  - Node
  - Binary18
  - Binaryvar
  - Built-in text search access method
  - Access to WebSphere® MQ

- Optimizer enhancements:
  - Index self-join
  - Support for directives in ANSI join queries
  - Statistics feedback in the explain file
- New last committed read isolation level for enhanced concurrency
- XML publishing functions
- Infrastructure improvements:
  - More about statistics, automatic statistics collection while creating the index, sysdbopen(), sysdbclose())
  - Distributed query support for new datatypes
  - New functions

There is quite a bit of information, so let us get started.

9.1 Hierarchical data type (Node DataBlade)

In this section, we present examples illustrating hierarchical relationships. Assume that John Smith is the father of Paul, and Paul is the father of Peter; therefore, John is an ancestor of Peter by transitivity. A crate of cola packages contains many cases of cola which in turn contains many six-packs of cola cans. Each of these entities, crates, cases, six-packs, and cans, has product identifiers. When you design the schema to store this type of hierarchical information, it is not enough to store the values for each node, it is also important to represent the hierarchical relationships and enable queries on relationships among the entities in the hierarchy.

You want to answer questions, such as from which crate did a can of cola come, so that you can determine from which bottling facility this can of cola came. Relational databases are good at handling relations and logical hierarchies. However, you write convoluted SQL if you attempt to model hierarchical data and try to implement transitive closure on multi-level hierarchies, such as process dependency.

IDS 11 provides a new node data type to model hierarchical relationships. The data type and the support functions, ancestor, depth, getparent, getmember, length, and so forth, are packaged as the *Node DataBlade Module Version 2.0* in IDS 11. You need to register this datablade in your database for using the Node datatype.

The Node data type is an opaque data type that models the tree structure instead of flattening hierarchies to relations. Each value represents the edge of the hierarchy, not simply a number or a string. Therefore, when you increment node 1.9, you get node 1.10, and not the numerical incremental value of 1.91 or 2.9.

Consider the relationships depicted in Figure 9-1 on page 244. There are vice presidents under the CEO, and vice presidents have administrative assistants and managers under them.

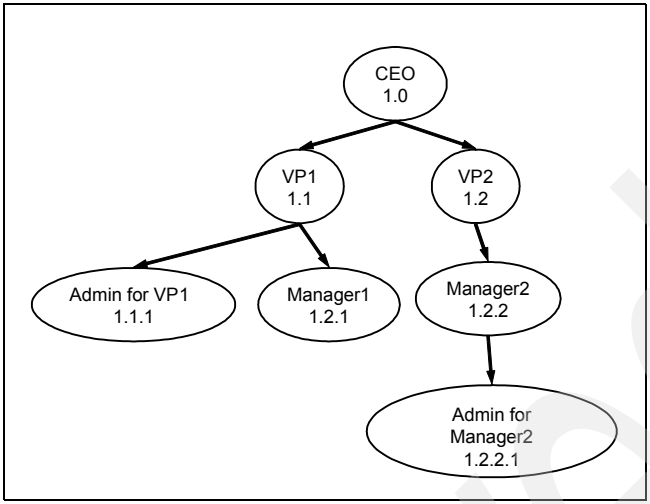


Figure 9-1 Organization

To develop the organization chart, we first created an `employees` table upon which it is based. That process is depicted in Example 9-1.

Example 9-1 Employees table

```
-- create employees table.
CREATE TABLE Employees(Employee_Id NODE, desc VARCHAR(60));

-- insert the hierarchical information.
INSERT INTO Employees VALUES ('1.0',     "CEO");
INSERT INTO Employees VALUES ('1.1',     "VP1");
INSERT INTO Employees VALUES ('1.1.1',   "Admin for VP1");
INSERT INTO Employees VALUES ('1.2',     "VP2");
INSERT INTO Employees VALUES ('1.2.1',   "Manager1");
INSERT INTO Employees VALUES ('1.2.2',   "Manager2");
INSERT INTO Employees VALUES ('1.2.2.1', "Admin for Manager2");

-- Retrieve the hierarchy for the "Admin for Manager2"

SELECT *
FROM Employees E
WHERE isAncestor(Employee_Id, '1.2.2.1')
ORDER BY E.Employee_Id ;

employee_id  desc
1.0              CEO
1.2              VP2
1.2.2           Manager2
```

```
-- return all the people under a manager, in this example under VP2

SELECT *
FROM   Employees E
WHERE  Employee_Id > '1.2';

employee_id        desc

1.2.1              Manager1
1.2.2              Manager2
1.2.2.1            Admin for manager2


-- retrieve and list each employee and their position in order.

select e.employee_id, e.desc from employees e order by depth(e.employee_id)
desc;

employee_id         desc

1.2.2.1             Admin for Manager2
1.1.1               Admin for VP1
1.2.1               Manager1
1.2.2               Manager2
1.1                 VP1
1.2                 VP2
1.0                 CEO

-- generate the hierarchy, bottom up.

select e.employee_id,
       e.desc,
       depth(e.employee_id) level,
       getparent(e.employee_id) manager_id
from employees e
order by 3 desc, 4

employee_id     desc                level       manager_id

1.2.2.1        Admin for Manager2     4            1.2.2
1.1.1          Admin for VP1          3            1.1
1.2.2          Manager2               3            1.2
1.2.1          Manager1               3            1.2
1.1            VP1                    2            1.0
1.2            VP2                    2            1.0
1.0            CEO                    1
```

See the IDS 11 documentation for further details. The developerWorks article about the external Node DataBlade module describes an earlier version of the node datablade. The article is located at:

http://www-128.ibm.com/developerworks/db2/zones/informix/library/techarticle/db_node.html

9.2 Basic Text Search index

SQL provides LIKE and MATCHES predicates for simple pattern matching on character columns: col LIKE 'joe%', col NOT MATCHES '%Tuesday%'. Because these predicates are evaluated either by a B-tree index scan or table scan, large numbers of index entries or rows are scanned to evaluate this predicate.

The *Basic Text Search DataBlade module (BTS)* provides more efficient word and phrase searching on an unstructured document repository. This document repository is stored in a column of a table. The column can be of the type char, varchar, nvarchar, lvarchar, Binary Large Object (BLOB), or Character Large Object (CLOB). IDS nvarchar can store multibyte strings, and this text search engine can index and search the nvarchar columns. The search predicate can be a simple word, a phrase, a simple Boolean operator (AND, OR, or NOT), single or multiple wildcard searches, a fuzzy search, or a proximity search. Use the bts_contains() function to specify the text search predicate. The optimizer considers the BTS index if and only if you specify the filter through bts_contains() and does not consider the BTS index to evaluate LIKE and MATCHES predicates.

The index works in the DIRTY READ isolation level regardless of the isolation level set in the server. This implies that any modification, INSERT, UPDATE, or DELETE, performed on the index by transaction T1 is immediately seen by any other transaction accessing the index after the modification and before T1 is committed (or rolled back). Notice that updates to the index are synchronous with table updates. Rolling back the transaction will remove the index entry.

The illustration depicted in Figure 9-2 on page 247 shows how BTS and CLucene work together. (Note the index structure is simplified.)

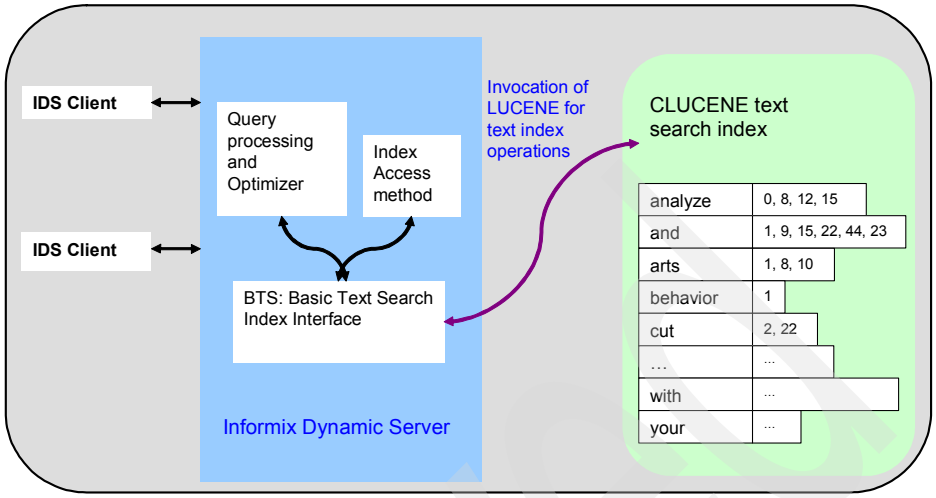


Figure 9-2 BTS and CLucene

The process for creating the example in Figure 9-2 is shown in Example 9-2.

Example 9-2 BTS and CLucene

```
Usage:
-- First, register bts.1.0 into your database using blademgr
mkdir /work/myapp/bts_expspace_directory
-- Create an external space to hold the index
onspaces -c -x bts_extspace -l /work/myapp/bts_expspace_directory

onmode -p +1 bts
-- you can add the bts noyield VP to ONCONFIG as well.

--Create a table with a BTS index
CREATE TABLE article_tab(id integer, title lvarchar(512));
-- Load the data in Table 9-1.
```

Table 9-1 Table data

id (integer)	Title (lvarchar(512))
0	Understand locking behavior and analyze lock conflicts in IDS.
1	Informix and Open source.
2	Use Informix to host a Java application service.
3	Optimize your BAR performance using parallel backups on IDS.
5	Flexible fragmentation strategy in Informix Dynamic Server 10.00.
6	Push the limits of Java user-defined routines (UDRs) in Informix Dynamic Server 10.00.
...	...

```
CREATE INDEX title_index ON article_tab(title bts_lvarchar_ops)
USING bts in bts_extspace;

Examples:

select id from article_tab where bts_contains(title, 'informix')

Results:

   id

    5
    1
    6
    2

Plan:
Estimated Cost: 0
Estimated # of Rows Returned: 1

  1) keshav.article_tab: INDEX PATH

    (1) VII Index Keys: title   (Serial, fragments: ALL)
        VII Index Filter:
informix.bts_contains(keshav.article_tab.title,informix )

SELECT id FROM article_tab
WHERE bts_contains(title, ' "use informix" ');
```

```
        id

          2

1 row(s) retrieved.

-- with the AND Boolean operator (&& and + are allowed as well)
SELECT id FROM article_tab WHERE bts_contains (title, 'informix AND dynamic') ;

        id

          5
          6

2 row(s) retrieved.

-- with the single character wildcard
SELECT id FROM article_tab WHERE bts_contains (title, 'inf*rmix') ;

        id

          5
          1
          6
          2

4 row(s) retrieved.

-- with the proximity search
SELECT id FROM article_tab WHERE bts_contains (title,'"java"~10') ;

        id

          6
          2

2 row(s) retrieved.
```

Example 9-3 depicts a query plan that is generated when the BTS index is chosen. Notice the bts_contains() filter is pushed down to the index to evaluate.

Also, notice that the estimated cost is zero and the estimated number of rows returned is 1. Both of these numbers are unreliable and the cost of using the BTS index is kept as zero to force the optimizer to choose the BTS index.

Example 9-3 Query plan with BTS index

```
QUERY:
------
SELECT id FROM article_tab WHERE bts_contains (title,'"java"~10')

Estimated Cost: 0
Estimated # of Rows Returned: 1

  1) keshav.article_tab: INDEX PATH

    (1) VII Index Keys: title   (Serial, fragments: ALL)
        VII Index Filter:
informix.bts_contains(keshav.article_tab.title,"java"~10 )

UDRs in query:
--------------
    UDR id  :   400
    UDR name:   bts_contains
```

9.3 Binary data types

In IDS, binary data can be stored in BYTE or BLOB data types. Both types are designed to handle large data volumes (BYTE up to 2 GB and BLOB up to 4 TB), are stored out of row, and do not support indexes. For applications using smaller binary strings, IDS 11 has two new types to consider: *binary18* and *binaryvar*. The input to these data types are ASCII strings with hexadecimal [0-9A-Fa-f] digits. Because the data is stored as bytes, the input hexadecimal string must have an even number of digits. Binary18 is a fixed 18 byte type. Binaryvar is a variable length and can store 255 bytes. IDS stores these types in-row and supports B-tree indexes on these types. IDS considers available indexes on these binary types as viable access paths. The Binary DataBlade module shipped with IDS provides these binary data types. Register the Binary DataBlade module with BladeManager and then create tables using the new types, as shown in Example 9-4.

Example 9-4 Create tables

```
create table bin18_test (int_col integer, bdt_col binary18);

insert into bin18_test values (0, '0102');
insert into bin18_test values (1, '01020304');
insert into bin18_test values (2, '0102030405060708');
insert into bin18_test values (3, '0102030405060708090A0B0C');

-- The hexadecimal prefix 0x is allowed as a prefix.
```

```
insert into bin18_test values (3, '0X0102030405060708090A0B0C');

create table bindata_test (int_col integer, bin_col binaryvar)

insert into bindata_test values (1, '30313233343536373839')
insert into bindata_test values (2, '0X30313233343536373839')

-- create indices on binary types
create index idx_bin18 on bin18_test(bdt_col);
create index idx_binvar on bindata(bin_col);
```

Both data types are indexable with a B-tree index and, therefore, are considered during optimization of the query. IDS does not create distributions on binary columns and the optimizer assumes a selectivity of 0.1 from the binary column. Use query directives to force the selection of alternative access paths when you see that these assumptions result in suboptimal plans for queries with binary data types.

The following new functions operate on binary columns:

- length(binary_column)
- octet_length(binary_column)

Here is an example of their use:

```
select length(bin_col) from bindata_test where int_col=1;
(expression)
10
```

The following bitwise operations are supported on binary data types:

- bit_and(arg1, arg2) implements the bitwise AND operator.
- bit_or(arg1, arg2) implements the bitwise OR operator.
- bit_xor(arg1, arg2) implements the bitwise XOR (exclusive OR) operator.
- bit_complement(arg1) implements the bitwise NOT.

Here is an example of their use:

```
create table bindata_test (int_col integer, bin_col binaryvar)

insert into bindata_test values (1, '00001000');
insert into bindata_test values (2, '00002000');
insert into bindata_test values (3, '00004000');
insert into bindata_test values (4, '023A2DE4');

select bit_or(bin_col, '00004000')
from bindata_test where int_col=2;  -- add CASE stmt here.
```

```
(expression)  00006000

select bit_and(bit_or(bin_col, '40404040'), '01010200')
from bindata_test where int_col=2;

(expression)  00000000

select bit_complement(bin_col) from bindata_test where int_col=4;

(expression)  FDC5D21B

select bit_xor(bin_col,'00004040') from bindata_test where
int_col=3;

(expression)  00000040
```

If either argument is NULL, a NULL binary is returned. If the lengths of the two arguments are different, the operation is performed up to the length of the shorter string. The rest of the data from the longer binary string is appended to the result of the binary operation thus far.

This is an example:

```
create table x(a binaryvar, b binaryvar);
insert into x values('aa', 'aaaa');

select bit_and(a,b), bit_or(a,b), bit_xor(a,b), bit_complement(a)
from x;

(expression)  AAAA
(expression)  AAAA
(expression)  00AA
(expression)  55
```

9.4 WebSphere MQ messaging in IDS applications

WebSphere MQ (WMQ) provides reliable messaging for distributed, heterogeneous applications to interact, exchange information, delegate jobs, and offer services by action upon information received. Historically, applications built for this scenario had to be written with custom code, manage multiple connections, and route data between WMQ and IDS. IDS 10.00.UC3 introduced built-in support to enable IDS applications to interact with WMQ through SQL, therefore eliminating the overhead. Subsequent releases included enhanced and

extended platform support for WMQ. MQ functions are provided in IDS as the MQ DataBlade module.

Whenever you buy a book through the Internet or enroll in e-business with ibm.com, the order event triggers a workflow of the information through multiple modules: user account management, billing, packaging and shipping, procurement, customer service, and partner services. The execution in triggered modules generates subsequent workflow. To meet reliability and scaling requirements, it is typical to have application modules on multiple machines.

If you use the same software on all systems, for example, the SAP® stack, the software usually comes with workflow management features. If the modules run in a homogeneous environment, such as Linux machines running WebSphere and Informix, it is easier to change information using distributed queries or enterprise replication. Alternatively, if the application is running on heterogeneous systems, such as combinations of WebSphere, DB2, Oracle®, and Informix, programming and setup of distributed queries or replication becomes complex and, in many cases, does not meet application requirements.

WebSphere MQ is designed to address integration issues such as this situation. It prefers no platform and enforces no paradigms. WebSphere MQ supports more than 80 platforms and APIs in C, C++, Java, Java Message Service (JMS), and Visual Basic®. WebSphere MQ is the mainstay for designing the enterprise service bus (ESB) for Service-Oriented Architecture (SOA).

WebSphere MQ provides a reliable store-and-forward mechanism so that each module can send and receive messages to and from it. This mechanism is achieved by persistent queues and APIs for programming. In addition, WebSphere MQ Message Broker, another product in the WebSphere MQ product suite, provides message routing and translation services. Simplicity of infrastructure means the applications must establish, for example, message formats and queue attributes. WebSphere MQ also supports publish and subscribe semantics for queues, making it easy to send a single message to multiple receivers and subscribing messages from queues by need, similar to mailing lists.

Figure 9-3 illustrates how an IDS applications can use WebSphere MQ.

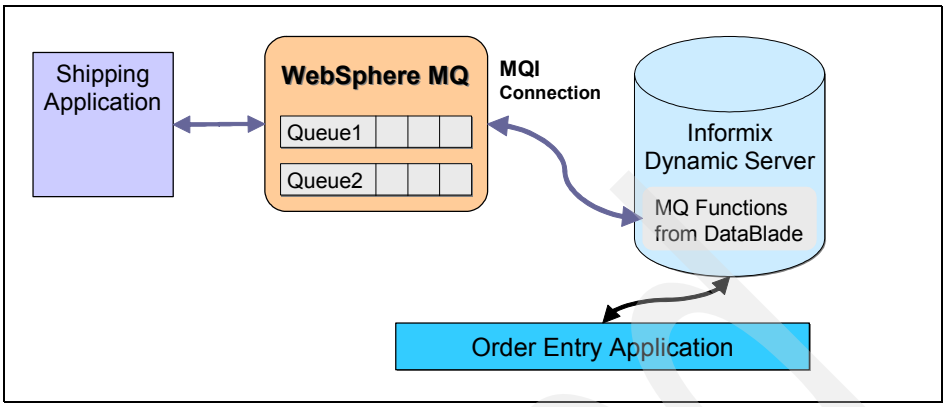


Figure 9-3 An IDS application and WebSphere

IDS provides SQL-callable functions to read, subscribe, and publish with WMQ. These SQL-callable functions expose MQ features to IDS applications and integrate the MQ operations into IDS transactions. For example, IDS uses two-phase commit protocol in distributed transactions; MQ operations commit and roll back along with IDS transactions.

Using IDS MQ functionality to send and receive a message to and from an MQ Queue is simple and is illustrated in Example 9-5.

Example 9-5 IDS and MQ

```
SELECT MQSend("CreditProc", customerid || ":" || address ||  ":"
|| product ":" || orderid)
FROM   order_tab
WHERE customerid = 1234;

INSERT into shipping_tab(shipping_msg) values(MQReceive());

CREATE PROCEDURE get_my_order();
define  cust_msg lvarchar[2048];
define  customerid  char[12];
define  address     char[64];
define  product     char[12];

-- Get the order from Order entry application.
EXECUTE FUNCTION MQReceive("OrderQueue") into cust_msg;
LET customerid = substr(cust_msg, 1, 12);
LET address    = substr(cust_msg, 14, 77);
LET product    = substr(cust_msg, 79, 90);

INSERT into shipping_table(custid, addr, productid, orderid)
      Values(customerid, address, product, :orderid);
```

```
-- send the status to CRM application
EXECUTE FUNCTION MQSend("CRMQueue",
:ordereid || ":IN-SHIPPING");
RETURN 1;
END FUNCTION;
```

Table 9-2 lists the MQ functions in IDS.

Table 9-2 MQ functions in IDS

Function name	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.
MQReadClob()	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.
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 Virtual Table Interface (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.

Figure 9-4 illustrates how IDS and MQ manage transactions.

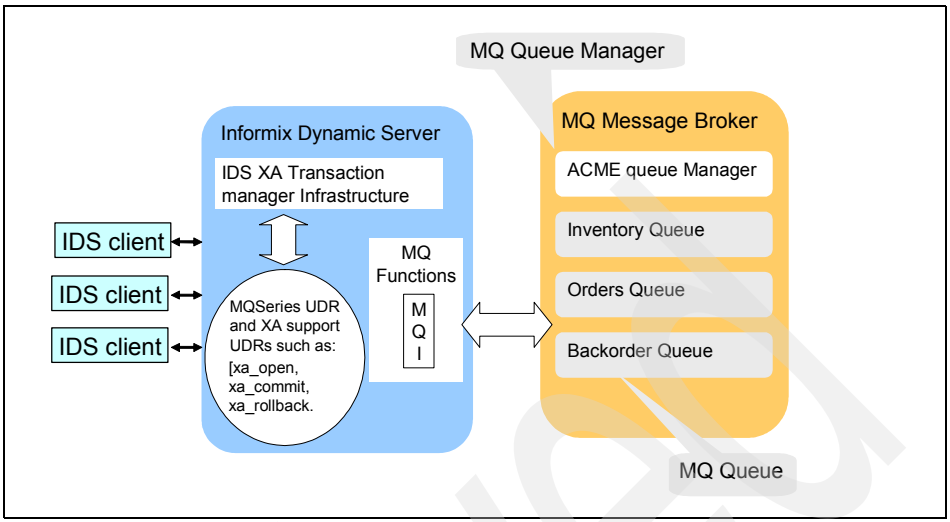


Figure 9-4 IDS and MQ

When you invoke any MQ function exchanging messages with MQ, you must be in a transaction, implicitly or explicitly. Transactions are necessary to provide reliable interaction between IDS and MQ. When the commit is successful, the application requires that all changes to data at IDS and MQ are persistent. When the application rolls back a transaction, any operations at MQ and on IDS are both rolled back. IDS implicitly starts a transaction when you issue Data Manipulation Language (DML) or (UPDATE, DELETE, INSERT, or SELECT) and Data Definition Language (DDL) statements (CREATE statements). You can explicitly start a new transaction with BEGIN WORK statements or use APIs, because JDBC starts a new transaction when autocommit is set to `off`. Note that the EXECUTE FUNCTION and EXECUTE PROCEDURE statements do not start a transaction; therefore, you need to start a transaction before invoking an MQ function in an EXECUTE statement.

Transaction management is transparent to an application. The application uses MQ functionality under a transaction and IDS handles the commit or rollback coordination between IDS and MQ using the open two-phase commit protocol. This process is integrated into the IDS transaction manager, which handles MQ along with its distributed transactions involving other IDS instances. During IDS-MQ interaction, IDS opens a connection to MQ. When the application invokes the first MQ function within a transaction, IDS begins a corresponding transaction at MQ. During commit or rollback, the IDS transaction manager is aware of MQ participation in the transaction and coordinates the transaction with it.

Table 9-3 shows on which platforms IDS supports MQ.

Table 9-3 Platform support for MQ

IDS Version	Supported platforms	WebSphere MQ Version
10.00.xC3 and higher	► Solaris - 32-bit ► HP/UX (PA-RISC) - 32-bit ► AIX - 32-bit ► Windows - 32-bit	V5.3 and higher
10.00.xC4 and higher	► AIX - 64-bit ► HP/UX (PA-RISC) - 64-bit	V6.0 and higher
10.00.xC5 and higher	► Linux (Intel®) - 32-bit ► Linux (System p™) - 64-bit ► Solaris - 64-bit	V6.0 and higher

IDS MQ functionality eliminates need for custom code development for IDS applications interacting with MQ. After you set up the queues, services, and policies, developers can use MQ functions just as they use other built-in functions in the development environment of their choice. If you set up the READ and RECEIVE tables, developers can directly query and insert data into them using SQL statements.

9.5 XML publishing and XPath functions

XML is used in document-centric and data-centric applications for integrating heterogeneous data and applications. For example, news feed with pictures, video, text, and data needs a document-centric approach, while an application that works with stock data or point of sale transactions needs a data-centric approach.

In IDS 11, you can use built-in functions to publish IDS result sets as XML documents, to verify that an XML document is well-formed and to extract parts of the document matching an XPATH pattern. These functions can be used directly from SQL statements or in stored procedures. Figure 9-5 illustrates these functions.

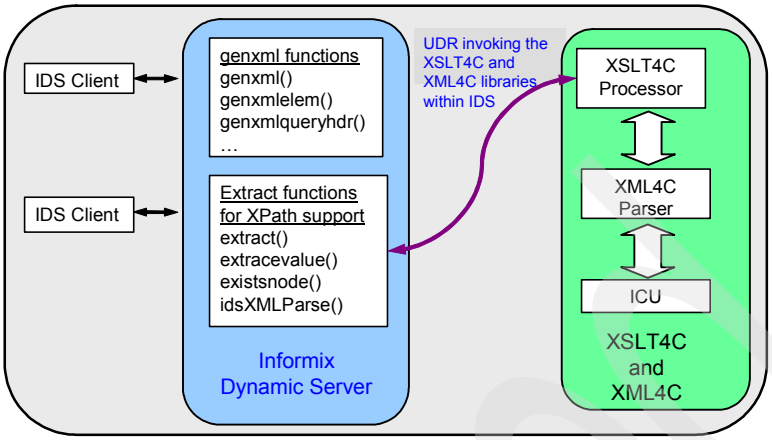


Figure 9-5 IDS and XML

XML Components

The XML4C processor is the open source XERCES XML parser library from IBM. It parses XML documents and can create a Document Object Model (DOM) object for a specific XML document. This is used by higher level libraries, such as XSLT4C, to validate and parse XML documents.

XSLT4C processor

Xalan is an XSLT processor for transforming XML documents into HTML, text, or other XML document types. XPath is included in the XSLT definition and uses the XPath support to provide XPath functionality in Extract functions.

The International Component for Unicode (ICU) is used for encoding, formatting, and sorting by XML4C and XSLT4C processors.

IDS XML functionality

The following are the set of functions added for publishing a query result set into an XML document.

- XML Publishing Functions: These functions return a single XML document after converting the result set into XML format. The functions with a Clob suffix are same as their equivalent functions, but they return CLOB to accommodate larger documents (that is, a larger result set).
  - Genxml() and GenxmlClob()
  - Genxmlqueryhdr() and GenxmlqueryhdrClob()
  - Genxmlelem() and genxmlelemClob()

- XPath functions for pattern matching and extraction within XML document: These functions apply the XPath expression on an XML document and return the matching portion from the XML document.
  - Extract() and Extractclob()
  - ExtractValue()
  - ExistsNode()

  In addition, the IDSXMLParse() function validates an XML document to be well formed.

Now, we illustrate publishing a result set into an XML document, and then use the extract functions to illustrate extracting parts of the XML document. To do this, we generated an XML document for all the rows in the table manufact, as depicted in Example 9-6.

Example 9-6 Publishing a result set into an XML document

```
execute function genxmlqueryhdr('manufact_set', 'select * from manufact');

(expression)  <?xml version="1.0" encoding="en_US.819" ?>
              <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
              <manufact_set>
              <row>
              <manu_code>SMT</manu_code>
              <manu_name>Smith          </manu_name>
              <lead_time>   3</lead_time>
              </row>
              <row>
              <manu_code>ANZ</manu_code>
              <manu_name>Anza           </manu_name>
              <lead_time>   5</lead_time>
              </row>
              <row>
              <manu_code>NRG</manu_code>
              <manu_name>Norge          </manu_name>
              <lead_time>   7</lead_time>
              </row>
              <row>
              <manu_code>HSK</manu_code>
              <manu_name>Husky          </manu_name>
              <lead_time>   5</lead_time>
              </row>
              <row>
              <manu_code>HRO</manu_code>
              <manu_name>Hero           </manu_name>
              <lead_time>   4</lead_time>
              </row>
              <row>
```

```
                <manu_code>SHM</manu_code>
                <manu_name>Shimara        </manu_name>
                <lead_time>  30</lead_time>
                </row>
                <row>
                <manu_code>KAR</manu_code>
                <manu_name>Karsten        </manu_name>
                <lead_time>  21</lead_time>
                </row>
                <row>
                <manu_code>NKL</manu_code>
                <manu_name>Nikolus        </manu_name>
                <lead_time>   8</lead_time>
                </row>
                <row>
                <manu_code>PRC</manu_code>
                <manu_name>ProCycle       </manu_name>
                <lead_time>   9</lead_time>
                </row>
                </manufact_set>

1 row(s) retrieved.

-- Generate an XML document for the list of customers with less than 105 as
their customer_num
> SELECT genxml( customer, 'customer') from customer where customer_num < 105;

genxml  <customer customer_num="101" fname="Ludwig         " lname="Pauli
         " company="All Sports Supplies " address1="213 Erstwild Court  " ci
        ty="Sunnyvale      " state="CA" zipcode="94086" phone="408-789-8075
          "/>
        <customer customer_num="102" fname="Carole         " lname="Sadler
         " company="Sports Spot         " address1="785 Geary St        " ci
        ty="San Francisco  " state="CA" zipcode="94117" phone="415-822-1289
          "/>
        <customer customer_num="103" fname="Philip         " lname="Currie
         " company="Phil&apos;s Sports       " address1="654 Poplar
         " address2="P. O. Box 3498      " city="Palo Alto      " state="CA" zi
        pcode="94303" phone="415-328-4543      "/>
        <customer customer_num="104" fname="Anthony        " lname="Higgins
         " company="Play Ball!          " address1="East Shopping Cntr. " ad
        dress2="422 Bay Road        " city="Redwood City   " state="CA" zipcode
        ="94026" phone="415-368-1100      "/>

1 row(s) retrieved.

-- Generate an XML document for the first 10 customers in customer table.
>  select genxml(first10customers, 'customer') FROM (select first 10 * from
customer) as first10customers;
```

```
genxml  <customer customer_num="101" fname="Ludwig         " lname="Pauli
            " company="All Sports Supplies " address1="213 Erstwild Court  " ci
        ty="Sunnyvale      " state="CA" zipcode="94086" phone="408-789-8075
          "/>
        <customer customer_num="102" fname="Carole         " lname="Sadler
            " company="Sports Spot         " address1="785 Geary St        " ci
        ty="San Francisco  " state="CA" zipcode="94117" phone="415-822-1289
          "/>
        <customer customer_num="103" fname="Philip         " lname="Currie
            " company="Phil&apos;s Sports       " address1="654 Poplar
         " address2="P. O. Box 3498      " city="Palo Alto      " state="CA" zi
        pcode="94303" phone="415-328-4543      "/>
        <customer customer_num="104" fname="Anthony        " lname="Higgins
            " company="Play Ball!          " address1="East Shopping Cntr. " ad
        dress2="422 Bay Road        " city="Redwood City   " state="CA" zipcode
        ="94026" phone="415-368-1100      "/>
        <customer customer_num="105" fname="Raymond        " lname="Vector
            " company="Los Altos Sports    " address1="1899 La Loma Drive  " ci
        ty="Los Altos      " state="CA" zipcode="94022" phone="415-776-3249
          "/>
        <customer customer_num="106" fname="George         " lname="Watson
            " company="Watson &amp;amp; Son        " address1="1143 Carver Place
        " city="Mountain View  " state="CA" zipcode="94063" phone="415-389-8789
             "/>
        <customer customer_num="107" fname="Charles        " lname="Ream
            " company="Athletic Supplies   " address1="41 Jordan Avenue    " ci
        ty="Palo Alto      " state="CA" zipcode="94304" phone="415-356-9876
          "/>
        <customer customer_num="108" fname="Donald         " lname="Quinn
            " company="Quinn&apos;s Sports      " address1="587 Alvarado
         " city="Redwood City   " state="CA" zipcode="94063" phone="415-544-872
        9      "/>
        <customer customer_num="109" fname="Jane           " lname="Miller
            " company="Sport Stuff         " address1="Mayfair Mart        " ad
        dress2="7345 Ross Blvd.     " city="Sunnyvale      " state="CA" zipcode
        ="94086" phone="408-723-8789      "/>
        <customer customer_num="110" fname="Roy            " lname="Jaeger
            " company="AA Athletics        " address1="520 Topaz Way       " ci
        ty="Redwood City   " state="CA" zipcode="94062" phone="415-743-3611
          "/>

1 row(s) retrieved.

-- Generate a report of a call from customer (with customer_num=106).
SELECT genxml(ROW(a.customer_num, a.fname, a.lname, b.call_dtime, b.call_code,
         b.call_descr, b.res_dtime, b.res_descr), 'customer_complaint_details')
FROM customer a, cust_calls b
WHERE a.customer_num = b.customer_num and a.customer_num = 106;
```

```
genxml  <customer_complaint_details customer_num="106" fname="George         "
        lname="Watson         " call_dtime="1998-06-12 08:20" call_code="D" cal
        l_descr="Order was received, but two of the cans of ANZ tennis balls wi
        thin the case were empty

                                              " res_dtime="1998-06-12 08:25" res_
        descr="Authorized credit for two cans to customer, issued apology. Call
        ed ANZ buyer to report the QA problem.

                                             "/>

1 row(s) retrieved.

-- Generate a report of a call from customer (with customer_num=106), but with
each column value as separate element.
SELECT genxmlelem(ROW(a.customer_num, a.fname, a.lname, b.call_dtime,
b.call_code,
               b.call_descr, b.res_dtime, b.res_descr),
'customer_complaint_details')
FROM customer a, cust_calls b
WHERE a.customer_num = b.customer_num and a.customer_num = 106;

genxmlelem  <customer_complaint_details>
            <row>
            <customer_num>106</customer_num>
            <fname>George        </fname>
            <lname>Watson        </lname>
            <call_dtime>1998-06-12 08:20</call_dtime>
            <call_code>D</call_code>
            <call_descr>Order was received, but two of the cans of ANZ tennis b
            alls within the case were empty

                                                                    </call_descr>
            <res_dtime>1998-06-12 08:25</res_dtime>
            <res_descr>Authorized credit for two cans to customer, issued apolo
            gy. Called ANZ buyer to report the QA problem.

                                                                  </res_descr>
            </row>
            </customer_complaint_details>

1 row(s) retrieved.

The extract functions take an XML document and XPath expression as
parameters and extract the match XML node, XML value or verify if the document
contains a specified XML pattern.
```

consider the following XML document in the booklist column of a row in table books:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
<book category="WEB">
  <title lang="en">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</price>
</book>
<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>
</bookstore>
```

The following examples extract part of the documents based the on the XPath expressions.

```
-- Extract all the books in the bookstore.
select extract(booklist, '/bookstore/book') from books where bookstore_id
=1234;

(expression)  <book category="COOKING">
<title lang="en">Everyday Italian</title>
<author>Giada De Laurentiis</author>
<year>2005</year>
<price>30.00</price>
</book>
       <book category="CHILDREN">
           <title lang="en">Harry Potter</title>
```

```
           <author>J K. Rowling</author>
<year>2005</year>
<price>29.99</price>
       </book>
      <book category="WEB">
           <title lang="en">XQuery Kick Start</title>
           <author>James McGovern</author>
           <author>Per Bothner</author>
           <author>Kurt Cagle</author>
           <author>James Linn</author>
           <author>Vaidyanathan Nagarajan</author>
           <year>2003</year>
           <price>49.99</price>
      </book>
     <book category="WEB">
          <title lang="en">Learning XML</title>
          <author>Erik T. Ray</author>
          <year>2003</year>
          <price>39.95</price>
     </book>

1 row(s) retrieved.

-- Extract the first book in the list.
select extract(booklist, '/bookstore/book[1]') from books  where bookstore_id
=1234;

(expression)  <book category="COOKING">
                   <title lang="en">Everyday Italian</title>
                  <author>Giada De Laurentiis</author>
                  <year>2005</year>
                  <price>30.00</price> </book>

1 row(s) retrieved.

-- Extract the last book from the list.
select extract(booklist, '/bookstore/book[last()]') from books  where
bookstore_id =1234;

(expression)  <book category="WEB">
                   <title lang="en">Learning XML</title>
                   <author>Erik T. Ray</author>
                   <year>2003</year>
                   <price>39.95</price>
                  </book>

1 row(s) retrieved.

-- Exctract just the values, without the XML tags
```

```
select extractvalue(booklist, '/bookstore/book[last()]') from books  where
bookstore_id =1234;

(expression)   Learning XML Erik T. Ray 2003 39.95

1 row(s) retrieved.

 -- Simply extract the price for all the books in the store.
-- Note this doesn't have the XML tags because text() is specified in the XPath
expression.
select extract(booklist, '/bookstore/book/price/text()') from books  where
bookstore_id =1234;

(expression)  30.00 29.99 49.99 39.95

1 row(s) retrieved.

-- extractvalue() expects only one return value.  Hence the error.
select extractvalue(booklist, '/bookstore/book/price/text()') from books;

 8356: Function (extractvalue) Returned multiple nodes for the query.
Error in line 10
Near character position 72

-- Xpath expression with a predicate (price > 45) in it.
select extract(booklist, '/bookstore/book[price>45]/title') from books;

(expression)  <title lang="en">XQuery Kick Start</title>

1 row(s) retrieved.

select extractvalue(booklist, '/bookstore/book[price>45]/title') from books;

(expression)  XQuery Kick Start

1 row(s) retrieved.

select extract(booklist, '/bookstore/book[price>45]/*') from books;

(expression)  <title lang="en">XQuery Kick Start</title>
                  <author>James McGovern</author>
                  <author>Per Bothner</author>
                  <author>Kurt Cagle</author>
                  <author>James Linn</author>
                  <author>Vaidyanathan Nagarajan</author>
                  <year>2003</year>
                  <price>49.99</price>

1 row(s) retrieved.
```

```
select extractvalue(booklist, '/bookstore/book[price>45]/*') from books;

 8356: Function (extractvalue) Returned multiple nodes for the query.
Error in line 17
Near character position 71

select extract(booklist, '//author') from books;

(expression)  <author>Giada De Laurentiis</author>
                    <author>J K. Rowling</author>
                    <author>James McGovern</author>
                    <author>Per Bothner</author>
                    <author>Kurt Cagle</author>
                    <author>James Linn</author>
              <author>Vaidyanathan Nagarajan</author>
              <author>Erik T. Ray</author>

1 row(s) retrieved.

select extract(booklist, '/bookstore/book[author="Kurt Cagle"]/title') from
books;

(expression)  <title lang="en">XQuery Kick Start</title>

1 row(s) retrieved.

select docid  from books
where existsnode(booklist, '/bookstore/book[author="Kurt Cagle"]/title') = 1;

      docid

          1

1 row(s) retrieved.

Database closed.
```

Refer to IDS 11 documentation for a complete list of the publishing and extract functions that are provided. Also, refer to the IDS 11 Information Center at:

http://publib.boulder.ibm.com/infocenter/idshelp/v111/index.jsp

9.6 Enhanced concurrency

In this section, we describe enhanced concurrency with committed read last committed isolation level.

Multi-user database systems implement Atomicity, Consistency, Isolation, and Durability (ACID) properties by locking the rows that were updated, or by locking the row's application's request and then serializing access to data in which multiple transactions are interested. These locking protocols come with drawbacks that include lost concurrency and throughput, lock conflicts and deadlocks, lock maintenance overhead, and so on.

For example, any modification to a table row can result in a lock being held on the row until the modifying transaction commits or rolls back. Readers for that row under any of the existing isolation levels, except *Dirty Read,* need to block until the modifying transaction is committed or rolled back. Similarly, a reader executing in the *Repeatable Read* isolation level locks the row for the duration of the transaction, preventing updates to that row by anyone else.

This feature implements a form of multi-versioning where readers can be returned one of two versions when you use table has row level locking: the last committed version of the data or the latest data. The two versions are the same under the existing lock-based isolation levels of Committed Read, Cursor Stability, and Repeatable Read, while they can be different for Dirty Read isolation level and for the new *Last Committed* option.

The Last Committed option is an optional extension to the existing Committed Read isolation level and has the following syntax:

```
Set Isolation To Committed Read Last Committed;
```

For application development debugging and support scenarios, the new SET ENVIRONMENT statement overrides the isolation mode for the current session when the user sets the Committed Read Last Committed isolation level:

```
SET ENVIRONMENT USELASTCOMMITTED ["None" | "Committed Read" | "Dirty
Read" | "All" ]
```

The configuration parameter USELASTCOMMITTED can be set to `None`, `Committed Read`, `Dirty Read`, or `All` to permanently override the Committed Read Last Committed isolation level.

We depict an example scenario in Table 9-4 on page 268. In this scenario, we transfer `$400.00` from account number `1234` to number `3456`. We then compare the different isolation levels to show the values that you get and the associated wait times.

Table 9-4 Last Committed option

Time	Transaction 1	Transaction 2 COMMITTED READ (default in logged database)	Transaction 3 Dirty Read	Transaction 4 LAST COMMITTED (New in IDS 11.10)
1	-- Current balance of customer 1234 is 1250.00			
2	set isolation to read committed;			
3	begin work;			
4	update cust_tab set balance =balance - 400 where cust_id = 1234;	begin work;	begin work;	begin work;
5	-- balance of customer 1234 is 850.00			
6	update cust_tab set balance = balance + 400 where cust_id = 3456;	select balance from cust_tab where custid = 1234; -- wait for the lock on row for customer 1234	select balance from cust_tab where custid = 1234; -- No waiting. will return 850.00.	select balance from cust_tab where custid = 1234; -- No waiting -- will return 1250.00
7	insert into daily_tab("transfer", 1234, 3456, 400);	-- Status: lock wait	-- Continue processing	-- Continue processing
8	Commit work;	-- Status: lock wait	-- do more	-- do more
9		--will return 850.00	-- do more	-- do more

We have illustrated the Last Committed read described in Table 9-4 in Figure 9-6 on page 269. However, we only show the first update and reads.

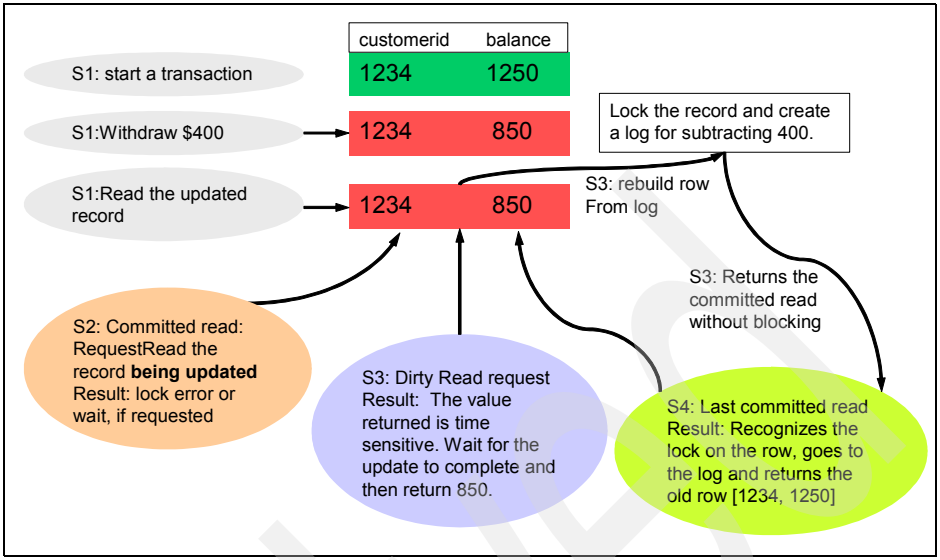


Figure 9-6 Last Committed read

To avoid waiting on the row lock, applications previously used Dirty Read isolation level. This isolation level provides dirty information for the row. In Table 9-4 on page 268, retrieving the dirty value of customer `1234` can be unreliable, because the transfer might not be successful and the transaction can be rolled back. The new isolation level, *Committed Read Last Committed*, provides the values that the row had before the latest update from an uncommitted transaction. In this case, `Transaction1` modified the row for the customer ID `1234`. That transaction is not guaranteed to commit or roll back. If an application needs to retrieve the value from the previously committed row, it can now use the Committed Read Last Committed isolation to enforce this isolation on the data it retrieves.

In this isolation level, when IDS finds a row that it is trying to read is locked, it goes to the logical log to find the log record created on the previous update of that row. IDS rolls back the update made to the current row, using the log record, to obtain the version of that row prior to the update, and then returns that row. If there are multiple updates to the same row in the same transaction, multiple log records are created. In last committed isolation mode, IDS finds all these records, after taking the current row, it rolls back each update to that row and returns the row.

9.7 Improved concurrency

In this section, we look at improved concurrency with private memory caches for Virtual Processors (VPs).

This feature adds an optional private memory cache for each CPU VP. This new cache contains blocks of free memory from 1 to 32 blocks in length, with each memory block containing 4,096 bytes. The purpose of this cache is to speed access to memory blocks. Normally, the server performs a bitmap search to find memory blocks of a requested length; this search requires a lock and can affect concurrency when large numbers of CPU VPs are configured on a multiprocessor system.

This feature can be enabled by setting the VP_MEMORY_CACHE_KB configuration parameter in the onconfig file. Setting it to 0 turns the private memory cache off. The minimum value for this parameter is 800; so at least 800 kilobytes must be allocated to each CPU VP. The memory used can be calculated by multiplying this value by the number of CPU VPs and must not exceed 40% of the memory limit as specified in the SHMTOTAL configuration parameter. The cache size must, in most circumstances, be set much smaller than the maximum value. This results in a high cache hit ratio and does not bind memory blocks to a single CPU VP. In addition, the normal memory management routines are capable of combining adjacent free blocks of memory whereas the CPU VP memory cache does not combine adjacent free memory blocks.

This feature can be turned on and off while the server is running with the **onmode** command:

```
onmode -wm VP_MEMORY_CACHE_KB=<value>  -- Configure current value
onmode -wf VP_MEMORY_CACHE_KB=<value>  -- modify in $ONCONFIG
```

The **onmode -wf** command updates the onconfig file so that the new value is used when you restart the server. If you set this value to 0 by using the **onmode** command, the memory caches are emptied and the feature is disabled. You can monitor the private cache by using the **onstat -g vpcache** command.

9.8 Automatic repreparation of SQL statements

It is typical for IDS applications to prepare large numbers of statements for SELECT, INSERT, UPDATE, and so forth and execute these statements or open cursors over these statements. This prepare once, execute multiple times model is efficient when used on dynamic statements for database interaction. However, in a 24x7 world, applications have long running sessions while the DBA modifies

the database to create and drop indexes, collect statistics, and alter tables, when prepared statements still refer to the database objects. With subsequent execution of the statement that uses a modified database object, prior to IDS 11, IDS returned the -710 error and the statement must be reprepared. This put the burden on the application developer to handle the error, reprepare the statement, and execute, which involves another interaction between the client and the server.

IDS 11 detects these situations and automatically reprepares the statement that the application trying to execute. Therefore, the client is unaware this is happening and continues to run smoothly, except for the extra time in response because of the time taken for reprepare. This helps long-running applications run smoothly and takes advantage of new statistics or indexes created by the DBA. Because this is automatic and transparent to the application, developers do not need to write code to detect the exception and reprepare manually. Because this feature is implemented within the database server, it is available in all APIs, such as embedded SQL, ODBC, and JDBC. Prior to IDS 11, Stored Procedures Language (SPL) stored procedures recognized these situations and automatically recompiled the statements, and that continues to happen.

Altering the table, by adding or dropping a column, modifying the type of column, altering the permissions on the table, granting or revoking permissions on the table, creating or dropping indexes, and updating statistics on the table require reprepare of the statements referring to that table.

The most common scenario is running UPDATE STATISTICS to collect statistics on a column. UPDATE STATISTICS on a column or a table, low, medium, and high, causes recompilation. This check is done at the table level and not at the column level. If you collect statistics on column `t1.a`, statements referring to just `t1.b` are recompiled as well.

Another common scenario is creating indexes ONLINE. This operation does not require an exclusive lock on the table for the duration of the index creation. The index does the dirty read of the table and then uses the logical log to apply any changes to the table during index creation. In IDS 11, we collect statistics on the leading key of the index and the optimizer immediately starts considering this index for subsequent query optimizations. After the index creation is complete, statements referencing the table are automatically recompiled.

When IDS prepares the statement the first time, it saves the OPTCOMPIND, pdq priority, and so forth and reuses them during automatic reprepare so the plan is created under similar conditions as when first created. Of course, it does use any new statistics and indexes available. So, of course, changes to the schema, such as dropping a column being referenced by prepare statements, results in an error during repreparation.

Consider the following example:

```
Create table customers (cid int primary key, fname varchar(32),
lname varchar(32), zip char(5));
Create table shipping(orderid  int, cid int, item_id int,
shipping_zip char(5), foreign key (cid) references customers(cid));

--  load some data

Statement s1: SELECT s.orderid, c.cid, c.lname, c.zip
  FROM shipping s, customers c
  WHERE s.shipping_zip = c.zip and lname like 'jo%';
```

Table 9-5 reflects the statement repreparation caused by the index creation.

Table 9-5 Repreparation results

Session 1	Session 2
Prepare the statement s1;	
declare cursor on s1; -- creates and uses the FirstPlan below.	
open cursor;	
fetch	
fetch	CREATE INDEX ONLINE icln ON customers(lname)
fetch	-- continuing with create index
fetch	-- continuing with create index
fetch	-- continuing with create index
fetch	-- continuing with create index
close	-- continuing with create index
do something else	-- complete the index creation
do something else	
fetch	
fetch	
…	

Session 1	Session 2
…	

Note: The IDS 11 optimizer recognizes the index you create immediately, eliminating the need to do update statistics low after creating the index.

Configuration for auto reprepare

Recognition of the need for reprepareation and automatic reprepareation of statements is the default behavior of IDS 11. You have the option to turn the feature on or off for the whole system or selectively for each session. Setting the **onconfig** variable AUTO_REPREPARE to 0 disables the feature and IDS behaves as in prior releases, returning a -710 error when the statement references any object that has been modified since the prepare. Setting AUTO_REPREPARE to 1 enables the feature, and it is the default behavior.

Separately, in each session, you can enable or disable this feature explicitly. To enable auto reprepare, use either of the following statements:

```
SET ENVIRONMENT  IFX_AUTO_REPREPARE  1;
SET ENVIRONMENT  IFX_AUTO_REPREPARE  "on";
```

To disable auto reprepare, use either of the following statements:

```
SET ENVIRONMENT  IFX_AUTO_REPREPARE  0;
SET ENVIRONMENT  IFX_AUTO_REPREPARE  "off";
```

9.9 Named Parameters support for JDBC

IDS 11 supports *Named Parameters*. This makes JDBC application code easier to write and more readable. Each function parameter has a name and a datatype. Prior to this release of IDS, JDBC was only able to bind values to these parameters by their position in the parameter signature. In IDS 11, parameter binding can happen by the implicit position of the function signature or by explicitly naming the parameter and the value. Consider the following sample code:

```
CREATE FUNCTION order_item(cust_id int,
                item_id int,
                count int,
                billing_addr varchar(64),
                billing_zip int,
shipping_addr varchar(64),
                shipping_zip int) return int status;
```

```
INSERT INTO order_tab(cust_id, item_id, count);
INSERT INTO billing_tab(cust_id, , billing_addr, billing_zip);
INSERT INTO shipping_tab(cust_id, , shipping_addr, shipping_zip);
Return 1;
END FUNCTION;
```

There are two ways to bind values to parameters:

- Implicit positional binding:

```
execute function order_item(5739, 8294, 5, "4100 Bohannon Dr.", 94025,
"345, university ave.", 94303);
```

- Explicit parameter naming:

```
execute function order_item(cust_id=5739,
    count=5,
    item_id=8294,
    shipping_addr="345, University ave.",
    shipping_zip=94303,
    billing_addr="4100 Bohannon Dr.",
    billing_zip=94025);
```

In the first case, you have to look up the signature to obtain the values to use in parameter mapping. It is easier when you specify the parameter names during invocation. The advantage of *Named Parameters* becomes obvious when you see its usage in a JDBC program.

This is an example of implicit positional binding:

```
CallableStatement cstmt = con.prepareCall("call order_item(?, ?, ?, ?, ?,
?, ?)");

// Set parameters (positional notation)
cstmt.setInt(1, 5739 );
cstmt.setInt(2, 8294);
cstmt.setString(6, "345, University ave.");
cstmt.setInt(7, 94303);
cstmt.setString(4,"4100 Bohannon Dr.");
cstmt.setInt(5, 94025);
cstmt.setInt(3,5);

// Execute
cstmt.execute();
```

Named parameters help you avoid looking up the function signature while writing code, which makes it less error prone and makes the code easier to read.

Here is the same code rewritten using named parameter notation:

```
// Set parameters (named notation)
cstmt.setInt("cust_id", 5739 );
```

```
cstmt.setInt("item_id", 8294);
cstmt.setString("shipping_addr", "345, University ave.");
cstmt.setInt("shipping_zip", 94303);
cstmt.setString("billing_addr","4100 Bohannon Dr.");
cstmt.setInt("billing_zip", 94025);
cstmt.setInt("count", 5);

// Execute
cstmt.execute();
```

Named notation self-documents the code through the obvious assignment of the parameter and its value. Moreover, you can define the used parameters in any order and omit parameters that have default values.

9.10 Full support for subqueries

In this section, we discuss the support for subqueries in the FROM clause (derived tables or table expressions).

The table reference in the FROM clause of a SELECT statement can be a table, a view, a table function (iterator functions), or collection-derived tables (CDT). Result sets from CDTs and table functions are treated as a transient table within the scope of the query. This transient is referred to as a *derived table* or a *table expression,* as depicted in Example 9-7.

Example 9-7 Derived table or table expression

```
SELECT *
FROM (SELECT tab1.a, tab2.x, tab2.y
          FROM tab1, tab2
          WHERE tab1.a = tab2.z
          ORDER BY tab1.b ) vt(va, vb, vc),
        emptab
WHERE  vt.va = emptab.id;
```

This approach is more flexible than creating views or temporary tables, inserting data into them and using them in a query. These subqueries in the FROM clause can do most of the things that a normal query does, including aggregation, pagination, sorting, and grouping. The IDS optimizer treats these queries similar to queries in views. It tries to fold the subquery into the parent query without changing the meaning or affecting the output. Failing that, the result set is materialized into a temporary table. After you have constructed the subquery, its table, and column reference, you can use it in all syntactical constructs, such as ANSI joins, UNION, UNION ALL, and so forth. In prior releases, this required using a collection-derived table (CDT), which required TABLE and MULTISET

keywords. This feature enables you to write and generate SQL-standard compliant syntax. There are samples depicted in Example 9-8.

Example 9-8 Subquery examples

```
-- CDT syntax
SELECT SUM(VC1) AS SUM_VC1, VC2
FROM TABLE (MULTISET(SELECT C1, C2 FROM T1 )) AS VTAB(VC1, VC2)
GROUP BY VC2;

     -- New syntax
SELECT SUM(VC1) AS SUM_VC1, VC2
FROM (SELECT C1, C2 FROM T1 ) AS VTAB(VC1, VC2)
GROUP BY VC2;

   -- New syntax
SELECT * FROM
( (SELECT C1,C2 FROM T3) AS VT3(V31,V32)
 LEFT OUTER JOIN
        ( (SELECT C1,C2 FROM T1) AS VT1(VC1,VC2)
        LEFT OUTER JOIN
        (SELECT C1,C2 FROM T2) AS VT2(VC3,VC4)
        ON VT1.VC1 = VT2.VC3)
ON VT3.V31 = VT2.VC3);

              -- New syntax
              SELECT *
              FROM table(foo(5)) AS vt(a), tab1 t
              WHERE vt.a = t.x;

-- Now the Function keyword is optional when you use the treat result from
function as a resultset.
-- In IDS 11, these functions can return a result set (table functions or
iterator functions) or simply return a single value (scalar function.)

SELECT * FROM TABLE(MYFUNC(9)) AS VX(XA);
SELECT * FROM TABLE(FUNCTION MYFUNC(9)) AS VX(VA);
```

9.11 Enhancements to distributed queries

Each IDS database server can have multiple databases. And, there can be multiple IDS instances, each with multiple databases. A query across multiple databases on the same IDS instance is called a *cross-database query.* A query across databases from multiple IDS instances is called a *cross-server* or *distributed query.* IDS 10 enabled cross-database queries for extended types

(Boolean, lvarchar, BLOB, and CLOB), distinct types, and User-defined types (UDTs) that are explicit casts of one of the supported types.

IDS 11 supports Boolean, lvarchar, and distinct types of all basic types, lvarchar, and Boolean in distributed queries (cross-server). It also supports C and Java UDRs, in addition to SPL procedures, in distributed queries.

Figure 9-7 depicts a configuration to use for subsequent distributed query examples.

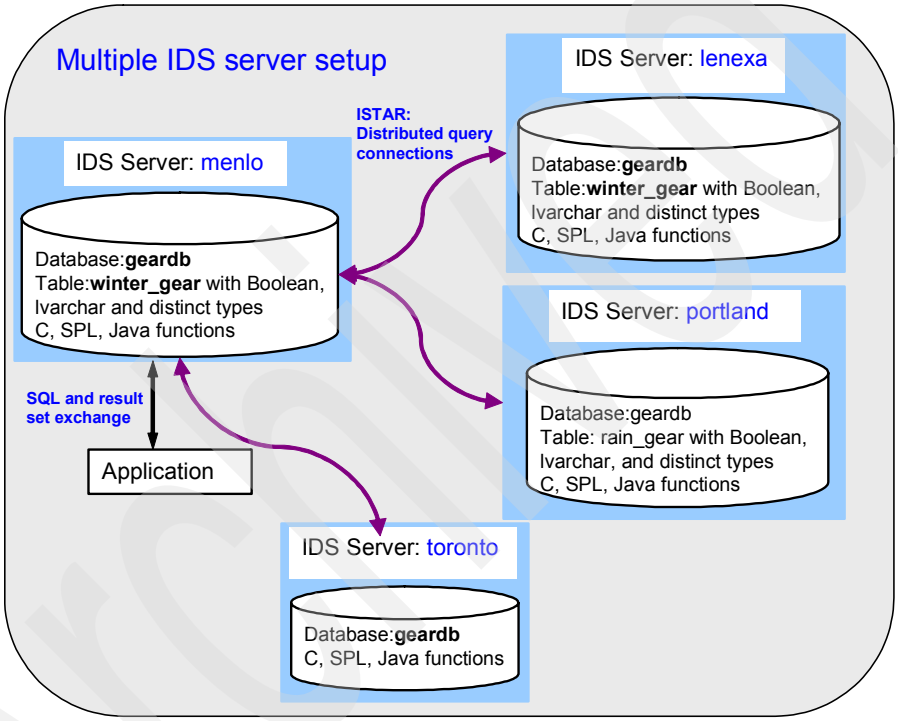


Figure 9-7 Distributed database

Example 9-9 on page 277 depicts querying for Boolean and lvarchar data types in a distributed query using the configuration depicted in Figure 9-7.

Example 9-9 Distributed query

```
    Connect to portland:

    CREATE TABLE rain_gear(partner  int, active Boolean, desc
lvarchar(4096));

    Connect to lenexa:
```

```
   CREATE TABLE winter_gear(partner  int, active Boolean, desc
lvarchar(4096));

   Connect to Menlo:
   CREATE TABLE sunny_gear(partner int, active Boolean, desc
lvarchar(4096));

   -- Select active partners at (Menlo, Lenexa and Portland)
   -- Access boolean and lvarchar columns for remote IDS instances.
   SELECT x.partner, x.desc, y.desc, z.desc
   FROM geardb@menlo:sunny_gear x,
        geardb@portland:rain_gear y,
        geardb@lenexa:winter_gear z
   WHERE x.partner = y.partner and
         x.partner = z.partner and
         x.active = 'T' and
         y.active = 'T' and
         z.active = 'T'
   ORDER BY x.partner;
```

Example 9-10 demonstrates distributed queries using distinct types (distinct types of basic types, Boolean, or lvarchar).

Example 9-10 Distributed queries using distinct types

```
Connect to members@portland:

         -- create two distinct types pound and kilos
   CREATE distinct type pound as float;
   CREATE distinct type kilos as float;

   -- create the cast function to convert from pound to kilos
   CREATE function ret_kilo(p pound) returns kilos;
   define x float;
   let x = p::float / 2.2;
   return x::kilos;
   end function;

   -- create the cast function kilos to pound
   CREATE function ret_pound(k kilos) returns pound;
   define x float;
   let x = k::float * 2.2;
   return x::pound;
   end function;

   -- create the casts so kilos and pounds convert to each other.
   create implicit cast (pound as kilos with ret_kilo);
   create explicit cast (kilos as pound with ret_pound);
```

```
-- Use the pound type.
CREATE table members (age int, name varchar(128), weight pound);

Connect to members@Toronto:

-- create the types, support functions and casts in this instance.
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);

CREATE table members (age int, name varchar(128), weight kilos);

    Connect to members@Menlo:

-- create the types, support functions and casts in this instance.
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);
```

```
   -- select the Portland members weighing less than  Toronto members of
the same age.
   select x.name, x.weight
   FROM members@portland:members x
         members@toronto:members y
   WHERE x.age = y.age and
         x.weight < y.weight    -- compares pounds to kilos.
   group by x.weight;
```

Similarly, distributed queries in IDS 11 can invoke C and Java UDRs in remote database servers. Prior versions of IDS allowed only SPL procedure invocation across database servers. Because IDS 11 has enabled the use of lvarchar, boolean and distinct types in cross-server queries, these types can be used in parameters and return values in cross-server invocation.

Consider this example:

```
From Menlo:
Select x.name, x.weight, x.height,
members@portland:body_mass_index(x.weight, x.height) as bmi        FROM
members@portland:members x WHERE x.id = 1234;
```

9.12 Trigger enhancements

A *trigger* is a database mechanism that automatically executes a set of SQL statements when a certain event occurs, such as an INSERT, UPDATE, DELETE, or SELECT on a table or a view. Each trigger can specify a set of statements to execute before the event, for each row affected, and at the end. IDS 11 enhances two aspects of triggers.

First, applications can now create multiple triggers for the same event. And second, a new type of user-defined routine that is called a *trigger routine*, which can be invoked from the FOR EACH ROW section of the triggered action and can access and modify values in the row. These SPL routines can apply procedural logic of the SPL language in operations on OLD and NEW column values in the rows processed by the triggered actions.

Multiple INSERT and DELETE triggers

Triggers are sets of SQL statements executed when a specific SQL operation, such as INSERT, UPDATE, DELETE or SELECT, is performed on a table or a view. In prior releases, IDS allowed an INSERT and a DELETE trigger per table. UPDATE and SELECT triggers can be for a complete table or selected columns, but only one trigger can be defined on a column.

IDS 11 allows creation of multiple INSERT and DELETE triggers on the same table and allows creation of multiple UPDATE and SELECT triggers on the same column. For each event, all available and applicable triggers are executed without a predetermined order, which is depicted in Figure 9-8.

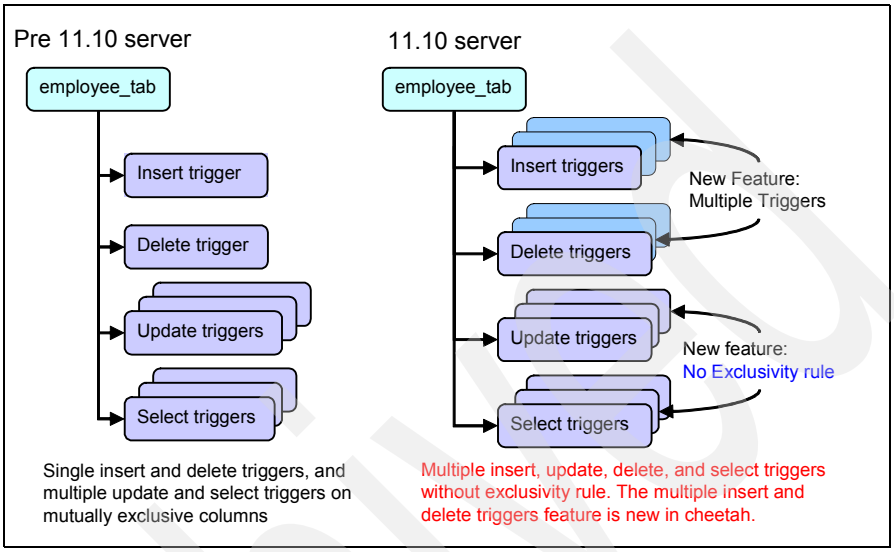


Figure 9-8 Multiple insert and delete triggers

Access to OLD and NEW row values:

Each trigger can create statements to execute at three events:

- Before the statement is executed: *Before* the trigger action
- After the statement is executed: *After* the trigger action
- For each row affected by the statement (*after the row has been affected*):
  - FOR EACH ROW (FER) trigger action

The FER triggered-action clause can access applicable OLD and NEW version values for the row. DELETE, UPDATE, and SELECT statements have OLD rows, while INSERT and UPDATE statements have NEW rows. UPDATE and INSERT triggers can determine the eventual value inserted or updated using the EXECUTE [PROCEDURE|FUNCTION] … INTO column-name feature of IDS.

INSTEAD OF triggers are created on views for INSERT, UPDATE, or DELETE operations. For views that can be updated, INSTEAD OF triggers are executed on DML operations on INSTEAD OF triggers on base tables.

IDS 11 simplifies the access to the OLD and NEW trigger-correlated variables (values in the affected row) in the SPL procedures invoked by FER trigger-action

clauses. After you declare that the procedure is attached to a table, the statements in the procedure can directly access trigger-correlated variables and modify appropriate values through LET assignment statements. These procedures have all the functionality of a normal procedure.

Example 9-11 depicts several examples of the use of triggers.

Example 9-11 Triggers

```
create table tab1 (col1 int,col2 int);
create table tab2 (col1 int);
create table temptab1 (old_col1 int, new_col1 int, old_col2 int, new_col2
int);
/*
 This procedure is invoked from the INSERT trigger in this example.

 This function also illustrates 4 new functions in IDS 11:
INSERTING will return true if the procedure is called from the For Each Row
action of the INSERT trigger. This procedure can also be called from other
trigger action statements: UPDATE, SELECT, DELETE.  UPDATING, SELECTING and
DELETING will be true when the procedure is invoked from the respective
trigger action.

*/
create procedure proc1()
referencing OLD as o NEW as n for tab1; -- new syntax.

if (INSERTING) then  -- INSERTING new boolean function
        n.col1 = n.col1 + 1;   -- You can modify the new values.
        insert into temptab1 values(0,n.col1,1,n.col2);
end if

if (UPDATING) then  -- UPDATING new boolean function
        insert into temptab1 values(o.col1,n.col1,o.col2,n.col2);
end if

if (SELECTING) then  -- SELECTING new boolean function
                -- you can access relevant old and new values.
        insert into temptab1 values (o.col1,0,o.col2,0);
end if
if (DELETING) then   -- DELETING new boolean function
        delete from temptab1 where temptab1.col1 = o.col1;
end if

end procedure;

-- This procedure automatically increments tab2.col1 by 10%.
create procedure proc2()
referencing OLD as o NEW as n for tab2
```

```
returning int;

LET n.col1 = n.col1 * 1.1 ; -- increment the inserted value 10%

end procedure;

-- use the proc1 inside for each row trigger.
create trigger ins_trig_tab1 INSERT on tab1 referencing new as post
for each row(execute procedure proc1() with trigger references);

-- use the proc1 inside for each row trigger.
create trigger ins_trig_tab2 INSERT on tab2 referencing new as n
for each row (execute procedure proc2() with trigger references);

insert into tab1 values (111,222);

The above statement will execute ins_trigger trig_tab1 and therefore will
execute procedure proc1().  The procedure will increment the value of col1
by 1.  So, the value inserted will be (112, 222).

insert into tab2 values (100);

The above statement will execute ins_trigger trig_tab2 and therefore will
execute procedure proc2().  The procedure will increment the value of col1
by 10%.  So, the value inserted into tab2 will be 110.
```

A procedures created with a REFERENCING clause can only be called from a FOR EACH ROW trigger and has to be executed using the WITH TRIGGER REFERENCES modifier to execute procedure statement. IDS raises an error for all other forms of invocation.

9.13 Index self-join access method

Traditionally, an index scan allows you to scan a single range (based on the start/stop key) of an index. The *index self-join access method* lets you scan many mini-ranges instead of a large single range, based on filters on non-leading keys of an index.

The index self-join is a new type of index scan where the table is logically joined to itself, such that for each unique combination of the leading key columns of an index, additional filters on non-leading key columns of the index are used to perform a more efficient mini-index scan. Results from multiple mini-index scans are then combined to generate the result set of the query.

Figure 9-9 is a before and after illustration of how a sample query is handled. This is the example query:

```
SELECT * FROM tab
WHERE c1 >= 1 and c1 <= 3 and c2 >= 10 and c2 <= 11 and c3 >= 100 and c3 <=
102
```

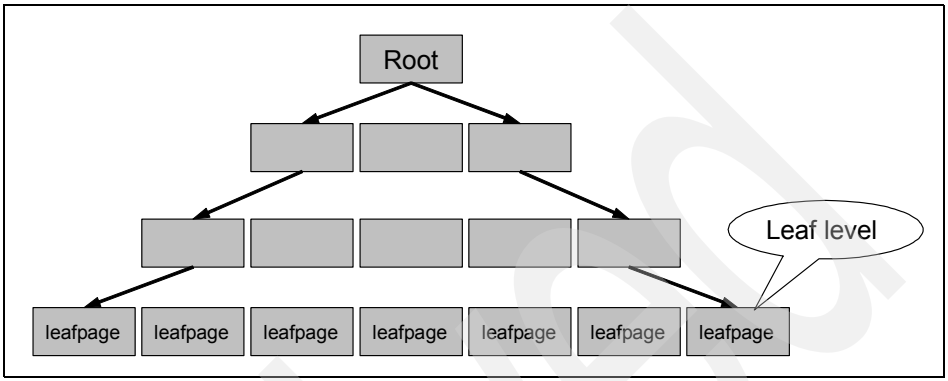


Figure 9-9 View of index on C1, C2, and C3

In prior releases of IDS, you were only allowed to use filters on c1 (c1 >= 1 and c1 <= 3) for positioning of the index scan, as depicted in Figure 9-10 on page 284.

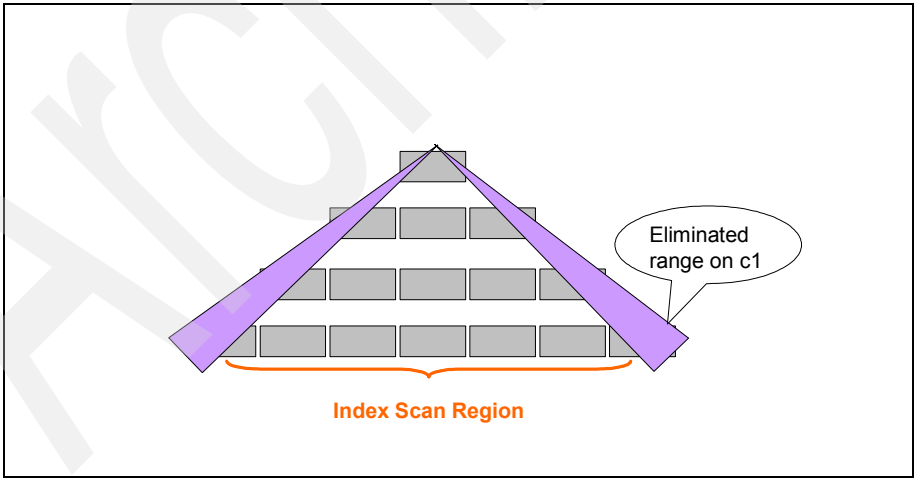


Figure 9-10 Index scan in prior releases of IDS

With this feature, we can use filters on c2 and c3 for positioning of the index scan, which allows us to skip unnecessary index keys at the two ends of the

index, and IDS only scans pockets of the index that are relevant, therefore, avoiding scanning a large portion of the index. This strategy improves the query performance by reducing the portion of the index IDS has to scan.

In Figure 9-11, we depict the scenario where:

```
Lead Keys: C1 and C2
Lower filter C1 = C1 and C2 = C2 and C3 >= 100
Upper filter C3 =< 102
```

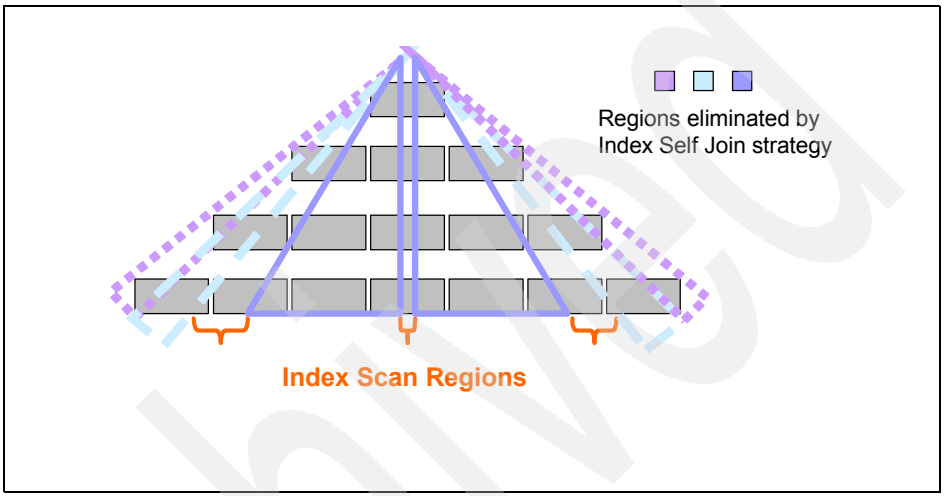


Figure 9-11 Index self-join

Example 9-12 shows the optimizer choosing the index self-join for a query.

Example 9-12 Index self-join

```
QUERY:
------
select  c1, c7, c11 from tab1
where c1 between 15 and 18 and c7 between '000000' and 'SSSSSS'
and c3 > 2356 and c3 <= 2400
order by 1 desc, 2 desc

Estimated Cost: 38
Estimated # of Rows Returned: 2

 1) sqlqa.tab1: INDEX PATH

   (1) Index Keys: c1 c7 c3 c6   (Key-First)  (Serial, fragments: ALL)
       Index Self Join Keys (c1 c7 )
         Lower bound: sqlqa.tab1.c1 <= 18 AND (sqlqa.tab1.c7 <= 'SSSSSS' )
         Upper bound: sqlqa.tab1.c1 >= 15 AND (sqlqa.tab1.c7 >= '000000' )
```

```
        Lower Index Filter: (sqlqa.tab1.c1 = sqlqa.tab1.c1 AND
sqlqa.tab1.c7 = sqlqa.tab1.c7 ) AND sqlqa.tab1.c3 <= 2400
        Upper Index Filter: sqlqa.tab1.c3 > 2356
        Index Key Filters:  (sqlqa.tab1.c7 >= '000000' ) AND
                            (sqlqa.tab1.c7 <= 'SSSSSS' )
```

9.14 Optimizer directives

In this section, we discuss optimizer directives in ANSI-compliant joined queries.

Optimizer directives influence the optimizer to choose better access paths and join orders for the query execution plan than a plan based only on the available data distribution. Directives help DBAs and developers gain better control over the query plan. IDS 11 allows common directives for ANSIJOIN queries. The optimizer attempts to use these directives in appropriate cases. When it cannot, either because an access method is invalid in certain scenarios or cannot be used and still return correct results, the optimizer prints the directives not used in the explain file.

Previously, ANSI join queries allowed the EXPLAIN, AVOID_EXECUTE, and FIRST_ROWS/ALL_ROWS directives. In this release, other directives, such as table access methods (FULL, INDEX, and so forth), join methods (USE_HASH, USE_NL, and so forth), and join order (ORDERED) are allowed in an ANSI join query. For example, --+, /*+, and {+ are the escape sequences for directives. We describe directives in Example 9-13.

Example 9-13 Directives

```
Example1 - Using directives:
   select --+ INDEX(mytab, myidx)
   a.col1, b.coly
     from tab1 a, tab2 b
     where a.col1 = 1234 and a.col1 = b.colx;

Example1 - Reading directives in Explain file:

select --+ FULL(t2), INDEX(t1,t1i1), ORDERED
* from t1 left outer join t2 on (t1.c1=t2.c1 and t1.c1=2)
where t2.c1 is not null

DIRECTIVES FOLLOWED:
FULL ( t2 )
INDEX ( t1 t1i1 )
ORDERED
DIRECTIVES NOT FOLLOWED:
```

```
Estimated Cost: 7
Estimated # of Rows Returned: 1

  1) sqlqa.t1: INDEX PATH

        Filters: sqlqa.t1.c1 = 2

    (1) Index Keys: c2   (Serial, fragments: ALL)

  2) sqlqa.t2: SEQUENTIAL SCAN

        Filters:
        Table Scan Filters: sqlqa.t2.c1 = 2

DYNAMIC HASH JOIN
    Dynamic Hash Filters: sqlqa.t1.c1 = sqlqa.t2.c1
```

9.15 Statistics collection and query explain

In IDS 11, statistics collection has been improved and there is a query explain file. The time when a user ran UPDATE STATISTICS LOW on a table is now stored in the systables system catalog table in the ustlowts column ("informix".SYSTABLES.ustlowts), which has a data type of datetime year to fraction(5), as depicted in Example 9-14.

Example 9-14 Update statistics low

```
> select a.tabname, a.ustlowts from systables a where tabname = 't1';

tabname          t1
ustlowts         2007-02-05 18:16:29.00000

1 row(s) retrieved.
```

In IDS 11, the CREATE INDEX operation automatically updates the statistics about the table and creates distributions for the leading key of the index. Applications no longer need to run UPDATE STATISTICS LOW on the table for the optimizer to start considering the newly-created index. It is now done automatically. For the B-tree indexes on basic IDS types, IDS exploits the sort done during index creation to create the distribution for the leading column of the key.

For example, when you create a composite index using the key (c1, c2, c3), IDS automatically creates distribution on c1, but no distribution is automatically

created for c2 and c3. IDS uses an innovative way to implicitly create this distribution for fragmented indexes as well. Because creation of distributions requires additional memory, make sure to tune the memory parameters used in update statistics execution. See the developerWorks article, “Tuning UPDATE STATISTICS,” at:

http://www-128.ibm.com/developerworks/db2/zones/informix/library/techarticle/miller/0203miller.html

The process to tune memory parameters is depicted in Example 9-15.

Example 9-15 Tune memory parameters

```
create table t1(a int, b varchar(32));
-- load data
set explain on;
create index index_t1_ab on t1(a,b);

Contents of sqexplain.out:

CREATE INDEX:
=============

Index:          index_t1_ab on keshav.t1
STATISTICS CREATED AUTOMATICALLY:
Column Distribution for:                keshav.t1.a
Mode:           MEDIUM
Number of Bins:       2 Bin size:    38.0
Sort data:          0.0 MB
Completed building distribution in:     0 minutes 0 seconds
```

Now "informix".SYSDISTRIB has four new columns, as depicted in Table 9-6.

Table 9-6 New columns

Column name	Column type	Comments
smplsize	float	Sampling size specified by the user; 0.0 when unspecified. A value between 0.0 and 1.0 is the sampling percentage of rows and a value above 1 is the number of rows to be sampled.
rowssmpld	float	Actual number of rows sampled to create the distribution.
constr_time	datetime year to fraction(5)	Time when the distribution was created.
ustnrows	float	Number of rows in the table when the distribution was created.

Explain file improvements

You can now set the location of the explain file in the SET EXPLAIN statement; it overrides the default location and file name. For example:

```
SET EXPLAIN FILE TO "/tmp/cheetah/myexplainfile.out';
```

You can dynamically enable EXPLAIN on any session using the **onmode -Y** command, and you can specify the location of the explain file as well.

Enable the explain for session 38 with either of the following commands:

```
onmode -Y   42 1
onmode -Y   38 1 /tmp/cheetah/explain_ses38.out
```

The first command lets the system choose the location and name of the file. When IDS starts writing to the explain file, it also puts a diagnostic message in the online.log file, which makes it easy to find the file:

```
20:09:55  Explain file for session 42: /home/keshav/sqexplain.out.42
20:26:43  Explain file for session 38: /tmp/cheetah/explain_ses38.out
```

When the query is executed, the explain file includes the estimated and actual cardinalities for each table in the query or subquery, as depicted in Example 9-16 on page 289.

Example 9-16 Explain improvements

```
QUERY:
------
select a from tab1 where a in (select x from tab2 where tab2.x < 10)

Estimated Cost: 46
Estimated # of Rows Returned: 677

  1) keshav.tab1: INDEX PATH

    (1) Index Keys: a b   (Key-Only)  (Serial, fragments: ALL)
        Lower Index Filter: keshav.tab1.a = ANY <subquery>

    Subquery:
    ---------
    Estimated Cost: 20
    Estimated # of Rows Returned: 87

      1) keshav.tab2: SEQUENTIAL SCAN

            Filters: keshav.tab2.x < 10

Query statistics:
-----------------
```

```
  Table map :
  ----------------------------
  Internal name     Table name
  ----------------------------
  t1                tab1

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t1     584        677       584        00:00:00   47

Subquery statistics:
--------------------

  Table map :
  ----------------------------
  Internal name     Table name
  ----------------------------
  t1                tab2

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t1     87         87        500        00:00:00   20

  type     rows_sort  est_rows  rows_cons  time
  ---------------------------------------------
  sort     9          87        87         00:00:00
```

You can enable or disable printing of these statistics by resetting the value of the EXPLAIN_STAT configuration parameter with the **onmode** command:

```
onmode -wm "EXPLAIN_STAT=1"
onmode -wm "EXPLAIN_STAT=0"
```

The **onmode -wm** command changes the preference in memory only, and the setting is lost when the server is restarted. The **onmode -wf** command changes the preference in both memory and in the onconfig file.

The **onmode -Y** command enables you to dynamically print the query plan on a session. As an example:

```
onmode -Y <ses_id> 0      -- off no query plan will be printed.
1       -- plan + statistics
2       -- plan only
```

9.16 Stored procedure language (SPL) enhancements

IDS clients use SPL procedures extensively in the applications, both to write complex application logic into the database and to create an abstraction layer for applications to simply execute them to perform respective tasks. We have seen clients using thousands of stored procedures and using them almost exclusively for database access. The situation is similar for clients using other database servers. So, while we continue to enhance the stored procedure language for IDS applications, we also make it easier to migrate applications and their stored procedures to port to IDS.

Here are the new SPL statements and syntaxes in IDS 11

First is the GOTO statement. This is not the most popular control statement, but it does help programs to break out of tight situations.

In SPL, you can define a label anywhere a new statement can be started. The label name is prefixed and suffixed with two angle brackets: <<mylabel>>. So, the GOTO statement is simply: GOTO mylabel. The scope of the label is the complete procedure. The label name has to be unique within a procedure, outside the exception block. So the GOTO can jump into one of the labels within the procedure, and you cannot use GOTO with an exception block. The rules for the label names are same as the identifiers in IDS- table names and column names. And, the DELIMIDENT has no effect on the label names. A simple depiction of the GOTO statement is illustrated in Example 9-17 on page 291.

Example 9-17 Creating a GOTO statement

```
create procedure simple_goto(a int) returns int;

   if a < 50 then
                goto lessval;  -- jump to lessval.
   end if;

   return a * a;  -- return something

   <<lessval>>   -- define the label lessval
   return a * -1;

end procedure ;
```

Next, we describe the LOOP statements introduced in IDS. You can use GOTO statements within a LOOP statement or to break of one or more LOOP statements.

LOOP and EXIT statements

In its simplest form, LOOP… END LOOP is an infinite loop, unless you use EXIT or GOTO to break out of it. A simple EXIT and EXIT LOOP break control out of the immediate loop. Use EXIT LOOP with WHEN to provide a condition on which to break out of the loop. The EXIT can be used to break out of the current loop or a specific labeled loop.

The examples here use simple arithmetic statements so we can be brief and focus on the control statements. You can use these with other SPL and SQL statements within SPL procedures, as depicted in Example 9-18.

Example 9-18 Loop and Exit statements

```
create procedure plusnum(a int) returns int;
define x int;

let x = a;

   loop   -- start the loop
       let x = x + 1;

       IF x < 0 THEN
            CONTINUE; -- increment until +ve number.
       ELIF x < 1000 THEN
           return x with resume;   -- return and come back.
       END IF;

       EXIT LOOP WHEN x = 1000; -- exit loop at 1000.
   END LOOP;

end procedure;
```

The following loop statement block can be used with both FOR and WHILE.

```
create procedure sumto_a(a int) returns int;
define x int;
define total int;


   let total = 0;
   for x in (1 TO a) LOOP   -- start the loop
   total = total + x;
   END LOOP;

end procedure;
```

You can also label your LOOP, FOR, and WHILE loops. And, you can also exit from these loops conditionally as shown:

```
-- returns factorials for each number from 1 to a.
create procedure facto(a int) returns int as x, int as factx;
define x,y, factx int;
define total int;

        let total = 0;
        let x = 1;

<<main_loop>>
WHILE (x <= a) LOOP   -- start the loop
      let factx = 1;
      <<facto_loop>>
      FOR y IN (1 TO x) LOOP
          -- Abruptly stop at 10.
          EXIT main_loop WHEN x = 10;
          LET factx = factx * y;
      END LOOP facto_loop;
      RETURN x, factx WITH RESUME;
      LET X = x+1;
END LOOP  main_loop;

end procedure;
```

The following are example runs:

```
> execute function facto(3);

          x       factx

          1           1
          2           2
          3           6

3 row(s) retrieved.


> execute function facto(10);

             x       factx

             1           1
             2           2
             3           6
             4          24
             5         120
             6         720
             7        5040
             8       40320
             9      362880
```

```
9 row(s) retrieved.


> execute function facto(20);

             x       factx

             1           1
             2           2
             3           6
             4          24
             5         120
             6         720
             7        5040
             8       40320
             9      362880

9 row(s) retrieved.

>
```

9.17 SYSDBOPEN() and SYSDBCLOSE() procedures

IDS 11 reserves two procedure names, *sysdbopen()* and *sysdbclose()*, which help you to set up the session environment for each database connection. The code in the sysdbopen() procedure can set the required environment (the session variables, data in temporary or permanent tables, and so forth using SET and SET ENVIRONMENT statements) before the application executes queries on the server.

Applications can query if and only if the sysdbopen() completes the execution successfully. So, while closing the database, the server simply executes the sysdbclose() procedure. The process to control general settings and log the entry is depicted in Example 9-19.

Example 9-19 Settings and logging the entry

```
create procedure public.sysdbopen();
           set pdqpriority 1;
           set environment optcompind '1';
           set isolation committed read;
           set environment uselastcommitted 'ALL';
           system "echo Logged IN: " || USER || "  " ||
                    CURRENT || "  >> /tmp/mylog";
```

```
end procedure;

-- log the entry.
create procedure public.sysdbclose();
    system "echo Logged OUT: " || USER || "  " ||
           CURRENT || "  >> /tmp/mylog";
end procedure;
```

The DBA can create these stored procedures, either common to all users or for individual users. The *sysdbopen* and *sysdbclose* procedure names are now reserved, and IDS gives an error if a non-DBA user tries to create a procedure with this name with or without any parameters. These are procedures that cannot return any values and therefore cannot be functions, but they can be created in all modes of IDS databases, logged (buffered and unbuffered logging), non-logged, and ANSI.

The public.sysdbopen() is executed for all users connecting to a database or when setting or changing the current database using the DATABASE statement. The public.sysdbclose() is executed for all users disconnecting from a database and then closing the database using CLOSE or changing the current database using the DATABASE statement when we implicitly close the current database and set a new one. With or without the public procedures, you can have these procedures for specific users, such as john.sysdbopen(), jack.sysdbclose(), and so forth. IDS executes the user's procedure if a procedure exists or else the public procedure if it exists.

Typically, in logged and non-logged databases, procedures with matching signatures cannot be created with distinct users. The procedures sysdbopen() and sysdbclose() are exceptions to this rule. This exception is only for the two procedures: sysdbopen() and sysdbclose().

You can use these procedures to set the environment before your application can issue any queries. That is, they can set up the environment, such as PDQPRIORITY, OPTCOMPIND, ISOLATION, or setup tables, temporary or permanent, for your sessions to use. Unlike a typical procedure session environment that you set up, a SYSDBOPEN() procedure remains in effect even after the procedure completes. And, because these procedures exist outside of application code, they can be used to tune or put in a session level change without recompiling and redeploying the application. With IDS 11, the DBA, and only the DBA, can disable execution of these procedures by setting the IFX_NODBPROC to 1 in the application environment.

9.18 Disabling logging for temporary tables

All DML operations on temporary tables create logs. Not only do these logs take logspace, but if you have enabled your system for high availability, they get transported to other systems, simply to be ignored.

To avoid this overhead, you can use the CREATE TEMP TABLE command and include the WITH NO LOG option. IDS 11 provides a method to disable logging on temporary tables by default, even when you do not use the WITH NO LOG modifier.

Caution: You cannot roll back any changes to the table if you do not have logging specified for it.

Logging can be disabled by either of the following two ways:

- ONCONFIG

```
TEMPTAB_NOLOG        1
```

- Dynamically

```
onmode -Wf  "TEMPTAB_NOLOG =1"
onmode -Wm "TEMPTAB_NOLOG =1"
```

Note however that -Wm changes the value and enables the behavior on the system immediately. And, -Wf does the same thing as -Wm and then writes the new value to the IDS configuration file.

9.19 New functions and expressions

IDS 11 has new functions and expressions and enhancements to several existing functions and expressions that enable you to manipulate data in new ways. The *SQL Syntax guide* has detailed explanations, nuances, and examples for each of these functions. Table 9-7 contains a list and a brief explanation of the new functions and expressions.

Table 9-7 New functions and expressions

Function	Description
ADD_MONTHS()	ADD_MONTHS(date/datetime expression, integer) Adds months to a date or datetime value or expression. The first argument is date or datetime expression and the second is an integer. Return type is same as first argument. add_months('4/16/2004', 20) returns: 12/16/2005 add_months(CURRENT, 3) returns: 2007-09-19 10:04:11.00000
ASCII	ASCII(character_expression) The ASCII function returns the decimal representation of the first character in a character string. ASCII('a') returns: 97 ASCII('Hello World!') returns: 72
Bitwise functions	BITAND(num1, num2) returns the bitwise ANDed value. BITOR(num1, num2) returns the bitwise ORed value. BITXOR(num1, num2) returns the bitwise XORed value. BITNOT(num1) returns the bitwise NOT value. BITANDNOT(num1, num2) is a short form for BITAND(arg1, BITNOT(arg2))
CEIL	CEIL(numerical_expession) Returns the DECIMAL(32) representation of the smallest integer that is greater than or equal to its single argument. FLOOR(numerical_expression) Returns the DECIMAL(32) representation of the largest integer that is smaller than or equal to its single argument. ceil(-54.232) returns -54 floor(-54.232) returns -55 ceil(54.232) returns 55 floor(54.232) returns 54

Function	Description
FORMAT_UNITS	FORMAT_UNITS(number, precision, units) Helps formatting of numbers in kilobytes to petabytes. Detailed explanation with examples is in *IDS SQL Syntax guide.* SELECT FORMAT_UNITS(SUM(chksize), 'P') SIZE, FORMAT_UNITS(SUM(nfree), 'p') FREE FROM syschunks; size 117 MB free 8.05 MB
LAST_DAY	LAST_DAY(date or datetime expression) Returns last day of the month in the argument. select TODAY as today, CURRENT as current, LAST_DAY(TODAY) as last_from_today, LAST_DAY(CURRENT) as last_from_current from systables where tabid = 1; today 06/19/2007 current 2007-06-19 10:23:01.000 last_from_today 06/30/2007 last_from_current 2007-06-30 10:23:01.00000
LTRIM	LTRIM(source_string, pad_string) Returns the source_string after removing specified leading pad characters from a string. LTRIM will remove leading blanks when you simply pass the source string. LTRIM('Hello Cheetah!', 'Hello ') returns: Cheetah!
MONTHS_BETWEEN	MONTHS_BETWEEN(date/datetime expr, date/datetime expr) Returns the difference between two date or datetime expressions in decimal, based on 31day months. select CURRENT, months_between(TODAY, LAST_DAY(CURRENT)) from systables where tabid = 1; (expression) (expression) 2007-06-19 10:51:57.000 -0.3694433243728

Function	Description
NEXT_DAY	Next_day(date or datetime expr, abbreviated day of the week) Returns the date or datetime for next day matching the second argument. execute function next_day(today, 'Mon') returns 06/25/2007 execute function next_day(current, 'Mon') returns 2007-06-25 6:57:52.00000
NULLIF	NULLIF(arg1, arg2) Returns NULL if arg1 and arg2 are equal, else returns arg1. If both are NULL - they will not be equal - but still returns NULL because arg1 is NULL.
TO_NUMBER	TO_NUMBER(character or numeric expression) converts a number or a character expression representing a number value to a DECIMAL.
Enhancements to existing functions	**Description**
TO_CHAR	TO_CHAR(numeric expression) In addition to existing functionality, in IDS 11, this function converts a number into a character string.
TRUNC and ROUND	These two functions can now take date and datetime expressions. *IDS Syntax guide* explains this in detail.
SYSDATE	Returns the same value as CURRENT datetime year to fraction(5)
POWER	Works same as POW() function.

10

Functional extensions to IDS

Extensions to the base functionality of a product is a prime direction for many software and platform vendors. Extension developers include vendors for Web servers, application servers, and application development platforms, such as Eclipse. It is an excellent way to extend the capabilities of a data server to better enable business solutions.

IDS was the first industrial-strength data server to include a comprehensive set of extensibility features that were above and beyond the base product. Over time, these features have been improved and have matured to provide a robust and flexible environment for functional extensibility.

For more detailed information about the benefits of building custom extensions to IDS, refer to Chapter 4, “Extending IDS for business advantages”, of the IBM Redbooks publication *Informix Dynamic Server V10 . . . Extended Functionality for Modern Business*, SG24-72999.

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 the application’s 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 that are available for a fee. In this chapter, we discuss the standard procedure for installing DataBlades and describe the DataBlade

modules currently available. Consult the most current release notice when you want a DataBlade to be sure that you are aware of all the most recent modules.

10.1 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. Any other DataBlade modules must first be installed in the data server. The installation process is described in detail in the manual *DataBlade Modules Installation and Registration Guide*, G251-2276-01. In general, the installation requires the following steps:

- Unload the files into a temporary directory. This might require the use of a utility, such as cpio, tar, or an uncompress utility, depending on the platform used.
- 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 is a new directory under $INFORMIXDIR/extend. The name reflects the DataBlade module and its version. For example, the current version of the spatial DataBlade module directory is named spatial.8.20.UC2.

You must register a DataBlade module 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 10-1 depicts the process for registering the spatial DataBlade module in a database called `demo`.

Example 10-1 DataBlade registration

```
informix@ibmswg01:~> blademgr
IDS 11>list demo
There are no modules registered in database demo.
IDS 11>show modules
8 DataBlade modules installed on server IDS 11:
             ifxrltree.2.00             mqblade.2.0
              Node.2.0 c                 LD.1.20.UC2
                bts. 1.00              ifxbuiltins.1.1
              binaryudt.1.0          spatial.8.21.UC1
                wfs.1.001UC1
A 'c' indicates the DataBlade module has client files.
If a module does not show up, check the prepare log.4
ids10>register spatial.8.21.UC1 demo
Register module spatial.8.21.UC1 into database demo? [Y/n]
Registering DataBlade module... (may take a while).
Module spatial.8.21.UC1 needs interfaces not registered in database demo.
The required interface is provided by the modules:
        1 - ifxrltree.2.00
Select the number of a module above to register, or N :- 1
```

```
Registering DataBlade module... (may take a while).
DataBlade ifxrltree.2.00 was successfully registered in database demo.
Registering DataBlade module... (may take a while).
DataBlade spatial.8.20.UC2 was successfully registered in database demo.
ids10>ids10>bye
Disconnecting...
informix@ibmswg01:~>
```

In this example, we started by executing the blade manager program. The prompt indicates with which instance we are working. This example uses the IDS 11 instance. The first command 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`** and provides a list of the DataBlade modules that are installed in the server under $INFORMIXDIR/extend. The names correspond to the directory names in the extend directory. The blade manager utility looks into the directories to make sure they are proper DataBlade directories. This means that other directories can exist under extend, but are not listed by the **`show modules`** command.

You register a DataBlade with the **`register`** command. Upon execution, it looks for dependencies 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 blade manager available on Windows.

10.2 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 11.10 currently defines multiple built-in DataBlades. In the following sections, we describe those built-in DataBlade modules.

10.2.1 Indexable binary data types

These new data types are found in binaryudt.1.0 in the extend directory. The data types added are an 18-byte fixed-length (binary18) data type and a variable length data type (binaryvar) of up to 255 bytes.

These data types allow you to store binary data in the database by using their ASCII representations. This can be useful with data, such as Universally Unique Identifiers (UUIDs). The DataBlade module also includes functions that operate on the binary data, such as bit_and(), bit_complement(), bit_or(), bit_xor(), LENGTH(), and OCTET_LENGTH(). It also includes support for the functions BDTRelease() and BTDTrace().

10.2.2 Basic text search

The Informix Dynamic Server has provided a chargeable text search DataBlade Module (Excalibur text) for many years. The goal of the basic text search (BTS) DataBlade module is to provide simple text search capabilities on ASCII text. It is located in bts.1.00 under the extend directory.

Before you can register the BTS DataBlade module, you must define a new type of Virtual Processor (bts) and create an external space to use to store the indexes. The Virtual Processor definition is:

```
VPCLASS bts, noyield, num=1
```

You use the **onspaces** command to create the external space. Here is an example of the creation of an external space:

```
onspaces -c -x bts_extspace -l
"$INFORMIXDIR/btsidx/bts_expspace_directory"
```

The BTS indexes are stored in a directory structure under the external space. The first directory level is the database name, and the second directory level is the index owner name. Under this second directory, you see the index file that has the name of the index.

You can create indexes on CLOB, BLOB, CHAR, LVARCHAR, and VARCHAR data types by using the BTS access method. The syntax is:

```
CREATE INDEX indexname
ON tablename(column_name operator_class)
USING bts(delete='deletion_mode')
IN extspace;
```

At this point, you can use the index in queries through the bts_contains() function. The search capability includes the use of wildcards, fuzzy searches,

proximity searches, stop words, and the use of logical operators (AND, OR, and NOT). For example:

```
SELECT id FROM product
WHERE bts_contains(brands, 'standard', score # real)
AND score > 70.0;
```

For more information, consult the *IBM Informix Built-in DataBlade Modules User's Guide*, G251-2269.

10.2.3 Large Object locator

This DataBlade module appears as LLD.1.20.UC2 in the extend directory. With this module, you can have a consistent interface if the design of the application calls for storing several of the large objects in the database (such as CLOBs or BLOBs) and other large objects outside of 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: You can use this interface 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 higher, 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.

10.2.4 The MQ DataBlade

The MQ DataBlade module, which is listed as mqblade.2.0 under the extend directory, is available on IDS 10.0 xC3 and higher. 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 supported with Linux 32-bit, Linux 64-bit on System p® machines, and Solaris 64-bit implementations. No new platforms were added with IDS 11.

The MQ DataBlade offers an interface between the IDS data server and the WebSphere MQ (WMQ) messaging products installed on the same machine as IDS. WMQ products are key components of the IBM enterprise service bus (ESB) to support the service-oriented architecture (SOA). WMQ enables the exchange of asynchronous messages in a distributed, heterogeneous environment.

The interface between IDS and WMQ is transactional and uses the two-phase commit protocol. This means that when a transaction that includes a message queue operation 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 can 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 and performed through function calls or even through a table interface. An application can read a message queue with a simple SQL statement such as:

```
SELECT * FROM vtiMQ;
```

Similarly, writing to a message queue can be as simple as using an INSERT statement:

```
INSERT INTO vtiMQ(msg) VALUES("<insert message here>");
```

Therefore, if the programmer knows how to execute SQL statements from an application, message queues can be easily manipulated. The MQ DataBlade module supports various ways to operate on message queues, such as read, receive, publish, and subscribe. In Table 10-1 on page 308, we list the functions available with the MQSeries® DataBlade, along with brief descriptions of those functions.

Table 10-1 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.
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 Virtual Table Interface (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 the *Built-In DataBlade Modules User's Guide*, G251-2770.

10.2.5 Node

The Node DataBlade module provides an indexable, hierarchically aware data type. It resides in Node.2.0 under the extend directory and provides the ability to track a hierarchy up to a depth of 64 levels.

You can see hierarchies everywhere, including types of hierarchies such as bills-of-material, management hierarchies, and even product hierarchies. For example, a company might classify its products as depicted in Figure 10-1 on page 309.

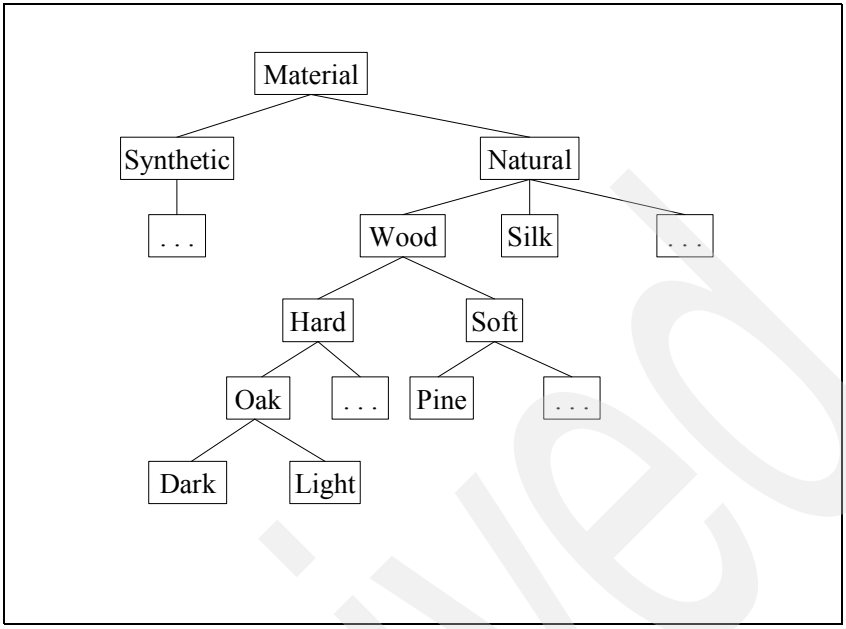


Figure 10-1 Example product hierarchy

By using the Node type, we can easily express queries that answer a question such as, “What natural materials do I have?” That can then easily lead to other queries, such as “What are the costs for a specific project if I use different natural materials?”

The ease of expressing these queries is one aspect of the advantage of using the Node type. In certain cases, it can even provide significant performance improvements.

You can see more examples of the use of the Node type in Chapter 2 of the book *Open-Source Components for Informix Dynamic Server 9.x*, Jacques Roy, William W. White, Jean T. Anderson, Paul G. Brown, 2002, ISBN 0-13-042827-2, from publisher Prentice Hall (Informix Press).

10.2.6 Web Feature Service

The Web Feature Service (WFS) DataBlade module complements the spatial and the geodetic DataBlade modules and cannot be used without one of them. Its role is to handle requests for geographical information from a Web server in a platform-independent manner.

The WFS DataBlade module is based on the transaction WFS specification from the Open Geospatial Consortium (OGC). This means that Web applications can take advantage of the high-performance implementation of spatial processing in IDS while using a standard interface. The WFS DataBlade module comes with a CGI Driver program that enables access to these features by Web applications.

10.3 Free-of-charge DataBlades

The Spatial DataBlade is currently the only free DataBlade module. It is not included with IDS, so you must download it separately from this Web site:

http://www14.software.ibm.com/webapp/download/search.jsp?go=y&rs=ifxsdb-hold&status=Hold&sb=ifxsdb

10.3.1 Spatial

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 latest version is 8.21.xC1. After you install it, it appears in the $INFORMIXDIR/extend directory with a name of spatial followed by the version number. For example, for a 32-bit UNIX version, it appears as spatial.8.21.UC1.

The spatial DataBlade implements a set of data types and functions that allow for the manipulation of spatial data within the database. With it, several business-related questions become easier to answer. As examples:

- Where are my stores located in relation to my distributors?
- How can I efficiently route my delivery trucks?
- How can I micro-market to customers fitting a particular profile near my worst performing store?
- How can I set insurance rates for customers that live near flood plains?
- Where are the parcels in the city that are impacted by a zoning change?
- Which bank branches do I keep after the merger based on my customers' locations (among other things)?

Locations and spatial objects are represented with new data types in the database, such as ST_Point, ST_LineString, and ST_Polygon. 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 SQL statement depicted in Example 10-2 on page 311 illustrates you can answer this question.

Example 10-2 SQL statement with spatial

```
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 three miles from the location that is passed as argument (:GPS_Loc).

The performance of this query can improve even more if it can 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 is depicted in Example 10-3.

Example 10-3 Index creation

```
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 we have mentioned and 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. Spatial information can 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 applications that must have a high level of precision over large distances, the geodetic DataBlade module is likely a better fit. This DataBlade takes into consideration the curvature of the earth to answer those specialized needs. We describe the Geodetic DataBlade module in 10.4.2, “Geodetic” on page 313.

10.4 Chargeable DataBlade modules

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.

10.4.1 Excalibur text search

People store descriptions, remark fields, and a variety of documents using different formats (such as PDF and WORD) in the database. A support organization is a simple example of this, because the support organization needs to store a description of their interactions with customers, including the problem description and the solution. Other examples include product descriptions, regulatory filings, news repositories, and so forth.

When people in an organization need to search the content of documents, they cannot depend simply on keywords. They need to use more complex search criteria. For example, how can you do this: Find all the documents that discuss using IDS with WAS CE.

This search causes numerous issues, such as:

- What if the document says Informix Dynamic Server rather than IDS or similarly, WebSphere Application Server Community Edition rather than WAS CE?
- 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 that 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.

After an Excalibur Text index is created, SQL queries can use it in statements, such as the statement depicted in Example 10-4.

Example 10-4 Excalibur text

```
SELECT id, description FROM repository
WHERE etx_contains(description,
ROW("Using IDS with WAS CE",
    "SEARCH_TYPE = PHRASE_APPROX"));
```

You can execute many more complex searches using this DataBlade module. Any company that needs these types of searches can greatly benefit from Excalibur Text.

10.4.2 Geodetic

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 can be used to solve queries that include the functions beyond(), inside(), intersect(), outside(), and within(). Consider the SQL statement depicted in Example 10-5.

Example 10-5 Geodetic query

```
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 is obviously extremely selective considering the size of the circle. An index on the location column eliminates most of the rows from the table and returns much faster than without the index.

The Geodetic Datablade module is best used for global data sets and applications.

10.4.3 TimeSeries

Many 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 analyze the price variation of a stock over time to determine if it is a good investment. This can 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 the product 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 features, such as user-defined timeseries data types, calendars and calendar patterns, regular and irregular timeseries, and a set of functions to manipulate the data.

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 Web site:

http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0510durity2/index.html

10.4.4 TimeSeries Real Time Loader

The TimeSeries Real Time Loader is a toolkit that complements the TimeSeries Datablade module by adding the capability to load 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 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.

10.4.5 Web

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 Standard Generalized Markup Language (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. You can use any tags in the generation of a page. This means that the Web DataBlade module can 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 clients. 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.

10.5 Conclusion

The use of DataBlade modules can greatly enhanced the capabilities of IDS and better serve your business purposes. 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.

11

Development tools, interfaces, and SOA integration

In this chapter, we provide an overview of all of the currently supported IBM development tools and application programming interfaces (APIs). We also provide a selection of additional Open Source and third-party tools, which can be used in combination with IDS 11. Because of the importance of service-oriented architecture (SOA), we also include a brief introduction of how to integrate IDS 11 with the SOA environment.

11.1 IDS 11 software development overview

Software developers have different backgrounds and different requirements to develop database-oriented applications. They basically need choice 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 actively motivate software developers to utilize the advanced features of IDS 11, IBM offers a broad selection of IDS programmer APIs and development tools, which complement the IDS 11 database server offering and provide choice to the IDS developer community. In this chapter, we introduce the new IDS 11 APIs and tools and provide an example of each API or tool.

11.1.1 IBM supported APIs and tools for IDS 11

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, standalone APIs and tools that we discuss throughout this chapter.

Even though we introduce each development API and tool in a separate section, we note all the tools, which are part of the CSDK in their section headers for easy identification.

11.1.2 Embedded SQL for C (ESQL/C): CSDK

ESQL/C allows the easy integration of structured query language (SQL) statements with C programming language applications. The SQL statement handling is a combination of an ESQL/C language preprocessor, 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 you deal with many SQL-related activities in a C-based application, and you need 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 11.

ESQL/C source code files typically have the extension .ec. In the first processing step, the ESQL/C preprocessor converts all of the embedded ESQL/C statements into C function calls and generates a .c file, which eventually is compiled 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;

We recommend 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 11-1 is an ESQL/C example, which comes with the product.

Example 11-1 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;
}
```

11.1.3 The IBM Informix JDBC 3.0 Driver: CSDK

Java database connectivity (JDBC) is the Java specification of a standard 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 11.

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, we recommend a JRE/JDK 1.4.2. A sample Java method is shown in Example 11-2.

Example 11-2 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
```

In Example 11-3 on page 322 we show a simple SQL/J code fragment.

Example 11-3 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());
        }
```

11.1.4 IBM Informix .NET Provider: CSDK

.NET is an environment that allows you to build and run managed applications. A *managed application* is an application 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 provides 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 its own classes that inherit from a common interface, for example, 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 through either of these two methods:

- Using the Microsoft .NET Framework Data Provider for ODBC along with the IBM Informix ODBC Driver
- Using the Microsoft .NET Framework Data Provider for OLE DB along with the IBM Informix OLE DB Provider

The IBM Informix .NET Provider can be used by any application that can be executed by the Microsoft .NET Framework.

Here are examples of programming languages that create applications that meet this criteria:

- Visual BASIC .NET
- Visual C#® .NET
- Visual J#® .NET
- ASP.NET

Figure 11-1 depicts how the Informix .NET Provider fits into the overall .NET framework.

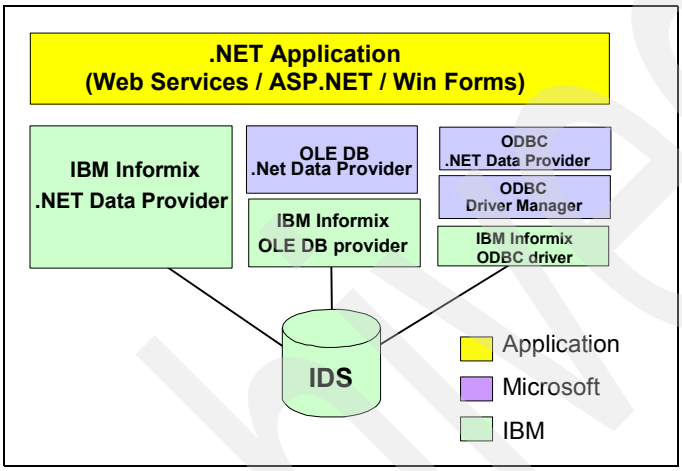


Figure 11-1 The Informix .NET Provider and the .NET framework

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 the IBM Informix Client SDK, Version 2.90, or later, installed.

Refer to Example 11-4 for a simple .NET code snippet, which accesses IDS 11 to select data from the `customer` table in the `stores_demo` database.

Example 11-4 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();
```

11.1.5 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 (PAMs) on UNIX and Linux plus LDAP authentication on Windows.

IBM Informix ODBC driver-based applications enable you to perform the following types of operations:

- Connect to and disconnect from data sources.
- Retrieve information about data sources.
- Retrieve information about the IBM Informix ODBC Driver.
- Set and retrieve IBM Informix ODBC Driver options.
- Prepare and send SQL statements.
- Retrieve SQL results and process them dynamically.
- Retrieve information about SQL results and process the information dynamically.

In Figure 11-2 on page 326, we depict a typical execution path of an Informix ODBC 3.0-based application.

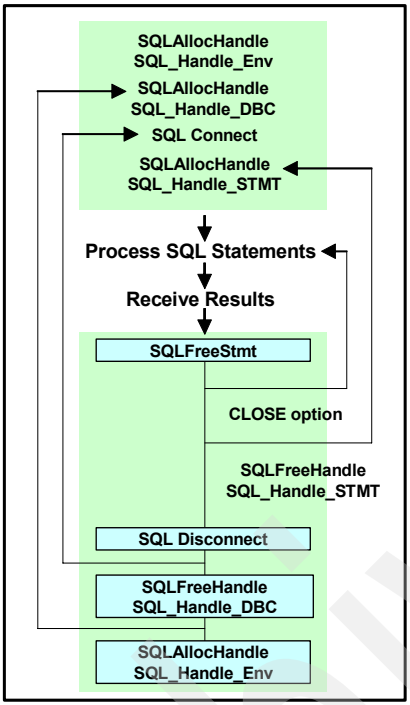


Figure 11-2 Typical execution path

Many third-party applications, such as spreadsheets, word processing, or analytical software, support at least ODBC connectivity to databases. Therefore, the Informix ODBC Driver might be the best option to connect this type of applications with IDS 11.

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, MULTISET, and SET)
  - DISTINCT
  - OPAQUE (fixed and unnamed)
  - Row (named and unnamed)
  - Smart large object (BLOB and CLOB)
  - Client functions to support certain extended data types
- Long identifiers
- Limited support of bookmarks

- Global Language Support (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

11.1.6 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 utilize 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 Microsoft .NET Provider for OLE DB. A code sample for the OLE DB Provider is shown in Example 11-5.

Example 11-5 An Informix OLE DB Provider code example

```
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 DataSouce 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);
}
```

11.1.7 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. A simple object interface is depicted in Example 11-6.

Example 11-6 A simple Informix Object Interface for a 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;
}
```

11.1.8 IBM Informix 4GL

For a long time, Informix Fourth Generation Language (4GL) has been a highly successful database (Informix) centric business application development language. Initially developed in the mid-1980s, it became extremely popular in the late 1980s until the mid 1990s.

The broad success of Informix 4GL among the Informix independent software vendor (ISV) community is heavily based on its 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 displaying input and output forms and being able to generate flexible reports.

At the time of the writing of this book, the classic 4GL is still being maintained (but no new major features are being added) on several current OS platforms and is currently at 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. The 4GL comes in two dialects:

- A precompiler version essentially generates ESQL/C code in a first phase, and in the second phase, it generates C code, which later compiles into OS dependent 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, you can also debug 4GL RDS-based applications with the optional Interactive Debugger (ID).

From a 4GL language perspective, both versions 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. A simple 4GL application is shown in Example 11-7.

Typical Informix 4GL applications consist of three file types:

- The .4gl files contain the actual Informix 4GL business logic and the driving code for the user interface components (for example, menus, windows, and screens).
- The .per files are the source code versions of 4GL forms and do not contain any procedural logic. They are a visual representation of how the forms must appear.
- You can use the message files to create customized error messages and application messages, for example, to accommodate different language settings for deployment.

Example 11-7 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
```

We have shown the associated .per 4GL form description file for Example 11-7 on page 331 in Example 11-8.

Example 11-8 The associated .per 4GL form description file for Example 11-7 on page 331

```
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 does not receive any further major enhancements, Informix 4GL developers might want to consider moving to the IBM Enterprise Generation Language, which incorporates many of the powerful features of the 4GL language.

11.1.9 IBM Enterprise Generation Language (EGL)

IBM 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, you 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 reuse, 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 causes the programmer to organize elements of a business application into highly structured, reusable, easily maintained, and superior performance program components.

IBM EGL is procedural, but it is also fourth generation. While IBM EGL supports all of the detailed programming capabilities you need 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 you can apply to various EGL components, and these properties greatly enhance or configure the capability of these objects. Last, the term *enterprise*, as in Enterprise Generation Language, connotes that EGL can satisfy programming requirements across the entire enterprise. For example with EGL, you 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, which results in 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 Create, Read, Update, and Delete (CRUD) database tasks
- Provide a test environment to efficiently test the application

EGL provides capabilities that make this task extremely easy for the business-oriented developer. See the simple EGL program in Figure 11-3.

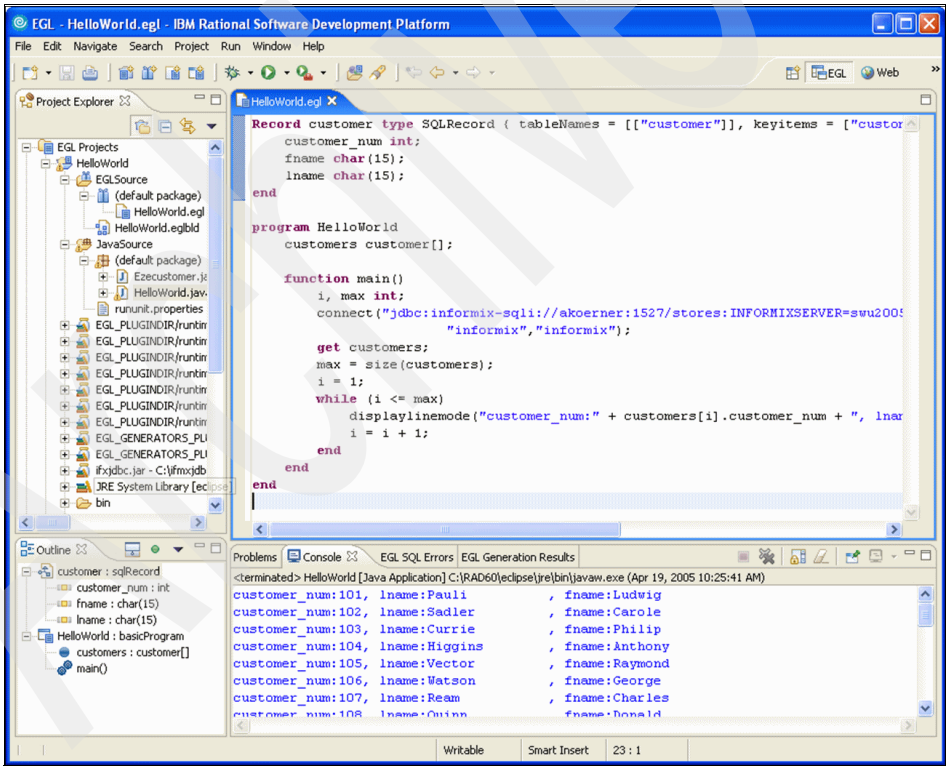


Figure 11-3 A simple EGL program accessing IDS 11

EGL benefits include:

- Connectivity: Wizards take the developer through a step-by-step process of defining connectivity.
- Database schema: If you use 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 automatically generates the typical CRUD functions that are the workhorse functions for database driven applications.
- 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.

11.1.10 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 you use static SQL statements, the program knows all the components at compile time.

ESQL/COBOL is currently only available on AIX, HP/UX, Linux, and Solaris. In Example 11-9, we depict an ESQL/COBOL snippet.

Example 11-9 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 'DEMO1 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.
```

11.2 Additional tools and APIs for IDS 11

In addition to the already broad support of IBM development tools and APIs for IDS 11, there are many additional options through either the Open Source community or third-party vendors. This section describes a selection of those offerings.

Certain material in this section is copyright (c) Jonathan Leffler 1998, 2006, and has been used with permission.

11.2.1 IDS 11 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 almost 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.

IBM Informix IDS supported drivers for PHP

There are currently three different options for integrating IDS 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 drawbacks.

For example, it always requests scrollable cursors, which can lead to slow query processing, and it might be warning-prone. In addition, there is no support for OUT/INOUT stored procedures in IDS 11.

- 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. It is developed and supported through the PHP community. You can download it from:

  http://us3.php.net/ifx

  The current version of the Informix Extensions has full featured support for IDS 7, but only partial support for IDS versions greater than 9.*x* (including IDS 11). Because it is based on ESQL/C (see also the 11.1.2, “Embedded SQL for C (ESQL/C): CSDK” on page 318), it provides high performance access to the underlying IDS database. Similar to the unified ODBC Driver, it also works with PHP 4 and PHP 5. Refer to Example 11-10 for an PHP and Informix PHP extensions code example.

Example 11-10 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 and 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 a PHP extension community library (PECL) 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 11, you can either use the PDO_ODBC or the new PDO_INFORMIX Driver.

  There are several advantages of using the new PDO_INFORMIX Driver, which you can download from:

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

  It is a native driver, so it provides high performance. And, it also has been heavily stress-tested, which makes it a great candidate for production environments. A simple PDO example is depicted in Example 11-11.

Example 11-11 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.

You can find ZendCore for IBM at:

http://www.zend.com/core/ibm

The user ID running the apache server must have a valid $INFORMIXDIR environment variable set to be able to access an IDS database. If the variable is not set properly, you might get an error about language translation.

Additional information

For more information about how to develop IDS-based PHP applications, refer to the IBM Redbooks publication *Developing PHP Applications for IBM Data Servers*, SG24-7218. Chapter 5 of that publication specifically discusses the combination of IDS and PHP.

11.2.2 PERL and DBD::Informix

Perl is an extremely popular scripting language, which was originally written by Larry Wall. Perl Version 1.0 was released in 1987. The current version of Perl is 5.8.8 and you can obtain 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, IDS (plus others). The current version of DBI as of writing this book is 1.52, and it requires Perl 5.6.1 or higher.

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 and you can download it from CPAN at:

http://www.cpan.org

You can obtain support for the current DBD::Informix Driver through the following channels:

- dbd.informix@gmail.com
- dbi-users@perl.org

To build the DBD::Informix Driver for Perl, you need the following components:

- IBM Informix ESQL/C 5.00 or higher (ClientSDK)
- An ANSI C compiler (code uses prototypes)
- A small test database with DBA privileges
- Perl Version 5.6.0 or higher (5.8.8 strongly recommended)
- DBI Version 1.38 or higher (1.50 or later strongly recommended)

We depict the use of the DBD::Informix Driver in a Perl application in Example 11-12.

Example 11-12 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});
$sth = $dbh->prepare(q%SELECT Fname, Lname, Phone
            FROM Customer WHERE Customer_num = ? %);
$sth->execute(106);
$ref = $sth->fetchall_arrayref();
for $row (@$ref) {
   print "Name: $$row[0] $$row[1], Phone: $$row[2]\n";
}
$dbh->disconnect;
```

11.2.3 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 book is 8.4.13. Tcl/Tk can be downloaded from:

http://www.tcl.tk/

Tcl/Tk is an interpreted environment. The Tcl interpreter can be extended by adding precompiled 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 11 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 11:

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:

- Prepares 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:

  For example:

```
sql run {delete from sometable where pkcol = ?} $pkval
```

To perform 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?
```

This command reopens the statement, with new parameters.

```
sql close $stmt
```

This indicates that you have no further use for the statement and it frees both the cursor and statement.

11.2.4 Python, InformixDB-2.2, and IDS 11

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 discover 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 provides extensive functionality and support for tasks, such 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 11.

The current Informix IDS implementation of the Python DB-API 2.0 specifications is called InformixDB-2.2, was developed by Carsten Haese, and was released in March 2006. You can obtain more detailed information and downloads related to the Informix Python module at:

http://informixdb.sourceforge.net/

For a simple Python code example, which connects to an IDS database server, refer to Example 11-13 on page 344.

In order to build the InformixDB-2.2 module, you need a current version of ESQL/C installed on your development machine. 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 access Python at:

http://www.python.org/

Example 11-13 A simple Python application to access IDS 11

```
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])
```

11.2.5 IDS 11 and the Hibernate Java framework

Hibernate is an extremely popular object-relational, Java-based persistence and query service. See also:

http://www.hibernate.org

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 utilize IDS 11 in combination with Hibernate, be sure to select the Informix dialect through a property setting in the hibernate.properties file. For IDS, the hibernate.dialect property must be set to: org.hibernate.dialect.InformixDialect.

You can execute Hibernate-based applications stand-alone or in an Java J2EE-based application server environment, such as IBM WebSphere Application Server.

Tip: During run time, a Hibernate-based application typically reuses 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 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 book, the current IBM Informix JDBC 3.0 Driver does not support that feature.

11.2.6 IDS 11 and WAS CE

WebSphere Application Server Community Edition (WAS CE) is a free-of-charge application server available from IBM. The current version is 1.1.0.1. It is possible to create J2EE applications that access IDS through WAS CE. The first thing to do is to make sure that the JDBC Driver is installed in WAS CE.

We suggest that you install the IDS JDBC Driver in WAS CE using these steps. These instructions assume that WAS CE is installed at $GERONIMO_HOME:

1. Create all the directories that might be missing in the following path:

   $GERONIMO_HOME/repository/com/Informix/ifxjdbc/11.10

2. Copy $INFORMIXDIR/jdbc/lib/ifxjdbc.jar to that directory.
3. Rename ifxjdbc.jar to ifxjdbc-11.10.jar.

From this point, when you create a data source, you see the IDS JDBC Driver available in the list of drivers.

11.2.7 IDS 11 support for Ruby and Rails (Ruby on Rails)

Ruby is an object-oriented programming language which has been inspired by Smalltalk, sharing features with Python, Lisp, Dylan, and CLU. Its is a reflective and single-pass interpreted language (scripting). The main implementation (interpreter et all) has been released under an MIT license.

The official Ruby Web site is:

http://www.ruby-lang.org/

Rails (or *Ruby on Rails*) is a full stack Web framework written in Ruby. It is basically Web development made easy through *Convention over configuration* and *Do not Repeat Yourself* principles. Rails development, deployment, and maintenance is made easy through patterns, structure, and built-in plumbing, such as MVC, ORM, Migrations, Testing, and AJAX.

You can obtain details about Rails and the associated downloads at:

http://www.rubyonrails.org/

At the time of this writing, the Ruby Driver is not available. However, the beta for this driver is planned to start in mid-June 2007. The Ruby support is provided through the IBM_DB Driver that applies to both IDS and DB2.

Although the IBM_DB Driver, which supports IDS, is not available at the time of this writing, here is a brief overview of how the IBM_DB syntax might look:

- Make a connection.

```
conn = IBM_DB::connect database, user, password
```

- Execute a Data Definition Language (DDL) statement.

```
drop = 'DROP TABLE test‘
result = IBM_DB::exec(conn, drop)
```

- Prepare and Execute.

```
insert = 'INSERT INTO test (col1, col2) VALUES (?, ?)'
stmt = IBM_DB::prepare conn, insert
IBM_DB::bind_param stmt, 1, 'name', IBM_DB::SQL_PARAM_INPUT
IBM_DB::bind_param stmt, 2, 'picture', IBM_DB::SQL_PARAM_INPUT
result = IBM_DB::execute stmt
```

- Prepare, Execute, and Fetch.

```
stmt = IBM_DB::prepare conn, "SELECT name FROM animals“
IBM_DB::execute stmt
while (data = IBM_DB::fetch_both stmt)
puts data[0]
end
```

11.2.8 IDS 11 software development direction

Much is happening with IDS application development, but unfortunately, it is not available at the time of this writing. However, it is still important to mention.

IBM is developing common drivers for IDS and DB2, which will include:

- IBM Data Server Client
- IBM Data Server Runtime Client
- IBM Data Server Driver for ODBC, CLI, and .NET
- IBM Data Server Driver for JDBC and SQLJ
- IBM Data Server Driver for Ruby, PHP, Perl, Python

In addition to the runtime offering, new common tools will be available which will include:

- IBM Data Server Administration Console
- IBM Data Server Developer Workbench

These are interesting offerings planned for late in 2007 that continue the evolving story of application development and IDS.

11.3 IDS 11 and SOA integration

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

As already discussed and introduced in Chapter 10, “Functional extensions to IDS” on page 301, IDS 11 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, easily integrate itself into any simple or complex SOA environment.

There are several ways to integrate IDS 11 into a SOA framework. You need to differentiate between service providing and service consumption in addition to foundation technologies, such as XML support and reliable messaging integration. Due to the IDS 11 leading extensible architecture and features that are unique to Version 11, you can easily achieve an SOA integration.

11.3.1 SOA foundation technologies in IDS 11

Service-oriented architectures rely on an XML-based messaging exchange. Although not necessarily required to be provided by the database server for SOA integration, having XML generating functions built into the server can dramatically help with the integration development tasks.

IDS 11 provides a sophisticated set of XML-related functions to create and transform XML-formatted documents.

Quite often SOA frameworks need more reliable network infrastructures in addition to the established Internet protocols, such as http or https in combination with SOAP. One example for a reliable messaging infrastructure is the WebSphere MQ messaging layer. IDS 11 supports the integration into an WebSphere MQ setup through the bundled WebSphere MQ DataBlade. For more information, refer to 10.2.4, “The MQ DataBlade” on page 306.

11.3.2 Service providing with IDS 11

In most SOA scenarios, the integration work is done on the application level and not so much on the database server level. Sometimes, it might be required to provide SOA compliant access on an IDS database object level.

There are several options for IDS 11 developers to provide Web services. Most of them utilize one of the many application development options that are available for IDS:

- Java-based Web services (through the IDS JDBC Driver)
- .NET 2.0-based Web services (through the new IDS .NET 2.0 Provider)
- IBM Enterprise Generation Language (EGL)-based Web services

- PHP, Ruby, Perl, and C/C++ Web services

In addition to the typical, application development language-based Web services listed, IDS 11 is also supported by the IBM Web Services Object Runtime Framework (WORF), which allows rapid Web service development against IDS 11 based on SQL statements, such as SELECT and INSERT, and stored procedure calls.

And finally, with the introduction of the new Web Feature Service (WFS) for geospatial data, IDS 11 is now capable of providing an Open Geospatial Consortium (OGC) compliant Web service (just add an HTTP server to IDS) to integrate easily with geospatial applications that are WFS compliant.

11.3.3 Service consumption with IDS 11

In addition to providing Web services, it can be interesting for an application developer to integrate existing Web services. Those Web services can be either special business-to-business scenarios or public accessible services, such as currency conversion, stock ticker information, news, weather forecasts, search engines, and many more. You can have dynamic access to an official currency conversion service on a database level if the application needs to deal with this information. Or what if an application wants to relate actual business data stored in an IDS database against news from news agencies?

The advantages of having Web services accessible from SQL include easy access through the SQL language and standardized APIs (for example, ODBC and JDBC), moving the Web service results closer to the data processing in the database server, which can speed up applications, and providing Web service access to the non-Java or non-C++ developers.

Due to the extensible architecture in IDS 11, such as the DataBlade API (for C/C++-based extensions) or the J/Foundation features (for the Java-based extension), it is extremely easy to develop Web service consumer routines that can be run within the Informix Dynamic Server context.

Example 11-14 C source code of the UDR from Figure 11-4 on page 351

```
#include <ctype.h>
#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include <ifxgls.h>
#include <mi.h>

#include "CurrencyExchange.h"
#include <soapH.h>
```

```
#include "CurrencyExchangeBinding.nsmap"

UDREXPORT mi_double_precision *CurrencyExchange
(
mi_lvarchar *country1,
mi_lvarchar *country2,
MI_FPARAM *Gen_fparam
)
{
   mi_double_precision *Gen_RetVal;
   MI_CONNECTION *Gen_Con;
   struct soap *soap;
   float result = (float)0.0;

   Gen_Con = NULL;

   soap = (struct soap*)mi_alloc(sizeof(struct soap));
   if (soap == 0)
   {
      DBDK_TRACE_ERROR("CurrencyExchange", ERRORMESG2, 10);
   }

   soap_init(soap);

   Gen_RetVal =
      (mi_double_precision *)mi_alloc( sizeof( mi_double_precision ) );
   if( Gen_RetVal == 0)
   {
      DBDK_TRACE_ERROR( "CurrencyExchange", ERRORMESG2, 10 );
   }

   if (soap_call_ns1__getRate(soap, NULL, NULL,
      mi_lvarchar_to_string(country1),
      mi_lvarchar_to_string(country2),
      &result) == SOAP_OK)
      *Gen_RetVal = result;
   else
      mi_db_error_raise(Gen_Con, MI_EXCEPTION, "SOAP Fault");

   return Gen_RetVal;
}
```

In Figure 11-4, a simple C User-Defined Routine (UDR) is accessing a public currency exchange Web service to retrieve up-to-date currency exchange rates (in our example, from USD to EUR) and calculate the euro value (PRICE_EURO column) dynamically.

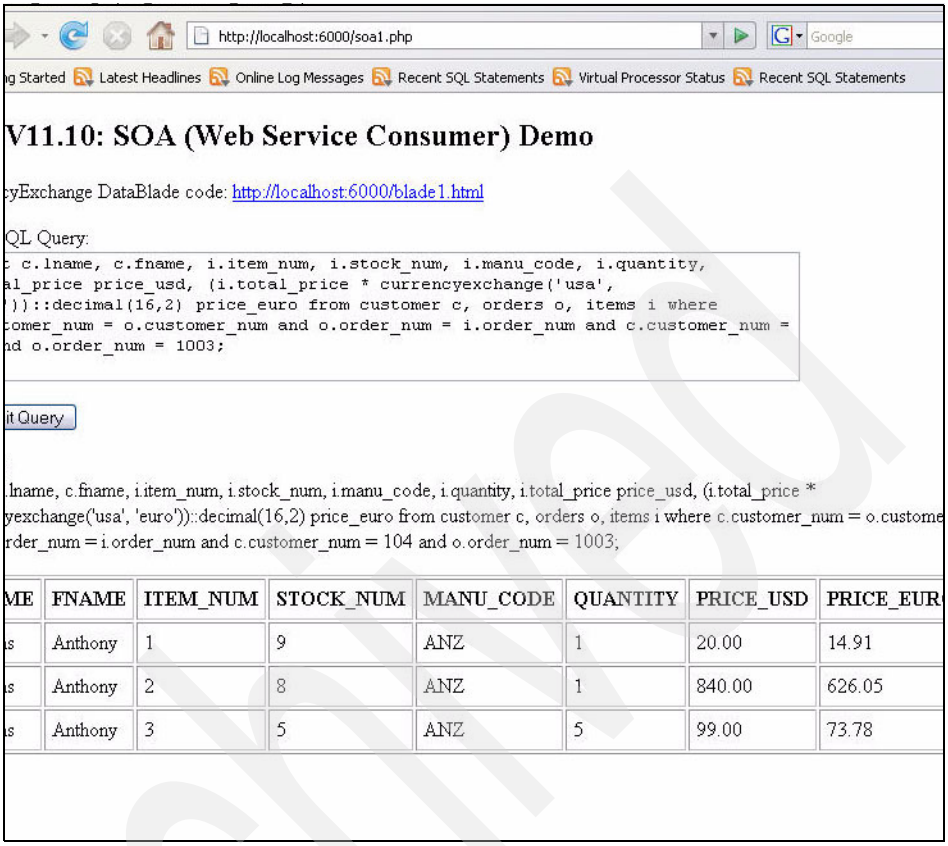


Figure 11-4 Use a C language-based UDR for Web service consumption

Further reading

For more details about how to integrate Informix Dynamic Server with an SOA environment, refer to the IBM Redbooks publication *Informix Dynamic Server V10 . . . Extended Functionality for Modern Business*, SG24-72999.

Chapter 11 of that publication introduces SOA concepts and guides you step-by-step through different options, which are available to IDS developers, to provide and consume Web services.

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.

RSAM. Relational Sequential Access Method, is the disk access method and storage manager for the Informix DBMS.

ROLAP. Relational OLAP. Multidimensional analysis using a multidimensional view of relational data. A relational database is used as the underlying data structure.

Roll-up. Iterative analysis, exploring facts at a higher level of summarization.

Server. A computer program that provides services to other computer programs (and their users) in the same or other computers. However, the computer that a server program runs in is also frequently referred to as a server.

Shared nothing. A data management architecture where nothing is shared between processes. Each process has its own processor, memory, and disk space.

Static SQL. SQL that has been compiled prior to execution. Typically provides best performance.

Subject area. A logical grouping of data by categories, such as customers or items.

Synchronous messaging. A method of communication between programs in which a program places a message on a message queue and then waits for a reply before resuming its own processing.

Task. The basic unit of programming that an operating system controls. Also see Multitasking.

Thread. The placeholder information associated with a single use of a program that can handle multiple concurrent users. 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
ADK	Archive Development Kit
API	application programming interface
AQR	automatic query rewrite
AR	access register
ARM	automatic restart manager
ART	access register translation
ASCII	American Standard Code for Information Interchange
AST	application summary table
BLOB	binary large object
BW	Business Information Warehouse (SAP)
CCMS	Computing Center Management System
CDR	Continuous Data Replication
CFG	Configuration
CLI	call-level interface
CLOB	character large object
CLP	command line processor
CLR	Continuous Log Recovery
CORBA	Common Object Request Broker Architecture
CPU	central processing unit
CS	Cursor Stability
DAS	DB2 Administration Server
DB	database
DB2 II	DB2 Information Integrator
DB2 UDB	DB2 Universal Database™
DBA	database administrator
DBM	database manager
DBMS	database management system
DCE	distributed computing environment
DCM	Dynamic Coserver Management
DCOM	Distributed Component Object Model
DDL	data definition language
DES	Data Encryption Standard
DIMID	Dimension Identifier
DLL	dynamic link library
DML	data manipulation language
DMS	database-managed space
DPF	data partitioning facility
DRDA	Distributed Relational Database Architecture
DSA	Dynamic Scalable Architecture
DSN	data source name
DSS	decision support system
EAI	Enterprise Application Integration
EBCDIC	Extended Binary Coded Decimal Interchange Code
EDA	enterprise data architecture
EDU	engine dispatchable unit
EGL	Enterprise Generation Language
EGM	Enterprise Gateway Manager
EJB™	Enterprise Java Beans
ER	Enterprise Replication
ERP	Enterprise Resource Planning
ESE	Enterprise Server Edition

ETL	Extract, Transform, and Load
FP	fix pack
FTP	File Transfer Protocol
Gb	gigabits
GB	gigabytes
GUI	graphical user interface
HADR	High Availability Disaster Recovery
HDR	High Availability Data Replication
HPL	High Performance Loader
I/O	input/output
IBM	International Business Machines Corporation
ID	identifier
IDE	Integrated Development Environment
IDS	Informix Dynamic Server
II	Information Integrator
IMS™	Information Management System
ISA	Informix Server Administrator
ISAM	Indexed Sequential Access Method
ISM	Informix Storage Manager
ISV	independent software vendor
IT	information technology
ITR	internal throughput rate
ITSO	International Technical Support Organization
IX	index
J2EE	Java 2 Platform Enterprise Edition
JAR	Java Archive
JDBC	Java Database Connectivity
JDK	Java Development Kit
JE	Java Edition
JMS	Java Message Service
JRE	Java Runtime Environment
JVM	Java Virtual Machine
KB	kilobyte (1024 bytes)
LBAC	Label-Based Access Control
LDAP	Lightweight Directory Access Protocol
LPAR	logical partition
LRU	Least Recently Used
LUN	Logical unit number
LV	logical volume
Mb	megabits
MB	megabytes
MDC	multidimensional clustering
MPP	massively parallel processing
MQI	message queuing interface
MQT	materialized query table
MRM	message repository manager
MTK	DB2 Migration Toolkit for Informix
NPI	non-partitioning index
ODBC	Open Database Connectivity
ODS	operational data store
OLAP	online analytical processing
OLE	object linking and embedding
OLTP	online transaction processing
ORDBMS	Object Relational Database Management System
OS	operating system
O/S	operating system
PAM	Pluggable Authentication Module
PDS	partitioned data set
PHP	Hypertext preprocessor. A general purpose scripting language.
PIB	parallel index build
PSA	persistent staging area

RBA	relative byte address
RBAC	Role-Based Access Control
RBW	red brick warehouse
RDBMS	Relational Database Management System
RID	record identifier
RR	repeatable read
RS	read stability
RSAM	Relational Sequential Access Method
RSS	Remote Standalone Secondary
RTO	Recovery Time Objective
SA	systems administrator
SCB	session control block
SDK	Software Developers Kit
SDS	Shared Disk Secondary
SID	surrogate identifier
SMIT	Systems Management Interface Tool
SMP	symmetric multiprocessing
SMS	System-Managed Space
SSJE	Server Studio Java Edition
SOA	service-oriented architecture
SOAP	Simple Object Access Protocol
SPL	Stored Procedure Language
SQL	structured query
TCB	thread control block
TMU	table management utility
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
WFS	Web Feature Service
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

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this IBM Redbooks publication.

IBM Redbooks publications

For information about ordering these publications, see “How to get IBM Redbooks publications” on page 364. Note that some of the documents referenced here may be available in softcopy only.

- *Informix Dynamic Server V10 . . . Extended Functionality for Modern Business*, SG24-7299
- *Developing PHP Applications for IBM Data Servers*, SG24-7218
- *Informix Dynamic Server V10: Superior Data Replication for Availability and Distribution*, SG24-7319

Other publications

These publications are also relevant as further information sources:

- IBM white paper, *Label-Based Access Control*, by Dave Desautels and Lynette Adayilamuriyil
- *Guide to SQL: Syntax*, G251-228
- *IBM Informix Administration Reference Guide*, G251-2268-01
- *Informix Migration Guide,* G251-2293
- *DataBlade Modules Installation and Registration Guide*, G251-2276-01
- *Built-In DataBlade Modules User’s Guide*, G251-2770
- Jacques Roy, William W. White, Jean T. Anderson, Paul G. Brown, *Open-Source Components for Informix Dynamic Server 9.x*, Prentice Hall (Informix Press), 2002, ISBN 0-13-042827-2
- “Introduction to the TimeSeries DataBlade” located at the following Web site:

  `http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0510durity2/index.html`

- *Informix JDBC Driver - Programmer's Guide*, Part No.000-5354

How to get IBM Redbooks publications

You can search for, view, or download IBM Redbooks publications, IBM Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy IBM Redbooks publications, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Symbols

$nopage>PDQ *see* Parallel Data Query 26

A

access methods 15, 17, 20, 23, 286, 348
access plans 8, 33
ACID 267
ActiveX 327
Administration xvi, 29, 33, 66, 74–75, 85, 90, 94, 201, 220, 224, 347, 363
administration xi, 2, 15, 24–25, 28, 33, 39, 69, 137, 144, 167, 195–196, 199, 206, 208, 239–240
ADO 323, 327
AIX 28, 97, 127, 257, 306, 310, 336
aliases 92, 158
architecture xi, 1–3, 10, 18, 27, 39, 91, 307, 323, 347, 349
array xii, 2, 24, 30, 137–140
array security component 137
asynchronous 8–9, 28, 82, 97, 104, 235, 307
asynchronous I/O 8–9
Authentication 123–124

B

Backup xiii, 29, 44, 87, 89, 103, 168–172, 174, 176
backup 23–26, 32, 39–40, 42, 82, 85–87, 98, 101, 103, 106, 109–110, 120, 165, 167–174, 176–179, 191, 199, 208
backup and recovery 120, 171
Backup and Restore 103
backup and restore 87, 103, 168–169, 171–173, 178
backup database 109
backup image 174
backup server 110
Backup Services 171
Backups 26, 168
backups 3, 26, 32, 39–40, 44, 62, 65, 168–170, 172–173, 176, 178–179, 201, 248
Binary data types 250
Binary Large OBject 246
binary large object 149
Blade Server 116, 118–119
blade server 115–118
Bladelets 23
BladeManager 45, 250
BLOB 149, 152, 209, 246, 250, 277, 305, 326
BLOB *See* binary large object
BLOBspace 206–207
BLOBspaces 67
Block 77–78, 227
Boolean 246, 249, 277
B-TREE 250, 287
B-tree 20, 100, 188, 212, 251
BTS index 249
buffer pool 7, 67, 78, 99, 208, 234
buffer pools 89–90
BUFFERS 99
Built-in functions 21
Built-in security label 145
built-in SQL functions 145
business intelligence xii
Business-to-Business 349

C

C xiv, 17, 83, 130, 149, 151, 212, 253, 306, 318
CA xiii, xv, 322
see Continuous Availability
Cache 8, 84, 230, 232–233
cache 8, 74, 82, 84–85, 97, 188, 212–213, 270, 323, 345
cataloging 22
CDT *See* Collection Derived Tables
challenge-response authentication 131
Character Large OBject 246
checkpoint duration 31, 81
Checkpoints 31, 79
checkpoints 29, 31, 33, 75, 78–80, 90, 100, 201
chunk 10, 103, 172, 188, 206–209
CLI 91, 347
CLOB 209, 246, 255, 277, 305, 308, 326
CLOB *See* character large objects
CLR 29, 31, 106, 109, 121
CLR *See* Continuous Log Restore
Cobol 336

Collection 16, 287, 326
collection derived tables 275
collection type 16
Column protection 144
commit 254, 256, 269, 307
Committed Read 96, 267, 269
Concurrency 267, 270
concurrency 82, 267, 270
configuration 1, 31, 39–40, 64–67, 73–75, 81–82, 86–87, 89–90, 94–96, 98–100, 105, 107, 110, 115–117, 124, 130, 148, 158, 165, 171–173, 182, 185, 188, 191, 194, 199, 208, 210, 213, 217–219, 221, 224–225, 233, 267, 270, 277, 290, 346
configuration file 40, 64–65, 87, 99, 192, 222
configuration parameters 73–74, 89, 189
conflict resolution 28
Consistency 267
consolidation 28
Continuous Availability 28
continuous availability 107
Continuous Log Restore 29, 106, 109–110, 120
cooked files 97
CopperEye Indexing DataBlade 20
cost-based optimizer 8, 14
CPU VPs 82–83, 270
Crash recovery 76
create a database 18
CREATE DATABASE statement 181
CREATE VIEW 156
Create, Read, Update, Delete (CRUD) 335–336
Cursor Stability 267

D

Data Definition Language 15
data distribution 286
Data Encryption 149
data integrity 15, 40
data movement 41
data replication 101, 103
Data Type Casting 18
data types xi, 1, 3, 9, 15–20, 67, 250–251, 277, 305, 310, 313–314, 326, 329, 348
data warehousing xii
database instance 73, 195, 208
database objects 31, 137, 165, 271, 329
database operations 3, 6–7, 69
Database Server xii
database server 38–41, 66, 76, 89, 96, 99, 109–110, 128, 132, 144, 148, 158, 168, 181, 184, 188, 195, 199, 210–211, 217–221, 230, 271, 276, 318, 321, 329, 344, 348–349
  data processing 349
Databases 323, 335
DataBlade xiii, 20–22, 243, 246, 250, 253, 301–316, 348–349, 363
  DataBlade API 23
DataBlade Manager 303
DataBlade modules 20–22, 301, 303–304, 316
DataBlades 20, 22–23, 301, 303–304, 310, 312
DB2 Universal Database 92
DBD 341
DBI 341
DBMS xiv, 15, 37, 41, 44–45, 61, 64–66, 97, 124, 126–128, 133, 157, 164–165, 320, 344
DBPATH 205
DBSERVERALIAS 65, 124, 130–131, 158
DBSERVERALIASES 64
DBSERVERNAME 64–65, 92, 124, 130–131, 158
dbspace 51, 64–65, 87, 94, 97–98, 169, 177, 207–210, 212
DDL 15, 90, 92, 187, 256, 346
deadlocks 34, 267
Decryption 149, 151
decryption 32, 149, 151–152, 155, 158, 163
Default roles 133
degree of parallelism 11
DELETE 34, 140, 246, 256, 280–281
Deprecated parameters 99
development APIs 318
development tool 336
DIRTY READ 34, 246
disaster 30–31, 110
disk 67
disk failure 76
Disk mirroring 103
disk mirroring 101, 103, 109, 114–115, 117, 120
distinct data type 17
distinct type 278–279
distributed queries 125–126, 231, 253, 276, 278, 280
distributed query 231, 276
Distributed Relational Database Architecture 91
  DRDA 91
DML 31, 256, 281
DRAUTO 29, 77, 105
DRDA 91
DRDA *See* Distributed Relational Database Archi-

tecture
DROP 90, 181, 209, 328, 346
DSA *See* Dynamic Scalable Architecture
dynamic logging 86
Dynamic Scalable Architecture xi, 1, 3
Dynamic Shared Memory 7
dynamic SQL 336

E

Eclipse 301
EGL 334, 336, 348
EGL code 336
EGL Program 335
Embedded SQL 321, 336
ENCRYPTION 148–152, 155–157, 159–161, 163–164
Encryption 123–124, 149, 151–152, 155–156, 158, 160, 163–164
encryption 77, 124, 130, 148–150, 152–153, 155, 157–158, 161, 163, 344
encryption with views 155
Enterprise Edition 24–25, 27
Enterprise Generation Language 333, 348
Enterprise Generation Language (EGL) 335
Enterprise Replication 24, 27, 38, 44, 101, 103, 105, 125, 148
    See ER
enterprise replication 86, 253
ENVIRONMENT 335–336
environment variables 64, 93, 98–99
ER 24, 27–28, 31, 70–71, 86, 101–103, 105, 120, 124
    See Enterprise Replication
ESQL/C 306, 318–319, 331, 339, 342, 344
Excalibur 21–22, 305, 312–313
Explain File 289
explain file 187, 286–287, 289
Expression-based 9
expression-based 10
Extended data types 17
extended data types 18, 326
extensibility 15, 28, 301, 321, 342
extent 32, 218
external backup 103, 173–174, 177–178
external backup and restore 173

F

Failover 110
failover 30, 105, 109–110, 120
fan-in parallelism 5–6
fan-out parallelism 5, 11
fast recovery 79–81, 90
Federal Information Processing Standards 148
FER 281
FIPS 148
FIPS *see* Federal Information Processing Standards
FIRST 65
forest 28
Fragmentation 8
fragmentation 10, 40, 248
Function keyword 276
fuzzy checkpoints 80, 100

G

Geodetic 22, 312–313
Geodetic DataBlade 22, 311, 313
Graphical Toolkit 342
GROUP BY 12, 95, 276
GUI 239, 342

H

HA *See* High Availability
HDR 24, 27–30, 67, 77, 86, 101, 103–107, 110–113, 117–118, 120, 124, 148, 212
    See High Availability Data Replication
heterogeneous data 257
Hibernate 345
hierarchical 21, 28, 138, 243–244
High Availability xiii, 94, 104
high availability xv, 30, 86, 101, 106, 111, 119–120
    mirroring 101
High Availability Data Replication 101, 103
high availability data replication 101
High Performance Loader 10
host variables 220, 228
hot backup 110, 168, 173
HP-UX 176, 306, 310
HTML 258, 315, 338–339
HTTP 338, 349
HTTP server 338
Hypertext Preprocessor 68, 338
    also see PHP 68

I

IBM EGL 334

compiler 334
IBM Informix 323
database 323
ODBC Driver 323
OLE DB Provider 323
IBM Tivoli 27
IBM WebSphere Application Server 321
IDS xi, xiii–xv, 1–4, 6–16, 18–21, 23, 25–28, 31–34, 37–41, 43–44, 47, 50–51, 61, 64–66, 69, 73–76, 79–83, 86–89, 91–92, 94, 97–106, 110–111, 117–119, 121, 124–125, 127, 130–132, 137, 139–140, 142–143, 145, 148, 158–159, 165, 168, 173–174, 177–178, 183–184, 193–195, 201, 206–207, 217–218, 221, 227, 231, 233–234, 239–241, 243, 246, 248, 250–253, 255–258, 266, 268–271, 273, 275–276, 278, 280–281, 283–284, 286–287, 301, 303–304, 306–308, 310, 312–313, 315–319, 321, 324, 326, 335, 338–341, 343–349, 351, 376
IDS Express 24
IDS instance 64, 66, 87, 276
IDS instances 195, 256, 276
IDS Workgroup Edition 24
IDSAdmin 239
IDS-Express 25
Index Distribution 184
index scans 9, 32, 283
Index Self-Join 283
Indexes 10, 154
indexes 8, 10, 13–14, 20, 23, 26, 31, 33, 67, 100, 137, 154, 165, 185–187, 287, 305
Indexing encrypted data 153
information integration xv
Informix xi–xv, 1, 3, 8, 19–23, 25–28, 38–41, 44, 52–53, 65–66, 69, 74, 76–79, 83–85, 87–91, 94, 104, 130, 171, 173, 176, 182, 188, 201, 226, 240, 248, 253, 301, 305–306, 309, 312, 318–320, 322–323, 325–327, 329–331, 333, 336, 338–341, 344–345, 349, 351, 363–364, 376
4GL xii, 330
Dynamic Server xii–xiii, 1, 53
OnLine xii
Informix .NET Provider 323
Informix 4GL 330, 333
Informix C-ISAM 23
Informix Client Software Development Kit 318
Informix Connect 41, 318
Informix Dynamic Server xii–xiii, 8, 23, 25, 27, 52–53, 77, 83, 176, 312, 329
See IDS
Informix Dynamic Server Workgroup Edition 25
Informix High Availability Data Replication 27
Informix Image Foundation DataBlade 22
Informix JDBC 53–54, 320
Informix ODBC 323, 325–326
Informix OLE DB 323, 327
Informix Open Database Connectivity 325
Informix Spatial DataBlade 22
Informix TimeSeries Real-Time Loader 22
INFORMIXSERVER 64
INSERT 34, 141, 145, 200, 202, 228, 230, 244, 246, 254, 256, 270, 274, 280–281, 283, 307, 322, 328, 347, 349
instance xi, 2, 6–9, 25–26, 28–29, 31–33, 38–41, 46, 48, 51, 64, 66–71, 73–76, 78–80, 82–89, 93–95, 118, 158, 167–170, 172–173, 175, 177–178, 187, 189–191, 193, 195–196, 199, 202, 204–206, 208, 224, 276, 279, 304, 306, 345
configuration 40
operation 8, 168, 172
shared memory 7, 64, 67
ISOLATION 334
isolation levels 34, 96, 267

J

J2EE 334, 345
Benefits 334
Java 17, 19–20, 23, 43–44, 47, 69, 91, 248, 253, 277, 280, 314, 320, 323, 334, 345, 348–349
Java/J2EE 334
JAXP 321
JDBC 41, 53–54, 61, 92, 256, 271, 273, 320, 322, 345, 347–349, 364
joins 12, 94, 109, 275

K

KAIO *See* Kernal Asynchronous I/O
Kernal Asynchronous I/O 82

L

Label Based Access Control 123, 132, 136
Large Object Locator 306
latency 10, 30, 107, 111, 120
LBAC 32, 123, 132, 136–137, 140, 142–144, 148
LBAC *See* Label Based Access Control
LDAP 123–124, 131–132, 325

LDAP directory 124
LDAP support 131
LEFT OUTER JOIN 276
Linux xii, 2, 6, 10, 25, 41–43, 45, 50–51, 54, 60–61, 64–66, 88, 97, 124, 127, 253, 257, 303, 306, 310, 325, 336, 344
LIST 326
load 6, 10, 22, 26, 33, 39, 142, 272, 288, 314, 343
Locks 77, 96
Logical Log 78, 89, 229
logical log buffer 80
logical logs 39, 67, 75–76, 85–88, 103, 168–169, 173
logical restore 173

M

memory 3, 7–12, 23, 25–27, 31, 39, 64, 67–68, 74, 77, 79, 82, 84–85, 94, 182, 188, 191, 200–201, 208, 212–213, 217–218, 220–221, 225–227, 229, 233, 235, 237–238, 270, 288, 290, 315, 323
Memory Grant Manager 11
memory management 270
memory usage 218, 221
metadata 22, 27, 33, 64–65
Microsoft .NET 323
migration process 41
mirrored disk 104, 115
Mirroring 103
mirroring 67, 103, 109, 114–115, 117, 120, 173, 177, 207, 212
model 16, 22, 26, 243, 270, 311
MQ DataBlade 306–307
MQ Messaging 252
MTK 41
multiple instances 193, 195
MULTISET 275, 326
MVC 334

N

Named Parameters 273
near-line 29

O

object oriented programming languages 334
object types 344
object-oriented 344, 346
object-relational 15, 24, 345
ODBC 42, 271, 320, 323, 325–326, 338, 341, 347, 349
OLAP xi, 1, 11
OLE DB 323, 327
OLTP xi–xii, 1, 7–8, 11, 31, 79, 81, 170
ON-Bar 26, 29, 44, 62, 169, 171–173
  components 171
ON-bar 174
onbar 44, 172
oncheck 40, 67, 76, 190–191, 196–197, 202, 204, 209
ONCONFIG 40, 79, 92, 96, 98, 186, 189–191, 219, 222–224, 270
onconfig 82–84, 88, 93, 97, 99, 174, 189–191, 201, 213, 270, 290
ONCONFIG file 219
onmode 40, 67, 76, 82–84, 89–90, 93, 95, 97, 174, 177–178, 186, 188–189, 191–192, 197–198, 208–210, 212, 214, 270, 290
onmode utility 188
onspaces 67, 206, 208–212, 305
onstat 66, 68, 78, 82–84, 93, 98, 182, 192, 207, 226–230, 234, 236–238, 240, 270
Ontape 172
ontape 29, 44, 62, 167–170, 172–174, 176, 178, 191, 208
Open Database Connectivity *See* ODBC
OpenAdmin Tool 68
optimization 3, 15, 29, 34, 183, 251
optimizer 8, 10–11, 13–14, 31–33, 39, 235, 251, 271, 275, 286–287
  directives 14, 286
  query plans 32
  statistics 14, 33
Optimizer Directives 286
Oracle 253
ORDER BY 94, 244, 275, 278
outer join 286

P

packages 243
page 67, 74, 76, 80, 90, 94, 177, 190–191, 234, 315
Pages 77–78
PAM 123–124, 130–131, 325
  Also see Pluggable Authentication Module 124
Parallel backup and restore 24
parallel data query 11, 24

parallel insert 12
parallel scan 11–12
parallelism 5–6, 8, 11–13
Partial Trust 125
Partitioning 10
partitioning 9–12, 28
Password 124, 158–159, 327
password authentication 127, 131–132
password encryption 130, 157–158
passwords 127, 150, 155, 157–159, 161
PDQ 11, 26, 67, 181, 186, 188, 212
Also see Parallel Data Query
performance tuning 39
PERL 341
PHP 33, 68, 239, 338–340, 347, 349
Also see Hypertext Preprocessor 68
Physical Log 78
Pluggable Authentication Module 124
Primary 20, 94, 114, 116
primary 20, 27, 29, 37, 74, 81, 103–118, 120, 148, 173, 175, 272, 322, 335
primary key 175
privileges 43, 132–133, 143, 196, 231, 342
processes 3–4, 6, 41, 51, 233, 344, 347
Programming languages
IBM EGL 334
Java/J2EE 334
structured 334
programming languages 324, 334
Python 344, 346–347

Q

query drill-down 217–219, 239–240
query execution plan 286
Query Explain 287
query plan 186, 218, 228, 231, 238, 286, 290
queues 33, 81, 253, 257, 307

R

RAID 103, 114
Rational 336
raw devices 97
raw disk 10
READ COMMITTED 96
READ UNCOMMITTED 96
read-ahead 9, 13
real-time 12, 21, 27, 30, 40, 70, 314
recoverability 95
Recoverable Groups 110–111, 113, 117
recovery xi, 3, 23, 30–31, 33, 75–77, 79–81, 90, 95, 100, 110, 114, 120, 168–169, 171–172
Redbooks Web site 364
Contact us xvi
referential integrity 11
remote administration 196
Remote Standalone Secondary server 106
Remote Standalone Secondary servers 111
REPEATABLE READ 182–183
Repeatable Read 267
replicate 28, 105
replication 27–29, 31, 38, 40, 86–87, 101, 103, 105–106, 113, 121, 124–126, 188, 253
repreparation 270–271
Restore xiii, 29, 44, 87, 103, 106, 109–110, 120, 174, 176–177
restore 24, 26, 29, 32–33, 39–40, 42, 85–87, 103, 110, 167–169, 171–178
Role Based Access Control 123, 132
Roles 132
roles 32, 123, 128, 132–133
rollback 254, 256, 269
roll-forward 76, 85, 100, 104, 110, 168, 173
Round robin 9
Row protection 143
row type 17
RSS *See* Remote Standalone Secondary server
RTO 75, 81
RTO checkpoints 81
R-tree 20, 22, 304
Runtime Environment 334

S

Scalability 3, 171
scalability xi, 1, 3, 8, 23–24, 26, 108, 316, 321
scheduled maintenance 199
schema 90, 234, 236–238, 243, 271, 335–336
scrollable cursors 339
SDS 30, 94, 101, 106, 108–109, 113–120
see Shared Disk Secondary
see Shared Disk Secondary servers
Search 21, 246, 312, 349
Secondary 20, 29, 94, 101, 106–109, 111, 113
secondary 20, 29–30, 101–102, 104–105, 107–108, 110–113, 117–118, 120, 148
security 24–25, 28, 32, 109, 132, 137–145, 148, 153, 158, 161, 164–165

levels 142
permissions 165
security label 137, 139–145
security policy 137, 139, 142–145
SELECT 34, 129, 140, 145, 150–151, 153, 156, 160–161, 232–233, 236, 244–245, 248–249, 254, 256, 270, 272, 275–276, 278, 280, 282, 284, 306–307, 311, 313, 321, 332, 337, 342, 347, 349
SELECT triggers 280
Sensors 199–200
Server xi–xiv, 1, 3, 8, 23, 25, 27–28, 38, 41, 44, 52–53, 65–66, 76–78, 83–84, 89–91, 94, 116, 118–119, 176, 182, 200, 226, 248, 305, 309, 312, 315, 321, 326–327, 329, 338, 340, 345, 347, 349, 351, 363, 376
Service Oriented Architecture 253
service oriented architecture 317
Session properties 179
session properties 180–182
SET 32, 95–96, 132–133, 138, 141, 145, 149–152, 155–157, 159–160, 163–164, 182–183, 186, 188, 210–211, 218, 267, 289, 326
SET ISOLATION 97, 180
Shared Disk Secondary 30, 106
shared library 19, 343
Shared Memory 7, 77
sharing 346
size of encrypted data 162
SMP 11
SOA xiii, 253, 307, 317, 347–348, 351
Also see Service Oriented Architecture
SOAP 348, 350
SOA*see* Service Oriented Architecture
Spatial DataBlade 22, 310
SPL routines 19, 90, 280
SQL 5, 8–12, 14, 18–21, 23, 25, 28, 31–34, 39, 68, 91–92, 95–96, 132, 137, 139, 144–145, 149, 154–155, 157, 159, 163, 167, 179, 181, 183–184, 186–188, 194–197, 199, 204, 206, 208, 210–214, 217–221, 224–225, 227–240, 243, 246, 252, 254, 257, 270, 276, 280, 306–307, 311, 313–315, 318–319, 321–322, 324–325, 329–330, 335–337, 345, 349, 363
SQLHOSTS file 65, 92, 105, 157
sqlhosts file 65, 126, 130, 158
SQLJ 347
SQLSTATE 320, 337
SQLTRACE 217–219, 221–224, 227, 240
sqltrace buffer 218
Startup sensors 199, 202
Startup tasks 199, 202
statistical information 13–14, 33, 217
statistics 14, 29, 40, 68, 85, 167, 183–187, 218, 220, 227, 229–231, 234, 237, 271, 287, 289–290
STDIO 65, 174
Storage Manager 27, 44, 171
Stored Procedure 19, 199
stored procedures 19, 179, 194–195, 208, 215, 257, 271, 339
stores_demo database 175, 324
subqueries 275
symmetric multiprocessing *See* SMP
synchronous 29, 104, 246
Syntax 163, 213, 363
syntax 18–19, 34, 92, 149, 174, 176, 178, 182–183, 185, 187, 197, 202, 208, 219, 223, 267, 276, 282, 344, 346
sysadmin 194–199, 204–205, 213–214, 223–225
sysmaster 40, 68, 218, 225–226, 234, 236–237
syssqltrace 218, 233–234, 236–237
syssqltrace_info 237
syssqltrace_iter 238
system catalog 19, 184, 287

T

tables 7–8, 10, 14–17, 19–20, 22, 26, 29, 31, 33, 35, 40, 76, 90, 96, 100, 105, 137, 145, 159–160, 165, 168, 170, 172, 176, 185, 187, 194, 227–228, 232–233, 236, 238, 250, 257, 271, 275, 281, 303, 314, 327
TAPEDEV 65, 98, 174, 178, 190–191
target platform 334
Tcl/Tk 342
TCP/IP 91–92
template 54
temporary tables 14
test environment 89
Text Search 246
text search 21, 246, 305, 312
Text User Interface 334
thread 4, 7, 11, 82, 92–93, 172, 184
Timeseries 314
Timeseries Real Time Loader 314–315
Tivoli Storage Manager 27, 44
Tool Command Language 342
Toolkit 41, 342
transaction logging 96

tree security component 138
Trigger 78, 280
Triggers 280
triggers 29, 31, 34, 40, 236, 253, 280–281
TRUNCATE 31
tuning 3, 6, 39, 71, 74, 81–82, 86, 89, 95
two-phase commit 256, 307

U

UDA 20
    Also see user defined aggregate
UDF
    Also see User Defined Function
UDR 19, 32, 199, 349–351
    Also see User Defined Routine
    see User Defined Routine
UDT 184, 321
UNION ALL 94, 275
UNIX xii, 2, 6, 61, 171, 188, 234, 310
UPDATE 14, 34, 136, 140, 145, 151–152, 160, 183–185, 187, 246, 256, 270, 280–281, 287
UPDATE STATISTICS 184–185, 271, 287
UPDATE STATISTICS statement 185
user authentication 130, 165
user defined aggregate 19–20
user defined function 19, 315
user defined functions 18, 149, 303, 306
User Defined Routine 350
user defined routine 17, 19, 280
User Defined Routines 19, 47
user defined routines 280
user defined types 18, 20
user interface
    business application programs 334
user interface (UI) 334
userids 127–128

V

video xi, 2, 22, 257
Views 94
views 34, 95, 149, 155, 275, 281, 303
Virtual Index Interface 20, 184
Virtual Processor 84
virtual processor 4–6, 11, 201, 213, 305
Virtual Processors 82, 84, 270
virtual processors 4–6, 8–9, 11–12, 90, 188
Virtual Table Interface 20
VP 4–6, 9, 33, 67, 74, 82, 270
    Also see Virtual Processors

W

WAS 336
Web DataBlade 315
Web service 349–351
Web Services 334
Web services 348–349, 351
WebSphere 252–254, 257, 307, 312, 321, 345, 348
WebSphere MQ 253
Windows xii, 2, 6, 10, 25, 28, 41–43, 45–46, 54, 56, 61, 64–65, 88, 92, 124, 132, 171, 188, 257, 303–304, 306, 310, 324–325, 343–344

X

XML xi, xiii, 2, 29, 34, 257–258, 315, 321, 344, 348
XPath 257–258
XPS 38

Z

Zend Core for IBM 340

Informix Dynamic Server 11: Advanced Functionality for Modern Business

Informix Dynamic Server 11: Advanced Functionality for Modern Business

Agile: Adaptable, flexible, and with blazing fast speed

Invisible: Small, embeddable, and an administration free zone

Resilient: Reliable, highly available, and secure

In this IBM Redbooks publication, we provide an overview of Informix Dynamic Server (IDS) 11. IDS is designed to help businesses leverage their existing information assets as they move into an on demand business environment. Requirements here call for a flexible data server that can accommodate growth, in applications, data volume, and numbers of users. And it offers the capability to minimize downtime and to provide the high availability required today. A new suite of business availability functionality provides greater flexibility and performance, automated statistical and performance metric gathering, improvements in administration, and reductions in operating costs. The IDS technology enables efficient use of existing hardware and software, including single and multiprocessor architectures. And it helps you keep up with technological growth, including such things as the use of nontraditional data types. Built on the IBM Informix Dynamic Scalable Architecture™ (DSA), IDS provides a next-generation parallel data server architecture that delivers mainframe-caliber scalability; manageability and performance; minimal operating system overhead; and automatic workload distribution. IDS delivers a lower total cost of ownership (TCO) by leveraging its well-regarded general ease of use and systems administration. It enables customers to use information in new and more efficient ways to create business advantage.

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

SG24-7465-00 ISBN 0738488798