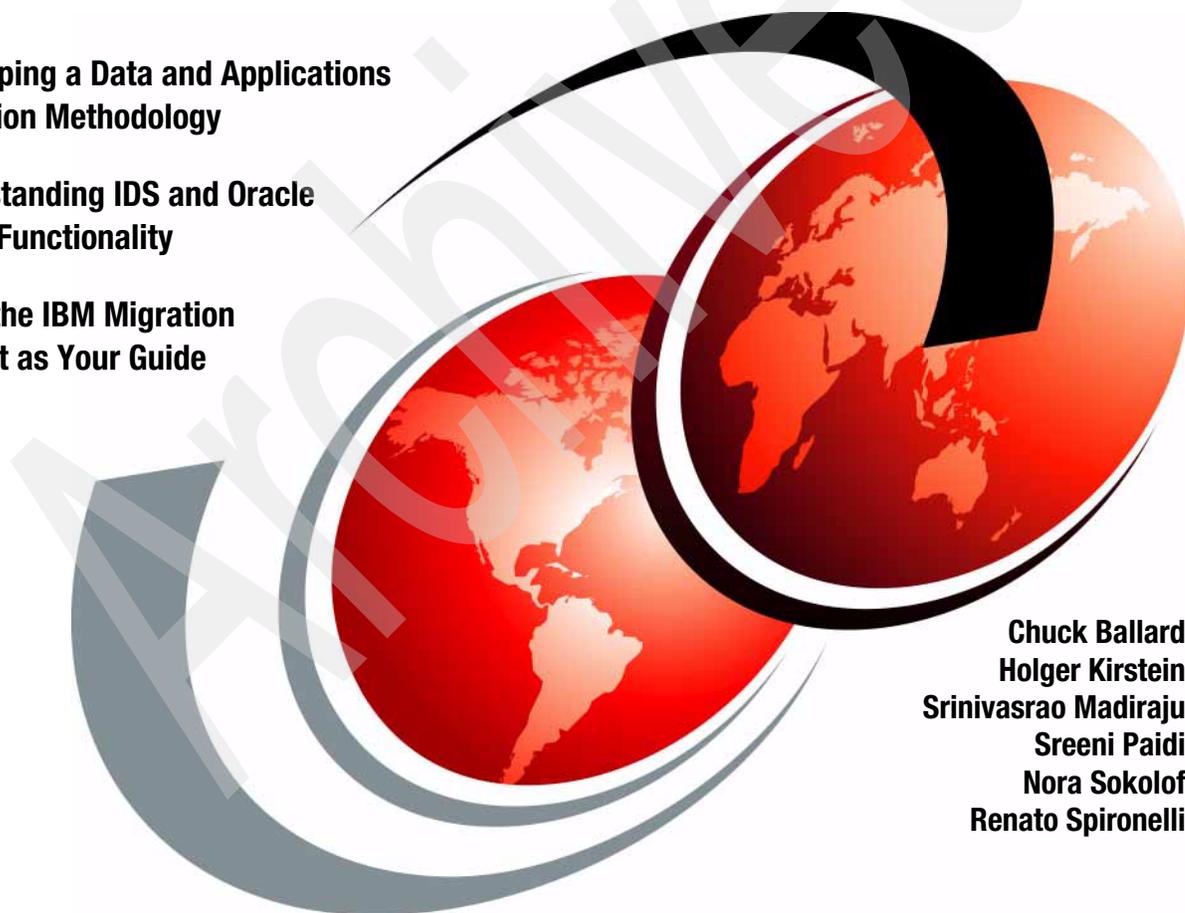


Migrating from Oracle . . . to IBM Informix Dynamic Server on Linux, UNIX, and Windows

Developing a Data and Applications
Migration Methodology

Understanding IDS and Oracle
DBMS Functionality

Using the IBM Migration
Tool Kit as Your Guide



Chuck Ballard
Holger Kirstein
Srinivasrao Madiraju
Sreeni Paidi
Nora Sokolof
Renato Spironelli



International Technical Support Organization

**Migrating from Oracle . . . to IBM Informix Dynamic
Server on Linux, UNIX, and Windows**

June 2009

Archived

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

First Edition (June 2009)

This edition applies to Oracle 10g and IBM Informix Dynamic Server 11.

© Copyright International Business Machines Corporation 2009. All rights reserved.

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

Contents

Notices	ix
Trademarks	x
Preface	xi
The team that wrote this book	xii
Become a published author	xiv
Comments welcome	xiv
Chapter 1. Introduction	1
1.1 Migrating	4
1.2 Positioning IDS	5
1.3 Informix Dynamic Server editions	6
1.4 IDS functionality	9
Chapter 2. Architectural overview	15
2.1 The basic architectures	16
2.1.1 Memory architectures	17
2.1.2 Process architectures	23
2.1.3 Physical database structures	28
2.1.4 Logical database structures	32
2.1.5 Data dictionary and system catalog	35
2.1.6 Database server communication	38
2.2 IDS licensing	40
2.3 Terminology	41
Chapter 3. Migration methodology	43
3.1 An IBM migration methodology	44
3.2 Migration preparation	45
3.2.1 Performing the migration assessment	45
3.2.2 Understanding and selecting migration tools	46
3.2.3 Estimating the effort required	47
3.2.4 Environment preparation	48
3.2.5 Getting educated on the Informix Dynamic Server	49
3.3 Migration	49
3.3.1 Database migration and design	49
3.3.2 Calibration	50
3.3.3 Application migration	51
3.4 The Test Phase	52
3.4.1 Migration refresh	52

3.4.2	Data migration	52
3.4.3	Testing	53
3.5	Implementation and cutover phase	56
3.6	Related information resources	57
Chapter 4. IBM Migration Tool Kit: An introduction		59
4.1	The MTK for Oracle migrations to IDS	60
4.2	Overview of features and functionality	61
4.2.1	The five step migration process	61
4.3	Inside the Oracle converter component	66
4.3.1	Translating tables, indexes, and views	68
4.3.2	Translating built-in functions	68
4.4	How to install, configure, and execute the MTK	69
4.4.1	System requirements	70
4.4.2	Installing MTK	71
4.4.3	Starting MTK	72
Chapter 5. An MTK tutorial		73
5.1	Part 1: Core database object migration	75
5.1.1	Create a project	75
5.1.2	Work with the project	77
5.1.3	Other useful features	91
5.1.4	Additional MTK features	99
5.1.5	Summary of best practices when using the MTK	100
5.2	Part II: Database application object migration	101
5.2.1	Migration of application objects: Lessons learned	109
Chapter 6. SQL considerations		111
6.1	DDL	112
6.1.1	Database creation	112
6.1.2	Tables	113
6.1.3	Views	129
6.1.4	Sequences	130
6.1.5	Synonyms	131
6.1.6	Triggers	133
6.1.7	DBLinks	133
6.2	DML	135
6.2.1	SQL	135
6.2.2	Selects	135
6.2.3	Pseudo-columns	138
6.2.4	Inserts	142
6.2.5	Outer joins	142
6.2.6	Sorts	145
6.2.7	Aliases	146

6.2.8 Truncate	146
6.2.9 Hierarchical queries	147
6.3 SPL and PL/SQL	147
6.4 Concurrency and transaction	154
6.4.1 Read concurrency	154
6.4.2 Update concurrency	158
6.5 Security	159
6.5.1 User authentication	159
6.5.2 Authorization	160
6.5.3 Column-level encryption	160
Chapter 7. Data conversion	165
7.1 Data conversion process	166
7.2 Time planning	167
7.3 Database schema conversion	167
7.3.1 Database schema extraction and conversion with the MTK	168
7.3.2 Database schema extraction with Oracle database interfaces	168
7.3.3 Move the database schema to the target IDS database server	171
7.4 Data movement	179
7.4.1 Unloading the data in Oracle	179
7.4.2 Load the data into the target IDS database server	197
7.4.3 Moving data using the Migration Tool Kit	209
7.5 Alternative ways for moving data	209
7.5.1 IBM InfoSphere Information Server	209
Chapter 8. Application conversion	211
8.1 Heterogeneous application environments	212
8.2 Client development APIs supported by IDS 11	212
8.2.1 Embedded ESQL/C	213
8.2.2 Embedded ESQL/Cobol	213
8.2.3 Informix JDBC 3.0 Driver	213
8.2.4 IBM Informix .NET Provider	214
8.2.5 IBM Informix ODBC 3.0 Driver	215
8.2.6 IBM Informix OLE DB Provider	216
8.2.7 IBM Informix Object Interface for C++	216
8.2.8 Additional APIs for accessing IDS 11	217
8.3 Migrating applications using unified interfaces	218
8.3.1 Package applications migration planning	218
8.3.2 Migrating applications based on ODBC	219
8.3.3 Migrating database applications based on JDBC	221
8.4 Conversion considerations for common client APIs	222
8.4.1 Application migration planning for source owned applications	223
8.5 Introduction to programming techniques	226

8.5.1 Embedded SQL	226
8.6 Migrate user-built applications	229
8.6.1 Converting Oracle Pro*C applications to Informix ESQL/C	229
8.6.2 Converting Oracle Java applications to IDS	240
8.6.3 Converting Oracle Call Interface (OCI) applications	248
8.6.4 Converting ODBC applications	254
8.6.5 Converting Perl applications	254
8.6.6 Converting PHP applications	259
8.6.7 Converting .NET applications	270
Chapter 9. Administration of Informix Dynamic Server	279
9.1 Administering the Informix database server	280
9.1.1 Configuring the database server	280
9.1.2 Set environment variables	280
9.1.3 Configure connectivity	282
9.1.4 Start and administer the database server	283
9.1.5 Preparing to connect to applications	285
9.1.6 Creating storage spaces and chunks	286
9.2 Data recovery and high availability	286
9.2.1 Backup and restore	287
9.2.2 Fast recovery	289
9.2.3 Mirroring	290
9.2.4 Data replication	290
9.3 Informix Dynamic Server admin utilities	296
9.3.1 Command line utilities	296
9.3.2 OpenAdmin tool for IDS	302
9.3.3 IBM Informix Server Administrator	304
9.4 Automatic monitoring and corrective actions	305
9.4.1 Administration API	305
9.4.2 The Scheduler	306
9.4.3 The sysadmin database	307
9.4.4 Query drill-down	307
9.5 IDS database server security	308
9.5.1 Server utility and directory security	308
9.5.2 Network data encryption	309
9.5.3 Connection security	310
9.5.4 Label-based access control (Enterprise Edition)	310
9.5.5 Auditing	311
Appendix A. Data types	313
A.1 Supported SQL data types in C/C++	314
A.2 Supported SQL data types in Java	316
A.3 Mapping Oracle data types to Informix data types	318

Appendix B. Terminology mapping	321
Appendix C. Function mapping	327
C.1 Numeric function mapping	328
C.2 Character function mapping	330
C.3 Date and time function mapping	332
C.4 Comparison and NULL-related function mapping	333
C.5 Encoding, decoding, encryption, and decryption function mapping	335
C.6 Implementation of new C-based functions in IDS	335
Appendix D. Database server monitoring	339
D.1 Memory monitoring	340
D.2 Process utilization and configuration	342
D.3 Disk space monitoring	344
D.4 Session monitoring	346
D.5 Cache monitoring	348
Appendix E. Database server utilities	351
Appendix F. Additional material	355
Locating the Web material	355
Using the Web material	356
System requirements for downloading the Web material	356
How to use the Web material	356
Glossary	357
Abbreviations and acronyms	361
Related publications	365
IBM Redbooks	365
Other publications	365
Online resources	365
Education support	366
How to get Redbooks	367
Help from IBM	367
Index	369

Archived

Notices

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

Trademarks

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

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

AIX 5L™	Distributed Relational Database	POWER®
AIX®	Architecture™	Rational®
C-ISAM®	DRDA®	Redbooks®
DataBlade®	IBM®	Redbooks (logo)  ®
DataStage®	Informix®	WebSphere®
DB2 Universal Database™	InfoSphere™	
DB2®	OS/390®	

The following terms are trademarks of other companies:

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

ACS, Interchange, Red Hat, and the Shadowman logo are trademarks or registered trademarks of Red Hat, Inc. in the U.S. and other countries.

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

EJB, Enterprise JavaBeans, J2EE, J2SE, Java, JavaBeans, JDBC, JDK, JRE, JVM, Solaris, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

ActiveX, Expression, Microsoft, SQL Server, Visual C#, Visual J#, Visual Studio, Windows Server, Windows Vista, 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 discuss considerations, and describe a methodology, for transitioning from Oracle® 10g to the Informix® Dynamic Server (IDS). We focus on the basic topic areas of data, applications, and administration, providing information about the differences in features and functionality in areas such as data types, data manipulation language (DML), data definition language (DDL), and Stored Procedures. Understanding the features and functionality of the two products will assist in developing a migration plan.

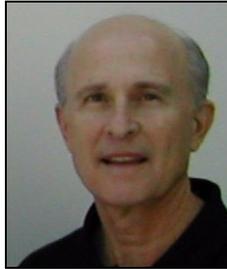
We provide a conversion methodology and discuss the processes for installing and using the IBM Migration Toolkit (MTK) to migrate the database objects and data from Oracle to IDS. We also illustrate, with examples, how to convert stored procedures, functions, and triggers. Application programming and conversion considerations are also discussed.

In addition, you will find script conversion samples for data loading, database administration, and reports. There is also information regarding procedures and tips for migration testing and database tuning. The laboratory examples are performed under Oracle 10g and IDS Version 11.5. However, the migration process and examples can also be applied to Oracle 7, 8, and 9i.

With this information, you can gather and document your conversion requirements, develop your required transition methodology, and plan and execute the conversion activities in an orderly and cost-effective manner.

The team that wrote this book

This book was produced by a team of specialists from around the world working with the International Technical Support Organization, in San Jose California. The team members are depicted below, 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.



Holger Kirstein is a resolution team engineer with the European Informix support team. He joined the Informix support team in 1996 and has over 15 years experience in application development and support for Informix database servers and Informix clients. He holds a Masters of Applied Computer Science from Technische Universität, Dresden.



Srinivasrao Madiraju is a senior software engineer working on the Informix Dynamic Server (IDS) integration team. He is responsible for quality integration of new features into IDS, particularly in the SQL area. He has recently been involved in Cloud Computing Technology with IDS. Srimi has also participated in projects focused on high availability, virtualization and cluster technologies, and working with a number of hardware vendors. Srimi holds a Masters degree in Computer Applications from Osmania University in India.



Sreeni Paidi has over 12 years of experience working on Informix, Oracle, and DB2® database servers as a Database Programmer, DBA, Database Integration Architect, and Partner Enablement Consultant. He worked with technology partners and customers around the world on various database related topics such as database porting, performance tuning, troubleshooting, benchmarking, training, and setting up servers for high availability. Sreeni joined IBM US as part of Trigo Technologies acquisition in 2004. He is currently working for the IBM Data Management Solutions Organization as a Lead Informix Consultant for India, Australia and New Zealand. Sreeni holds a Bachelors degree in Computer Science from the Osmania University, Hyderabad, India.



Nora Sokolof is a Consulting Technical Sales specialist with IBM Software North America. She is a member of the Data Servers and Application Development team and specializes in Information Management software products. She has held positions as a DB2, Informix, Oracle and PeopleSoft® development DBA. She is also the author of several white papers including *Transitioning from IBM Informix to DB2 - Database Comparisons and Migration Topics*, and has co-authored the following IBM Redbooks publications: *Planning for a Migration of PeopleSoft 7.5 from Oracle/UNIX to DB2 for OS/390*, SG24-5648, and *Oracle to DB2 Conversion Guide for Linux, UNIX, and Windows*, SG24-7048. Nora holds a Master of Science degree in Software Engineering from Pace University and has been an IBM software professional for more than 22 years.



Renato Spironelli is a Senior Oracle DBA and Oracle Certified Professional on Oracle 8i, 9i and 10g. He has more than 10 years of experience working in production environments, supporting clients in areas such as database porting, backup and recovery solutions, performance tuning, troubleshooting, and setting up server environments for high availability. Renato has been working for IBM since 2005 in the IBM Integrated Technology Delivery (ITD) organization in Sao Paulo, Brazil.

Other Contributors:

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

From the International Technical Support Organization, San Jose Center

Mary Comianos - Publications Management

Deanna Polm - Residency Administration

Emma Jacobs - Graphics

James Hoy - Editor

Become a published author

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

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability. Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

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

redbooks@us.ibm.com

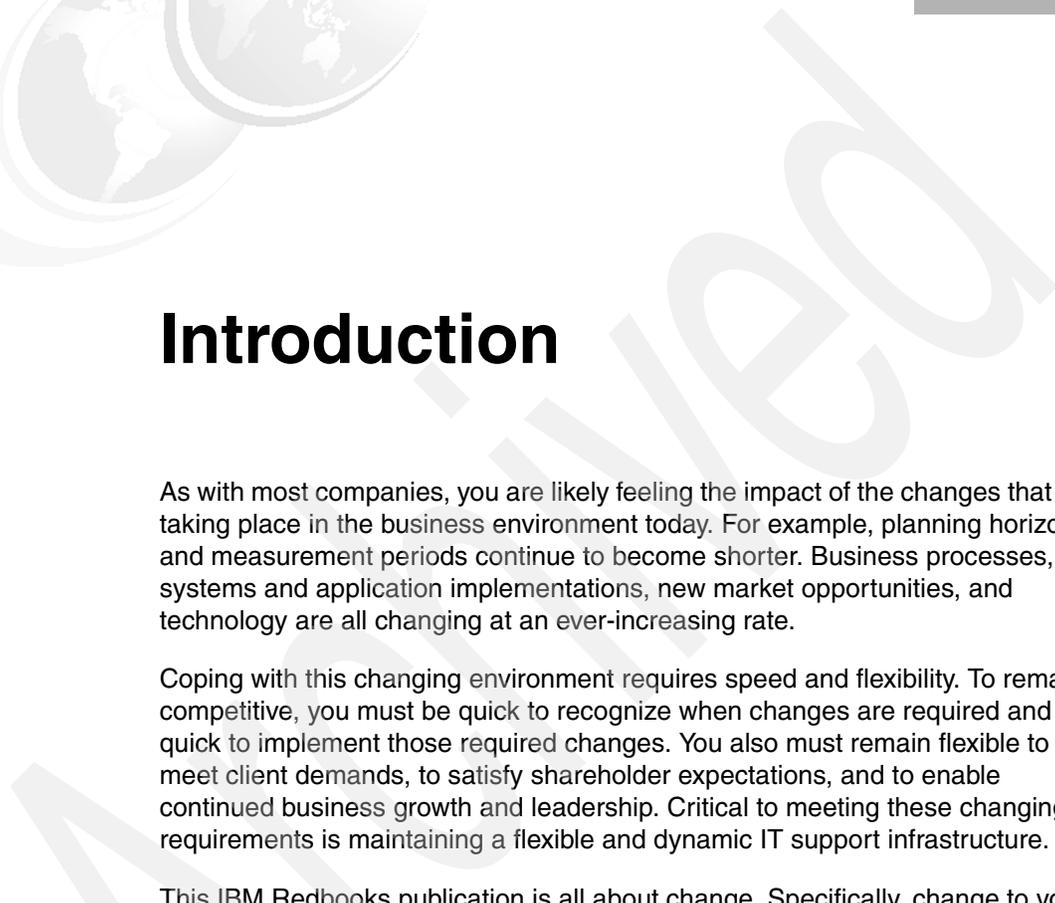
- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization

Dept. HYTD Mail Station P099

2455 South Road

Poughkeepsie, NY 12601-5400



Introduction

As with most companies, you are likely feeling the impact of the changes that are taking place in the business environment today. For example, planning horizons and measurement periods continue to become shorter. Business processes, systems and application implementations, new market opportunities, and technology are all changing at an ever-increasing rate.

Coping with this changing environment requires speed and flexibility. To remain competitive, you must be quick to recognize when changes are required and be quick to implement those required changes. You also must remain flexible to meet client demands, to satisfy shareholder expectations, and to enable continued business growth and leadership. Critical to meeting these changing requirements is maintaining a flexible and dynamic IT support infrastructure.

This IBM Redbooks publication is all about change. Specifically, change to your IT support infrastructure, and your database management system (DBMS), which is the foundation of that infrastructure. It is critical that your DBMS can house, organize, manage with integrity, and deliver to you, on demand, the data that is collected and stored by your organization. However, it also needs to change with the business requirements and needs to be enabled with a robust set of tools and applications that can implement the change and growth in your organization.

In addition to the DBMS, the migration of applications must be considered. Converting an application across different platforms and different databases is certainly not a trivial task. The decision to convert is generally made at high level

and when there is full justification in terms of costs and expected returns on investment. The major issues that bring up the need to convert (and are the main components for building a business case), are as follows:

- ▶ Performance

Aspects of performance include scalability, availability, data movement, response time, and the ability to support multiple query workloads.

- ▶ Configuration costs

Realistic cost assessment is based on overall development, maintenance, tuning cost, and the full exploitation of current investments, both in terms of skill sets and reduced licence costs.

- ▶ Data integration

Market trends are highlighting the importance of enterprise servers and the need to avoid data structure fragmentation to increase the value of business-critical systems. Fragmentation may cause cross-functional currency and consistency issues, and hamper innovation.

- ▶ Data infrastructure

Data is no longer an application-specific resource, but an enterprise-wide tool to provide critical business information for a competitive advantage. Often, it is not enough to navigate through the data, but is necessary to invest in their infrastructure to integrate enterprise views of data more easily.

Changing your DBMS platform can be a big challenge. Complexity, total cost, and the risk of downtime are key considerations when deciding whether or not to start such a project. But with good planning, education, and product knowledge, these concerns can be minimized.

In this IBM Redbooks publication, we discuss considerations and describe a methodology for transitioning from Oracle 10g to the Informix Dynamic Server (IDS). We focus on the basic topic areas of data, applications, and administration, providing information about the differences in features and functionality in areas such as data types, DML, DDL, and Stored Procedures. Understanding the features and functionality of the two products will assist in developing a migration plan.

We provide a migration methodology and discuss the processes for installing and using the IBM Migration Toolkit (MTK) to migrate the database objects and data from Oracle to IDS. We also illustrate, with examples, how to convert stored procedures, functions, and triggers. Application programming and migration considerations are also discussed.

In addition, you will find script conversion samples for data loading, database administration, and reports. There is also information regarding procedures and tips for migration testing and database tuning. The laboratory examples are performed under Oracle 10g and IDS Version 11.5. However, the migration process and examples can also be applied to Oracle 7, 8, and 9i.

With this information, you can gather and document your migration requirements, develop your required transition methodology, and plan and execute the migration activities in an orderly and cost-effective manner.

Archived

1.1 Migrating

Migrating from Oracle to IDS requires a certain level of knowledge of both environments. Because you are reading this book, you are likely already using, and are familiar with, Oracle. Therefore, the purpose of this section is to start your education with an introduction to the Informix Dynamic Server (IDS). It is not meant to be an exhaustive description of IDS, but a high level overview of some of the functions and features to help you understand the structure and capabilities available, so you can more easily relate it to the Oracle environment.

Many of these features are unique in the industry today, enabling clients to use information in new and more efficient ways to create a business advantage. It is designed to help businesses better use 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). 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 volumes, and numbers of users. 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. 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

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.

1.2 Positioning IDS

IBM Informix Dynamic Server 11 (IDS 11) combines the robustness, high performance, availability, and scalability needed in businesses today.

Complex, mission-critical database management applications typically require a combination of OLTP and batch and decision-support operations, including 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 use efficiently all the capabilities of the existing hardware and software configuration, including single and multiprocessor architectures. 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. It provides 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.

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.

1.3 Informix Dynamic Server editions

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 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.
Scalable	2 CPUs/4 GB RAM maximum.	<ul style="list-style-type: none"> ▶ 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)

Not all license terms and conditions are contained in this document. See an authorized IBM Marketing Representative, IBM Business Partner, or the following Web page for details:

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

Express Edition

Targeted toward small to mid-size businesses or applications requiring enterprise-level stability, manageability, and security, 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 not only to install the data server but also to configure an operational instance with a set of default parameters.

Workgroup Edition

Informix Dynamic Server Workgroup Edition (IDS-WGE) is for any 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. In addition, its ability to process more complicated SQL operations on larger databases is enhanced because of the Parallel Data Query (PDQ) and Memory Grant Manager (MGM) components. With these, data server resources can be pre-reserved and then fully deployed without interruption to process any given SQL operation.

To manage large databases requires the ability to insert or extract data quickly, as well as perform full or targeted backups. IDS-WGE includes functionality to do both. For example, you can use the high performance loader (HPL) 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. Also included is the ON-Bar

utility suite for partial or full instance backups and restores. These can be multi-threaded, so 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.

If you want to provide continuation of database services in the event of a natural or human-made outage, you 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.

If you want to use Enterprise Replication (ER) technology to distribute and consolidate data throughout your environment, IDS-WGE servers can be leaf or target nodes of replication events.

Enterprise Edition

Informix Dynamic Server Enterprise Edition (IDS-EE) 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.

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 handle automatically near simultaneous changes to the same object on different servers.

ER is flexible, and 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 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.

Developer Edition

For application development and testing only, the developer edition (IDS-DE) packs a full suite of functionality into an attractive price point: free. IDS-DE includes all the functionality available in the enterprise edition. It contains scalability limits for non-production use including processing, memory, and storage limitations. It is available on a wide range of operating systems in 32-bit and 64-bit versions where appropriate. IDS-DE comes without formal support from IBM. A number of forums exist in the IDS development community, which you can join for help and support using IDS-DE. You can upgrade IDS-DE directly to any other edition simply by installing the new data server binaries.

1.4 IDS functionality

In this section we provide a brief overview of just a few of the IDS features and functionality for familiarity.

Replication and high availability

In many respects, the functionality included here is one of the key advantages of IDS. The high availability data replication (HDR) technology has 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 needed to support 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 IDS ontape utility and the ON-Bar suite. 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 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, however. You

can have as many RSS instances as you like. And although an RSS instance cannot be promoted to become an HDR primary, it can become an HDR secondary, after which it can be promoted to primary if needed.

However, RSS instances must be considered disaster recovery instances, not high availability (HA) instances. RSS instances are deployed to expand the real-time failover capability of an HDR environment. Another use is to provide promotable redundancy. In the event of a primary failure, the 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. SDS instances provide redundancy and failover options because they can be anywhere in the network. However, they do not protect against disk-related failures.

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

Performance

A number of performance features are built into IDS. The following list details a few examples:

- ▶ The high performance loader (HPL) utility can load data 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. 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.
- ▶ Parallel Data Query (PDQ) takes advantage of the CPU power provided by SMP systems and an IDS virtual processor to execute fan-out parallelism. PDQ is of greatest benefit to more complex analytical SQL operations. The operation is divided into a number of subtasks, which are given higher or lower priority for execution within the data servers 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. 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 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. This can provide a significant performance boost.
- ▶ The IDS cost-based optimizer determines the fastest way to retrieve data based on detailed statistical information about the data within the database generated by the UPDATE STATISTICS SQL command. 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.
- ▶ Interval checkpoints are operations that are conducted by the data server to insure the logical consistency of the data. However, with interval checkpoints, service interruptions have been virtually eliminated. Memory writes occur in the background allowing user transaction processing to continue. The net result is a significant improvement in the number of transactions that can be processed over time.
- ▶ SQL optimizer has the ability to rewrite SQL operations when it recognizes query plans that require entire index scans. This enables query results to be returned more quickly.

Security

IDS has security features to help businesses such as yours meet any regulatory requirements. The first is the addition of Label-Based 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.

Another capability enables the instance to invoke automatically a specific user-defined routine (UDR) whenever a session connects or disconnects from the instance. This means 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, setting the isolation level or the output location of optimizer reports, turning on (or off) monitoring functionality, and sending an alert.

IDS also has the ability to use filters in backup and restore operations. For example, it 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

IDS has a 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. It has tasks and sensors to gather information and execute operations.

Allowing a DBA more capability to maintain instances remotely is one of the administration goals. With the OpenAdmin Tool, 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.

IDS delivers what is referred to as an *administration free zone*. Database administration is integrated with the SQL Application Programming Interface (API). This enables the DBA to have more control over the instance and the ability to automate many tasks that typically require intervention. Many tasks can be scripted by using SQL and stored procedures, making those tasks platform independent and allowing a DBA to consolidate many of the tasks into a single environment, or to monitor many environments.

Extensibility

IDS provides a complete set of extensibility features, including support for new data types, routines, aggregates, and access methods. This object-relational extensibility supports transactional consistency and data integrity while simplifying database optimization and administration. Much of the functionality is delivered with IDS 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.

Several DataBlades are bundled as part of the data server, enabling application developers to enrich their applications quickly and easily. This includes the following blades:

- ▶ 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.

- ▶ 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 following list details some of the 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.

- ▶ IBM Informix TimeSeries Real-Time Loader

A companion component to the IBM Informix TimeSeries DataBlade, the TimeSeries Real-Time Loader is 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 manage complex geospatial information intelligently, 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. 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. While the IBM Informix Geodetic DataBlade is a for-charge item, the IBM Informix Spatial DataBlade is available at no charge to 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.
- ▶ IBM Informix Image Foundation DataBlade
This DataBlade provides functionality for the storage, retrieval, transformation, and format conversion of image-based data and metadata.
- ▶ IBM Informix C-ISAM® DataBlade
This DataBlade provides functionality to the storage and use of Indexed Sequential Access Method (ISAM)-based data.

The current list of available IBM Informix DataBlade technologies is available at the following Web page:

<http://www.ibm.com/informix>

Getting into the details

In the subsequent chapters of this book, we provide you with more detailed information about both Oracle 10g and IDS. Understanding these environments will better equip you to make your migration project a successful one.

Architectural overview

In this chapter we give you an architectural overview of both Oracle and the Informix Dynamic Server (IDS). This is meant to facilitate the understanding of both architectures, taking into consideration that the reader may already be familiar with either Oracle or IDS.

We include the following topics:

- ▶ Oracle and IDS architectural overview including the following topics:
 - Processes architecture
 - Memory architecture
 - Physical database architecture
 - Logical database architecture
 - Oracle data dictionary and IDS system catalog
 - Database connectivity
- ▶ IDS editions and licensing
- ▶ Terminology
- ▶ Reasons to select IDS

2.1 The basic architectures

It is useful to understand the differences between Oracle and IDS architectures before attempting the Oracle to IDS migration process. Both products include their own memory architecture, background processes, database-related files, and different configuration files. Both Oracle and IDS consist of an instance and the databases attached to that instance. This section provides a general description of the architectures of each vendor.

Figure 2-1 provides a visual overview of the Oracle architecture. The upper level shows the memory architecture, which includes the system global area memory (SGA) and the program global area memory (PGA). The middle level is the processes component, and the bottom level shows the database components.

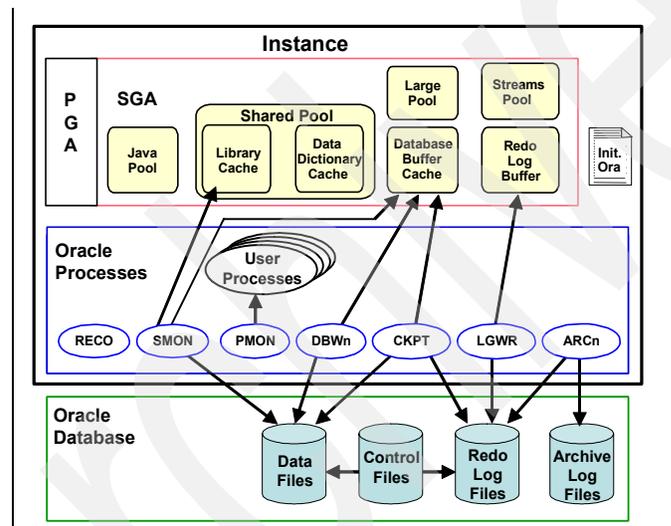


Figure 2-1 Oracle architecture overview

Figure 2-2 on page 17 provides a visual overview of the IDS Architecture. The upper level shows the memory architecture, the middle level the processes, and the bottom level shows the databases in the instance. As you can see, there can be multiple databases on one IDS instance.

Oracle does not support multiple databases in a single instance, but uses the concept of schemas to separate data. These schemas simulate multiple databases, but are not equivalent to having multiple databases.

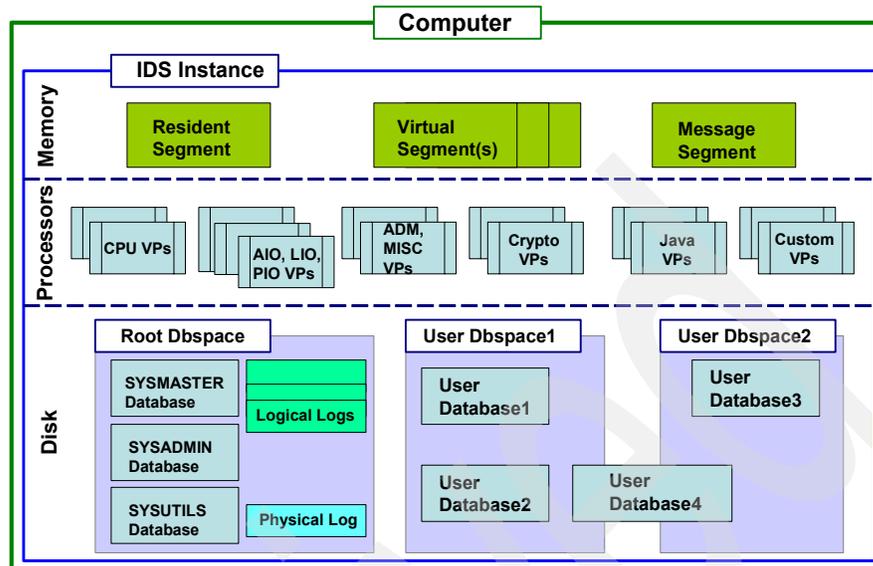


Figure 2-2 IDS architecture overview

2.1.1 Memory architectures

Shared memory is an operating-system feature that allows the database server threads and processes to share data by sharing access to pools of memory. The database server uses shared memory for the following purposes:

- ▶ Reducing memory usage and disk I/O
- ▶ Performing high-speed communication between processes

Shared memory enables the database server to reduce overall memory usage because the participating processes do not need to maintain private copies of the data that is in shared memory.

Shared memory reduces disk I/O, because buffers, which are managed as a common pool, are flushed on a database server-wide basis instead of a per-process basis. Furthermore, a virtual processor can often avoid reading data from disk because the data is already in shared memory as a result of an earlier read operation. The reduction in disk I/O reduces execution time.

The following subsections discuss the memory architecture in both Oracle and IDS. Oracle and IDS allocate and use memory for instance and database operation. There are various memory structures used for the various processes. In this section we give an overview about how memory is allocated and used in a simple Oracle and IDS server.

The memory architecture of an Oracle database consists of the memory area allocated to the Oracle instance and database upon startup. The amount of memory allocated is controlled by parameters defined in the Oracle configuration file. The memory architecture of IDS is similar, but slightly different from Oracle's.

Oracle

Oracle uses memory to run the code and share data among users. The two basic components of the Oracle memory structure are the Program Global Area (PGA) and the System Global Area (SGA). Figure 2-3 shows the memory architecture of an Oracle server.

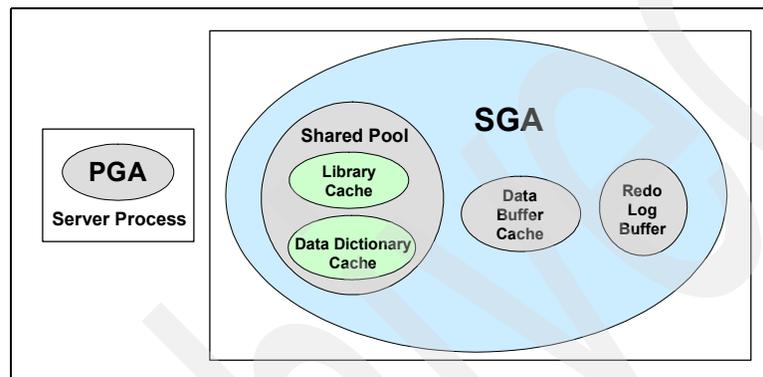


Figure 2-3 Oracle memory architecture

Program Global Area

The PGA is associated with the server process and contains the data and control information. For the dedicated server configuration, the primary contents of the PGA are the sort area, session information, cursor state, and stack space. This is non-sharable memory, which is writable only by the server process. The PGA is allocated whenever a server process starts. The total size of the PGA is controlled by the PGA_AGGREGATE_TARGET initialization parameter in version 10g.

System Global Area

The SGA is the shared memory region allocated for the entire Oracle instance. It is comprised of a group of shared memory structures in which the basic components are the shared pool, data buffer cache, and the redo log buffer. The shared pool contains the library cache, data dictionary cache, along with buffers for parallel execution messages and control structures. The library cache holds the SQL statement text, the parsed SQL statement, and the execution plan. The data dictionary cache contains reference information about tables, views, object definitions, and object privileges.

The shared pool size is controlled by the SHARED_POOL_SIZE initialization parameter. The data buffer cache stores the most recently used Oracle data blocks. Oracle reads the data blocks from the datafiles and places them in data buffers before processing the data. The number of buffers allocated is controlled by DB_CACHE_SIZE. The redo log buffer is a circular buffer that contains redo entries of the change information made to the database. These redo log buffers are written into the redo log files and are required for database recovery. The sizes of the redo log buffers are controlled by the LOG_BUFFER initialization parameter. The other memory structures of the SGA include the large pool (used for backup purposes) the Java™ pool (used for Java objects), and the streams pool (used for streams memory). For a shared server configuration in version 10g the session information and the sort areas are in SGA instead of PGA.

Informix Dynamic Server

IDS creates the following portions of the shared memory:

- ▶ Resident portion
- ▶ Virtual portion
- ▶ Message portion (for Inter Process Communication on UNIX)
- ▶ Virtual Extension portion

Figure 2-4 illustrates the shared memory segments in IDS.

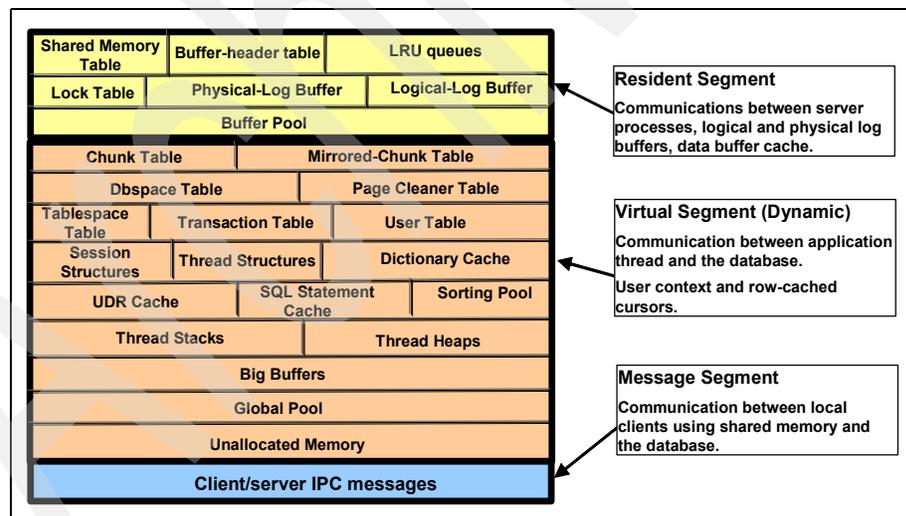


Figure 2-4 Shared memory segments in IDS

Resident portion of the shared memory

The resident portion of the IDS shared memory stores the following data structures that do not change in size while the database server is running:

- ▶ Shared-memory header

The shared-memory header contains a description of all other structures in shared memory, including internal tables and the buffer pool.

The shared-memory header also contains pointers to the locations of these structures. When a virtual processor first attaches to shared memory, it reads address information in the shared-memory header for directions to all other structures.

- ▶ Buffer pool

The buffer pool in the resident portion of shared memory contains buffers that store dbspace pages read from disk. The pool of buffers comprise the largest allocation of the resident portion of shared memory.

In IDS, the bufferpool is created at the instance level and one instance can have one or many user databases. You use the BUFFERPOOL configuration parameter in the IDS instance ONCONFIG parameter file to specify information about a buffer pool, including the number of buffers in the buffer pool.

If you are creating a dbspace with a non-default page size, the dbspace must have a corresponding buffer pool. For example, if you create a dbspace with a page size of 8 kilobytes, you must create a buffer pool with a page size of 8 kilobytes.

If a buffer pool for a non-default page size does not exist, the database server will automatically create a large-page buffer.

- ▶ Logical log buffer

The database server uses the logical log to store a record of changes to the database server data since the last dbspace backup. The logical log stores records that represent logical units of work for the database server. The logical log contains the following five types of log records, in addition to many others:

- SQL data definition statements for all databases
- SQL data manipulation statements for databases created with logging
- Record of a change to the logging status of a database
- Record of a checkpoint
- Record of a change to the configuration

The IDS logical log buffer is similar to the Oracle redo log buffer. The database server uses only one of the logical log buffers at a time. This buffer is the current logical log buffer. Before the database server flushes the current

logical log buffer to disk, it makes the second logical log buffer the current one so that it can continue writing while the first buffer is flushed. If the second logical log buffer fills before the first one finishes flushing, the third logical log buffer becomes the current one.

The LOGBUFF configuration parameter in the ONCONFIG file specifies the size of the logical log buffers. There is one set of logical log buffers for all the databases in the IDS instance. The recommended value for the size of a logical log buffer is 64 kilobytes.

► Physical log buffer

The physical log buffer holds before-images of the modified pages in the buffer pool. This is similar to the Oracle UNDO space. The before-images in the physical log and the logical log records enable the database server to restore consistency to its databases after a system failure.

The physical log buffer is actually two buffers. Double buffering permits the database server processes to write to the active physical log buffer while the other buffer is being flushed to the physical log on disk.

The PHYSBUFF parameter in the ONCONFIG file specifies the size of the physical log buffers. The default value for the physical log buffer size is 512 kilobytes.

► Lock table

Locks can prevent sessions from reading data until after a concurrent transaction is committed or rolled back. A lock is created when a user thread writes an entry in the lock table. The lock table is the pool of available locks, and a single transaction can own multiple locks.

The following information is stored in the lock table:

- The address of the transaction that owns the lock
- The type of lock (exclusive, update, shared, byte, or intent)
- The page or ROWID that is locked
- The table space where the lock is placed
- Information about bytes locked (byte-range locks for smart large objects)

The LOCKS configuration parameter specifies the initial size of the LOCK table. If the number of locks allocated by sessions exceeds the value specified in the LOCKS configuration parameter, the database server doubles the size of the lock table, up to 15 times.

Virtual Portion of the Shared Memory

The virtual portion of shared memory is expandable by the database server and can be paged out to disk by the operating system. As the database server executes, it automatically attaches additional operating-system segments, as needed, to the virtual portion. The database server uses memory pools to track memory allocations that are similar in type and size.

The database server allocates virtual shared memory for each of its subsystems (session pools, stacks, heaps, control blocks, system catalog, SPL routine caches, SQL statement cache, sort pools, and message buffers) from pools that track free space through a linked list. When the database server allocates a portion of memory, it first searches the pool free-list for a fragment of sufficient size. If it finds none, it brings new blocks into the pool from the virtual portion. When memory is freed, it goes back to the pool as a free fragment and remains there until the pool is destroyed. When the database server starts a session for a client application, for example, it allocates memory for the session pool. When the session terminates, the database server returns the allocated memory as free fragments.

The SHMVIRTSIZE parameter in the ONCONFIG file specifies initial size of the virtual shared-memory portion. SHMADD or EXTSHMADD configuration parameters specify the size of segments that are added later to the virtual shared memory.

The virtual portion of shared memory stores the following data:

- ▶ Internal tables
- ▶ Big buffers
- ▶ Session data
- ▶ Thread data (stacks and heaps)
- ▶ Data-distribution cache
- ▶ Dictionary cache
- ▶ SPL routine cache
- ▶ SQL statement cache
- ▶ Sorting pool
- ▶ Global pool

See the *Informix Dynamic Server Administrator's Guide*, G229-6359, for more information about these components.

Message portion of the shared memory

The database server allocates memory for the IPC communication portion of shared memory if you configure at least one of your connections as an IPC shared memory connection. The database server performs this allocation when you set up shared memory.

The communications portion contains the message buffers for local client applications that use shared memory to communicate with the database server. The size of the communications portion of shared memory equals approximately 12 kilobytes multiplied by the expected number of connections needed for shared-memory communications.

2.1.2 Process architectures

Now lets take a look at the process architectures of Oracle and IDS.

Oracle

There are two types of Oracle processes:

- ▶ User
- ▶ Background

Figure 2-5 illustrates these two types of processes in Oracle.

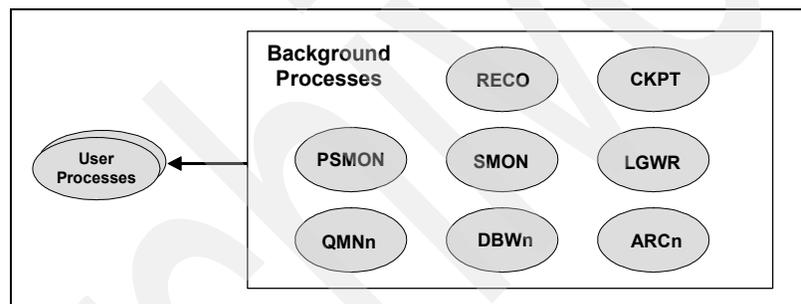


Figure 2-5 Oracle process architecture

User processes

Oracle creates a user process when the user or application connects to the database. For each user process, a server process is created by Oracle to handle the user process request to an Oracle instance. This architecture works when the client is on a different machine. When the client and the server are on the same machine, the user process and server process are combined into a single process. The function of the server process is to parse the SQL statement, read the Oracle data blocks from the datafiles to the data buffer, and return the result set to the client.

Background processes

Oracle requires a number of processes to be running in the background to be operational and open to users. These processes are as follows:

- ▶ Database writer (DBWR)

This background process writes all dirty data blocks from the database buffer cache to the datafiles on disk. The DBA can configure multiple DBWR processes to improve performance.
- ▶ Log writer (LGWR)

This is the process that handles writing data from the redo log buffer cache onto the redo log files.
- ▶ System monitor (SMON)

This process performs an instance recovery when the Oracle instance fails, and it coalesces smaller fragments of disk space together.
- ▶ Process Monitor (PMON)

This process cleans up any remaining Oracle processes resulting from a failing user process. Furthermore, it rolls back any uncommitted transactions that were performed by the user.
- ▶ Checkpoint (CKPT)

This process writes log sequence numbers to the database headers and control files.
- ▶ Recoverer (RECO)

This process automatically resolves failures in distributed transactions when using the distributed database configuration.
- ▶ Archiver (ARCn)

This process is used for ARCHIVELOG mode when automatic archiving is enabled, to copy redo log files to a designated storage device after a log switch.
- ▶ Queue Monitor (QMn)

This optional process monitors message queues when using Oracle Streams Advanced Queuing.

Informix Dynamic Server

Informix Dynamic Server processes are called virtual processors because the way they function is similar to the way that a CPU functions in a computer. Just as a CPU runs multiple operating-system processes to service multiple users, a database server virtual processor runs multiple threads to service multiple SQL client applications.

Threads

A thread is a task for a virtual processor in the same way that the virtual processor is a task for the CPU. The virtual processor is a task that the operating system schedules for execution on the CPU. A database server thread is a task that the virtual processor schedules internally for processing.

Threads are sometimes called lightweight processes because they are like processes, but they make fewer demands on the operating system. Database server virtual processors are multi-threaded because they run multiple concurrent threads.

A virtual processor runs threads on behalf of SQL client applications (session threads) and also to satisfy internal requirements (internal threads). In most cases, for each connection by a client application, the database server runs one session thread. The database server runs internal threads to accomplish, among other things, database I/O, logging I/O, page cleaning, and administrative tasks.

A user thread is a database server thread that services requests from client applications. User threads include session threads, called `sqlxexec` threads, which are the primary threads that the database server runs to service client applications. Figure 2-6 illustrates virtual processes (VPs) in IDS instance.

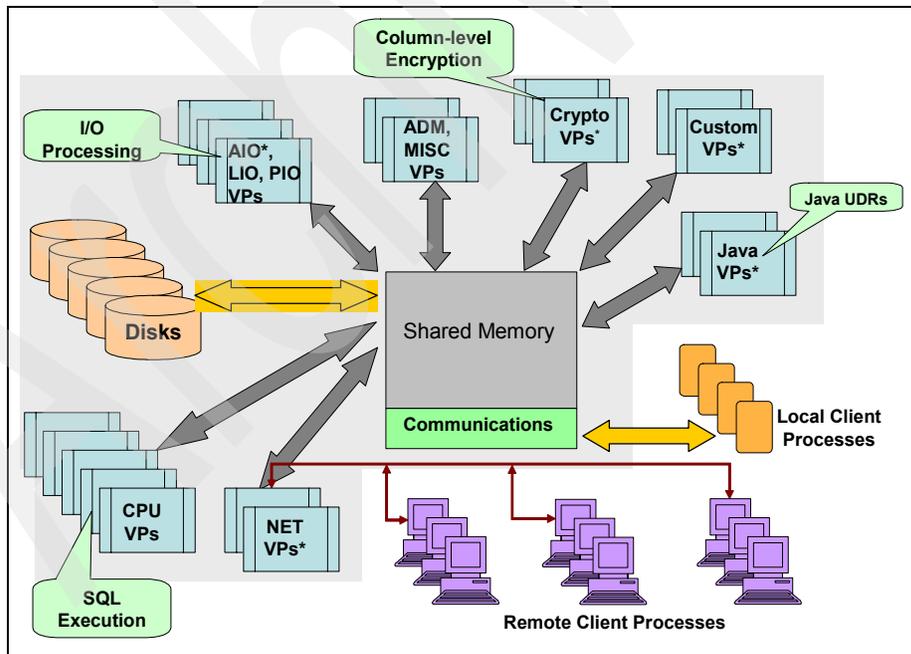


Figure 2-6 IDS instance with VPs

Types of VPs in IDS

A VP is a process that the operating system schedules for executing tasks. IDS has different classes of VPs, and each class is dedicated to processing certain types of threads.

The number of VPs of each class that you configure depends on the availability of physical processors (CPUs), hardware memory, and the database applications in use.

The following list details different types of VPs in IDS:

- ▶ CPU VP
Runs all session threads and some system threads. Runs thread for kernel asynchronous I/O (KAIO) where available. Can run a single poll thread, depending on the configuration.
- ▶ PIO VP
Writes to the physical log file (internal class) if it is in cooked disk space.
- ▶ LIO VP
Writes to the logical log files (internal class) if they are in a cooked disk space.
- ▶ AIO VP
Performs nonlogging disk I/O. If KAIO is used, AIO virtual processors perform I/O to cooked disk spaces.
- ▶ SHM VP
Performs shared memory communication.
- ▶ TLI VP
Uses the transport layer interface (TLI) to perform network communication.
- ▶ SOC VP
Uses sockets to perform network communication.
- ▶ OPT VP
Performs I/O to optical disk in UNIX.
- ▶ ADM VP
Performs administrative functions.
- ▶ ADT VP
Performs auditing functions.
- ▶ MSC VP
Services requests for system calls that require a large stack.

- ▶ CSM VP
Communications Support Module. Performs communications support service operations.
- ▶ Encrypt VP
Used by the database server when encryption or decryption functions are called.
- ▶ Java VP (JVP)
Contains the Java Virtual Machine (JVM™), and executes Java UDRs.
- ▶ User Defined VP
Runs user-defined routines in a thread-safe manner so that if the routine fails, the database server is unaffected. Specified with the VPCLASS configuration parameter.

Advantages of IDS Virtual Processors (VPs)

Compared to a database server process that services a single client application, the dynamic, multi-threaded nature of IDS virtual processors provide the following advantages:

- ▶ Can share processing.
- ▶ Saves memory and resources.
- ▶ Performs parallel processing.
- ▶ You can add virtual processors to meet increasing demands for service while the database server is running, and then terminate them when they are no longer needed.
- ▶ You can bind virtual processors to CPUs.

See *Informix Dynamic Server Administrator's Guide*, G229-6359 for more information about the IDS Virtual Processors.

2.1.3 Physical database structures

In this section we look at the Oracle and IDS physical database structures.

Oracle

The physical structure of Oracle database consists of datafiles, control files, redo log files, archive log files, parameter file, alert log files, and trace log files.

Figure 2-7 illustrates these components of a physical Oracle database.

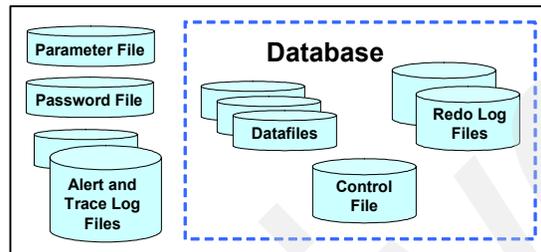


Figure 2-7 Oracle database files

Datafiles

Every Oracle database consists of one or more table spaces. Each table space is a logical collection of one or more datafiles. Oracle data objects (tables and indexes) are stored in the table spaces. You can expand the size of the table space by either adding more datafiles or by resizing the datafiles that are already added to the table space.

An Oracle datafile can be set to grow automatically as, and when, more space is needed.

Control files

Every Oracle database includes a control file. A control file records the changes made to the physical structure of the database. For example, if you add a new datafile to a table space, an entry is made in the control file to record this change.

A control file contains the following information:

- ▶ Database name
- ▶ Name and path of all the redo log files
- ▶ Name and path of all the datafiles in the table spaces
- ▶ Database creation time

Oracle can maintain multiple copies of the control files to protect itself from disk failures.

Redo log files and archive log files

Oracle redo log files store transactional records which are also known as redo records. An Oracle database consists of two or more redo log files that can be multiplexed. That is, two or more copies of the redo logs can be maintained to prevent losing the redo files because of disk failures. The redo logs are used to recover the database from system and media failures.

When the Oracle database is running in archive log mode, you can archive the redo log files. Archived redo log files are called archive log files.

Initialization parameter files

The Oracle initialization parameter file contains the values for many initialization parameters used to allocate memory and start the process for the instance to run. When you start an Oracle database instance, Oracle reads the initialization parameters from the initialization parameter file. You can also choose to store the initialization parameters in a binary server parameter file called SPFILE.

Alert log file and trace log files

The alert and trace log files are the diagnostics files used by the Oracle instance to record all the dump information of the database such as internal errors, block corruption errors, and so forth.

These log files are used by Oracle DBAs and Oracle support team for troubleshooting issues with the database.

Password file

The password file is a security file used for authenticating which users are permitted to start up or shut down an instance or perform other privileged maintenance on a database with SYSDBA or SYSOPER privileges, and additionally, OSDBA or OSOPER privileges.

Informix Dynamic Server

The physical structures of a IDS instance include chunks, the ONCONFIG configuration file, logical log files, a physical log file, and a message file.

Chunks

A chunk in IDS is same as a datafile in Oracle. A chunk is the largest unit of physical disk dedicated to database server data storage. The maximum size of an individual chunk is four terabytes. A chunk can be a regular operating system file (cooked file) or a raw disk device. We will discuss more about chunks when we talk about dbspaces later in this chapter.

Configuration file (ONCONFIG file)

It is important to know that one Oracle instance consists of only one Oracle database, whereas one IDS Instance (also known as IDS Database Server) consists of one or more IDS databases.

Each IDS configuration file consists of configuration parameters that are used to initialize the IDS instance. The IDS instance configuration file is also known as ONCONFIG file.

You can set the ONCONFIG environment variable to specify the name and location of the IDS instance configuration file, as shown in the following command:

```
set ONCONFIG=$INFORMIXDIR/etc/ONCONFIG.prod
```

If you start the database server with **oninit** command and do not explicitly set the ONCONFIG environment variable, the database server looks for configuration values in the \$INFORMIXDIR/etc/onconfig.std file.

Logical log files

To keep a history of transactions and database server changes since the time of the last storage-space backup, the database server generates log records. The database server stores the log records in the logical log, a circular file that is composed of three or more logical log files. The log is called logical because the log records represent logical operations of the database server, as opposed to physical operations. At any time, the combination of a storage-space backup plus logical log backup contains a complete copy of your database server data.

In Oracle, each database instance has its own set of redo log files, whereas in IDS all the databases in the instance share one set of logical log files.

When you initialize or restart the database server, it creates the number of logical log files that you specify in the LOGFILES configuration parameter. The size of the logical log files is specified with LOGSIZE parameter in the ONCONFIG parameter file.

When the database server initializes disk space, it places the logical log files in the default root dbspace. You have no control over this action. To improve performance (specifically, to reduce the number of writes to the root dbspace and minimize contention), move the logical log files out of the root dbspace to a dbspace on a disk that is not shared by active tables or the physical log.

To improve performance further, separate the logical log files into two groups and store them on two separate disks (neither of which contains data). For example, if you have six logical log files, you might locate files 1, 3, and 5 on disk 1, and files

2, 4, and 6 on disk 2. This arrangement improves performance because the same disk drive never has to handle writes to the current logical log file and backups to tape at the same time.

The logical log files contain critical information and should be mirrored for maximum data protection. If you move logical log files to a different dbspace, plan to start mirroring on that dbspace.

Physical log file

The physical log is used to record before images (first copy) of pages that have been modified in shared memory. This is same as the Oracle UNDO space.

When the database server initializes disk space, it places the physical log in the default root dbspace. To improve performance, you can move the physical log out of the root dbspace to another dbspace, preferably to a disk that does not contain active tables or the logical log files.

The value stored in the ONCONFIG parameter PHYSFILE defines the size of your physical log when the database server is initially created. Once the database server is online, use the onparams utility to change the physical log location and size.

Note: See the *IDS Administrator's Guide*, G229-6359, for more information about logical and physical log files.

Message log file

The database server writes status and error information to the message log file. You specify the filename and location of the message log with the MSGPATH configuration parameter. A message log file is similar to the Oracle alert log file.

The default name and location of the message log is on UNIX is \$INFORMIXDIR/tmp/online.log.

Reserved Pages in Root DB Space

The first 12 pages of the initial chunk of the root dbspace are reserved pages. Each reserved page contains specific control and tracking information used by the database server. These reserved pages serve approximately, but not exactly, the same function of the control file in Oracle. You can obtain a listing of the contents of the reserved pages by executing the **oncheck -pr** command.

2.1.4 Logical database structures

In this section we look at the Oracle and IDS logical database structures.

Oracle

Oracle logical database structures include data blocks, extents, segments, and table spaces.

Data block

Data block in Oracle is same as page in IDS. This is the smallest unit of I/O that is read/written to disk. The default Oracle block size is specified with the DB_BLOCK_SIZE initialization parameter. The default value of db_block_size is 8192 bytes and it can range from 2048 to 32768 bytes.

Extent

An extent is collection of contiguous data blocks. Database space for the tables and indexes is allocated in units called extents. In Oracle, you can specify the initial and next extent size of a table or index at table or index creation time.

Segment

A segment is a logical collection of extents that belong to a specific database object, such as a table or index. There are four types of segments in Oracle:

- ▶ Data Segment
Each table has its own data segment in which to store its data.
- ▶ Index Segment
Each index has its own segment in which to store its data.
- ▶ Temporary Segment
Oracle creates temporary segments when it needs space for certain operations, such as sorts and grouping.
- ▶ Rollback Segment
In earlier versions of Oracle (and still supported in 10g) Oracle allowed users to use rollback segments to manage undo space. Oracle supports automatic undo space management, which is the recommended method of managing undo space.

Tablespace

An Oracle database is a logical collection of storage units called tablespaces. Each tablespace is used to group together logically database objects such as tables and indexes. And there are two types of tablespaces:

- ▶ Smallfile tablespace
- ▶ Bigfile tablespace

A smallfile tablespace is a traditional Oracle tablespace. The name comes from how you allocate space to the tablespace. As the name implies, a smallfile tablespace is a collection of small datafiles. You can add space to a smallfile tablespace either by adding new datafiles to it or by extending the size of already added datafiles.

Every Oracle database has two default tablespaces, SYSTEM and SYSAUX. Both of these tablespaces are created as smallfile tablespaces.

A bigfile tablespace contains only a single large datafile. This is useful in 64-bit environments where you can create one large datafile on the operating system and attach just one large datafile to the database.

Informix Dynamic Server

IDS logical database structures include pages, extents, tablespaces, and dbspaces.

Page

A page is the minimum I/O that is read/written to disk. The default size of a page is 2 KB on most UNIX systems, and 4 KB on AIX and Windows.

IDS introduced configurable page sizes in IDS version 10, which allows users to create dbspaces with multiple page sizes up to 16 K.

Extent

Disk space for a table or index is allocated in units called extents. An extent is an amount of contiguous pages on disk. Extent size is specified for each table or index when the table or index is created.

The default size of initial extent and next extent is four times the size of the page in the dbspace.

Tablespace

An IDS tablespace is the same as an Oracle segment. A tablespace in IDS is a logical collection of extents that are allocated for a database object.

Dbospace

IDS dbospace is similar to Oracle tablespace, with some differences. A dbospace is a logical collection of one or more chunks. It can have between 1 and 32,767 chunks, and a dbospace chunk can be a cooked file or a raw device.

Important characteristics of IDS dbospaces are as follows:

- ▶ IDS dbospaces can contain the objects of one or more databases, whereas an Oracle tablespace can contain the objects of only one database.
- ▶ Every IDS instance contains a default dbospace called root dbospace, and logical and physical logs are created in the root dbospace by default.

A dbospace can be of type dbospace, temporary dbospace, simple blobospace, smartblob space, and external dbospace.

- ▶ **Dbospace**

An IDS dbospace is a regular dbospace that can contain tables, indexes, logical logs, and physical logs. It is recommended to move the logical and physical logs from root dbospace to user defined dbospace for better performance.

You need to create at least one user dbospace before you create an IDS database. Specify the name of the default dbospace for each database, with the `<dbospace>` option, when you create the database (with the `CREATE DATABASE` command). If you do not specify the `in <dbospace>` option at the time of the database creation, the database is created in the default root dbospace. This is not recommended.

- ▶ **Temporary dbospace**

A temporary dbospace is a dbospace reserved for the storage of temporary tables. This temporary dbospace is temporary only in the sense that the database server does not preserve any of the dbospace contents when the database server shuts down abnormally.

The database server does not perform logical or physical logging for temporary dbospaces. Because temporary dbospaces are not physically logged, fewer checkpoints and I/O operations occur, which improves performance.

- ▶ **BlobSpace dbospace**

A blobospace is a type of dbospace that is composed of one or more chunks that store only simple large objects. Simple large objects consist of TEXT or BYTE data types from any columns or any tables (from any database).

A blobpage is the basic unit of storage for blob data types stored in a blobospace. It is configured to be a multiple of the system page size. The database server writes data stored in a blobospace directly to disk. That is, this data does not pass through resident shared memory.

- ▶ SBspace dbspace

An SBspace is a special type of dbspace composed of one or more chunks that store smart large objects that are of type BLOB (binary large object), CLOB (character large object), or a user-defined data type (UDT).

Smart large object pages have the same size and format as standard system data pages. If buffering is turned on in the database, the database server uses private buffers in the virtual portion of shared memory for buffering sbospace pages.

2.1.5 Data dictionary and system catalog

Every RDBMS has a form of metadata that describes the database and its objects. Essentially, the metadata contains information about the logical and physical structure of the database, integrity constraints, user and schema information, authorization, privilege information, and so on.

Oracle

In the Oracle database, metadata is stored in a set of read-only tables and views called the data dictionary. These tables and views are updated by the Oracle server. The data dictionary is owned by the user SYS and stored in the SYSTEM tablespace. The base tables are all normalized and are seldom accessed directly, hence, user accessible views are created using the catalog.sql script. The data dictionary is organized under three qualifiers:

- ▶ USER_xxx views

The USER_xxx views show the object information owned by the current user.

- ▶ ALL_xxx views

The ALL_xxx views show all the object information that can be accessed by the current user.

- ▶ DBA_xxx views

The DBA_xxx view is the database administrator view and contains information about all the objects in the database.

Apart from the data dictionary, Oracle maintains another set of virtual tables called the dynamic performance views. The views created on them are prefixed by V\$. These views are called the fixed views, and are available when the instance is started, without the need of the database to be opened.

Informix Dynamic Server

In IDS, the data dictionary is divided into two parts, system catalog tables and the system monitoring interface (SMI) database.

System catalog tables

In IDS, the system catalog tables are automatically created when you create a database. Each system catalog table contains specific information about elements in the database, much the same as the Oracle data dictionary.

System catalog tables track objects such as the following database objects:

- ▶ Tables, views, sequences, synonyms, and sequence objects
- ▶ Columns, constraints, indexes, and fragments
- ▶ Triggers
- ▶ Procedures, functions, routines, and associated messages
- ▶ Authorized users and privileges
- ▶ User-defined routines
- ▶ Data types and casts (IDS)
- ▶ Aggregate functions (IDS)
- ▶ Access methods and operator classes (IDS)
- ▶ Inheritance relationships (IDS)
- ▶ External optimizer directives (IDS)

System monitoring interface

In addition to the system catalog tables, IDS also maintains a separate system database called system monitoring interface (SMI). Each IDS instance contains one SMI database. It is created when you initialize the database server instance for the first time.

The SMI consists of tables and pseudo-tables that the database server maintains automatically. While the SMI tables appear to the user as tables, they are not recorded on disk like normal tables. Instead, the database server constructs the tables in memory, on demand, based on information in shared memory at that instant. When you query an SMI table, the database server reads information from these shared-memory structures. Because the database server continually updates the data in shared memory, the information that SMI provides lets you examine the current state of your database server.

The SMI tables provide information about the following topics:

- ▶ Auditing
- ▶ Checkpoints
- ▶ Chunk I/O
- ▶ Chunks
- ▶ Database-logging status
- ▶ Dbspaces

- ▶ Disk usage
- ▶ Environment variables
- ▶ Extents
- ▶ Locks
- ▶ Networks
- ▶ SQL statement cache statistics
- ▶ SQL tracing
- ▶ System profiling
- ▶ Tables
- ▶ User profiling
- ▶ Virtual-processor CPU usage

The data in the SMI tables changes dynamically as users access and modify databases that the database server manages.

SMI tables can be queried directly with select statements. IDS system monitoring utilities, such as onstat, also read data from SMI tables.

Table 2-1 gives you some examples of Oracle data dictionary entries, and equivalent IDS catalog tables.

Table 2-1 Oracle and IDS system catalog tables

Oracle Data Dictionary	IDS System Catalog
DBA_TABLES	SYSTABLES
DBA_TAB_COLUMNS	SYSCOLUMNS
DBA_TABLESPACES	SYSDbspaces (SMI Table)
DBA_INDEXES	SYSINDICES
DBA_TAB_PRIVS	SYSTABAUTH
DBA_TRIGGERS	SYSTRIGGERS
DBA_VIEWS	SYSVIEWS
DBA_SEQUENCES	SYSSEQUENCES
DBA_PROCEDURES	SYSPROCEDURES

Note: The complete list of IDS system catalog tables can be found in the *IDS 11.5 Guide to SQL: Reference*, G229-6374. The complete list of SMI tables can be found in *the IDS 11.5 Administrator's Reference*, G229-6360.

2.1.6 Database server communication

In this section we give an overview of the communication architecture that enables simple client-server communication in Oracle and IDS environments, and some of the tools used to connect to the database.

Database access

This section discusses the tools and utilities used in Oracle and IDS for connecting to the database on the database server, and from a client application.

Oracle

Oracle provides the SQL*Plus tool for command line access to the database server. SQL*Plus also comes with a GUI version. The Oracle client installation installs the SQL*Plus tool, Oracle Net Services software, ODBC drivers, and other tools. This software provides basic client-server communication to access the database server.

The client-server communication in Oracle Server is handled by Oracle Net Services. Oracle Net Services support communications on all major protocols. The Oracle Net Services provide a communication interface between the client user process and the Oracle server process, enabling the data transmission and message exchange between Oracle server and client. The Oracle Net Services uses a technology called Transparent Network Substrate (TNS) to perform these tasks. The TNS enables peer-to-peer application connectivity, where the two nodes communicate with each other directly.

The Oracle Net Services provides the listener process that resides in the Oracle server, which listens for incoming client connection requests, and maps them to the Oracle instance. The listener is configured with one or more protocol addresses. The client is configured with one of these protocol address and can send connection requests to listener. A configuration file, listener.ora, is maintained in the Oracle server that contains the protocol address, database service information, and listener configuration parameters. The listener process is controlled by the LSNRCTL utility. The LSNRCTL utility reads the listener.ora file and starts the listener process. The server services information in the client is maintained in a file called tnsnames.ora. Oracle Net Configuration Assistant and Net Manager are graphical utilities used to configure the Oracle Net Services such as listener, service naming, and so on.

Informix Dynamic Server

IDS provides DB Access Utility, a cursor-based utility installed on the database server for local SQL connections to run ad-hoc queries. Application programmers can install IBM Informix Client Software Development Kit (Client SDK) on the remote clients to connect to IDS databases remotely. IDS databases can also be connected to using JDBC™, as can Oracle.

DB Access Utility

An IDS installation includes the DBACCESS utility. DBACCESS is a cursor-based tool, providing a user interface for entering, executing, and debugging SQL statements and Stored Procedure Language (SPL) routines. You can also start DBACCESS in menu mode and select options from the menus.

As examples, DBACCESS can be used for the following tasks:

- ▶ Ad hoc queries that are executed infrequently.
- ▶ Connecting to one or more databases and displaying information about a database.
- ▶ Displaying system catalog tables and the information schema.
- ▶ Practicing the statements and examples provided in the *IBM Informix Guide to SQL: Tutorial*, G229-6427.

Client Software Development Kit (Client SDK)

The IBM Informix Client Software Development Kit (Client SDK) includes several application programming interfaces (APIs) that developers can use to write applications for Informix database servers in ESQL, C, Java, and .Net.

Applications that run on client computers require Informix Connect to access database servers. Informix Connect is a runtime connectivity product composed of runtime libraries that are included in Client SDK.

The components included in client SDK are as follows:

- ▶ IBM Informix .NET provider (Windows only)
- ▶ IBM Informix add-Ins for visual studio 2003 and 2005
- ▶ IBM Informix ESQL/C with XA support
- ▶ The finderr utility on UNIX systems
- ▶ The Informix error messages utility on Windows systems
- ▶ The global security kit
- ▶ IBM Informix Object Interface for C++
- ▶ IBM Informix ODBC driver with MTS support
- ▶ IBM Informix OLE DB provider (Windows only)
- ▶ The ILogin utility (Windows only)
- ▶ IBM Informix password communications support module (CSM)

SQLHOSTS file and registry key

IDS supports TCP/IP, IPX/SPX, sockets, shared memory, stream pipe, and named pipe connections. The Informix SQLHOSTS file (on UNIX) or SQLHOSTS registry key (on Windows) contains client/server connectivity information that enables a client application to find and connect to any Informix database server in the network.

On UNIX, the sqlhosts file resides in the \$INFORMIXDIR/etc directory by default. As an alternative, you can set the INFORMIXSQLHOSTS environment variable to the full path name and filename of a file that contains the sqlhosts file information. On UNIX, each computer that hosts a database server or a client must have an sqlhosts file. Each entry (each line) in the sqlhosts file contains the sqlhosts information for one database server.

On Windows, When you install the database server, the setup program creates the following key in the Windows registry:

```
HKEY_LOCAL_MACHINE\SOFTWARE\INFORMIX\SQLHOSTS
```

This branch of the HKEY_LOCAL_MACHINE subtree stores the sqlhosts information. Each key on the SQLHOSTS branch is the name of a database server. When you click the database server name, the registry displays the values of the HOST, OPTIONS, PROTOCOL, and SERVICE fields for that particular database server.

Each computer that hosts a database server or a client must include the connectivity information either in the sqlhosts registry key or in a central registry. When the client application runs on the same computer as the database server, they share a single sqlhosts registry key.

Example 2-1 depicts the an SQLHOSTS file on a UNIX server.

Example 2-1 Sample SQLHOSTS file entries

dbservername	nettype	hostname	servicename	options
inst_tcp	onsoctcp	localhost	inst_svc	b=8192
inst_tcp	onsoctcp	192.1.1.11	1523	

2.2 IDS licensing

IDS can be licensed through one of the following three pricing metrics, depending on your needs.

Value unit

This is also known as processor-based pricing. It is calculated on the number of processor cores in the physical server multiplied by the corresponding value units based on the processor architecture. This could be considered an unlimited user or connection license and is usually the optimal choice when the user or session load cannot be controlled or counted.

Authorized user

Authorized user is a single named user accessing one installation of IDS on a single physical server. That authorized user can establish multiple connections to an IDS instance on the server. Each connection is for the exclusive use of that one authorized user from a single client device.

Concurrent session

Concurrent session is a single logical connection from a client device to an IDS instance on a single physical server. Each connection, whether active or not, requires a license, regardless of whether it comes from one client device with multiple users or a single user establishing multiple connections. If connection concentrators or multiplexers are used in the application pathway, the number of concurrent sessions is counted from the client device, not at the IDS level.

Important: The previous descriptions provide only a summary of the licensing definitions. They are not intended to be full and legally binding. For a full and complete description, refer to the IDS licensing agreement. Not all pricing models are available for all IDS editions.

2.3 Terminology

Before getting into the migration process, a clear understanding of the terminologies used in Oracle and IDS can help you map functionality used in Oracle and IDS. This section discusses some of the terminology used by Oracle and IDS, and how they map to each other.

Terminology mapping

Table 2-2 on page 42 provides a quick reference of some of the commonly used terminology in Oracle and IDS.

Table 2-2 Oracle and IDS terminology mapping

Oracle	IDS
Instance	Instance
Database	Database
Initialization File (init.ora file)	Configuration File (onconfig file)
Tablespace	DB Space
Data block	Page
Extent	Extent
Segment	Table space
Datafile	Chunk
Redo Log File	Logical Log File
PL/SQL	SPL (Stored Procedural Language)
Data Buffer	Buffer Pool
SGA	Shared Memory
Data Dictionary	System Catalog and SYSMASTER Database
UNDO Space	Physical Log
Hints	Optimizer Directives
Data Dictionary Cache	Dictionary Cache
SYSTEM table space	Root DB Space
Background Process	Virtual Processor
Partitioning	Fragmentation
Temporary Tablespace	Temporary DB Space
Multiplex	Mirror
Oracle Parallel Query	Parallel Data Query (PDQ)
Archive Log	Logical Log Backup
Oracle Lable Security	Informix Lable Based Access Control (LBAC)
Checkpoint	Checkpoint

Migration methodology

Database migration requires proper planning, regardless of how it is performed or the resources used. Unexpected delays in project execution can be avoided by following well-planned migration procedures. In this chapter we describe an IBM migration methodology and the resources that are available to assist you in making that migration a success.

The following topics are discussed in this chapter:

- ▶ Pre-migration steps
- ▶ Migration
- ▶ Post-migration steps

3.1 An IBM migration methodology

The IBM Data Management Services Migration Center has developed a best-practices methodology to help clients develop a plan for how to best migrate their database environment to Informix. You can use this methodology to perform your own migration project, or you can contract the fee-based services of the IBM Migration Center.

The migration team from IBM provides free assistance for many pre-migration tasks. The team consists of technical specialists whose mission is to facilitate and assist with all phases of a migration to Informix. The team has assisted hundreds of clients with their migrations and has database administration and application development skills for the source databases (Oracle, SQL Server®, and Sybase) that will be migrated to Informix.

Some of the tasks that the migration team provides assistance with are as follows:

- ▶ Selection of application areas to migrate
- ▶ Assessment of migration complexity
- ▶ Ballpark migration estimates delivered in hours
- ▶ Sample database and code migrations
- ▶ Migration tool selection and demonstrations
- ▶ Problem resolution related to the migration
- ▶ Selection of migration services
- ▶ Database administration
- ▶ SQL application development comparisons

The IBM Data Management Services Migration Center consists of a team of migration specialists that can perform the entire migration, or can partner with your team for a more cost-effective solution. Alternatively, the Migration Center can simply provide direction and advice on an as-needed basis at an hourly rate. The IBM Data Management Services Migration Center team can be contacted through your IBM Sales Representative.

For more information about the IBM Data Management Services Center, visit the following Web page:

<http://www-01.ibm.com/software/solutions/softwaremigration/dbmigteam.html>

The IBM migration methodology consists of the following primary phases:

- ▶ Preparation
- ▶ Migration
- ▶ Test
- ▶ Implementation and cutover

Typically a migration project is an iterative process that consists of multiple rounds of migration, testing, and refinement. Each phase has specific objectives and deliverables, and is described in the following sections of this chapter.

3.2 Migration preparation

The migration preparation phase includes all the activities that take place before setting up the system and migrating the database, database objects, and data. In this phase, it is essential to focus on system architecture analysis and information gathering to make decisions about how the migration will be performed. The primary objectives of this phase are to develop an overall migration strategy, perform initial project planning, and assess the benefits and risks of various migration options.

The major tasks involved in preparation are as follows:

- ▶ Migration assessment
- ▶ Understanding and selecting migration tools
- ▶ Estimating the effort required
- ▶ Environment preparation
- ▶ Getting educated on Informix Dynamic Server technology.

3.2.1 Performing the migration assessment

Planning a migration begins with an assessment of the current system and environment, as well as an understanding of the resources that can be used.

An accurate profile of the system-wide architecture is key to the success of the project. The assessment begins with collecting information about both the source and target environments. This detailed information is used to determine the scope of the migration, and is the main input for the project planning process.

Before undertaking a migration project, there are several planning activities that should be performed. The following list summarizes those areas and the type of information that you need to gather and consider:

- ▶ Perform a hardware and software architectural assessment
 - Decide on the target hardware platform for the production system.
 - Understand the workload characteristics and requirements.
 - Know the number of concurrent users.
 - Take an inventory of software and hardware upgrades.
 - Determine the machine on which the migration will be performed.

- ▶ Investigate the scope, duration, and cost of the project
 - Which application areas will be migrated?
 - How complex is the migration?
 - How many database objects and applications will be migrated?
 - What are the differences between the source and target databases?
 - How long will the migration take?
 - What is an estimate of the proposal for work.
- ▶ Identify business dependencies
 - Are there timeline or business cycle requirements?
 - Do renewal of licensing agreements impact schedules?
- ▶ Identify the people resources and skills required
 - Will resources be in-house, outsourced, or a combination of both?
 - Do in-house resources have skills for the source and target databases?
 - Are in-house resources available to perform the migration?
 - Are in-house resources available to support an outsourced migration?
- ▶ Identify the services and tools that can be used during the migration.

The following list details objectives:

- ▶ Devise a strategy for key issues
 - Coexistence of source and target databases
 - Ongoing application development
 - Change management
 - Performance requirements
 - Tool selection
 - Naming conventions
 - Database physical design
 - Define standards
 - Data migration strategy
 - Security planning
 - Cutover strategy

3.2.2 Understanding and selecting migration tools

Although a migration can be performed without the help of tools, IBM has created a tool that is specifically designed to make a migration as simple as possible. The tool is introduced here and covered in detail in later chapters.

IBM Migration Toolkit

The IBM Migration Toolkit (MTK) helps in the migration from Oracle database to Informix. This tool can be used to generate DDL scripts that create database objects including tables, indexes, views, triggers, stored procedures, and user-defined functions. It also aids in moving the data from the original source system to the target Informix database. For example, the MTK can either connect

directly to the source system and extract the database structure and database object definitions, or it can use a valid database schema script extracted by other tools.

The IBM Migration Toolkit is a free migration tool that can be downloaded from the following Web page:

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

3.2.3 Estimating the effort required

An accurate estimate of the effort, resources needed, and total costs requires knowledge of the products, applications, and migrating experience.

The MTK can be used as an assessment tool to determine the complexity of the migration. Using the MTK in this manner can highlight complex stored procedures or SQL statements that may require manual migration effort. A working knowledge of the MTK is mandatory when using it in this manner. Training costs and time required for DBAs and users should also be factored into the estimates.

Each migration is different, but there are general signs that can indicate the overall complexity and effort expected. For instance, in applications that frequently use stored procedures, the number and complexity of the stored procedures to be migrated greatly affects the migration time. The same applies to the use of special data types and large objects. Physical requirements (use of raw or formatted disk areas, spaces, and nodes) may also represent a large amount of work, especially if the data grows significantly over time. If there is replication or high availability involved, it needs a thorough study to understand the existing database environment, current topology, and storage requirements. This is to properly design the database and implement the similar technology with the target database system.

The IBM migration team migration estimate

A type of assistance that is frequently requested from the IBM migration is an estimate for the migration of database objects and applications. The team uses prior experiences to deliver an estimate for a minimum and maximum range for the number of hours the project is expected to take. To deliver that estimate, the team provides a questionnaire and the client collects and returns metrical information for the objects to be migrated.

The following list details the types of metrics collected:

- ▶ Number of database objects including tables, indexes, and views.
- ▶ Number of database application objects (including packages, procedures, triggers and user defined functions), average lines of code, and SQL statements for each.
- ▶ Number and language of application programs including lines of code and average number of SQL statements per module.
- ▶ Volume of production data to be migrated.

In addition to metrical information and project tasks, there are additional factors that influence the size and complexity of a migration. Some of these factors are as follows:

- ▶ Amount and type of proprietary SQL used
- ▶ Quality of data
- ▶ Existence of system documentation
- ▶ Database design requirements such as high availability and replication
- ▶ Third party software dependencies
- ▶ Operating system and hardware platform change
- ▶ Availability of a dedicated machine for migration development
- ▶ Extent of code freeze enforcement during migration
- ▶ Length of the cutover window
- ▶ Skill level of resources performing the migration

3.2.4 Environment preparation

Before starting the migration, the target development environment should be set up. Ensure that all hardware and software prerequisites are met, that the network is properly configured, and that there is enough hardware to properly support the migration and development.

In this step, you should complete the following tasks:

- ▶ Configure operating system
- ▶ Configure disk storage
- ▶ Install Informix Dynamic Server
- ▶ Create an Informix instance
- ▶ Install and configure tools
- ▶ Configure the Informix Server
- ▶ Configure the database environment and registry variables
- ▶ Test the environment with a sample application

3.2.5 Getting educated on the Informix Dynamic Server

It is important that members of the migration team have a good understanding of important Informix features, such as transaction management, locking, authorities, data recovery techniques and memory management. Although most database vendors implement similar sets of database features, there are often substantial differences in how they behave.

The fastest way to prepare your technical staff for effectively working with the Informix Dynamic Server is through some form of education. The following Web page provides numerous references to sites within IBM that offer classroom training, online tutorials, and reference materials:

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

3.3 Migration

This phase is the core of the migration project. The steps required to migrate a database and application to Informix are introduced in this section. The methods employed can vary, but they can be divided into three primary sub-phases:

- ▶ Database migration and design
- ▶ Calibration
- ▶ Application migration

3.3.1 Database migration and design

The first step in the migration process involves moving or duplicating the structure of the source database into an Informix database. In this process, the differences between the source and destination structures must be addressed. These differences can result from different interpretation of SQL standards, or the addition and omission of particular functions. Many differences can be fixed by altering the existing syntax. But in some cases, custom functions must be added and the application must be modified.

In this phase the final Informix physical database design for the production system is planned and designed. The physical layout of the Informix database is critical to the success of the project. It must be implemented according to best practices to ensure optimal performance. It must be structured to support future maintenance and to provide maximum flexibility to meet future requirements, without compromising existing business processes and program logic.

The most popular means of migrating a database structure is through the use of a migration tool. These tools not only connect to and extract structural information from the source database, but they can also convert it into a format acceptable by the target database system. The IBM MTK can be used to perform the migration using this method.

Capturing the definition of database objects can often occur at the same time the database structure is captured, if the objects are written in an SQL-like procedural language and stored within the database. A good portion of the database object migration is automated. The MTK, an IBM product that is free to download, is the tool of choice for converting database objects such as tables, data types, constraints, indexes, views, stored procedures, user-defined functions, built-in functions, and sequences. However, the tool does not migrate database security features such as user IDs, authorizations, or privileges (GRANTS and REVOKEs). In addition, the creation of tablespaces and the placement of tables and indexes within table spaces along with disk layout is a manual part of the physical design process. While the MTK is the tool of choice to convert procedures, and user-defined functions to Informix, the migration of these application objects usually requires a certain amount of manual intervention depending on the SQL constructs used. The MTK reports those statements that cannot be converted automatically.

The migration of database objects requires testing of the resulting objects. This implies that test data should be available before testing. Preliminary data migration effort is therefore required to complete this migration step. After the object migration is completed, some adjustments may still be required. As an example, issues such as identifier length may need to be addressed. During this phase, a test database is set up and used for migration development and functional testing of converted objects.

3.3.2 Calibration

The calibration sub-phase is designed to validate the planned strategy for the migration including quality assurance review, standards compliance, and performance. During the calibration sub-phase, a few selected programs are migrated as samples to ensure that the code produced complies with requirements and possesses characteristics that ensure the future maintainability of the applications. Typically, 20 to 60 programs of a selected group are migrated.

There are several considerations in defining the system to be used in the calibration. The primary requirement, however, is that the chosen programs should be representative of how the entire application portfolio uses the source database and application languages. They are also expected to provide baseline performance and data usage that can be easily generalized. The generalized

results will then be used to predict the ability to migrate each of the other application subsystems, as well as their likely post-migration performance characteristics.

During this phase there is a review of the data transfer requirements and design of a data transfer strategy. The data transfer strategy must ensure that all the production data can be transferred into the production system within the cutover time frame.

In addition, detailed resource and schedule planning is finalized, as well as the test strategy for user acceptance and parallel testing.

3.3.3 Application migration

Insights gained during the calibration phase are incorporated into the migration strategy. In this phase, migration of the entire portfolio of objects and applications is completed.

Although converting the database structure and objects can be automated to some extent using migration tools, application code changes mostly require manual effort. Aside from the SQL language differences, you also need to understand the transaction, locking, and isolation level differences between Oracle Server and IDS. These differences influence how the application should be coded, and may require some adjustments to be made for it to run as expected.

Applications written in a high-level language with embedded SQL can be converted automatically by extracting SQL statements from the source code, adjusting them with a migration tools such as MTK, and reinserting them into the source.

Because this phase is mostly executed using manual techniques, expertise with both the source server and ID is essential. Not all SQL may need to be converted because the same SQL syntax often runs on both the source and IDS databases. However, all SQL must be examined to determine whether or not migration is needed. Besides syntax, the semantics of the SQL needs to be examined to ensure that the SQL statement behaves the same way and returns the same results on IDS as on the source database. The complete details of application migration are discussed in detail in the later chapters.

3.4 The Test Phase

In this section we discuss the test phase, which consists of three sub-phases:

- ▶ Migration refresh
- ▶ Data migration
- ▶ Testing

3.4.1 Migration refresh

Because most clients are unable to freeze their application for the duration of a migration, and because the calibration programs are typically migrated early in the process, a migration refresh is required just prior to client testing and production implementation. Based on change control logs, those programs, applications, and data structures that changed during the migration phase are reconverted. This process relies heavily upon the inventory and analysis activities that took place during the assessment phase as that effort cross-referenced programs, procedures, functions, data structures, and application code. Using the baseline as a guide, the migration team reconverts those migrated objects that were impacted by ongoing development.

3.4.2 Data migration

Data migration is usually performed quite late in the migration process. However, subsets of the data should be migrated with the database structure, to verify that everything is migrated correctly.

The process of data migration consists of four activities: unload, transform, cleanse, and reload. This sub-phase is accomplished by a combination of automated and manual methods.

In this phase, the MTK can be used to automate the migration of the data or to generate scripts that can be used to migrate the data manually. The MTK can be used to unload the data from the source database and load the data into IDS. However, and most importantly, the MTK automates the transformation of the source data into a format that IDS will accept during load without errors. Tools other than the MTK may be used to unload the data. The format of the data should be examined to insure that it is in a format that can be loaded into IDS. If it is not, the data will have to be manipulated by an intermediate process into a format that can be loaded.

In many cases, as the data is moved, it is also converted to a format that is compatible with the target database (date and time data are a good example). This process can be quite lengthy, especially when there is a large amount of

data. This makes it imperative to have the data migrations well-defined and tested. In some cases, it is still necessary to perform customized migrations of specialized data, such as time series and geospatial data. This can sometimes be accomplished through the creation of a small program or script.

It is a good idea to perform some tests after migrating subsets of the production data. These tests include performing simple comparisons against the source and target databases, such as row counts. You can also identify if any transformations or adjustments are required and implement them before migrating the full set of data. Once you are completely satisfied with the results of your tests, you can migrate the full set of production data. After moving all the production data from the source to target system, correct any referential integrity issues, then perform a thorough verification of the migrated data against the source data for accuracy and completeness.

Note: If the data is encrypted, it must be manually decrypted first using the source database APIs, transferred to target database, and then encrypted.

3.4.3 Testing

The purpose of the testing is to determine the differences between the expected results (the source environment) and the observed results (the migrated environment). The detected changes should be synchronized with the development stages of the project. In this section, we describe the test objectives and a generic testing methodology that can be employed to test migrated applications.

Testing methodology

The first step in the testing process is to prepare a thorough test plan. The test planning and documenting are important parts of this testing process which details the activities, dependencies, and effort required to conduct the test of the migrated solution. A detailed test plan should describe all test phases, scope of the tests, validation criteria, and specify the time frame. It should list what features of the migration are to be tested and what is not to be tested from both the user viewpoint of what the system does and a configuration management view. Plans should consider the environmental needs (hardware, software, and any tools) for testing, test deliverables, pass/fail criteria for tests, and any required skills to implement the testing.

For large projects, it might be necessary to use supportive software to improve testing productivity. As examples, the IBM Rational Functional Tester and IBM Rational Performance Tester can be used for that purpose. The IBM Rational Functional Tester provides testers with automated capabilities for data-driven

testing and a choice of scripting language and an industrial-strength editor for test authoring and customization. IBM Rational Performance Tester can create, execute, and analyze tests to validate the reliability of complex e-business applications. Additionally, there are many other Rational products that may suit your testing needs. For more information about testing products, visit the following Web page:

<http://www.ibm.com/software/rational>

All stages of the migration process should be validated by running a series of carefully designed tests. The different phases and techniques to consider for testing will be discussed in the remainder of this section.

Functional testing

Functional testing involves a set of tests in which new and existing functionality of the system are tested after migration. Functional testing includes all components of the RDBMS system (such as stored procedures, triggers, user-defined functions), networking, and application components. The objective of functional testing is to verify that each component of the system functions as it did before migrating, and to verify that new functions are working properly.

Integration testing

Integration testing examines the interaction of each component of the system. All modules of the system and any additional applications (such as WEB, supportive modules, and Java programs) running against the target database instance should be verified to ensure that there are no problems with the new environment. The tests should also include GUI and text-based interfaces with local and remote connections.

Performance testing

Performance testing of a target database compares the performance of various SQL statements in the target database with the statement performance in the source database. Before migrating, you should understand the performance profile of the application under the source database. Specifically, you should understand the calls the application makes to the database engine. If performance issues are found, it may be necessary to revisit the physical database design and implement some changes. For detailed information about performance measurement and tuning methods that are relevant to daily database server administration and query execution, see *IBM Informix Dynamic Server Performance Guide*, G229-6385.

Volume/Load stress testing

Volume and load stress testing tests the entire migrated database under high volume and loads. The objective of volume and load testing is to emulate how the migrated system might behave in a production environment. These tests should determine whether any database or application tuning is necessary.

Acceptance testing

Acceptance tests are typically carried out by the users of the migrated system. Users are simply asked to explore the system, test usability and system features, and give direct feedback. After acceptance, tests are usually the last step before going into production with the new system.

Post-migration tests

Because a migrated database can be a completely new environment for the IT staff, the test plan should also encompass examination of new administration procedures, such as database backup/restore, daily maintenance operation, and software updates.

Data migration testing considerations

The extracting and loading process entails migration between source and target data types. The migrated database should be verified to ensure that all data is accessible, and was imported without any failure or modification that could cause applications to function improperly.

Data checking technique

Data movement is the first thing any migration should focus on. Without having all your tables and data properly moved, all other migration testing is in vain. The test process should detect if all rows were imported into the target database, verify that data type migrations were successful, and check random data byte-by-byte. The data checking process should be automated by appropriate scripts. When testing data migration you should perform the following tasks:

- ▶ Check IMPORT/LOAD messages for errors and warnings.
- ▶ Count the number of rows in source and target databases and compare them.
- ▶ Prepare scripts that perform data checks
- ▶ Involve data administration staff familiar with the application and its data to perform random checks.

Code and application testing considerations

The most important part of the testing process is to verify that each component of the system functions as it did before migrating. All components of the RDBMS system, including views, stored procedures, triggers, application components, and security systems should be verified as to whether they are working properly.

Views

The next step after data migration testing is to check all migrated views. Once data is moved over, the views can be tested to make sure they are working in the same way as in the source database. Depending upon the view, different checking scenarios can be created. Basic views can be easily checked by counting the number of rows the views return or by comparing calculated values, and similarly the tests performed on regular tables. More complicated views need a little more thought on how they should be checked. However, all views should be reviewed by executing queries against them. The views created with the intention to modify data should also be tested for inserting, updating, and deleting.

Procedures and user defined functions:

All stored procedures, user-defined functions, and triggers should be unit-tested individually before they are promoted for further testing. This means that after objects are migrated (either manually, with the MTK, or some combination) they need to be checked to make sure they function properly.

Application code check

The scope of application testing depends on the migrated application. For self-built applications, the testing process should be started with the application queries. All queries should be independently tested to ensure that they return the expected results. With the application queries successfully migrated, the surrounding client programs should be rebuilt, and the application should be tested against the target database. Each module of the application, and possibly each panel form, should be run and checked for errors or improper functionality. All the supported application connectivity interfaces should also be checked.

Security considerations

Before going into production, security must be checked in detail. Each database system handles security quite differently, so it is not trivial to compare the user rights between the two systems. The existence and authorities of all required users should be verified to allow the appropriate persons to connect to the database. The access privileges and restrictions on the database objects need to be tested to make sure only users with proper authority can access them.

3.5 Implementation and cutover phase

After the testing phase has completed, final testing is performed to validate implementation of the new production environment and to gain user acceptance. Upon acceptance of the system, an additional test run of the cutover procedure is performed, usually over a weekend prior to the final cutover weekend. Any issues

discovered during the test run are corrected. On the cutover weekend, the database is refreshed with production data and the system goes live. In preparation for this phase, a backup of the source production system should be taken in case there is an unexpected need to back out the new system. In addition, all database maintenance procedures such as backups are put into production.

In some cases, the final cutover phase does not allow for downtime of the production system. In these cases, special planning and procedures are developed to accommodate the phasing in of the new system without impacting the production environment.

3.6 Related information resources

IBM Database Migration:

<http://www-01.ibm.com/software/solutions/softwaremigration/dbmigteam.html>

IBM Migration Toolkit (MTK):

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

IBM Informix Dynamic Server Information Center:

<http://publib.boulder.ibm.com/infocenter/idshep/v115/index.jsp>

Oracle to IDS porting Guide:

<http://www.ibm.com/developerworks/data/library/long/dm-0608marino/#download>

IBM Redbooks:

<http://ibm.com/redbooks>

Archived



IBM Migration Tool Kit: An introduction

In this chapter, we introduce you to the IBM Migration Toolkit. When referring to it throughout this book, we use the abbreviation MTK.

Database migrations vary in size and complexity due to the many ways that a database can be designed and used. The result of this is that each migration usually has a set of challenges that are specific to that project. Nonetheless, in addition to the unique set of issues that are presented in each migration there are also underlying elements that are universal to all migrations no matter what the source or target databases may be. It is this principle that is the driving force behind tool development for database migrations. Indeed, the MTK was designed by IBM to do just that. Although the MTK cannot automatically resolve all your conversion and migration issues, it can simplify many tasks and shorten the migration phase of your project.

4.1 The MTK for Oracle migrations to IDS

The MTK was developed as a joint venture between the IBM Silicon Valley Lab in The product was developed to support migrations from a number of source databases to DB2 and Informix Dynamic Server (regardless of platform). In this book we only cover the toolkit for migrations from Oracle 10g to Informix Dynamic Server 11. As a matter of reference, the chart in Figure 4-1 outlines all the source and target combinations that are supported.

	Source DBMS					
	Oracle versions 8i, 9i and 10g	Sybase Adaptive Server Enterprise (ASE) versions 11, 12, 12.5 and 15	Sybase SQL Anywhere (ASA) version 9	Microsoft SQL Server versions 7, 2000 and 2005	MySQL versions 4 and 5	
Target DBMS	DB2® Database for Linux®, UNIX®, and Windows® - Versions 8.1, 8.2, 9 and 9.5	✓	✓	✓	✓	✓
	DB2 for i5/OS® V5R2, V5R3 and V5R4	✓	✓		✓	✓
	DB2 for z/OS® Version 8	✓				
	DB2 for z/OS® Version 9	✓	✓			
	Informix® Dynamic Server Version 10	✓		✓		✓
	Informix® Dynamic Server Version 11.10	✓		✓	✓	✓

Figure 4-1 Source-Target combinations

The MTK version used in this book is 2.0.5.0. This version can be downloaded for free from the following IBM Web site:

<http://www-01.ibm.com/software/data/db2/migration/mtk/>

In this chapter, we present several topics related to using the MTK for an Oracle to Informix migration:

- ▶ Overview of features and functionality
- ▶ Inside the Oracle converter component
- ▶ How to install, configure, and execute the MTK

4.2 Overview of features and functionality

The MTK is designed to manage your migration through projects. In this chapter, a project is actually a directory where all the information associated with your migration is stored. A project directory is created under the MTK installation directory for every project that you create. Each project directory contains imported and extracted source SQL files, converted SQL files, data files (in the DataOutScripts subdirectory), and deployment scripts. Although the MTK provides a GUI interface to automate deployment of converted objects, manual deployment is also possible by running the SQL scripts found in the project directory.

4.2.1 The five step migration process

The MTK provides a graphical user interface that you can use to perform a five step migration process. The GUI interface contains five pages, as depicted by the tabs in Figure 4-2, each representing a step in the migration process. These five steps are as follows:

1. Specify the source.
2. Convert the source.
3. Refine the migration.
4. Generate the data transfer scripts.
5. Deploy to the target IBM database.

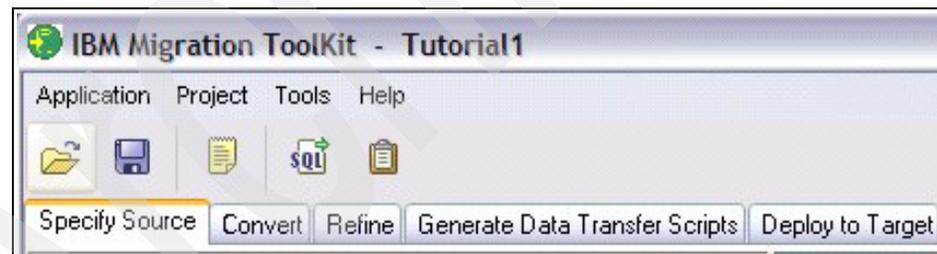


Figure 4-2 The MTK-GUI five step migration process

You start the migration process by creating a migration project and using the Specify Source page to obtain the Oracle files to be converted to the IDS server. Using an ODBC or Java connection, you can either extract the source directly from Oracle, or you can import source scripts from previous extractions.

The main component of MTK is the converter, which translates Oracle SQL to SQL that runs on IDS. After you specify the source information, use the MTK Convert page to specify options for the migration. On the Convert page you can customize some aspects of the migration such as overriding default data type

mappings. The converter supports translations of many constructs used in tables, indexes, constraints, views, sequences, procedures, user-defined functions, triggers, and standalone DML statements. The MTK also extracts data from Oracle and reformats data for insertion into IDS. It should be noted that Oracle tablespaces are not converted to IDS dbspaces and are only minimally supported in the translation process. In addition, SQL statements contained in application languages such as Java, and C are not translated by the toolkit. Only SQL scripts can be input into the MTK. However, the MTK includes an SQL translator, which can be used to test particular phrases of SQL code contained within application languages.

The Refine page in MTK provides various resources to help you modify the code, after which you can reconvert the code to produce the desired results. You can repeat this process until you are satisfied that the new script is ready to be deployed. The resources on the Refine page include views to help you cross reference between the source and target database objects, the script files, messages, and the online documentation.

Use the Generate Data Transfer Scripts page to set data transfer options, generate the deployment and data-transfer scripts, and export and load Oracle data to the target IDS database. If inserting LOB (large object) data, the data must reside locally to the target IDS database. Because the MTK extracts data into a file system where it is installed, it is desirable to install MTK on the same machine as the target database.

During the deployment step, you create the IDS database, run the SQL scripts to create the IDS objects, and move data from Oracle into IDS. The MTK creates various logs and reports of activity to help you verify successful deployment.

After deploying the database, you might need to rewrite some SQL code and fine tune the database as you test your applications. It should be obvious that the team performing the migration should have programming and administration skills for both Oracle and IDS databases. Different behaviors between the two databases will most probably require some redesign and reprogramming to fully take advantage of new IDS functionality.

ISV migrations

We do not recommend using the MTK to convert independent software vendor (ISV) schema definitions. Most ISV packages include their own migration tools for platform migrations, and most vendors require that their tools are used in this process.

A review of the five step GUI process is depicted in Figure 4-3 on page 63.

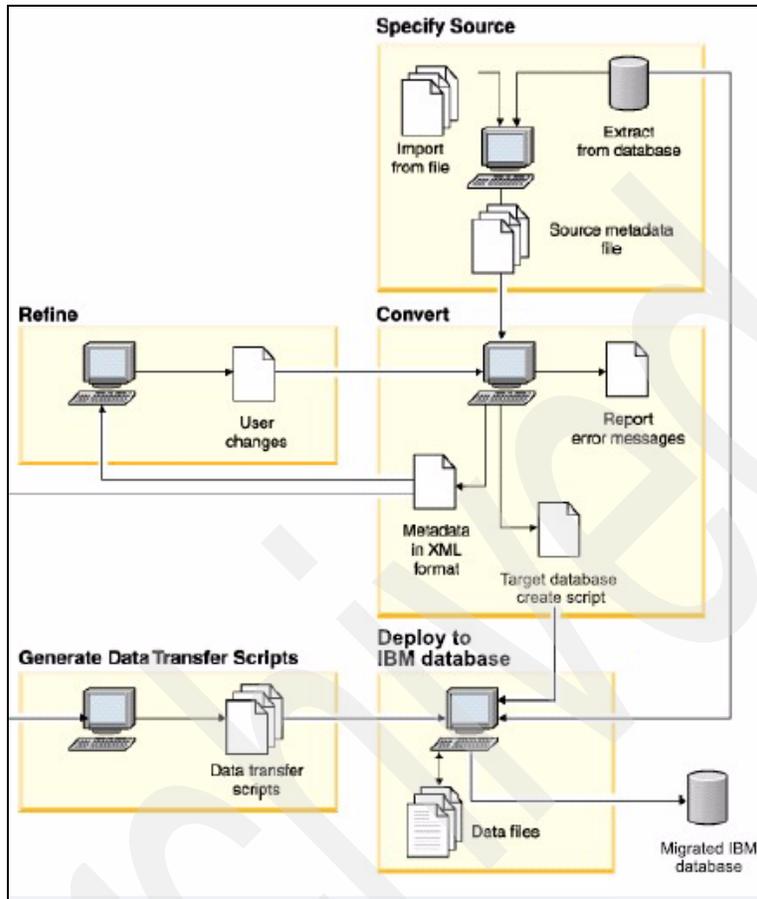


Figure 4-3 Five step GUI process

Environment considerations

You can have different systems in your migration environment. The Oracle database, IDS database, and MTK do not have to be on the same system. A practical scenario is to install the MTK and the target database on the same system. One reason for this is that the data migration step requires that LOB data reside locally to the target database. As depicted in Figure 4-4 on page 64, the MTK and the target database are on the same system, and the source database is installed on another system that is accessible by using either ODBC or a Java native driver.

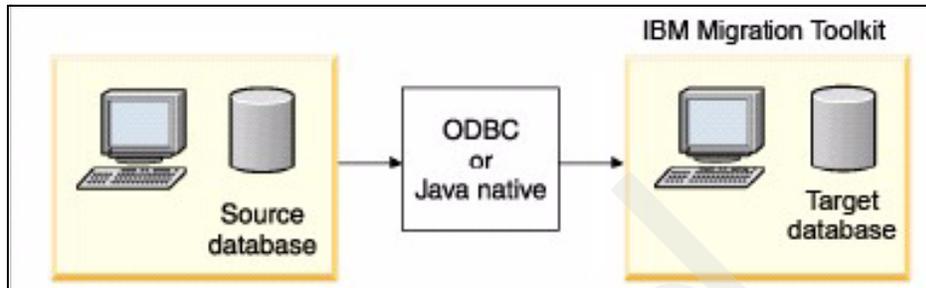


Figure 4-4 Drivers

You can deploy the database objects to a system that is not supported by MTK. During the deployment phase, the MTK runs a set of batch files and scripts that contain the necessary components for deployment. You can copy the deployment scripts from your MTK project directory, modify them as necessary, and run them on the IDS system. The data, however, can only be deployed through the MTK interface for an IDS target.

User interfaces

The MTK is available from three types of user interfaces:

- ▶ Graphical user interface (GUI)

The GUI interface offers the MTK migration functionality by using a Java interface. It is customizable and easy to use, especially for those unfamiliar with the command line interface or database migrations in general.

- ▶ Wizard

The wizard interface offers the basic migration functionality in a guided sequence of steps. This wizard is intended for simple database migrations where manual changes are not required.

- ▶ Command line

The command line interface offers a way to operate the MTK using commands, a configuration file, and arguments. The command line interface is intended for experienced users who want to run the migration without user interaction. You can run the entire five step migration process, from opening a project to deployment, or just a single step in the migration process, such as convert, by issuing just one command. Before you issue the command, you must provide all the necessary information in the configuration file. A sample configuration file, named config.xml, is included in the MTK installation directory (for example, c:\mtk\config.xml). The configuration file contains all of the migration details. It is an XML file whose structure is defined in the mtk.dtd file included in the MTK installation directory (for example, c:\mtk\mtk.dtd).

The basic structure of the configuration file is depicted in Example 4-1 on page 65.

Example 4-1 Configuration file structure

```
<MTK>
  <PROJECT...>
    <SPECIFY_SOURCE></SPECIFY_SOURCE>
    <CONVERSIONS></CONVERSIONS>
  </PROJECT...>
</MTK>
```

The order of the arguments is irrelevant. An example of how to issue the command line interface is shown in the following examples:

From a Windows command prompt, issue the following command:

```
MTKmain.bat -CONFIG configfile.xml argument
```

From a UNIX/Linux command prompt, issue the following command:

```
MTKmain.sh -CONFIG configfile.xml argument
```

The config.xml file contains all the arguments that are needed and directs the migration. Each of the configuration file arguments correspond to one of the five steps described used in the GUI process. There is an additional argument, ALL, that can be used to run the migration from end to end. The following list details the supported command line arguments:

- ▶ CONFIG
- ▶ IMPORT
- ▶ EXTRACT
- ▶ CONVERT
- ▶ GENSCRIPT
- ▶ DEPLOY
- ▶ ALL

The following example shows you how to import source SQL files by using the MTK command line interface. To import file1.src and file2.src into the project, issue the following command with the configuration file shown:

```
MTKMain.bat -CONFIG config.xml -IMPORT
```

The contents of config.xml are as depicted in Example 4-2 on page 66.

Example 4-2 Configuration file

```
<MTK>
  <PROJECT ...
    <SPECIFY_SOURCE>
      <IMPORT>c:\file1.src</IMPORT>
      <IMPORT>c:\file2.src</IMPORT>
    </SPECIFY_SOURCE>
    ... >
  </PROJECT>
</MTK>
```

Besides arguments, there are also elements and attributes that further describe how to perform each task in the process. Example 4-3 depicts the PROJECT elements that specify project name, directory name, the source database type, and the target database type.

Example 4-3 Project elements

```
<MTK>
  <PROJECT NAME=""
    DIRECTORY=""
    DESCRIPTION=""
    SRCDBTYPE=""
    TRGTDBTYPE=""
    ...
  </PROJECT>
</MTK>
```

When using the command line interface, refer to the MTK User's Guide for values and descriptions of the arguments, elements and attributes supported. The User's Guide can be downloaded from the following Web page:

<http://www-01.ibm.com/software/data/db2/migration/mtk/>

4.3 Inside the Oracle converter component

Now we look at how the Translator works. The converter (also referred to as the Translator) is written in Java and uses Another Tool for Language Recognition (ANTLR) as its parsing engine. The converter is a language translator that operates much like a compiler for a conventional programming language. It takes as input a sequence of Oracle SQL scripts and generates a corresponding IDS SQL script as output. It also generates metadata information about each Oracle object definition and corresponding generated IDS object definition.

The metadata is encoded in the XML-based Metadata Interchange™ (XMI) format for easy reuse. The metadata information summarizes important properties about source objects and is used by the MTK to generate an overview of source and target objects in the Refine step (where the results of the translation are reported).

An Oracle SQL script is a sequence of SQL statements and PLSQL commands. The SQL statements are translated as they are encountered in the script. Therefore, the order in which Oracle objects are defined in the source script is critical for proper migration. The converter requires that an object be defined before it is used. Queries of an object cannot be translated if the object has not been defined.

When the source of the object definitions are extracted (using the MTK) directly from the Data Dictionary, the object definitions are in dependency order. When the source of the object definitions are imported into the MTK from an external file, some manual reordering to satisfy dependencies might be required. For each IDS SQL statement generated in a converted script or stored procedure, the converter will normally copy the corresponding Oracle statement as a comment preceding the generated IDS statement. This is depicted in Example 4-4.

Example 4-4 Converter output

```
--| CREATE INDEX IND_ACCT_ID
--|   ON ACCOUNTS ( "ACCT_ID" ) TABLESPACE USER_IND_TBS
--| ;

--*
[600059]"C:\MTK\projects\Tutorial1\tables.src"(140:28)-(140:50)Ignored
input - not translated.

CREATE INDEX IND_ACCT_ID
      ON ACCOUNTS(ACCT_ID);
```

In Example 4-4, the Converter's message Ignored Input - not translated is referring to the TABLESPACE clause in the Oracle CREATE INDEX statement, which was not translated in the IDS CREATE INDEX statement. For more information about actually performing these translations, see "Mapping Oracle tablespaces to IDS dbspaces" on page 91.

The commenting annotation makes it easier to understand how the generated code relates to the source and how to perform manual refinement of the generated code if necessary. If an error occurs during the migration, the error message will appear after the source code, and any invalid IDS statement that results from the error will be commented out. If you prefer not to see the

commented source in the output file, you can disable it from the Converter page by using the **Advanced Options** button.

4.3.1 Translating tables, indexes, and views

This phase of the translation is the most straightforward. The MTK converts DDL with little manual intervention required. However, the MTK does not convert tablespaces, so you will need manually to develop a script to create your IDS dbspaces. You can edit your IDS DDL to assign dbspaces to your tables and indexes and add other DDL changes to optimize performance. You should then deploy the DDL manually to IDS. In the beginning phase of your migration, usually the development phase, you can use the MTK to deploy automatically your tables and indexes. Later, when deployment is to production, you should edit your scripts in accordance with the database physical design requirements of your application.

4.3.2 Translating built-in functions

The MTK comes with pre-written IDS user-defined functions that match the functionality of many Oracle built-in functions. There are three possible scenarios when converting Oracle built-in functions to IDS:

- ▶ An Oracle built-in function has an equivalent IDS function. In this case, function calls are mapped directly to IDS.
- ▶ An Oracle built-in function does not have an equivalent IDS function, but a similar IDS function is available.
- ▶ An Oracle built-in function has no IDS equivalent. In most of these cases, an IDS SPL or Java UDF is provided by the MTK to provide similar functionality.

The user-defined functions that are packaged with the MTK are contained in the ORA schema. They can be found in the MTK installation directory in files named `mtkorainfxt.udf` and `orainfxtUDFs.jar`. The MTK automatically installs the Java and SPL UDFs during the Deploy to Target step. The `DEPLOY_yourprojectname_UDF.log` file contains information about the success or failure of the UDF deployment.

Translating functions, procedures, and triggers

The migration of functions, procedures, and triggers will generally require more manual intervention than the migration of tables and indexes. However, the MTK will provide you with a quick start for these conversions and simplify this task. The Oracle Converter section of the User's Guide documents the constructs that the MTK supports during the Convert step.

The Generate Data Transfer Scripts step involves two tasks:

- ▶ Generating scripts that transform Oracle data into IDS format and extract the data to a file. The MTK formats data that is compatible for insertion into IDS by building SELECT statements that use built-in functions. Example 4-5 was taken from a data extract script generated by MTK.

Example 4-5 Selects using built-in functions

```
SELECT REPLACE("CHANGE_TYPE",' ','') "CHANGE_TYPE",  
REPLACE("CHANGED_BY",' ','') "CHANGED_BY",  
TO_CHAR("TIMESTAMP",'YYYY-MM-DD-HH24.MI.SS'.00000'''),  
"OLD_EMPLOYEE_ID", REPLACE("OLD_DEPT_CODE",' ','') "OLD_DEPT_CODE",  
"OLD_ACCT_ID", REPLACE("OLD_BAND",' ','') "OLD_BAND", "NEW_EMPLOYEE_ID",  
REPLACE("NEW_DEPT_CODE",' ','') "NEW_DEPT_CODE", "NEW_ACCT_ID",  
REPLACE("NEW_BAND",' ','') "NEW_BAND" from "ORA_USR"."MANAGER_AUDIT"
```

- ▶ Generating scripts to read the data from a file and insert the data into IDS. Generally, the MTK transformation and extraction phase takes more execution time than inserting the data into IDS. This is due to the data migration process which is performed while data is extracted. The data can be loaded into a local IDS database or to a remote IDS database provided that the MTK client has a connection to the remote IDS server. However, the following considerations apply:
 - The MTK cannot automate the creation of a remote IDS database.
 - There must be sufficient disk space on the MTK machine for the data.
 - LOB data needs to reside on the same machine as the IDS database.

Note: We do not recommend that you extract Oracle data on one system and transport the data files over the network to the IDS system.

4.4 How to install, configure, and execute the MTK

In this section we summarize hardware and software system requirements, installation, and how to start up MTK. In Chapter 5, “An MTK tutorial” on page 73, we lead you through a step-by-step tutorial on how to use the product.

4.4.1 System requirements

To use MTK, you need the required software and hardware.

Hardware requirements

Disk space:

- ▶ 50 MB for installation
- ▶ 5 MB per project
- ▶ Additional space for the project script and data files.

Memory:

1 GB or more (increase for large SQL files).

Software requirements

In this section we describe the software required to use the MTK.

General requirements:

- ▶ Operating systems supported:
 - Windows XP, Windows 2000
 - AIX 5L™ 5.2
 - Linux RHEL 3
 - Solaris™ 2.9/9
 - HP-UX B.11.11
- ▶ Java Runtime Environment Required:
 - You must have Java Runtime Environment 1.4.2 installed and accessible through the PATH environment variable.
 - JDBC or ODBC driver (for source database connections):
- ▶ UNIX and Linux-specific requirements:
 - On Linux, increase the message queue number to at least 128:
`sysctl -w kernel.msgmni=128`
 - To view HTML reports, include the browser directory in the \$PATH variable. If the browser cannot be found, MTK will launch an internal Java Web browser which can display HTML files, but does not handle frames or format the tables well.
 - If you are extracting from a data source by using ODBC or Java, configure the client connection.

4.4.2 Installing MTK

You can install MTK on Windows, UNIX, or Linux. Download the appropriate MTK version for your platform from the following IBM WEB site:

<http://www-01.ibm.com/software/data/db2/migration/mtk/>

Prerequisites:

- ▶ Uninstall earlier releases of MTK before installing the latest release.
- ▶ All versions of MTK are compatible with each other.
- ▶ To simplify deployment, run MTK while logged-in with a system user ID that has system administration authority for the target Informix Dynamic Server database

Installing on Windows:

Perform the following steps to install MTK can be installed on Windows:

1. Download and unzip the MTK file from the MTK download site. The zip file contains the MTKSilentInstall.iss file, the readme.txt file, and the mtk.exe file.
2. Navigate to the directory where you unzipped the files and double-click the mtk.exe file to start the installation and follow the instructions.
3. Verify that you can access Java and that it is the correct version (1.4.2):

```
java -version
```

Installing on UNIX or Linux

Perform the following steps to install on UNIX or Linux:

1. Log in with the user ID you want to install MTK with (do not install MTK as root).
 - For example, on Linux or Solaris if you issue `echo $LD_LIBRARY_PATH`, it should list `$HOME/sql/lib/lib`, where `$HOME` is the home directory.
 - Install MTK by using a user ID in an administrative group.
2. Create a new directory. Download the IBM Migration Toolkit into that directory.
3. Untar and extract the mtk.tar.gz file into the MTK directory that you specify by issuing the following commands:

```
tar -xvf  
mtk.tar
```
4. Verify that you can access Java and that it is the correct version.

Note: When you uninstall and reinstall the MTK product, existing project directories are not removed and can be accessed by the new MTK version.

4.4.3 Starting MTK

In this section we show the steps for starting the MTK on Windows and UNIX.

Starting on Windows

Perform the following steps to start the MTK on Windows:

1. From the Start menu, navigate to Programs → **IBM Migration Toolkit** and choose either Toolkit (preferred) or Wizard.
 - ▶ From a command prompt, navigate to the directory where MTK is installed, type `MTKMain.bat`, and press Enter.

Starting on UNIX or Linux

To start MTK on UNIX or Linux, navigate to the directory where MTK is installed, type `MTKMain.sh`, and press Enter.

An MTK tutorial

We designed this chapter as a tutorial and use coding examples so that you can follow along and walk through the important features of the IBM Migration Toolkit, commonly known as the MTK. When using a tool to perform migrations, it is almost certain that some type of manual intervention will be required because no tool can handle every scenario that comes along. However, knowing how to use a tool to its fullest advantage can give you a good start with the migration task. For this reason, the focus in this chapter is on how to best use MTK rather than on how to convert specific SQL. As we go through the tutorial and come upon items that the MTK does not translate, we point you to other sections of this book that can help you address those particular issues.

We present the tutorial in two parts:

- ▶ Part 1 illustrates how to migrate the core database objects, which includes the following objects:
 - Tables (creates and alters)
 - views
 - Indexes
 - Sequences
 - Constraints
 - Data transfer script creation
 - Data movement
 - Core object deployment to an IDS database

- ▶ Part 2 illustrates how to migrate database application objects and includes the following objects:
 - Procedures
 - User-defined functions
 - Packages
 - Triggers
 - Application object deployment to an IDS database

This order is strictly adhered to because the migration of procedures, functions, and triggers depends on the existence of converted table, view, and sequence definitions in the MTK repository.

Note: DML statements such as SELECT, INSERT, UPDATE, and DELETE can be translated when imported by way of script files. However, like database application objects, the table definitions that the DML depends on must be first translated and exist in MTK.

How to use this tutorial

The tutorial is based on converting a set of source objects that we created on an Oracle 10.2.0g database specifically for use in this Redbooks publication. We refer to these Oracle objects as the *example database*. The example database consists of a set of objects owned by the ora_usr schema. The list of the Oracle source object definitions are included in orclsetup.rar file described in Appendix F, “Additional material” on page 355. Script files containing the Oracle source DDL is included in the orclsetup.rar file as well. The scripts can be used to construct a database identical to the one used in this tutorial. However, it is not a requirement to construct the database to get value from the tutorial. You can choose to follow the MTK process by simply viewing the figures throughout the chapter.

Note: The tutorial was designed to be transparent to the operating system and hardware used. Any portion of the tutorial that is dependent on a platform-specific feature will be noted as such.

Getting started

If you have an Oracle 10g database installed and plan to use it with this tutorial, then you will also require the following products:

- ▶ IBM Migration Toolkit (MTK) Version 2.0.5.0 (or higher) preferably installed on the same machine as the target IDS server
- ▶ IBM Informix Dynamic Server (IDS) Version 11.10 (or higher)

Note: This tutorial uses MTK Version 2.0.5.0 which was the latest version available at the time this IBM Redbooks publication was published. Refer to Chapter 4, “IBM Migration Tool Kit: An introduction” on page 59, for additional MTK product information and for assistance on acquiring, installing and configuring the MTK.

5.1 Part 1: Core database object migration

We begin the migration by converting the core database objects. When you bring up the MTK you will see the following menu options:

- ▶ View a brief product overview
- ▶ Quickly convert a database using the wizard
- ▶ Launch the IBM Migration Toolkit product

To begin, click **Launch the IBM Migration Toolkit**.

5.1.1 Create a project

The first panel that opens is the Project Management panel, as shown in Figure 5-1 on page 76.

To create a project, perform the following steps:

1. Create a new project called Tutorial1.
2. On the lower-left side of the panel, select Oracle as the source database from the **Select source database version you want to migrate** pull-down menu.
3. On the lower-right side of the panel, select Informix Dynamic Server (v11.10 preferred) as the target of this migration on the **Target Platform and Version** pull-down menu.

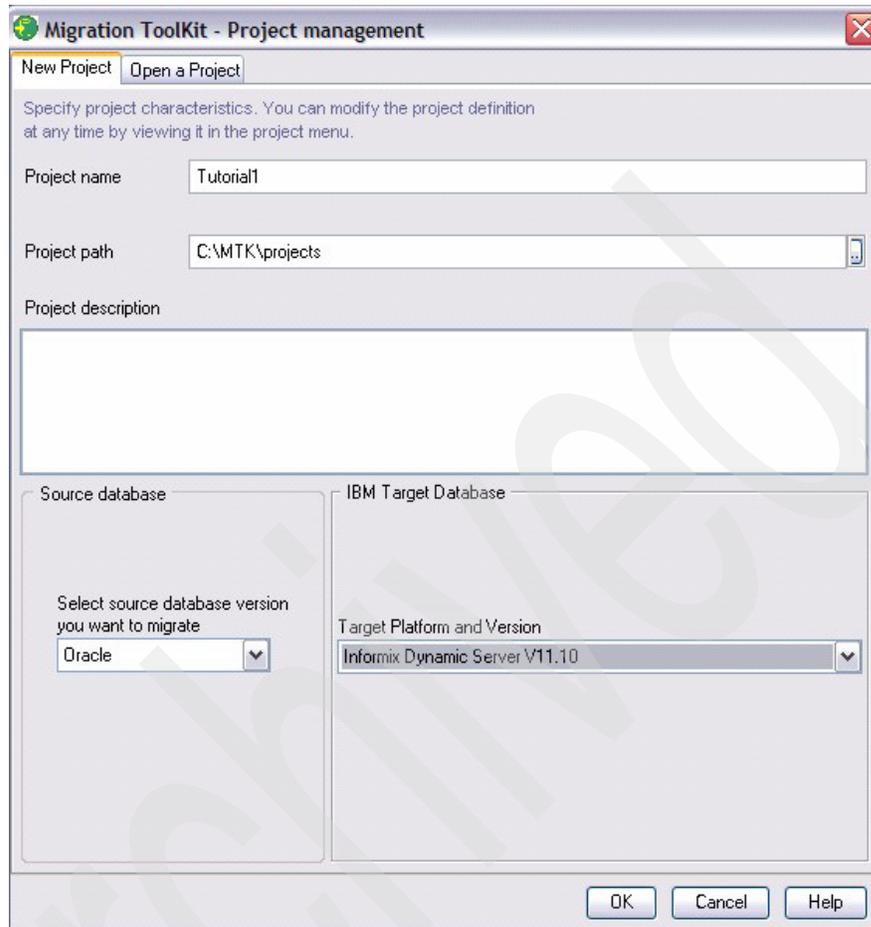


Figure 5-1 MTK Creation Panel

Note: The current version of MTK supports IDS version 11.10 as its highest target, but the MTK can also be used to convert and deploy to newer versions of IDS. However, each new release of IDS may include additional Oracle compatibility features. For example, IDS 11.50 now supports many of the same built-in functions as Oracle. It is for this reason that you should inspect MTK translations to versions of IDS higher than 11.10.

5.1.2 Work with the project

After the project is created, you are presented with the MTK user interface, as shown Figure 5-2. You will notice five tabs at the top of the panel:

- ▶ Specify Source
- ▶ Convert
- ▶ Refine
- ▶ Generate Data Transfer Scripts
- ▶ Deploy to Target

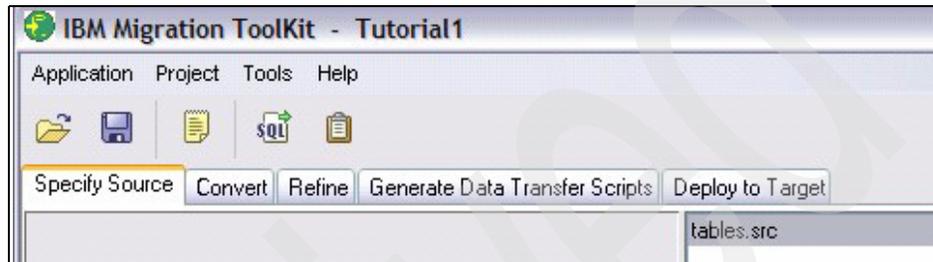


Figure 5-2 MTK interface

Note: If at any time during your implementation you would like to see a description for a button or any other item on a panel, you can place your cursor over that item and a message box with a description of that item will appear.

Each tab represents a phase of the migration process, and we discuss each in turn.

Specify Source tab

The Specify Source tab enables you to specify the data source to be migrated, and is depicted in Figure 5-3 on page 78. You will notice that there are two buttons:

- ▶ Import
- ▶ Extract

The MTK can extract database object definitions directly from the Oracle Data Dictionary tables or import database object definitions contained in scripts. For this tutorial, we extract directly from the database just as you would in an actual migration project.

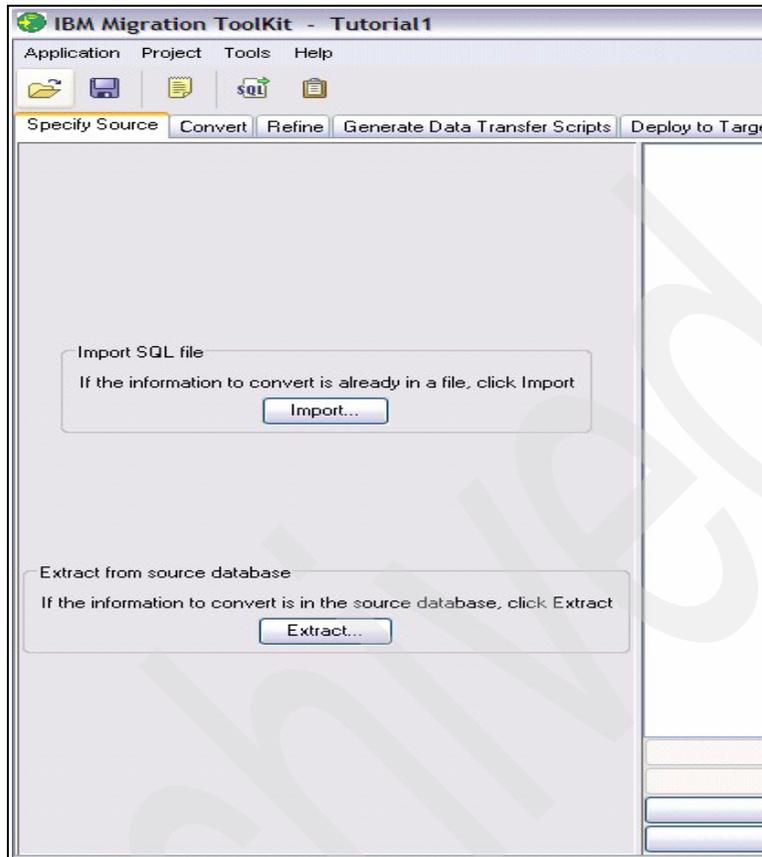


Figure 5-3 Specify Source Tab

Note: If you do not have an Oracle database, then select **Import**, and import the DDL for the core database objects using the scripts provided.

Using the extract capability, it is possible to perform the following tasks:

- ▶ Extract only a subset of objects into its own file. For example, extract only tables into a tables.src file and only functions into a functions.src file. For large conversions, this is a recommended best practice.
- ▶ Automatically resolve dependencies. When extracting a view also determine and extract needed tables, and when extracting a table also extract the primary key, as examples.
- ▶ Extract each procedure and trigger to its own text file.

Tables and indexes are generally the easiest to migrate. As previously mentioned, we do this first because all other objects, such as functions, procedures, and triggers, depend on having the table definitions. Click **Extract** to begin the process of extracting the DDL into the project.

Before you can extract the DDL you must first connect to the Oracle database. The Connect to Database panel opens next, as shown in Figure 5-4 on page 80. For your connection we recommend using the Oracle JDBC driver (ojdbc14.jar) that can be found in a subdirectory in your ORACLE_HOME. Select the **Use native JDBC driver** check box on the panel. Enter the information needed to establish a connection to the Oracle instance and click **OK**. The required information is as follows:

- ▶ Service Name/ODBC DSN Alias
- ▶ Check box for the JDBC driver
- ▶ IP address of the host machine where the Oracle server is installed
- ▶ Port number for the Oracle server
- ▶ User ID
- ▶ Password

Note: Note that on Windows, an ODBC or a JDBC connection can be used.

Figure 5-4 also shows an example of the values to be inserted. Enter the values that are specific to your system and click **OK**.



Figure 5-4 MTK: Connect to the database

In the “Connect to Database” panel, the service name (orcl) in the Oracle `tnsnames.ora` file is used. The full path of the `ojdbc14.jar` JDBC driver must be added to the `CLASSPATH` environment variable before MTK can use the `jdbc` driver. The user ID (`ora_usr`) must have read privileges on the Data Dictionary tables.

Upon a successful connection, a graphical extract tree structure representing all created Oracle schemas (logical databases) in the Oracle database opens, as shown in Figure 5-5 on page 81. Expand the tree for the `ora_usr` schema (our example database) and view all of its contents. You should see two sequences, 10 tables, two views, nine procedures/functions, seven triggers, and two packages.

For this part of the tutorial, we extract only the sequences, tables, and views.

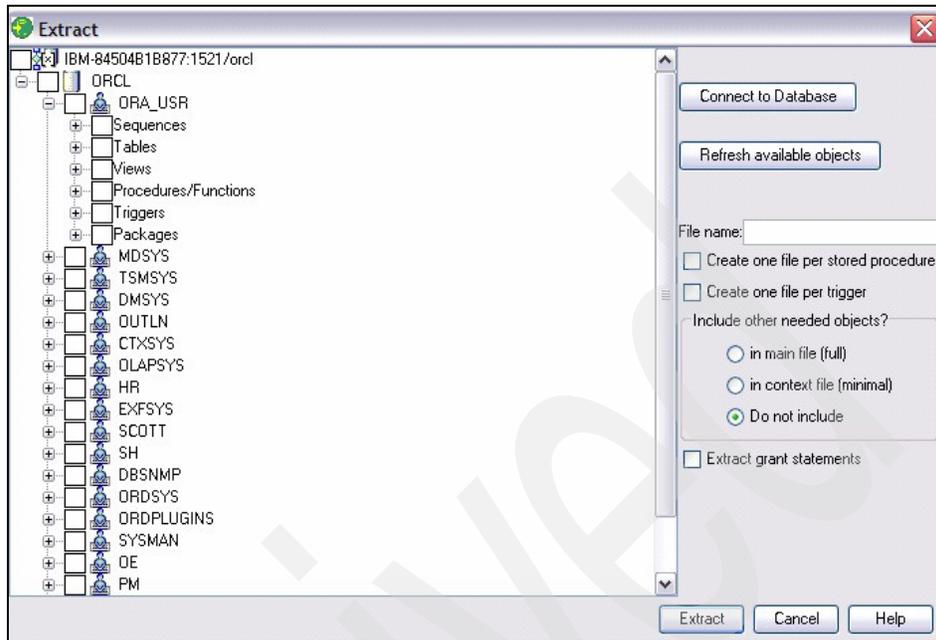


Figure 5-5 Extract tree structure

Take note of the **Extract grant statements** check box. Selecting this option will extract all the grant definition statements. However, the MTK will not convert or deploy them. This feature is helpful when reconstructing grants for IDS, but the deployment of grants will need to be performed manually.

Next, we name this migration task as tables, and enter that name into the File name box, as depicted in Figure 5-6. We check the boxes for sequences, tables, and views and click **Extract**.

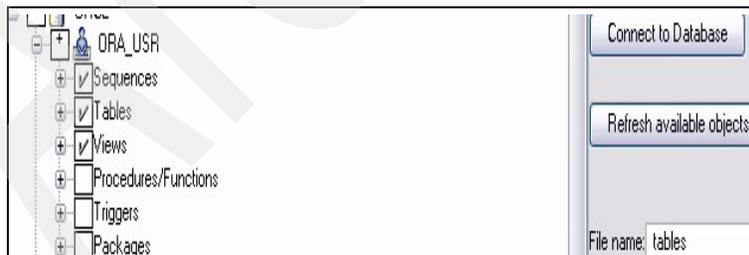


Figure 5-6 MTK: Checking the objects

After examining the script we find that definitions for sequences, tables, indexes, views, and alter statements to create primary and foreign key constraints are included. Note the `CONNECT ORA_USR` statement at the beginning. To continue to the next step, click the **Convert** tab.

Convert tab

We are now ready start the migration. To do this, use the Convert tab to display the “Convert” panel (Figure 5-8). The left side of the panel lists the Oracle source file (denoted as `type.src`). The right side of the panel is where the translated files for IDS (denoted as `type.ids`) will be listed after the migration has been run. In the middle of the panel, you will see the **Convert** button, which initiates a migration of the currently selected source file.

To start the migration, highlight the `tables.src` file on the left. Be careful to select the correct file when more than one source file is listed.

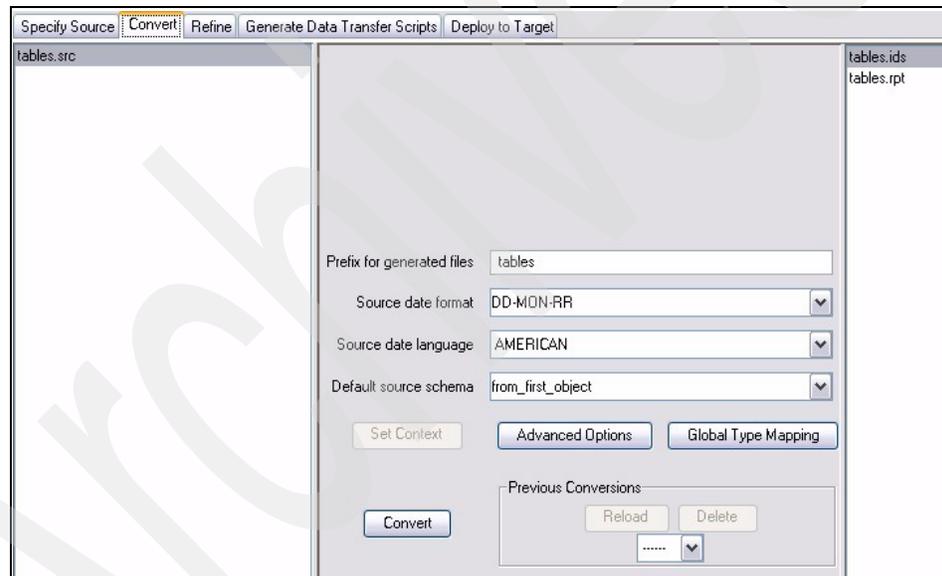


Figure 5-8 MTK: Convert Tab

There are some optional features to note on the “Convert” panel. Recall that help text will appear when the cursor is placed over each feature.

- Specify schema for all objects

If this option is chosen, the schema name specified in the `CONNECT` statement of the source file is added as the schema name to each target object.

- ▶ **Advanced Options**

These are used to control the characteristics of the migration file that will be output. For example, by default, the source Oracle object definitions are written as comments in the translation file. This can be disabled. Also, drop statements can be added before create statements to ease repeated deployment.

- ▶ **Global Type Mapping**

Click this button to view the MTK default data type mappings between Oracle and IDS, and to override those default mappings identified with an edit icon. Refer to Figure 5-9 on page 85 to see the mappings. After the data type mapping has been changed, the effect is global to all conversions in a project.

- ▶ **Previous Conversions**

Click **Reload** when you want to make a previous conversion the active conversion again. Once a file has been converted, the output of that conversion may not be immediately available for viewing. Clicking **Reload** will bring back these files for viewing without performing the conversion again.

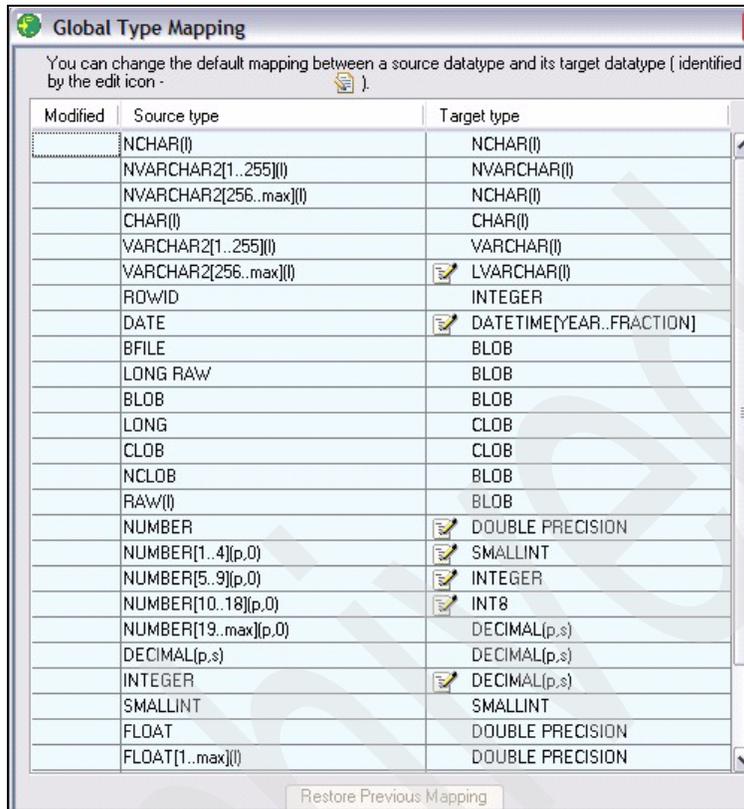


Figure 5-9 Global type mapping

Click **Convert**. When conversion completes, the MTK automatically takes you to the Refine tab. At the refine stage, you have the opportunity to review errors, warnings, and other information associated with the translation.

Refine tab

The Refine tab has two main panels. On the left panel, you see the messages tree. Expanding each node on the tree enables you to view details for each message. On the right panel, additional information about the currently selected tree node is displayed.

When you select the root tree node, you see a summary of the types of errors encountered. Error messages are grouped into the following types:

- ▶ Script Input Error
- ▶ Translator Error
- ▶ Translator Warning
- ▶ Translator Information

Translator Error means that in some situations the MTK cannot be sure that the translation was correct and manual intervention might be required. The user needs to validate the translation. This differs from the Translator Warning, which indicates that the MTK has automatically made a translation decision.

Expand the messages tree in the left pane to view the message numbers within each message group as shown in Figure 5-10. Each message number provides a short description for that message type. As an example, Message Number 11: Reference to an unknown object. Expand each message number to see a list of occurrences and select a line number for that occurrence.

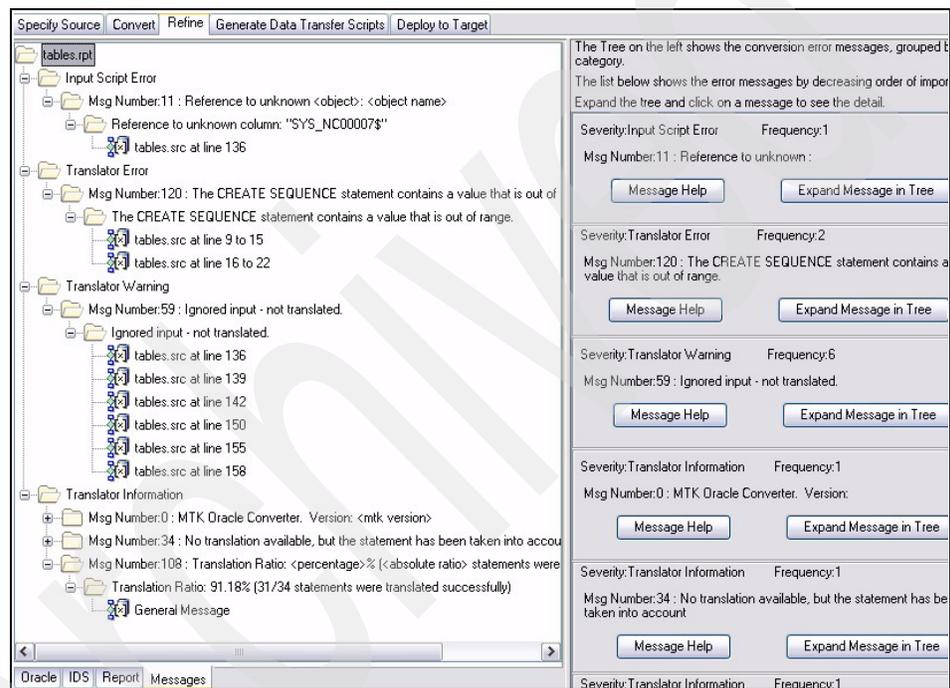


Figure 5-10 MTK translator messages

On the right pane, you will see two sub-tabs, as shown in Figure 5-11 on page 87.

- ▶ Source file: tables.src
The original Oracle extract file
- ▶ Target file: tables.ids
The translation made by the MTK for IDS

Also, note the Translation Rate message at the bottom of the tree. For this conversion, the rate is 91.18% indicating that 31 out of 34 statements were translated successfully.

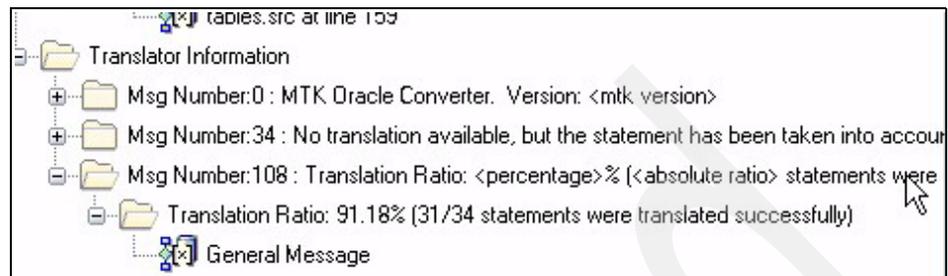


Figure 5-13 Examining the Translator Information

Click **Message Help** on the top right panel for the documentation of MTK features. Also included are Converter Message descriptions. Figure 5-14 illustrates the Help feature.

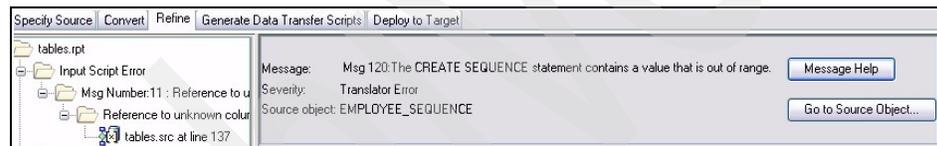


Figure 5-14 Message Help Button

Review the translation file generated by the Refine step to see how the MTK converted Oracle objects to IDS. Table 5-1 summarizes the translation issues encountered:

Table 5-1 Summary of Translation Issues

Oracle Statement	MTK Message	IDS Translation Comments
Create sequence: Employee_sequence Office_sequence	Msg 120 - Error	Oracle MAXVALUE out of range for IDS. MTK generated correct MAXVALUE for IDS sequence.
Create index: on Accounts table	Msg 11 - Error	This is an Oracle functional index and not translated by MTK. This translation is a manual activity. Refer to Chapter 6, “SQL considerations” on page 111 for these types of conversions.
Create index statements with TABLESPACE clause	Msg 59 – Warning	The Oracle TABLESPACE clause of the CREATE INDEX was not translated to IDS. IDS create index statements will be manually updated with dbspace names.

Note: The option to assign Oracle tablespace names to dbspace names for IDS tables and indexes, was not selected. If desired, this feature can be chosen on the Convert tab on the **Advanced Options** button.

At this point, we are satisfied that the MTK has successfully completed as much of the first half of our migration as possible. There are only a few manual translations left to do and we make a note to complete them before deployment.

This ends your first MTK refine.

A word about making manual changes

In this example, the MTK automatically compensated for Oracle-specific DDL constructs. However, during the migration process, as we will see in 5.2, “Part II: Database application object migration” on page 101, some manual intervention is frequently required.

When manual intervention is needed for data type in a table, make the change within the MTK or to the source file and run the Convert step again. All data type changes must be made through the MTK or it might affect how the MTK later extracts and deploys data and converts application objects. There are some changes that can be made outside of the MTK and these occur when the following circumstances are present:

- ▶ It is the final step in the translation process and running the Convert step again will remove all manual changes made to the translation file.
- ▶ You are adding table spaces, or making other physical database design changes, to the DDL in the translation file and deploying the script manually outside of the MTK.
- ▶ For procedures, functions, and triggers, the manual change can be made either to the source or the translation file. When making a change to the translation file, it should be the final step in the translation process. Running the Convert step again will remove all manual changes made to the translation file.

Other common migration considerations

In this section, we discuss some additional common migration considerations.

Renaming object names

For various reasons MTK may assign a new name to an object during migration. One example of this is when converting an Oracle trigger that combines more than one triggering event (INSERT, UPDATE, and DELETE). The MTK will convert this trigger for IDS by generating a separate trigger for each event and also assigning new trigger names. But perhaps you have a standard naming

convention for object names, and the name suggested by the MTK was not appropriate. You can provide a different name for any migrated object by performing the following steps:

1. Under the Refine table, click **Go to Source Object** button, as shown in Figure 5-15. Click this button to take you to a panel where you can see both the Oracle object name and the IDS object name. If the **Go to Source Object** button is not highlighted, you can also get there by selecting the **Oracle** tab on the bottom-left side of the panel.

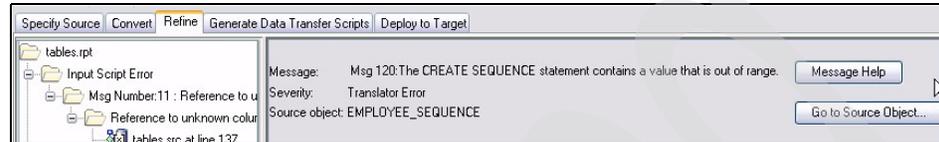


Figure 5-15 Example of a Go to Source object

2. In the panel shown in Figure 5-16, you can select any object to rename by expanding the graphical tree structure on the left side of the panel and highlighting that object.

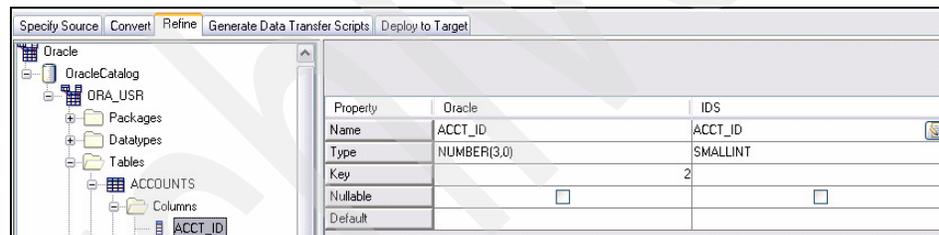


Figure 5-16 Renaming objects

3. When you have highlighted the object that you want to rename, click the edit icon on the right side of the panel, and a window opens for you to provide a new name for the object.
4. Type in the new name and click **Apply** to apply that name.

For the change to take effect, you need to return to the Convert tab and click **Convert** again. The **Convert** button will reread the original source file and apply any changes defined to generate a new tables.ids file. The previous tables.ids file is discarded and a new tables.ids file is generated. You are again taken to the Refine tab of the MTK. The effects of renaming an object persist for the lifetime of the project.

Mapping Oracle tablespaces to IDS dbspaces

You will notice that Oracle tablespaces were not converted. Oracle tablespace definitions do not map easily to IDS dbspaces definitions due to fundamental architecture differences. Also, tablespace definitions that might be optimal in Oracle might not be optimal for IDS. Because the MTK does not map Oracle tablespaces to IDS dbspaces this is a manual migration activity. In addition, under Advanced Options on the Convert tab, if the option to **Use no table spaces in DDL** is selected, then assigning converted tables and indexes to dbspaces is also a manual process and requires editing of the final translation file.

Recommendation: In your own migrations, accept the translated DDL without dbspace definitions for use in unit testing. Revisit the placement of tables and indexes and their dbspaces when ready to create the production database and perform system and performance testing.

5.1.3 Other useful features

In this section, we give a brief overview of some of the other useful features of the MTK.

Viewing the Changes Report

All changes made through the MTK, such as data type mappings and renaming objects, are tracked for you on a project basis. You can view these changes using the Changes Report, which can be accessed from the MTK menu bar under Tools. Figure 5-17 shows the Changes Report menu item.

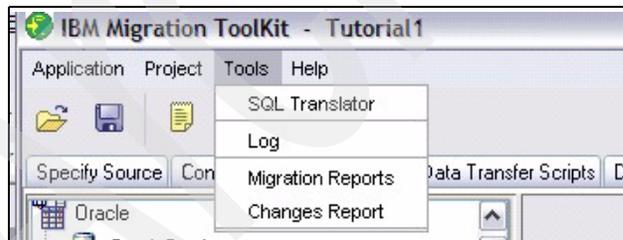


Figure 5-17 Tools menu

Click the Tools tab, then click the Changes Report selection. You will then be presented with the report (Figure 5-18 on page 92). Our current project did not require any changes and our Changes Report reflects that fact.

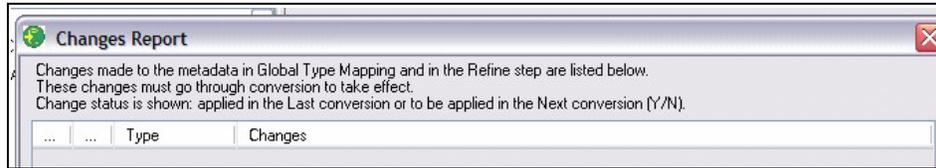


Figure 5-18 Changes report

Generating Data Transfer Scripts tab

The Generate Data Transfer Scripts tab generates the following scripts:

- ▶ Scripts to extract data to flat files from an Oracle database
- ▶ Scripts to load data into an IDS database

This step creates scripts that can be used by MTK to perform data migration.

Note: A common misconception is that data is moved from Oracle to IDS at this point. To be clear, no data is actually moved at this point. Only the scripts are generated. In fact, there is not even a connection to either source or target database at this point.

IDS data loading options

MTK supports 2 methods for loading data into IDS:

- ▶ Use JDBC
- ▶ Use dbaccess

These methods are depicted in Figure 5-19 on page 93. The MTK provides the Informix JDBC driver required for using the JDBC data movement option.

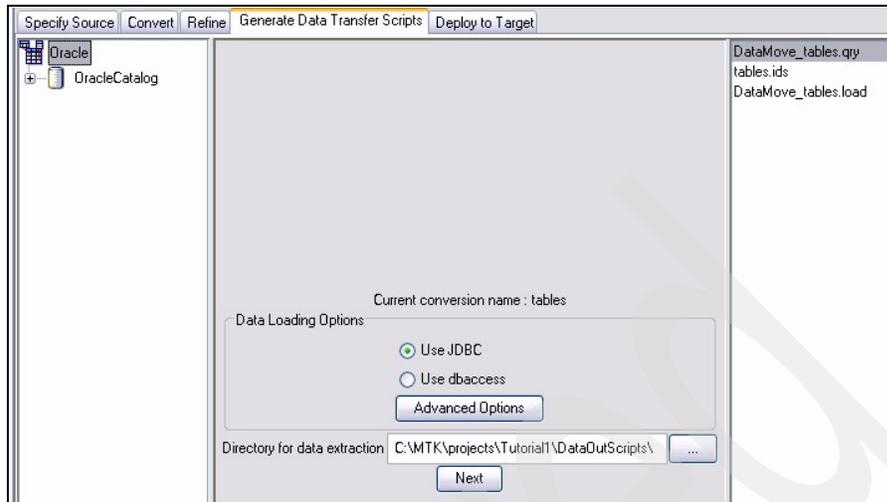


Figure 5-19 Data Loading options

To generate data transfer scripts perform the following steps:

1. Select the data loading option (JDBC or dbaccess).
2. Specify the Data Extraction option.

This entails specifying the column delimiter used to separate the data fields. The default delimiter is |. If your data contains the | character, you can change the delimiter to another character so that deployment can be successful.

3. Specify the directory for data extraction or accept the default directory DataOutScripts, that is located in the project directory.
4. Click **Next** to generate the transfer scripts.

The data transfer scripts are listed in the right panel.

The following types of files are generated:

- ▶ Files used to extract data from the source database: *DataMove_filename.qry*.
A file with SQL statements to select and convert data from the Oracle database. For our migration this file is named *DataMove_tables.qry*
- ▶ Files used to execute the load data into IDS: *DataMove_filename.load* and *filename.ids*.
A file to INSERT data into tables. For our migration these files are named *DataMove_tables.load* and *tables.ids*.

Deploy to target tab

After you have generated the data transfer scripts, you are ready to deploy DDL and data to IDS. Click the Deploy to Target tab to get started.

Deployment is a three-part process:

1. Create objects in IDS by launching the converted script.
2. Extract data from source database to flat files.
3. Deploy data from flat files to DB2.

Deploy conversion name to IDS

Perform the following steps for the deployment:

1. Specify the conversion name, as shown in Figure 5-20 on page 95. In this case, the conversion name is tables (tables.ids) and is the only deployable conversion at this time. If you have completed other conversions such as functions, triggers, and procedures, the pull-down menu will also show those conversion names.
2. Enter information for the IDS target. This information consists of host name, port number, server name, and database name. The MTK can deploy the migrated DDL to an existing database or create a new database. For this tutorial, we deploy our DDL to a new database named example. If your target database is local to the MTK, you can use the (Re)create option. This option will allow you to create and recreate your local database every time you deploy, if necessary. If your database is remote to the MTK, you must manually create your database on the remote system and configure your MTK machine as a client to that database.
3. Enter the user ID and password that you will use to connect to the IDS database and has the authority to create objects.
4. For now, we are only creating the IDS objects from the three part deployment process. Therefore, make sure that only the **Launch tables.ids in the database** option is selected.
5. Click **Deploy** to create the tables in the Example database. The deployment panel is shown in Figure 5-20 on page 95.

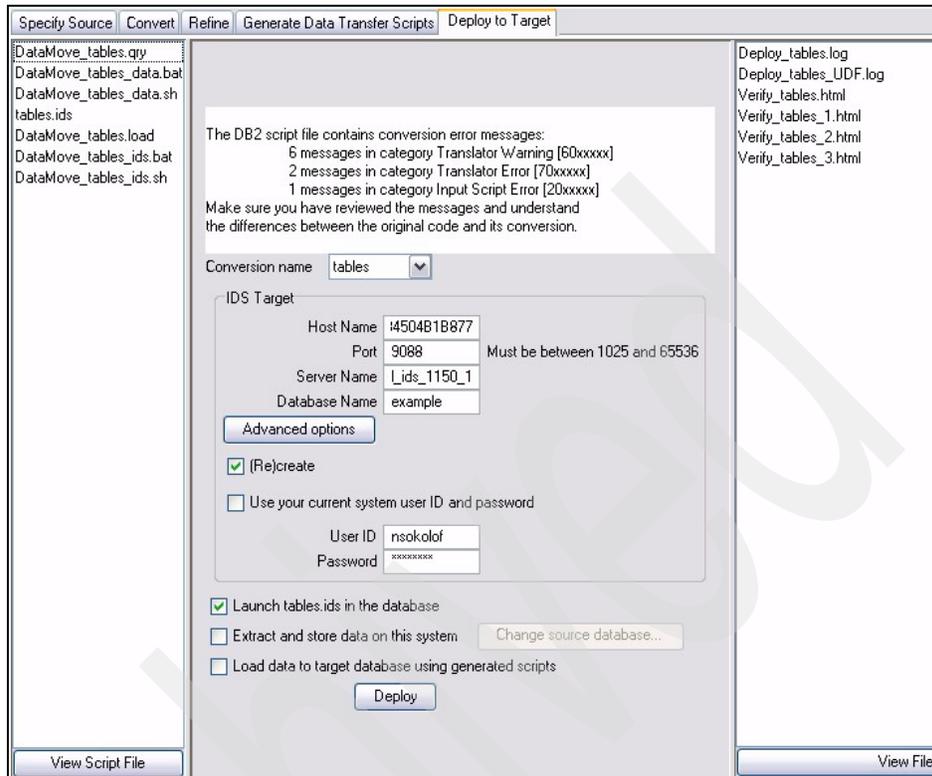


Figure 5-20 MKT: Deploy to target

6. When the deployment completes, a Verification report opens, as shown in Figure 5-21. Confirm that everything deployed properly.

Source NAME	IDS NAME	IDS SCHEMA	TYPE	In IDS	Data extracted	Data in IDS
EMPLOYEE_SEQUENCE	EMPLOYEE_SEQUENCE	nsokolof	SEQUENCE	true		
OFFICE_SEQUENCE	OFFICE_SEQUENCE	nsokolof	SEQUENCE	true		
ACCOUNTS	ACCOUNTS	nsokolof	TABLE	true	not extracted	0 rows
ACCOUNTS_DEPT_CODE_ACCT_ID	ACCOUNTS_DEPT_CODE_ACCT_ID	nsokolof	PRIMARYKEY	true		
FK_ACC_DEPT_CODE	fk_acc_dept_code	nsokolof	FOREIGNKEY	true		
FB_IND_ACCOUNT_RATE	FB_IND_ACCOUNT_RATE	-	INDEX	false		
IND_ACCT_ID	IND_ACCT_ID	nsokolof	INDEX	true		
DEPARTMENTS	DEPARTMENTS	nsokolof	TABLE	true	not extracted	0 rows
PK_DEPT_CODE	PK_DEPT_CODE	nsokolof	PRIMARYKEY	true		
IND_DEPT_NAME	IND_DEPT_NAME	nsokolof	INDEX	true		
DESTINATION	DESTINATION	nsokolof	TABLE	true	not extracted	0 rows
EMPLOYEES	EMPLOYEES	nsokolof	TABLE	true	not extracted	0 rows
SYS_C0012108	SYS_C0012108	nsokolof	PRIMARYKEY	true		
FK_EMP_MGR_ID	fk_emp_mgr_id	nsokolof	FOREIGNKEY	true		
FK_EMP_OFFICE_ID	fk_emp_office_id	nsokolof	FOREIGNKEY	true		

Figure 5-21 Verification Report

Our report showed that the deployment step ran successfully, except for the creation of the FB_IND_ACCOUNT_RATE index which is highlighted in red. Previously, our Refine step reported that the MTK could not convert the functional index and we flagged this item for a manual conversion. We cover how to convert Oracle functional indexes in Chapter 6, “SQL considerations” on page 111.

Check the deploy_filename.log, named deploy_tables.log, as shown in Figure 5-22 on page 97 for the status of each IDS statement executed.

```
Deploy_tables.log - Notepad
File Edit Format View Help
CREATE SEQUENCE EMPLOYEE_SEQUENCE INCREMENT BY
9223372036854775807 MINVALUE 1 NOCYCLE
The SQL command completed successfully.

CREATE SEQUENCE OFFICE_SEQUENCE INCREMENT BY
9223372036854775807 MINVALUE 1 NOCYCLE
The SQL command completed successfully.

CREATE TABLE ACCOUNTS( ACCT_ID SMALLINT NOT N
ACCT_DESC LVARCHAR(2000), MAX_EMPLOYEES SMALL
NUM_PROJECTS SMALLINT);
The SQL command completed successfully.

ALTER TABLE ACCOUNTS ADD CONSTRAINT PRIMARY KEY
ACCOUNTS_DEPT_CODE_ACCT_ID;
The SQL command completed successfully.
```

Figure 5-22 Deploy log

We are now ready to extract and load data into IDS.

Migrating the data

When you perform your actual migration, we recommend that you proceed with the data deployment, as described in the following steps, to give you more control and to avoid too much activity in one step:

1. When you are comfortable with the results of step 1 of the deployment process, proceed to the second option, **Extract and store data on this system**, shown in Figure 5-23. Click **Deploy** again to perform data extraction to flat files. Click the **Change source database** button to make sure that your Oracle database connection information is provided.

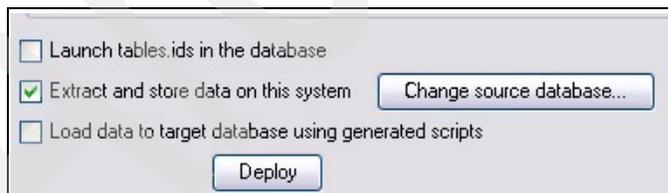


Figure 5-23 Deployment of data

2. If the extraction completed successfully, select the third check box, **Load data to target database using generated scripts**, as shown in Figure 5-24 on page 98, and click **Deploy**. This will load the data into Informix from the flat files created in the previous step.

Launch tables.ids in the database
 Extract and store data on this system Change source database...
 Load data to target database using generated scripts

Deploy

Figure 5-24 Deployment option 3

- When the deployment completes, a verification report, as shown in Figure 5-25, launches to show the results. The report lists the number of rows extracted versus the number of rows loaded. In our example, some source tables were empty and so data was not loaded for those tables. If your data has loaded as expected, you have successfully completed part 1 of the migration.

Source NAME	IDS NAME	IDS SCHEMA	TYPE	In IDS	Data extracted	Data in IDS
EMPLOYEE_SEQUENCE	EMPLOYEE_SEQUENCE	nsokolof	SEQUENCE	true		
OFFICE_SEQUENCE	OFFICE_SEQUENCE	nsokolof	SEQUENCE	true		
ACCOUNTS	ACCOUNTS	nsokolof	TABLE	true	0 rows	0 rows
ACCOUNTS_DEPT_CODE_ACCT_ID	ACCOUNTS_DEPT_CODE_ACCT_ID	nsokolof	PRIMARYKEY	true		
FK_ACC_DEPT_CODE	fk_acc_dept_code	nsokolof	FOREIGNKEY	true		
FB_IND_ACCOUNT_RATE	FB_IND_ACCOUNT_RATE	.	INDEX	false		
IND_ACCT_ID	IND_ACCT_ID	nsokolof	INDEX	true		
DEPARTMENTS	DEPARTMENTS	nsokolof	TABLE	true	5 rows	5 rows
PK_DEPT_CODE	PK_DEPT_CODE	nsokolof	PRIMARYKEY	true		
IND_DEPT_NAME	IND_DEPT_NAME	nsokolof	INDEX	true		
DESTINATION	DESTINATION	nsokolof	TABLE	true	0 rows	0 rows
EMPLOYEES	EMPLOYEES	nsokolof	TABLE	true	0 rows	0 rows
SYS_C0012108	SYS_C0012108	nsokolof	PRIMARYKEY	true		
FK_EMP_MGR_ID	fk_emp_mgr_id	nsokolof	FOREIGNKEY	true		
FK_EMP_OFFICE_ID	fk_emp_office_id	nsokolof	FOREIGNKEY	true		
M_DEPT_CODE_ACCT_ID	m_dept_code_acct_id	nsokolof	FOREIGNKEY	true		
IND_EMP_NAME	IND_EMP_NAME	nsokolof	INDEX	true		
EMP_PHOTO	EMP_PHOTO	nsokolof	TABLE	true	0 rows	0 rows

Figure 5-25 Data load verification report

You can view the MTK log from the menu bar by navigating to **Tools** → **Log**. This log records activities by time stamp to check the elapsed time for the data load and other processes. An example is depicted in Figure 5-22 on page 97.

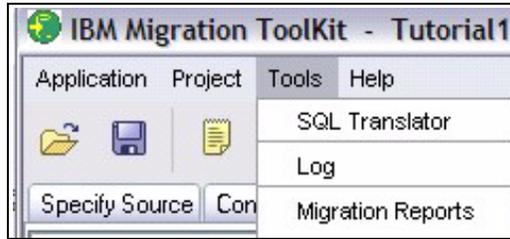


Figure 5-26 MTK Log

As you will see when you are migrating functions, triggers, and procedures in 5.2, “Part II: Database application object migration” on page 101, you will simply repeat deployment step 1. However, you will not have to perform step 2 or 3 again.

5.1.4 Additional MTK features

This section describes some additional features of the MTK.

SQL Translator

Now that you have successfully migrated the objects and data, we can examine another useful feature of the MTK, the SQL Translator. From the MTK menu bar, navigate to **Tools** → **SQL Translator**, as shown in Figure 5-27.

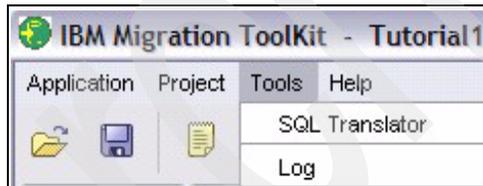


Figure 5-27 SQL Translator

The SQL Translator is the GUI interface for translating ad hoc SQL (SQL can also be imported in a script). Before you can translate SQL this way, you must first point the MTK to the conversion names for the converted object definitions on which the SQL depends. To do this, ensure that the default **Use all files of current conversion** is selected from the pull-down list.

Running the MTK debugger

From the OS command line in the default MTK installation directory (the C:\MTK directory on Windows and a user-named directory on UNIX), start the debugger by executing the command:

```
MTKMain -debug
```

Run the MTK task that you are attempting to debug and view the MTK log (mtk.log) to see logged messages at a more granular level.

5.1.5 Summary of best practices when using the MTK

The following summarizes the practices we employed when converting Oracle objects to IDS. Although some of these practices are fairly intuitive, we state them again as a review.

- ▶ The MTK translates by parsing a text input file. Even if no source database is available, it supports a user-provided input file for migration. If you have a source database, the MTK extracts the DDL from the database into a text file from which to translate.
- ▶ When extracting objects for migration, do not try to do too much in one step. In the example we provided, we start with tables and related items (such as indexes and constraints) only. Consider migrating tables and related objects separately from triggers, functions, and procedures. If your database is large, you might want to further divide your migration effort.
- ▶ Some objects might require renaming. Within the REFINE step, you have the opportunity to override the default name mapping if desired.
- ▶ Every time you click **Convert**, the MTK rereads the input file, applies user-defined changes, and generates an entirely new output file.
- ▶ The MTK might not be able to provide REFINE capability for some warnings for errors. You can manually change the input file to suit your needs and click **Convert** again. This applies to any objects that have problems converting.
- ▶ An index TABLESPACE clause is not directly convertible and requires manual intervention based on physical database design and tuning considerations.
- ▶ After you have a clean conversion, you can reduce the use of commenting in the output file by disabling it in the Advanced Options.
- ▶ Insert statements can be used to populate your tables with a few rows of test data. This practice has been referred to as seeding. If your application seeds the database with initial data using INSERT statements, data extraction and deployment might not be necessary. Instead, you can convert the INSERT statements directly.

- ▶ If your source database is small, you can have the MTK create a database for the initial deployment. However, if the database is large, you should pre-create the database and do some initial tuning before deployment (such as dbspace space layout, buffers, and so on).
- ▶ When you deploy to IDS, do not deploy everything at the same time. Start with DDL and ensure that all objects were created successfully before proceeding with data. Then, perform data extraction and data deployment as separate steps.

5.2 Part II: Database application object migration

Now that the table, view, index, and constraints DDL has been migrated, database application objects, such as functions, procedures, packages, and triggers, can be migrated. Application object migration tends to be more involved, but the MTK helps to make this task far more manageable.

Converting procedures, functions, packages and triggers

This section describes the steps required to migrate the application objects. We extract these objects from the Oracle example database to show you the process. With that introduction, let's get started.

To start the migration, perform the following steps:

1. In the same project that we used to convert “Part 1: Core database object migration”, click the Specify Source tab at the top of the panel and click **Extract**.
2. In the **Extract** pane, expand the tree for the ora_usr schema and select the boxes for **Procedures/Functions, Triggers and Packages**. Enter the name procedures in the filename box as the name of this conversion. You should see a window similar to the one shown in Figure 5-28.



Figure 5-28 Extract procedures/functions, triggers and packages

3. Click **Extract**.

You should now see the procedures.src file listed on the right side of the Specify Source panel (as depicted in Figure 5-29). Briefly familiarize yourself with the objects by clicking **View** on the bottom of the panel.

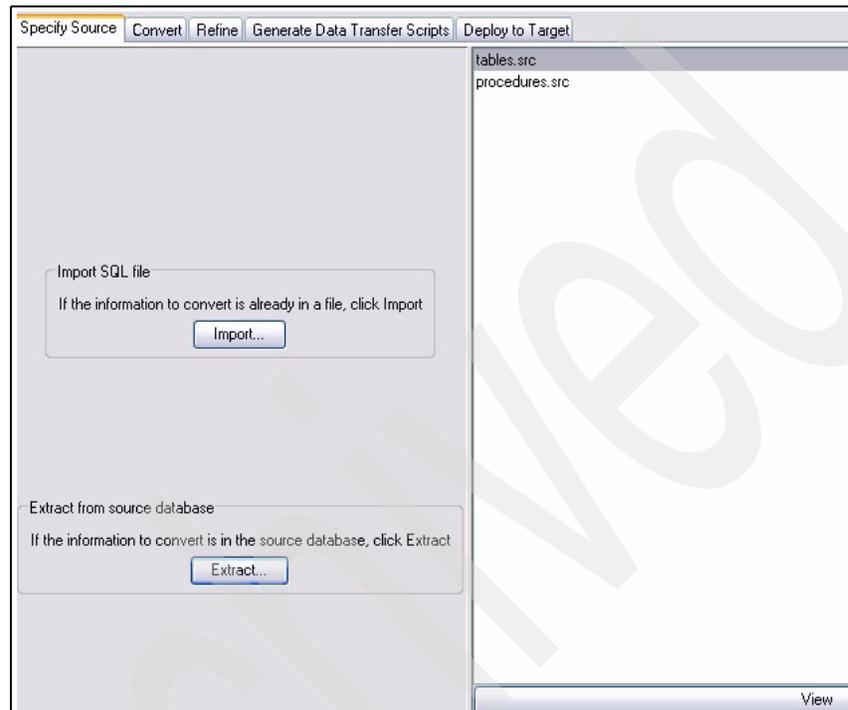


Figure 5-29 Specify Source Tab

4. Move to the Convert tab.

5. Before we convert the objects, we must first set the context (dependencies) of the conversion because each function, procedure, and trigger references tables, views and sequences converted in Part 1. Click **Set Context**, as in Figure 5-30 on page 103.

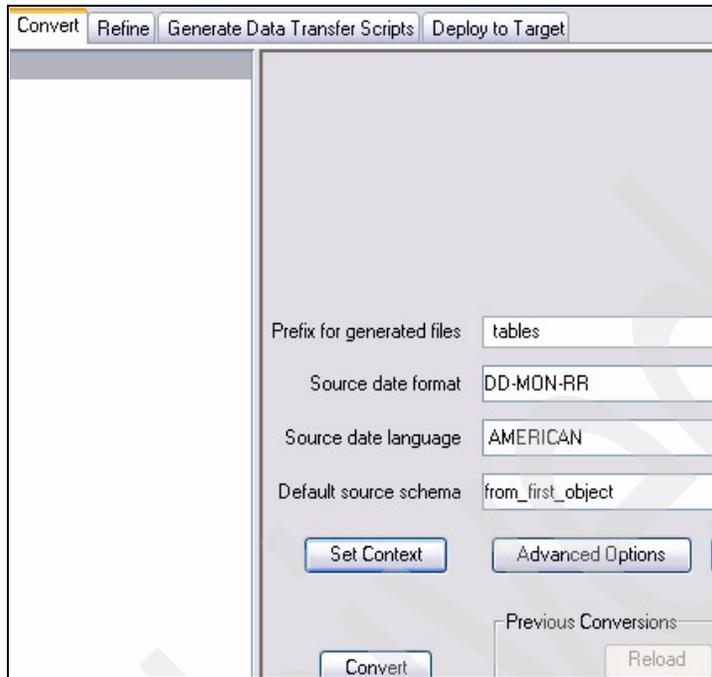


Figure 5-30 Mtk: Setting context

6. In the Set Context panel, shown in Figure 5-31, indicate to the MTK that your next conversion will have dependencies on tables.src by clicking the single arrow (>) to send the tables.src file to the right side of the panel. Click **OK** to close the window.



Figure 5-31 Set Context panel

7. Click **OK** to return to the Convert tab.

Notice that the context indicator (context) now appears next to the tables.src file. Before starting, click **Advanced Options** and select the options as shown in Figure 5-32. This will make the target.ids translation file cleaner and easier to read.

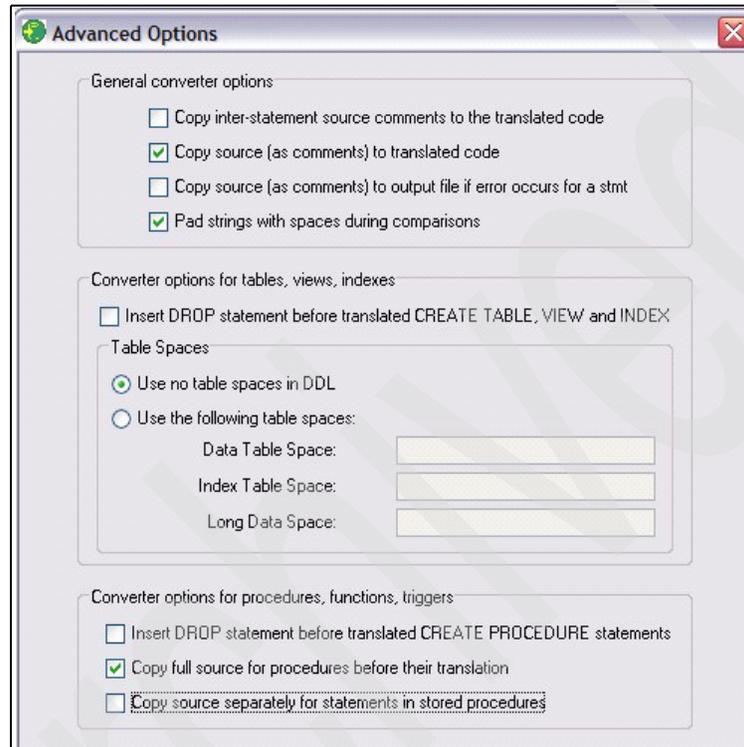


Figure 5-32 Advanced Options panel

8. We can now proceed with the migration, using the following steps:
 - a. Select the procedures.src file on the left side of the panel.
 - b. Click **Convert**.

This will begin converting the objects to IDS and take you to the Refine tab. We now begin refining the conversion the same way we refined the conversion of objects in Part 1 (tables, views, indexes, and so on) by working through the messages tree.

9. Expand the messages tree, as in Figure 5-33.

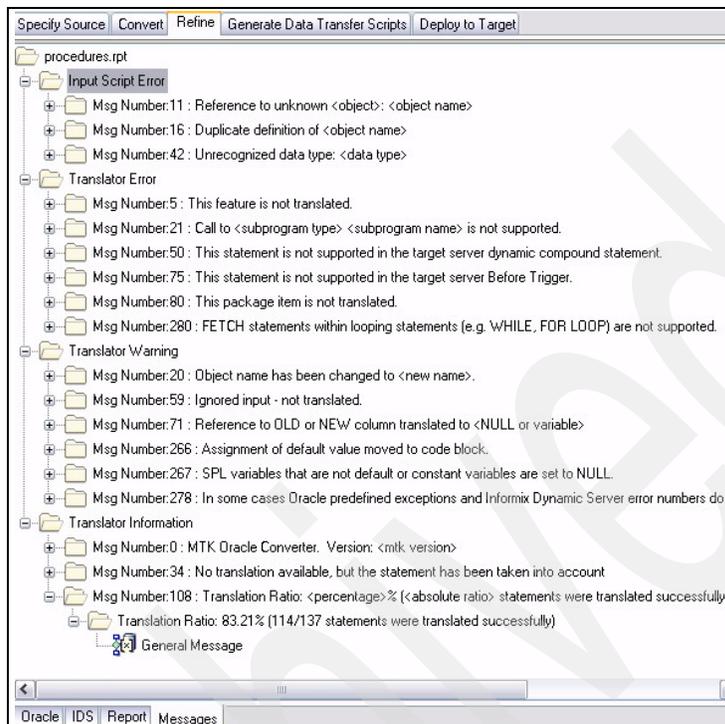


Figure 5-33 Refine messages tree

Make note of the Translation Information. The translation ratio is 83.21%. We find that 114 out of 137 statements were translated successfully.

Next, expand the tree for each Msg Number to view the number of occurrences found. As we select each occurrence by line number, we are brought to the Oracle statement in the source file that it references. When we switch tabs from the Source File tab to the Target file tab, we are brought to the translation in the target file or to a converter message if the translation does not exit. If you need further clarification for a message, refer to **Converter Messages**, which can be accessed through the MTK help menu bar.

Note: The conversion information in the Refine Messages Tree is cumulative. That is, even though we are reviewing messages for the procedures conversion, the Messages Tree also holds messages from the tables conversion. Each occurrence of a message number references the source file it refers to (procedures.src or tables.src). This is shown in Figure 5-34.

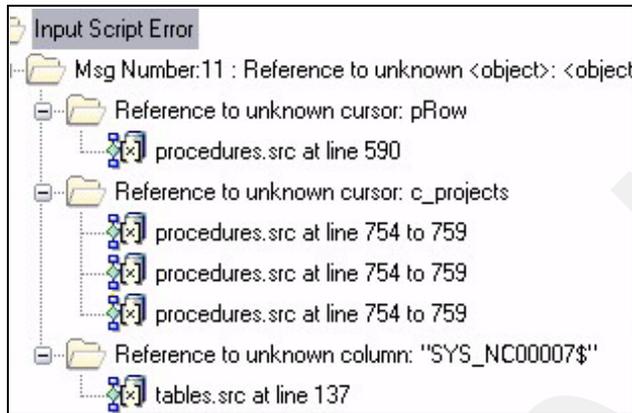


Figure 5-34 Msg numbers and references to conversion names

Review the translation file generated by the Refine step to see the results of the translation. Most objects will generate more than one message. Some messages are just informational and do not mean that a translation did not occur.

Because of the number of messages generated, we summarize our observations about the translation in Table 5-2. The “State of Translation” column contains either “Complete” or “Incomplete”. A state of “Complete” indicates that the translation appears to be complete but still requires further validation. A state of “Incomplete” means that MTK was only able to complete a portion of the translation and that manual intervention is required.

Table 5-2 Summary of observations

Object Oracle	State of Translation	Translation to IDS - Comments
AccountFull (function)	Complete	Execute and Test
AverageBand (function)	Complete	Execute and Test
CountProjects (function)	Complete	Execute and Test
MaxProjects (function)	Complete	Execute and Test
ShowfullAccounts (procedure)	Complete	Execute and Test
AddNewEmployee (procedure)	Complete	Execute and Test
AccountPackage (package):	Complete	Package contained 3 procedures. MTK converted the package into 3 procedures and used the package name as the schema name for each procedure.

Object Oracle	State of Translation	Translation to IDS - Comments
- AddEmployee (procedure)	Complete	Renamed Accountpackage.AddEmployee - execute and test
- RemoveEmployee (procedure)	Incomplete	Renamed Accountpackage.RemoveEmployee: DBINFO('sqlca.sqlerrd2') added to handle the Oracle SQL%NOTFOUND for the row count of a DELETE statement.
- AccountList (procedure)	Incomplete	Renamed Accountpackage.Accountlist: This procedure was converted to a Function on IDS because it returned data. A cursored LOOP statement in Oracle was manually replaced with a FOREACH construct on IDS.
Refpkg (package)	Incomplete	This package is not needed. It declares a variable as a REF cursor and will not be used on IDS.
Assign (procedure)	Incomplete	Oracle Exception translated to ON EXCEPTION on IDS for a "not found" condition.
Employeeedynamic (procedure)	Incomplete	Oracle dynamic SQL requires manual conversion to dynamic SQL on IDS.
SelectRow (procedure)	Incomplete	This procedure with an Oracle REF cursor variable will be manually converted to an IDS function using RETURN WITH RESUME.
EmployeeOfficesInsert (instead trigger)	Incomplete	Convert to an IDS trigger with a procedure call.
InsertEmployee (before trigger)	Incomplete	Convert to an IDS trigger with a procedure call.
ManagersChange (multiple triggered events)	Incomplete	MTK converted the multiple triggered events into separate triggers for IDS. Convert to triggers with procedure calls.
Office_summary_delete (instead trigger)	Incomplete	Manual completion of IDS Instead trigger syntax required.
UpdateDepartments (trigger)	Incomplete	Convert to an IDS trigger with a procedure call.
UpdateEmployees (multiple triggeredevents)	Incomplete	MTK converted multiple triggered events into separate triggers for IDS. Convert trigger to call procedures.
CreateEmployeeID (trigger)	Incomplete	Convert to IDS trigger with procedure call. Dual table conversion.

Note: IDS triggers do not support SPL statements in the trigger body. Convert an Oracle trigger with PLSQL statements to an IDS trigger that executes a procedure.

After the procedures, functions, packages, and triggers have been translated, deploy to IDS using the same steps used in Part 1 when we deployed the core database objects in the tables.ids file. To do so, continue with the migration steps:

10. Click the Deploy to Target tab.
11. From the Conversion name list, select the procedure.ids file.
12. Ensure that the target database name indicates example. Do not select (Re)create database.
13. Enter your connection user ID and password. Use a user ID that has authority to create objects.
14. Select **Launch procedures.ids in the database**.
15. Click **Deploy**.

After the deployment completes, check the Verification report. You should see a status of each object that was deployed, as depicted in Figure 5-35.

EMPLOYEE_SEQUENCE	EMPLOYEE_SEQUENCE	nsokolof	SEQUENCE	true	
OFFICE_SEQUENCE	OFFICE_SEQUENCE	nsokolof	SEQUENCE	true	
ACCOUNTFULL	AccountFull	nsokolof	PROCEDURE/FUNCTION	true	
AVERAGEBAND	AVERAGEBAND	nsokolof	PROCEDURE/FUNCTION	true	
COUNTPROJECTS	COUNTPROJECTS	nsokolof	PROCEDURE/FUNCTION	true	
MAXPROJECTS	MAXPROJECTS	nsokolof	PROCEDURE/FUNCTION	true	
SHOWFULLACCOUNTS	ShowFullAccounts	nsokolof	PROCEDURE/FUNCTION	true	

Figure 5-35 Verification report

Important: Even if the Refine step reports that translation is 100% and the object deployed successfully, the final test for correct conversion of an application object occurs when it is executed. You must ensure that the semantics of the application object were not changed and that it performs exactly as it did on the source database.

5.2.1 Migration of application objects: Lessons learned

The following list summarizes the lessons we learned by migrating procedures, functions, packages, and triggers from Oracle to IDS:

- ▶ Limit the copying of source code in the Advanced Options to generate an easier to read output file.
- ▶ When a project involves multiple files, you might need to identify dependencies by setting the context of the migration using the **Set Context** button.
- ▶ When the MTK identifies a construct that cannot be converted, manual intervention is required. Changes can be made either to the source or translation file depending on the nature of the change. If changes are made to the translation file, do not run **Convert** again.
- ▶ In some cases, an incorrect translation might not be detected until deploying or executing the application object.
- ▶ The MTK does not translate Oracle dynamic SQL, reference cursors, or cursor variables such as %NOTFOUND.
- ▶ SQL error codes differ between Oracle and IDS.
- ▶ The MTK can translate %TYPE and %ROWTYPE.
- ▶ Oracle dual table can be converted to the IDS sysmaster:sysdual table.
- ▶ The MTK converts Oracle packages by extracting the procedures and functions into separate objects and uses the package name as the schema name for the procedure or function.
- ▶ Oracle triggers that contain PLSQL statements are converted to IDS triggers that call an SPL procedure using the EXECUTE PROCEDURE statement.
- ▶ Oracle procedures that return data are converted to functions with the RETURNING clause on IDS.
- ▶ Oracle procedures that return data using a REF cursor variable are converted to an IDS function using RETURN WITH RESUME.
- ▶ When a procedure is converted to a function or visa versa, application changes might be required in the calling program.
- ▶ You can switch back and forth between conversions using the **Reload** button.

Note: To view a Verification report other than the current report just generated after deployment, click the Convert tab. From Previous Conversions, select the name of the past conversion from the list and click Reload. This will make all files of the past conversion available for viewing, including the Verification report.

Click the Deploy tab, and on the right side of the panel, double-click **Verify_conversion_name.html**. The Verification report from a previous conversion opens.

In addition, cumulative migration reports can be viewed from the MTK menu by selecting **Tools** → **Migration Reports**.

SQL considerations

In this chapter, we discuss the SQL functionality and syntax for Oracle and the Informix Dynamic Server (IDS). Although the focus for this chapter is on Oracle 10g Release 2 and Informix Dynamic Server Version 11.50, it can also be used for migrations that are at other version levels. In addition to DDL, DML, and overall SQL syntax, we also discuss differences between Oracle and Informix with regards to server-side programming, with Oracle PL/SQL and Informix SPL.

6.1 DDL

In this section we describe how Oracle and Informix handle DDL statements, such as create database, create and alter tables, indexes, triggers, sequences, procedures, functions, and synonyms.

6.1.1 Database creation

Informix supports both ANSI and non-ANSI databases. However, Oracle transaction behavior is more closely aligned with ANSI databases. In Oracle and Informix ANSI databases, a transaction implicitly begins at the start of the program and ends after execution of a commit or rollback statement.

To create an ANSI-compliant database, the `WITH LOG MODE ANSI` clause must be added to the create database statement.

The following command creates an ANSI-complaint database called STORES:

```
CREATE DATABASE STORES WITH LOG MODE ANSI;
```

ANSI-compliant databases are different from databases that are not ANSI-compliant in several ways:

- ▶ All SQL statements are automatically contained in transactions.
- ▶ All databases use unbuffered logging.
- ▶ For sessions, the default isolation level is REPEATABLE READ.
- ▶ Default privileges on objects differ from those in databases that are not ANSI compliant. When you create a table or a synonym, other users do not receive access privileges (as members if the PUBLIC group) to the object by default.
- ▶ All DECIMAL data types are fixed-point values. If you declare a column as DECIMAL(p), the default scale is zero, meaning that only integer values can be stored. In a database that is not ANSI-compliant, DECIMAL(p) is a floating-point data type of a scale large enough to store the exponential notation for a value.
- ▶ Owner naming is enforced. You must qualify with the owner name any table, view, synonym, index, or constraint that you do not own. Unless you enclose the owner name between quotation marks, alphabetic characters in owner names default to uppercase. To prevent this upshifting of lowercase letters in undelimited owner names, set the ANSIOWNER environment variable to 1.

6.1.2 Tables

When creating tables, the Oracle CREATE TABLE statement must be converted to Informix, taking advantage of the Informix options IN, EXTENT, NEXT, and LOCK MODE.

- ▶ IN specifies the dbspace in which the table will reside.
- ▶ EXTENT specifies the amount of space that will initially be allocated to the table.
- ▶ NEXT specifies the amount of space that will be allocated when additional space is needed.

Lock mode option

LOCK MODE specifies whether to use row or page locks for the table. In Oracle, the default is row level locking. In Informix the default setting is page level locking. However, the default can be changed by using one of the following methods (resolved in the following order of precedence):

1. LOCK MODE specified using an attribute of the CREATE TABLE or ALTER TABLE command syntax
2. IFX_DEF_TABLE_LOCKMODE environment variable setting
3. DEF_TABLE_LOCKMODE parameter setting in the ONCONFIG file

If the DEF_TABLE_LOCKMODE parameter cannot be found in the ONCONFIG file, it can be added to make the specification for every database within the instance. The Informix instance must be restarted for this parameter to take effect.

Storage clauses

The Oracle STORAGE INITIAL and STORAGE NEXT clauses must be changed to the Informix EXTENT SIZE and NEXT SIZE clauses, respectively. In Oracle the STORAGE clause options MINEXTENTS, MAXEXTENTS and PCTINCREASE should be removed.

When an Oracle CREATE TABLE statement does not include a STORAGE clause, the table will be created using the tablespace STORAGE clause by default. If an Oracle tablespace is created specifying a STORAGE INITIAL of 100 KB and a STORAGE NEXT of 50 KB, then all tables created within that tablespace will have a default value of STORAGE INITIAL 100 KB and STORAGE NEXT 50 KB. Oracle tables cannot be created with smaller STORAGE clause values than the tablespace default in which they are created.

Informix dbspaces do not have storage clauses attached to them. As previously stated, the default Informix EXTENT and NEXT sizes are four times the disk page size on your system. For example, if you have a 4 KB page system, the minimum length is 16 KB.

Constraints

The Oracle constraint syntax must be changed to the Informix syntax for primary keys, foreign keys, unique, and so on.

Primary Key Constraint

Oracle Syntax:

```
CONSTRAINT name PRIMARY KEY(column(s))
```

Oracle Example:

```
CONSTRAINT PK_ACCT_ID PRIMARY KEY(ACCT_ID)
```

Informix Syntax:

```
PRIMARY KEY (column(s)) CONSTRAINT name
```

Informix Example:

```
PRIMARY KEY(ACCT_ID) CONSTRAINT PK_ACCT_ID
```

Foreign Key Constraint

Oracle Syntax:

```
CONSTRAINT name FOREIGN KEY(column(s)) references_clause  
[ON DELETE CASCADE / ON DELETE SET NULL]
```

Oracle Example:

```
CONSTRAINT FK_ACC_DEPT_CODE FOREIGN KEY ( DEPT_CODE )  
REFERENCES DEPARTMENT ( DEPT_CODE ) ON DELETE CASCADE
```

Informix Syntax:

```
FOREIGN KEY(column(s)) references_clause  
[ON DELETE CASCADE] CONSTRAINT name
```

Informix Example:

```
FOREIGN KEY ( DEPT_CODE ) REFERENCES DEPARTMENT ( DEPT_CODE )  
ON DELETE CASCADE CONSTRAINT FK_ACC_DEPT_CODE
```

Note: There is no ON DELETE SET NULL in Informix. However, you could write a stored procedure or a trigger to get that capability. Refer to the CREATE TABLE statement syntax in the *Informix Guide to SQL: Syntax*, G229-6375, for information about the Informix referential integrity constraint.

Unique Constraint

Oracle Syntax:

```
CONSTRAINT name UNIQUE(column(s))
```

Oracle Example:

```
CONSTRAINT UK_PHONE UNIQUE(PHONE)
```

Informix Syntax:

```
UNIQUE (column(s)) CONSTRAINT name
```

Informix Example:

```
UNIQUE(PHONE) CONSTRAINT UK_PHONE
```

Check Constraint

Oracle Syntax:

```
CONSTRAINT name CHECK (condition)
```

Oracle Example:

```
CONSTRAINT CHK_ORD_MONTH CHECK (ORDER_MONTH BETWEEN 1 AND 12)
```

Informix Syntax:

```
CHECK (condition) CONSTRAINT name
```

Informix Example:

```
CHECK (ORDER_MONTH BETWEEN 1 AND 12) CONSTRAINT CHK_ORD_MONTH
```

Note: An Oracle check constraint statement may contain functions that are not available in Informix. To get similar functionality, you can create stored procedures in Informix.

Parallel

The Oracle PARALLEL clause, sets the degree of parallelism for creation of the table and also INSERT, UPDATE, DELETE, and MERGE operations after the table is created.

The Oracle PARALLEL clause must be removed from any SQL statements (such as CREATE TABLE and ALTER TABLE) when porting the statement to Informix.

In Informix, you can accomplish parallel execution using Parallel Database Query (PDQ) feature. PDQ divides large database queries into multiple parallel tasks, which can dramatically improve performance when the database server processes queries that are initiated by decision-support applications.

In Informix, PDQ operations include scans, sorts, joins, aggregates, inserts, and deletes. Unlike Oracle, you cannot use PDQ at the time of creation of the table. PDQ can be turned on for a particular query or set of queries using the SQL statement SET PDQPRIORITY. It can also be turned on for all queries run by particular users, with the environmental parameter PDQPRIORITY.

Note: Refer to *Informix Dynamic Server Performance Guide*, G229-6385, for more information about the Parallel Database Query (PDQ) feature in Informix.

Example 6-1 shows how Oracle and Informix tables are typically created, and how they differ from each other.

Example 6-1 Create table example

Oracle Create Table:

```
CREATE TABLE accounts(  
    acct_id NUMBER(3) NOT NULL,  
    dept_code CHAR(3) NOT NULL,  
    acct_desc VARCHAR2(2000),  
    max_employees NUMBER(3),  
    current_employees NUMBER(3),  
    num_projects NUMBER(1),  
    CONSTRAINT PK_ACCT_ID PRIMARY KEY(ACCT_ID),  
    CONSTRAINT FK_ACC_DEPT_CODE FOREIGN KEY ( DEPT_CODE )  
    REFERENCES DEPARTMENTS (DEPT_CODE)  
)  
STORAGE (INITIAL 36k NEXT 36K MINEXTENTS 1 MAXEXTENTS 50)  
TABLESPACE user_data_tbs;
```

Informix Create Table:

```
CREATE TABLE accounts(  
    acct_id SMALLINT NOT NULL,  
    dept_code CHAR(3) NOT NULL,  
    acct_desc LVARCHAR(2000),  
    max_employees SMALLINT,  
    current_employees SMALLINT,  
    num_projects SMALLINT,  
    PRIMARY KEY(acct_id) CONSTRAINT PK_ACCT_ID,  
    FOREIGN KEY (dept_code) REFERENCES DEPARTMENTS(DEPT_CODE)  
    CONSTRAINT FK_ACC_DEPT_CODE  
)  
IN user_data_tbs  
EXTENT SIZE 36 NEXT SIZE 36;
```

Table fragmentation

Table partitioning in Oracle is what is known as table fragmentation in Informix. Informix recommends using table fragmentation to improve single-user response time, concurrency, data availability, backup/restore characteristics, and data-load performance.

Informix fragmentation strategies

Informix allows table fragmentation based on two strategies, round robin and expression-based fragmentation.

1. Round robin:

As records are inserted into the table, they are placed in the first available fragment in round robin fashion. The database server balances the load among the specified fragments as you insert records and distributes the rows in such a way that the fragments always maintain approximately the same number of rows. In this distribution scheme, the database server must scan all fragments when it searches for a row.

2. Expression®-based fragmentation:

In an expression-based distribution scheme, each fragment expression in a rule specifies a storage space. Each fragment expression in the rule isolates data and aids the database server in searching for the rows.

To fragment a table by expression, specify one of the following rules:

– Range rule:

A range rule specifies fragment expressions that use a range to specify which rows are placed in a fragment, as shown in the following example:

```
FRAGMENT BY EXPRESSION
c1 < 100 IN dbsp1,
c1 >= 100 AND c1 < 200 IN dbsp2,
c1 >= 200 IN dbsp3;
```

– Arbitrary rule:

An arbitrary rule specifies fragment expressions based on a predefined SQL expression that typically uses OR clauses to group data, as shown in the following example:

```
FRAGMENT BY EXPRESSION
zip_num = 95228 OR zip_num = 95443 IN dbsp2,
zip_num = 91120 OR zip_num = 92310 IN dbsp4,
REMAINDER IN dbsp5;
```

Oracle partitioning strategies

In Oracle, there are four table partitioning strategies:

▶ Range partitioning

Range partitioning creates partitions based on the range of column values of the specific column. Each partition is defined by a partition bound that limits the scope of the partition.

▶ HASH partitioning

Hash partitioning creates partitions based on a hash value generated from the value of a specific column that is used for partitioning. Hash partitioning is based on an internal hash algorithm used by Oracle.

▶ List partitioning

In this partitioning method, the table data are partitioned based on the list of values of a column.

▶ Composite Range-Hash partitioning

This partitioning method is a combination of range and hash partitioning where partitions can have subpartitions of the following types:

- Range partition can contain multiple hash subpartitions.
- Range partition can contain multiple list subpartitions.

In Table 6-1 on page 119 we show the Oracle partitioning strategies and a proposed Informix alternative.

Table 6-1 Informix fragmentation strategy alternatives to Oracle strategies

Oracle Partitioning Strategy	Informix Fragmentation Strategy
Range Partitioning	Expression Based Fragmentation
List Partitioning	Expression Based Fragmentation
Hash Partitioning	Round Robin Fragmentation
Composite Range-Hash Partitioning	Expression Based Fragmentation

In Example 6-2 we show an example of Oracle range partitioning and the equivalent functionality with Informix expression-based fragmentation.

Example 6-2 Range partitioning example

Oracle Range Partitioning:

```
CREATE TABLE accounts
(
    acct_id NUMBER(3) NOT NULL,
    dept_code CHAR(3) NOT NULL,
    acct_desc VARCHAR2(2000),
    max_employees NUMBER(3),
    current_employees NUMBER(3),
    num_projects NUMBER(1)
)
PARTITIONING BY RANGE (acct_id)
(PARTITION part1 VALUES LESS THAN (100)
    TABLESPACE part1_tbs,
(PARTITION part2 VALUES LESS THAN (200)
    TABLESPACE part2_tbs,
(PARTITION part3 VALUES LESS THAN (300)
    TABLESPACE part3_tbs,
(PARTITION part4 VALUES LESS THAN (MAXVALUE)
    TABLESPACE part4_tbs);
```

Equivalent Informix Expression-based Fragmentation:

```
CREATE TABLE accounts(
    acct_id SMALLINT NOT NULL,
    dept_code CHAR(3) NOT NULL,
    acct_desc LVARCHAR(2000),
    max_employees SMALLINT,
    current_employees SMALLINT,
    num_projects SMALLINT,
)
FRAGMENT BY EXPRESSION
acct_id < 100 IN dbspace1,
acct_id >= 100 AND acct_id < 200 IN dbspace2,
acct_id >= 200 AND acct_id < 300 IN dbspace3,
REMAINDER IN dbspace4;
```

Date Types

Oracle data types need to be replaced with one of the Informix equivalent data types, depending on the type and size of the column. For more information, refer to Appendix A, “Data types” on page 313, which summarizes the mapping from Oracle data types to corresponding IDS data types. The mapping is one to many and depends on the actual usage of the data.

In this section we discuss in detail the special types of data, such as large objects, ROWID, and user-defined data types.

Large objects

Informix incorporates two types of large objects. These are typically referred to as simple large objects and smart large objects, or *SLOBs*. Simple large objects consist of two data types, TEXT and BYTE, and have the theoretical size limit of 2^{31} bytes. SLOBs contain the character large object (CLOB) and binary large object (BLOB) data types. SLOB's may be up to 2^{42} bytes, or 4 terabytes, in size. Simple large objects are stored either in tables or blobspaces while SLOBs are stored in sbspaces.

Oracle 8.0.x introduced LOBs (large objects) consisting of CLOB and BLOB data types. Traditional Oracle MLSLABEL, RAW, and LONG RAW data types are also used for data that is not to be interpreted by the engine (not converted when moving data between different systems). Oracle CLOB, BLOB, MLSLABEL, RAW, and LONG RAW data types should be replaced with Informix BYTE, TEXT, CLOB, or BLOB, depending on the column size and usage. CLOB and BLOB data types should be used under the following conditions and considerations:

- ▶ The large object needs to be referenced by multiple sources.
- ▶ The object size exceeds 2 GB.
- ▶ Fragmentation is required.
- ▶ Byte level locking makes it possible to lock only a portion of the object.
- ▶ Logging can be specified at the object level.
- ▶ Updates are done in place or moved as needed.
- ▶ Data storage is required for an index referenced by a third party datablade.
- ▶ It is the ideal storage method for large user defined types (UDTs).

Be aware of the following restrictions for simple large objects:

- ▶ Data can only be inserted into a BYTE or TEXT column with the dbload or onload utilities, the DBACCESS load statement, BYTE or TEXT host variables, respectively, in ESQL/C, or by declaring a file in ESQL/COBOL.
- ▶ BYTE and TEXT columns cannot be inserted or updated with a literal, neither quoted text string nor number.
- ▶ BYTE or TEXT columns cannot be used in string operations, with aggregate functions, with the GROUP BY, ORDER BY, IN, LIKE, or MATCHES clauses.

ROWID Data Type

The implementation of ROWID differs in Oracle and Informix. The Oracle ROWID is both a pseudocolumn and a data type. Informix implements ROWID as a pseudo-column only. Oracle stores the ROWID column and ROWID data type column values in hexadecimal format. Informix stores the ROWID column values in INTEGER format. The Oracle and Informix engines maintain the values of the ROWID pseudocolumn. But, the Oracle engine does not maintain the values of other columns of type ROWID.

In Informix, the term ROWID refers to an integer that defines the physical location of a row. Informix assigns rows in a non-fragmented table a unique ROWID, which allows applications access to a particular row in a table. Rows in fragmented tables, in contrast, are not assigned a ROWID. To access data by ROWID in a fragmented table, a ROWID column must be explicitly created as is described in the *Informix Guide to SQL: Reference*, G229-6374. If applications attempt to reference a ROWID in a fragmented table that does not contain a ROWID that was explicitly created, Informix displays an appropriate error message and execution of the application is halted.

When used as an Oracle data type for a column, it is not guaranteed that the values are valid ROWIDs. For example, if the ROWID is used as a data type on a column, such as a column in a child table named `parent_rowid` containing the ROWID of the parent, then the values are maintained by the application, outside of the engine. The application must read the ROWID of the parent and insert it into the parent's corresponding child table rows. This is usually done so that the two tables can be joined on the parent's ROWID and the child's `parent_rowid` column. This functionality can be duplicated in Informix by using primary and foreign keys. Informix recommends that primary keys be used as a method of access in applications rather than ROWIDs, because primary keys are defined in the ANSI specification of SQL, and using them to access data makes applications more portable.

User-defined data types

Both Oracle and Informix Dynamic Server allow the user to create and define data types. Oracle refers to these as abstract data types while Informix refers to these as user-defined types. In Oracle, all user-defined types are referred to as abstract data types while Informix may refer to user-defined types as COLLECTION types, ROW types, DISTINCT types and OPAQUE types. Currently, the Informix ROW type is closest in form and function to the Oracle abstract data type. An example of the syntax for creating both an Oracle abstract data type and an Informix ROW type is shown in Example 6-3 on page 122.

Example 6-3 User-defined types

Oracle abstract data type:

```
CREATE TYPE name_type AS OBJECT
(
  first_name VARCHAR2(25),
  middle_initial CHAR(1),
  last_name VARCHAR2(30)
);
```

Informix row type:

```
CREATE ROW TYPE name_type
(
  first_name VARCHAR(25),
  middle_initial CHAR(1),
  last_name VARCHAR(30)
);
```

Both Oracle and Informix support the use of user-defined types for table definitions or column definitions within a table.

Building on Example 6-3, the following create table examples show a table definition and column definition within a table, based on user-defined types:

Oracle table creation:

```
CREATE TABLE name OF name_type;
```

Informix table creation:

```
CREATE TABLE name OF type name_type;
```

Oracle column definition:

```
CREATE TABLE employee
(empno NUMBER(9),
name name_type);
```

Informix column definition:

```
CREATE TABLE employee
(empno INTEGER,
name name_type);
```

For more information about User Defined Types, see the *Informix Guide to SQL: Syntax*, G229-6375.

Temporary tables

Informix and Oracle implemented the concept of temporary tables differently. Oracle implements temporary tables as a work area during batch data loads, whereas Informix implements temporary tables just like regular database tables. These temporary tables can be created, manipulated, and dropped only within a session. Once the session ends, any remaining temporary tables created by that session are automatically dropped.

Using WITH NO LOG option

When creating temporary tables in Informix, use the WITH NO LOG option to reduce the overhead of transaction logging. If you specify WITH NO LOG, operations on the temporary table are not included in the transaction-log operations. The WITH NO LOG option is required on all temporary tables that you create in temporary dbspaces. In Informix you can also choose to create temporary table in the dbspace of your choice.

Differences between temporary and permanent tables

Permanent tables differ from temporary tables in the following ways:

- ▶ They have fewer types of constraints available.
- ▶ They have fewer options that you can specify.
- ▶ They are not visible to other users or sessions.
- ▶ They do not appear in the system catalog tables.
- ▶ They are not preserved.

Duration of temporary tables in Informix

The duration of a temporary table depends on whether or not it is logged. A logged temporary table exists until one of the following situations occurs:

- ▶ The application disconnects.
- ▶ A DROP TABLE statement is issued on the temporary table.
- ▶ The database is closed.

A non-logging temporary table exists until one of the following events occurs:

- ▶ The application disconnects.
- ▶ A DROP TABLE statement is issued on the temporary table.

Example 6-4 on page 124 shows how an Oracle temporary table is typically created using ON COMMIT PRESERVE ROWS, which tells the system to delete rows in the temporary table at the end of the session. You can also see how an Informix temporary table is created in the same example.

Example 6-4 Oracle Temporary Table creation

Oracle Temporary Table Definition:

```
CREATE GLOBAL TEMPORARY TABLE my_employee_temp_table
(
    first_name varchar(40),
    last_name varchar(40)
) ON COMMIT PRESERVE ROWS;
```

Informix Temporary Table Definition:

```
CREATE TEMPORARY TABLE my_employee_temp_table
(
    first_name varchar(40),
    last_name varchar(40)
) WITH NO LOG;
```

Oracle DUAL table

Oracle's DUAL table is a system table with only one row containing one column named DUMMY. This table is used to complete the syntax of an SQL statement that does not involve an existing table, such as the following examples:

- ▶ A SELECT statement which assigns some constant value (which can only be derived in an SQL statement) to a host variable:

```
SELECT TO_CHAR(SYSDATE) INTO :date_var FROM DUAL;
```

- ▶ A SELECT statement which assigns a value of another variable to a variable:

```
SELECT :source_var INTO :dest_var FROM DUAL;
```

- ▶ A SELECT statement which unions application host variable values with another SELECT statement to return a result containing a row of application variable values:

```
SELECT col1, col2, col3 FROM table_name
UNION
SELECT :host_var1, :host_var2, :host_var3 FROM DUAL;
```

How you convert the Oracle DUAL table functionality to Informix depends on the way it is used. Converting statements that just assign constants is as simple as replacing the DUAL statement with an Informix assignment statement. To convert more complex DUAL statements, a temporary table named DUAL should be created with columns whose types depend on how they are used in the DUAL statement. Before selecting values from this table, as in the case of the previous UNION example, values must be inserted in the table first and then selected from it. If multiple UNION statements use DUAL, multiple rows must be inserted in the

table and then selected. The temporary table should be either dropped or emptied for future use, immediately after the DUAL processing.

If references to DUAL are widespread throughout the application, and minimizing the number of code changes is the priority, a dummy table can be created to mimic the behavior of simple SELECT INTO statements used to assign values to variables. Example 6-5 shows how to create an Oracle Dual table in Informix.

Example 6-5 Informix Dual table creation

```
CREATE TABLE dual (dummy VARCHAR(1));
INSERT INTO dual VALUES('X');
```

Table 6-2 summarizes a comparison Oracle and Informix tables.

Table 6-2 Mapping of Oracle and Informix tables

Oracle	Informix	Observation
Heap Organized	Regular table	A heap organized table in Oracle is a table with rows stored in no particular order.
Temporary	Temporary	
Index-Organized (IOT)	None	
External	Informix Virtual Table Interface	Read Informix Virtual Table Interface Programmer's Guide for more information.
Clustered	None	
Partitioned	Fragmented	
Nested	Named Row Type	
Index clustered table	Informix Index Cluster	Use TO CLUSTER option with CREATE / ALTER INDEX statement
Hash cluster table	None	

Both Oracle and Informix use indexes for query performance optimization. However, there are differences in the implementation of indexes between Oracle and Informix.

Informix composite indexes

A composite index can have up to 16 key parts that are columns, or up to 341 key parts that are values returned by a UDR. This limit is language-dependent, and applies to UDRs written in SPL or Java. Functional indexes based on C language UDRs can have up to 102 key parts. A composite index can have any of the following items as an index key:

- ▶ One or more columns
- ▶ One or more values that is returned by a user-defined function (referred to as a functional index)
- ▶ A combination of columns and user-defined functions

Informix maximum key size

The total widths of all indexed columns in a single CREATE INDEX statement have a limitation based upon the page size of the dbspace in which the index resides. However, this limitation does not apply to functional indexes.

An enhancement, called configurable page sizes for dbspaces, was introduced from Informix Dynamic Server Version 10.00. Prior to this, the page size of dbspaces were restricted to that of the operating system and could not be changed, thus IDS was limited to 390 byte indexes. With page sizes now configurable from 2 K to 16 K, wider indexes are supported.

Referring to Table 6-3, 16 K page sizes raises the limit on index width to over 3000 bytes. Lifting the 390 byte limit also permits LVARCHAR columns larger than 387 bytes to be included in an index definition.

Table 6-3 Page sizes for dbspaces

Page Size	Maximum Key Size
2 kilobytes	387 bytes
4 kilobytes	796 bytes
8 kilobytes	1615 bytes
12 kilobytes	2435 bytes
16 kilobytes	3245 bytes

In addition to aiding the porting process, wider indexes have satisfied requirements for Unicode data, and have provided performance gains by reducing B-Tree index traversal costs. This is because indexing can be built with more items on an index page and produces fewer levels.

Index fragmentation

Both Oracle and Informix support index fragmentation (partitioning). Oracle indexes that are partitioned in the same manner as the table they are indexing are referred to as local indexes. Example 6-6 shows how a Local Partitioned Index is created in Oracle based on the oracle partitioned table created in Example 6-2 on page 119.

Example 6-6 Oracle Local Partition Index

```
CREATE TABLE accounts
(
    acct_id NUMBER(3) NOT NULL,
    dept_code CHAR(3) NOT NULL,
    acct_desc VARCHAR2(2000),
    max_employees NUMBER(3),
    current_employees NUMBER(3),
    num_projects NUMBER(1)
)
PARTITIONING BY RANGE (acct_id)
(PARTITION part1 VALUES LESS THAN (100)
    TABLESPACE part1_tbs,
PARTITION part2 VALUES LESS THAN (200)
    TABLESPACE part2_tbs,
PARTITION part3 VALUES LESS THAN (300)
    TABLESPACE part3_tbs,
PARTITION part4 VALUES LESS THAN (MAXVALUE)
    TABLESPACE part4_tbs);

CREATE INDEX idx_account_local ON accounts (acct_id)
LOCAL
PARTITION BY RANGE(acct_id)
(PARTITION pi100 values less than(100) TABLESPACE part_ind_100,
PARTITION pi200 values less than(200) TABLESPACE part_ind_200,
PARTITION pi300 values less than(300) TABLESPACE part_ind_300,
PARTITION pi400 values less than (MAXVALUE) TABLESPACE part_ind_400);
```

In Informix, local indexes are known as attached indexes. The database server fragments the attached index according to the same distribution scheme as the table by using the same rule for index keys as for table data. As a result, attached indexes have the following physical characteristics:

- ▶ The number of index fragments is the same as the number of data fragments.
- ▶ Each attached index fragment resides in the same dbspace as the corresponding table data, but in a separate tblspace.
- ▶ An attached index or an index on a non-fragmented table uses 4 bytes for the row pointer for each index entry.

Unlike Oracle, you do not have to mention the fragmentation strategy again when creating the attached index. It just follows the same strategy used when creating the fragmented table. You just have to create an Index with a simple create index statement, such as that shown in Example 6-7. For additional information, refer to the *Informix Dynamic Server Performance Guide*, G229-6385.

Example 6-7 Informix Attached Index

```
CREATE TABLE accounts(  
    acct_id SMALLINT NOT NULL,  
    dept_code CHAR(3) NOT NULL,  
    acct_desc LVARCHAR(2000),  
    max_employees SMALLINT,  
    current_employees SMALLINT,  
    num_projects SMALLINT  
)  
FRAGMENT BY EXPRESSION  
acct_id < 100 IN dbspace1,  
acct_id >= 100 AND acct_id < 200 IN dbspace2,  
acct_id >= 200 AND acct_id < 300 IN dbspace3,  
REMAINDER IN dbspace4;
```

```
CREATE INDEX idx_account_local ON accounts (acct_id);
```

In Oracle, you can also have Global Partitioned Indexes which are flexible and there is no dependency on how the table is partitioned. Example 6-8 shows how to create an Oracle Global Partition based on the Oracle Partition Table created on Example 6-2 on page 119.

Example 6-8 Oracle global partition index

```
CREATE INDEX idx_account_local ON accounts (acct_id)  
GLOBAL  
PARTITION BY RANGE(acct_id)  
(PARTITION pi100 values less than(100) TABLESPACE part_ind_100,  
PARTITION pi100 values less than(MAXVALUE) TABLESPACE part_ind_400);
```

Informix refers to these types of indexes as detached fragmented indexes. However, Informix detached indexes may also refer to indexes that do not follow the same fragmentation scheme as the table they are indexing. Example 6-9 on page 128 shows how to create the corresponding Oracle global partition index shown in Example 6-8 as an Informix detached fragmented index.

Example 6-9 Informix detached index

```
CREATE INDEX idx_account_local ON accounts (acct_id)
```

```
FRAGMENT BY EXPRESSION
acct_id < 100 IN dbspace1,
REMAINDER IN dbspace4;
```

Oracle bitmap indexes

The Oracle bitmap index is not available in Informix Dynamic Server. This type of index is aimed at data warehousing and is suitable for an index where there are few key values (low cardinality). For example, as with gender or state.

Note: Oracle bitmap indexes are available in the Informix Extended Parallel Server.

Oracle function-based indexes

A function-based index computes the values of the functions or expressions and stores them in the index as key values. Informix functional indexes are the same as the Oracle function-based indexes.

General index information

The Informix CLUSTER option should be used when appropriate. This option orders the data within the table in the order the index requires. It is important to note that the CLUSTER index is not maintained over time. The primary factor determining the frequency of CLUSTER index rebuilds is the amount of DML performed against the table. Frequent INSERTS, UPDATES, and DELETES make it necessary to closely monitor CLUSTER indexes. There is only one clustered index per table because a table data can only be stored in one order.

Oracle INITRANS and PCTFREE functionality is similar to the Informix index FILLFACTOR option of the CREATE INDEX statement or the FILLFACTOR configuration parameter in the ONCONFIG file. In either case, the functionality is handled by the Informix engine once it is set, and can be removed from the application. FILLFACTOR specifies the degree of index-page compactness. A low value provides room for growth in the index, and a high value compacts the index. If an index is fully compacted (100 percent), any new inserts result in splitting nodes. The setting on the CREATE INDEX statement overrides the ONCONFIG file value. The FILLFACTOR default value for both the CREATE INDEX statement as well as the ONCONFIG is 90.

6.1.3 Views

A view is a virtual table based on a specified SELECT statement. Views are used to restrict the access to the contents of the base tables. The following are the characteristics of a view in IDS:

- ▶ To create a view, the base table must exist and you must have the SELECT privilege on all columns from which the view is derived.
- ▶ Deletes, inserts, and updates can be performed through the view.
- ▶ Because a view is not really a table, it cannot be indexed and it cannot be the object of such statements as ALTER TABLE and RENAME TABLE.
- ▶ You cannot rename the columns of a view with RENAME COLUMN. To change anything about the definition of a view, drop the view and recreate it.
- ▶ The view reflects changes to the underlying tables with one exception. If a SELECT * specification defines the view, the view has only the columns that existed in the underlying tables when the view was defined by a CREATE VIEW. Any new columns that are subsequently added to the underlying tables with the ALTER TABLE statement do not appear in the view.
- ▶ The SELECT statement on which a view is based cannot contain INTO TEMP and ORDER BY clauses.

The Oracle CREATE VIEW statement contains three options:

- ▶ FORCE creates the view regardless of whether the view's base tables exist or the owner of the schema containing the view has privileges on them.
- ▶ NO FORCE creates the view only if the base tables exist and the owner of the schema containing the view has privileges on them.
- ▶ WITH READ ONLY specifies that no deletes, inserts, or updates can be performed through the view.

These options do not exist with create view statement in IDS, so they must be removed from the Informix CREATE VIEW statements. Example 6-10 shows a view creation statement in Oracle and Informix.

Example 6-10 View in Oracle and Informix

```
CREATE VIEW employees_details
AS SELECT emp_id, first_name, last_name
FROM employees
WITH READ ONLY;
```

```
CREATE VIEW employees_details
AS SELECT emp_id, first_name, last_name
FROM employees;
```

6.1.4 Sequences

An Oracle sequence is a database object from which multiple users, or transactions, may generate unique integers. For example, sequences can be

used automatically to generate primary key values. When a sequence object is queried, the sequence number is incremented and passed to the query, independent of the transaction committing or rolling back. If two applications increment the same sequence, the sequence numbers each application acquires may not be sequential because sequence numbers are being generated by the other application.

One user, or transaction, can never acquire the sequence number generated by another user, or transaction. Once a user or transaction generates a sequence value, that value will never be generated and passed to another user or transaction.

In Oracle, MAXVALUE has 28 or less digits and it must be greater than MINVALUE.

The IDS implementation of sequence objects is identical to that of Oracle. Issues regarding syntax compatibility should be negligible with the exception of some keywords which have no effect on the behavior of the sequence. We have shown the create sequence statement for Oracle and Informix in Example 6-11.

Informix supports DML statements (CREATE SEQUENCE, ALTER SEQUENCE, RENAME SEQUENCE, DROP SEQUENCE) for sequence objects that multiple users can access concurrently to generate unique integers in the 8-byte integer range. GRANT and REVOKE statements support access privileges on sequence objects, and the CREATE SYNONYM and DROP SYNONYM statements can be used to reference synonyms for sequence objects in the local database. The operators, CURRVAL and NEXTVAL, can read or increment the value of an existing synonym with behavior the same as Oracle.

Example 6-11 Create Sequence in Oracle and Informix

Create Sequence Statement in Oracle and Informix:

```
CREATE SEQUENCE account_sequence
START WITH 1
MAXVALUE 99999999
INCREMENT BY 1
NOCACHE
NOCYCLE;
```

6.1.5 Synonyms

In Oracle, you can have public and private synonyms. All users in the database are able to see the public synonyms regardless the privileges granted to them and can be referred to it without the owner. On the other hand, private synonyms

can only be seen for its owner and for those ones that have been granted privilege, and must be referred using the schema owner.

By default Oracle creates private synonym, whereas Informix creates a public synonym. We have shown a number of Oracle synonyms in Example 6-12.

Example 6-12 Synonyms in Oracle

Private Synonym in Oracle:

```
CREATE SYNONYM acct FOR appl.account;
```

```
SELECT * from appl.acct;
```

Public Synonym in Oracle:

```
CREATE PUBLIC SYNONYM acct FOR appl.account;
```

```
SELECT * FROM acct;
```

In Example 6-13 we have shown a number of Informix synonyms.

Example 6-13 Synonyms in Informix

Private Synonym in Informix:

```
CREATE PRIVATE SYNONYM acct FOR appl.account;
```

```
SELECT * from appl.acct;
```

Public Synonym in Informix:

```
CREATE SYNONYM acct FOR appl.account;
```

```
SELECT * FROM acct;
```

6.1.6 Triggers

Oracle supports the use of multiple trigger events. As examples, insert, update, and delete within the same trigger. Informix only supports one trigger event per trigger. And, just as in Oracle, you can definite multiple insert, delete, update, and select triggers on the same table in Informix.

In this section, we demonstrate a high-level overview of the differences in trigger definitions for Oracle and Informix. Example 6-14 shows a simple Oracle trigger, which has set a value to a table column before inserting a row.

Example 6-14 Oracle Trigger

```
CREATE OR REPLACE TRIGGER up_price
AFTER UPDATE OF unit_price ON stock
REFERENCING OLD AS pre NEW AS post
FOR EACH ROW
WHEN(post.unit_price > pre.unit_price * 2)
BEGIN
INSERT INTO warn_tab VALUES(pre.stock_num, pre.manu_code,
pre.unit_price, post.unit_price, SYSDATE);
END;
```

Example 6-15 shows how to define the corresponding trigger in Informix.

Example 6-15 Informix trigger

```
DROP TRIGGER up_price;

CREATE TRIGGER up_price
AFTER UPDATE OF unit_price ON stock
REFERENCING OLD AS pre NEW AS post
FOR EACH ROW
WHEN(post.unit_price > pre.unit_price * 2)
(INsert INTO warn_tab VALUES(pre.stock_num, pre.manu_code,
pre.unit_price, post.unit_price, CURRENT));
```

6.1.7 DBLinks

Oracle uses database links (DBLINK) to connect to another Oracle database. Dblink is an object in one database (source) that defines the target database name, and the user and password to connect to the target database. The target database name has to be in the tnsnames.ora entry and the user has to created

in the target database. Once the DBLink is created, you can access objects on the target database from the database source. Example 6-16 shows how to create and use a DBLINK in Oracle.

Example 6-16 Oracle DBLink creation

```
CREATE DATABASE LINK dbtarget
  CONNECT TO targetuser IDENTIFIED BY targetpwd
  USING 'targetentry';

SELECT * FROM my_schema.my_table@dbtarget;
```

In Informix, there is no dblink but you can access any table or routine in an external database simply by qualifying the name of the database object (table, view, synonym, or routine). When the external database is on the same database server as the current database, you must qualify the object name with the database name and a colon.

In the following example, the query returns data from the contacts table in the salesdb database.

```
SELECT name, number FROM salesdb:contacts;
```

When the external database is on a remote database server, you must qualify the name of the database object with the database server name and the database name.

In the following example, the query returns data from the contacts table that is in the salesdb database on the remote database server, distantserver:

```
SELECT name, number FROM salesdb@distantserver:contacts;
```

In ANSI databases, the owner of the object is part of the object name: `ownername.objectname`. When both the current and external databases are ANSI databases, you must include the owner name unless you are the owner of the object.

The following SELECT statement shows a fully-qualified table name:

```
SELECT name, number FROM salesdb@aserver:ownername.contacts;
```

Important: When accessing the external database objects, the following factors must be in effect

- ▶ You must have the appropriate permissions on these objects.
- ▶ Both databases must be set to the same locale.

6.2 DML

Most relational database management systems (DBMS) are either fully compliant or conform to one of the SQL standards. Thus, many SQL statements and queries written for Oracle can be converted to IDS without modification. However, certain SQL syntax and semantic differences exist across DBMS types, depending on what standard is implemented and the level of conformance.

In this section we discuss some differences between Oracle and Informix with regards to how they handle DML statements.

6.2.1 SQL

Oracle displays labels in SQL statements that are Informix reserved words, such as UNITS, YEAR, MONTH, DAY, HOUR, and so on. These must be converted by using the Informix AS clause between the column name and the display label.

In addition to the Informix supported “not equal to” symbols “<>” and “!=”, Oracle supports the symbol “^=”. All occurrences of ^= must be replaced with one of the Informix supported symbols.

The ANSI standard for comments within SQL statements is to precede the comment with a double dash, --. Once the SQL editor recognizes a double dash, the rest of the text on the line is treated as a comment and ignored. For multi-line comments, a double dash must precede the comment on each line. Oracle also delimits comments within SQL statements with /* comment */. This is helpful for multi-line comments.

Informix allows curly brackets for multi-line processing. Therefore, /*...*/ must be replaced with either the ANSI standard -- or the Informix {...}. It should be noted that Informix optimizer directives support syntax with the directive contained within /*... */. This type of commenting can remain in stored procedure code. The best practice recommendation is to replace it.

6.2.2 Selects

Queries are executed based on how the optimizer determines the best path to the data. Oracle and Informix developed their own query optimizers, with their proprietary methods and rules. As such, a query that looks exactly the same in both environments may run differently in each, following a different path and executing faster or slower. Therefore, every query in the ported application should be tested for performance, regardless of whether a query had to be converted.

Furthermore, Oracle supports optimizer hints within a SELECT statement up to the Oracle 9i R2 version. They are only present on 10g and 11g to provide backwards compatibility during the migration (upgrade) to the query optimizer (known as the cost based optimizer, or CBO). On the other hand, prior to IDS 7.3 and IDS 9.2, Informix did not allow optimizer hints. They do now have optimizer directives.

Optimizer directives

Informix offers optimizer directives, which are similar to Oracle optimizer hints. This feature provides the query developer the flexibility to direct the optimizer to follow specific paths, rather than choosing a plan through its analysis.

A query is processed in two steps. First, it is optimized and compiled to produce a query plan. Second, it is executed to produce results. Optimizer directives is a method for influencing the choice of the optimizer in the creation of the plan.

Optimizer directives address the following key issues:

- ▶ Reduced time for correcting performance problems. Rewriting queries can be time consuming, and this allows you to have a quick way to alter a plan.
- ▶ Competitive pressure. Users accustomed to this function were looking for it in Informix.
- ▶ Product development tuning. This allows product development to alter plans dynamically rather than through code changes to evaluate the effect of different optimization techniques.

The syntax and behavior of the Informix implementation is compatible with Oracle optimizer hints, which eases the porting of applications from Oracle to Informix.

Directives are written as a comment whose first character is a '+' (plus sign), as shown in Example 6-17.

Example 6-17 Hint on select statement in Oracle

```
SELECT /*+ INDEX(employee emp_deptno_idx) */  
e.empno, e.fname, e.lname, e.salary, d.dept_name  
FROM employee e, department d WHERE e.dept_no = d.dept_no;
```

This example specifies that the emp_deptno_idx must be used on the employee table. The Informix equivalent of the same hint in Oracle is depicted in Example 6-18 on page 137.

Example 6-18 Hint on Select statement in IDS

```
SELECT /*+ INDEX(employee emp_deptno_idx) */  
e.empno, e.fname, e.lname, e.salary, d.dept_name  
FROM employee e, department d WHERE e.dept_no = d.dept_no;
```

Informix supports the notion of negative directives, whereas Oracle only supports direct hints. A directive that tells the optimizer what to avoid, rather than what to choose, is unique to Informix. The query result can be realized by writing a directive to avoid certain actions known to cause performance issues, but allow any new indexes or table attributes to be explored by the optimizer as they are added over the life of the table and query. This allows a DBA to continue to add indexes to tables and not have to rewrite directives

Additionally, Informix supports full recognition of directives with SET EXPLAIN output. Informix will highlight semantic and syntactic errors. Optimizer directives support control in the following areas of the optimization process:

- ▶ Access methods: Index versus scans
- ▶ Join Methods: Forcing hash joins, nested loop joins
- ▶ Join Order: Specify which order the tables are joined
- ▶ Goal: Specify first rows or all rows (response time versus throughput)

See the *IBM Informix Guide to SQL: Syntax*, G229-6375 or *IBM Informix Guide to SQL: Reference*, G229-6374, for more information.

External optimizer directives

External optimizer directives give the DBA the ability to specify query directives and save them in the database. These directives are applied automatically to subsequent instances of the same query.

The SAVE EXTERNAL DIRECTIVES statement associates one or more optimizer directives with a query, and stores a record of this association in the *sysdirectives* system catalog table, for subsequent use with queries that match the specified query string. The query string must be an exact match that includes case and white space positioning. This statement establishes an association between the list of optimizer directives and the text of a query, but it does not execute the specified query. Only the DBA or user Informix can execute SAVE EXTERNAL DIRECTIVES.

This associates AVOID_INDEX and FULL directives with the specified query, as depicted in Example 6-19 on page 138.

Example 6-19 Select in Informix

```
SAVE EXTERNAL DIRECTIVES
  /*+ AVOID_INDEX (table1 index1)*/, /*+ FULL(table1) */
ACTIVE FOR
  SELECT col1, col2 FROM table1, table2
  WHERE table1.col1 = table2.col1
```

These directives are applied automatically to subsequent instances of the same query. You must include one of the `ACTIVE`, `INACTIVE`, or `TEST ONLY` keyword options to enable, disable, or restrict the scope of external directives. When external directives are enabled and the `sysdirectives` system catalog table is not empty, the database server compares every query with the query text of every `ACTIVE` external directive, and for queries executed by the `DBA` or user `Informix`, with every `TEST ONLY` external directive.

External directives are ignored if the `EXT_DIRECTIVES` parameter is set to 0 in the `ONCONFIG` file. In addition, the client system can disable this feature for its current session by setting the `IFX_EXTDIRECTIVES` environment variable to 0. If an external directive has been applied to a query, output from the `SET EXPLAIN` statement indicates “EXTERNAL DIRECTIVES IN EFFECT” for that query. Any inline directive is ignored by the optimizer when the external directives are applied to a query that matches the `SELECT` statement.

Like inline optimizer directives that are embedded within a query, external directives can improve performance in some queries for which the default behavior of the query optimizer is not satisfactory. Unlike inline directives, external directives can be applied without revising or recompiling existing applications.

6.2.3 Pseudo-columns

Pseudo-columns are columns that exist in tables but are not displayed from a `SELECT * FROM table_name` query. Informix pseudo-columns are generally reserved for use by the engine. Eliminating the use of such columns increases application portability and eliminates the need for future modifications if the engine is ever changed to process the column differently or not at all.

LEVEL

In Oracle, for each row returned by a hierarchical query, the pseudo-column `LEVEL` returns 1 for a root node, 2 for a child of a root, 3 for a child whose parent is at level 2, and so on. A root node is the highest node within an inverted tree, and therefore has no parent. A child node is any non-root node, a parent node is any node that has children, and a leaf node is any node without children. To

define a hierarchical relationship in a query, you must use the **START WITH** and **CONNECT BY** clauses. Informix does not support the **LEVEL** clause.

ROWID

The Oracle **ROWID** pseudo-column returns the row address for each row in the database. **ROWID** values contain the following information to locate a row:

- ▶ Which data block in the data file
- ▶ Which row in the data block (first row is 0)
- ▶ Which data file (first file is 1)

Usually, a **ROWID** value uniquely identifies a row in the database. However, rows in different tables that are stored together in the same cluster can have the same **ROWID**. Values of the **ROWID** pseudo-column have the data type **ROWID**. The following list details several important uses of **ROWID** values:

- ▶ They are the fastest way to access a single row.
- ▶ They can show you how a table's rows are stored.
- ▶ They are unique identifiers for rows in a table.

A **ROWID** does not change during the lifetime of its row. However, you should not use **ROWID** as the primary key of the table. If a row is deleted and re-inserted, its **ROWID** may change, even when using the export and import utilities. If a row is deleted, Oracle may re-assign its **ROWID** to a new row inserted later. Although you can use the **ROWID** pseudo-column in the **SELECT** and **WHERE** clauses of a query, these pseudo-column values are not actually stored in the database. The value of a **ROWID** pseudo-column cannot be inserted, updated, or deleted. Example 6-20 shows the Oracle select of all rows that contain data for employees in department 20.

Example 6-20 Select ROWID in Oracle

```
SELECT ROWID, ename FROM emp WHERE deptno = 20
```

Table 6-4 shows the results of the select from Example 6-20.

Table 6-4 Employees in department 20

ROWID	ENAME
0000000F.0003.0002	SMITH
0000000F.0000.0002	JONES
0000000F.0007.0002	SCOTT
0000000F.000A.0002	ADAMS
0000000F.000C.0002	FORD

The equivalent pseudo-column in Informix is also known as ROWID, and although the behavior is similar, the data type is different from Oracle's. For more information about data types, see Appendix A, "Data types" on page 313.

ROWNUM

In Oracle, for each row returned by a query, the ROWNUM pseudo-column returns a number indicating the order in which Oracle selected the row from a table or set of joined rows. The first row selected has a ROWNUM of 1, the second has 2, and so on. ROWNUM can be used to limit the number of rows returned by a query, as shown in Example 6-21, which reads nine rows:

Example 6-21 Querying the first nine rows

```
SELECT * FROM accounts WHERE ROWNUM < 10;
```

You can also use ROWNUM to assign unique values to each row of a table, as in Example 6-22.

Example 6-22 Assigning values to each row using rownum

```
UPDATE accounts SET acct_id = ROWNUM
```

Oracle assigns a ROWNUM value to each row as it is retrieved, before the rows are sorted with an ORDER BY clause. Therefore, the final order of the rows is in ORDER BY order not in ROWNUM order. An ORDER BY clause does not affect the ROWNUM values except in the case where the ORDER BY clause causes Oracle to use an index. In that case, the ROWNUM values are set in the order the rows are retrieved using the index and hence in ORDER BY order. This order is different than if the rows were retrieved without the index. So in this case the ORDER BY can affect the ROWNUM values.

Conditions which test for ROWNUM values greater than zero are always false. In Example 6-23 the query returns no rows.

Example 6-23 ROWNUM values greater than zero are always false

```
SELECT * FROM employee WHERE ROWNUM > 1
```

The first row fetched is assigned a ROWNUM of 1, thus making the condition false. The second row to be fetched is now the first row and is also assigned a ROWNUM of 1 making the condition false. All rows subsequently fail to satisfy the condition, so no rows are returned.

Although there is no ROWNUM equivalent in Informix, there are solutions. Beginning with IDS 7.3 and in Informix Dynamic Server 2000, Informix provides a feature which allows the user to limit the results of a query to the first N rows. This was implemented for TPC-D compliance and to support rank queries.

Rank queries can be assembled by using FIRST N with the ORDER BY clause. Informix will return the top N rows according to some ordering criteria. Therefore the following Oracle SELECT utilizing ROWNUM in Example 6-24 can be ported to Informix Dynamic Server as shown in Example 6-25.

Example 6-24 Oracle SELECT utilizing ROWNUM

```
SELECT * FROM emp WHERE ROWNUM < 10
```

Example 6-25 SELECT ported to IDS

```
SELECT FIRST 9 * FROM emp
```

Informix does not support SELECT FIRST N in complex cases such as the following examples:

- ▶ Sub-queries (SELECT within a SELECT)
- ▶ View definitions (Cannot be used by a SELECT statement which references a view)
- ▶ UNION queries

The behavior of SELECT FIRST N depends on the presence of an ORDER BY clause. If no order by clause is specified, the first N rows returned may be in any order. Example 6-26 shows the SELECT FIRST N statement.

Example 6-26 Using SELECT FIRST

```
/* find 10 highest paid employees */  
SELECT FIRST 10 name, salary FROM emp ORDER BY salary;  
  
/* find 10 highest salary values */  
SELECT FIRST 10 DISTINCT salary FROM emp ORDER BY salary;
```

Informix Dynamic Server supports column names of *first*. Thus, without a positive integer following the word *first*, the token is parsed as a column name rather than a keyword. The value of N may be between 1 and (231-1).

If using ROWNUM as an assignment, (as in UPDATE accounts SET col1 = ROWNUM), the example shown in Example 6-27 on page 142 would have to be used.

```
int counter = 0;
EXEC SQL declare c1 cursor for
    SELECT col1, col2
    INTO :col1, :col2
    FROM table 1
    WHERE col1 = "XX";
EXEC SQL open c1;
while (SQLCODE == 0)
{
EXEC SQL fetch c1;
    counter++; /* Increment counter for each row fetched */
    if (counter >= 5)
    {
        break;
        /* logic... */
    } /* End if */
} /* End while */
```

6.2.4 Inserts

Informix does not support Oracle INSERT with SELECT statements utilizing a UNION clause (although a view with an UNION clause can be referenced, resulting in an UNION operation). Additionally, Informix does not support an INSERT statement with a VALUE clause with SELECT statements. The SELECT statement within a VALUE clause can be rewritten by omitting the VALUE clause and performing an INSERT INTO table_name SELECT ... FROM other_table_name.

6.2.5 Outer joins

Both Oracle (starting with Oracle 9i) and Informix support ANSI standard syntax for outer join specification. The syntax for left and right outer join using the plus sign (+) in the WHERE clause is only supported for backward compatibility however it is still in existence. The use of ANSI outer join syntax is recommended.

The Oracle outer join syntax + (placed after the column which has values not matching the corresponding column in the joined table) can be replaced with the Informix equivalent, OUTER. The OUTER keyword is placed within the SELECT statement before the name of the subservient table. Unlike Oracle, in Informix multiple outer joins per SELECT statement are allowed, grouped with parentheses.

Table 6-5 demonstrates how to map this Oracle syntax to the Informix equivalent for some simple examples.

Table 6-5 Mapping of join definition

Oracle	Informix
SELECT A.last_name, A.id,B.name FROM emp A, Customer B WHERE A.id (+) = B.sales_rep_id;	SELECT A.last_name,A.id,B.name FROM emp A RIGHT OUTER JOIN customer B ON A.id = B.sales_rep_id;
SELECT A.last_name, A.id,B.name FROM emp A, Customer B WHERE A.id = B.sales_rep_id (+);	SELECT A.last_name,A.id,B.name FROM emp A LEFT OUTER JOIN customer B ON A.id = B.sales_rep_id;
SELECT A.last_name, A.id,B.name FROM emp A, Customer B WHERE A.id (+) = B.sales_rep_id (+);	SELECT A.last_name,A.id,B.name FROM emp A FULL OUTER JOIN customer B ON A.id = B.sales_rep_id;

The MTK provides basic support for Oracle outer joins, with the following restrictions:

- ▶ Only the equality (=) operator is supported.
- ▶ The (+) operator cannot follow a complex expression; it can follow a column reference only.

In some cases, the MTK will not be able to convert complex outer join syntax. The following example shows how a complex SQL statement involving multiple outer joins can be mapped from Oracle to Informix syntax.

It is important to realize that in Oracle, outer joins are defined in the WHERE clause. But in Informix, they are defined in the FROM clause. Further, the outer join condition of the two tables must be specified in the ON clause, not in the WHERE clause. Example 6-28 on page 144 shows the Oracle outer join syntax.

Example 6-28 Oracle outer joins

```
SELECT
    t1.surname
FROM
    EXAMPLE_TABLE1 t1,
    EXAMPLE_TABLE2 t2,
    EXAMPLE_TABLE3 t3,
    EXAMPLE_TABLE4 t4
WHERE
    ((t1.emptype = 1) OR (t1.position = 'Manager'))
    AND (t1.empid = t2.empid(+))
    AND (t2.empid = t3.empid(+))
    AND (t2.sin = t3.sin(+))
    AND (t3.jobtype(+) = 'Full-Time')
    AND (t2.empid = t4.empid(+))
    AND (t2.sin = t4.sin(+))
ORDER BY
    t1.emptype, t2.other
```

Example 6-29 shows how to make the Informix conversion.

Example 6-29 Informix outer join conversion

```
SELECT
    t1.surname
FROM
    EXAMPLE_TABLE1 t1 LEFT OUTER JOIN
        EXAMPLE_TABLE2 t2 ON (t2.empid = t1.empid) LEFT OUTER JOIN
            EXAMPLE_TABLE3 t3 ON (t3.sin = t2.sin)
                AND (t3.empid = t2.empid)
                AND (t3.jobtype = 'Full-Time')
        LEFT OUTER JOIN
            EXAMPLE_TABLE4 t4 ON (t4.sin = t2.sin)
                AND (t4.empid = t2.empid)
WHERE
    ((t1.emptype = 1) OR (t1.position = 'Manager'))
ORDER BY
    t1.emptype, t2.other
```

6.2.6 Sorts

In Oracle sorts, NULL values are considered the highest value and are therefore ordered last in ascending sorts. Informix sorts with NULLS as the lowest value, ordering them first in ascending sorts. First, it must be determined whether this sorting difference will affect the outcome enough to warrant a work around. In most cases it will not. If it does, the NVL function can be used to replace the column value with the highest value if the column contains a NULL value so that it will sort last. Once the application receives the results, it must replace the high values with NULL if the column is going to be displayed, or manipulated.

Correlation names

Informix does not support correlation names with aliases on the main table for an UPDATE or DELETE statement. Such cases must be converted, as shown in Example 6-30.

Example 6-30 Correlation Names in Oracle and Informix

Oracle:

```
UPDATE customer C
SET zip_code = '92612'
WHERE zip_code IN
(SELECT zip_code FROM zip_table Z
WHERE C.create_date <= Z.effective_date)
```

Informix:

```
UPDATE customer
SET zip_code = '92612'
WHERE zip_code IN
(SELECT zip_code FROM zip_table Z
WHERE customer.create_date <= Z.effective_date)
```

-and-

Oracle:

```
DELETE customer C
WHERE order_date <=
(SELECT control_value FROM control_table T
WHERE T.control_field = 'archive_date'
AND C.state = T.state )
```

Informix:

```
DELETE customer
WHERE order_date <=
(SELECT control_value FROM control_table T
WHERE T.control_field = 'archive_date'
AND customer.state = T.state )
```

Note: Informix allows correlation names with aliases on tables other than the updated or deleted table.

6.2.7 Aliases

In Informix, if an alias is used in the FROM clause of a SELECT statement, it must also be used in the SELECT list and WHERE clauses. The original table name should be replaced with the alias wherever aliases and original names are used together within a SELECT statement in an Oracle application.

6.2.8 Truncate

Informix has implemented support of the TRUNCATE DML statement, as shown in the following example:

```
TRUNCATE [TABLE] table [ [ DROP | REUSE ] STORAGE ];
```

The TRUNCATE statement drops all the data for the table and keeps just the initial extents, as shown in the following example:

```
TRUNCATE my_table;
```

The following example drops all the data for the table but retains all the space allocated:

```
TRUNCATE my_table REUSE STORAGE;
```

Note: There is a minor syntactical difference between Oracle and IDS: IDS does not require the keyword TABLE. The permissions needed for TRUNCATE TABLE depend on the existence of DELETE triggers on the table.

For tables with DELETE triggers, ALTER permission on the table is required and RESOURCE or DBA privileges because the DELETE triggers will be bypassed. For tables without DELETE triggers, you need DELETE permission on the table and CONNECT privilege to the database.

IDS permits a TRUNCATE TABLE in a transaction, but it must be the last statement before COMMIT or ROLLBACK. Any other statement will generate an error. In an Informix non-ANSI database, TRUNCATE is run as a singleton statement. Outside any explicit transaction in MODE ANSI databases, COMMIT immediately.

6.2.9 Hierarchical queries

In Oracle, if a table contains hierarchical data, rows can be selected in a hierarchical order using the following clauses.

START WITH

The START WITH clause identifies the row(s) to be used as the root(s) of a hierarchical query.

This clause specifies a condition that the root(s) must satisfy. If this clause is omitted, Oracle uses all rows in the table as root rows. A START WITH condition can contain a subquery.

CONNECT BY

The CONNECT BY clause specifies the relationship between parent and child rows in a hierarchical query. This clause contains a condition that defines this relationship. The part of the condition containing the PRIOR operator must have one of the following forms:

- ▶ PRIOR expression comparison_operator expression
- ▶ expression comparison_operator PRIOR expression

IDS provides a node data type to model hierarchical relationships. The data type and the supported 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 to use the node data type.

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 string. Therefore, when you increment node 1.9, you get 1.10, and not the numerical increment value of 1.91.

See IBM Redbooks publication *Informix Dynamic Server 11: Advanced Functionality for Modern Business*, SG24-7465 for more information about how the node datablade can be used to migrate Oracle hierarchical data to node type data in IDS.

6.3 SPL and PL/SQL

Informix procedures and functions are considered to be user-defined routines (UDR). A UDR can be written in the Informix Stored Procedure Language (SPL) or an external language, such as C or Java. A procedure is a routine written that does not return a value. A function is a routine written that returns a single value, a value with a complex data type, or multiple values.

Informix SPL is an extension to Informix SQL and provides procedural flow control such as looping and branching. An SPL routine is a generic term that includes both SPL procedures and SPL functions, and you can use SQL and SPL statements to write an SPL routine. SPL statements can be used only inside the the following statements:

- ▶ CREATE PROCEDURE
- ▶ CREATE PROCEDURE FROM
- ▶ CREATE FUNCTION
- ▶ CREATE FUNCTION FROM

SPL routines are parsed, optimized, and stored in the system catalog tables in executable format.

Like Oracle, Informix stored procedures support input, output and input/output parameters and can be used in SQL statements wherever expressions are allowed.

The way stored procedures and functions are created differ between Oracle and Informix. Oracle allows a REPLACE clause in the CREATE statement to handle situations when the object already exists, but Informix does not support the REPLACE option. Instead, a DROP PROCEDURE or DROP TRIGGER statement must precede the CREATE PROCEDURE and CREATE TRIGGER statements to handle the situation where the object already exists.

Size limit

Informix stored procedures have a size limit of approximately 64 K. Oracle stored procedures greater than 64 K in size can be segmented into smaller Informix stored procedures communicating with each other as though they were one by passing and returning the necessary values.

Parameter limit

Informix has a limit of 341 parameters for each stored procedure. This limit also applies to UDRs written in the Java language. UDRs written in the C language can have no more than 102 parameters.

Packages

Oracle's user-defined database level functions such as stored functions, stored procedures, packages, and package bodies all need to be converted to Informix stored procedures using the Informix Stored Procedure Language. Oracle's stored functions and stored procedures are similar in concept to Informix's stored procedures. Oracle's packages and package bodies, however, have no Informix equivalent.

A package body contains a group of stored functions and stored procedures, and a package is a list of those stored functions and stored procedures.

All procedures written in an Oracle package body must be rewritten as separate Informix procedures. When rewriting package bodies into separate procedures, Oracle's variable declarations can be replaced at the package level (global) with Informix global variables. The Informix global variables must be defined in the first stored procedure of the logical group.

Exceptions

The method of handling exceptions is different between Informix and Oracle. Oracle has many predefined and user defined exception labels such as `CURSOR_ALREADY_OPEN`, `NO_DATA_FOUND`, `ZERO_DIVIDE`, and so on. The labels can be used in the logic to identify errors. Only one exception check is allowed per `BEGIN` and `END` block using the `EXCEPTION` construct, and it is normally placed just before the `END` statement.

Informix also supports predefined and user-defined exceptions, however, they are represented numerically, rather than with labels. In Informix, all the exceptions checked in the stored procedure control blocks, delimited with `BEGIN` and `END` statements, must be declared explicitly at the top of each control block with the Informix `EXCEPTION` construct. Therefore, Oracle procedure code must be restructured to manipulate these exceptions.

In Oracle, an exception can be raised globally and acted upon globally. In Informix, the scope of an exception mechanism is always restricted to the block in which it is located. This leads to the addition of multiple Informix exception handling mechanisms for each Oracle exception raised as well as a redesign of the overall stored procedure to a more blocked format.

Error handling

In Oracle, errors are processed as part of a function call return value. In Informix, errors are retrieved as a separate function call that has multiple structures that have to be created to retrieve error and status. The `SQLCA` structure is not fully available in Informix stored procedures. It is not possible to check the `SQLCODE` variable in the stored procedure body. Instead, the function `DBINFO` can be used to extract the value of two members of the `SQLCA` structure: `sqlca.sqlerrd1` and `sqlca.sqlerrd2`. This function should be used inside the `FOREACH` construct while the cursor is open. Once the cursor is closed, the `DBINFO` function cannot return proper values.

The `sqlca.sqlerrd1` option returns different values depending on the type of SQL statement. For `INSERT` statements, if the table contains a serial column, then the `sqlca.sqlerrd1` option will return the serial value of the last row inserted into the table. For `SELECT`, `DELETE`, and `UPDATE` statements the `sqlca.sqlerrd1` option

will return the number of rows processed by the query. In these cases, upon each pass through the FOREACH loop, sqlca.sqlerrd1 is the same value: the number of rows the FOREACH loop will process. The sqlca.sqlerrd1 value can still be interrogated during each pass through the FOREACH loop.

The sqlca.sqlerrd2 option returns the number of rows processed by a SELECT, INSERT, UPDATE, DELETE, or EXECUTE PROCEDURE statement. It should not be interrogated until after the cursor has finished processing. In other words, within the FOREACH loop, the sqlca.sqlerrd2 value can be moved to a variable which is then interrogated outside of the FOREACH loop.

Cursors

Unlike Informix, in Oracle, cursors can be global to a package. In Informix, Oracle's global cursors can be replaced with temporary tables. A temporary table can be created and populated in place of where the cursor is opened. The temporary table can then be processed the same way the cursor is processed. The temporary table should be dropped in place of where the cursor is closed.

Cursors are explicitly declared, opened, and fetched in Oracle stored procedures. In Informix, cursors do not need to be explicitly declared, opened and fetched in stored procedures. Instead, Oracle stored procedure cursors can be replaced with Informix's FOREACH construct. The Oracle cursor name should be used in the FOREACH statement, for example: FOREACH cursor_name SELECT ... END FOREACH.

To update a cursor row in Oracle, the cursor must be declared with the FOR UPDATE clause. In Informix, if the SELECT statement in the FOREACH construct is not a multi-table join, then each fetched row can be updated using the UPDATE <table_name> WHERE CURRENT OF <cursor_name> statement.

Informix does not support the SELECT FOR UPDATE statement in a stored procedure. Therefore, the SELECT FOR UPDATE clause cannot be used in the FOREACH construct.

Flow control

Oracle allows unconditional jump statements such as GOTO and EXIT <label>. The EXIT <label> statement is used to break a loop to anywhere within the procedure.

Therefore, if the loop is within another loop, flow can be directed to somewhere within the parent loop or above or below the parent loop. Informix supports only the EXIT flow control statement within a loop, and it performs differently than the Oracle implementation. The Informix EXIT statement causes control to be moved to the first statement after the loop. Therefore, the Oracle code must be restructured to take advantage of structured programming techniques.

Variable declaration and assignment

Oracle allows implicit variable definitions within a FOR loop. The variables are defined using an assignment operator and a column name. See Example 6-31.

Example 6-31 Variables defined using an assignment operator and a column name

```
out_customer_name := customer.name;
```

The variable `out_customer_name` is defined as the same type as the column name in the `customer` table. Informix requires that all variables be defined at the beginning of each stored procedure control block. See Example 6-32.

Example 6-32 Variables defined at the beginning of each stored procedure control block

```
DEFINE out_customer_name LIKE customer.name;  
DEFINE counter INTEGER;
```

Therefore, the Oracle implicit variable declaration must be replaced with the Informix explicit variable declaration before the control block.

Additionally, assignment operations will need to be ported. Oracle uses a colon and an equal sign (`:=`) for assignment while Informix uses the `LET` keyword in combination with an equal (`=`) sign. For example, `LET var = 1`. Therefore, you must modify assignment statements with the `LET` keyword and an equal sign.

Boolean

The Oracle `BOOLEAN` variable type can be replaced with the Informix `BOOLEAN` data type.

Binary data types

Binary data types are different between Oracle and Informix. In Oracle, the binary type is defined as a type `RAW` and is explicitly defined with a maximum length as part of the declaration. In Informix, the binary type is defined as `REFERENCES BYTE`, which deals only with pointers to `BLOBs`. It implicitly defines a maximum length as part of the function call in which the pointer to the `BLOB` and maximum length (as used in the function call) was passed as values. As a result the binary type cannot be assigned a binary value in a stored procedure nor can it be inserted into a table through the `SQL INSERT` statement in a stored procedure. These operations that initialize a binary type must be performed outside of stored procedures.

Dynamic SQL

Starting with IDS version 11.50 Informix also supports dynamic SQL in SPL. SQL statements can now be dynamically constructed and executed.

The following example shows is the syntax of dynamic SQL statement in IDS:

```
EXECUTE IMMEDIATE { SQL_quoted_string | Str_variable }
```

The example above can be explained as follows:

- ▶ SQL_quoted_string: A string containing a single SQL statement
- ▶ Str_variable: A character variable containing the SQL statement

We provide the sample code for using dynamic SQL in Informix SPL, in Example 6-33.

Example 6-33 Dynamic SQL in Informix SPL

```
CREATE PROCEDURE MYPROC()  
  RETURNING INT;  
  DEFINE A0 VARCHAR(30);  
  DEFINE A1 VARCHAR(5);  
  DEFINE A2 INT;  
  DEFINE A3 VARCHAR(60);  
  DEFINE A4 INT;  
  LET A0 = "INSERT INTO DYN_TAB VALUES (";  
  LET A1 = ")";  
  FOR A2 = 1 TO 100  
    LET A3 = A0 || A2 || A1;  
    EXECUTE IMMEDIATE A3 ;  
  END FOR;  
  SELECT COUNT(DISTINCT C1) INTO A4 FROM T1;  
  RETURN A4;  
END PROCEDURE;
```

Refer to the *Informix Guide to SQL: Syntax*, G229-6375, for more information about dynamic SQL.

Compiler

Unlike Oracle, the Informix compiler was developed with the concept that developers can create stored procedures independently of the database. Therefore, the Informix stored procedure compiler has a limited set of errors that it checks compared to Oracle's. For example, the Informix stored procedure compiler does not validate database object names such as table, column, and view names. Therefore, database object naming errors will occur at runtime, not during the compilation time. Additionally, the error messages generated from the

compiler are not specific. Identifying a problem in a stored procedure may require segmenting it and recompiling the segments to find it. This may impact the time necessary to unit test the stored procedures.

Macros

Oracle contains macros, sometimes referred to as language constructs, that can determine the status of an object. Informix does not support Oracle macros and so they must be removed from the code and the associated logic implemented with equivalent Informix logic, where applicable.

%ISOPEN and %NOT FOUND

The macro `cursor_name%ISOPEN` returns a Boolean value to indicate whether the cursor is open, and `cursor_name%NOTFOUND` indicates whether there are any more rows yet to be fetched from the cursor `cursor_name`. These macros, used in condition statements, are followed by the logic to handle the true and false conditions.

In the case of `ISOPEN`, this macro and its associated logic can just be omitted because the `FOREACH` loop processing does not explicitly open a cursor, and therefore the condition cannot be checked. In the case of `NOTFOUND`, this macro should be omitted and its associated logic converted to Informix logic within an Informix `FOREACH` loop.

%ROWTYPE

The Oracle macro prefix `%ROWTYPE` is used in the declarative section as a data type to declare variables, similar to the structure construct of C or `RECORD` of `INFORMIX`. The prefix can be either a table name or another structure name (and structure names ultimately refer back to a table name). This macro may be replaced with the Informix statement `LIKE table_name.*`. However, the construct `LIKE table_name.*` cannot be used in SPL.

Instead, the construct `DEFINE column1 LIKE table_name.column1; DEFINE column2 LIKE table_name.column2; and so on`, should be used to declare variables corresponding to all the columns of the table. Then operations on the whole or part of the structure should be replaced with operations on single SPL variables.

%TYPE

The Oracle macro prefix `%TYPE` is used in the declarative section as a data type to declare single variables. The prefix is either a `table_name.column_name` or a `structure_name.variable_name`. A `structure_name.variable_name` ultimately refers back to a `table_name.column_name`. This macro can be replaced with the Informix statement `LIKE table_name.column_name`.

6.4 Concurrency and transaction

Both concurrency control and locking are used to ensure the data integrity in any DBMS. The concurrency can be categorized into *read concurrency* and *update concurrency*.

6.4.1 Read concurrency

Oracle uses a feature called *read consistency*, which lets a query return a result based on the state of the data when the query starts, regardless of other processes such as update or delete on the same data while the query is running. This mechanism is called *Multi-Version Read Consistency* and is implemented by using *undo data* in the undo segments.

Informix isolation levels

Informix implements various isolation levels to support read concurrency.

Informix dirty read isolation

The simplest isolation level, ANSI read committed or Informix dirty read, amounts to virtually no isolation. When a program fetches a row, it places no locks, and it respects none. It simply copies rows from the database without regard to what other programs are doing.

A program always receives complete rows of data. Even under ANSI read uncommitted or Informix dirty read isolation, a program never sees a row in which some columns are updated and some are not. However, a program that uses ANSI read committed or Informix dirty read isolation sometimes reads updated rows before the updating program ends its transaction. If the updating program later rolls back its transaction, the reading program processes data that never really existed.

ANSI read committed or Informix dirty read is the most efficient isolation level. The reading program never waits and never makes another program wait. It is the preferred level in any of the following cases:

- ▶ All tables are static, so concurrent programs read but never modify data.
- ▶ The table is held in an exclusive lock.
- ▶ Only one program is using the table.

Informix committed read isolation

When a program requests the ANSI read committed or Informix committed read isolation level, the database server guarantees that it never returns a row that is not committed to the database. This action prevents reading data that is not committed and that is subsequently rolled back.

ANSI read committed or Informix committed read is implemented simply. Before it fetches a row, the database server tests to determine whether an updating process placed a lock on the row; if not, it returns the row. Because rows that have been updated (but that are not yet committed) have locks on them, this test ensures that the program does not read uncommitted data.

ANSI read committed or Informix committed read does not actually place a lock on the fetched row, so this isolation level is almost as efficient as ANSI read uncommitted or Informix dirty read. This isolation level is appropriate to use when each row of data is processed as an independent unit, without reference to other rows in the same or other tables.

Locking conflicts can occur in ANSI read committed or Informix committed read sessions. However, if the attempt to place the test lock is not successful because a concurrent session holds a shared lock on the row. To avoid waiting for concurrent transactions to release shared locks (by committing or rolling back), IDS supports the last committed option to the committed read isolation level. When this Last Committed option is in effect, a shared lock by another session causes the query to return the most recently committed version of the row.

The last committed feature can also be activated by setting the USELASTCOMMITTED configuration parameter to COMMITTED READ or to ALL, or by setting USELASTCOMMITTED session environment option in the SET ENVIRONMENT statement in the sysdbopen() procedure when the user connects to the database. For more information about the Last Committed option to the ANSI read committed or Informix committed read isolation levels, see the description of the SET ISOLATION statement in *IBM Informix Guide to SQL: Syntax*, G229-6375. For information about the USELASTCOMMITTED configuration parameter, see *IBM Informix Dynamic Server Administrator's Reference*, G229-6360.

Informix cursor stability isolation

When cursor stability is in effect, IDS places a lock on the latest row fetched. It places a shared lock for an ordinary cursor or a promotable lock for an update cursor. Only one row is locked at a time. That is, each time a row is fetched, the lock on the previous row is released (unless that row is updated, in which case the lock holds until the end of the transaction).

Because cursor stability locks only one row at a time, it restricts concurrency less than a table lock or database lock. Cursor stability ensures that a row does not change while the program examines it. Such row stability is important when the program updates some other table based on the data it reads from the row. Because of cursor stability, the program is assured that the update is based on current information and prevents the use of stale data.

Informix repeatable read isolation

Where ANSI serializable or ANSI repeatable read are required, a single isolation level is provided, called Informix repeatable read. This is logically equivalent to ANSI serializable. Because ANSI serializable is more restrictive than ANSI repeatable read, Informix repeatable read can be used when ANSI repeatable read is desired (although Informix repeatable read is more restrictive than is necessary in such contexts).

The repeatable read isolation level asks the database server to put a lock on every row the program examines and fetches. The locks that are placed are shareable for an ordinary cursor and promotable for an update cursor. The locks are placed individually as each row is examined. They are not released until the cursor closes or a transaction ends. Repeatable read allows a program that uses a scroll cursor to read selected rows more than once and to be sure that they are not modified or deleted between readings. (Programming with SQL describes scroll cursors.) No lower isolation level guarantees that rows still exist and are unchanged the second time they are read.

Repeatable read isolation places the largest number of locks and holds them the longest. Therefore, it is the level that reduces concurrency the most. If your program uses this level of isolation, think carefully about how many locks it places, how long they are held, and what the effect can be on other programs. In addition to the effect on concurrency, the large number of locks can cause issues. The database server records the number of locks by each program in a lock table. If the maximum number of locks is exceeded, the lock table fills up, and the database server cannot place a lock. In this case, an error code is returned. The person who administers an Informix database server system can monitor the lock table and tell you when it is heavily used.

Default isolation levels

In Informix, the default isolation level is established when the database is created. We list the Informix default isolation levels in Table 6-6.

Table 6-6 Informix Default Isolation Levels

Informix Name	ANSI Name	When is it used
Dirty Read	Read Uncommitted	Database without transaction logging
Committed Read	Read Committed	Databases with logging that are not ANSI compliant
Repeatable Read	Serializable	ANSI compliant databases

Setting the isolation level

The isolation level is set by the SET ISOLATION statement, as shown in Example 6-34.

Example 6-34 Setting the isolation level

```
SET ISOLATION TO dirty read;  
SET ISOLATION TO repeatable read;
```

You can also set the isolation level using SET TRANSACTION ISOLATION LEVEL statement as shown in Example 6-35.

Example 6-35 Set the isolation level using set transaction statement

```
SET TRANSACTION ISOLATION LEVEL TO REPEATABLE READ
```

The major difference between the SET TRANSACTION and SET ISOLATION statements is the behavior of the isolation levels within transactions. The SET TRANSACTION statement can be issued only once for a transaction. Any cursors opened during that transaction are guaranteed to have that isolation level. With the SET ISOLATION statement, after a transaction is started, you can change the isolation level more than once within the transaction.

Recommendation for the appropriate isolation level

Choosing the appropriate isolation level to use for a transaction is important. The isolation level not only influences how well the database supports concurrency, but it also affects the overall performance of the application containing the transaction. That is because the resources needed to acquire and free locks vary with each isolation level.

If readers not blocking writers is a requirement, then the Informix dirty read isolation level can be used. The readers will not block the writers with dirty read processing; however, the query result will contain any updates performed while the query was running. If read consistency is a requirement, then the Informix repeatable read isolation level can be used to lock all the resulting rows in a query from updates while the query is running. If both are required, in other words duplicate the Oracle read consistency functionality, then the next question is, why? It will most likely be due to performance, in which case the application must be analyzed to see where changes can be made in the database schema and access methods to eliminate the performance issue.

6.4.2 Update concurrency

When an update, such as an INSERT, DELETE, or UPDATE, statement occurs in the database, a lock is used to support the update concurrency. In Oracle, locks placed on the table elements can be on individual rows, or on pages of rows in the table. In Informix, the database instance imposes locks on objects such as row, page, and database as they are required. The default lock mode used in Oracle is row level locking.

Lock mode in Informix

In Informix, use the LOCK MODE options to specify the locking granularity of the table. LOCK MODE Options available in Informix are PAGE and ROW.

PAGE lock obtains and releases one lock on the entire page of rows. This is the default locking granularity. Page-level locking is especially useful when you know that the rows are grouped into pages in the same order that you are using to process all the rows. For example, if you are processing the contents of a table in the same order as its cluster index, page locking is appropriate.

ROW lock obtains and releases one lock per row. Row-level locking provides the highest level of concurrency. If you are using many rows at one time, however, the lock-management overhead can become significant. You might also exceed the maximum number of locks available, depending on the configuration of your database server, but IDS can support up to 18 million locks on 32-bit platforms, or 600 million locks on 64-bit platforms. Only tables with row-level locking can support the LAST COMMITTED isolation level feature.

Informix lock mode can be changed by using one of the following methods: (resolved in the following order of precedence)

- ▶ LOCK MODE specified using an attribute of the CREATE TABLE or ALTER TABLE command syntax.
- ▶ IFX_DEF_TABLE_LOCKMODE environment variable setting.
- ▶ DEF_TABLE_LOCKMODE parameter setting in the ONCONFIG file.

If the DEF_TABLE_LOCKMODE parameter cannot be found in the ONCONFIG file, it can be added to make the specification for every database within the instance. The Informix instance must be restarted for this parameter to take effect.

SET LOCK MODE command in Informix

Use the SET LOCK MODE statement to define how the database server handles a process that tries to access a locked row or table. The syntax is:

```
SET LOCK MODE TO [NO WAIT / WAIT seconds]
```

When NO WAIT is used, database server ends the operation immediately and returns an error code. This condition is the default.

When WAIT is used, database server suspends the process until the lock releases.

When WAIT seconds is used, database server suspends the process until the lock releases or until the waiting period (specified in number of seconds) ends. If the lock remains after the waiting period, the operation ends and an error code is returned.

6.5 Security

In this section we discuss security options.

6.5.1 User authentication

A database user is created with the create user command in Oracle, and Oracle stores the user ID and password information in the database. This database user is granted permissions to connect to the database and use resources in the database.

Informix uses operating system password authentication to authenticate the users connecting to the database. This technique requires an OS user ID and password for each user connecting to the database. The user ID and password are submitted by the user or application program and the Informix DBMS verifies the username and password using OS library functions. If the OS function determines that the user ID and / or password are not in the OS set of user IDs and passwords, then the connection is rejected with error 951 for the incorrect user ID or 952 for the incorrect password.

To create the users (with passwords) to connect to the database, use OS specific administrative tools (such as SMIT in AIX) or commands (such as useradd).

In Oracle, the OPS\$User_name is sometimes used to connect to the database where OPS\$User_name is a valid Oracle user and User_name is a valid operating system level user. In this case, the connection is made to the database using just the '/' character without the user id and password. The Informix technique of using the /etc/passwd file for security works fine for this type of connection.

6.5.2 Authorization

After creating the OS level users, you can grant the privileges to the users to connect to the database and use database resources. To achieve this, you can use a simple grant statement just as in Oracle, and is shown in Example 6-36.

Example 6-36 Creating OS user and granting presumptions to connect to the db in IDS

OS Command to create a user harry:

```
useradd -d /home/harry -p ids4orc1 harry
```

SQL Command to grant connect and resource premissions to user harry:

```
grant connect, resource to harry;
```

Role-based authority can help you manage the database permissions. With IDS (starting with V10.00), you can create a role and assign that as a default role for individual users (or to PUBLIC). A role is a work-task classification, such as payroll or manager. Each defined role has privileges on the database object granted to the role. You use the CREATE ROLE statement to define a role and GRANT DEFAULT ROLE to establish the user's initial setting when connecting to the database. A user's role can be changed after connecting by using the SET ROLE statement.

6.5.3 Column-level encryption

You can use column-level encryption to improve the confidentiality of your data. IDS has new built-in SQL scalar functions that provide methods for data in columns to be stored in an encrypted format. Use the ENCRYPT_AES or the ENCRYPT_TDES function to define encrypted data. Use the DECRYPT_BINARY() and DECRYPT_CHAR() functions to query encrypted data. When using the routines the encryption of data is under application control and the DBMS is not aware that data is encrypted. Informix support of column level encryption is analogous to the Oracle DBMS Obfuscation Tool Kit

Built-in ENCRYPT functions provide methods for encrypting and decrypting the following character data types or smart large object data types:

- ▶ CHAR
- ▶ NCHAR
- ▶ VARCHAR
- ▶ NVARCHAR
- ▶ LVARCHAR
- ▶ BLOB
- ▶ CLOB

Passwords and hints

Passwords are necessary when using the encryption functions. Only users who can provide a secret password can view, copy, or modify encrypted data. The password must be a minimum of 6 bytes and can be a maximum of 128 bytes. When you set an encryption password, you have the option of specifying a password hint. Hints can be up to 32 bytes of text. If you specify a hint, you can store the hint with the encrypted password or in another location. Passwords (and hints) to be used by default can be set for a session using the SET ENCRYPTION PASSWORD statement:

```
SET ENCRYPTION PASSWORD 'password' [ WITH HINT 'hint string' ];
```

The password and hint parameters can be optional when calling the encryption function. Explicit values override the session default values.

The password or hint can be a single word or several words. It should be a word or phrase that helps you to remember the password, but does not include the password. You can subsequently execute the built-in GETHINT function (with an encrypted value as its argument) to return the plain text of the hint. Calling the GETHINT function with an argument of encrypted data will return the hint (if any) from the encrypted data or an empty string (NULL) if no hint was provided. There is no privilege needed, anybody can get any hint at any time.

When you set a password, IDS transfers the password and any hint to 128-bit key that is used to encrypt the password and hint. Passwords and hints are not stored as plain text in any table of the system catalog. The key is a time-based random value per instance. The database server initializes the key when the server starts the key is destroyed when the database server shuts down.

Encrypting a column

Column data can be encrypted through the use of two functions, ENCRYPT_TDES and ENCRYPT_AES, corresponding to the two data encryption standards: the Triple Data Encryption Standard (Triple-DES) and Advanced Encryption Standard (AES) which describes the cipher algorithm used to protect data from unauthorized viewing. AES encryption (also known as Rijndael) uses a 128-bit key size and Triple-DES encryption uses two 56-bit keys for 112-bits overall. Both functions are variant functions and will return a different result every time it is used.

Example 6-37 on page 162 demonstrates how to use the encryption functions with a column that contains a social security number.

Example 6-37 Creating table with encrypted columns

```
CREATE TABLE emp (name CHAR (40), salary MONEY, ssn LVARCHAR(64));
  INSERT INTO emp
    VALUES ('Alice', 50000, ENCRYPT_AES('123-456-7890','one two three
123'));
or
CREATE TABLE emp (name CHAR(40), salary MONEY, ssn LVARCHAR(64));
  SET ENCRYPTION PASSWORD "one two three 123";
  INSERT INTO emp
    VALUES ('Alice', 50000, ENCRYPT_AES('123-456-7890'));
```

Querying an encrypted column

Encrypted data can be translated using the Informix DECRYPT_CHAR and DECRYPT_BINARY scalar functions. Encrypted data contains information about the encryption method and all the other data needed to decrypt it except the password. The encrypted data value is passed as the first argument and a password as the second, unless the SET ENCRYPTION statement has specified for this session with the same session password by which the first argument was encrypted. If the data is not encrypted, the function call will return an error.

The DECRYPT_CHAR function is used to invoke a decryption routine on character data types (as examples, CHAR, LVARCHAR, NCHAR, NVARCHAR and VARCHAR) while the DECRYPT_BINARY function accepts an encrypted large object of type BLOB or CLOB.

If the first argument to DECRYPT_CHAR or DECRYPT_BINARY is not an encrypted value, or if the second argument (or the default password specified by SET ENCRYPTION) is not the password that was used when the first argument was encrypted, IDS will issue an error and the call fails. If the call to the decryption function is successful, it returns the plain text version of the encrypted data argument. Example 6-38 demonstrates how to use the decrypt function to query encrypted data.

Example 6-38 Querying encrypted columns

```
SELECT name, salary, DECRYPT_CHAR(ssn, 'one two three 123')
FROM emp
WHERE DECRYPT_CHAR(ssn) = '123-456-7890';
or
SET ENCRYPTION PASSWORD 'one two three 123';
SELECT name, salary, DECRYPT_CHAR(ssn)
FROM emp
WHERE DECRYPT_CHAR(ssn) = '123-456-7890';
```

The first argument to `DECRYPT_BINARY` is expected to be an encrypted value of a large object data type. However, if it is called with a character data type `IDS` invokes the `DECRYPT_CHAR` function and attempts to decrypt the specified value.

Do not use decryption function to create a functional index on an encrypted column. This would store the decrypted values as plain text data in the database, and defeat the purpose of encryption.

Archived

Archived

Data conversion

Data conversion is a sensitive task in the porting project. You have to ensure that all data is moved to the target database, both correctly and in time.

In this chapter, we discuss the data conversion methods for deploying the data from Oracle to the IDS database server. The data can be transformed by the following methods:

- ▶ Using the IBM Migration Toolkit (MTK) generated scripts and data files
- ▶ Using the MTK to move data online
- ▶ Exporting the data manually from Oracle to flat files and importing or loading it into IDS
- ▶ Using operating system named pipes
- ▶ Using IBM InfoSphere™ Information Server

We give some hints for time planning of the data movement process.

7.1 Data conversion process

The data conversion process is quite complex. Before you define a porting method, you should perform some tests with only a portion of the data to verify that the chosen method works successfully for your database environment. Generally, it is a good idea that tests cover all potential cases. For these reasons, we recommend that you start early with the testing.

The tasks of the test phase are as follows:

- ▶ Calculate the source data size and calculate the required space for the files on disk.
- ▶ Select the tools and the conversion method.
- ▶ Test the conversion using the chosen method with only a small amount of data.

Before starting the load of the data into the production environment, it would be a good idea to set up a development environment for initiating a proof of concept scenario of the chosen data load strategy, and the estimated time based on the test for the load of partial data. This means after loading all the data into the development environment, a more detailed estimate for the amount of time the complete load needs to take can be done. Additionally, consistency checks can be applied to the data offline without affecting the production in terms of additional workload. Also, you can make sure that all kind of data types used in the current database infrastructure are tested and possible errors could be identified in a early stage of the migration.

With the result of the test, you should be able to perform the following tasks:

- ▶ Estimate the time for the complete data conversion process.
- ▶ Create a plan for the development environment conversion.
- ▶ Create a plan for the production environment conversion, using the information from the development environment conversion.
- ▶ Schedule the time for the data conversion.

The following factors influence the time and complexity of the process:

- ▶ Amount of data and data changes

The more data that you have to move, the more time you need. Consider the data changes as well as the time stamp conversions.

- ▶ System availability
You can run the data movement either when the production system is down or when the business process is running, by synchronizing source and target database. Depending on the strategy you choose, you will need more or less time.
- ▶ Hardware resources
Be aware that you need up to three times the disk space during the data movement for the following items:
 - The source data in Oracle
 - The unloaded data stored in the file system
 - The loaded data in the target IDS

7.2 Time planning

After testing the data movement and choosing the proper tool and strategy, create a detailed time plan. This time plan should include the following tasks:

- ▶ Depending on the data movement method:
 - Implementing or modifying scripts for data unload and load
 - Learning how to use the chosen data movement tools
- ▶ Data unload from Oracle
- ▶ Data load to Informix
- ▶ Backup of the target IDS database server
- ▶ Testing the loaded data in Informix for completeness and consistency
- ▶ Switching of the applications, including database interfaces
- ▶ Fallback process in case of incidents

7.3 Database schema conversion

There are two major tasks belonging to the migration of an existing database infrastructure. They are the schema conversion and the real data movement. In this section we give you a detailed overview how to successfully migrate an existing database schema.

The first step for successfully converting an existing database schema is the selection of the appropriate database objects you want to move. These are objects such as tables, indexes, constraints, triggers sequences, and stored procedures. Upon selecting any of these base objects, you have to also consider permissions, roles and synonyms. After you have extracted the DDL for the objects you have to apply all necessary changes in regards to syntax differences.

This is most likely an iteration in case you apply the changes manually. You have to consider factors such as space management, logging, and locking.

7.3.1 Database schema extraction and conversion with the MTK

The MTK utility should be your first choice to extract and convert the currently defined Oracle database schema for the database you want to move. There are a number of reasons why you should use the MTK doing this task.

The MTK provides you a complete environment for extracting the definitions for all, or a partial selection of, database objects from the source database. You can easily choose only the objects for one defined schema, or only for a certain set of objects. This fits with the requirements for setting up a test strategy in view of time, disk space, and feasibility of the data movement for the most critical database objects.

Dynamically changing options, such as for the target storage environment and the data type mapping, can be used for performance testing and identification of additional needs for extracting the data in the expected format. There is no need to apply the necessary changes manually to the target database creation scripts.

7.3.2 Database schema extraction with Oracle database interfaces

The Oracle data dictionary is the base for extracting the DDL SQL statements for your database objects in case you want to generate and migrate the schema manually. There are multiple ways to generate a complete schema of your database infrastructure or at least for the necessary objects chosen to be moved. In the following sections we introduce some of them.

Using DBMS_METADATA functions for schema creation

The easiest way to generate the complete database schema is the usage of the DBMS_METADATA utility with the function GET_DDL. You can decide with a specification of an object type which DDL you want to generate. Example 7-1 on page 169 shows you the usage of the package in a select query. You can either use the dual table to extract the DDL for a given table, or use the dba_% views for extracting all the tables belonging to a provides schema or owner.

Example 7-1 Use DBMS_METADATA.GET_DDL to get the DDL for Oracle db objects

```
#extracting the DDL of a specific object

set long 10000
SELECT DBMS_METADATA.GET_DDL('TABLE','TABLENAME') from dual;

CREATE TABLE "INFORMIX"."STATE"
  ("CODE" CHAR(2),
   "SNAME" CHAR(15)
 USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
 TABLESPACE "USERS" ENABLE
 ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
 TABLESPACE "USERS"

#extracting the DDL of all objects owned by a specific schema
#DDL for the tables

SELECT DBMS_METADATA.GET_DDL('TABLE',table_name) from dba_tables
where owner='INFORMIX';
select dbms_metadata.get_ddl('INDEX',index_name) from dba_indexes
where owner='INFORMIX';
select dbms_metadata.get_ddl('REF_CONSTRAINT',constraint_name)
from dba_constraints where owner='INFORMIX';
```

Create the object DDL with the help of data pump

Another way that you can extract the database schema is to use the data pump export and import utilities. In the export phase you need to specify that you only want to export the metadata of the database, and a dump file is generated. This dump file can be used for generating a DDL script file for the database objects, which are specified at the time of the export. You are able to specify a schema, specific tables, or the entire database at the time of starting the export. In the example shown in Example 7-2 on page 170, we decided to unload the metadata of the Informix schema. The data pump import in the second step generates the SQL script specified with the SQLFILE option. Specify only the name of the file, but do not specify a path.

The benefit of using this way of the schema extraction is that all DDL statements are generated, including indexes, constraints, object types, and all permissions. There is no need to run a different SQL statement for each kind of objects, as in the usage of the DBMS_METADATA package.

Example 7-2 Using data dump export and import to generate a schema

```
$expdp informix/pwd SCHEMAS=INFORMIX CONTENT=METADATA_ONLY
$impdp informix/pwd DUMPFILE=expdat.dmp SQLFILE=SYS_EXPORT_SCHEMA_01
```

#Output :

```
-- new object type path: SCHEMA_EXPORT/TYPE/TYPE_SPEC
CREATE TYPE "INFORMIX"."ADDRESS_T"
  OID '63039F01F25A430FE040007F02002343' as object
  (
    address1 char(64),
    city char(32),
    state char(32),
    zipcode integer
  );

/

ALTER TYPE "INFORMIX"."ADDRESS_T"
  COMPILE SPECIFICATION
    PLSQL_OPTIMIZE_LEVEL= 2
    PLSQL_CODE_TYPE= INTERPRETED
    PLSQL_DEBUG= FALSE    PLSCOPE_SETTINGS= 'IDENTIFIERS:NONE'
```

Using the desc statement

There is a much easier way to extract column names and data types if you are only interested in the raw layout of the table. The desc statement executed in SQL*Plus gives you in a table layout the column names, null allowance, and the defined data type. The desc statement is a useful source later in the data movement process. You can use it to extract the table columns generating the selects for the unload files. The missing parts are certainly the constraints, the indexes and any permissions. In case you need to have all parts of the table DDL statements, the preferred approach is to use of the DBMS_METADATA package. This is depicted in Example 7-3 on page 171.

Example 7-3 Using desc for showing the columns and their data types in a table

```
SQL> desc informix.classes
```

Name	Null?	Type
CLASSID		NUMBER(38)
CLASS		NUMBER(38)
SUBJECT		CHAR(32)

7.3.3 Move the database schema to the target IDS database server

After you have extracted the DDL statements out of Oracle database server for the database objects to be moved, you have to apply some changes manually. The changes are depending on the type of the database object. In the following section we give you a short introduction for the major changes you need to apply to migrate database objects to the Informix database server.

This introduction is intended to show you the differences in the DDL with the special focus on the syntax. For that, we describe the DDL of an Oracle database object and show the corresponding syntax for a similar IDS database object. We give you some advice about which clauses you should be looking for. Because most of the statements can contain many different clauses we do not address everything in this chapter. However, we want to provide you with an idea of where to look.

For a detailed discussion about the SQL considerations during a migration, especially about a the comparison of DDL statements, refer to the following sections.

Tables

We want to start our introduction with the table objects. In particular, the definition of the table contains the columns and data type mappings, the storage definition, and constraints, such as the not null constraints and primary key constraint definition. Depending on which tool the schema has generated, the primary key definitions are included in the create table or are created by an alter table statement later on.

Example 7-4 on page 172 shows you the examples for the definition of a sample table object in the database server.

Example 7-4 Table objects in IDS and in Oracle

```
#Oracle database table DDL generated by impdp
CREATE TABLE "INFORMIX"."CUST_CALLS"
  ("CUSTOMER_NUM" NUMBER(38,0),
   "CALL_DTIME" TIMESTAMP (6),
   "USER_ID" CHAR(32) DEFAULT user,
   "CALL_CODE" CHAR(1),
   "CALL_DESCR" CHAR(240),
   "RES_DTIME" TIMESTAMP (6),
   "RES_DESCR" CHAR(240) NOT NULL ENABLE
 ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
 TABLESPACE "USERS" ;
```

```
#same Oracle object with primary key included created by DBMS_METADATA
CREATE TABLE "INFORMIX"."CUST_CALLS"
  ("CUSTOMER_NUM" NUMBER(38,0),
   "CALL_DTIME" TIMESTAMP (6),
   "USER_ID" CHAR(32) DEFAULT user,
   "CALL_CODE" CHAR(1),
   "CALL_DESCR" CHAR(240),
   "RES_DTIME" TIMESTAMP (6),
   "RES_DESCR" CHAR(240) NOT NULL ENABLE,
   PRIMARY KEY ("CUSTOMER_NUM", "CALL_DTIME")
 USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
 TABLESPACE "USERS" ENABLE,
 CONSTRAINT "A5" FOREIGN KEY ("CUSTOMER_NUM")
 REFERENCES "INFORMIX"."CUSTOMER" ("CUSTOMER_NUM") ENABLE,
 CONSTRAINT "A6" FOREIGN KEY ("CALL_CODE")
 REFERENCES "INFORMIX"."CALL_TYPE" ("CALL_CODE") ENABLE
 ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
 TABLESPACE "USERS"
```

```
#same Object in Informix
```

```
create table "informix".cust_calls
(
```

```
customer_num integer,  
call_dtime datetime year to minute,  
user_id char(32) default user,  
call_code char(1),  
call_descr char(240),  
res_dtime datetime year to minute,  
res_descr char(240) not null,  
primary key (customer_num,call_dtime)  
) in rootdbs extent size 200 next size 100 lock mode row;
```

The following list details the areas where changes have to be applied:

- ▶ The storage clause (table storage location and extent sizes)
- ▶ The lock mode in the IDS database server (row or page)
- ▶ The table type in the IDS database server (standard or raw. In the example, standard)
- ▶ Use of names like column names and table name, do not use quotation marks
- ▶ Remove the enable literal from the not null constraint specification

You need to have a special focus on the data type mapping in the create table SQL statement. There is one major factor influencing the appropriate data type on the target IDS database server. Check the contents of your source table columns in terms of the data value range for numeric data types, the current length for character-based and unstructured data types and the real intention for the usage of the date data type. We discuss the data type considerations in more detail later in this section.

For guidance in choosing the appropriate data type on IDS, look at Figure 7-1 on page 174. It shows the standard conversion table used by the MTK utility.

Source type	Target type
NCHAR(l)	NCHAR(l)
NVARCHAR2[1..255](l)	NVARCHAR(l)
NVARCHAR2[256..max](l)	NCHAR(l)
CHAR(l)	CHAR(l)
VARCHAR2[1..255](l)	VARCHAR(l)
VARCHAR2[256..max](l)	LVARCHAR(l)
ROWID	INTEGER
DATE	DATETIME[YEAR..FRACTION]
BFILE	BLOB
LONG RAW	BLOB
BLOB	BLOB
LONG	CLOB
CLOB	CLOB
NCLOB	BLOB
RAW(l)	BLOB
NUMBER	DOUBLE PRECISION
NUMBER[1..4](p,0)	SMALLINT
NUMBER[5..9](p,0)	INTEGER
NUMBER[10..18](p,0)	INT8
NUMBER[19..max](p,0)	DECIMAL(p,s)
DECIMAL(p,s)	DECIMAL(p,s)
INTEGER	DECIMAL(p,s)
SMALLINT	SMALLINT
FLOAT	DOUBLE PRECISION
FLOAT[1..max](l)	DOUBLE PRECISION
DOUBLE PRECISION	DOUBLE PRECISION
REAL	DOUBLE PRECISION
LONG VARCHAR	CLOB
TIMESTAMP(p)	DATETIME[YEAR..FRACTION]

Figure 7-1 Default Oracle to IDS data type mapping in the MTK

Indexes

Similar to our introduction about the DDL changes for tables, we now discuss the index objects. Looking at a simple index definition in Example 7-5, you can see the biggest difference regarding the syntax in the storage clause. Make sure that if you use the interfaces provided by Oracle, to pull out the schema for the database objects that you always remove the quotation marks. It is not necessary in those cases where you have a schema generated by an external tool because both database servers also understand the object name and object owner clause without quotations.

Example 7-5 Creating Indexes in Oracle

```
#Creating a simple index within Oracle
CREATE INDEX "INFORMIX"."ZIP_IX" ON "INFORMIX"."CUSTOMER" ("ZIPCODE")
  PCTFREE 10 INITRANS 2 MAXTRANS 255
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
```

```

TABLESPACE "USERS" PARALLEL 1 ;

#Same statement on Informix
create index "informix".zip_ix on "informix".customer (zipcode)
    using btree ;

#You could also see the following simple statements
#There are no differences in the use between Oracle and IDS

#Create a unique index
create unique index idx1 on "INFORMIX".customer ( customer_num )

#Composite with desc clause
create index idx1 on customer ( lname, fname desc )

```

There are quite a number of options that you can specify when creating an index. Be aware of the following major differences to consider:

- ▶ Remove the storage part from the Oracle definition
- ▶ For partitioned (fragmented) Indexes:
 - Check if the original the fragmentation strategy is supported
 - Change the syntax for the fragmentation strategy
- ▶ Remove bitmap indexes
- ▶ Remove NOSORT, COMPRESS, PARALLEL clauses
- ▶ Cluster indexes have a different meaning

Sequences

Sequences are a database server implementation of generating unique values served to different clients working in parallel on the same base tables. They can also be used to accomplish the request of unique primary keys. A sequence object is also created by SQL DDL statements. Looking at Example 7-6, there is no significant difference in the creation of a sequence in the two database servers.

Example 7-6 create a sequence object

```

#Oracle
CREATE SEQUENCE EX_SEQ MINVALUE 2000 MAXVALUE 100000000
INCREMENT BY 1 START WITH 8995 CACHE 100 NOCYCLE NOORDER
#Informix
CREATE SEQUENCE EX_SEQ
    INCREMENT BY 1 START WITH 8995 MAXVALUE 100000000
    MINVALUE 2000 NOCYCLE CACHE 100 NOORDER;

```

In addition to the syntax, there is a small restriction with IDS in terms of the range of the values for MAXVALUE. This is an integer value and has to be in the range of an Integer value. Specifying another value would lead to an SQL error and the object would not be created.

Constraints

We previously started the discussion about constraints in our look at the tables. You can define NOT NULL constraints, check constraints, referential key constraints, and primary key constraints. They can be specified in two different ways. You can include the constraints in the table definition, or you can use the alter table SQL statement to add the constraint after you already created the table. We have compiled the two different specifications of the constraint definition in Example 7-7.

Example 7-7 Different types of constraints in create table and alter table in Oracle Syntax

```
#Create a table with the constraints
CREATE TABLE "INFORMIX"."CUST_CALLS"
  ("CUSTOMER_NUM" NUMBER(38,0),
   "CALL_DTIME" TIMESTAMP (6),
   "USER_ID" CHAR(32) DEFAULT user,
   "CALL_CODE" CHAR(1),
   "CALL_DESCR" CHAR(240),
   "RES_DTIME" TIMESTAMP (6),
   "RES_DESCR" CHAR(240) NOT NULL ENABLE,
   PRIMARY KEY ("CUSTOMER_NUM", "CALL_DTIME")

#Creating Primary and Foreign key constraints with alter table
ALTER TABLE "INFORMIX"."ITEMS" ADD PRIMARY KEY ("ITEM_NUM",
"ORDER_NUM")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
  TABLESPACE "USERS" ENABLE;

ALTER TABLE "INFORMIX"."ORDERS" ADD CONSTRAINT "A1" FOREIGN KEY
("CUSTOMER_NUM")
  REFERENCES "INFORMIX"."CUSTOMER" ("CUSTOMER_NUM") ENABLE;

ALTER TABLE "INFORMIX"."CUST_CALLS" ADD CONSTRAINT "A95" CHECK
(CUSTOMER_NUM < 10000);
```

In Informix, the appropriate SQL looks similar to that in Example 7-8.

Example 7-8 Constraint definition in create and alter table in IDS

```
create table "informix".cust_calls
(
    customer_num integer,
    call_dtime datetime year to minute,
    user_id char(32)
        default user,
    call_code char(1),
    call_descr char(240),
    res_dtime datetime year to minute,
    res_descr char(240) not null,
    primary key (customer_num,call_dtime)
);

ALTER TABLE "informix". cust_calls ADD constraint
PRIMARY KEY ( customer_num, call_dtime ) constraint constrname

alter table "informix".orders add constraint (foreign key
(customer_num) references "informix".customer );

ALTER TABLE "informix". cust_calls ADD constraint
CHECK      ( customer_num < 1000      )
constraint chkconstr;
```

Looking at both constraint definition statements, you have to verify the following clauses:

- ▶ For the create table SQL statement, with respect to the constraints, remove the following information:
 - *enable* literal from the constraint
 - storage clause from the primary key
- ▶ For the alter table SQL statement, with respect to the constraints:
 - In case you want to specify the name, move the name to the end of the statement, followed the key word *constraint*
 - Remove the enable literal
 - Informix also allows constraints without names. So, you can also remove the name completely, and an internal name is automatically generated.
 - If you use the owner in quotations be aware it is case sensitive.

Grants, Revokes, and Roles

We now take a closer look at the security. You can grant and revoke permissions to and from a user or role, as shown in Example 7-9. The statements generated by the Oracle tools are similar to those for the Informix database server. You have to remove the quotation marks from the object name, as described in previous examples. Roles creation and granting permissions to roles are also considered in the Example 7-9. But they are also similar in both schemas.

Example 7-9 Grant and revoke permissions

```
#Oracle Syntax for Grant and Revoke
GRANT DELETE ON "INFORMIX"."CALL_TYPE" TO PUBLIC;
GRANT INDEX ON "INFORMIX"."CALL_TYPE" TO PUBLIC;
GRANT INSERT ON "INFORMIX"."CALL_TYPE" TO PUBLIC;
GRANT SELECT ON "INFORMIX"."CALL_TYPE" TO PUBLIC;
GRANT UPDATE ON "INFORMIX"."CALL_TYPE" TO PUBLIC;
GRANT SELECT ON "INFORMIX"."CALLTYPE" TO PUBLIC WITH GRANT OPTION;

REVOKE DELETE ON "INFORMIX"."CALL_TYPE" FROM PUBLIC;

CREATE ROLE ROL_TEST;
GRANT ROL_TEST TO USER;
DROP ROLE ROL_TEST

#Informix

GRANT DELETE ON "INFORMIX".CALL_TYPE TO PUBLIC;
GRANT INDEX ON "INFORMIX".CALL_TYPE TO PUBLIC;
GRANT INSERT ON "INFORMIX".CALL_TYPE TO PUBLIC;
GRANT SELECT ON "INFORMIX".CALL_TYPE TO PUBLIC;
GRANT UPDATE ON "INFORMIX".CALL_TYPE TO PUBLIC;
GRANT SELECT ON "INFORMIX".CALLTYPE TO PUBLIC WITH GRANT OPTION;

REVOKE DELETE ON "INFORMIX".CALL_TYPE FROM PUBLIC;

CREATE ROLE ROL_TEST;
GRANT ROL_TEST TO USER;
DROP ROLE ROL_TEST
```

Miscellaneous database objects

You can remove the creation of the user, directory, data link, and table space objects from the database schema if they are included. These Oracle database server objects are handled by the Informix server in a different way. They are either managed by the base operating system, as in the case of directory access

control, or by the onspaces utility for dbspace and chunk management that comes with the IDS distribution. Database access links have to be substituted in Informix by a specific syntax clause in the specification for remote objects in SQL statements.

For a further description of triggers, stored procedures, and views, refer to Chapter 6, “SQL considerations” on page 111. Because they are not essential for creating the target database schema and the loading and unloading of the data, we do not provide a comparison here. Additionally, a variety of definitions of these type of objects is much more comprehensive than we can completely cover in this book.

7.4 Data movement

After you have created all the necessary database objects in the target Informix database server, you can proceed to the next step, which is to move the data. Start working with a small subset of tables and try to restrict the number of rows for the unload. For testing purposes, load the data into a database on a test instance or a development system. From the results of the load you can determine a much better estimate of the time required for the load, and the required disk space. Be sure to calculate the disk space for the unload of the raw data, and on the target IDS database server.

The final data movement step has two parts. You first unload the data on the source database server, and then load it onto the target database. The unload itself includes the select of the necessary data for the move, and also the changes required due to the differences in the data type. This is necessary to enable the target IDS database server to successfully read the data without data loss.

7.4.1 Unloading the data in Oracle

The main principle of data movement is to export the Oracle data to flat files in a well-defined format. This format enables the target Informix database server to load the data properly in the next step. There are several ways to initiate the unload. You can use either procedures or the sqlplus command line processor. The only way to extract the data from the Oracle database is to perform a sequential scan of the database tables and execute a row projection to the target unload file. The other export utilities provided by Oracle, such as desc or data pump, generate an output which is in a binary format and which cannot be extracted.

How the content of an unload file appears

There are multiple formats in the output file that could be used by IDS to read the data. For example, you can use an output file where each column is separated by a defined delimiter. The end of the row is defined by the delimiter and a line feed. The major benefit of this form of output is in the saving of disk space. This is because the content of the columns do not always fill the entire column space, and there is empty space left in the columns. Having the delimiter enables all that empty space to be eliminated and not included in the unload file.

Another way to generate an unload file is the creation of a position oriented file. This means the output of each column starts at a certain position in the row, as defined by the size of the preceding columns. However, the disadvantage in using this form of output is that the file needs much more disk space than that occupied by the real data. This is especially the situation when you use large char or varchar2 columns with only a small percentage of the space used.

Generating delimited files

Typically the IDS prefers a format with delimited columns. This format allows all of the utilities that we introduce later, in 7.4.2, “Load the data into the target IDS database server” on page 197, to read and understand the data. A possible delimiter could be the “|” character. Depending on the content of the data, you are free to choose any other special character.

A possible simple select issued in sqlplus, and generating a delimited row format and the appropriate output, looks similar to Example 7-10.

Example 7-10 Simple unload of data with a full table scan and output row projection

```
SQL> select customer_num,'|',trim(lname),'|',trim(fname),'|'  
       from unload;
```

CUSTOMER_N	'	TRIM(LNAME)	'	'TRIM(FNAME)	'
1		Miller		Henry	
2		Fischer		Henry	

You can take this output as a start, but there is much room for improvement. As you may have noticed, we used trim to trim the output, but it was finally generated in that tabulator style. There was not much of a space saving. Worse yet, because we used the delimiter, the size was bigger than the real data size. A simple change of the query, as shown in Example 7-11 on page 181, shows that you can save a lot of space in the output file by changing the expression in the projection list of the query.

Example 7-11 Saving space using a well formatted projection expression

```
SQL> select trim(customer_num)||' '||trim(lname)||' '||trim(fname)||' '
from customer;
```

```
TRIM(CUSTOMER_NUM)||' '||TRIM(LNAME)||' '||TRIM(FNAME)||' '
-----
```

```
1|Miller|Henry|
2|Fischer|Henry|
3|Carpenter|Joanne|
```

Another problem with this output is the heading. There is no need for the target database server to have a description of how the source has created the output rows. So, we need to find a simple way to suppress it. Using the **set heading off** in the CLP removes that output. You will notice that there is another empty line between the top and the first row. Because we have to create a consistent output, this line also needs to be removed. Setting the pagesize to 0 will result in the removal of that line. The statements in sqlplus and the appropriate output of the query after the settings, are shown in Example 7-12.

Example 7-12 Remove the heading an empty lines from the top of a select output

```
SQL> set heading off
SQL> select trim(customer_num)||' '||trim(lname)||' '||trim(fname)||' '
from customer;
```

```
1|Miller|Henry|
2|Fischer|Henry|
3|Carpenter|Joanne|
```

```
SQL> set pagesize 0
SQL> select trim(customer_num)||' '||trim(lname)||' '||trim(fname)||' '
from customer;
1|Miller|Henry|
2|Fischer|Henry|
3|Carpenter|Joanne|
```

The only problem left is generating a consistent output that is readable by the target database server. The output of the select query can be redirected to stdout in a shell script. Otherwise you can spool the output or use the DBMS_OUTPUT package from PL/SQL. We tried to minimize the amount of data going to the output file. This can be done by the trim or rtrim built-in functions. But the content of the data in the rows is not consistently even. That is, they vary in size. Normally you specify the line size with the **set linesize <count>** statement. This should be used to maximize the output to avoid line breaks initiated by sqlplus.

But this leads to having trailing blanks after the last delimiter, which could be taken as another column during the load to the Informix database. Use `set trims ON` to cut the lines after the last delimiter.

Generating column size based output files

Column size-based output files are organized like a table. Each column has a defined space assigned regardless of how much space is actually required, based on the size of the real content of the column. The generation of this type of unload file is much easier when compared with the delimiter-based file previously described. Some simple output is depicted in Example 7-13.

Example 7-13 Column size-based output files

```
SQL> select customer_num, lname, fname from customer;
```

CUSTOMER_N	LNAME	FNAME
1	Miller	Henry
2	Fischer	Henry
3	Carpenter	Joanne

The representation of the column in the output depends on the data type. Similar to the delimited files, you have to cut the headings and all leading empty lines from the header. You can also use the `set headings off`, `set pagesize 0`, and `set trims on variables` to influence the output in the spool file or the redirected stdout output. There is one thing you should keep in mind. If you want to load these types of data files, put an additional byte in the control file of the load utility for each column except the last column, because a blank is used as a separator between the output columns.

Automation of the unloads

We have previously discussed the required format of the unload files and some of the data type layout transition requirements. At this stage you should be able to manually generate unload files. Now we want to give you an example of how to organize a batch unload of multiple tables within a sample shell script, and that includes creation and execution of an Oracle PL/SL based stored procedure in the source database.

Using a Shell script for unloading tables in a batch file

In this section, we show you the final compilation of settings for generating a delimited or column oriented unload file with a shell script. This is depicted in Example 7-14 on page 183. Given you have a defined user and a specific number of tables that you want to unload into separate files, this script could be useful. The script has two primary parts. First it reads the tables from a

parameter file and investigates the columns and their data types, by using the **desc** statement. Based on this output the final table scan is generated as a select and the output is stored in the output file for further use. You can either switch on the spool to save the output to a file, or process the stdout output. The stdout can also be redirected to a pipe with a subsequent compress or an attached parallel load. This means an unload and load on the target IDS database server side can be done in parallel.

Example 7-14 A shell script for unloading a subset of tables for a specific user

```
#!/bin/ksh
#####
# Shell script:  data_unload.sh
#
# Syntax:      data_unload.sh <table_list_file>
#
# Starting from an flat file containing a list of all the table,
# extracts data from Oracle for each table and writes data into
# a file named table_name.DAT, formatted in columns
#
#####

# Define the environment variables for the oracle user and password
export ORACLE_USR=informix
export ORACLE_PWD=password
#
# Start of main program

# Loop on all the tables listed in the input file
for i in `cat $1`
do
    # Define some environment variables for temporary files
    export OUTFILE=/tmp/$i.DAT
    rm -f $OUTFILE
    mkfifo $OUTFILE
    DSCFILE=$i.dsc
    SQLFILE=$i.sql
    VARFILE=$i.var
    ALLFILE=$i.all
    POSFILE=$i.pos
    #rm -f $OUTFILE
    rm -f $DSCFILE
    rm -f $SQLFILE

    # Extract the table description from Oracle catalog
```

```

sqlplus -s $ORACLE_USR/$ORACLE_PWD <<EOF >/dev/null 2>&1
clear columns
clear breaks
set pagesize 100
set newpage 1
set feedback off
spool $DSCFILE
desc $i
EOF

# Cut head and tail from the file containing the descriptions of the
tables
# Change also the NOT NULL clause in a blank string
# and cut the blanks in the first column
tail +3 $DSCFILE | sed 's/NOT NULL/ /; s/^ //' > $DSCFILE.tmp1
NL=`wc -l < $DSCFILE.tmp1`
NLM1=`expr $NL - 1`
head -$NLM1 $DSCFILE.tmp1 > $DSCFILE.tmp2
cp $DSCFILE.tmp2 $VARFILE

sed -e 's/ VARCHAR2(/ /' \
-e 's/ NUMBER(/ /' \
-e 's/ NUMBER/ 41/' \
-e 's/ INTEGER(/ /' \
-e 's/ INTEGER/ 41/' \
-e 's/ CHAR(/ /' \
-e 's/ CHAR/ 1/' \
-e 's/ RAW(/ /' \
-e 's/ VARCHAR(/ /' \
-e 's/)//' \
-e 's/\([0-9]*\) \, \([0-9 ]*\)\/\1' \
$DSCFILE.tmp2 > $DSCFILE.tmp3
mv $DSCFILE.tmp3 $DSCFILE
rm -f $DSCFILE.tmp*

# Prepare the heading of the query statement on the table
# by echoing the statements into the sql file
echo "clear columns" > $SQLFILE
echo "clear breaks" >> $SQLFILE
echo "set pagesize 0" >> $SQLFILE
echo "set linesize 10000" >> $SQLFILE
echo "set feedback off" >> $SQLFILE
echo "set heading off" >> $SQLFILE
echo "set space 0" >> $SQLFILE
echo "set newpage NONE" >> $SQLFILE

```

```

echo "set trim ON" >> $SQLFILE
echo "set trims ON" >> $SQLFILE
# echo "spool $OUTFILE" >> $SQLFILE
echo "select  '' " >> $SQLFILE

# Append to the query statement file the list of the table fields
# to obtain the column layout, using the desc.awk awk script
awk -f desc.awk $VARFILE >> $SQLFILE

# Append to the query statement file the "from" clause
# and the closing instructions
echo "from $i;" >> $SQLFILE
# echo "spool off" >> $SQLFILE
echo "quit" >> $SQLFILE

# Execute the query statement
sqlplus -s $ORACLE_USR/$ORACLE_PWD @$SQLFILE > $OUTFILE
done

```

To generate a delimited unload file, use the desc.awk script, as shown in Example 7-15. It calls the awk utility on UNIX to setup the projection list of the select statement which is referenced by the script.

Example 7-15 awk script for generating delimited output

```

#Use the following awk in a separate script desc.awk
BEGIN {}
{
  if ($2 == "DATE")
    print " ||rtrim(TO_CHAR("$1",'MM/DD/YYYY') ) || '|' "
  if (substr($2,1,9) == "TIMESTAMP")
    print " ||rtrim(TO_CHAR("$1",'YYYY-MM-DD HH24:MI:SSxFF') ) || '|' "
  if (substr($2,1,4) == "CHAR")
    print " ||rtrim("$1")||'|' "
  if (substr($2,1,8) == "VARCHAR2")
    print " ||rtrim("$1")||'|' "
  if (substr($2,1,6) == "NUMBER")
    print " ||rtrim("$1")||'|' "
}

```

A similar, but much easier, script can be used for creating the column-sized output file. That script is shown in Example 7-16.

Example 7-16 Generate a column size oriented output file separated by a blank

```
#Use the following awk in a separate script desc.awk
BEGIN {}
{
  if ($2 == "DATE")
    print " TO_CHAR('$1','MM/DD/YYYY') "
  if (substr($2,1,9) == "TIMESTAMP")
    print " TO_CHAR('$1','YYYY-MM-DD HH24:MI:SSxFF') "
  if (substr($2,1,4) == "CHAR")
    print " $1"
  if (substr($2,1,8) == "VARCHAR2")
    print " $1 "
  if (substr($2,1,6) == "NUMBER")
    print " $1"
}
```

In Example 7-17 we show the statement that was used to create the customer table. In addition, the sample output of some rows for the delimited format are depicted.

Example 7-17 Sample sqlplus statements to generate a delimited output file

```
clear columns
clear breaks
set pagesize 0
set linesize 10000
set feedback off
set heading off
set space 0
set newpage NONE
set trim ON
set trims ON
spool customer.DAT
select ' ||rtrim(MANU_CODE)||'|' ||rtrim(MANU_NAME)||'|'
||rtrim(LEAD_TIME)||'|'
from manufact;
spool off
quit
```

```
#after running the select the spool file content looks like this
SMT|Smith|3|
ANZ|Anza|5|
```

```

NRG|Norge|7|
HSK|Husky|5|
HRO|Hero|4|
SHM|Shimara|30|
KAR|Karsten|21|

```

Using Oracle stored procedures for a delimited unload

In this example, we explain an Oracle stored procedure, `export_table`, that was written by the authors of this book. It demonstrates how to unload the data from Oracle by using a stored procedure, and subsequently to load that data into IDS. This stored procedure can only be used for CHAR, VARCHAR2, NUMBER, and DATE, and TIMESTAMP data types. As in the previously described shell script, this stored procedure gets the table name as an input parameter. It then constructs the SELECT query for output, and exports the table data to an output flat file. This output file format is also a delimited ASCII file format. The delimiter is a '|' sign. Example 7-18 shows the definition for the `export_table` stored procedure.

Example 7-18 Procedure to export data

```

/*****
/* This stored procedure accept the table name as input      */
/* and exports the data into flat file identified by the     */
/* UTL_FILE_DIR with the format acceptable by the DB2       */
/* IMPORT utility or LOAD utility as Delimited ASCII file   */
/* Note : this procedure can be used for the table with data */
/* types CHAR,VARCHAR2 and NUMBER.                          */
*****/
CREATE OR REPLACE PROCEDURE export_table(
    i_table_name IN VARCHAR2           -- table name to be exported
)
IS
    stmt_1      VARCHAR2(4000) := 'select ';           -- first part of select
    stmt_2      VARCHAR(50) := ' as linecol from ';   -- second part of the select
    stmt_cursor INTEGER;                             -- statement handle
    linecol     VARCHAR2(4000);                       -- output buffer for utl_file
    ret         INTEGER;                              -- dbms_sql handle
    filepath    VARCHAR(40) := 'c:\temp';            -- path for output file
    filename    VARCHAR(40);                          -- output filename
    filemode    CHAR(1) := 'w';                      -- output file mode for write
    filelnsz    INTEGER := 4000;                    -- max file line size
    dtype_excp EXCEPTION;
    fhandle     utl_file.file_type;                 -- file handle for utl_file
    CURSOR col_csr(tab_col_name IN VARCHAR2) IS
        SELECT column_name, data_type
        FROM user_tab_columns
        WHERE table_name = upper(tab_col_name);

```

```

BEGIN
  stmt_1 := stmt_1 || '/*parallel(' || i_table_name || ',4)*/' || '''' || '''';

  /*****
  /* Build the select statement */
  *****/
  FOR my_rec IN col_csr(i_table_name) LOOP
    IF my_rec.data_type = 'DATE' THEN
      stmt_1 := stmt_1 || ' || rtrim(DECODE(' || my_rec.column_name
        || ',NULL,' || '' '' || ',TO_CHAR(' ||
        my_rec.column_name || ', '' ||
        'MM/DD/YY' || ''')) || '' || '''';
    ELSIF my_rec.data_type = 'TIMESTAMP(6)' THEN
      stmt_1 := stmt_1 || ' || rtrim(DECODE(' || my_rec.column_name
        || ',NULL,' || '' '' || ',TO_CHAR('
        || my_rec.column_name || ', '' ||
        'YYYY-MM-DD HH24:MI:SSxFF' || ''')) || '' || '''';
    ELSIF my_rec.data_type = 'CHAR' THEN
      stmt_1 := stmt_1 || ' || '' || '' || '' || rtrim(' || my_rec.column_name || ')
        || '' || '' || '' || '''';
    ELSIF my_rec.data_type = 'VARCHAR2' THEN
      stmt_1 := stmt_1 || ' || '' || '' || '' || '' || rtrim(' || my_rec.column_name || ')
        || '' || '' || '' || '''';
    ELSIF my_rec.data_type = 'NUMBER' THEN
      stmt_1 := stmt_1 || ' || rtrim(' || my_rec.column_name || ') || '' || '' || '' || '''';
    ELSE RAISE dtype_excp;
    END IF;

  END LOOP;
  stmt_2 := stmt_2 || i_table_name;
  stmt_1 := stmt_1 || stmt_2;

  /*****
  /* Execute the statement and open the cursor */
  *****/
  stmt_cursor := dbms_sql.open_cursor;
  dbms_sql.parse(stmt_cursor,stmt_1,dbms_sql.native);
  dbms_sql.define_column(stmt_cursor,1,linecol,4000);
  ret := dbms_sql.execute(stmt_cursor);
  filename:=i_table_name || '.DAT';
  fhandle:= utl_file.fopen('BLOBS',filename,filemode,filelpsz);

  /*****
  /* Fetch the rows and write it to output file */
  *****/
  WHILE dbms_sql.fetch_rows(stmt_cursor)>0 LOOP
    dbms_sql.column_value(stmt_cursor,1,linecol);
  
```

```

        utl_file.put_line(fhandle,linecol);

END LOOP;

/*****
*/ Close the cursor and file */
/*****
dbms_sql.close_cursor(stmt_cursor);
utl_file.fclose(fhandle);

EXCEPTION
WHEN dtype_excp THEN dbms_output.put_line('Invalid Data type');
END;
```

This stored procedure uses the Oracle DBMS_SQL package to construct the SELECT statement and retrieve the result set. It uses the UTL_FILE Oracle package to create the output file, open it, and write the output data to the output file. For using the UTL_FILE, the existence of a directory object is required. We named that object BLOBS in our example. It points to the c:\temp directory. The output file created will be named the <TABLE_NAME>.DAT file in the target directory. For example, to export the data in the ACCOUNTS table, the stored procedure is called using the CALL EXPORT_TABLE('ACCOUNTS') command.

The table name can be specified in upper and lower case. The stored procedure automatically converts the parameter in upper case for further processing.

Data type representation considerations

Planning and executing the unload of your data includes another important task. This is the data type mapping and the data representation in the unload file. The data type mapping in the database schema is basically done when you start the unload of the data. It is related to the rewrite of the create table statements.

Typically, not every database data type that is on the source side has an equivalent data type on the target database server. Therefore, a data type mapping between the two must be done. In addition, the name of the data type can be same but the meaning could be quite different.

In the following section we give you some details on where to look when unloading the data, in terms of the best fits for the used data types.

Numeric data

Oracle accepts in the create table statement the data types integer and small integer. Internally, these data types are mapped to NUMERIC(38), which allows the application to insert values much larger than that data type range normally accepts. In difference to this behavior, IDS follows the data ranges defined in the

C programming language. Example 7-19 shows the difference in behavior, based on a small sample table. Make sure that you choose the appropriate mapping for these data types. It would be a good idea either to check all the data in these columns for their ranges or choose, per default, a data type such as bigint or decimal to make sure that no rows are rejected during the load because of data values range constraint reasons.

Example 7-19 Some considerations using integer and smallint data types

```
#Create the table
SQL> create table numeric_data ( aa integer, bb smallint );

#Check the representation
SQL> desc numeric_data
Name                                     Null?    Type
-----
AA                                         NUMBER(38)
BB                                         NUMBER(38)
#inserting some large values
SQL> insert into numeric_data values ( 12345678912345678, 1234567890);

SQL> select * from numeric_data;
      AA      BB
-----
1.2346E+16 1234567890

#Try the same in informix dbaccess- table with same structure
SQL: New Run Modify Use-editor Outpu Choose Save Inf Drop Exit

insert into numeric_data (aa) values ( 220002022020202202202020202020 )

1215: Value exceeds limit of INTEGER precision
```

Character data types

The mapping for the char-based data type is straight forward, and there are similar data types on the Informix database server. Be careful when handling char-based data. Make sure that the length of the column on the target is defined large enough to store all of the column data. IDS allows you to create databases in ANSI mode and in non-ANSI mode. The default is non-ANSI mode. In this mode, if you try to insert a char value that is larger than the column definition, the current value is truncated to the defined column length. If you create an ANSI database, an error is returned and the insert of this value is rejected. Example 7-20 on page 191 shows you the difference in the behavior.

Example 7-20 Truncating column content versus rejecting the SQL statement

```
#Oracle an error is returned
SQL> create table char_data ( char_col char(20));

SQL> insert into testchar values ('220002022020202202202020202020');
insert into testchar values ('2200020220202022022020202020')
*
ERROR at line 1:
ORA-12899: value too large for column "INFORMIX"."CHAR_DATA"."CHAR_COL"
(actual: 30,maximum: 20)

#Informix behaviour depends on the database style
#Non ANSI
create database char_db;
create table char_data ( char_col char(20));

SQL: New Run Modify Use-editor Output Choose Save Info Drop Exit
Run the current SQL statements.

----- char_db@c_prim ----- Press CTRL-W for Help -----

insert into char_data values ('2200020220202022022020202020');

1 row(s) inserted.

#see the truncate
select "****" || char_col || "*****" from char_data

(expression)

***22000202202020220220***

#ANSI behavior
#Insert is rejected

create database char_db with log mode ansi
create table char_data ( char_col char(20));

insert into char_data values ('2200020220202022022020202020');

1279: Value exceeds string column length.
```

Time-based data types

You have to be careful when mapping time-based data types from Oracle to IDS. That is why we want to have a closer look at the DATE and the TIMESTAMP data type.

If you look at the DATE data type on the Oracle and compare it with the DATE data type on IDS, you will notice a slight difference. In Oracle, the DATE contains a time stamp from the day to the second. IDS also saves a specific day in this data type. Looking closer at the DATE data type behavior in the Oracle database you will see that the normal select * from table will only return the day description. Any additional information has to be extracted by the to_char built-in function. If you have DATE data types in the original table definition, be aware of the original goal when using this data type. Was it to store only the day, or a time stamp? If the intention was to save the day, you can use DATE on the target IDS database server for this column. Otherwise, a DATETIME year to second would be the better choice.

Regardless of which data type is used, you have to change the representation of the data type. The standard output for a DATE column in Oracle for a select * from table is DD-MON-YY. On IDS the default format is MM/DD/YY. For a different representation, you need to have a different setting in the DBDATE environment variable on IDS before you start loading the data. The comparison of the default output behavior in a simple select, and the change of the data representation for this specific data type, is summarized in Example 7-21.

Example 7-21 Using DATE only for a Day in Oracle, and how to match

```
#Oracle definition
SQL> desc orders
Name                                Null?    Type
-----
ORDER_NUM                           NOT NULL NUMBER(38)
ORDER_DATE                           DATE
```

```
#Informix Definition
create table "informix".orders (
    order_num serial not null ,
    order_date date )

#select the data on Oracle
SQL> select * from orders;
1001 20-DEC-98

#select the same data on IDS
select * from orders;
order_num      1001
```

```
order_date      12/20/1998
```

```
#Change the representation of the column
```

```
SQL> select order_num,to_char(order_date,'MM/DD/YYYY') from orders;
      1001 12/20/1998
```

If you use the DATE data type on Oracle to store a date time, you need to define a DATETIME year to second for the target Informix data type. The output of the data in the unload file has to be modified as described in Example 7-22.

Example 7-22 Using the DATE as a time stamp on Oracle

```
SQL> select to_char(order_date,'YYYY-MM-DD HH24:MI:SS') from orders;
```

```
TO_CHAR(ORDER_DATE,'YYYY-MM-D
```

```
-----
```

```
1998-12-20 05:21:31
```

If you use the data type TIMESTAMP in your original database table you also have to change the representation of the data in the unload file. The default output and the TO_CHAR expression for the appropriate output are shown in Example 7-23.

Example 7-23 Exporting data from a TIMESTAMP value

```
#Original table definition
```

```
SQL> desc cust_calls
```

Name	Null?	Type
CUSTOMER_NUM	NOT NULL	NUMBER(38)
CALL_DTIME	NOT NULL	TIMESTAMP(6)

```
#Default representation
```

```
SQL> select CALL_DTIME from cust_calls;
```

```
CALL_DTIME
```

```
-----
```

```
12-JUN-98 08.20.27.858698 PM
```

```
12-JUN-98 08.25.27.858698 PM
```

```
07-JUL-98 10.24.27.858698 PM
```

```
#Informix schema defintion
```

```
create table "informix".cust_calls
```

```
(
```

```
  customer_num integer,
```

```
  call_dtime datetime year to second
```

```

)

#Default representation
select CALL_DTIME,RES_DTIME from cust_calls;

call_dtime
1998-06-12 20:20:27
1998-07-07 22:24:06

#Necessary Conversion using target format year to second
#Query on Oracle
select ''
  ||rtrim(CUSTOMER_NUM)||'|'|
  ||rtrim(TO_CHAR(CALL_DTIME,'YYYY-MM-DD HH24:MI:SS'))
from cust_calls;

#Necessary Conversion using target format year to fraction
#Informix schema
create table "informix".cust_calls (
  customer_num integer,
  call_dtime datetime year to fraction(5))

#Query for the unload
select ''
  ||rtrim(CUSTOMER_NUM)||'|'|
  ||rtrim(TO_CHAR(CALL_DTIME,'YYYY-MM-DD HH24:MI:SSxFF'))
from cust_calls;

sqlplus -s informix/support1 < cust_calls.sql
106|1998-06-12 20:20:27.858698|

```

Using Object data types

If you have defined your own data types, the transfer of the data type definition in the schema needs only a small change in the SQL DDL. On the unload side the representation of user-defined data types is nearly the same, but does need a small change. The change has to be applied after the data is unloaded. We now look at the default representation of a user defined data type in both database servers. It is depicted in Example 7-24 on page 195.

Example 7-24 User-defined types and their appearance

```
#Oracle schema
SQL> desc employee
Name                                Null?    Type
-----
GIVENNAME                           CHAR(32)
FAMILYNAME                           CHAR(32)
ADDRESS                              ADDRESS_T
PHONE                                CHAR(32)

SQL> select ADDRESS from employee;

ADDRESS(ADDRESS1, CITY, STATE, ZIPCODE)
-----
ADDRESS_T('123, First Street ', 'Denver ', 'CO ', 80111)

#Informix Schema :

create table "informix".employee
(
  givenname char(32),
  familyname char(32),
  address "informix".address_t,
  phone char(32)
);

select      * from employee;
ROW('123, First street ', 'Denver ', 'CO ', 80111)
```

The Oracle output for the user type starts with the name of the type, whereas Informix starts with the literal ROW. Therefore, you have to change the type name to the row. You can make that change by using a pipe for the unload and redirecting the content through a sed script.

Handling Blobs

The recommended way to unload blobs is either by using an existing solution, or by writing your own procedure using the `utl_file` package and the `put_raw` function. This function is able to write binary data, up to 32000 bytes, in one step. If your data in a BLOB column is larger, you have to create a loop so you can perform multiple writes.

There is a reason why you have the BLOB in a stored procedure rather than by using a `select * from table`. The BLOB data is shown in SQL*Plus 11g in their ASCII representation on screen. Older versions do not show the BLOB content at

all. In Example 7-25 we show you the detailed steps for the BLOB unload and how the output for a BLOB column would look if using a sample select * from table.

Example 7-25 handling Blobs during the UNLOAD

```
#Defintion of a simple table with BLOB column in Oracle
SQL> desc blob_table
Name                                     Null?    Type
-----
AA                                         BLOB

SQL>select * from blob_table
AA
-----
74726973746172702F6E6577646666D2F646666D6578742E632044464D706B73697A6520
44464D706B73697A652875696E7434207265715F73697A652C20696E7434202A6C6F77

#Unload procedure

define the directory object for the unload path
select the rows from your blob table within a cursor
open the output file for the blob with utl_file.fopen for write
select the length of the current blob , use dbms_lob.getlength()
depending on the size < 32000 write the blob
    use  utl_file.put_raw();
        utl_file.fflush();
if the blob is larger than 32000 read pieces of the blob
    use  dbms_lob.read()
        utl_file.put_raw();
        utl_file.fflush();
close the output file
move over to the next BLOB row
```

Performance considerations

The generation of both of the covered output formats with the SELECT statement requires a full table scan. For the final execution of the data move you should definitely parallelize the full table scans. Use the parallel clause in the final select that will be initiating the full table scan. Additionally you should set up an unload and load strategy where multiple tables are unloaded in parallel to use the existing resources in terms of available processors and I/O throughput. That is, set up multiple clients for unloading different schema data. To do this, you could use the sample scripts and stored procedures we showed in our previous examples.

It would be good to use named pipes for the unload to save disk space. You can define the pipe on UNIX with the `mkfifo` command. Apply the appropriate permissions with the `-m` option to avoid unauthorized access to the data during your tests and for the final move. If using pipes you can combine the unload and the load together. However, ensure that the process for writing to the pipe is the first one. Otherwise, you will have synchronization problems.

7.4.2 Load the data into the target IDS database server

The next step in the data move chain is to load the extracted data into the IDS database server. There are several utilities available for loading data generated by external data sources. The following are examples:

- ▶ `dbload` utility
- ▶ `LOAD` SQL statement provided by `dbaccess`
- ▶ High Performance Loader

Depending on the format of the output file and the volume of data you want to load you can choose one of them. In the following sections, we give a short introduction to those tools. More detailed information about the load and unload utilities and their use in migration activities can be obtained from the High Performance Users Guide and the IBM Informix Migration Guide. You can download them from the following Web page:

<http://www-01.ibm.com/support/docview.wss?uid=swg27013894>

The SQL `LOAD` statement provided by the `dbaccess` utility

The SQL load statement is an Informix extension to the SQL standard and is only available in the `dbaccess` utility. You can either use `dbaccess` and run it step-by-step from the menu, or you can use it in a batch file. For batch, save all your SQL statements in a file and redirect the file to `dbaccess` at execution time.

The load statement expects a delimited file on disk, but can also be used to read delimited rows from a named pipe. The default delimiter is a “|”, but can be changed with the `DBDELIMITER` environment variable. This variable has to be set before starting the `dbaccess`. We show the prerequisites, and how to load the data with the `LOAD` statement, in Example 7-26 on page 198.

```
#UNLOAD file
$cat /tmp/manufact.DAT
SMT|Smith|3|
ANZ|Anza|5|
NRG|Norge|7|
HSK|Husky|5|
HRO|Hero|4|
SHM|Shimara|30|
KAR|Karsten|21|
NKL|Nikolus|8|
PRC|ProCycle|9|

create table "informix".manufact
(
    manu_code char(3),
    manu_name char(15),
    lead_time interval day(3) to day,
    primary key (manu_code)
);

$echo "load from /tmp/manufact.DAT insert into manufact;" | dbaccess -e
target_databasename

Database selected.

load from 'test.txt' insert into manufact;
9 row(s) loaded.

Database closed.
```

Loading data with the dbload utility

Another utility for loading the data is dbload. That utility is an executable provided with the IDS distribution. It can be used for loading data files in delimited or a column offset oriented fashion. A control file is provided as an input parameter at execution time that will define the layout of the load file, column mapping and the target database table. In brief there are the following options available to dbload as an input parameter to be more comfortable with the load behavior:

- ▶ Define commit points after a load, or specific number of loads
- ▶ Skip of rows in the data files
- ▶ Lock the table for the load
- ▶ Create a logfile specification
- ▶ Specify fault tolerance

The key input for the dbload is the specification of a control file. There, you determine if you load data from delimited or column-oriented file. Additionally, you specify the filename of the load file and define the target table where the rows will be loaded. You can specify multiple load files in one control file that enables the dbload utility to load more than one table at execution time. We show some sample control files for handling delimited rows in Example 7-27.

Example 7-27 sample dbload control files

```
#Control file with loadfile with delimited rows
FILE manufact.unl DELIMITER '|' 3;
INSERT INTO manufact;

#Loading multiple Files with one dbload start
FILE stock.unl DELIMITER '|' 6;
INSERT INTO stock;
FILE customer.unl DELIMITER '|' 10;
INSERT INTO customer;
FILE manufact.unl DELIMITER '|' 3;
INSERT INTO manufact;
```

The control file for column space-oriented rows looks a little bit different. Here you have to specify the name of the column and the offset for each column the row contains. You can also specify some default values for NULL values. We show samples of how to specify these values in Example 7-28.

Example 7-28 How to define column offsets in a control file

```
#Control File with column size oriented rows.
#the select from Oracle looked like this
CUSTOMER_N LNAME          FNAME
```

```
-----
1 Miller                Henry
2 Fischer               Henry
3 Carpenter             Joanne
```

Truncated by the head we finally got this file

```
1 Miller                Henry
2 Fischer               Henry
3 Carpenter             Joanne
```

```
#define the start of the column in the row
#keep in mind that you hve to add the byte for the space to the
#first 2 column sizes
$cat controlfile
FILE customer.unl
```

```
(
customer_num 1-11,
fname 12-33,
lname 33-53
);
```

```
INSERT INTO customer VALUES (customer_num, fname, lname);
```

Because most of the data types have a fixed length, or in case of varchar the select output is set as the maximum size, you should be able easily to define the control file for most of your data exports.

If you have your own object data types it works in the same way. We show you, in Example 7-29, how the definitions in the control file are structured for delimited and column size-based input files.

Example 7-29 Handling of user defined row types

```
#Table definition statement in Informix
create row type address ( street char(100),town char(100),zip integer);
create table customer (customer_num integer, lname char(100), fname
char (100), adr address);
#Load file delimited
1|Fisher|Henry|ROW('First Street','San Jose',95110)|
#Load File column oriented
      1 Fisher      Henry      ROW('First Street','San Jose',95110)

#Controlfile for a delimited load file
FILE rowtype.txt
DELIMITER '|' 4;
INSERT INTO customer;

#Controlfile for column sized load file
FILE rowtype.txt
(
customer_num 1-7,
lname 8-19,
fname 20-31,
address 32-67);

INSERT INTO customer VALUES (customer_num, lname,fname,address);
```

Setting up the control file according to the load file is the prerequisite for the execution of the dbload. When finished, you can start the dbload utility with the following command:

```
dbload -d <database_name> -c <controlfilename> -l <logfilefilename>
```

Loading data with the High Performance Load Utility

The High Performance Load Utility (HPL), similar to dbload, is shipped with the IDS product. It is for loading and unloading data in parallel, and is based on jobs. There can be different data sources, such as pipes, tapes or files, and file arrays for the load. It contains a graphical interface to create and monitor the jobs, but the job definition and execution can be also maintained by shell based executables. The HPL is able to load delimited, binary, ASCII and multibyte data sources. It should be the tool of choice when there is a requirement to load huge volumes of data.

When using the HPL utility for loading data into the database, there are a number of steps to take to create and execute a load job. The steps are always required regardless of which interface you use for the setup. You can execute the steps either with the graphical based utility, named ipload, which is available on UNIX platforms or you can use the onpladm utility for the setup. Perform the following steps to do so:

1. Create a job under a project
2. Define the data source as a device, which may be an array, a tape, or a pipe.
3. Define the source file format, such as delimited, ASCII or binary.
4. Create the SQL query to define the target database and the table.
5. Map the input stream position and fields to the target columns.
6. Define filter constraints in case not all rows should be inserted.

After you have created the job you can either execute it from the ipload GUI. If you used onpladm for the definition task, you have to use onpload for running the jobs. There are two types of jobs you can define. The choice when loading large volumes of data should be the Express mode. It takes advantage of performance improvements during the load, such as switching off logging, and disabling of indexes and constraints. After the load, all objects that are disabled during the load by the HPL are automatically reestablished. The only requirement for the administrator, after running an express load, is taking a Level 0 archive. During the load of the data into the production system you should combine the load of multiple tables before you initiate the Level 0 backup, to save time.

Next, we look at some brief examples on how to set up a data load job with the help of onpladm, and how to execute the job. The first two examples are in the Windows environment. The output of the job execution may vary a little on UNIX platforms. We start with a simple job definition, the load of a delimited unload file into a target database table in express mode. This is depicted in Example 7-30

on page 202. For such an example, make sure that you specify a log file on WINDOWS for the execution of the job to see the status. On UNIX, the output is automatically generated on /dev/stdout.

Example 7-30 How to setup and execute a Job using the HPL utility

```
#Define the job
onpladm create job load_table -d tabledata.unl -D target -t customer
-f1
#execute the job
C:\temp\HPL>onpload -j load_table -f1 -l output

C:\temp\HPL>type output
Thu Feb 19 20:47:25 2009

SHMBASE      0x0c000000
CLIENTNUM    0x49010000
Session ID 1

Load Database  -> target
Load Table     -> customer
Device Array   -> load_table
Record Mapping -> load_table
Convert Reject -> c:\temp\load_table.rej
Filter Reject  -> c:\temp\load_table.flr
Set mode of index zip_ix to disabled
Reset mode of indexes "holgerk".zip_ix to original enabled mode
Table 'customer' will be read-only until level 0 archive

Database Load Completed -- Processed 56 Records
Records Inserted-> 56
Detected Errors--> 0
Engine Rejected--> 0

Thu Feb 19 20:47:27 2009

#you can additionally run the job with
C:\temp\HPL>onpladm run job load_table -f1
```

This definition only maps columns one-to-one from the delimited data file. In addition we only have one data file. If you want to have an array of multiple data file as an input you have to modify the array definition. To do so, write the definition in a file and create the new device object in the database server. The definition file is specified by the -F option. After that, create a job using this new object. Make sure that you use the -fla option when specifying the array. All

necessary definitions and commands for creating and executing the job are described in detail in Example 7-31.

Example 7-31 Definition of a device array and the usage in a load job

```
#device definiton
c:\temp\HPL>type devicefile
BEGIN OBJECT DEVICEARRAY load_table_array
BEGIN SEQUENCE
TYPE FILE
FILE C:\temp\HPL\tabledata.un1
TAPEBLOCKSIZE 0
TAPEDEVICESTRIP 0
PIPECOMMAND
END SEQUENCE
BEGIN SEQUENCE
TYPE FILE
FILE C:\temp\HPL\tabledata1.un1
TAPEBLOCKSIZE 0
TAPEDEVICESTRIP 0
PIPECOMMAND
END SEQUENCE
BEGIN SEQUENCE
TYPE FILE
FILE C:\temp\HPL\tabledata2.un1
TAPEBLOCKSIZE 0
TAPEDEVICESTRIP 0
PIPECOMMAND
END SEQUENCE
END OBJECT

#Create the Device
C:\temp\HPL>onpladm create object -F devicefile
Successfully created object DEVICEARRAY load_table_array

#Check existence
C:\temp\HPL>onpladm list device
load_table_array

#Create the Job we use -fla specifying the device array
C:\temp\HPL>onpladm create job load_arr -d load_table_array -D target
-t customer -fla
Successfully created Job load_arr

C:\temp\HPL>onpload -j load_arr -f1 -l output
```

In the final stage we give you an overview of what is needed when using a pipe as the input source. The steps for creating the pipes as an input source and the definition of the load jobs are similar. Example 7-32 shows the detailed commands using the HPL and LINUX as the base operating system.

Example 7-32 Using a pipe as an input for HPL

```
#Defintion of the pipes in a device array
#cat pipedesc

BEGIN OBJECT DEVICEARRAY pipedesc
BEGIN SEQUENCE
TYPE PIPE
FILE
TAPEBLOCKSIZE 0
TAPEDEVICESTR 0
PIPECOMMAND "cat /tmp/taledata"
END SEQUENCE
BEGIN SEQUENCE
TYPE PIPE
FILE
TAPEBLOCKSIZE 0
TAPEDEVICESTR 0
PIPECOMMAND "cat /tmp/taledata1"
END SEQUENCE
BEGIN SEQUENCE
TYPE PIPE
FILE
TAPEBLOCKSIZE 0
TAPEDEVICESTR 0
PIPECOMMAND "cat /tmp/taledata2"
END SEQUENCE
END OBJECT

$create object -F pipedesc
Successfully created object DEVICEARRAY loadfrompipe1

$onpladm create job load_from_pipe -d pipedesc -D target -t customer
-fla
Successfully created Job load_from_pipe

$onpload -j load_from_pipe -fla

Thu Feb 19 21:42:48 2009
```

```
SHMBASE      0x0c000000
CLIENTNUM    0x49010000
Session ID 6
```

```
Load Database -> target
Load Table    -> customer
Device Array  -> pipedesc
Record Mapping -> load_from_pipe
Convert Reject -> /tmp/load_from_pipe.rej
Filter Reject -> /tmp/load_from_pipe.flt
Set mode of index zip_ix to disabled
Reset mode of indexes "holgerk".zip_ix to original enabled mode
Table 'customer' will be read-only until level 0 archive
```

```
Database Load Completed -- Processed 168 Records
Records Inserted--> 168
Detected Errors--> 0
Engine Rejected--> 0
Thu Feb 19 21:42:53 2009
```

There is rich variety of options for setting up the load jobs with the HPL functionality provided by IDS, although we primarily focused on the load of delimited files. There are of course possibilities for setting up column offset based load files, other than default column mapping and row filtering based on filter values. For more detailed information about the HPL, refer to the IBM Informix High-Performance Loader User's Guide, SC23-9433. You can download the pdf file from the following URL:

<http://publib.boulder.ibm.com/infocenter/idshe1p/v115/index.jsp>

Special considerations loading BLOB and SBLOB data

Loading BLOB or SBLOB data require special treatment. Not all of the introduced utilities for the load do support the direct load of BLOB or SBLOB data types. We want to give in the next section some guidances how you can develop your own solution loading the data.

For loading BLOB data you should refer to the file blobload.ec. Its distributed with the Informix Client SDK in the demo/esqlc directory down of the \$INFORMIXDIR. This file contains the demo source code for a sample client inserting BLOB located in a file to a database table. Basically the treatments of blobs is be done in ESQL/C by a specific structure loc_t. This structure contains the directions for the blob like size and file location. After an initialization you can use the variable like all other build data type based variables as a host variable. Example 7-33 on page 206 shows you the details how your code could look.

Example 7-33 How to insert a BLOB to a table in ESQL/C

```
#Informix Database schema
create table blob_tab ( the_blob BYTE );

#sample parts of the esql/c program
#include sqlca;
#include <stdio.h>
main( int argc, char **argv)
{
$loc_t images;
$char the_statement[200]
$char blobfile[200];

$database blob_database;
sprintf(blobfile,"%s",argv[1]);
strcpy(the_statement,"insert into blob_tab values ( ? )");
$prepare sql_statement from :the_statement
images.loc_loctype = LOCFNAME;      /* blob is named file */
images.loc_fname = blobfile;       /* here is its name */
images.loc_oflags = LOC_RDONLY;    /* contents are to be read in IDS*/
images.loc_size = -1;              /* read to end of file */
images.loc_indicator = 0;          /* not a null blob */
}

$execute sql_statement using :images;
if ( sqlca.sqlcode != 0L)
    printf("SQL error %ld occured during insert\n",sqlca.sqlcode);
$close database
}
```

Similar to standard BLOB data type, the SBLOB data type requires special treatment in IDS. The Informix Client SDK product shipped with a complete IDS distribution provides the library function set, which can be used by the client to load SBLOB data. Be aware that using SBLOB data types requires the existence of sblob spaces in the IDS database server. These spaces are dbspaces intended to maintain SBLOB data types.

As an exercise, we developed a small sample ESQL/C program for loading SBLOB data. The code for the program, the base table definition, and a sample onspaces call for creating a Sbspace are shown in Example 7-34 on page 207. The file name containing the SBLOB is provided at execution time. The database is opened first, and the sblob descriptor needed for communication with the database server is initialized in the next step. After that the description of the sblob column is requested from IDS. A file descriptor is created and the data is

sent to the database server before the real insert happened. The sblob descriptor is used as the host variable for the insert to make sure the appropriate sblob data is assigned to the column.

Example 7-34 Inserting data into SBLOB data types in IDS

```
#create a sample SBSPACE -- more Options for logging are available
onspaces -c -S sblobspace -g 4 -p <path of the OS file> -o 0 -s 200000

#Create a table with a SBLOB
create table "informix".sblob_tab
(
    catalog_num serial8 not null ,
    advert_descr "informix".clob
) PUT advert_descr in ( sblobspace)
( extent size 20, keep access time)
extent size 16 next size 16 lock mode page;

#the sample esql/c program for a load of an SBLOB
#include sqlca;
#include <stdio.h>

main(int argc, char **argv)
{
EXEC SQL BEGIN DECLARE SECTION;
int8 estbytes;
int error, numbytes, lofd, ic_num, buflen = 256;
char buffer[150000];
ifx_lo_create_spec_t *create_spec;
fixed binary 'clob' ifx_lo_t_descr;
EXEC SQL END DECLARE SECTION;

FILE *fp ;

if ( argc < 2 ) exit(2);

fp=fopen(argv[1],"r");
if ( fp ) {
    estbytes=fread(buffer,100000,1,fp);
    fclose (fp);
}
else exit(1);

$database sblob_db;
```

```

/* create the sblob descriptor */
if ( ifx_lo_def_create_spec(&create_spec) < 0 )
printf(" Error ifx_lo_def_create_spec %d \n",error);

/* get the column description from IDS */
if (ifx_lo_col_info
    ("sblob_db@on10fc4tcp:sblob_tab.advert_descr", create_spec) < 0 )
    printf(" Error ifx_lo_def_create_spec %d \n",error);

/* create the SLOB structure and initialize with server data */
if ((lofd=ifx_lo_create
    (create_spec,LO_RDWR|LO_NOBUFFER,&descr, &error)) == -1)
    printf(" Error ifx_lo_create %d\n",error);

/* send the BLOB to the server */
numbytes=ifx_lo_write(lofd, buffer, estbytes, &error);

if (numbytes<size )
    printf(" Error ifx_lo_write %d\n",error);

/* Do the insert we use a serial , first column value is 0 */
EXEC SQL insert into catalog values (0, :descr);
ifx_lo_close(lofd);

$disconnect all;
}

```

Performance considerations for loading the data

You can significantly influence the time required for the data load by considering several areas for performance improvements. Running multiple load processes in parallel, ideally for tables located on different disks, reduces the wait time. Think about fragmenting those tables containing the largest amount of data. You should not see this only under the load performance perspective, but also as a chance for improvement for the later data access. You can reach fragment elimination in queries when the application which is filtering data in queries follows the fragmentation schema. In addition you can much better optimize your queries with Parallel Data Query (PDQ) means running multiple scans in parallel when there is table fragmentation.

Additionally you should disable logging, indexes and constraints during the load phase in case you insert a larger amount of data. Load the data first, create the indexes and constraints after, and then switch on the logging for the table. This would reduce the I/O for writing the log entries first. Additionally the time creating the index once based on all data compared with inserting each index entry in a

single step and ensuring a balanced index tree is significant faster. Finally the create of the index at the end ensures in IDS, at least for the index columns, that actual distribution data is available for the SQL optimizer for achieving optimal access plans.

7.4.3 Moving data using the Migration Tool Kit

In Chapter 4, “IBM Migration Tool Kit: An introduction” on page 59, we explain how to use the Migration Tool Kit (MTK) to generate scripts for data unload and data load. The correlation of scripts and table definitions of the source and target are defined in the MTK. The MTK also allows you to move (deploy) data through its GUI, online, without the need for the generated scripts.

7.5 Alternative ways for moving data

In addition to the MTK, there are many other tools and products for data movement. In this section we show a few of them. There are also a number of third-party tools that work with both Oracle and IDS, but we do not describe them in this IBM Redbooks publication. Performing an Internet search can quickly identify many of them for you. You should choose the tool according to your environment and the volume of data that will be moved.

7.5.1 IBM InfoSphere Information Server

In addition to the possibility of moving the data manually triggered either with shell scripts, Oracle stored procedures, or with the MTK as a GUI-based solution, consider using the IBM InfoSphere Data Stage product as a information movement and data integration tool.

There are several benefits to be gained by using the DataStage® product. For example, it is intended to be used as an ETL (Extract, Transform, and Load) tool, which fulfills the requirements of data movement. You can access both databases with so called stages, provided by the product. The only requirement when using DataStage is that the database schema be already successfully migrated from one database to another. In addition to extracting the data from the source database server you can also define sources of different types, such as flat files with database data, and combine them. You can define translation jobs in the DataStage product itself, which can be used to achieve an automated data representation translation (for example, for the date and time based data types). Another advantage is job scheduling. Jobs can be created and scheduled at specific times. In addition, multiple jobs can be run in parallel and so call job chains can be specified. This means that the successor can only run if the current job was successfully completed.

The job scheduling process enables you to execute the load in off-peak times to get exclusive access to the data or at least more system resources for the data scan. Scheduling multiple jobs can be useful to achieve a better utilization of the source database server and a significant reduction of the unload time itself.

For more information, visit the IBM InfoSphere Information Server product site:

<http://publib.boulder.ibm.com/infocenter/iisinfsv/v8r0/index.jsp>

In addition, there is another IBM Redbooks publication available on this topic, *IBM InfoSphere DataStage Data Flow and Job Design*, SG24-7576. You can download that IBM Redbooks publication from the following Web page:

<http://www.redbooks.ibm.com/abstracts/sg247576.html?open>

Application conversion

With the successful completion of the data movement to the target Informix Dynamic Server (IDS) database server, you have set up the base for the next step. Now you must enable access to the data for your applications.

In this chapter we give you an overview of the commonly used programming interfaces that are supported by IDS. We also discuss the requirements and the major tasks of an existing migration strategy. This strategy should enable you to plan the appropriate resources and time to accomplish the development and QA requirements in your migration plan.

Depending on the application programming interface used, there are different requirements for a database client application migration. The major focus in this chapter is a detailed discussion about available database interfaces and the differences between the access techniques in the source and target database server. We describe the conversion of applications written with Oracle Pro*C and Java, based on Oracle OCI, ODBC, Perl, PHP, and .NET.

8.1 Heterogeneous application environments

Most of the modern software infrastructures contain a mix of different applications and servers. There are business driven applications especially developed by the companies own development teams. On the other hand, there are also a wide variety of products and solutions available that have been provided by independent software companies. They provide either a complete solution for unified company processes, such as HR operations, order and sales processing, and accounting systems. In cases where internal processes are defined that cannot be unified, most of the software providers also define customizing facilities for the software.

In the middle of the complete application architecture are the servers. Commonly Web servers, application servers, and database servers are in use. Our special focus is directed toward the database server. We are interested in the available ways to exchange data between an application and the database server or across the server. This is important under the focus of porting applications across different database providers.

If you take an inventory of the type of applications you run in your companies database client environment, you would likely define two types of applications:

- ▶ Applications where you own the source and changes can be internally applied
- ▶ Applications defined on standard interfaces where you have to rely on the database server certification by the software provider.

Both types of applications can use the same unified database interfaces, but the strategy for how to migrate the particular type of application is quite different. We take a closer look at both in the following sections.

8.2 Client development APIs supported by IDS 11

The majority of IBM Informix development application programming interfaces (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 other standalone APIs and tools that we introduce throughout this section. In this section we provide an introduction to those development APIs, their specifics, and their supported functionality.

We provide more details later in this chapter as we discuss the specific porting issues for each supported database interface.

8.2.1 Embedded ESQL/C

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 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 complex call level interfaces to achieve the same goal. Even though ESQL/C provides a 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.

8.2.2 Embedded 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. 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.

8.2.3 Informix JDBC 3.0 Driver

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 *IBM 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 (JVM).

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.

8.2.4 IBM Informix .NET Provider

.NET is an environment that allows you to build and run managed applications. A managed application is an application in which memory allocation and deallocation are handled by the runtime environment. Another good example for a managed environment is a JVM.

The .NET key components are as follows:

- ▶ 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.

The following list details examples of programming languages that create applications that meet this criteria:

- ▶ Visual BASIC .NET
- ▶ Visual C#® .NET
- ▶ Visual J#® .NET
- ▶ ASP.NET

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.

8.2.5 IBM Informix ODBC 3.0 Driver

The IBM Informix Open Database Connectivity (ODBC) Driver is based on the Microsoft ODBC 3.0 standard, which is, in turn, 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 it dynamically.

8.2.6 IBM Informix OLE DB Provider

Microsoft OLE DB is a specification for a set of data access interfaces designed to enable a variety of data stores to work together seamlessly. OLE DB components are data providers, data consumers, and service components. Data providers own data and make it available to consumers. Each provider's implementation is different, but they all expose their data in a tabular form through virtual tables. Data consumers use the OLE DB interfaces to access the data.

You can use the IBM Informix OLE DB Provider to enable client applications, such as ActiveX® Data Object (ADO) applications and Web pages, to access data on an Informix server.

Due to the popularity of the Microsoft .NET framework, Informix developers on the Microsoft platform typically prefer the .NET database Provider and integrating existing OLE DB-based applications through a Microsoft .NET Provider for OLE DB.

8.2.7 IBM Informix Object Interface for C++

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.

8.2.8 Additional APIs for accessing IDS 11

The following sections describes additional APIs for accessing IDS 11.

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. The term *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.

The following list details the PHP drivers for accessing IDS that are available:

- ▶ Unified ODBC
 - Uses the standard ODBC interface for database communication
 - Is commonly combined with a generic ODBC driver
- ▶ The Informix PHP driver (ifx interface)
 - Uses a native database connection to IDS provided by ESQL/C connection libraries
- ▶ Informix PDO
 - Object-oriented database development interface
 - Available from PHP 5
 - Uses a native database connection to the database server

PHP can be obtained in different ways. Either you download the source, compile the base package and all necessary programming interfaces and plug it into a pre-existing Web server like Apache. Or, download a precompiled version like ZEND Core for IBM or install the XAMPP package.

We suggest downloading and installing the ZEND Core for IBM to use PHP with IDS because there is a full integration of the available driver for the database server.

Perl database interface

To better understand how the interface works, let us examine the PERL database interface (DBI). A Perl program uses a standard API to communicate with the DBI module for Perl, which supports only dynamic SQL. It defines a set of methods, variables, and conventions that provide a consistent database interface

independent of the actual database being used. DBI gives the API a consistent interface to any database that the programmer wishes to use. DBD::Informix is a Perl module which, when used in conjunction with DBI, allows Perl to access the IDS database.

Figure 8-1 illustrates the Perl/ Informix environment.

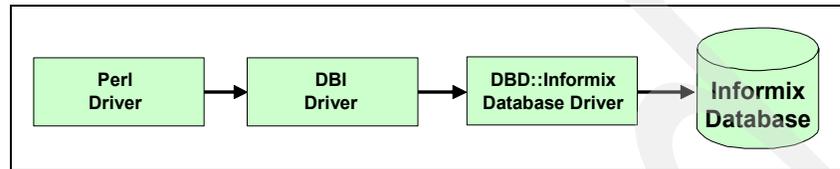


Figure 8-1 Perl/Informix invocation and data flow

Tcl/Tk and the Informix (isqltcl) extension

There is an extension available to access the IDS database server within a Tcl/Tk programming environment. The name of the extension is `isqltcl` and can be obtained on the Web by using the following URL for the download:

<http://isqltcl.sourceforge.net/>

8.3 Migrating applications using unified interfaces

In this section we discuss how to set up a migration plan for database applications bought from an external source with limited or no access to the implementation logic. Based on an ODBC application on Windows implemented with SQL and ODBC standards, we show you how to change the database connectivity module to enable the client application to access the target database server.

Finally, we provide some hints on how to apply a new database connection to a JDBC based application.

8.3.1 Package applications migration planning

For third party package applications, the vendor delivers the application and ensures the database connectivity to IDS. In comparison with a self-written source code owned application the migration is limited to the following major tasks:

- ▶ Checking software and hardware availability and compatibility
- ▶ Education of developers and administrators
- ▶ Analyzing of customized changes in the application and database

- ▶ Setting up the target environment
- ▶ Changing of customized items
- ▶ Testing of data migration and customized items
- ▶ Roll-out

To keep the support from the vendor, you have to meet the prescribed migration plan and process.

8.3.2 Migrating applications based on ODBC

There are two common implementation techniques for accessing the database server seen in current third party database application packages. Either they rely on the SQL standards and try to implement all the SQL statements following this standard, or they provide a database specific connection library which will be needed to attach to the application (such as a cartridge).

In any migration project where you want to move the application from one database server to another, you have to check which implementation of the database connectivity is provided. In both cases you have to contact the software manufacturer for this particular solution to make sure that the target database server and which version is supported. In case there is a connection library needed, you would typically get this from the application vendor.

If the application is implemented with a standard SQL Interface, such as SQL-92 or newer standards, and it is not based on Java, it likely uses the ODBC interface for the connection. The ODBC standard enables the database provider to implement the database connectivity only once and certify a specific database for the product. The ODBC standard can be used on both Windows and UNIX.

If you want to move an application based on ODBC, you need to apply a new ODBC driver. This type of database application can be either based on pure ODBC connection library, a .NET Windows-based application using the ODBC .NET Data Provider, or a PHP application using the ext/ODBC or PDO_ODBC database interface. The installation of a new Informix ODBC driver can be done by the installation of either the Informix SDK or the Informix Connect product. You will need the data source name (DSN). After the installation, it follows an easy to use server connection specification and ODBC driver registration. When you have enabled the new Informix ODBC DSN, change the connection database server in the application by the specification of the new DSN. After the change of the DSN you can start testing the integration of the application in the new database environment.

The setup for the new ODBC driver can be done in two steps:

1. Set up the communication parameter for IDS with the setnet32 utility.
2. Create a new ODBC DSN in the ODBC manager on Windows.

The setup window for the setnet32 utility is shown in Figure 8-2.

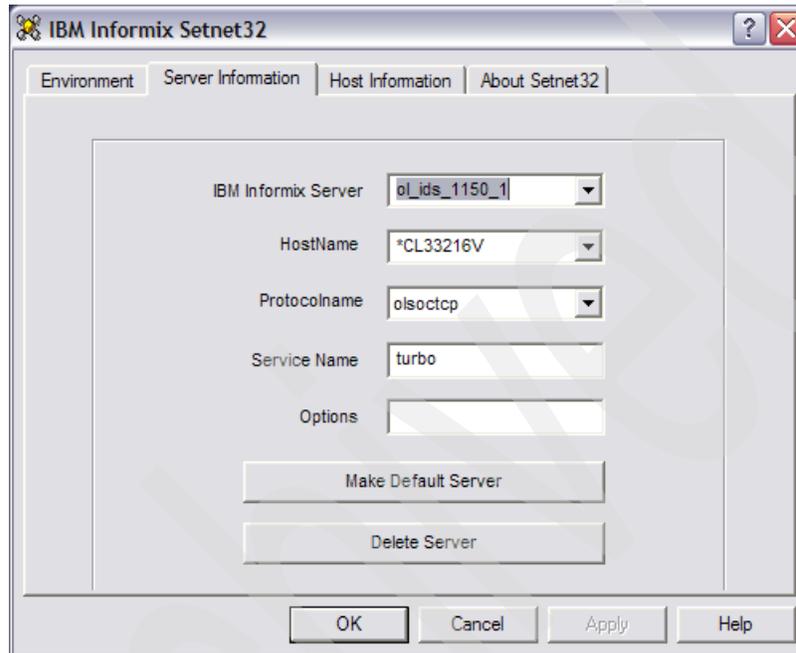


Figure 8-2 Using setnet32 to specify the remote IDS database server

After the specification of the server parameter you have to add a new DSN to the ODBC administrator in Windows. The appropriate window for setting up a Informix data source on Windows is shown in Figure 8-3 on page 221.

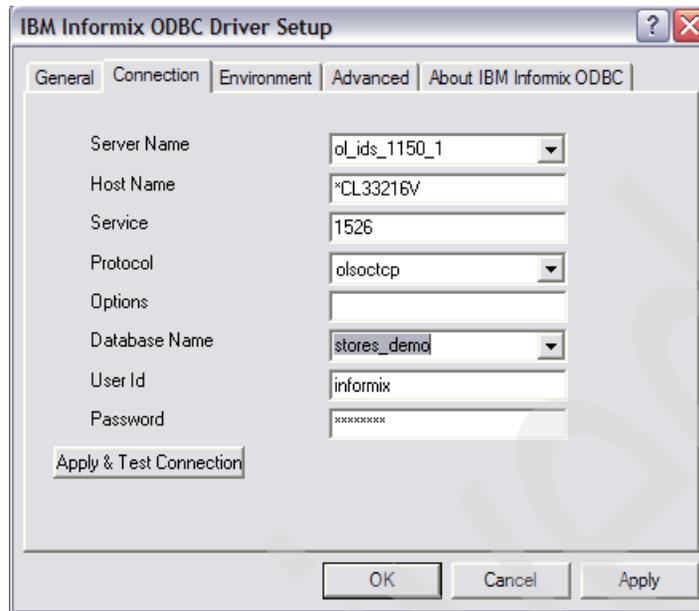


Figure 8-3 Setting up a Informix DSN with the ODBC manager on Windows

8.3.3 Migrating database applications based on JDBC

Java and Eclipse-based applications commonly involve a JDBC driver for the maintenance of the database access. In general, the JDBC driver represents a jar file that needs to be included in the CLASSPATH of the client application environment. Then, the specification of a URL for the target database server, and a user and password for the user issuing the connection, is required.

In summary, you have to perform the following steps to add a new JDBC connection to your application:

1. Install the Informix JDBC driver on the client application machine.
2. Depending on the application, select one of the following actions:
 - Add the driver location to the CLASSPATH. The name of the jar file is `ifxjdbc.jar`.
 - Select the Informix driver. Some applications include JDBC drivers for a set of database providers, so you can simply select Informix from that set.
 - Register the new driver in the application and specify the driver location.

3. Define a new database source in the application and specify the necessary connection parameter for the URL, along with the user ID and the password.
4. Test the connection, and start application testing with that connection.

Figure 8-4 shows an example of a configuration window for the specification of JDBC-based database access to IDS, and was created using Rational Data Architect. The configuration for your application should be similar.

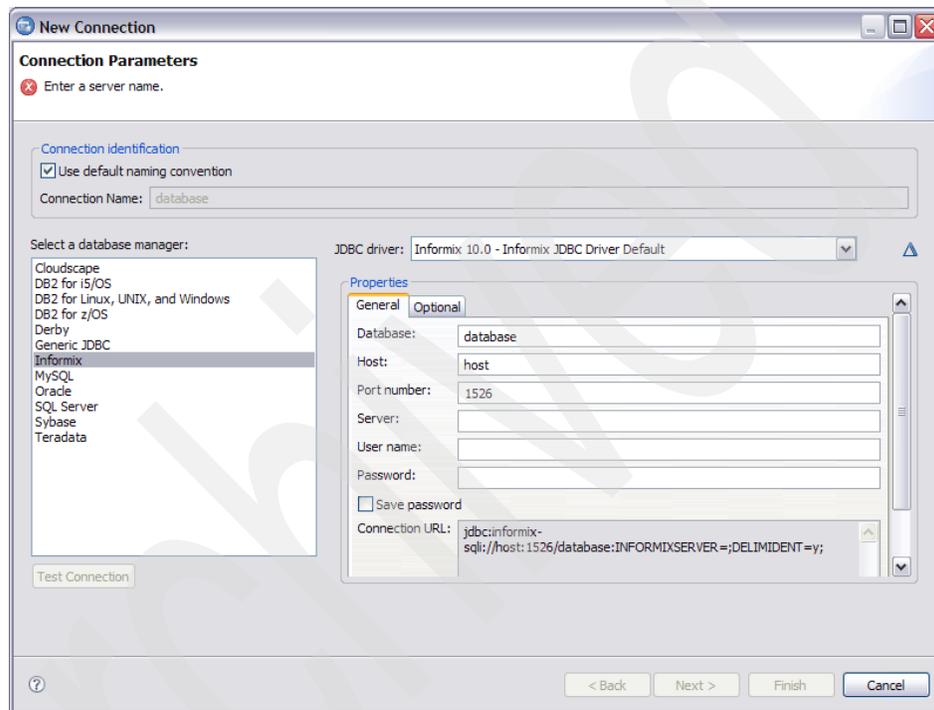


Figure 8-4 Configure a new database connection based on a JDBC driver

8.4 Conversion considerations for common client APIs

In this section we focus on application migration where you also want to apply code changes. Most of the time these applications are user-written. They use either native database interfaces such as those for embedded SQL or to access the database with common standards such as ODBC, JDBC or the .NET framework.

We start the database application conversion discussion with a brief introduction and the definition of a planning strategy. Then we follow with a verification of the

existing programming techniques for using database interfaces. We complete the topic with an overview of porting issues typically required by the most commonly used database application development environments. We also introduce a sample database application task and show the differences in the specification of parameters and implementation in the source.

Because each database client API provides a wide variety of functions and parameters which can be used for connection management, executing statements, processing the result sets, and error handling, presenting them all is beyond the scope of this document. We instead provide links to other sources of information required for your specific porting needs.

8.4.1 Application migration planning for source owned applications

Before you migrate existing user written applications in your database environment, develop a strategy for the migration. This strategy defines the major steps you should follow in this process. In addition, it should include the definition of goals, user resources, and timestamps to track the status and result in a successful migration.

This migration strategy should include the following tasks:

- ▶ Determining software and hardware availability and compatibility
- ▶ Education of developers and administrators
- ▶ Analysis of application logic and source code
- ▶ Setting up the target environment
- ▶ Change of database-specific items
- ▶ Application testing
- ▶ Application tuning
- ▶ Roll-out of the migrated applications
- ▶ User education

You should also create a project plan, which includes sufficient time and resources for each task. IBM and IBM Business Partners can help you with this activity to assure you have a well-defined and complete project plan.

Check software and hardware availability and compatibility

The architecture profile is one of the outputs of the first tasks in the migration planning assessment. While preparing the architecture profile, determine the availability and compatibility of all software and hardware in the new environment.

Education of developers and administrators

Understanding the new products is essential for analyzing the source system. Ensure that the staff gets educated and has the skills for all products and the system environment that you will use for the migration project.

Analyzing application logic and source code

In this analysis phase, you should identify all the Oracle proprietary features and the affected sources. Examples of Oracle proprietary features are direct SQL queries to the Oracle Data Dictionary, Oracle-styled Optimizer hints and Oracle joins, which are not supported by Informix. You also need to analyze the database calls within the application for the usage of database APIs.

Setting up the target environment

The target system, whether it is the same or a different one, has to be set up for application development. The environment can include the following parts:

- ▶ The Integrated Development Environment (IDE)
- ▶ Database framework
- ▶ Repository
- ▶ Source code generator
- ▶ Configuration management tool
- ▶ Documentation tool

A complex system environment usually consists of products from a number of different vendors. Check the availability and compatibility issues before starting the project.

Change of database-specific items

Regarding the use of the database API, you need to change the database calls in the applications. The changes are as follows:

- ▶ Language syntax changes

The syntax of database calls varies in the different programming languages. In 8.6, “Migrate user-built applications” on page 229, we discuss the varieties of C/C++ and Java applications. For information regarding other languages, contact IBM Technical Sales.

- ▶ SQL query changes

Oracle supports partly nonstandard SQL queries, such as including optimizer hints and table joins, with a (+) syntax. To convert such queries to standard SQL, you can use the MTK SQL Translator.

You need to modify the SQL queries to the Oracle Data Dictionary as well, and change them to select the data from the Informix system tables.

- ▶ Changes in calling procedures and functions

Sometimes there is a need to change procedures to functions and vice versa. In such cases, you have to change all the calling commands and the logic belonging to the calls, in the database and the applications.

- ▶ Logic changes

Because of architectural differences between Oracle and Informix, changes in the program flow might be necessary. Most of the changes will be related to the difference in concurrency models.

Application test

A complete application test is necessary after the database conversion, along with any application modifications to ensure that the database conversion is completed and all the application functions work properly.

It is prudent to run the migration several times in a development system to verify the process. Run the same migration on a test system with existing test data, then on a copy or subset of production data. Only when all results are positive should you consider running the process in a production environment.

Application tuning

Tuning is a continuous activity for the database because data volume, number of users, and applications change over time. After the migration, application tuning should also be performed. Having a good understanding of the architectural differences between Oracle and Informix is required for a good understanding of how perform application tuning. For more details refer to the IDS Performance Guide, G251-2296.

Roll-out

The roll-out procedure varies depending on the type of application and database connection you have. Prepare the workstations with the proper drivers (for example, Informix SDK or Informix Connect) and server that is consistent with the IDS version.

User education

In dealing with changes in the user interface, the business logic, and the application behavior caused by system improvements, user education is required. Provide sufficient user education, because the acceptance of the target system is largely determined based on the skills and satisfaction of the users.

8.5 Introduction to programming techniques

To develop applications that access the IDS database server, you have to embed the data access method of the high-level language used into the application. Informix SDK provides various programming interfaces for data access and manipulation.

There are various methods for performing data interaction from your application, including:

- ▶ Embedded static SQL
- ▶ Dynamic SQL
- ▶ Native API calls
- ▶ Methods provided by Informix drivers for a specific application environment.

In this section we give you a brief introduction to the existing implementation techniques.

8.5.1 Embedded SQL

The SQL statements can be embedded within a host language where SQL statements provide the database interface, while the host programming language provides all remaining functionality. Embedded SQL applications require a specific precompiler for each language environment to preprocess (or translate) the embedded SQL calls into the host language. Only then can the applications be compiled, linked, and bound to the database.

Figure 8-5 on page 227 illustrates the precompile-compile-bind process for creating an application with embedded SQL.

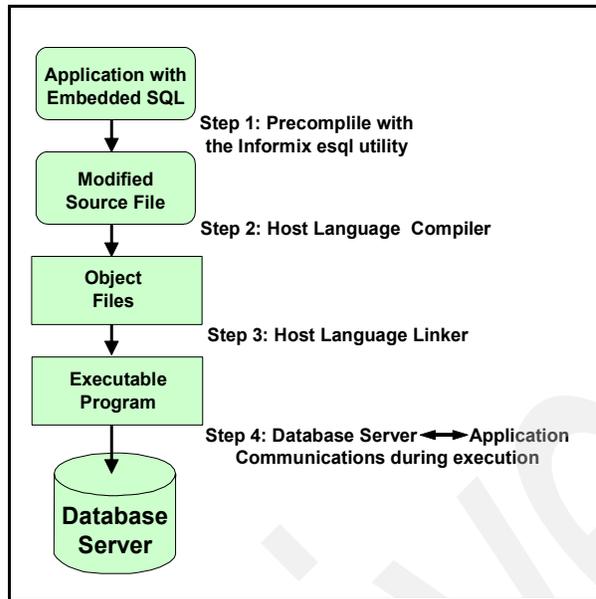


Figure 8-5 Precompile process

The Informix SDK supports the C/C++, COBOL, and Java (SQLJ) programming languages for embedded SQL.

Embedded SQL applications can be categorized as follows:

- ▶ **Static embedded SQL**

In embedded SQL, you are required to specify the complete SQL statement structure. This means that all the database objects (including columns and table) must be fully known at precompile time, with the exception of objects referenced in the SQL WHERE clause. However, all the host variable data types still must be known at precompile time. Host variables should be declared in a separate EXEC SQL DECLARE section and be compatible with IDS data types.

Example 8-1 on page 228 shows a fragment of a C program with static embedded SQL.

Example 8-1 Using embedded SQL in a C program

```
EXEC SQL include sqlca;
#include <stdio.h>

main(int argc, char **argv)
{
EXEC SQL int custno=110;
EXEC SQL database stores_demo;
EXEC SQL update customer set fname="Miller-Smith" where
customer_num=:custno;
printf("%ld\n",sqlca.sqlcode);
}
```

► **Dynamic embedded SQL**

If not every database object in the SQL statement is known at precompile time, you can use dynamic embedded SQL. The dynamic embedded SQL statement accepts a character string host variable and a statement name as arguments. These character string host variables serve as placeholders for the SQL statements to be executed later. Dynamic SQL statements are prepared and executed during program runtime.

Example 8-2 is a fragment of a C program with a dynamic SQL statement.

Example 8-2 A dynamic SQL C program

```
EXEC SQL BEGIN DECLARE SECTION;
char query[80];
char parm_var[19];
EXEC SQL END DECLARE SECTION;

strcpy( query, "SELECT tabname FROM systables" );
strcat( query, " WHERE tabname <> ? ORDER BY 1" );

EXEC SQL PREPARE s1 FROM :query;
EXEC SQL DECLARE c1 CURSOR FOR s1;
strcpy( parm_var, "customer" );
EXEC SQL OPEN c1 USING :parm_var;
```

Host variable PARM_VAR still needs to be declared in the EXEC SQL DECLARE SECTION.

8.6 Migrate user-built applications

User-built (in-house developed) applications are unique in every case. There are a variety of languages used in these applications and each one can have its unique way of using APIs. In this section we describe the steps for converting user-built applications from Oracle to IDS, and we provide some examples in C/C++ and Java, which show you how to convert the database calls.

The examples included in this chapter are excerpts from the actual programs, and so they cannot be compiled and executed by themselves.

8.6.1 Converting Oracle Pro*C applications to Informix ESQL/C

ESQL/C is the appropriate programming environment for the migration of Oracle Pro*C applications. It is part of the Informix SDK product and provides a rich set of programming techniques, such as static and dynamic SQL, the full support of handling all supported IDS data types, and an interface for user-defined routines (UDRs) and user-defined types (UDTs).

In the following section, we give you some guidance on where to focus on existing application code to achieve a successful migration to the target database server. This discussion includes the following areas:

- ▶ Connection and authentication
- ▶ Declaration and usage of host variables
- ▶ Exception handling
- ▶ SQL execution status monitoring

Connecting to the database and user authentication

There is a difference in how C programs connect to the database. In Oracle, each instance (service name) can manage only one database. IDS instances can be used to manage multiple databases, and thus the database name needs to be implicitly provided by a connection statement.

Simple singleton connect

To connect to the Oracle database, specify the Oracle user and the password for that user, as shown in the following statements:

```
EXEC SQL CONNECT :user_name IDENTIFIED BY :password;  
EXEC SQL CONNECT :connectionstring
```

Users are treated differently in the IDS. For example, there are no separate user objects in the server, and the authentication is done by the database server with the help of the base operating system. Connection requests to a database are by two different methods. One, they can be done with a trusted connection to the

remote database server. This means there is an entry for the server where the client resides on the target .rhosts or hosts.equiv file. In this case there is no need for the user to specify a password. The second way is to use the connect SQL statement and specify a user and a password to use a non-trusted connection.

Example 8-3 shows the different SQL statement which can be used to establish the connection to the database server in an client application.

Example 8-3 Possible connection SQL statement for IDS

```
#Using a trusted connection
$database <database_name>
$connect to "database@dbservername";
#Using a connection with user and password specification
$connect to "database@dbservername" USER :userid USING :password;
```

Note that password needs to be declared as a host variable. You can use a hard coded user ID and database name, and you are allowed to specify them with a host variable. In the example, we used a static database name and a host variable for the user.

Multiple connections in the same client

Similar to Oracle Pro*C, Informix ESQL/C supports the specification of multiple concurrent transactions in the same client to the same time. But there are differences in the specification and in the handling of attaching the statements to the appropriate session. Example 8-4 shows the Oracle Pro*C syntax for the definition of a session and the reference of the statement to a specific session.

Example 8-4 Using a named connection for the database access in Oracle

```
char user[10] = "scott";
char password[10] = "tiger";
char db_name[20] = "database_name";

CONNECT :user IDENTIFIED BY :password
AT CONN_NAME USING :db_name
...
AT CONN_NAME SELECT col1 INTO :var1 FROM table_name
AT CONN_NAME SELECT col1 INTO :var1 FROM table_name1
```

Informix ESQL/C provides a similar way in the naming of a session, but the reference of the current statement is handled differently. The client provides a statement to specify a name for the session, as shown in Example 8-5 on page 231.

Example 8-5 Named connections in Informix ESQL/C

```
$Connect to "database_name" as "CONN_NAME" USER :user USING :password;  
$set connection "CONN_NAME";  
$SELECT col1 INTO :var1 FROM table_name;  
$set connection "CONN_NAME1";
```

Host variable declaration

Host variables are C or C++ language variables that are referenced within SQL statements. They allow an application to pass input data to and receive output data from the database server. After the application is precompiled, host variables are used by the compiler as any other C/C++ variable.

Host variables should not only be compatible with Informix SQL data types (accepted by the esql precompiler that is shipped with the Informix SDK product), but also must be acceptable for the programming language compiler.

As the C program manipulates the values from the tables using host variables, the first step is to convert the Oracle Pro*C host variable definitions to IDS data types. See Appendix A, "Data types" on page 313 for more details. Note that this mapping is one-to-many because it depends on the actual usage of data. For example, Oracle DATE data can be converted to Informix DATE, if it only stores the actual date. But it needs to be converted to Informix DATETIME YEAR to SECOND if it stores DATE and TIME.

The next step is to match IDS SQL data types with C data types. The table in Appendix A, "Data types" on page 313 shows the mapping between data types.

General definition of basic data types

Following the ANSI standard for embedded programming, all host variables in a C program need to be declared in a special declaration section. This is so that the Informix ESQL/C precompiler can identify the host variables and the data types. This is shown in the following statements:

```
EXEC SQL BEGIN DECLARE SECTION;  
    char emp_name[31];  
    bigint ret_code = 0;  
EXEC SQL END DECLARE SECTION;  
    strcpy(emp_name, "");
```

You are also allowed to use declarations that do not follow the ANSI standard, such as in the following example:

```
int some_c_variable=0;
$int some_esqlc_variable=0;
char some_c_char[100];
$char some_esqlc_char[100];
```

Define a VARCHAR data type

Within this declaration section, there are rules for host variable data types that are different from Oracle precompiler rules. The Oracle precompiler permits host variables to be declared as VARCHAR. VARCHAR[n] is a pseudo-type recognized by the Pro*C precompiler. It is used to represent blank-padded, variable-length strings. The Pro*C precompiler converts it into a structure with a 2-byte length field followed by an n-byte character array. Informix ESQL/C requires usage of standard C constructs. If you declare a VARCHAR host variable in your ESQL/C program, it will be similar to the following example:

```
$include sqlca;
main()
{
  $varchar buffer[100];
}
```

The Informix esql precompiler would substitute this construct with the following code:

```
main()
{
  /*
   * $varchar buffer[100];
   */
  #line 6 "hostvars.ec"
  char buffer[100];
}
```

Define host variables based on user-defined C structures

There are no differences in the definition of host variables based on user-defined C structures. You only have to make sure that the structure is included in the declare section for the host variables.

The definition of a host variable on Pro*C will look like the following example:

```
typedef struct user_s
    {short int userNum;
      char userName[25];
      char userAddress[40];
    } theUser_t;
EXEC SQL BEGIN DECLARE;
      theUser_t *myUser;
EXEC SQL END DECLARE SECTION;
```

The implementation in Informix ESQL/C looks similar. The only change is to have the typedef also included to the declare section to make sure that the preprocessor is able to identify the types of the members of the structure used in the subsequent SQL statements, as shown in Example 8-6.

Example 8-6 Define a host variable based on a structure in ESQL/C

```
$include sqlca;
main()
{
EXEC SQL BEGIN DECLARE SECTION;
int cust_no=110;

struct customer {
    integer customer_num;
    char lname[20];
    char fname[20];
};
typedef struct customer customer_t;
customer_t cust_var;
EXEC SQL END DECLARE SECTION;

$database stores_demo;
$select customer_num, lname, fname
into :cust_var.customer_num, :cust_var.lname,
:cust_var.fname
from customer where customer_num=:cust_no;
printf(" Output customer_num: %d , name %s fname %s \n",
cust_var.customer_num, cust_var.lname, cust_var.fname);
}
```

Using pointers in host variables

To advance the example we can also use a pointer as a host variable in the embedded SQL statement. The definition of the host variable as a pointer, and the usage is shown in Example 8-7.

Example 8-7 Using a pointer to a structure as a ESQL/C host variable

```
$include sqlca;

#include <stdio.h>
#include <malloc.h>

main()
{
    $int cust_no=110;
    $struct customer {
        integer customer_num;
        char lname[20];
        char fname[20];
    };
    $typedef struct customer customer_t;
    $customer_t *cust_var;

    cust_var=(customer_t *)malloc(sizeof(customer_t));

    $database stores_demo;
    $select customer_num, lname,fname
        into :cust_var->customer_num, :cust_var->lname,
    :cust_var->fname
        from customer where customer_num=:cust_no;

    printf(" Output customer_num: %d , name %s fname %s \n",
        cust_var->customer_num, cust_var->lname, cust_var->fname);
}
```

Oracle host variable arrays

In Pro*C programs, you can declare host variables using arrays, and then declare a cursor from which to get the results. You can then issue a fetch statement that will get all rows from the cursor into that host array.

The following code is a fragment of PRO*C that demonstrates this method:

```
EXEC SQL BEGIN DECLARE SECTION;
long int    dept_num[10];
char        dept_name[10][14];
char        v_location[12];
EXEC SQL END DECLARE SECTION;
/* ..... */

EXEC SQL DECLARE CUR1 CURSOR FOR
    SELECT DEPTNUMB, DEPTNAME
        FROM org_table
        WHERE LOCATION = :v_location;
/*..... */

EXEC SQL FETCH CUR1 INTO :dept_num, :dept_name;
```

The last statement will get all 10 rows from the cursor into arrays.

When the target database in IDS is created with ANSI, Informix ESQL/C does not support a bulk fetch into a host variable, which is defined as an array. You need to fetch each row separately using the cursor. But from a performance perspective this is not that big disadvantage. The server sends, by default, more than one row from the result set to the client, based on specified communication buffer size parameters. The rows are already available on the client and not requested from the server one by one. The above Pro*C code needs to be converted as shown in the following code:

```
EXEC SQL BEGIN DECLARE SECTION;
char        v_location[12];
int         dept_num[10];
char        dept_name[10][14];
EXEC SQL END DECLARE SECTION;
/* define just C variables */
short int   i = 0;

/* ..... */

EXEC SQL DECLARE CUR1 CURSOR FOR
    SELECT DEPTNUMB, DEPTNAME
        FROM org_table
        WHERE LOCATION = :v_location;

/*we need Fetch row by row into the host variable array */

for (i=0;i<11;i++){
    EXEC SQL FETCH CUR1 INTO :h_dept_num[i], :h_dept_name[i];
    if (sqlca.sqlcode == 100) break;
}
```

If your database is not created with ANSI mode, a fetch array can be implemented within an ESQL/C program. But the implementation is different than fetching using a host variables array. You have to use the sqlda structure that can be used for specifying relationships between local variable types and memory layout, and the target table column. Additionally, you have to specify a FetBufSize global variable. The row size of the result set describes how many rows can be returned with one fetch.

A good example for the implementation of a fetch array in an Informix ESQL/C program can be obtained from the following Web page:

<http://publib.boulder.ibm.com/infocenter/idshep/v115/index.jsp>

This site defines the Information Center home for IDS. Go to the Development tab on the left, select the ESQL/C manual, choose the Dynamic SQL Chapter, and go to *Determining SQL Statements*.

Using stored procedures in an embedded C program

In Oracle, to invoke a remote database procedure, the following statements are used:

```
EXEC SQL EXECUTE
    BEGIN
        Package_name.SP_name(:arg_in1,:arg_in2,:arg_out1);
    END;
END-EXEC;
```

The value transfer between the calling environment and the stored procedure may be achieved through arguments. You can choose one of three modes for each argument: IN, OUT, or INOUT. For example, the above stored procedure may be declared as shown in the following example:

```
CREATE PACKAGE package_name IS
    PROCEDURE SP_name(
        arg_in1 IN NUMBER ,
        arg_in2 IN CHAR(30)
        status_out OUT NUMBER);
```

When this stored procedure (SP) is invoked, values passed from the calling program will be accepted by the stored procedure correspondingly.

For the implementation of the same kind of stored procedures in IDS you have to return the out parameter instead of a specification in the parameter list at execution time.

The definition of the SP look like the following code:

```
create procedure SP_name( arg_in1 integer, arg_in2 char(30) ) returning
integer;
return 1;
end procedure;
```

Depending on how many rows the stored procedure returns, you can either execute the stored procedure directly and fetch the result into a host variable, or you can define a cursor and fetch the result set using the cursor. The following code is an example:

```
/* singleton row */
EXEC SQL execute procedure SP_name (:arg_in1, :arg_in2) into :arg_out1;

/* singleton row with cursor */
EXEC SQL declare c1 cursor for execute procedure SP_name (:arg_in1,
:arg_in2);
EXEC SQL open c1;
EXEC SQL fetch c1 into :arg_out1;
EXEC SQL close c1;
```

Default stored procedures defined in the IDS database server do not allow the OUT parameter in the parameter list. They can only be used in user-defined functions (UDF) written in C or Java provided by an external library. For an example of how to create a UDF, refer to Appendix C, “Function mapping” on page 327.

Exception handling in embedded environments

The mechanisms for trapping errors are quite similar in Oracle and IDS, with both using the same concept of separating error routines from the mainline logic. There are different **WHENEVER** statements that could be used to define program behavior when there is an error in Informix, as shown in the following code:

```
EXEC SQL WHENEVER SQLERROR GOTO error_routine;
EXEC SQL WHENEVER SQLERROR STOP;
EXEC SQL WHENEVER ERROR STOP;
EXEC SQL WHENEVER SQLWARNING CONTINUE;
EXEC SQL WHENEVER NOT FOUND CALL not_found_routine;
```

Although the **WHENEVER** statement is prefixed by **EXEC SQL**, it is not an executable statement. Instead, a **WHENEVER** statement causes the precompiler to generate code in a program to check the **SQLCODE** attribute from the **SQLCA** after each SQL statement, and to perform the action specified in the **WHENEVER** statement. **SQLERROR** means that an SQL statement returns a

negative SQLCODE indicating an error condition. SQLWARNING indicates a positive SQLCODE (except +100), while NOT FOUND specifies SQLCODE = +100, indicating that no data rows were found to satisfy a request.

A compilation unit can contain as many WHENEVER statements as necessary, and they can be placed anywhere in the program. The scope of one WHENEVER statement reaches from the placement of the statement in the file onward in the character stream of the file until the next suitable WHENEVER statement is found, or end-of-file is reached. No functions or programming blocks are considered in that analysis. For example, you may have two different SELECT statements. One must return at least one row, and the other may not return any. You will need two different WHENEVER statements, as shown in the following example:

```
EXEC SQL WHENEVER NOT FOUND GOTO no_row_error;
      EXEC SQL SELECT  address
                INTO   :address
                FROM   test_table
                WHERE  phone = :pnone_num;

.....
EXEC SQL WHENEVER NOT FOUND CONTINUE;
EXEC SQL SELECT  commis_rate
                INTO :rate :rateind
                WHERE prod_id = :prodId;
      if (rateind == -1) rate = 0.15;
.....
```

The Oracle precompiler also supports a DO clause as an action in the WHENEVER statement. There is currently no such clause in the ESQL/C precompiler, and it needs to be converted to CALL clause.

Another alternative in comparison using the WHENEVER statement, is to check SQLCODE explicitly after each EXEC SQL statement, because that allows more context-sensitive error handling.

Error messages and warnings

The SQL Communication Area (SQLCA) data structure in Informix ESQL/C is similar to the same structure of Oracle. The SQLCA provides information for diagnostic checking and event handling.

To get the full text of longer (or nested) error messages, you need the sqlglm() function, as shown in the following example:

```
sqlglm(message_buffer, &buffer_size, &message_length);
```

In the example above, `message_buffer` is the character buffer in which you want Oracle to store the error message; `buffer_size` specifies the size of `message_buffer` in bytes; Oracle stores the actual length of the error message in `*message_length`. The maximum length of an Oracle error message is 512 bytes.

IDS contains the SQL error code in the `sqlcode` member of the global `SQLCA` structure. If a further description of this error is required, you can use the function `rgetmsg()`. This function will return the message based on the provided error code. The usage of the function in a small C program looks like the following code:

```
$database stores_demo;
$select customer_num, lname, fname
        into :cust_var.customer_num, :cust_var.lname,
        :cust_var.fname
        from customer1 where customer_num=:cust_no;

printf(" Output customer_num: %d , name %s fname %s \n",
       cust_var.customer_num, cust_var.lname, cust_var.fname);

printf("%ld \n", sqlca.sqlcode);

rgetmsg((short)SQLCODE, errmsg, sizeof(errmsg));
printf(errmsg, sqlca.sqlerrm);
```

Building ESQL/C based applications

The Informix SDK is the development environment in which you have to precompile, compile, and link your final application. This product supplies you with the build utilities, the libraries needed for the successful execution of the application, and a number of code samples for different programming tasks.

Building the application based on ESQL/C requires a number of internally referenced libraries in a certain order during the link process. That is why it is not advisable to use your own link scripts. Use the `esql` utility shipped with the Informix SQL to do this task.

You can build your executable with shared libraries. This means that at execution time the application needs to know where the libraries reside. Depending on the operating system used, there are environment variables (such as `LIB_PATH`) where the linker looks for the libraries. Shared linked applications are smaller than static applications. But they require the installation of Informix SDK or Informix Connect on the target system. In addition, you can build your application as static. In this case all used functionality is statically linked to the executable.

For more information about building embedded C applications, see the *IBM Informix ESQL/C Programmer's Manual*, G229-6426.

8.6.2 Converting Oracle Java applications to IDS

For Java programmers the Informix SDK offers two APIs: JDBC and SQLJ.

JDBC is a mandatory component of the Java programming language, as defined in the Java 2, Standard Edition (J2SE™) specification. To enable JDBC applications for IDS, an implementation of the various Java classes and interfaces, as defined in the standard, is required. This implementation is known as a JDBC driver. IDS provides a fully capable JDBC Type 4 driver for client applications in UNIX and Windows environments.

SQLJ is a standard development model for data access from Java applications. The SQLJ API is defined in the SQL 1999 specification. The Informix JDBC supports both JDBC and SQLJ APIs in a single implementation. JDBC and SQLJ can interoperate in the same application. SQLJ provides the unique ability to develop using static SQL statements and control access.

The Java code conversion is rather easy. The API itself is well-defined and database-independent. For instance, the database connection logic is encapsulated in standard J2EE™ DataSource objects. The Oracle or IDS-specific things (such as user name and database name) are then configured declaratively within the application.

However, there is the need to change your Java source code regarding the following factors:

- ▶ The API driver (JDBC or SQLJ).
- ▶ The database connect string.
- ▶ Oracle proprietary SQL, such as CONNECT BY for recursive SQL or SQL syntax such as the (+) operator instead of LEFT/RIGHT OUTER JOIN. The MTK provides support here with the SQL Translator.
- ▶ Remove or simulate proprietary optimizer hints in SQL queries.

For complete information regarding the Java environment, drivers, programming, and other relevant information, consult *IBM Informix JDBC Driver Programmer's Guide*, G229-6380.

Java access methods to IDS

Informix has rich support for the Java programming environment. You can access IDS data by putting the Java class into a module in one of the following ways:

- ▶ IDS database server
 - Stored procedures (JDBC or SQLJ)
 - SQL functions or user-defined functions (JDBC or SQLJ)
- ▶ Browser
 - Applets based on JDBC (JDBC)
- ▶ J2EE Application Servers (such as WebSphere Application Server)
 - Java ServerPages (JSPs) (JDBC)
 - Servlets (SQLJ or JDBC)
 - Enterprise JavaBeans™ (EJBs) (SQLJ or JDBC)

JDBC drivers for Informix

The IBM Informix Driver for JDBC and SQLJ supports the following situations:

- ▶ All of the methods that are described in the JDBC 3.0 specifications.
- ▶ SQLJ statements that perform equivalent functions to most JDBC methods.
- ▶ Connections that are enabled for connection pooling. WebSphere Application Server or another application server does the connection pooling.
- ▶ Java user-defined functions and stored procedures
- ▶ Global transactions that run under WebSphere Application Server Version 5.0 and above.
- ▶ Support for distributed transaction management. This support implements the Java 2 Platform, Enterprise Edition (J2EE) Java Transaction Service (JTS) and Java Transaction API (JTA) specifications, which conform to the X/Open standard for distributed transactions.

JDBC driver declaration

To connect from a Java application to an Oracle database using the OCI driver, perform the following steps:

1. Import the Oracle driver.
2. Register the driver manager.
3. Connect with a user ID, the password, and the database name.

Example 8-8 shows an Oracle JDBC connection through OCI.

Example 8-8 Oracle JDBC connection

```
import java.sql.*;
import java.io.*;
import oracle.jdbc.driver.*;

class rsetClient
{
    public static void main (String args []) throws SQLException
    {
        // Load the driver
        DriverManager.registerDriver(new
oracle.jdbc.driver.OracleDriver());

        // Connect to the database
        Connection conn =
            DriverManager.getConnection
("jdbc:oracle:oci8:@oracle","uid","pwd");

        // ...
    }
}
```

It is not necessary to import a JDBC library when connecting to Informix. The registration and connection to Informix is demonstrated in Example 8-9.

Example 8-9 Informix JDBC connection

```
import java.sql.*;

class rsetClient
{
    public static void main (String args []) throws SQLException {

        // Load Informix JDBC application driver
        try
        {
            Class.forName("com.informix.jdbc.IfxDriver");
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        // Connect to the database
    }
}
```

```

// Connection parameter definition details see below
Connection conn = DriverManager.getConnection(url);
// ...
}
}

```

JDBC Type 4 connectivity URL specification

For IBM Informix Driver for JDBC and SQLJ connectivity, the `getConnection` method must specify a user ID and a password, through parameters or through property values, as illustrated in Example 8-10.

Example 8-10 getConnection syntax for Type 4 connectivity

```

getConnection(String "url; user; password;");

```

For the URL definition using the Informix JDBC driver, specific parameters are required. The URL has to be headed with the literal `jdbc:informix-sqli://`. After this literal, specify the machine name of the database server and the port number for the IDS listener thread. Finally, the database server needs to be specified.

Example 8-11 shows you how to set a complete URL string followed by the user name and password specification in the user and password parameters.

Example 8-11 Setting the user ID and password in the user and password parameters

```

// Set URL for data source
String
connection="jdbc:informix-sqli://machine:1533:informixserver=dbservername;user=informix;password=passwd;
Connection con = DriverManager.getConnection(url);

```

Stored procedure calls

The handling of input and output parameters in stored procedure calls differs between Oracle and Informix. The following examples explain the different types of procedure calls, and the usage of parameters and result sets.

Stored procedure with an input parameter

Assume a stored procedure has been created in Oracle as in the following code:

```

CREATE OR REPLACE PROCEDURE sp_testcall_1( parm1 IN INTEGER
                                           ,parm2 IN INTEGER)

```

And in IDS as the following code:

```

CREATE PROCEDURE sp_testcall_1( parm1 INTEGER , parm2 INTEGER )

```

The procedures have two input parameters and no output parameters. There is no difference in the call between Oracle and Informix. In both cases the parameter values need to be set before the stored procedure can be executed. Example 8-12 demonstrates this point.

Example 8-12 Java call of Oracle or Informix procedure with input parameter

```
String SP_CALL = "{call sp_testcall_1(?,?)}";

// Connect to the database
Connection conn =
    DriverManager.getConnection (url);

CallableStatement stmt;
try {
    stmt = conn.prepareCall(SP_CALL);
    stmt.setInt(1,10);
    stmt.setInt(2,15);
    stmt.execute();
    // ...
}
```

Stored procedure with a result set

The next example shows a procedure without an input parameter, but defines a result set as an output parameter. The result set is an opened cursor defined in the procedure. The rows are fetched in the Java application with a loop.

The Oracle stored procedure is defined as in the following code:

```
TYPE CursorType IS REF CURSOR;
CREATE PROCEDURE sp_testcall_3(oCursor OUT CursorType) AS
BEGIN
    open oCursor for select last_name from employees;
END;
```

The output parameter type is registered as CURSOR before the procedure is called. Handling the result for the cursor specified as an output parameter in JDBC is shown in Example 8-13.

Example 8-13 Java call to Oracle procedure: Result set specified in output parm

```
String SP_CALL = "{call sp_testcall_3}";

// Connect to the database
Connection conn =
    DriverManager.getConnection (url, userName, password);
```

```

try {
    CallableStatement stmt = conn.prepareCall(SP_CALL);
    stmt.registerOutParameter (1, OracleTypes.CURSOR);
    stmt.executeUpdate();
    ResultSet rs = (ResultSet) stmt.getObject(1);
    while(rs.next())
    {
        // ...
    }
}

```

The implementation of the appropriate SP in IDS looks different. The values of the current row are returned. The return is defined with the RESUME key word indicating that the cursor in the procedure remains open and the next call of the procedure will fetch the next row within the cursor. The corresponding Informix procedure code looks like that shown in Example 8-14.

Example 8-14 SP definition in IDS

```

create procedure sp_testcall_3 ( ) returning char(30);
define lname char(30);

FOREACH cursor1 FOR
    Select last_name into lname from employees;
    RETURN lname WITH RESUME;
END FOREACH
END procedure ;

```

With Informix JDBC, you do not need to register the result set with the method registerOutParameter() in the Java application. To get the result set, call the method getResultSet() instead of getObject(), as in Example 8-15.

Example 8-15 Java call of Informix procedure with result set

```

String SP_CALL = "{call sp_spcall_v3}";

// Connect to the database
Connection conn = DriverManager.getConnection (url);

try {
    CallableStatement stmt = conn.prepareCall(SP_CALL);
    ResultSet rs = null;
    stmt.executeQuery();
    rs = stmt.getResultSet();
}

```

```

        while(rs.next())
        {
            // ...
        }
    }

```

Stored procedure with an input parameter and result set

Example 8-16 is a combination of Example 8-12 on page 244 and Example 8-14 on page 245. Note the numbering of the parameters. The first input parameter, `value2`, is numbered with 2, the result set `rs` is numbered with 1.

Example 8-16 Java call to Oracle procedure with input parameter and result set

```

private static final String SP_CALL = "{call sp_testcall4 (?) }";

CallableStatement stmt1 = conn.prepareCall(SP_CALL);

stmt1.registerOutParameter(1, OracleTypes.CURSOR);
stmt1.execute();
ResultSet rs = (ResultSet) stmt1.getObject(1);

while(rs.next()) {
    int value1 = rs.getInt(1);
    stmt2.setInt(2, value2);
    stmt2.execute();
    ResultSet rs = (ResultSet) stmt1.getObject(1);
    // ...
}

```

In Informix JDBC, input parameters and result sets are defined as shown in Example 8-17. Input parameters are numbered, beginning with 1, and are independent from the retrieval of result sets.

Example 8-17 Java call to Informix procedure with input parameter and result set

```

String SP_CALL = "{call sp_spcall_v4(?)}";

Connect to the database
    Connection conn = DriverManager.getConnection (url);

try {
    CallableStatement stmt = conn.prepareCall(SP_CALL);
    stmt.setInt(1, emp_id);
    ResultSet rs = null;
    stmt.executeQuery();
}

```

```

        rs = stmt.getResultSet();
        while(rs.next())
        {
            System.out.println (rs.getString (1));
            // ...
        }
    }
}

```

Stored procedure converted from a function

Calling an Oracle function is similar to calling a stored procedure with an input parameter and a result set. The function is defined as shown in the following code:

```

CREATE TYPE CursorType IS REF CURSOR;
CREATE FUNCTION sp_testcall_4(v_num IN INTEGER)
    RETURN CursorType

```

Example 8-18 show a sample .NET implementation retrieving the result set returned by the Oracle function.

Example 8-18 Java call to Oracle function with input parameter and result set

```

String SP_CALL = "{? = call sp_testcall_4(?)}";

// Connect to the database
Connection conn =
    DriverManager.getConnection (url, userName, password);

try {
    CallableStatement stmt = conn.prepareCall(SP_CALL);
    stmt.registerOutParameter (1, OracleTypes.CURSOR);
    stmt.setInt(2, 6);
    stmt.execute();
    ResultSet rs = (ResultSet) stmt.getObject(1);
    while(rs.next())
    {
        // ...
    }
}

```

The IDS function would look similar to the appropriate procedure definition. As in the case of the definition of a procedure, the cursor is defined in the function. The current row values are returned and the cursor remains open.

The next call of the function in the same result set will retrieve the next row as shown in the following example:

```
create function sp_testcall_3 ( cust_no integer) returning char(30);
define lname char(30);
FOREACH cursor1 FOR
    Select last_name into lname from customer where
        customer_num=cust_no;
    RETURN lname WITH RESUME;
END FOREACH
END function ;
```

The appropriate Java code is the same as the program we discussed for the procedure returning a result, as shown in Example 8-18 on page 247.

8.6.3 Converting Oracle Call Interface (OCI) applications

You may want to consider rewriting applications that use the Oracle Call Interface (OCI) by using CLI or ODBC. The OCI is specific to the Oracle database and cannot be used with any other databases.

In most cases, you can replace OCI functions with the appropriate CLI or ODBC functions, followed by relevant changes to the supporting program code. The remaining non-OCI program code should require minimal modification. The examples in this section show a comparison of the OCI and CLI or ODBC statements required for establishing a connection to an Oracle and Informix database.

Introduction to CLI

Informix Call Level Interface (Informix CLI) is the IBM callable SQL interface to the Informix database servers. It is shipped as a separate product. The current version of this product is 2.8. It is a C and C++ API for relational database access that uses function calls to pass dynamic SQL statements as function arguments. It is an alternative to embedded dynamic SQL, but does not require a precompiler.

Informix CLI is based on the Microsoft Open Database Connectivity (ODBC) specification, and the International Standard for SQL/CLI. These specifications were chosen as the basis for the Informix Call Level Interface in an effort to follow industry standards, and to provide a shorter learning curve for application programmers already familiar with either of these database interfaces. In addition, some Informix-specific extensions have been added to help the application programmer specifically exploit IDS features.

Possible client architectures using Informix CLI

There are two different architectures for the Informix CLI client. In one, the client uses the interface provided by a driver manager. In this case the application is loading at execution time a library called the driver manager. This manager then loads the appropriate library (CLI library for the target DSN (Informix database server)). On UNIX you can use the Informix CLI provided driver manager or any external provided manager providing the same functionality. On Windows you can use the Microsoft ODBC Manager for the invocation of the Informix CLI driver.

You can also directly invoke the Informix CLI driver to your application. In this architecture there is no need to invoke the ODBC driver manager.

Setting up the Informix CLI

There are differences in the setup of the Informix CLI on Windows and UNIX. In Windows you will commonly use the ODBC Administrator provided by Microsoft to create a data source for an IDS database using the Informix CLI driver.

On UNIX you have to specify the data source in a File named `.odbc.ini`. The content of the file has to look like the following code:

```
[stores_demo]
Driver=Informix
Database=stores_demo
GetDBListFromInformix=1
HostName=srv
LogonID=informix
Password=123456
Protocol=onsoctcp
ServerName=on1150FC1soc
Service=1500
DriverODBCVer=03.51
```

Additionally, a file `odbcinst.ini` is needed to specify the driver location. The file content could look like the following code:

```
[Informix]
Description=Informix
Driver=/sqldists/350/lib/cli/libifcli.so
APILevel=1
DriverODBCVer=03.51
FileUsage=1
SQLLevel=1
smProcessPerConnect=Y
```

Necessary changes to the OCI database interface

All Oracle Call Interface (OCI) calls in your application need to be changed to CLI calls. The program flow is retained, but you need to modify the definition and processing of database handles. There may not be an exact match in the conversion process. Your program code might require additional revisions to obtain similar functionality.

Global Area Pointers

Another difference between Informix CLI and Oracle OCI, is the use of pointers to three global areas by IDS, versus the Oracle use of pointers to three different, but similar, global areas. The Oracle OCI calls that use HDA (Handle Data Area), LDA (Logon Data Area) and CDA (Cursor Data Area) pointers must be translated over to Informix CLI calls that use HENV (Environment Area), HDBC (Database Connection Area) and HSTMT (Statement Area) pointers, respectively, in conjunction with an ODBC interface.

Fetch cycle

Another difference between Informix CLI and Oracle OCI calls is that the overall fetch cycle is different for each and must be modified accordingly. In Oracle, the fetch cycle for selecting multiple rows using a cursor area involves parsing, binding, defining, fetching, and executing into a cursor data area. These must be translated to a cycle that involves preparing, binding, executing, binding, and fetching through the For Each cycle.

Parameter bindings

In Oracle, multiple output bindings of parameters are declared as type OUTPUT in the ODEFIN() calls. In Informix, convert the calls to Informix SQLBindCol() calls and bind the output parameters as columns. Differences also exist between Informix CLI and Oracle OCI in the use of input/output bindings. In Oracle, these bindings are declared as type INPUT/OUTPUT in the ODEFIN() calls. In Informix, a round robin approach is used with three variables in the following manner: A=B, B=C, C=A. The variable (A) is declared before the SQL bindings as an input variable. The function is called using a second input variable (B) that is set to the first input variable (A). The function returns a third variable (C) that is bound as an output column. Finally, the first variable (A) is now set to the returned third variable (C) that made the first variable (A) appear as an input/output variable to the rest of the program.

Differences in the function set

The following examples show you the different SQL statements to connect to a database. In Oracle you need to define variables for the environment handles as well as the database name, user name, and password:

```
ociRC = OCILogon( env_hp,           // environment handle
                  err_hp,           // error handle
                  &svc_hp,         // service context
                  user_name,        // username
                  strlen (user_name), // length of username
                  password,         // password
                  strlen (password), // length of password
                  db_name           // database name
                  strlen (db_name)); // length of database name
```

In Informix CLI you also need to specify the connection handle, database name, user name, and password. So, the OCI statement will be converted as in the following example:

```
cliRC = SQLConnect( *pHdbc,        // connection handle
                    db_name,       // database name
                    strlen (db_name), // length of database name
                    user_name,     // username
                    strlen (user_name), // length of username
                    password,      // password
                    strlen (password)); // length of password
```

The following classes of OCI functions have no equivalents in Informix CLI. The functionality must be implemented either in SQL or in C (or C++) directly:

► Navigational functions

```
OCIObject__()
OCICache__()
```

► Datatype mapping and manipulation functions

```
OCIColl__()
OCIDate__()
OCINumber__()
OCIStrng__
```

and so on.

However, CLI performs conversion of data between data types wherever possible.

► External procedure functions

```
OCIExtProc__()
```

Error handling and diagnostics

Diagnostics refers to dealing with warning or error conditions generated within an application. Two levels of diagnostics are returned when calling Informix CLI functions:

- ▶ Return codes
- ▶ Detailed diagnostics consisting of SQLSTATEs

Each CLI function returns the function return code for a basic diagnosis. The function `SQL_Error()` provides more detailed diagnostic information.

Table 8-1 lists the mapping of all possible return codes of Oracle OCI functions and Informix CLI functions.

Table 8-1 Return code mapping from OCI to CLI functions

OCI return code	CLI return code	Explanation
OCI_SUCCESS	SQL_SUCCESS	The function completed successfully, no additional SQLSTATE information is available.
OCI_SUCCESS_WITH_INFO	SQL_SUCCESS_WITH_INFO	The function completed successfully with a warning or other information. Call <code>SQL_Error()</code> to receive the SQLSTATE and any other informational messages or warnings. The SQLSTATE will have a class of 01.
OCI_NO_DATA	SQL_NO_DATA_FOUND	The function returned successfully, but no relevant data was found. When this is returned after the execution of an SQL statement, additional information may be available and can be obtained by calling <code>SQL_Error()</code> .
OCI_ERROR	SQL_ERROR	The function failed. Call <code>SQL_Error()</code> to receive the SQLSTATE and any other error information.

OCI return code	CLI return code	Explanation
OCI_INVALID_HANDLE	SQL_INVALID_HANDLE	The function failed due to an invalid input handle (environment, connection or statement handle). This is a programming error. No further information is available.
OCI_NEED_DATA	SQL_NEED_DATA	The application tried to execute an SQL statement but Informix CLI lacks parameter data that the application had indicated would be passed at execute time.
OCI_STILL_EXECUTING	SQL_STILL_EXECUTING	The function is running asynchronously and has not yet completed.
OCI_CONTINUE	no equivalent	

The OCI function OCIErrorGet() returns the diagnostic record according to the SQLSTATE. Within a Informix CLI application, the functions SQLError() return three items of information:

- ▶ SQLSTATE
- ▶ Native error

If the diagnostic is detected by the data source, this is the SQLCODE. Otherwise, this is set to -99999.

- ▶ Message text

This is the message text associated with the SQLSTATE.

Sample CLI database client application and further Information

You can find more information about CLI applications and development in the following resources:

- ▶ *Informix CLI Programmer's Manual, G251-0297*
- ▶ Web page:

<http://www.elink.ibm.link.ibm.com/publications/servlet/pbi.wss?CTY=US&FNC=SRX&PBL=G251-0297-00>

Sample applications using static and dynamic SQL in Informix CLI can be obtained from the Programmers manual as well. Refer to the section "Constructing an INFORMIX-CLI Application."

8.6.4 Converting ODBC applications

The Open Database Connectivity (ODBC) is similar to the CLI standard. Applications based on ODBC can connect to the most popular databases. Thus, the application conversion is relatively easy. You have to perform the conversion of database-specific items in your application, such as the following items:

- ▶ Proprietary SQL query changes
- ▶ Possible changes in calling stored procedures and functions
- ▶ Possible logical changes
- ▶ Test, roll-out, and education tasks

Your current development environment will be the same. For a more detailed description of the necessary steps, refer to 8.4.1, “Application migration planning for source owned applications” on page 223.

8.6.5 Converting Perl applications

In this section, we discuss the use of Perl for connecting to Oracle and IDS databases. We demonstrate the conversion from Oracle to IDS using some simple Perl programs.

Where to get DBD::Informix programming interface

You can download all Perl related source code and binaries, if they are not already shipped in a package with your UNIX distribution, from the following Web page:

<http://www.cspan.org>

You can obtain a download of the current DBD::Informix interface from the following Web page:

<http://search.cpan.org/~john1/DBD-Informix-2005.02/Informix.pm>

Additionally, this site provides a detailed overview of the programming techniques based on examples needed for developing applications with the Informix Perl interface and the DBD.

Define a DBD::Oracle program in Perl

We created a stored procedure and a Perl program to demonstrate the following syntactical differences between Oracle and IDS:

- ▶ Connecting to a database using Perl
- ▶ Calling a stored procedure with an input and an output parameter
- ▶ Returning an output parameter

Example 8-19 is an Oracle stored procedure called Greeting. It contains an input parameter name, and an output parameter message.

Example 8-19 Oracle stored procedure: Greeting

```
CREATE OR REPLACE PROCEDURE Greeting (name IN
    VARCHAR2, message OUT VARCHAR2)
AS
    name2 varchar2(30);
BEGIN
    name2 := UPPER(name);
    message := 'Hello ' || name2 || ', the date is: ' ||
        SYSDATE;
END;
```

Example 8-20 shows the Perl program oraCallGreeting.pl, which connects to the Oracle database, binds the input and output parameters, executes the call to the Greeting stored procedure, and returns the output parameter.

Example 8-20 Oracle Perl program oraCallGreeting.pl

```
#!/usr/bin/perl
use DBI;

$database='dbi:Oracle:xp10g';
$user='sample';
$password='sample';

$dbh = DBI->connect($database,$user,$password);
print " Connected to database.\n";

$name = 'Oracle';
$message;

$sth = $dbh->prepare(q{
    BEGIN
        Greeting(:name, :message);
    END;
});

$sth->bind_param(":name", $name);
$sth->bind_param_inout(":message", \$message, 100);
$sth->execute;
print "$message", "\n";

# check for problems ...
warn $DBI::errstr if $DBI::err;

$dbh->disconnect;
```

Converting the Perl application to Informix

In this section, we demonstrate how to connect to IDS using Perl.

Example 8-21 is a Informix stored procedure that has the same function and same name as the Oracle stored procedure shown in Example 8-19 on page 255. The procedure Greeting also contains an input parameter name and returns the result as an VARCHAR generated by the stored procedure in the local message variable.

Example 8-21 Informix stored procedure: Greeting

```
dbaccess << EOF
database stores_demo;
drop PROCEDURE Greeting ;
CREATE PROCEDURE Greeting (name VARCHAR(30)) returning varchar(200)
  define timeofday DATE;
  define message varchar(200);

  select today into timeofday from systables where tabid=1;
  let message = 'Hello ' || ltrim(UPPER(name)) || ', the date is: ' ||
               timeofday || '.';
  return message;
END procedure;

execute procedure Greeting ( "User");
EOF
```

Minor changes are necessary to convert the preceding Oracle Perl application (Example 8-20 on page 255) to use Informix DBD::Informix. The steps (besides entering the correct values for user and password) are as follows:

- ▶ Observing the syntax difference in the parameters for the connect method, and making the necessary changes.
- ▶ Observing the syntax differences for calling stored procedures, and making the necessary changes.

DBD::Informix Connect method syntax

The syntax for a database connection to IDS is shown in Example 8-22.

Example 8-22 Generic syntax for a Informix connection string in a Perl application

```
$dbhandle = DBI->connect("dbi:Informix:$database");
$dbhandle = DBI->connect("dbi:Informix:$database", $userID, $password);
```

The parameters of this connection are as follows:

- ▶ `dbhandle`
This represents the database handle returned by the connect statement.
- ▶ `database`
Specify the name of the database to be connected to
Oracle requires the `sid` for the database in the place where Informix would require `database`; the Oracle syntax can be summarized as `dbi:oracle:sid`. In our example this is coded as `dbi:oracle:xp10g`.
- ▶ `userID`
This represents the user ID used to connect to the database.
- ▶ `password`
This represents the password for the user ID that is used to connect to the database.

The IDS database server name is specified in the `INFORMIXSERVER` environment variable taken from the user environment where the Perl application is started. This is similar to how specifications are made in the `INFORMIXDIR` environment variable, for connection definition files such as `sqlhosts`. In case no default Informix environment variables are set, the environment from the build is taken. Set the environment variable `LD_LIBRARY_PATH` or equivalent shared library location parameter for the load utility to locate the communication libraries needed to set up the communication between the Perl client and the database server.

Syntax for calling a stored procedure in Perl

In Oracle, a stored procedure is called from an anonymous block, that is, `BEGIN...END`; within a `PREPARE` statement. The input and output parameters of the Oracle stored procedure are defined as host variables, for example, `::name`, `::message`. Example 8-23 demonstrates these points.

Example 8-23 Calling a stored procedure in an Oracle Perl program

```
$sth = $dbh->prepare(q{
    BEGIN
        Greeting(:name, :message);
    END;
});
```

In contrast, an Informix stored procedure is executed by issuing a execute procedure statement from within a PREPARE statement. Also, the stored procedure input parameter is designated as parameter markers (?). This is shown in Example 8-24.

Example 8-24 Calling a stored procedure in a Informix Perl program

```
$sth = $dbh->prepare(" execute procedure Greeting(?)");
$sth->bind_param(1,$name);
$sth->execute;
```

The prepare and execute methods are used to handle SQL statements that return result sets. The parameter binding is used because of handling the call of the SP dynamic. In case you apply statements that are static and do not return result sets, you can also use the do method executing the statement. Example 8-25 show the usage of the do method in Informix DBD for Perl.

Example 8-25 Perl routines for executing static SQL

```
$stmt = "execute procedure withoutresultset('inputonly')";
$dbh->do($stmt);
$stmt ="create table newtable ( column1 integer );
$dbh->do($stmt);
```

Using DBD::Informix for a simple database client application

There is a significant difference in comparison to coding of the Oracle stored procedure. The output parameter used in Oracle has to be returned by the Informix stored procedure. It cannot be specified in the parameter list. As a result of this, the Oracle-based Perl program in Example 8-26 is changed in that the stored procedure takes now only one input parameter. The execution of the procedure, because it returns a value, has to be treated as a cursor statement. This means the result returned by the SP has to be fetched. After that, the result can be proceed further.

The complete Perl program, converted to DBD::Informix, is shown in Example 8-26.

Example 8-26 DBD::Informix Perl program ExecuteGreeting.pl

```
#!/usr/bin/perl
use DBI;

$database='dbi:Informix:stores_demo';
$user='informix';
$password='informix';
```

```

$dbh = DBI->connect($database, $user, $password) or die "Can't connect
to $database: $DBI::errstr";

print " Connected to database.\n";

$name = 'Informix';
$message;

$sth = $dbh->prepare("execute procedure Greeting(?)");
$sth->bind_param(1,$name);

$sth->execute;
$ref=$sth->fetch();
for $row (@$ref) {
    print "$$row[0]\n";
}

# check for problems...
warn $DBI::errstr if $DBI::err;

$sth->finish;

$dbh->disconnect;

```

8.6.6 Converting PHP applications

PHP is a commonly used scripting based language in Web server environments. It contains in most environments a core system represented as a library which is attached to the Web server processes. In case PHP is invoked in the HTML of the current site of the browser, the library is invoked for parsing the script and executing the statements. To access database server you have either to use the existing languages interfaces provided by the core PHP system or download additional libraries for your preferred programming environment and attach them to the core system. The language interfaces are available in a procedural or object-oriented style.

In this section, we give you an idea of which areas you need to consider for the migration of an existing application. If you are interested in a much more detailed discussion of PHP application development, in view of the discussion about the different database interfaces available for developing application clients with IDS, refer to the IBM Redbooks publication, *Developing PHP Applications for IBM Data Servers*, SG24-7218.

PHP setup considerations

Before using PHP in your client applications you have to ensure that PHP is available on the client application side and that you have identified the target PHP database interface according to your needs or according the currently used Oracle interface.

Download and setup directions

For the installation of PHP there are two options. You can install a PHP package provided with the distribution of your operating system. All LINUX system have a core PHP system included in the distribution. Or you can download the source of the current version from the following Web page:

<http://www.php.net/downloads.php>

After that you have to build the package and install the package. You need to plug in the package to the existing Web server and attach the appropriate file type mapping to the PHP system. Depending on the Web server used, the configuration file which has to be changed is different. For Apache it is httpd.conf. If you need additional interfaces (such as Informix PDO) that are not in the currently installed distribution, you can obtain the source from the following Web page:

<http://pecl.php.net>

The location for Informix PDO database programming interface for instance is the following Web page:

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

You also have to change the php.ini file to introduce new interfaces into the PHP core. This file can be considered as a type of registry.

Existing PHP database interfaces for Informix and Oracle

Oracle supports access to Oracle databases in a PHP application through two extensions:

- ▶ OCI8
The functions in this extension allow access to Oracle 10, Oracle 9, Oracle 8, and Oracle 7 databases using the OCI. They support binding of PHP variables to Oracle placeholders, have full LOB, FILE, and ROWID support, and allow you to use user-supplied define variables. This is the preferred extension for PHP connections to an Oracle database.
- ▶ PDO_OCI
This driver implements the PHP Data Objects (PDO) interface to enable access from PHP to Oracle databases through the OCI library.

There are several PHP database interfaces supporting the access to IDS:

- ▶ Informix (ifx)
This extension is based on a native connection. The interface requires at build time the installation of Informix SDK and binds the communication libraries needed for the message flow between the server and the client. The interface is available up from PHP 3. It supports all major functionality provided by the server, and is procedurally oriented.
- ▶ unixODBC
This extension uses an external ODBC manager. Additionally, an ODBC connection library for the access to the database is required. This library is provided by Informix SQK or Connect. The interface supports a procedural interface and provides the functionality similar to ODBC interface.
- ▶ Informix PDO
This is a new object-oriented interface that requires at least the core PHP 5 version. It combines the benefits of using the object orientation with the function set of the IDS database server. It uses native connections, so it provides fast communications performance.
- ▶ ODBC PDO
This is a combination of using a ODBC manager and providing an ODBC-based programming interface with an object-oriented programming style.

Migrating existing PHP programs also includes a target interface decision. Most of the time your existing PHP scripts are grown over time and are based on procedural interfaces. Current programming style tends to use object orientation, which also requires, in addition to changing the database interface, a change of

the complete script logic. Include an inventory of which interface and style is used and which could be the target style for your new needs at the beginning of the migration.

Sample PHP application environment migration

To demonstrate some of the differences between PHP programming in Oracle and Informix, we show a sample program that demonstrates the following information:

- ▶ Connecting to a database through PHP
- ▶ Calling a stored procedure with an input parameter
- ▶ Returning an output parameter

Similar to the Perl interface, we use the same Oracle-based stored procedure, as shown in Example 8-27.

Example 8-27 Oracle stored procedure Greeting

```
CREATE OR REPLACE PROCEDURE Greeting (name IN
    VARCHAR2, message OUT VARCHAR2)
    AS
    name2 varchar2(30);
BEGIN
    name2 := UPPER(name);
    message := 'Hello ' || name2 || ', the date is: ' ||
        SYSDATE;
END;
```

Connecting to Oracle using PHP (OCI8)

Example 8-28 is the Oracle PHP program oraGreeting.php. This program connects to the Oracle database using the OCI8 extension, binds the input and output parameters, executes the call to the Greeting stored procedure, and returns the output parameter.

Example 8-28 Oracle PHP program oraGreeting.php

```
<?php
$conn = oci_connect('sample','sample') or die;

$sql = 'BEGIN Greeting(:name, :message); END;';

$stmt = oci_parse($conn,$sql);

// Bind the input parameter
oci_bind_by_name($stmt,':name',$name,32);
```

```

// Bind the output parameter
oci_bind_by_name($stmt, ':message', $message, 100);

// Assign a value to the input
$name = 'Oracle';
oci_execute($stmt);

// $message is now populated with the output value
print "$message\n";
?>

```

Connecting PHP applications to Informix

The PHP database interfaces supporting connections to IDS may differ in their requirements for specification of connection parameters. That is why we want to provide several examples to give you an idea of which type of connection you prefer in your production environment. Keep in mind that there is, for specific interfaces, a differentiation between trusted and untrusted connections as we have already seen previously in the discussion about the embedded interface. Example 8-29 shows the details.

Example 8-29 Connection strings using different PHP interfaces

```

#Informix procedural PHP interface
/* standard untrusted */
$link = ifx_connect("sysmaster@srv", "informix", "123456");
/* persistent untrusted */
$link = ifx_pconnect("sysmaster@srv", "informix", "123456");
/* standard trusted */
$link = ifx_connect("sysmaster@srv");
/* persistent trusted */
$link=ifx_pconnect("sysmaster@srv");

/* Informix PDO */
/* standard connect */
$dbh = new PDO("informix:; database=sysmaster; server=srv;",
"informix", "123456");
/* standard connect trusted user */
$dbh = new PDO("informix:; database=sysmaster; server=srv;");
/* persistent connect untrusted user */
$dbh = new PDO("informix:; database=sysmaster; server=srv;"
,"informix", "123456", array(PDO::ATTR_PERSISTENT=> true));
/* persistent connect trusted user */
230 Developing PHP Applications for IBM Data Servers
$dbh = new PDO("informix:; database=sysmaster; server=srv;"
,NULL, NULL, array(PDO::ATTR_PERSISTENT=> true));

```

```

/* unixODBC */
/*trusted*/
$cn = odbc_connect
("Driver=Informix;Server=srv;Database=sysmaster",NULL,NULL);
/* not trusted */
$cn = odbc_connect
("Driver=Informix;Server=srv;Database=sysmaster","informix","123456");
/* persistant not trusted*/
$cn = odbc_pconnect
("Driver=Informix;Server=srv;Database=sysmaster","informix","123456");
/* persistant trusted*/
$cn = odbc_pconnect
("Driver=Informix;Server=srv;Database=sysmaster",NULL,NULL);

```

As you can see, PHP distinguishes between persistent and non-persistent connections. This means that the connection may open after the PHP is finished. This is possible because the client of the database server is not the PHP script itself. It is the Web server process that attaches the PHP library. You are able to specify user and password for untrusted connections, but you also would be able to set up trusted connections, which is not really advisable in a production environment in terms of security.

oci_connect and ifx_connect

`oci_connect` takes the required and optional parameters shown in the following example (in []):

```
(string $username, string $password [, string $db [, string $charset [,
int $session_mode]])
```

Database (\$db) is an optional parameter. If the database is not specified, PHP uses the environment `ORACLE_SID` and `TWO_TASK` to determine the name of the local Oracle instance and the location of `tnsnames.ora`

If we use a mapping into a procedural informix PHP interface, `oci_connect` has to be converted to the Informix function `ifx_connect`, which takes the following required and optional parameters (in []):

```
(string database, [string username], [string password], [array
options])
```

Handling statements in PHP applications

After a successful connection you have to look at the handling of the statements in your existing application and how to move the implementation to the target PHP interface. We want to have closer look at how to execute the statements and how to bind variables for the execution.

Using parameters for preparing statements

At the time of the statement definition there are two types of statements. Either you have a complete statement or want to prepare the statement for the execution with a later binding of additional values which change in every loop. Looking at the Oracle example, you can see that the query was first parsed with parameters that do not have the appropriate values. Later, the parameters are bound to the statement. We take a closer look at how to prepare a statement for later execution. We used Informix PDO for preparing the call of the SP with one place holder for later binding in Example 8-30.

Example 8-30 Preparing statement for execution

```
/* Informix PDO example code piece */
$sph= $dbh->prepare(" EXECUTE PROCEDURE greetings ( ?);");
$name = "Informix";
$sph->bindParam(1, $name,PDO::PARAM_STR, 20);
$stmt->execute();
$name = "Oracle";
$stmt->execute();
```

There is another way of implementing the logic of executing this procedure. You can generate the complete string for the procedure call into a string. After that, you can apply the string for an immediate execution. The sample code is shown in Example 8-31.

Example 8-31 Build up the statement in a string and execute it without parameter

```
$statement="EXECUTE PROCEDURE greetings ( " . $name . " );";
$dbh->exec($statement);
```

Parameter binding

Using the '?' place holder during statement preparation requires a parameter bind before the statement can be executed. You can use the bindParam method of the statement handle. You need to specify the position, the name of the variable and the type. Depending the type, an additional length specification is required. Some possible parameter bindings are shown in Example 8-32 on page 266. Be aware that the binding has to be done before the execution of the query is issued.

Example 8-32 Specification of parameter to an already prepared statement

```
$sph->bindParam(1, $_POST["parm1"],PDO::PARAM_STR, 20);
$sph->bindParam(2, $_POST["parm2"],PDO::PARAM_STR, 20);
$sph->bindParam(3, $_POST["parm3"],PDO::PARAM_INT);
$sph->bindParam(4, $_POST["parm4"],PDO::PARAM_INT);
$sph->bindParam(5, $DEBUG,PDO::PARAM_INT);
```

Retrieving the result set

After we execute the stored procedure we have to verify the result. Differing from the OUT parameter in Oracle, in the output is returned in the result set Informix PDO and has to proceed further. In our simple example we only print out the message returned by the stored procedure. For the final code that handles the result, refer to Example 8-33.

Example 8-33 Retrieving the result returned by the stored procedure in IDS

```
$sph= $dbh->prepare(" EXECUTE PROCEDURE greetings (?);");
$name = "Informix";
$sph->bindParam(1, $name,PDO::PARAM_STR, 20);

$sph->execute();
$error=$dbh->errorInfo();
if ( $error["1"]) print_r ($error);
$row=$sph->fetch(PDO::FETCH_NUM);
if ($row)
print_r ( $row) ;

$sph->execute();
$error=$dbh->errorInfo();
if ( $error["1"]) print_r ($error);
/* fetch as an assoc array */
$row=$sph->fetch(PDO::FETCH_ASSOC);
print_r ( $row) ;
```

Cursors and array specification for the result set

In the previous example, only the Informix PDO code showed the handling of a cursor in PHP. Now we introduce an example based on a simple select query returning some values in Oracle OCI8. The code for the example, the definition of the query, and the return of the result set is shown in Example 8-34 on page 267. The fetch can be executed by using the function `oci_fetch_array`. The second parameter specifies the kind of output. In our case we used an assoc array for getting the row. Additionally, the script could use the `oci_fetch_assoc` or `oci_fetch_object` function returning results.

Example 8-34 Handling a result set in Oracle OCI8

```
<?php
$conn = oci_connect("sample", "sample");

$sql = "SELECT DEPTNUMB, DEPTNAME FROM org_table";

$stmt = oci_parse ($conn, $sql);
oci_execute ($stmt);

while ($row = oci_fetch_array ($stmt, OCI_ASSOC)) {
printf(" %s %s %s \n", $row["DEPTNUMB"], $row["DEPTNAME"]);
}
/* assoc array */
oci_execute ($stmt);
while ($row = oci_fetch_assoc($stmt)) {
printf(" %s %s \n", $row["DEPTNUMB"], $row["DEPTNAME"]);
}

?>
```

The implementation of a similar query in Informix PDO is shown in Example 8-35. You can generate the result set in different representations using assoc arrays, enumerated arrays, objects, and so on. The style of the result set is specified with the input parameter used in the fetch method for the statement object.

Example 8-35 Handling the cursor and the different ways of the result

```
<?php
$dbh = new PDO("informix:host=localhost; service=1000; database=vps;
server=srv; protocol=onsoctcp ", "informix", "123456");
if (!$dbh) { exit(); }
$stmt=$dbh->query('SELECT DEPTNUMB,DEPTNAME FROM org_table');

/* assoc Array */
$row=$stmt->fetch(PDO::FETCH_ASSOC);
if ($row)
printf(" %s %s \n", $row["DEPTNUMB"], $row["DEPTNAME"]);

/* a numerated Array */
$row=$stmt->fetch(PDO::FETCH_NUM);
If ($row)
printf(" %s %s \n", $row[0], $row[1]);

/* into an object */
```

```

$row=$stmt->fetch(PDO::FETCH_OBJ);
if ( $row)
printf(" %s %s \n",$row->DEPTNUMB,$row->DEPTNAME);

/* hybrid array structure enum and assoc */
$row=$stmt->fetch(PDO::FETCH_BOTH);
if ($row)
printf(" %s %s \n",$row["DEPTNUMB"],$row[1]);

/* hybrid array structure/object enum/assoc/object */
$row=$stmt->fetch(PDO::FETCH_LAZY);
if ( $row)
printf(" %s %s \n",$row["DEPTNUMB"],$row->DEPTNAME);

$deptnumb=0;
$deptname="";
$stmt->bindParam(1, $deptnumb, PDO::PARAM_INT);
$stmt->bindParam(2, $t_model, PDO::PARAM_STR,50);
/*bound output variable */
$row=$stmt->fetch(PDO::FETCH_BOUND);
printf(" %s %s \n",$deptnumb,$deptname);
exit;
?>

```

You can handle all cursor based statements in the same way. This is also valid for procedures returning values, as we have seen previously. SQL statements that do not return a result set, such as insert, delete, and update statements and all DDL statements only need to have exception handling and a error code checking applied.

Error handling

PHP (OCI8) provides the function `oci_error` for determining the status of the last executed database statement. Depending on the context where `oci_error` is used there are different parameter required. Either the connection handle, the statement handle or, in case the connection could not be established, no parameter. The function returns the current error status, which can proceed further, as shown in Example 8-36 on page 269.

Example 8-36 Error handling in OCI8

```
$conn = oci_connect("sample", "sample");
if (!$conn) {
    $error = oci_error();
    printf(" %d %s \n", $error["code"], $error["message"]);
}

$stmt = oci_parse($conn, "select * from nonexistingtable");
if (!$stmt) {
    $error = oci_error($conn);
    printf(" %d %s \n", $error["code"], $error["message"]);
}
```

Using PDO you are able to catch exceptions that occurred in a specific defined code area. In addition to the exception, you are also able to get the SQL error that occurred in the processing of the current object. Look at Example 8-37 for the difference between using an exception and checking an error status.

Example 8-37 Exceptions and status determination in Informix PDO

```
/* Catch an exception with Informix PDO */
try
{
    $dbh->beginTransaction();
    $dbh->beginTransaction();
}
catch (PDOException $e )
{
    printf("Error: %s \n", $e->getMessage());
}

Error: There is already an active transaction.

/* Determine an error with Informix PDO */
$stmt=$dbh->query('SELECT * FROM nonexistingtable');
$error=$dbh->errorInfo();
print_r($error);
```

```
Array
(
    [0] => 42S02
    [1] => -206
    [2] => [Informix][Informix ODBC Driver][Informix]The specified table
(nonexistingtable) is not in the database. (SQLPrepare[-206] )
)
```

8.6.7 Converting .NET applications

The supported operating systems for developing and deploying .NET Framework 1.1 applications are as follows:

- ▶ Windows 2000
- ▶ Windows XP (32-bit edition)
- ▶ Windows Server® 2003 (32-bit edition)

The supported operating systems for developing and deploying .NET Framework 2.0 applications are as follows:

- ▶ Windows 2000, Service Pack 3
- ▶ Windows XP, Service Pack 2 (32-bit and 64-bit editions)
- ▶ Windows Vista® (32-bit and 64-bit editions)
- ▶ Windows Server 2003 (32-bit and 64-bit editions)
- ▶ Windows Server 2008

Supported development software for .NET Framework applications

In addition to a Informix client, you need one of the following options to develop .NET Framework applications:

- ▶ Visual Studio 2003 (for .NET Framework 1.1 applications)
- ▶ Visual Studio 2005 (for .NET Framework 2.0 applications)
- ▶ .NET Framework 1.1 Software Development Kit and .NET Framework Version 1.1 Redistributable Package (for .NET Framework 1.1 applications)
- ▶ .NET Framework 2.0 Software Development Kit and .NET Framework Version 2.0 Redistributable Package (for .NET Framework 2.0 applications)

In addition to a Informix client, the following two options are needed to deploy .NET Framework applications:

- ▶ .NET Framework Version 1.1 Redistributable Package (for .NET Framework 1.1 applications)
- ▶ .NET Framework Version 2.0 Redistributable Package (for .NET Framework 2.0 applications)

.NET Data Providers

Informix SDK with the current version 3.50 includes three .NET Data Providers:

- ▶ Informix .NET Data Provider

A high performance, managed ADO.NET Data Provider. This is the recommended .NET Data Provider for use with IDS databases. ADO.NET database access using the Informix .NET Data Provider has fewer restrictions, and provides significantly better performance than the OLE DB and ODBC .NET bridge providers.
- ▶ OLE DB .NET Data Provider

A bridge provider that feeds ADO.NET requests to the Informix OLE DB provider (by way of the COM interop module).
- ▶ ODBC .NET Data Provider

A bridge provider that feeds ADO.NET requests to the IBM ODBC driver.

For a better illustration of the available Informix .NET data provider see Figure 8-6.

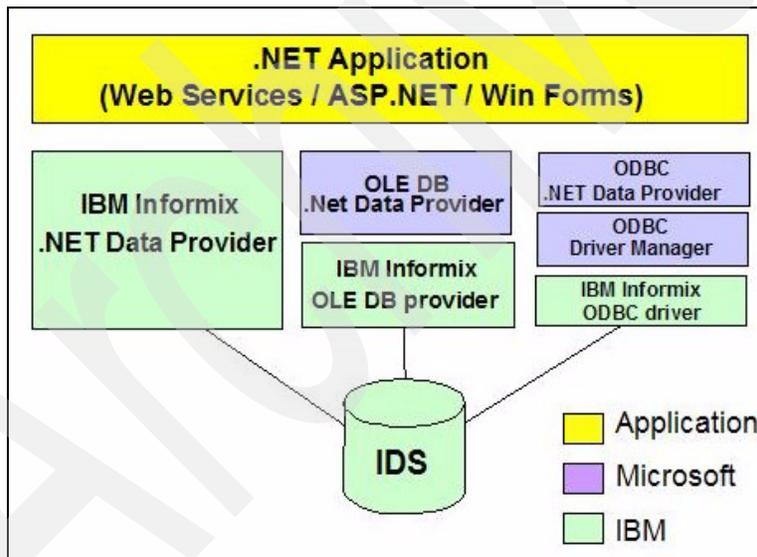


Figure 8-6 Informix .NET in the general .NET framework

Informix .NET name space

The IBM.Data.Informix name space contains the Informix .NET Data Provider. To use the Informix.NET Data Provider, you must add the Imports or using statements for the IBM.Data.Informix name space to your application .DLL, as shown in Example 8-38 on page 272.

```
[Visual Basic]
Imports IBM.Data.Informix
```

```
[C#]
using IBM.Data.Informix;
```

Also, references to IBM.Data.Informix.dll must be added to the project.

If you want to use IDS data types in your .NET application you can use the IBM.Data.IfxTypes namespace. For a complete mapping between the Informix data types and .NET data types refer to the IBM publication *IBM Data Server Provider for .NET Programmer's Guide*, SC23-7688.

VB .NET conversion example

In general, converting a .NET application from Oracle to Informix is simple. In most cases it will entail replacing the classes that are available in the Oracle .NET Data Provider with functionally equivalent classes that are available in the Informix .NET Data Provider (for example, OracleConnection with IfxConnection or OracleCommand with IfxCommand, and so on).

In this section, we demonstrate this point using a simple VB .NET application that connects to a database, executes a SELECT, and returns a result set. This example is demonstrated in Oracle and then converted to Informix. In the Informix example, the changes that are necessary for converting from Oracle to Informix are outlined.

Components of the GUI for the conversion example

The GUI, used for both examples, consists of several CONTROLS:

▶ **RUN QUERY** button

When this button is clicked, the code in the Click event will perform the following actions:

- Connect to the database
- Execute the query

▶ Query Results Text box (for aesthetic purposes only)

▶ List Box

Once the query has been executed, the results are displayed in this list box.

▶ **Quit** button

Clicking this button ends the application.

Figure 8-7 shows the GUI used to demonstrate the VB .NET application conversion example.

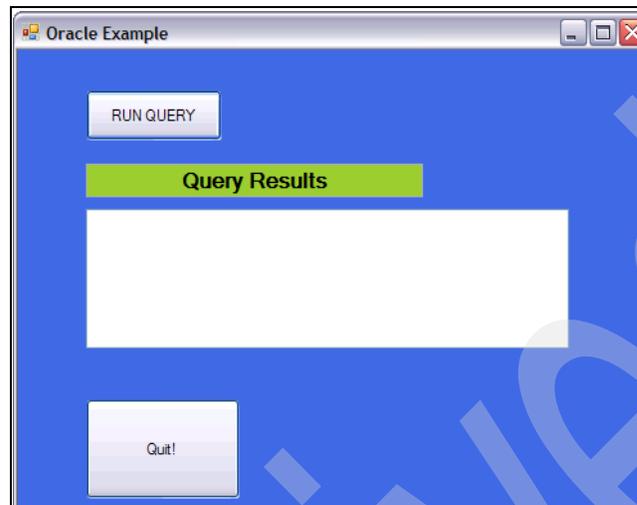


Figure 8-7 GUI for the VB .NET application conversion example

The essential components of this application are contained within the Click Event for the RUN QUERY control button. Example 8-39 shows the code in the Button1_Click event as it might appear in an Oracle application.

Example 8-39 The Button1_Click event

```
Imports Oracle.DataAccess.Client ' ODP.NET Oracle managed provider [1]

Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        Dim oradb As String = "Data Source=(DESCRIPTION=(ADDRESS_LIST=" _ +
"(ADDRESS=(PROTOCOL=TCP) (HOST=9.10.11.12) (PORT=1521)))" _ +
"(CONNECT_DATA=(SERVER=DEDICATED) (SERVICE_NAME=ora10g));" _ + "User
Id=sample;Password=sample;" [2]

        Dim conn As New OracleConnection(oradb) [3]
        conn.Open()

        Dim cmd As New OracleCommand [4]
        cmd.Connection = conn
```

```

cmd.CommandText = "select first_name, last_name from employees
                    where dept_code = 'IT'"

cmd.CommandType = CommandType.Text

Dim dr As OracleDataReader = cmd.ExecuteReader()           [5]

    While dr.Read()
        ListBox1.Items.Add("The name of this employee is: " +
dr.Item("first_name") + dr.Item("last_name"))           [6]
    End While

    conn.Dispose()

End Sub

```

Notes pertaining to Example 8-39 on page 273

- ▶ **[1]** `IMPORT Oracle.DataAccess.Client` is added to the application .DLL.
- ▶ **[2]** A String, `OraDb`, is declared as the connection string for the Oracle database.
- ▶ **[3]** A connection (`conn`) is defined as an `OracleConnection`
- ▶ **[4]** A command (`cmd`) is defined as an `OracleCommand` and populated with the text of the query.
- ▶ **[5]** A `DataReader` (`dr`) is defined as an `OracleDataReader` and the query is executed.
- ▶ **[6]** The List Box is populated with the results of the query.

When the application executes, clicking **RUN QUERY** yields the results that are shown in Figure 8-8 on page 275.

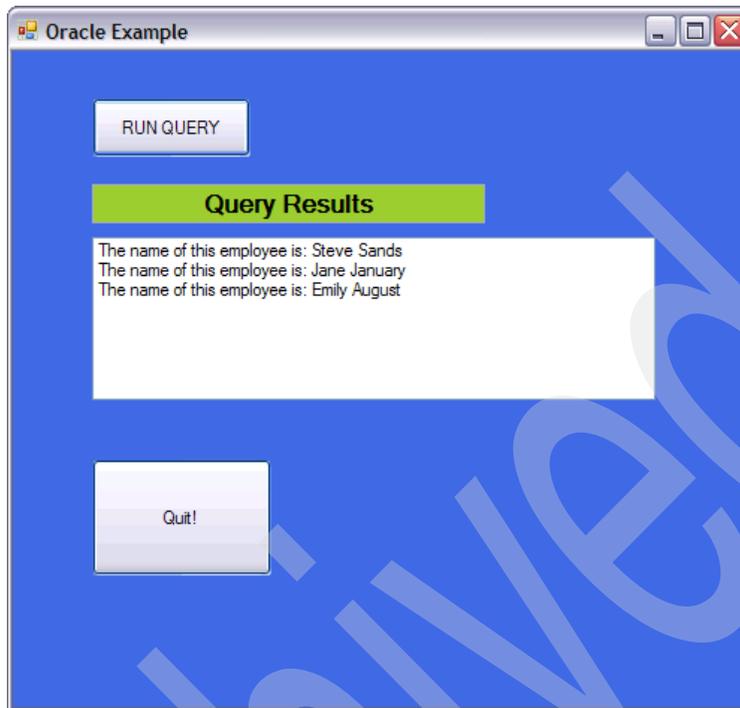


Figure 8-8 The results of the Oracle Example are displayed in the List Box

Informix example (conversion)

Because the essential components of this application are contained within the Click Event for the RUN QUERY control button, the focus of the conversion centers on this control. Example 8-40 on page 276 shows the code in the Button1_Click event as it will appear after conversion to Informix. Some explanations of the changes are documented in the Notes that appear after the code example.

Example 8-40 The code in the Button1_Click event after conversion

```
Imports IBM.Data.Informix [1]

Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
Dim ifxdb As String = [2]
"Host=localhost:Service=1526;Server=on1150soc;Database=stores.demo;User
Id=informix;Password=infomix"

Dim conn As New IfxConnection(ifxdb) [3]

conn.Open()

Dim cmd As New IfxCommand [4]
cmd.Connection = conn

cmd.CommandText = "select first_name, last_name from customer where
customer_num = '110'"

cmd.CommandType = CommandType.Text

Dim dr As IfxDataReader = cmd.ExecuteReader() [5]
    While dr.Read()

        ListBox1.Items.Add("The name of this employee is: " +
dr.Item("first_name") + dr.Item("last_name"))

    End While [6]

conn.Dispose()
End Sub
```

Notes pertaining to Example 8-40

- ▶ **[1]** To use the Informix .NET Data Provider, you must add the Imports (VB) or using (C#) statement for the IBM.Data.Informix namespace to your application .DLL.
- ▶ **[2]** A String, ifxdb, is declared and populated as the connection string for the Informix database (converted from OraDb).
- ▶ **[3]** A connection (conn) is defined as IfxConnection (converted from OracleConnection).

- ▶ **[4]** A command (cmd) is defined as lfxCommand. (converted from OracleCommand).
- ▶ **[5]** A DataReader (dr) is declared as a lfxDataReader (converted from OracleDataReader).
- ▶ **[6]** The List Box is populated with the results of the query.

For complete information about the Informix .NET provider, consult the information at the following Web page:

<http://publib.boulder.ibm.com/infocenter/idshep/v115/index.jsp>

Go to the development tab and check the IBM publication *IBM Data Server Provider for .NET Programmer's Guide*, SC23-7688.

Additionally, information with examples for specific database SQL statement programming techniques is available on IBM Developer works at the following Web pages:

<http://www.ibm.com/developerworks/data/library/techarticle/dm-0503padmanabhan/>

<http://www.ibm.com/developerworks/data/library/techarticle/dm-0510durity/index.html>

Archived



Administration of Informix Dynamic Server

In this chapter we present topics on the concepts, procedures, and reference information for database and database server administrators to use when managing IBM Informix Dynamic Server.

In this chapter, the following topics are discussed:

- ▶ Configuration and Initialization of Informix Dynamic Server
- ▶ Data Recovery and High Availability
- ▶ Admin Utilities
- ▶ Automatic Monitoring and Corrective Actions
- ▶ IDS Database Server Security

9.1 Administering the Informix database server

In this section we provide detailed information about administering the Informix database server.

9.1.1 Configuring the database server

After installing the IBM Informix Dynamic Server (IDS), it needs to be configured before it is initialized. Configuration refers to setting specific parameters that customize the database server for your data processing environment. Those include parameters for such items as volume of data, number of tables, types of data, hardware, number of users, and security needs.

Configuring a database management system requires many decisions (such as where to store the data, how to access the data, and how to protect the data). How you install and configure IDS can significantly impact the performance of database operations.

You can customize the database server so that it functions optimally in your particular data processing environment. For example, using a database server to serve 1000 users that execute frequent, short transactions is different from using a database server to serve a few users executing long and complicated queries.

The configuration parameters are stored in the ONCONFIG file. The product installation script sets the defaults for most of the configuration parameters. The `onconfig.std` template file in the `etc` subdirectory of `INFORMIXDIR` contains initial values for many of the configuration parameters. Make a copy of this template and tailor it for your specific configuration.

The template files contain initial values for many of the configuration parameters. You can use a text editor or Informix Server Administrator (ISA) to change the configuration parameters in the ONCONFIG file.

For information about the configuration parameters and how to monitor them, see *IBM Informix Dynamic Server Administrator's Reference*, G229-6360.

9.1.2 Set environment variables

To start, stop, or access a database server, each user must hold the required database access privileges, and must set the appropriate environment variables. Some environment variables are required, and others are optional.

To set the required environment variables, perform the following steps:

1. Set INFORMIXDIR to the directory where you installed the IBM Informix products.
2. Set the PATH environment variable to include \$INFORMIXDIR/bin (UNIX) or %INFORMIXDIR%\bin (Windows).
3. Set INFORMIXSERVER to the name of the database server.

Tip: Set the environment variables in the appropriate startup file for your shell file or Windows.

You can include the environment variable \$INFORMIXDIR in the ONCONFIG file, and it should be the first path name value in path name specifications.

Table 9-1 shows the environment variables that you must set before you access the database server or perform most administrative tasks.

Table 9-1 Required Environment Variables

Environment Variable	Description
INFORMIXSERVER	Specifies the name of the default database server. It has the value specified for the DBSERVERNAME or DBSERVERALIASES configuration parameter.
INFORMIXDIR	Specifies the directory where you installed your IBM Informix database server.
ONCONFIG	Specifies the name of the active ONCONFIG file. All users who use database server utilities, such as onstat, must set the ONCONFIG environment variable as follows: <ul style="list-style-type: none">▶ On UNIX: \$INFORMIXDIR/etc/onconfig▶ On Windows: %INFORMIXDIR%\etc\onconfig
JVPHOME	If using J/Foundation, this specifies the directory where you installed the IBM Informix JDBC Driver.
CLASSPATH	If using J/Foundation, this specifies the location of jvphome/krakatoa.jar file so that Java Development Kit (JDK™) can compile the Java source files.
PATH	Specifies the location of executable files. <ul style="list-style-type: none">▶ On UNIX: \$INFORMIXDIR/bin▶ On Windows: %INFORMIXDIR%\bin

9.1.3 Configure connectivity

In this section we discuss how to configure connectivity.

The sqlhosts information

The sqlhosts information, in the sqlhosts file on UNIX or the SQLHOSTS registry key on Windows, contains connectivity information for each database server. The sqlhosts information also contains definitions for groups. The database server looks up the connectivity information when the database server is started, when a client application connects to a database server, or when a database server connects to another database server.

The connectivity information for each database server includes four fields of required information and one optional field. The group information contains information in only three of its fields.

The sqlhosts file on UNIX

On UNIX, the sqlhosts file contains connectivity information. The default location of this file is \$INFORMIXDIR/etc/sqlhosts. If you store the information in another location, you must set the INFORMIXSQLHOSTS environment variable.

The five fields of connectivity information form one line in the UNIX sqlhosts file. On Windows, the database server name is assigned to a key in the SQLHOSTS registry key, and the other fields are values of that key.

The sqlhosts registry on Windows

On Windows, the HKEY_LOCAL_MACHINE registry contains the sqlhosts information. The database server installation procedure prepares the registry information. You should not edit the HKEY_LOCAL_MACHINE registry.

Table 9-2 summarizes the fields used for the sqlhosts information.

Table 9-2 Fields of SQLHOSTS file

UNIX Field Name	Windows Field Name	Description of Connectivity Information	Description of Group Information
dbservername	Database server name key or database server group key.	Database server name.	Database server group name.
nettype	PROTOCOL	Connection type	The word group

UNIX Field Name	Windows Field Name	Description of Connectivity Information	Description of Group Information
hostname	HOST	Host computer for the database server.	No information. Use a hyphen as a placeholder in this field.
servicename	SERVICE	Alias for the port number.	No information. Use a hyphen as a placeholder in this field.
options	OPTIONS	Options that describe or limit the connection.	Group options.

9.1.4 Start and administer the database server

After you install and configure the database server, you need to perform one or more of the following tasks:

- ▶ Start the database server and initialize disk space.
- ▶ Prepare to connect to applications.
- ▶ Create storage spaces.
- ▶ Set up your backup and restore system.
- ▶ Perform administrative tasks.

Starting the database server and initializing disk space

Initialization of the database server refers to two related activities:

- ▶ Shared memory initialization
- ▶ Disk space initialization

Shared memory initialization

Shared memory initialization on bringing up or starting the server establishes the contents of database server shared memory as follows: internal tables, buffers, and the shared-memory communication area. Shared memory is initialized every time the database server starts up. You use the oninit utility from the command line to initialize database server shared memory and bring the database server online. Shared-memory initialization also occurs when you restart the database server.

One key difference distinguishes shared memory initialization from disk space initialization. Shared-memory initialization has no effect on disk space allocation or layout, and no data is destroyed.

► On UNIX

To bring the database server to online mode on UNIX, enter `oninit`.

► On Windows

On Windows, the database server runs as a service. Use the Service control application to bring the database server to online mode.

Another way to initialize the database server on Windows is to use the following command, where `dbservername` is the database server name:

```
starts dbservername
```

Disk space initialization

Disk space initialization uses the values stored in the configuration file to create the initial chunk of the root dbspace on disk. When you initialize disk space, the database server automatically initializes shared memory as part of the process. Disk space is initialized the first time the database server starts up. It is only initialized thereafter during a cold restore or at the request of the database server administrator.

Important: When you initialize disk space, you overwrite what is on that disk space. If you re-initialize disk space for an existing database server, all the data in the earlier database server becomes inaccessible and is destroyed.

The database server must be in offline mode when you begin initialization. If you are starting a database server for the first time, or you want to remove all dbspaces and their associated data, use the methods in Table 9-3 to initialize the disk space and to bring the database server into online mode.

Table 9-3 Commands to start the server with disk-space initialization

Operating System	Action to Bring Database Server into Online Mode
UNIX	You must be logged in as <code>informix</code> or <code>root</code> to initialize the database server. Execute <code>oninit -iy</code> .
Windows	<p>You must be a member of the Administrators or Power Users group to initialize the database server.</p> <ul style="list-style-type: none"> ► The database server runs as a service. In the Services control application, choose the database server service and type <code>-iy</code> in the Startup parameters field. Then click Start. ► On the command line, use <code>starts dbservername -iy</code>

Attention: When you execute these commands, all existing data in the database server disk space is destroyed. Use the `-i` flag only when you are starting a new instance of the database server

You can use the `oninit -s` option to initialize shared memory and leave the database server in quiescent mode.

You should monitor the message log file referred by the `MSGPATH` configuration parameter of your `ONCONFIG` file to monitor the state of the database server. Often, the database server provides the exact nature of the problem and the suggested corrective action in the message log. For more information, see the *IBM Informix Dynamic Server Administrator's Reference*, G229-6360.

▶ For Windows Only:

When you install the database server and choose to initialize a new instance of the database server, or when you use the instance manager program to create a new instance of the database server, the database server is initialized for you.

9.1.5 Preparing to connect to applications

When the database server is online, you can connect client applications and begin to create databases. Before you can access information in a database, the client application must connect to the database server environment. To connect to and disconnect from a database server, you can issue SQL statements from the following client programs:

- ▶ DB-Access
- ▶ SQL Editor
- ▶ IBM Informix ESQL/C
- ▶ IBM Informix ODBC Driver
- ▶ IBM Informix JDBC Driver

For information about how to use client applications, refer to the following:

- ▶ *IBM Informix DB-Access User's Guide*, G229-6369.
- ▶ *IBM Informix ESQL/C Programmer's Manual*, SC23-9420.
- ▶ *IBM Informix ODBC Driver Programmer's Manual*, SC23-9423.
- ▶ *IBM Informix JDBC Driver Programmer's Guide*, SC23-9421.

9.1.6 Creating storage spaces and chunks

You are responsible for planning and implementing the storage configuration. However, be aware that the way you distribute the data on disks can affect the performance of the database server. A chunk is the same as a logical volume, logical unit, or regular file that has been assigned to the database server. The maximum size of an individual chunk is 4 terabytes, and the number of allowable chunks is 32,766. A logical storage space is composed of one or more chunks.

Tip: To take advantage of the large limit of 4 terabytes per chunk, assign a single chunk per disk drive. This data distribution can increase performance.

After the database server is initialized, you can create storage spaces such as dbspaces, blobspaces, or sbspaces. Use the onspaces utility or ISA to create storage spaces and chunks, as demonstrated in Example 9-1.

Example 9-1 Creating a dbspace named dbs1 of size 500 MB

```
onspaces -c -d dbs1 -p /dev/rdisk/c0t3d0s4 -o 0 -s 500000
```

You must create an sbpace if you are using the following functions:

- ▶ J/Foundation (to hold Java JAR files)
- ▶ Enterprise Replication (to hold spooled row data)
- ▶ Smart large objects (BLOB and CLOB data types)
- ▶ Multi-representational data types (such as DataBlades or HTML data types)

For a detailed discussion of the allocation and management of storage spaces, see *IBM Informix Dynamic Server Administrator's Guide*, G229-6359.

9.2 Data recovery and high availability

Informix Dynamic Server uses the following logging and recovery mechanisms to protect data integrity and consistency if an operating-system or media failure occurs:

- ▶ Backup and restore
- ▶ Fast recovery
- ▶ Mirroring
- ▶ High-Availability Clusters
- ▶ Enterprise Replication

9.2.1 Backup and restore

Use the ON-Bar or ontape utility to back up your database server data and logical logs as insurance against lost or corrupted data. A program error or disk failure can cause data loss or corruption. If a dbspace, an entire disk, or the database server goes down, use ON-Bar or ontape to restore the data from the backup copy. You must use the same utility for both the backup and restore.

ontape utility

If you use ontape as your backup tool, you must set up storage devices (tape drives) before you can back up and restore data. The ontape utility does not require a storage manager.

Use ontape to perform the following tasks:

- ▶ Back up and restore storage spaces and logical logs.
- ▶ Change database-logging status.
- ▶ Start continuous logical-log backups.
- ▶ Use data replication.
- ▶ Rename chunks to different path names and offsets.

ON-Bar utility

If you use ON-Bar as your backup tool, you must set up a storage manager such as the IBM Informix Storage Manager (ISM) and storage devices before you can backup and restore data.

Use ON-Bar to perform the following tasks:

- ▶ Backup and restore storage spaces and logical logs.
- ▶ Perform point-in-time restores.
- ▶ Start continuous logical log backups.
- ▶ Verify a backup with the archecker utility.
- ▶ Perform external backups and restores.
- ▶ An external backup and restore allows you to copy and physically restore data without using ON-Bar. Then you use ON-Bar for the logical restore.
- ▶ Rename chunks to different path names and offsets.

ON-Bar is packaged with IBM Informix Storage Manager (ISM), which manages data storage for the Informix database server. ISM resides on the same computer as ON-Bar and the database server. The storage manager handles all media labeling, mount requests, and storage volumes. ISM receives backup and restore requests from ON-Bar and directs data to and from storage volumes that are mounted on storage devices. It also tracks backed-up data through a data life cycle that the database or system administrator determines and also manages storage devices and storage volumes. In addition, it can backup data to as many

as four storage devices at a time. ISM stores data on simple tape drives, optical disk devices, and file systems. However, you can purchase a storage manager from another vendor if you want to use more sophisticated storage devices, backup to more than four storage devices at a time, or backup over a network.

When you plan your storage space and logical log backup schedule, make sure that the storage devices and backup operators are available to perform backups. For information about configuring and managing storage devices and media, see *IBM Informix Storage Manager Administrator's Guide*, G229-6388, or your vendor-acquired storage manager documentation.

To ensure that you can recover your databases in the event of a failure, make frequent backups of your storage spaces and logical logs. You also can verify ON-Bar backups with the archecker utility.

How often you back up the storage spaces depends on how frequently the data is updated and how critical it is. A backup schedule might include a complete (level-0) backup once a week, incremental (level-1) backups daily, and level-2 backups hourly. You also need to perform a level-0 backup after performing administrative tasks such as adding a dbspace, deleting a logical-log file, or enabling mirroring. Back up each logical-log file as soon as it fills. You can back up these files manually or automatically.

Performing a level-0 backup of all storage spaces

To perform a standard, level-0 backup of all online storage spaces and used logical logs, use the command shown in Example 9-2.

Example 9-2 Performing level-0 backup of all storage spaces

```
onbar -b  
or  
onbar -b -L 0
```

ON-Bar never backs up offline storage spaces, temporary dbspaces, or temporary sbspaces.

Important: Save your logical logs so that you can restore from this backup.

Performing a level-0 backup of specified storage spaces

To perform a level-0 backup of specified storage spaces and all logical logs (for example, two dbspaces named `fin_dbspace1` and `fin_dbspace2`), use the `-b` option as depicted in Example 9-3 on page 289. You could also specify the `-L 0` option, but it is not necessary.

Example 9-3 Performing a Level-0 Backup of Specified Storage spaces only

```
onbar -b fin_dbSPACE1 fin_dbSPACE2
```

Performing an incremental backup

An incremental backup saves all changes in the storage spaces since the last level-0 backup and performs a level-0 backup of used logical logs. To perform a level-1 backup, use the **-L 1** option, as depicted in Example 9-4.

Example 9-4 Command to perform incremental backup

```
onbar -b -L 1
```

Performing a restore

To perform a complete cold or warm restore, use the **onbar -r** command. ON-Bar restores the storage spaces in parallel. To perform a restore, use the command as in the Example 9-5.

Example 9-5 Command to perform a full system Restore

```
onbar -r
```

Restoring specific storage spaces

To restore particular storage spaces (for example, dbspaces named `fin_dbSPACE1` and `fin_dbSPACE2`), use the **-r** option, as in the Example 9-6.

Example 9-6 Command to restore specific storage spaces only

```
onbar -r fin_dbSPACE1 fin_dbSPACE2
```

For detailed information about all options of using ON-Bar and ontape, see *IBM Informix Backup and Restore Guide*, G229-6361.

9.2.2 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.

When the database server starts up it checks the physical log, which contains pages that have not yet been written to disk. If the physical log is empty, the database server was shut down in a controlled fashion. If the physical log is not empty, the database server automatically performs fast recovery. For information about fast recovery, see *IBM Informix Administrator's Guide*, G229-6359.

9.2.3 Mirroring

When you use disk mirroring, the database server writes each piece of data to two locations. Mirroring is a strategy that pairs a primary chunk of one storage space with an equal-sized mirrored chunk. Every write to the primary chunk is automatically accompanied by an identical write to the mirrored chunk. If a failure occurs on the primary chunk, mirroring lets you read from and write to the mirrored chunk until you can recover the primary chunk, all without interrupting user access to data.

It is recommended that you mirror the following data:

- ▶ Root dbspace
- ▶ Dbspaces that contain the physical log and logical-log files
- ▶ Frequently queried data

This concept of mirroring is illustrated in Figure 9-1.

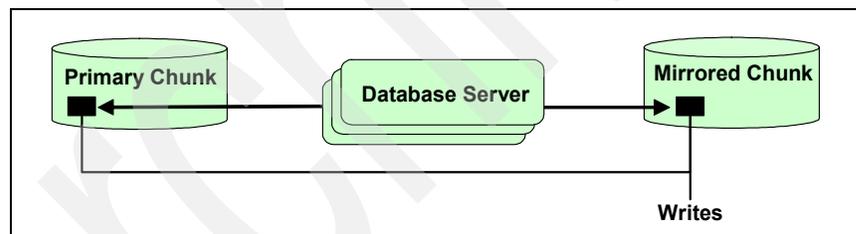


Figure 9-1 Writing data to both the primary chunk and the mirror chunk

For detailed information about mirroring, see *IBM Informix Administrator's Guide*, G229-6359.

9.2.4 Data replication

Data replication refers to the process of representing database objects at more than one distinct site, which allows an enterprise to share corporate data throughout the organization. Data replication provides a backup system in case of a catastrophic failure. Data replication configurations consist of a primary server and one or more secondary servers, such as an High Availability Data Replication (HDR) secondary server, one or more Shared Disk (SD) secondary

servers, and one or more Remote Standalone (RS) secondary servers. In addition, IDS supports IBM Informix Enterprise Replication (ER). You can combine data replication and Enterprise Replication on the same database server. For more information, see *IBM Informix Dynamic Server Enterprise Replication Guide*, SC23-6228.

High availability clusters

Data replication provides a way to duplicate database objects at more than one distinct site. IBM Informix Dynamic Server provides several high availability cluster configuration options. When you configure a set of database servers to use data replication, one database server is called the primary database server, and the others are called secondary database servers. A high availability cluster consists of a primary server and one or more secondary servers on which data from the primary server is replicated. IDS provides several secondary server configuration options and the secondary server can include any combination of the SD secondary, RS secondary, and HDR secondary servers.

High-availability data replication (HDR) secondary server

HDR provides synchronous data replication for IDS. Use an HDR secondary server if you require a hot standby. Configuring an HDR secondary server provides a way to maintain a backup copy of the entire database server that applications can access quickly in the event of a catastrophic failure of the primary server.

As Figure 9-2 illustrates, the secondary database server is dynamically updated, with changes made to the data that the primary database server manages.

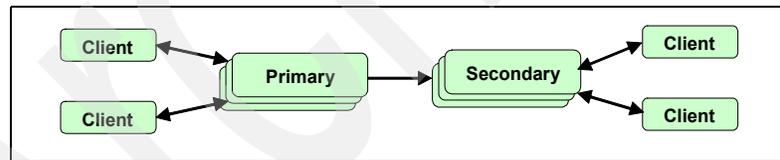


Figure 9-2 A Primary and Secondary Database Server with Replication

If one of the database servers fails, as depicted in Figure 9-3, you can redirect the clients using that database server to the other database server in the pair, which then becomes the primary server.

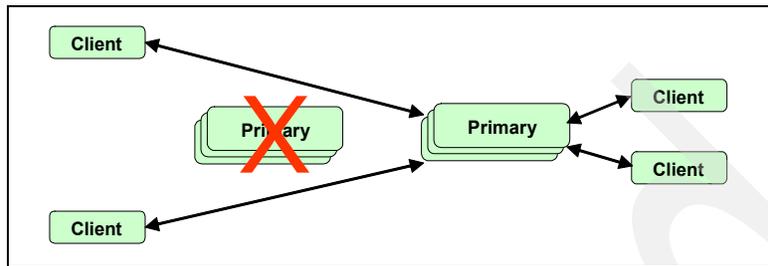


Figure 9-3 Servers and clients in a Data Replication configuration after a failure

Shared Disk Secondary server

A Shared Disk Secondary server shares disk space with a primary server. An SD secondary server does not maintain a copy of the physical database on its own disk space, rather it shares disks with the primary server.

Figure 9-4 shows an example of a primary server with two SD secondary servers. In this case the role of the primary server could be transferred to either of the two SD secondary servers. This is true whether the primary needed to be taken out of service because of a planned outage, or because of failure of the primary server.

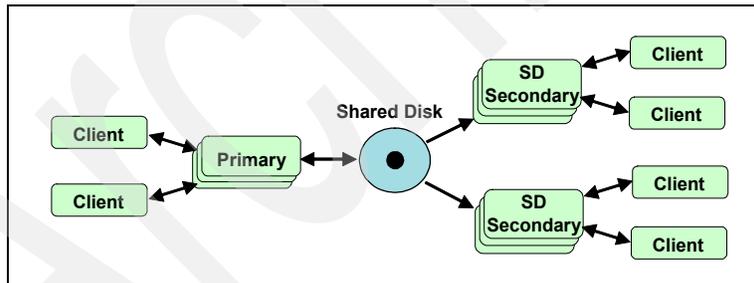


Figure 9-4 Primary Server configured with two SD Secondary Servers

Because both of the SD secondary servers are reading from the same disk subsystem, they are both equally able to assume the primary server role. Figure 9-5 illustrates a situation in which the primary server is offline.

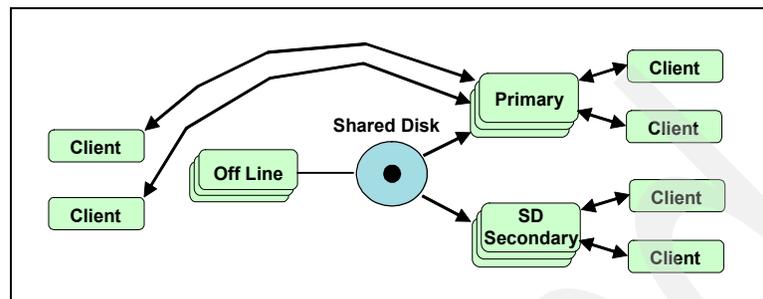


Figure 9-5 SD Secondary Server assuming role of Primary Server

Remote Standalone Secondary server

A Remote Standalone Secondary server is one that is updated asynchronously from the primary server. RSS servers can be geographically distant from the primary server, serving as remote backup servers in disaster recovery scenarios. Each RSS server maintains a complete copy of the database, with updates transmitted asynchronously from the primary server over secure network connections.

Figure 9-6 on page 294 illustrates a configuration consisting of multiple RSS servers. This configuration would be useful in a situation where the primary server is located a long distance from the RS secondary servers, or if the network speed between the primary server and the RS secondary server is slow or erratic. Because RS secondary servers use fully duplexed communication protocols, and do not require checkpoints processed in SYNC mode, the additional servers should not have a significant impact on the performance of the primary server.

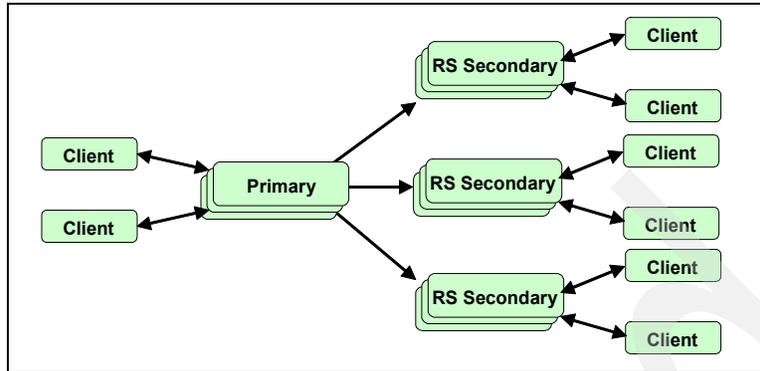


Figure 9-6 Primary server with three RS Secondary servers

Figure 9-7 illustrates an example of a configuration of an RS secondary server along with an HDR secondary server. In this example, the HDR secondary provides high availability while the RS secondary provides additional disaster recovery if both the primary and HDR secondary servers are lost. The RS secondary server can be geographically remote from the primary and HDR secondary servers so that a regional disruption, such as an earthquake or flood, would not affect the RS secondary server.

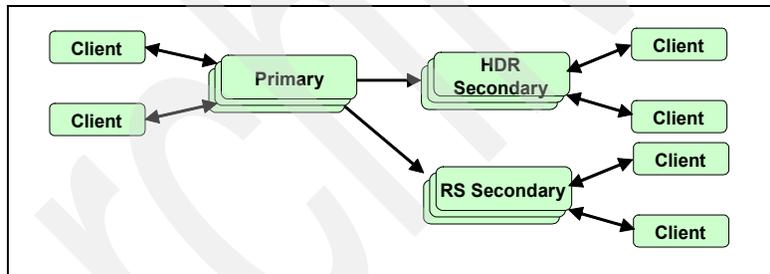


Figure 9-7 Primary Server with HDR Secondary and RS Secondary Servers

If a primary database server fails, it is possible to convert the existing HDR secondary server into the primary server, as shown in the Figure 9-8.

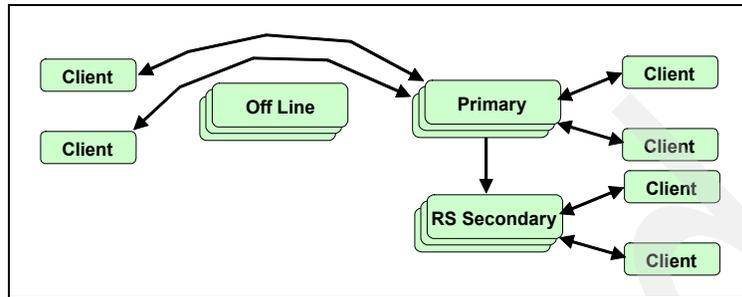


Figure 9-8 Failover of Primary Server to HDR Secondary Server

If it appears that the original primary is going to be off line for an extended period of time, then the RS secondary server can be converted to an HDR secondary server. Then, when the original primary comes back on line, it can be configured as an RS secondary server, as depicted in Figure 9-9.

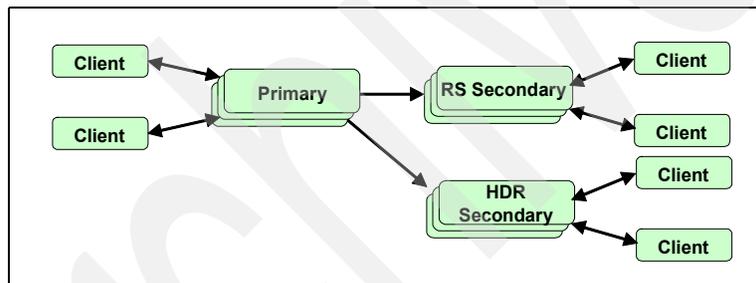


Figure 9-9 RS Secondary Server assuming role of HDR Secondary Server

Any of the previous configurations can be combined with Enterprise replication. For information about HDR secondary servers, RS secondary servers, and SD secondary servers, see *IBM Informix Administrator's Guide*, G229-6359.

Enterprise replication

Enterprise replication supports asynchronous data replication over geographically distributed database servers and allows you to replicate both entire databases and subsets of databases and tables. Enterprise replication offers limited support of user-defined data types.

Enterprise replication captures transactions to be replicated throughout the enterprise. On the source database server, Enterprise replication reads the logical log and transmits each transaction to the target database servers. At each target database server, Enterprise replication receives and applies each

transaction to the appropriate databases and tables. Enterprise replication can be combined with other data replication solutions.

For detailed information about how to setup and monitor Enterprise replication, see *IBM Informix Dynamic Server Enterprise Replication Guide*, G229-6371.

9.3 Informix Dynamic Server admin utilities

This section provides reference material for the Informix database server utilities. These utilities allow you to perform administrative tasks directly from the command line or through a GUI interface. For the complete details on each of these utilities, see *IBM Informix Dynamic Server Administrator's Reference*, G229-6360.

9.3.1 Command line utilities

In the following sections, we present utilities that allow you to perform administrative tasks directly from the command line.

onstat

The `onstat` utility provides a way to monitor database server shared memory from the command line. It reads data from shared memory and reports statistics that are accurate for the instant during which the command executes. That is, `onstat` provides information that changes dynamically during processing, including changes in buffers, locks, indexes, and users.

The `onstat` utility displays a wide variety of performance-related and status information contained within the system monitoring interface (SMI) tables. You can use the `onstat` utility to check the current status of the database server and monitor the activities of the database server.

For a complete list of all `onstat` options, use `onstat - -`. For a complete display of all the information that `onstat` gathers, use `onstat -a`.

Tip: Profile information displayed by `onstat` commands, such as `onstat -p`, accumulates from the time the database server was started. To clear performance profile statistics so that you can create a new profile, run `onstat -z`. If using `onstat -z` to reset statistics for a performance history or appraisal, ensure that other users do not also enter the command at different intervals.

Some of the `onstat` options for general information are shown in Table 9-4.

Table 9-4 Some onstat command options

onstat option	Description
onstat -b	Displays information about buffers currently in use.
onstat -l	Displays information about the physical and logical logs.
onstat -R	Displays information about buffer pools, including information about buffer pool page size.
onstat -d	Displays information of all storage spaces and the information of chunks in each storage space.
onstat -u	Displays a user activity profile that provides information about user threads including the thread owner's session ID and login name.

For the complete list of all onstat options and their explanation, see *IBM Informix Dynamic Server Administrator's Reference*, G229-6360.

oncheck

The oncheck utility displays information about the database disk configuration and usage, such as the number of pages used for a table, the contents of the reserved pages, and the number of extents in a table. For the complete list of oncheck command options and usage, see *IBM Informix Dynamic Server Administrator's Reference*, G229-6360.

The following is an example oncheck command. To obtain the physical layout of information in a chunk, execute:

```
oncheck -pe
```

The following list details the types of information that will be displayed as the result of that command:

- ▶ The name, owner, and creation date of the dbspace.
- ▶ The size in pages of the chunk, the number of pages used, and the number of pages free.
- ▶ A listing of all the tables in the chunk, with the initial page number and the length of the table in pages.

The tables within a chunk are listed sequentially. This output is useful, for example, for determining chunk fragmentation. If the database server is unable to allocate an extent in a chunk despite an adequate number of free pages, the chunk might be badly fragmented.

An example of this type of information is depicted in Figure 9-10.

```

DBSpace Usage Report:  rootdbs          Owner:  informix  Created: 08/08/2006

```

Chunk	Pathname	Size	Used	Free
1	/home/server/root_chunk	75000	19420	55580

Description	Offset	Size
RESERVED PAGES	0	12
CHUNK FREELIST PAGE	12	1
rootdbs:'informix'.TBLSpace	13	250
PHYSICAL LOG	263	1000
FREE	1263	1500
LOGICAL LOG: Log file 2	2763	1500
LOGICAL LOG: Log file 3	4263	1500
...		
sysmaster:'informix'.sysdatabases	10263	4
sysmaster:'informix'.systables	10267	8
...		

Chunk	Pathname	Size	Used	Free
2	/home/server/dbspace1	5000	53	4947

Description	Offset	Size
RESERVED PAGES	0	2
CHUNK FREELIST PAGE	2	1
dbspace1:'informix'.TBLSpace	3	50
FREE	53	4947

Figure 9-10 *oncheck -pe* Output

ondblog

The **ondblog** utility lets you change the logging mode for one or more databases. All of the activity for the **ondblog** utility is output in the `BAR_ACT_LOG` file.

If you turn on transaction logging for a database, you must create a level-0 backup of all of the storage spaces that contain data in the database before the change takes effect.

However, you cannot use the **ondblog** utility on High-Availability Data Replication (HDR) secondary servers, remote standalone (RS) secondary servers, or shared disk (SD) secondary servers.

oninit

You can run the **oninit** utility from the command line to initialize database server shared memory and to bring the database server online. If you use the **oninit -i** option, you can also initialize disk space.

Before initializing the database server, set the `INFORMIXSERVER` environment variable to the database server name that you chose when you set the configuration parameter `DBSERVERNAME`. The `INFORMIXSERVER`

environment variable is not required for initialization, but if it is not set, the database server does not build the sysmaster tables. In addition, the DB-Access utility requires the INFORMIXSERVER environment variable to be set.

The commands to take the database server offline and bring it back online are depicted in Example 9-7.

Example 9-7 Database server commands

```
onmode -ky
oninit
```

The following list details some of the prerequisites for the oninit command:

On UNIX, you must be logged in as user root or informix to run **oninit**. User informix should be the only member of the group informix.

- ▶ On Windows, IDS runs as a Windows service. Users with permissions can start and stop IDS through one of the following methods:
 - Control Panel Administrative Tools
 - The **net start** and **net stop** commands

Use the starts utility to pass command line arguments to oninit. For example, to start IDS in single-user mode, use the following command:

```
%INFORMIXDIR%\bin\starts %INFORMIXSERVER% -j
```

The onparams utility is used to modify the configuration of logical logs or physical logs. If you do not use any options, onparams returns a usage statement. Some of the options are listed in Table 9-5.

Be aware that any onparams command will fail if a storage-space backup is in progress.

onparams

Table 9-5 onparams options

onparams command	Purpose
onparams -a -d dbspace [-i]	Add a logical-log file
onparams -d -l lognum	Drop a logical-log file
onparams -p	Change the size or location of the physical log
onparams -b	Add a new buffer pool

We show examples of the `onparam` command in action in Example 9-8.

*Example 9-8 Examples of **onparams** commands*

```
onparams -a -d rootdbs -s 1000 # adds a 1000-KB log file to rootdbs

onparams -a -d rootdbs -i      # inserts the log file after the
current log

onparams -d -l 7                # drops log 7

onparams -p -d dbpace1 -s 3000 # resizes and moves physical-log to
dbpace1
```

onspaces

The `onspaces` utility is used for space management. With it, you can create or modify the `dbspaces`, `blobspaces`, `sbspaces`, or `extspaces`. We have listed some of the command options for `onspaces` in Table 9-6.

You can specify a maximum of 2047 chunks for a storage space, and a maximum of 2047 storage spaces on the database server system. The storage spaces can be any combination of `dbspaces`, `blobspaces`, and `sbspaces`.

On UNIX, you must be logged in as user `root` or user `informix` to execute `onspaces`. On Windows, you must be a member of the Informix Admin group.

You cannot use the `onspaces` utility on High Availability Data Replication (HDR) secondary servers, remote standalone (RS) secondary servers, or shared disk (SD) secondary servers.

Table 9-6 onspaces command options

onspaces Options	Purpose
-c -d	Create a dbspace
-c -b	Create a blobspace
-c -S	Create a sbspace
-a	Add a chunk to a dbspace or blobspace or sbspace
-c -x	Create an extspace
-d	Drop a blobspace, dbspace, extspace, or sbspace
-d	Drop a chunk in a dbspace, blobspace, or sbspace
-ren	Rename a dbspace, blobspace, sbspace, or extspace

onspaces Options	Purpose
-m	Start Mirroring
-r	Stop Mirroring
-s	Change status of a mirrored chunk
-f	Specify DATASKIP parameter-f

In Example 9-9 we show how to create a sbspace named vspace1 and a dbspace named dbs1, each 20 MB in size.

Example 9-9 Creating an sbspace and a dbspace

On Unix:

```
onspaces -c -S vspace1 -g 2 -p /home/informix/chunk2 -o 0 -s 20000
onspaces -c -d dbs1 -p /home/informix/chunk3 -o 0 -s 20000
```

On Windows:

```
onspaces -c -S vspace1 -g 2 -p \home\informix\chunk2 -o 0 -s 20000
onspaces -c -S dbs1 -p \home\informix\chunk3 -o 0 -s 20000
```

onmode

The onmode utility can be used for many purposes. For example, it can be used to change the database server mode, to add shared memory, and to add or remove virtual processors.

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.

Some of the most commonly used onmode options are displayed in Table 9-7.

Table 9-7 Commonly used onmode options

onmode options	Purpose
-k	Takes the database server to offline mode and removes shared memory
-m	Takes the database server from quiescent or administration mode to online mode
-s	Shuts down the database server gracefully
-u	Shuts down the database server immediately
-j	Puts the database server into administration mode
-a	Add a shared-memory segment

onlog

The onlog utility displays the contents of a logical-log file, either on disk or on backup. The onlog output is useful in debugging situations when you want to track a specific transaction or see what changes have been made to a specific tblspace.

Any user can run all of the onlog options except the -l option. Only user informix on UNIX or a member of the Informix-Admin group on Windows can run the -l option.

If the database server is in offline mode when you execute onlog, only the files on disk are read. If the database server is in quiescent or online mode, onlog also reads the logical-log records stored in the logical-log buffers in shared memory (after all records on disk have been read).

9.3.2 OpenAdmin tool for IDS

The Open Admin Tool (OAT) provides a platform-independent interface, used in a browser, to maintain your IDS database servers. It does not matter whether they are on the same machine as the OAT or on remote machines. All basic administration tasks, such as disk space management, performance monitoring, cluster management, server startup and shutdown, task scheduling and logfile analysis, can be performed with the OAT. OpenAdmin Tool for IDS is a PHP-based Web browser administration tool that can administer multiple database server instances using a single installation on a Web server. You access the Web server through any browser to administer all your database servers.

You can perform the following tasks with OAT:

- ▶ Add new connections to servers and server groups
- ▶ View server information on a map
- ▶ Customize the help system to add your own content to help topics
- ▶ Configure the display to change the sort order of reports by clicking on column headings
- ▶ Manage dbspaces, chunks, and recovery logs
- ▶ Monitor performance statistics, including recent SQL statements, combine graphs and reports to create custom reports
- ▶ View the online message log and the ON-Bar activity log
- ▶ Execute ad hoc or saved SQL statements
- ▶ View database, table, and column information
- ▶ Monitor high-availability clusters: HDR servers, Shared Disk Secondary servers, and Remote Standalone Secondary servers
- ▶ View information about executed SQL Administration API commands

Figure 9-11 contains a snapshot of a dash board view of the OAT, which provides a current view of the server memory usage and the number of transactions run.

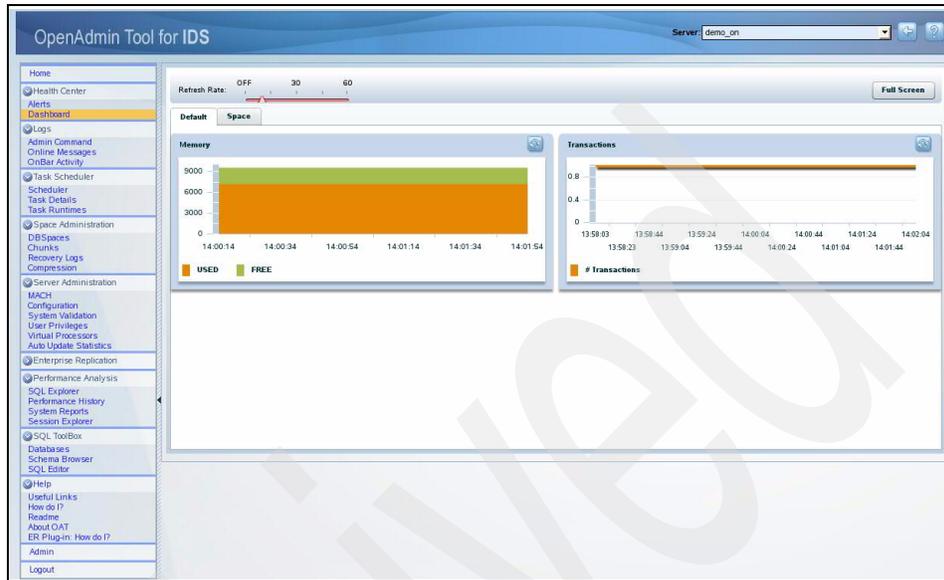


Figure 9-11 Dash board view of the OpenAdmin Tool

Figure 9-12 shows the information of all the current dbspaces of the server and their usage information. This tool allows creating or dropping spaces, adding chunks to the existing dbspaces, and many other options for space management.

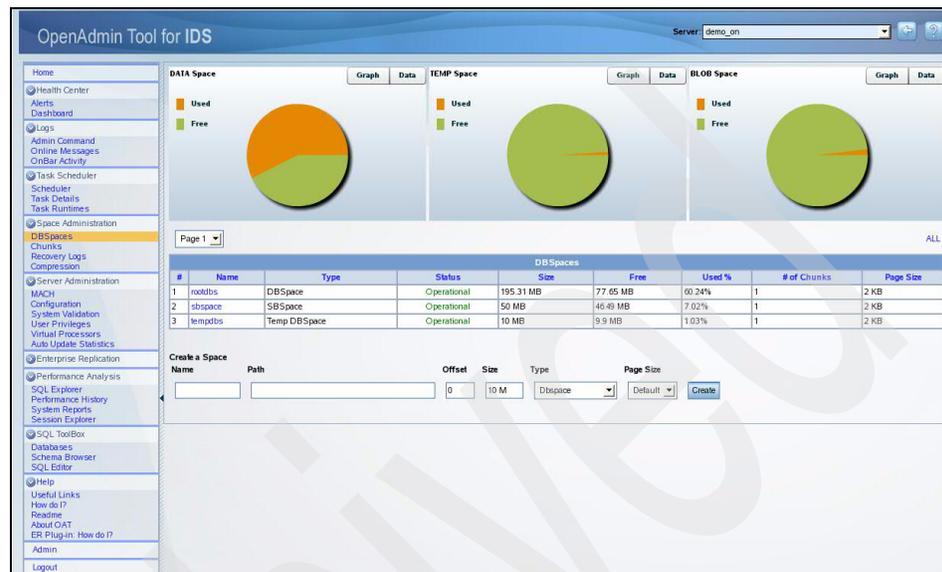


Figure 9-12 DBspaces view of the OAT

The OAT is an open-source program that you can download from the following Web page:

https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?lang=en_US&source=swg-informixfpd

Refer to IBM Redbooks publication *Informix Dynamic Server 11: Extending Availability and Replication*, SG24-7488 for details on how to install and configure the OAT.

IBM Redbooks are available at the following Web site:

<http://www.redbooks.ibm.com>

9.3.3 IBM Informix Server Administrator

IBM Informix Server Administrator (ISA) allows a DBA to manage Informix database servers by executing Informix commands from any Web browser. And you do not need to be familiar with the syntax and format of database server commands. ISA presents the command output in an easy-to-read format.

ISA is available for download at the following Web page:

<http://www.ibm.com/software/data/informix/downloads.html>

With ISA, you can perform the following database server administrative tasks:

- ▶ Change configuration parameters temporarily or permanently.
- ▶ Use Server Setup to configure or reconfigure the database server.
- ▶ Change the database server mode.
- ▶ Modify connectivity information in the sqlhosts file.
- ▶ Check dbspaces, blobspaces, and sbspaces.
- ▶ Manage logical logs and physical logs.
- ▶ Examine and modify memory usage.
- ▶ Read the message log.
- ▶ Back up and restore dbspaces, blobspaces, and sbspaces.
- ▶ Run various onstat commands to monitor performance.
- ▶ Enter SQL statements and examine database schemas.
- ▶ Add and remove chunks, dbspaces, blobspaces, sbspaces.
- ▶ Examine and manage user sessions.
- ▶ Examine and manage virtual processors (VPs).
- ▶ Use the High-Performance Loader (HPL), dbimport, and dbexport.
- ▶ Manage Enterprise Replication.
- ▶ Manage a Informix MaxConnect server.
- ▶ Set up primary and secondary database servers for High-Availability Data Replication.
- ▶ Use the following utilities: dbaccess, dbschema, onbar, oncheck, ondblog, oninit, onlog, onmode, onparams, onspaces, onstat, and onpladm.
- ▶ Enter any Informix utility, UNIX shell command, or Windows command.

9.4 Automatic monitoring and corrective actions

In this section we provide you with an overview of the components of IDS that enable you to simplify the collection of information and maintenance of the server in complex systems. You can use the Administration API, the Scheduler, information stored in the sysadmin database, and Query Drill-Down functionality to manage automatic maintenance, monitoring, and administrative tasks. Refer to *IBM Informix Security Guide*, G229-6389, for complete details.

9.4.1 Administration API

The SQL Administration API enables you to perform remote administration using various, specific SQL commands for tasks such as managing spaces, managing configuration, running routine jobs, and system validation.

You can use EXECUTE FUNCTION statements to invoke the built-in admin() or task() functions to accomplish administrative tasks that are equivalent to executing various administrative utilities of Dynamic Server.

In the EXECUTE FUNCTION statements, depicted in Example 9-10, items in the argument list specify the utility and its command-line arguments. As in SQL, the SQL statement, which is equivalent to the **oncheck -ce** command, instructs the database server to check the extents.

Example 9-10 Executing admin functions

```
EXECUTE FUNCTION admin('check extents');
```

If you want to increase the virtual memory that the database server could use in an application, the application could execute the SQL statement depicted in Example 9-11.

Example 9-11 Executing task functions

```
EXECUTE FUNCTION task('add memory', '10 MB');
```

For more information about all the Administration API commands you can use and information about defining admin() and task() functions, see *IBM Informix Guide to SQL: Syntax*, G229-6375.

9.4.2 The Scheduler

The Scheduler manages and executes scheduled maintenance, monitoring, and administration tasks. This tool enables you to monitor activities (for example, space management or automatic back up any new log data at timed intervals since the last log backup) and create corrective actions that run automatically.

The Scheduler enables the database server to execute database functions and procedures at predefined times or as determined internally by the server. The functions and procedures collect information and monitor and adjust the server, using an SQL-based administrative system and a set of tasks.

The Scheduler manages the following items:

- ▶ Tasks, which can run 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.

9.4.3 The sysadmin database

The sysadmin database contains tables that store task properties. You use the task properties to define the information that the Scheduler collects and the statements that the Scheduler runs.

The sysadmin database also contains the following items:

- ▶ The built-in task() function
- ▶ The built-in admin() function
- ▶ The command_history table, which contains information about the commands that the Administration API ran

9.4.4 Query drill-down

Query drill-down functionality provides statistical information about recently executed SQL statements, enabling you to track the performance of individual SQL statements and analyze statement history.

You can perform query drill-down to gather statistical information about each SQL statement executed on the system and analyze statement history. The query drill-down feature helps you answer questions as follows:

- ▶ 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?

The statistical information is stored in a circular buffer, which is an in-memory pseudo table called syssqltrace, that is stored in the sysmaster database. You can dynamically resize the circular buffer.

By default this feature is turned off, but you can turn it on for all users or for a specific set of users. When this feature is enabled with its default configuration, the database server tracks the last 1000 SQL statements that ran, along with the profile statistics for those statements. As the Administrative API operations occur entirely in SQL, client tools can use these features to administer the database server.

You can also use a PHP-based Web browser administration tool, the OpenAdmin Tool for IDS, to administer multiple database server instances from a single location. For detailed information refer to *IBM Informix Dynamic Server Administrator's Guide*, G229-6388.

9.5 IDS database server security

In this section we give a high-level description of the security features offered by IDS. Database security features include the following types of tasks:

- ▶ Secure server utilities
- ▶ Encrypt data across the network
- ▶ Encrypt column-level data
- ▶ Secure connections
- ▶ Control user privileges
- ▶ Control user access to data
- ▶ Auditing

9.5.1 Server utility and directory security

Informix Dynamic Server utilities and product directories are secure by default. The database server utilities make security checks before the database server starts. You can also increase directory security for some environments with the `DB_LIBRARY_PATH` configuration parameter.

When you install a new version of your database server, you should follow the installation instructions to ensure that the permissions of all key files and directories are set appropriately.

To provide increased security, key server utilities check to determine if your environment is secure. Examples of things checked, are as follows:

- ▶ The permissions on `$INFORMIXDIR` and directories under it
- ▶ The permissions on the `ONCONFIG` file
- ▶ The permissions on the `sqlhosts` file
- ▶ The length of both the filenames `$INFORMIXDIR/etc/onconfig.std` and `$INFORMIXDIR/etc/$ONCONFIG` must be less than 256 characters

If the tests for any of these conditions fail, the utilities exit with an error message.

You can also use the `DB_LIBRARY_PATH` configuration parameter to control the location from which shared objects, such as external modules, can be loaded.

Use the `DB_LIBRARY_PATH` configuration parameter to specify a comma-separated list of valid directory prefix locations from which the database server can load external modules, such as DataBlade modules.

`DB_LIBRARY_PATH` takes effect when the database server is restarted after the parameter has been set.

The `DB_LIBRARY_PATH` configuration parameter allows you to control the location from which shared objects can be loaded, and it allows you to enforce policies and standards on the formats of the `EXTERNAL NAME` clause of the `CREATE FUNCTION`, `CREATE PROCEDURE`, and `CREATE ROUTINE` statements. For more information about the `DB_LIBRARY_PATH` configuration parameter, see *IBM Informix Dynamic Server Administrator's Reference*, G229-6388.

9.5.2 Network data encryption

Use network encryption to encrypt data transmitted between server and client as well as between server and other server. Encryption is the process of transforming data into an unintelligible form to prevent the unauthorized use of the data.

Column level encryption

All values in a specific column of a database table are encrypted with the same password (word or phrase), the same encryption algorithm, and the same cipher mode. For column-level encryption, you can store the hint outside the encrypted column, rather than repeating it in every row.

You can use column-level encryption to store sensitive data in an encrypted format. After encrypting sensitive data, such as credit card numbers, only users who can provide a secret password can decrypt the data.

Use the built-in `ENCRYPT_AES()` and `ENCRYPT_TDES()` encryption functions to encrypt data in columns containing the following character data types or smart large object data types:

- ▶ CHAR
- ▶ NCHAR
- ▶ VARCHAR
- ▶ NVARCHAR
- ▶ LVARCHAR
- ▶ BLOB
- ▶ CLOB

You can also use the `SET ENCRYPTION PASSWORD` statement to set an encryption password for a session. If you do this, only users who can provide a secret password can view, copy, or modify encrypted data. Refer to *IBM Informix Dynamic Server Administrator's Reference*, G229-6388, for more information.

9.5.3 Connection security

You can prevent unauthorized connections to your database server, keep passwords secure, deploy single sign-on authentication, provide secure connections for enterprise replication and high availability clusters, keep local connections secure, and limit denial-of-service attacks. Most of these solutions can be used in conjunction with each other.

Discretionary access control

Discretionary access control verifies whether the user who is attempting to perform an operation has been granted the required privileges to perform that operation. You can perform the following types of discretionary access control:

- ▶ Create user roles to control which users can perform operations on which database objects.
- ▶ Control who is allowed to create databases by setting permission to create databases.
- ▶ Prevent unauthorized users from registering user-defined routines.
- ▶ Control whether other users besides the DBSA are allowed to view executing SQL statements.

9.5.4 Label-based access control (Enterprise Edition)

Label-based access control (LBAC) is an implementation of multi-level security (MLS) that enables you to control who has read access and who has write access to individual rows and columns of data.

MLS systems process information with different security levels, permit simultaneous access by users with different security clearances, and allow users access only to information for which they have authorization. MLS is a well-known implementation of mandatory access control (MAC). If you hold the database security administrator (DBSECADM) role in IDS, you can configure the LBAC objects to meet your security requirements. The following are examples of configuration options:

- ▶ Security policies
You attach a security policy to a table that you want to protect from unauthorized access. To create a security policy, you define security labels that determine who can access the table's data. You can have one or more security policies on your system, depending on your organization's needs.

- ▶ Security labels

You associate security labels with one or more objects in a table (data labels) and with users (user labels). When a user attempts to access an LBAC-protected table object, the system compares the user label to the data label to determine if the user can have access. If the user was not granted any label, access in most circumstances is automatically blocked.
- ▶ Security label components

These components are the building blocks of LBAC security policies. You use these components to form security policies, which, in combination with security labels, represent different user access privileges. The variety of security label components that you can create, and the flexibility that you have in constructing security policies and security labels, offers you flexibility in the way you design your organization's LBAC solution.

LBAC complements discretionary access control (DAC). When a user attempts to access a protected table, IDS enforces two levels of access control. The first level is DAC. With DAC, 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 is LBAC, which controls access at the row level, column level, or both levels. The combination of DAC privileges and LBAC-protected data access granted to a user is referred to as the user's credentials.

9.5.5 Auditing

Auditing creates a record of selected activities that users perform. IDS conforms to all regulatory compliances and ensures that all necessary details for auditing purposes are provided. IDS provides two main utilities, `onaudit` and `onshowaudit`, for auditing purposes. The `onaudit` utility is used to set the masks that specify the activities to be logged in an audit trail. You can set masks for a particular user as well. The audit trail report generated by IDS is in simple text format. It contains audit trails of all defined masks for all users. The `onshowaudit` utility is used to read this audit trail report. To make reading audit trail report easy you can use some of the options with the `onshowaudit` utility, which filters out unnecessary information from the audit trail report.

The audit administrator who analyzes the audit trail can use these records for the following purposes:

- ▶ To detect unusual or suspicious user actions and identify specific users who performed those actions
- ▶ To detect unauthorized access attempts
- ▶ To assess potential security damage
- ▶ To provide evidence in investigations, when necessary
- ▶ To provide a passive deterrent against unwanted activities, as long as users know that their actions might be audited

Refer to *IBM Informix Security Guide*, G229-6389, for the complete details on all the security aspects of Informix Dynamic Server.

Data types

In this appendix we explain data types in various environments:

- ▶ Supported SQL data types in C/C++
- ▶ Supported SQL data types in Java
- ▶ Mapping Oracle data types to Informix data types

A.1 Supported SQL data types in C/C++

Table A-1 provides a complete list of SQL data types, C and C/C++ data type mapping, and a quick description of each.

For more information about mapping between SQL data types and C and C++ data types, refer to *IBM Informix ESQL/C Programmer's Manual*, SC23-9420 available for download from the following Web page:

<http://publib.boulder.ibm.com/infocenter/idshe1p/v115/index.jsp>

Table A-1 SQL to C/C++ data type mapping

	SQL data type sqltype	C/C++ type	sqlle n	Description
integer	SMALLINT	short	2	<ul style="list-style-type: none"> ▶ 16-bit signed integer ▶ Range between (-32,768 and 32,767) ▶ Precision of 5 digits
	INTEGER INT SERIAL	long	4	<ul style="list-style-type: none"> ▶ 32-bit signed integer ▶ Range between (-2,147,483,648 and 2,147,483,647) ▶ Precision of 10 digits
	SERIAL	long	4	<ul style="list-style-type: none"> ▶ 32-bit unsigned integer ▶ Range between (1 and 2,147,483,647) ▶ Precision of 10 digits
	BIGSERIAL SERIAL8	ifx_int8_t	12	<ul style="list-style-type: none"> ▶ unsigned 64 Bit integer ▶ Range between 1 and 9,223,372,036,854,775,807
	BIGINT INT8	long long long __int64 sqlint64	8	<ul style="list-style-type: none"> ▶ 64-bit signed integer ▶ Range between -9,223,372,036,854,775,807 and 9,223,372,036,854,775,807
floating point	REAL FLOAT	float	4	<ul style="list-style-type: none"> ▶ Single precision floating point ▶ 32-bit approximation of a real number ▶ FLOAT(n) can be synonym for REAL if $0 < n < 25$

	SQL data type sqltype	C/C++ type	sqlle n	Description
	DOUBLE DOUBLE PRECISION	double	8	<ul style="list-style-type: none"> ▶ Double precision floating point ▶ 64-bit approximation of a real number ▶ Range in (0, -1.79769E+308 to -2.225E-307, 2.225E-307 to 1.79769E+308) ▶ FLOAT(n) can be synonym for DOUBLE if $24 < n < 54$
Decimal	DECIMAL(p,s) DEC(p,s) NUMERIC(p,s) NUM(p,s) MONEY	dec_t	p/2+1	<ul style="list-style-type: none"> ▶ Packed decimal ▶ If precision /scale not specified, default is (5,0) ▶ Max precision is 32 digits, and max range between (-10E31+1 ... 10E31 -1) ▶ Consider using char / decimal functions to manipulate packed decimal fields as char data
Date / Time	DATE	int	4	<ul style="list-style-type: none"> ▶ integer value similar to UNIX time
	DATETIME	dtime_t	22	
	INTERVAL	intrvl_t	24	
character	CHAR	char	n	<ul style="list-style-type: none"> ▶ - Fixed-length character string consisting of n bytes ▶ Use char[n+1] where $1 \leq n \leq 254$ ▶ If length not specified, defaults to 1
	VARCHAR	char	n	<ul style="list-style-type: none"> ▶ Null-terminated variable length character string ▶ Use char[n+1] where $1 \leq n \leq 254$
	LVARCHAR	char	len	<ul style="list-style-type: none"> ▶ Non null-terminated varying character string with 2-byte string length indicator ▶ Use char[n] in struct form where $1 \leq n \leq 32739$

	SQL data type sqltype	C/C++ type	sqlle n	Description
Binary	CLOB(n)	ifx_lo_t		<ul style="list-style-type: none"> Informix provides a defined interface handling CLOBs like ifx_lo_open, ifx_lo_read or ifx_lo_write and ifx_lo_close
	BLOB (TEXT/BYTE)	Locator structure loc_t		<ul style="list-style-type: none"> Defines the attributes of the BLOB, where he resides, filename, size and indicates a possible NULL value

A.2 Supported SQL data types in Java

Table A-2 shows the Java equivalent of each SQL data type, based on the JDBC specification for data type mappings. The JDBC driver converts the data exchanged between the application and the database using the following mapping schema. Use these mappings in your Java applications and your PARAMETER STYLE JAVA procedures and UDFs.

For more information about mapping between SQL data types and Java types refer to *IBM Informix Embedded SQLJ Users Guide*, G251-2278. You can download this Documentation from the following Web page:

<http://www-01.ibm.com/software/data/informix/pubs/library>

Table A-2 SQL data types mapped to Java declarations

	SQL data type sqltype	Java type	sqlle n	Description
Integer	SMALLINT	short	2	16-bit, signed integer
	INTEGER	int	4	32-bit, signed integer
	INT8 SERIAL8	long	4	32-bit, signed integer
	BIGINT BIGSERIAL	bigint	8	64-bit, signed integer
floating point	SMALLFLOAT	float	4	Single precision floating point

	SQL data type sqltype	Java type	sqlen	Description
	FLOAT DOUBLE PRECISION	double	4	Double precision floating point
Decimal	DECIMAL(p,s)	java.math. BigDecimal	n/2	Packed decimal
	MONEY	java.math. BigDecimal	n/2	Packed decimal
Date / Time	DATE	java.sql.Date	10	10-byte character string
	DATETIME	java.sql.Time	8	8-byte character string
	DATETIME	java.sql. Timestamp	26	26-byte character string
	INTERVAL	IntervalDF		
character	CHAR	String	n	Fixed-length character string of length n where n is from 1 to 254
	CHAR FOR BIT DATA	byte[]		Fixed-length character string of length n where n is from 1 to 254
	VARCHAR LVARCHAR	java.lang.Stri ng	n	Variable-length character string, n <= 32739
	NVARCHAR	java.lang.Stri ng	n	Variable-length character string, n <= 32739
	NCHAR	java.lang.Stri ng	n	Variable-length character string, n <= 254
	VARCHAR FOR BIT DATA	byte[]		Variable-length character string
Binary	CLOB(n)	byte[]	n	Large object variable-length character string
	BLOB(n)	byte[]	n	Large object variable-length binary string

A.3 Mapping Oracle data types to Informix data types

Table A-3 summarizes the mapping from Oracle data types to corresponding Informix data types. The mapping is one to many and depends on the actual usage of the data.

Table A-3 Mapping Oracle data types to Informix data types

	Oracle data type	Informix data type	Notes
Numeric/ floating Point	NUMBER(p)	SMALLINT INTEGER BIGINT	SMALLINT, if $1 \leq p \leq 4$ INTEGER, if $5 \leq p \leq 9$ BIGINT, if $10 \leq p \leq 18$
	NUMBER(p,s)	DECIMAL(p,s) DOUBLE PRECISION	if $s > 0$
	NUMBER	FLOAT REAL DOUBLE PRECISION	
	FLOAT FLOAT(n) DOUBLE PRECISION	DOUBLE PRECISION	
Date/Time	DATE (only the date)	DATE	Use Oracle TO_CHAR() function with the MM/DD/YYYY format string to extract data for subsequent Informix load.
	DATE (only the time)	Datetime hour to second	Use Oracle TO_CHAR() function with the HH24:MI:SS format string to extract for subsequent Informix load.

	Oracle data type	Informix data type	Notes
	TIMESTAMP	DATETIME year to fraction (5)	Use Oracle TO_CHAR() function with the YYYY-MM-DD HH24:MI:SSxFF format string to extract for subsequent Informix load. Oracle default format is DD-MON-YY
	INTERVAL	INTERVAL	
Character	CHAR(n)	CHAR(n)	1 <= n <= 254
	VARCHAR2(n) VARCHAR(n)	VARCHAR(n) LVARCHAR(n)	n <= 255 n <= 32739
	NCHAR(n)	NCHAR(n)	1 <= n <= 254
	NVARCHAR2(n)	NVARCHAR(n)	1 <= n <= 254
Large (binary) data	LONG	LVARCHAR(n)	n <= 32739 bytes
	LONG	CLOB TEXT	n > 32739 bytes
	BLOB	BLOB	n <= 4TB
	BFILE	BLOB	
	RAW	CHAR(n) VARCHAR(n) BLOB(n)	CHAR, n <= 254 VARCHAR, n <= 32739 BLOB, n <= 4TB
	LONG RAW	LVARCHAR(n) BLOB(n)/CLOB(n)	LONG, n <= 32739 (C)BLOB, n <= 4TB
	CLOB	CLOB(n)	if n <= 4TB
	NCLOB	CLOB(n)	if n <= 4TB

Archived

Terminology mapping

In this appendix we provide a mapping of the terminology used in Oracle to that used in the Informix Dynamic Server (IDS).

Table 9-8 Mapping of Oracle terminology to Informix IDS

	Oracle	Informix	Comments
Products	Oracle Enterprise Edition	Informix IDS Enterprise	
	Oracle Express Edition	Informix Express Edition	
Configuration	Instance	Instance	In an Oracle Environment the instance describes the processes and the memory. Informix IDS describes processes, Memory and the appropriate data files

	Oracle	Informix	Comments
	Database	Database	In Oracle the database describes the data files on disk. An Informix IDS instance can maintain multiple databases. Multiple databases can share the same data files.
	Server parameter files	ONCONFIG	Configuration files containing all tunable parameter making up the database (instance)
	tnsnames.ora sqlnet.ora	sqlhosts file	Config files describing the communication from client to the database server
	Shared or Dedicated Server		Define how incoming application requests are served by the Oracle database server. In Informix all client applications are served shared. No additional processes are started for new incoming client requests.
Storage	tablespace	dbspace	Logical storage unit, describes a set of existing file system objects attached to this database server
	SYSTEM tablespace	dbspace	The storage clause of the create database statement in Informix IDS describes where the system catalog is stored. There is no restriction in which dbspace a Database can be created.
	data files	chunks	Physical unit describes size and location of a database file
	Segments	Partitions	Logical unit described a set of extents and is related to certain database objects like tables or indexes

	Oracle	Informix	Comments
	Extents	Extents	Logical unit contains a amount of pages or data blocks
	data blocks	pages	smallest physical storage unit contains the real data
Memory	SGA	Shared Memory Segments	Memory attached and maintained by the processes representing the database instance. Contains e.g Pools, Buffer and Dictionary cache.
	UGA	Session Pools	Depending on Shared or Dedicated Server configuration the UGA is either in PGA or SGA. Contains all necessary session specific data. Informix IDS maintains all session specific data in separate pools for each session always in the Shared memory segments
	PGA		The listener in the oracle starts for new incoming client requests a new server process. The memory needed for session specific activities like cursors or sorts in addition with the heap is maintained in the PGA. Informix always maintains the session memory in session pools in the Shared memory
	Data(Buffer) cache	Bufferpool	Cache already read data
	Library cache	Datadictionary Procedurecache Statementcache ...	Caches contain already evaluated and currently active and valid database objects in the shared memory.
Features and Technology	Redo Logs	Logfiles	Recovery logs for rollback of open transactions in case of database recovery

	Oracle	Informix	Comments
	Undo Segments	Physical log	Contains the before images of the datablocks for open transactions. Needed for rollbacks and row versioning. Before Images in Informix IDS are maintained in the physical Log. The maintenance of the previous row versions is attached to the lock structures in USELASTCOMMITTED is enabled.
	Data Dictionary	System catalog	Metadata for the database
	Checkpoint	Checkpoint	Synchronization between instance memory and the database files, triggered by certain event like expiration of time or manually forcing a checkpoint. Dirty Buffers are written to disk. Logs are synchronized.
	Oracle Parallel processing	PDQ	Splitting a specific query in a execution tree processed in parallel in the database server
	Oracle RAC	Informix CDC (SDS, RSS)	High Availability and Application partitioning solution. Informix IDS provides with SharedDiskServer and Remote SecondaryServer similar solutions which also can be combined together and with other stand by and data replication solutions
Tools	sqlplus	dbaccess	SQL Command line interface to the database server
Programming languages and Application development	PL/SQL	SPL 4GL EGL	Programming language extension to SQL. Similar language interfaces provide the Informix Stored procedure language or the 4GL and EGL programming environments

	Oracle	Informix	Comments
	Pro*C	SDK Software Development Kit (ESQL/C)	Interface for embedding dynamic and static SQL statements into c/c++ programs
Connectivity	Oracle Gateway	Informix DRDA®	Access to a host or distributed access to DB2 (LUW)
	Database Link	ISTAR	Access from one database to another. Oracle need to define an object for distributed access. In Informix there is no need to create an extra database object. The remote database must be defined in sqlhosts file used by the connection coordinator.

Archived

Function mapping

This appendix contains the function mapping of Oracle to IDS. The function mapping covered includes the following:

- ▶ Numeric function mapping
- ▶ Character function mapping
- ▶ Date and time function mapping
- ▶ Comparison and NULL-related function mapping
- ▶ Encoding, decoding, encryption, and decryption function mapping
- ▶ Implementation of new C-based functions in IDS

Not all the Oracle functions have a direct mapping to the IDS build in function set. We want to show you at the end of this appendix how to create your own functions to achieve the mapping for your applications.

C.1 Numeric function mapping

Table C-1 shows the numeric function mapping between Oracle and Informix.

Table C-1 Numeric functions

Oracle	Informix	Comments
ABS	ABS	Returns the absolute value.
ACOS	ACOS	Returns the arc cosine.
ASIN	ASIN	Returns the arc sine.
ATAN	ATAN	Returns the arc tangent.
ATAN2	ATAN2	Returns the arc tangent (two value).
BITAND	BITAND	Returns the bit-wise AND of two non-negative integers.
CEIL	CEIL	Returns the smallest integer greater or equal to the argument.
COS	COS	Returns the cosine.
COSH	N/A Easy implementation of an C UDR possible	Returns the hyperbolic cosine. Example for the implementation in a C-UDR at the end of the appendix.
EXP	EXP	Returns the exponential function of the argument.
FLOOR	FLOOR	Returns the largest integer less or equal to the argument.
LN	LN	Returns the natural logarithm.
LOG	LOG	Returns the natural logarithm.
LOG(10,n1)	LOG10(n1)	Returns the common logarithm (base 10).
MOD	MOD	Returns the remainder of the first argument divided by the second argument.
NANVL	N/A	Returns an alternative value n1 if the input value n2 is NaN (not a number). If n2 is not NaN, then Oracle Returns n2.

Oracle	Informix	Comments
POWER®	POWER	Returns the result of raising the first argument to the power of the second argument.
REMAINDER	Use a SP or your own C function	The REMAINDER function is similar to MOD function except that it uses ROUND in its formula, whereas MOD uses FLOOR.
ROUND(arg1,arg2)	ROUND(arg1,arg2)	Rounds a value of the first argument to the number of decimal places specified by the second argument.
SIGN	N/A can be accomplished by a Stored procedure	Returns: 1 when number is negative, 0 when number is zero, or 1 when number is positive.
SIN	SIN	Returns the sine.
SINH	N/A Easy implementation of an C UDR possible	Returns the hyperbolic sine. Example for the implementation in a C-UDR at the end of the appendix.
SQRT	SQRT	Returns the square root.
TAN	TAN	Returns the tangent.
TANH	N/A Easy implementation of an C UDR possible	Returns the hyperbolic tangent.
TRUNC(n[,m])	TRUNC(n[,m])	Returns the truncated number n to the number of decimal places specified by m.
TO_NUMBER	TO_NUMBER	Converts a number or a character expression representing a number value to a DECIMAL data type.

The missing functions like cosh, sinh, tanh are quite easy to implement in the C language because the library calls into the libm library already exists. Only a mapping to a new database server function has to be done. We want to show you at the end of the appendix the steps how this can be done in IDS. We use the implementation of the cosh function in that example.

C.2 Character function mapping

Table C-2 shows the character function mapping between Oracle and Informix.

Table C-2 Character functions

Oracle	Informix	Comments
ASCII	ASCII	Returns the decimal representation of the first character in a character string, based on its codepoint in the ASCII character set.
CHR(n)	N/A Easy implementation of an C UDR possible	Returns an ASCII code that represents n. Example how to implement the function in a C-UDR at the end of the appendix.
CONCAT	CONCAT	Returns the concatenation of two strings.
INITCAP	INITCAP	Returns characters with the first letter of each word in uppercase and the reset of letters in lowercase.
INSTR	N/A Implementation as a C_UDR possible	Returns an integer indicating the position of the character in string that is the first character of this occurrence.
INSTRB INSTRC INSTR2 INSTR4	N/A. Implementation as a C_UDR possible	Returns an integer indicating the position of the character in different character sets.
LENGTH	LENGTH	Returns a length of a value.
LENGTHB LENGTHC LENGTH2 LENGTH4	N/A	Returns the length of a character in different character sets.
LOWER	LOWER	Returns the lower case of a character string.
LPAD(arg1,arg2, arg3)	LPAD	Returns arg1, left-padded to length arg2 characters with the sequence of characters in arg3.
LTRIM	LTRIM	Removes blanks from the beginning of a string expression.

Oracle	Informix	Comments
REPLACE(arg1,arg2,arg3)	REPLACE(arg1,arg2,arg3)	Replaces all occurrences of srg2 in arg1 with arg3.
RPAD	RPAD	Returns the first argument value, right-padded to the length specified in the 2nd argument with characters specified in the third argument.
RTRIM	RTRIM	Remove blanks from the end of a string expression.
SOUNDEX	N/A	Returns a 4-character code representing the sound of the argument.
SUBSTR	SUBSTR SUBSTRING	Returns a substring of a string.
SUBSTRB SUBSTRC SUBSTR2 SUBSTR4	N/A	Returns a substring of a string.
TRANSLATE	N/A	Returns a string in which one or more characters in a string are converted to other characters.
TREAT	implement by CAST	Changes the declared type of an expression.
TRIM	TRIM	Removes leading or trailing blanks or other specified leading or trailing characters from a string expression.
UPPER	UPPER	Returns a string in which all the characters have been converted to uppercase characters.
TO_CHAR	TO_CHAR	Converts an expression that evaluates to a DATE, DATETIME or numeric value to a character string.

The missing functions like CHR and INSTR can be easily added using a C language based UDR. If your applications require the SOUNDEX function you can also check the usage of the VERITY of EXCALBUR datablade where this particular functionality is already implemented.

C.3 Date and time function mapping

Table C-3 shows the date and time function mapping between Oracle and Informix.

Table C-3 Date and time functions

Oracle	Informix	Comments
ADD_MONTHS	ADD_MONTHS	Returns the date argument plus integer months.
CURRENT_DATE	TODAY	Returns the current date.
CURRENT_TIME STAMP	CURRENT	Returns the current date and time.
DBTIMEZONE	N/A	Returns the value of the database time zone.
EXTRACT(datetime)	YEAR() MONTH() DAY()	Extracts and returns the value of a specified datetime field from a datetime expression.
FROM_TZ	N/A	Converts a time stamp value and a time zone. SQL> SQL> SELECT FROM_TZ(TIMESTAMP '2005-05-13 07:15:31.1234', 'EST') 2 FROM dual; FROM_TZ(TIMESTAMP'2005-05-1307: 15:31.1234','EST') ----- 13-MAY-05 07.15.31.1234000 AM EST SQL>
LAST_DAY	LAST_DAY	Returns the date of the last day of the month of the input date.
MONTHS_BETW EEN(arg1,arg2)	MONTHS_BETWEE N	Returns number of months between dates specified in arg1 and arg2.
NEW_TIME(date, arg1,arg2)	N/A. Can be implemented by C-UDR.	Returns the date and time in the time zone specified in arg2 of the date specified in the first input argument.

Oracle	Informix	Comments
NEXT_DAY(arg1, arg2)	NEXT_DAY	Returns the date of the first weekday named by arg2 that is later than the date arg1.
NUMTOSDINTERVAL NUMTOYMINTERVAL	available through CAST	Convert a value to an interval value.
ROUND(date,fmt)	N/A. Can be implemented by an UDR.	Returns date rounded to the unit specified by the format model fmt.
SESSIONTIMEZONE	DBINFO ('get_tz')	Returns the time zone of the current session.
SYS_EXTRACT_UTC	Implement by CURRENT TIMESTAMP - CURRENT TIMEZONE	Returns the UTC time from a datetime value with time zone offset or time zone.
SYSDATE	SYSDATE	Returns the current date.
SYSTIMESTAMP	CURRENT	Returns the current time stamp.
TRUNC(arg1,arg2)	N/A Can be implemented by a UDF	Returns arg1 with the time portion of the day truncated to the unit specified by arg2.
TZ_OFFSET	N/A	Returns the time zone offset

C.4 Comparison and NULL-related function mapping

Table C-4 shows the comparison and NULL-related function mapping between Oracle and Informix.

Table C-4 Comparison and NULL-related functions

Oracle	Informix	Comments
COALESCE	N/A Implementation by a C UDR or a Stored procedure	Returns the first argument that is not null.

Oracle	Informix	Comments
LNNVL	N/A	Evaluates a condition when one or both operands of the condition may be null.
NULLIF	NULLIF	Compares two expressions. Returns a null value if the arguments are equal, otherwise it returns the value of the first argument.
NVL	NVL	Replaces null with a string.
NVL2(arg1,arg2, arg3)	Can be implemented by NVL.	Returns arg2 when arg1 is not null and Returns arg3 when arg1 is null.

C.5 Encoding, decoding, encryption, and decryption function mapping

Table C-5 shows the encoding, decoding, encryption, and decryption function mapping between Oracle and Informix.

Table C-5 Encoding, decoding, encryption, and decryption functions

Oracle	Informix	Comments
DECODE(expr, srch_val1,result1,,, ,default)	DECODE	Compares the expr to each srch_val one by one. If the expr is equal to a srch_val, then returns the corresponding result.
N/A	DECRYPT_BIN DECRYPT_CHAR	Returns a value that is the result of decrypting encrypted-data.
N/A	ENCRYPT_AES ENCRYPT_TDES	Returns a value that is the result of encrypting data.
N/A	GETHINT	Returns the password hint if one is found in the encrypted-data.

C.6 Implementation of new C-based functions in IDS

If your application requires a function that does not exist in the standard built-in function set of IDS you can create your own implementation of this function. This can be done easily and requires the following steps:

1. Create a C file with your own C code.
2. Compile a relocatable object file with an available C compiler on your system.
3. Create a shared library with the ID utility.
4. Give the library the appropriate permissions and copy the library to a directory according your definitions.
5. Create a appropriate SQL function definition in your database where you need to use the new function.
6. Restart the server and use the function.

Here is an example that shows you all these steps based on a 32-bit library build on SUN Solaris. This library should finally provide the new functions `ascii`, `chr` and `cosh` as a sample implementation. Based on this sample, you should be able to implement all other missing libraries required by your applications. Example C-1 shows the sample C code for the functions. We used a naming of `ifmx<function> name`. There is a mapping to the SQL interface name later on in the creation of the function.

Example: C-1 Create your own UDF in C programming language within IDS

```
#include <mi.h>
#include <math.h>
#include <stdio.h>

mi_lvarchar *ifmxcosh(mi_lvarchar *input, MI_FPARAM *fparam)
{
    mi_lvarchar *RetVal;          /* The return value. */
    double ret;
    mi_char buffer[20];
    mi_char *buffer1;

    buffer1=mi_lvarchar_to_string(input);
    if (buffer1);
    ret=atof(buffer1);
    sprintf(buffer,"%g",cosh(ret));

    RetVal = mi_string_to_lvarchar(buffer);

    /* Return the function's return value. */
    return RetVal;
}

mi_lvarchar *ifmxchr(mi_integer input, MI_FPARAM *fparam)
{
    mi_lvarchar *RetVal;          /* The return value. */
    mi_char buffer[20];
    sprintf(buffer, "%c", input);
    RetVal = mi_string_to_lvarchar(buffer);

    /* Return the function's return value. */
    return RetVal;
}

mi_lvarchar *ifmxascii(mi_lvarchar *input, MI_FPARAM *fparam)
```

```

{
    mi_integer    ret;
    mi_lvarchar  *RetVal;      /* The return value. */
    mi_char      buffer[20];
    mi_char      *buffer1;

    buffer1=mi_lvarchar_to_string(input);
    if ( buffer1 )
        sprintf(buffer, "%d", buffer1[0]);
    RetVal = mi_string_to_lvarchar(buffer);

    /* Return the function's return value. */
    return RetVal;
}

```

After the implementation of the function, you must compile and link the code into a shared library, as shown in Example C-2. These calls depend on the base operating system and the memory layout. In case you use a 64-bit operating system the calls to the compiler and linker most likely require different options for the build. We build the library and copy the library into the extend subdirectory of the IDS distribution. This is the default place where all datablade objects reside. Make sure that the library has only read and execute permissions. The IDS do not accept the specification of a file with write permissions enabled.

Example: C-2 Compile and link a shared library on SUN Solaris 32 Bit

```

#Build example for the Library -- Solaris -- 32 Bit !
cc -xs -I$INFORMIXDIR/incl/public -c -o functions.o functions.c
ld -G -o $INFORMIXDIR/extend/oracle/functions.bld functions.o
chmod a+x $INFORMIXDIR/extend/oracle/functions.bld

```

After creating the library, you only need to register the library functions in the IDS server. You can do this with a create function and the specification of the input and output parameter. The location of the new library and the used programming language is additional required. The SQL statements for the registration are shown in Example C-3 on page 338. You can see that this is where the mapping to the final name is made. Finally, we show in the example how to use the functions in a simple SQL statement.

Example: C-3 Register user defined C- UDR with SQL statements in IDS

```
$dbaccess -e stores_demo << EOF
CREATE FUNCTION "holgerk".ascii(vvarchar(1))
RETURNING varchar(10)
WITH (NOT VARIANT, PARALLELIZABLE)
EXTERNAL NAME
"/sqldists/10.00.UC8/extend/oracle/functions.bld(ifmxascii)"
LANGUAGE C
END FUNCTION;

CREATE FUNCTION "holgerk".chr(integer)
RETURNING varchar(10)
WITH (NOT VARIANT, PARALLELIZABLE)
EXTERNAL NAME
"/sqldists/10.00.UC8/extend/oracle/functions.bld(ifmxchr)"
LANGUAGE C
END FUNCTION;

CREATE FUNCTION "holgerk".cosh(vvarchar(10))
RETURNING varchar(10)
WITH (NOT VARIANT, PARALLELIZABLE)
EXTERNAL NAME
"/sqldists/10.00.UC8/extend/oracle/functions.bld(ifmxcosh)"
LANGUAGE C
END FUNCTION;
EOF

#Use the functions in SQL statements

Database selected.

select cosh(1 ) from systables where tabid=1;
(constant)
1.54308
1 row(s) retrieved.

select ascii('A' ) from systables where tabid=1;
(constant)
65
1 row(s) retrieved.

select chr(100 ) from systables where tabid=1;
(constant)

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

Database server monitoring

Monitoring the database server, focussing on operating system and database server resource utilization, is one of the essential tasks for database administration. In this appendix we give you a brief mapping of the outputs of common monitoring facilities such as onstat and the sysmaster database interface in Informix IDS, and some common SQL queries in Oracle sqlplus. We included some of the major resources, as follows:

- ▶ Operating system and database server memory utilization
- ▶ System utilization
- ▶ Sessions, active queries, temp tables and session memory
- ▶ Cache utilization
- ▶ Disk space utilization
- ▶ Performance monitoring

D.1 Memory monitoring

One of the major resources on the host operating system is the memory used by the database server and the client applications. With a focus on IDS, the monitoring will provide information about such things as the amount of allocated shared memory segments and their sizes. The database server processes maintain all their data in the shared memory segments. The use of the local data segments in the process is itself low, and so no special monitoring activity is required. As shown in Example D-1, you can see the current amount of allocated shared memory segments and their utilization. You can map the onstat output with an `ipcs -m` operating system call with the segment ID in the first column.

Example: D-1 Monitoring shared memory segments in Informix IDS

```
holgerk {orion} onstat -g seg
```

```
IBM Informix Dynamic Server Version 11.10.FC2W3 -- On-Line -- Up 58
days 18:31:05 -- 148480 Kbytes
```

```
Segment Summary:
```

id	key	addr	size	ovhd	class	blkused	blkfree
1	1396918273	10a00000	116391936	1795104	R	28376	40
2	1396918274	110f0000	8388608	99704	V	2048	0
3	1396918275	11170000	9437184	112040	V	2304	0
4	1396918276	11200000	1048576	13544	M	138	118
5	1396918277	11210000	8388608	99704	V	1309	739
6	1396918278	11290000	8388608	99704	V	26	2022
Total:	-	-	-	152043520	-	-	-
34201	2919						

```
(* segment locked in memory)
```

```
dbaccess -e sysmaster << EOF
select * from sysseglst;
EOF
```

```
seg_address    4462739456
seg_next       4579131392
seg_prev       4606394368
seg_class      1
seg_size       116391936
seg_osshmid    1
seg_osmaxsize  134217728
seg_osshmkey   1396918273
```

```

seg_procid      4337
seg_userid      200
seg_shmaddr     4462739456
seg_ovhd        1795104
seg_lock        4462740016
seg_nextid     268435486
seg_bmapsz      1363968
seg_blkused     28376
seg_blkfree     40

```

If you want to drill down deeper into the memory utilization of the allocated shared memory segments, you can use the **onstat -g mem**. A common output is depicted in Example D-2. This output shows you the allocated pools in the shared memory segments, the amount of memory allocated for each pool, and the distribution of the memory in free and used memory.

Example: D-2 Memory monitoring in view of memory pools their memory utilization

```
{orion} onstat -g mem
```

```
IBM Informix Dynamic Server Version 11.10.FC2W3 -- On-Line -- Up 58
days 19:16:17 -- 148480 Kbytes
```

Pool Summary:

name	class	addr	totalsize	freesize	#allocfrag	
#freefrag						
afpool	V	10f28040	8192	1560	7	3
tpcpool	V	112fe040	20480	3368	14	3
seqpool	V	11339040	4096	752	2	1
pnlpool	V	11301040	28672	2280	23	3
sbtlist	V	11005040	20480	7216	4	3
dstpool	V	112fd040	8192	3304	2	2
sqcrypto	V	115fb040	4096	480	2	1
EXE.20.52	V	123ea040	4096	760	2	1
EXE.20.52	V	12543040	4096	760	2	1
EXE.20.52	V	1258d040	4096	696	3	1
EXE.20.52	V	1253e040	4096	376	8	1

```

dbaccess -e sysmaster << EOF
select * from syspoollst;
EOF

```

```

po_id          7
po_address

```

```

po_next
po_prev
po_lock      0
po_name      sbtlist
po_class     2
po_flags     0
po_freeamt   7216
po_usedamt   13264
po_freelist  4580200648
po_list      4580200608

```

D.2 Process utilization and configuration

A specific IDS database server instance is defined by a specific number of processes. The number of processors dedicated to the instance is stable and independent from the number of clients connecting to the database server. If you want to monitor the number of server processes and their workload you can use the **onstat -g glo** statement. Example D-3 shows the details of a possible output. The `sysmaster` table `sysvplst` can be selected for the same information using the SQL interface.

Example: D-3 Monitoring IDS server processes

```

onstat -g glo
Virtual processor summary:
class      vps      usercpu   syscpu    total
cpu        4        1947.94   461.54    2409.48
aio        10       228.86    447.69    676.55
lio        1        20.12     38.17     58.29
pio        1        24.68     56.40     81.08
adm        1        182.66    299.36    482.02
soc        1        130.97    540.31    671.28
msc        1        0.00      0.01      0.01
total     19       2535.23   1843.48   4378.71

```

```

Individual virtual processors:
vp  pid    class   usercpu  syscpu  total
1   4337   cpu     343.62   97.09   440.71
2   4338   adm     182.66   299.36  482.02
3   4339   cpu     525.95   112.75  638.70
4   4340   cpu     535.50   114.64  650.14
5   4341   cpu     542.87   137.06  679.93
6   4342   lio     20.12    38.17   58.29

```

7	4343	pio	24.68	56.40	81.08
8	4344	aio	20.74	42.63	63.37
9	4345	msc	0.00	0.01	0.01
10	4346	aio	24.14	56.59	80.73
11	4347	aio	21.00	40.94	61.94
12	4348	aio	27.68	53.03	80.71
13	4349	aio	19.06	37.33	56.39
14	4350	aio	21.86	39.81	61.67
15	4351	aio	23.37	42.26	65.63
16	4352	aio	29.99	56.47	86.46
17	4353	aio	18.72	39.23	57.95
18	4354	aio	22.30	39.40	61.70
19	4355	soc	130.97	540.31	671.28
		tot	2535.23	1843.48	4378.71

```
dbaccess -e sysmaster << EOF
select * from sysvplst;
EOF
```

```
vpid      1
address   4579319848
pid       4337
usecs_user 343.6400000000
usecs_sys 97.0900000000
scputimep 4579319864
....
```

Monitoring the processes and their system utilization is useful from the operating system monitoring point of view. From the database administration point of view a much better view of the granularity of the workload of the database server can be achieved by looking at the threads. IDS internally creates its own threads doing the work for incoming client requests. They are executed on all processes of the name type CPU VP. Look at the output of **onstat -g ath**, **onstat -g act** or **onstat -g rea** for threads. These outputs give you either the complete view of the existing threads in the system or a particular view on threads with the same status in the system. Example D-4 on page 344 shows an example for an output of all the threads that currently exist in the server.

```
holgerk {orion} onstat -g ath
```

Threads:

tid	tcb	rstcb	prty	status	vp-class	name
2	111455d18	0	1	IO Idle	6lio*	lio vp 0
3	111475c98	0	1	IO Idle	7pio*	pio vp 0
4	111495c98	0	1	IO Idle	8aio*	aio vp 0
5	1114b4c98	0	1	IO Idle	9msc*	msc vp 0
6	1114e3c98	0	1	IO Idle	10aio*	aio vp 1
7	111502c98	0	1	IO Idle	11aio*	aio vp 2
8	111521c98	0	1	IO Idle	12aio*	aio vp 3
9	111540c98	0	1	IO Idle	13aio*	aio vp 4
10	11155fc98	0	1	IO Idle	14aio*	aio vp 5
11	11157ec98	0	1	IO Idle	15aio*	aio vp 6
12	11159dc98	0	1	IO Idle	16aio*	aio vp 7
13	1115bcc98	0	1	IO Idle	17aio*	aio vp 8
14	1115dbc98	0	1	IO Idle	18aio*	aio vp 9
15	111607028	1111ca028	3	sleeping secs: 1	4cpu	main_loop()
16	111620028	0	1	running	1cpu*	sm_poll
17	11163a2a8	0	1	running	19soc*	soctcppoll
18	111656028	0	2	sleeping forever	1cpu	sm_listen
19	11167e208	0	1	sleeping secs: 1	3cpu	sm_discon
20	111693348	0	2	sleeping forever	1cpu*	soctcplst

D.3 Disk space monitoring

In addition to the memory and processor utilization there is another critical resource provided by the operating system and used by the database server, and that is the disk space. IDS views dbspaces as a logical organization unit, combining disks together for the use in the server instance for objects, such as logs, tables and indexes. There is also the definition of chunks as the physical unit describing the view of the database server to the physical files and raw devices. You can use the **onstat -d output** command for monitoring the dbspace and chunk statistics. Example D-5 on page 345 shows the sample output for a system with 5 dbspaces, where each dbspace maintains one chunk. Important from the administration point of view is the parameter attached to the chunk. Look particularly at the sizes, the location and the free parameter. Define high watermarks for the free parameter to react appropriately before the chunk runs out of disk space and your production environment is impacted by an out of disk space condition.

Example: D-5 onstat -d to monitor the disk utilization

```
onstat -d
Dbspaces
address number  flags      fchunk  nchunks  pgsz  flags  owner
name
1110dee88 1      0x60001  1 1 2048    N B     informix rootdbs
111221ce0 2      0x60001  2 1 2048    N B     informix dbspace2
111221e78 3      0x60001  3 1 2048    N B     informix dbs1
11120ac50 4      0x60001  4 1 2048    N B     informix dbs2
11120ade8 5      0x60001  5 1 2048    N B     informix dbs3
5 active, 2047 maximum

Chunks
address chunk/dbs  offset  size  free  bpages  flags
pathname
1110e0028 1 1      0 50000 23606 P0-B  /dbspaces/rootdbs.1110fc2w3
1110e0278 2 2      0 10240 10212 P0-B  /dbspaces/dbs2.00
1110e0460 3 3      0 25000 24667 P0-B  /dbspaces/11.10.FCW3/dbs1
1110e0648 4 4      0 25000 24667 P0-B  /dbspaces/11.10.FCW3/dbs2
1110e0830 5 5      0 25000 19947 P0-B  /dbspaces/11.10.FCW3/dbs3
5 active, 32766 maximum

dbaccess -e sysmaster << EOF
select * from sysdbspaces;
select * from syschunks;
EOF

dbsnum      1
name        rootdbs
owner       informix
pagesize    2048
fchunk      1
nchunks     1
is_mirrored 0
is_blobspace 0
is_sbspace  0
is_temp     0
flags       393217

chknum      1
dbsnum      1
nxchknum    0
pagesize    2048
chksize     50000
```

```

offset          0
nfree          23606
mdsize         -1
udsize         -1
udfree         -1
is_offline     0
is_recovering  0
is_blobchunk   0
is_sbchunk     0
is_inconsistent 0
flags          196672

```

D.4 Session monitoring

IDS provides two types of views for session monitoring. You can use a table oriented global view showing the session basics, such as the user name, the current statement type, database and status information and some accumulated values for used memory. You can also use either **onstat -g sql** or **onstat -g ses** for generating the output, as depicted in Example D-6.

Example: D-6 Session monitoring

```
holgerk {orion} onstat -g sql
```

Sess Id	SQL Stmt type	Current Database	Iso Lvl	Lock Mode	SQL ERR	ISAM ERR	F.E. Vers
49	UPDATE	stores_demo	NL	Not Wait	0	0	9.24 Off
47	SELECT	sysmaster	CR	Not Wait	0	0	9.24 Off
21		sysadmin	DR	Wait 5	0	0	- Off
20		sysadmin	DR	Wait 5	0	0	- Off
19		sysadmin	DR	Wait 5	0	0	- Off

```
holgerk {orion} onstat -g ses
```

session dynamic id	user	tty	pid	hostname	#RSAM threads	total memory	used memory	
50	informix	-	0	-	0	12288	11576	off
49	informix	5	28389	orion	1	86016	75456	off
47	huser1	5	27999	orion	1	184320	175816	off
21	informix	-	0	-	1	479232	395192	off
20	informix	-	0	-	1	487424	354528	off
19	informix	-	0	-	1	258048	194048	off
8	informix	-	0	-	0	12288	11576	off

```

7      informix -      0      -      0      16384      13192      off
6      informix -      0      -      0      12288      11576      off
4      informix -      0      -      0      16384      13192      off
3      informix -      0      -      0      12288      11576      off
2      informix -      0      -      0      12288      11576      off

```

```

dbaccess -e sysmaster << EOF
select * from sysessions;
EOF

```

If you want to have more details about a particular session, use the same **onstat -g ses** or **onstat -g sql** with the additional specification of the session number you obtained from the previous examples. A sample output is depicted in Example D-7.

Example: D-7 Detailed session information for a particular session

```
holgerk {orion} onstat -g ses 47
```

```

IBM Informix Dynamic Server Version 11.10.FC2W3  -- On-Line -- Up 58 days
21:39:18 -- 148480 Kbytes

```

```

session      effective      #RSAM      total      used
dynamic
id           user      user      tty      pid      hostname threads      memory
memory      explain
47          informix -      5      27999      orion      1      184320
175816      off

tid          name      rstcb      flags      curstk      status
81          sqlexec  1111d0148  Y--P---  7791      cond wait  sm_read  -

Memory pools      count 2
name              class addr      totalsize freesize      #allocfrag #freefrag
47                V      11235e040  180224      7680      222      16
47*00            V      1122cb040  4096        824        1        1

name              free      used      name              free      used
overhead          0         6544      mtmisc            0         80
scb                0         144       opentable         0         4592
filetable         0         1160      log               0         16520
temprec           0         10104     keys              0         656
ralloc            0         101472    gentcb            0         1584
ostcb             0         2872      sqscb             0         20392
sql               0         72        rdahead           0         608
hashfiletab       0         552       osenv             0         2752
sqtcb             0         4616     fragman           0         680
sapi              0         88        udr               0         232

```

```

sqscb info
scb          sqscb          optofc  pdqpriority sqlstats optcompind
directives
1123000b0    11258f028      0      0          0          0          1

Sess      SQL      Current      Iso Lock      SQL ISAM F.E.
Id        Stmt type Database      Lvl Mode      ERR ERR Vers
Explain
47        SELECT    sysmaster    CR Not Wait  0  0  9.24 Off

Current statement name : slctcur

Current SQL statement :
  select * from syschunks

Last parsed SQL statement :
  select * from syschunks

```

D.5 Cache monitoring

The database server uses different caches to maintain the information of already evaluated and still valid database object information. The object is evaluated once by reading the data from the system catalog from disk. All subsequent readers can use the cache to access the information either exclusive or shared, depending on the current requests. Once the object is invalidated in cache it will be removed. You can monitor the different caches in the database server for objects such as tables, stored procedures, and user-defined types or statements. Use **onstat -g dic**, **onstat -g prc** or **onstat -g cac** for this. See Example D-8 for how the sample output will appear.

Example: D-8 Display the content of several database caches

```

{orion} onstat -g dic
Dictionary Cache: Number of lists: 31, Maximum list size: 10

list#  size  refcnt  dirty?  heaptr      table name
-----
  0     1     0      no     1123e0038   sysmaster@srv:informix.sysdbstab
  1     2     0      no     1123be038   sysmaster@srv:informix.syspoolst
                0      no     112373038   sysmaster@srv:informix.systabnames
  3     3     0      no     112571838   sysmaster@srv:informix.sysdbspaces
                0      no     11236c838   sysadmin@srv:informix.mon_vps
                0      no     1123ab838   sysmaster@srv:informix.systcblst

```

4	3	0	no	1122bb038	sysmaster@srv:informix.syscheckpoint
		0	no	1128ba038	sysadmin@srv:informix.systables
		0	no	1128c6838	sysadmin@srv:informix.ph_version
5	1	1	no	112340038	sysadmin@srv:informix.mon_config
6	2	0	no	112388038	sysmaster@srv:informix.syssmhdr
		0	no	1123f8838	sysmaster@srv:informix.systables
7	1	0	no	112270838	sysmaster@srv:informix.sysseglst
8	1	0	no	11231e038	sysmaster@srv:informix.sysenv
9	1	0	no	1128db838	sysadmin@srv:informix.sysindices
11	4	1	no	1122e5838	stores_demo@srv:informix.systables
		0	no	11238f838	sysmaster@srv:informix.sysvplst
		0	no	1123d5838	sysmaster@srv:informix.sysptnhdr
		6	no	1128e6838	sysadmin@srv:informix.ph_run
14	1	7	no	1112f6568	sysadmin@srv:informix.ph_task

holgerk {orion} onstat -g prc

UDR Cache:

Number of lists : 31
PC_POOLSIZE : 127

UDR Cache Entries:

list#	id	ref_cnt	dropped?	heap_ptr	udr name
0	27	0	0	1123f7038	sysadmin@srv:.destroy
2	380	0	0	11231b438	sysadmin@srv:.check_backup
3	33	0	0	1122ab438	sysadmin@srv:.destroy
3	133	0	0	112319438	sysadmin@srv:.assign
4	28	0	0	112312838	sysadmin@srv:.assign
8	378	0	0	1122a0c38	sysadmin@srv:.onconfig_save_diffs
8	49	0	0	1123e4838	sysadmin@srv:.destroy

Archived

Database server utilities

IDS includes the following utilities that let you perform administrative tasks and capture information about configuration and performance. These utilities are described in detail in the relevant publication of the IDS documentation set, as shown in the last column of the table.

Table 9-9 IDS utilities

Utility	Description	Where described
archecker	Verify backups and perform table-level restores.	<i>IBM Informix Backup and Restore Guide</i>
cdr	Control enterprise replication operations.	<i>IBM Informix Dynamic Server Enterprise Replication Guide</i>
dbexport	Unload a database into text files for later import into another database and create a schema file	<i>IBM Informix Migration Guide</i>
dbimport	Create and populate a database from text files. Use the schema file with dbimport to recreate the database schema.	<i>IBM Informix Migration Guide</i>
dbload	Load data into databases or tables	<i>IBM Informix Migration Guide</i>

Utility	Description	Where described
dbschema	Create a file that contains the SQL statements needed to replicate a specified table, view, or database, or view the information schema.	<i>IBM Informix Migration Guide</i>
imcadmin	Start or stop Informix MaxConnect, or gather statistics on it.	<i>IBM Informix MaxConnect User's Guide</i>
ism	Manage IBM Informix Storage Manager, storage devices, and media volumes.	<i>IBM Informix Storage Manager Administrator's Guide</i>
onaudit	Manage audit masks and auditing configurations.	<i>IBM Informix Security Guide</i>
onbar	Back up and restore storage spaces and logical logs.	<i>IBM Informix Backup and Restore Guide</i>
oncheck	Check specified disk structures for inconsistencies, repair inconsistent index structures, and display information about disk structures.	<i>IBM Informix Administrator's Reference</i>
oncmsm	Start the Connection Manager, which manages and redirects client connection requests based on service level agreements configured by the system administrator.	<i>IBM Informix Administrator's Reference</i>
ondblog	Change the logging mode.	<i>IBM Informix Administrator's Reference</i>
oninit	Bring the database server online.	<i>IBM Informix Administrator's Reference</i>
onload	Load data that was created with onunload into the database server.	<i>IBM Informix Migration Guide</i>
onlog	Display the contents of logical-log files.	<i>IBM Informix Administrator's Reference</i>
onmode	Change the database server operating mode and perform various other operations on shared memory, sessions, transactions, parameters, and segments.	<i>IBM Informix Administrator's Reference</i>
ON-Monitor	Perform administrative tasks using the ON-Monitor menus.	<i>IBM Informix Administrator's Reference</i>

Utility	Description	Where described
onparams	Modify the configuration of logical logs or physical logs.	<i>IBM Informix Administrator's Reference</i>
onpassword	Encrypts and decrypts password files for Enterprise Replication and Connection Manager.	<i>IBM Informix Administrator's Reference</i>
onperf	Monitor database server performance (create graphs, query trees, and show status and metrics).	<i>IBM Informix Performance Guide</i>
onpladm	Write scripts and create files that automate data load and unload jobs.	<i>IBM Informix High-Performance Loader User's Guide</i>
onshowaudit	Extract information from an audit trail.	<i>IBM Informix Security Guide</i>
onspaces	Modify dbspaces, blobspaces, sbspaces, or extspaces.	<i>IBM Informix Administrator's Reference</i>
onstat	Monitor the operation of the database server.	<i>IBM Informix Administrator's Reference</i>
ontape	Log, back up, and restore data.	<i>IBM Informix Backup and Restore Guide</i>
onunload	Unload data from the database server.	<i>IBM Informix Migration Guide</i>

Archived

Additional material

This book refers to additional material that can be downloaded from the Internet as described in the sections that follow.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

<ftp://www.redbooks.ibm.com/redbooks/SG247730>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** from the domains list in the upper left corner of the web page, and then open the directory that corresponds with the IBM Redbooks form number, SG24-7730. See the readme.txt file for instructions, and the ALL ORACLE DDL.TXT file for a list of all the required DDL.

Using the Web material

The additional Web material that accompanies this book includes the following files:

<i>File name</i>	<i>Description</i>
orclsetup.rar	Zipped File Setup

This rar file contains the scripts, commands and instructions for creating the example database used in the MTK Tutorial in Chapter 5, “An MTK tutorial” on page 73.

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	5 MB minimum
Operating System:	Windows 2000
Processor:	Intel® 386 or higher
Memory:	16 MB

To download to AIX Linux or UNIX, use the equivalent or higher system capacity as specified for Windows

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

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.

Embedded Database. A database that works exclusively with a single application or appliance.

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. A 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	DBM	Database manager
ADK	Archive Development Kit	DBMS	Database management system
API	Application programming interface	DCE	Distributed computing environment
AQR	Automatic query rewrite	DCM	Dynamic Coserver Management
AR	Access register	DCOM	Distributed Component Object Model
ARM	Automatic restart manager	DDL	Data definition language
ART	Access register translation	DES	Data Encryption Standard
ASCII	American Standard Code for Information Interchange	DIMID	Dimension Identifier
AST	Application summary table	DLL	Dynamic link library
AUS	Auto Update Statistics	DML	Data manipulation language
BLOB	Binary large object	DMS	Database managed space
BW	Business Information Warehouse (SAP)	DPF	Data partitioning facility
CCMS	Computing Center Management System	DRDA	Distributed Relational Database Architecture™
CDR	Continuous Data Replication	DSA	Dynamic Scalable Architecture
CFG	Configuration	DSN	Data source name
CLI	Call-level interface	DSS	Decision support system
CLOB	Character large object	EAI	Enterprise Application Integration
CLP	Command-line processor	EBCDIC	Extended Binary Coded Decimal Interchange Code
CLR	Continuous Log Recovery	EDA	Enterprise data architecture
CORBA	Common Object Request Broker Architecture	EDU	Engine dispatchable unit
CPU	Central processing unit	EGL	Enterprise Generation Language
CS	Cursor Stability	EGM	Enterprise Gateway Manager
DaaS	Data as a Service	EJB™	Enterprise Java Beans
DAS	DB2 Administration Server	ER	Enterprise Replication
DB	Database	ERP	Enterprise Resource Planning
DB2 II	DB2 Information Integrator		
DB2 UDB	DB2 Universal Database™		
DBA	Database administrator		

ESE	Enterprise Server Edition	JDK	Java Development Kit
ETL	Extract, Transform, and Load	JE	Java Edition
FP	Fix Pack	JMS	Java Message Service
FTP	File Transfer Protocol	JRE™	Java Runtime Environment
Gb	Gigabits	JVM	Java Virtual Machine
GB	Gigabytes	KB	Kilobyte (1024 bytes)
GLS	Global Language Support	LBAC	Label Based Access Control
GUI	Graphical user interface	LDAP	Lightweight Directory Access Protocol
HADR	High Availability Disaster Recovery	LPAR	Logical partition
HDR	High Availability Data Replication	LRU	Least Recently Used
HPL	High Performance Loader	LUN	Logical unit number
I/O	Input/output	LV	Logical volume
IBM	International Business Machines Corporation	Mb	Megabits
ID	Identifier	MB	Megabytes
IDE	Integrated Development Environment	MDC	Multidimensional clustering
IDS	Informix Dynamic Server	MPP	Massively parallel processing
II	Information Integrator	MQI	Message queuing interface
IMS	Information Management System	MQT	Materialized query table
ISA	Informix Server Administrator	MRM	Message repository manager
ISV	Independent Software Vendor	MTK	IBM Migration Toolkit
ISAM	Indexed Sequential Access Method	NPI	Non-partitioning index
ISM	Informix Storage Manager	OAT	Open Admin Tool
ISV	Independent software vendor	ODBC	Open Database Connectivity
IT	Information technology	ODS	Operational data store
ITR	Internal throughput rate	OEM	Original Equipment Manufacturer
ITSO	International Technical Support Organization	OLAP	Online analytical processing
IX	Index	OLE	Object linking and embedding
J2EE	Java 2 Platform Enterprise Edition	OLTP	Online transaction processing
JAR	Java Archive	ORDBMS	Object Relational Database Management System
JDBC	Java Database Connectivity	OS	Operating System
		O/S	Operating System
		PAM	Pluggable Authentication Module
		PDS	Partitioned data set

PHP	Hypertext preprocessor. A general purpose scripting language.	TMU	Table management utility
PIB	Parallel index build	TS	Table space
PSA	Persistent staging area	UDB	Universal Database
RBA	Relative byte address	UDF	User defined function
RBAC	Role Based Access Control	UDR	User defined routine
RBW	Red brick warehouse	URL	Uniform Resource Locator
RDBMS	Relational Database Management System	VG	Volume group (RAID disk terminology).
RHEL	Red Hat® Enterprise Linux	VLDB	Very large database
RID	Record identifier	VP	Virtual processor
RR	Repeatable read	VSAM	Virtual sequential access method
RS	Read stability	VII	Virtual Index Interface
RSAM	Relational Sequential Access Method	VTI	Virtual Table Interface
RSS	Remote Standalone Secondary	WFS	Web Feature Service
RTO	Recovery Time Objective	WSDL	Web Services Definition Language
SA	Systems administrator	WWW	World Wide Web
SCB	Session control block	XBSA	X-Open Backup and Restore APIs
SDK	Software Developers Kit	XML	Extensible Markup Language
SDS	Shared Disk Secondary	XPS	Informix Extended Parallel Server
SID	Surrogate identifier		
SLES	SuSE Linux Enterprise Server		
SMI	System Monitoring Interface		
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		

Archived

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 367. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Oracle to DB2 Conversion Guide for Linux, UNIX, and Windows*, SG24-7048
- ▶ *Developing PHP Applications for IBM Data Servers*, SG24-7218
- ▶ *Informix Dynamic Server 11: Advanced Functionality for Modern Business*, SG24-7465
- ▶ *IBM InfoSphere DataStage Data Flow and Job Design*, SG24-7576
- ▶ *Informix Dynamic Server 11: Extending Availability and Replication Guide*, SG24-7488

Other publications

These publications are also relevant as further information sources:

- ▶ *IBM Data Server Provider for .NET Programmer's Guide*, SC23-7688
- ▶ *IBM Informix Security Guide*, G229-6389

Online resources

These Web sites are also relevant as further information sources:

- ▶ *Oracle to IBM Informix Dynamic Server Porting Guide*, available at the URL:
<http://www.ibm.com/developerworks/data/library/long/dm-0608marino/>
- ▶ *IBM Migration Toolkit*, available at the URL:
<http://www-01.ibm.com/software/data/db2/migration/mtk/>

Education support

Available from IBM training, the newest offerings to support your training needs, enhance your skills and boost your success with IBM software.

IBM offers a range of training options from traditional classroom to Instructor-Led Online to meet your demanding schedule.

Instructor-Led Online (ILO) is an innovative learning format where students get the benefit of being in a classroom with the convenience and cost savings of online training.

Go green with IBM Onsite training for groups as small as three or as large as fourteen. Choose from the same quality training delivered in classrooms, or customize a course or a selection of courses to best suit your business needs.

Enjoy further savings when you purchase training at a discount with an IBM Education Pack – online account – flexible and convenient way to pay, track and manage your education expenses online.

Check your local Information Management Training and Education website or with your training representative for the most recent training schedule.

Table 10 lists related education offerings.

Table 10 Education offerings

Course Code	Course Title	Course Type
IX13	Informix Structured Query Language	Classroom
3X13	Informix Structured Query Language	Instructor Led Online
IX22	Informix Dynamic Server Database Administration: Managing and Optimizing Data	Classroom
3X22	Informix Dynamic Server Database Administration: Managing and Optimizing Data	Instructor Led Online
IX40	Informix Dynamic Server Performance Tuning	Classroom
IX42	Informix Dynamic Server Replication	Classroom
3X42	Informix Dynamic Server Replication	Instructor Led Online
IX81	Informix Dynamic Server Systems Administration	Classroom

Course Code	Course Title	Course Type
3X81	Informix Dynamic Server Systems Administration	Instructor Led Online

Descriptions of courses for IT professionals and managers are available at the following Web page:

http://www.ibm.com/services/learning/ites.wss/tp/en?pageType=tp_search

Visit ibm.com/training or call IBM training at 800-IBM-TEACH (426-8322) for scheduling and enrollment.

IBM Professional Certification

Information Management Professional Certification is a business solution for skilled IT professionals to demonstrate their expertise to the world. Certification validates skills and demonstrates proficiency with the most recent IBM technology and solutions. Table 11 lists the related certification exam.

Table 11 Professional certification

Exam #	Exam Name	Certification Title
918	System Administration for IBM Informix Dynamic Server V11	IBM Certified System Administrator - Informix Dynamic Server V11

For additional information about this exam, see:

<http://www-03.ibm.com/certify/certs/30001104.shtml>

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Archived

Index

Symbols

.NET 211
.NET Data Providers 271
.NET Framework applications 270
.NET key components 214

A

acceptance testing 55
access methods 12
Active X Data Object 216
administration xiv, 4–7, 12
administration free zone 12
administrative tasks 25, 281, 283, 288, 296,
305–306, 351–352
aggregates 12
AIX 8
alert log file 29
alert log files 28
aliases 146
ANTLR (Another Tool for Language Recognition)
66
Apache Web Server 12
API (application programming interface) 12, 224
applets 241
application environments 212
application migration 49, 51, 223
application migration test 225
application object migration 101
application programming techniques 226
application testing 55–56
application tuning 225
architecture 4–5, 15–18, 23, 214
archive log files 28–29
archiver 24
array 5–6
arrays 234
asynchronous 8
attached indexes 127
auditing utilities 311
authorized user 41
availability 5, 167
awk utility 185

B

backup 6–8, 12
backup and restore 286
big buffers 22
bigfile tablespace 33
binary data types 151
Binary DataBlade 13
BLOB (binary large object) 35, 120, 195, 286, 316,
357
BLOB or SBLOB loading 205
blob unloading 195
blobpage 34
blobspace 34, 300
BOOLEAN variable type 151
buffer pool 20–21, 297, 299
buffer_size 239
bufferpool 20
built-in functions 13, 181
business intelligence xii

C

C 231
C++ 231
calibration 50
cartridge 219
catalog tables 36–37, 39, 123, 148
cataloging 14
change management 46
Changes Report 91
character data type 190
Check Constraint 115
checkpoint 20, 24, 42, 289, 324
chunk 29, 34, 284, 286, 290, 297, 300–301,
344–345
chunk management 179
C-ISAM® DataBlade 14
Client SDK 39, 205–206, 215
CLOB (character large object) 35, 120, 286, 316,
361
CLOB and BLOB data types 120
CLR 9, 361
column-level encryption 160, 309
composite index 126

- Composite Range-Hash partitioning 118
- concurrency contro 154
- concurrent session 41
- configurable page sizes 126
- configuration 5
- configuration costs 2
- configuration parameters 280
- conflict resolution 8
- connecting to the database 229
- connection 229
- connectivity configuration 282
- consolidation 9
- constraint definition statements 177
- constraints 73, 114
- Continuous Log Recovery 361
- Continuous Log Restore 9
- control blocks 22, 149
- control files 24, 28, 199
- conversion 166
- conversion name 94
- cooked disk space 26
- correlation names 145
- cost-based optimizer 11
- creating tables 113
- cumulative migration reports 110
- cursor 234
- cursor stability 155
- cursor state 18
- cursors 150
- cutover strategy 46

D

- data blocks 19, 23–24, 32, 323
- data buffer cache 18–19
- data conversion 165
- data definition language xi, 357
 - Also see DDL
- data deployment 97
- data dictionary 15, 18, 35, 37, 168
- data dictionary cache 18
- data fragments 127
- data infrastructure 2
- data integration 2
- data integrity 12
- data manipulation language xi
- data migration 52
- data migration strategy 46
- data movement 73, 167

- data pump export and import utilities 169
- data segment 32
- data source name (DSN) 219
- data transfer script creation 73
- Data Transfer Scripts 69, 92
- data type mapping 84
- data type migrations 55
- data types 4–5, 12
- data unload 179
- data warehousing xii
- database xii
- database administrator view 35
- database connectivity 15
- database creation 112
- database I/O 25
- database management system 1
 - Also see DBMS
- database migration 49
- database objects xi, 2, 36, 45–47, 50, 56, 62, 64, 73, 75, 78, 108, 134, 167–169, 171, 174, 178–179, 216, 227, 290–291, 310, 322–323
- database permissions 160
- database physical design 46
- database writer 24
- DataBlade 12–14
- DataBlade Developer's Kit 13
- DataBlade modules 12–13
- DataBlades 14
 - Basic Text Search DataBlade 13
 - Binary DataBlade 13
 - C-ISAM DataBlade 14
 - Excalibur Text DataBlade 14
 - Geodetic DataBlade 14
 - Image Foundation DataBlade 14
 - Node DataBlade 13
 - Spatial DataBlade 14
 - TimeSeries DataBlade 13
 - TimeSeries Real-Time Loader 13
 - Video Foundation DataBlade 14
- data-distribution cache 22
- datafiles 23–24, 28, 33
- DataStage 209
- DATE data type 193
- DB Access Utility 39
- DBA 12
- dbload utility 197, 201
- DBMS (database management system) 1–2
- DBMS_METADATA utility 168
- dbspace management 179

dbspace pages 20
DDL (data definition language) xi, 2, 46, 68, 74, 79, 111, 167, 268, 357
DDL statements 112
decision-support operations 4–5
decryption 27, 162, 335
default isolation level 156
default privileges 112
degree of parallelism 10
delimited unload 187
deploy DDL 94
deployable conversion 94
deployment scripts 64
desc statement 170
dictionary cache 22, 323
disaster 10
disaster recovery 294
discretionary access control 310
disk scan threads 10
DML 2
double buffering 21
DRAUTO 10
DRDA (Distributed Relational Database Architecture) 325, 361
drop statements 84
DSA (Dynamic Scalable Architecture) 4–7, 361
Dynamic embedded SQL 228
Dynamic Scalable Architecture 4–5, 361
Dynamic SQL 152, 236

E

EJB (Enterprise JavaBean) 241
embedded error handling 237
embedded SQL applications 226
encryption 27, 160–161, 309, 335
encryption functions 161
encryption method 162
Enterprise Edition 6, 8
environment preparation 45, 48
ER (enterprise replication) 6, 8–9, 286, 291, 295, 351
error handling 149
error messages 238
ESQL/C 213
esql/c applications 239
Excalibur 14
Excalibur Text DataBlade 13
execution plan 18

EXIT statement 150
expression-based fragmentation 117
extensibility 12
Extensible Markup Language 4
extents 32–33, 146, 297, 306, 322
external directives 138

F

failover 10
fan-out parallelism 10
fast recovery 286, 289
fetch cycle 250
five-step migration process 61
flow control 150
Foreign Key Constraint 114
forest 8
fragment expressions 118
fragmentation 2
functional testing 54
function-based index 129
functions 101

G

Generate Data Transfer Scripts 62
generator 224
geodetic 4
Geodetic DataBlade 14
global area memory 16
global area pointers 250
Global pool 22
Global Type Mapping 84
grant statement 160
grant statements 81

H

HA (High Availability) xii, 6, 9, 47, 279, 286
HA *See* High Availability
handling exceptions 149
hash partitioning 118
HDR (High Availability Data Replication) 6, 8–10, 290–291, 294–295, 298, 300, 302, 362
hierarchical 8, 13
hierarchical query 147
high availability 10
high availability cluster 291
high availability cluster configuration 291
High Performance Load Utility 201

- High-Availability Clusters 286
- host name 94
- host variable 231
- host variable arrays 234
- host variable data types 232
- host variables 231
- HPL (high performance loader) 10, 197
- Hypertext Preprocessor 217

I

- IBM classroom training 49
- IBM Data Management Services Center 44
- IBM Informix .NET Provider 215
- IBM Informix Client Software Development Kit 212
- IBM Informix DataBlades
 - See DataBlades
- IBM Informix ESQL/COBOL 213
- IBM Informix OLE DB Provider 216
- IBM Informix Storage Manager 287
- IBM migration team 44, 47
- IBM Rational Functional Tester 53
- IBM Rational Performance Tester 53
- IDS 4–5, 8–10, 12–13
 - Also see Informix Dynamic Server
- IDS Architecture 16
- IDS Client SDK 39
- IDS database 34, 62–63, 69, 74, 92, 94, 167, 171, 173, 179, 183, 192, 197, 206, 218, 220, 226, 237, 241, 249, 257, 261, 302, 308, 342
- IDS dbspace 34
- IDS dbspaces 62, 91
- IDS editions 15, 40–41
- IDS Express 6
- IDS security features 308
- IDS system catalog 15
- IDS tablespace 33
- IDS trigger 108
- IDS virtual processor 26–27
- IDS Workgroup Edition 6
- Image Foundation DataBlade 14
- incremental backup 289
- index builds 10
- index fragmentation 127
- index fragments 127
- index objects 174
- index scans 11
- Index Segment 32
- indexes 73, 125

- India xiii
- Informix xii, 4–6, 12–14
 - Dynamic Server 5
- Informix Call Level Interface 248
- Informix C-ISAM 14
- Informix CLI architectures 249
- Informix CLUSTER option 129
- Informix committed read 154
- Informix Connect 39, 212, 219, 225, 239, 256
- Informix DataBlade Developer's Kit 13
- Informix dbspaces 114
- Informix Dirty Read 154
- Informix Dynamic Server 2, 4, 6, 8, 15, 45, 60, 74, 111, 211, 279, 321, 340, 351
 - Also see IDS
- Informix Dynamic Server technology 45
- Informix Dynamic Server Workgroup Edition 7
- Informix esql precompiler 232
- Informix ESQL/C 230
- Informix Extended Parallel Server 129
- Informix Geodetic DataBlade 14
- Informix High Availability Data Replication 8
- Informix Image Foundation DataBlade 14
- Informix JDBC 245
- Informix JDBC driver 221, 243
- Informix JDBC drivers 241
- Informix SDK 219
- Informix Spatial DataBlade 14
- Informix SPL 111, 148
- Informix stored procedure 258
- Informix syntax 114
- Informix TimeSeries DataBlade 13
- Informix TimeSeries Real-Time Loader 13
- initialization parameters 29
- inline directive 138
- INOUT 236
- instance 4–5, 7–8, 11–12
- integration testing 54
- integrity constraints 35
- interval checkpoints 11
- isolation level 156
- isolation levels 154
- ISV migrations 62

J

- J2SE 240
- Java applications 240
- Java environment 240

- Java native driver 63
- Java Runtime Environment 70
- Java ServerPages 241
- Java virtual machine 214
- JDBC applications 221
- JDBC applications 240
- JDBC drivers 214
- Join Methods 137

K

- kernel asynchronous I/O (KAIO) 26

L

- Language syntax changes 224
- large objects 120
- latency 10
- LBAC (Label Based Access Control) 11, 42, 310–311, 362
- lessons learned 109
- library cache 18
- lightweight processes 25
- Linux 5, 7
- list partitioning 118
- load 7, 10, 12–13
- load stress testing 55
- loading data 92, 197
- Local Partitioned Index 127
- LOCK MODE 113
- lock table 21, 156
- locking 154
- LOCKS configuration parameter 21
- Log writer 24
- logging and recovery 286
- logging I/O 25
- logging status 20, 36, 287
- logical consistency 11
- logical database architecture 15
- logical database structures 32–33
- logical log backup 30, 288
- logical log buffer 20
- logical log files 26, 29–30
- logical volume 286
- logical-log file 288

M

- mandatory access control 310
- memory 6, 8, 10

- memory allocations 22
- memory architecture 15–18
- memory grant manager (MGM) 10
- memory pools 22, 341
- message buffers 22–23
- message file 29
- message log file 31, 285
- message_buffer is 239
- metadata 14, 35, 66, 169
- Metadata Interchange 67
- metrics collected 48
- migrated views 56
- migration 43
- migration assessment 45
- migration methodology 2
- migration preparation 45
- migration project 61
- migration refresh 52
- Migration Toolkit 47
- migration tools 45
- migration tutorial 80
- mirrored chunk 290
- Mirroring 286
- model 7, 14
- monitoring functionality 11
- MTK (IBM Migration ToolKit) xi, 2, 46, 57, 59, 63, 71, 73, 75, 143, 165, 224, 365
- MTK command line arguments 65
- MTK command line interface 64
- MTK converter 61
- MTK debugger 100
- MTK GUI interface 64
- MTK hardware requirements 70
- MTK installation 61
- MTK installation directory 64
- MTK log 98, 100
- MTK on UNIX or Linux 71
- MTK on Windows 71
- MTK prerequisites 71
- MTK refine page 62
- MTK software requirements 70
- MTK system requirements 70
- MTK User's Guide 66
- MTK wizard interface 64
- multiple concurrent transactions 230
- multiprocessor architectures 5

N

named pipe 40, 197
named pipes 165
naming conventions 46
near-line 9
NET applications 270
network communication 26
network encryption 309
Node DataBlade 13
non-default page size 20
non-logging temporary table 123
non-sharable memory 18
non-trusted connection 230
numeric data type 189

O

object data types 194
object definitions 18, 47, 67, 74, 77, 84, 99
object deployment 73
Object Interface for C++ 216
object privileges 18
object-relational 6, 12
OCI (Oracle Call Interface) 248
OCI driver 241
ODBC (Open Database Connectivity) 38, 61, 79, 211, 215, 219, 248, 254, 285, 359, 362
OLAP (Online Analytical Processing) 4, 10
OLE DB data providers 216
OLTP 4–5
ON-Bar 7, 9, 287
oncheck utility 297
ondblog utility 298
ongoing application development 46
oninit utility 283, 298
online analytical processing 4
online transaction processing 4–5
onlog utility 302
onmode utility 301
onparams utility 299
onpladm utility 201
onspaces utility 286, 300
onstat utility 296
ontape 9
ontape utility 287
opaque data type 147
Open Admin Tool 302
OpenAdmin Tool 12
optimization 12

optimizer 11
optimizer directives 136
Oracle 4
Oracle alert log file 31
Oracle architecture 16
Oracle background process 24
Oracle based Perl program 258
Oracle based stored procedure 262
Oracle bitmap index 129
Oracle block size 32
Oracle built-in functions 68
Oracle Call Interface 248, 250
Oracle constraint syntax 114
Oracle CREATE TABLE 113
Oracle data dictionary 15, 36
Oracle data types 120
Oracle database 18, 28–30, 33, 35, 46, 63, 78–80, 92–93, 97, 133, 168, 171, 178–179, 192, 229, 241, 248, 255, 261–262, 274, 322
Oracle datafile 28
Oracle datafiles 19
Oracle DBLink 133
Oracle DUAL table 124
Oracle Global Partition 128
Oracle INSERT 142
Oracle instance 18, 23–24, 29–30, 38, 79, 264
Oracle Java applications 240
Oracle JDBC driver 79
Oracle LONG RAW 120
Oracle macros 153
Oracle MLSLABEL 120
Oracle native driver 63
Oracle Net Configuration Assistant 38
Oracle Net Manager 38
Oracle Net Services 38
Oracle Net Services. 38
Oracle OCI 211
Oracle optimizer hints 136
Oracle packages 148
Oracle PARALLEL clause 116
Oracle Partition Table 128
Oracle partitioning 118
 composite range-hash 118
 hash 118
 list 118
 range 118
Oracle PHP program 262
Oracle PL/SQL 111
Oracle Pro*C 211

- Oracle Pro*C applications 229
- Oracle RAW 120
- Oracle redo log buffer 20
- Oracle redo log files 29
- Oracle ROWID 121
- Oracle segment 33
- Oracle sequence 130
- Oracle stored procedure 257
- Oracle tablespace 33–34, 62, 89, 91, 113
- Oracle trigger 108, 133
- Oracle user process 23
- outer join specification 142
- owner naming 112

P

- packages 74, 101
- page 20–21, 32–34, 61–62, 68, 113, 126, 129, 158, 173, 207, 297
- page cleaning 25
- page locking 158
- page-level locking 158
- parallel backup and restore 6
- parallel data query 6, 10, 208
- parallel scan feature 11
- parallelism 10
- parameter bindings 250
- parameter marker 258
- partitioning 9
- password file 29
- PDQ (Parallel Data Query) 7, 10, 42, 116, 208, 324
- PDQ priority level 10
- performance 2, 4
- performance considerations 208
- performance features 10
- performance requirements 46
- performance testing 54
- Perl 211
- PERL database interface 217
- Perl programs 254
- permanent tables 123
- PHP 12, 211
- PHP applications 259
- PHP cursor handling 266
- PHP database interfaces 261, 263
- PHP non-persistent connections 264
- PHP persistent connections 264
- PHYSBUFF parameter 21
- physical database architecture 15

- physical log buffer 21
- physical log file 26, 29, 31
- planning a migration 45
- pluggable authentication modules 215
- post-migration steps 43
- post-migration tests 55
- pre-migration steps 43
- primary 9
- primary chunk 290
- Primary Key Constraint 114
- privilege information 35
- Pro*C precompiler 232
- procedures 74, 101
- process architectures 23
- Process Monitor 24
- Processes architecture 15
- processor-based pricing 41
- Program Global Area 18
- Project Management panel 75
- proprietary application features 224
- pseudo-columns 138
- pseudo-tables 36

Q

- quality assurance 50
- query drill-down 307
- query optimizers 135
- Queue Monitor 24

R

- range partitioning 118
- rank queries 141
- read consistency 154
- real-time 8, 10, 13
- recoverer 24
- recovery 4, 10
- Recovery Time Objective 363
- Redbooks Web site 367
 - Contact us xiv
- redo log buffer 18–20, 24
- redo log files 19, 24, 28–30
- referential integrity 10
- Refine tab 85, 104
- regulatory requirements 11
- Remote Standalone secondary server 291
- renaming objects 90
- repeatable read 156
- replicate 8

- replication 8–10
- repository 224
- restore 6, 9, 12
- restore spaces 289
- rich data types 4
- role-based authority 160
- rollback segments 32
- roll-out 225
- root dbspace 30–31, 34, 284
- round robin 117
- ROW lock 158
- Row-level locking 158
- RSS (Remote Standalone Secondary) 9–10, 293, 302, 324, 363
- RTO 363

S

- SBspace 35
- scalability 4–7
- scalable architecture 8
- scan thread 11
- Scheduler 306
- schema conversion 167
- Script Input Error 85
- SDS (Shared Disk Secondary) 10, 290, 292, 302, 324
- secondary 9
- security 6–7, 12, 29, 39, 50, 55–56, 161, 178, 264, 280, 308, 310–312
- security features 11
- security label components 311
- security labels 311
- security options 159
- security planning 46
- security policies 310
- semantic errors 137
- sequence object 175
- sequence objects 131
- sequences 73
- server 5–6
- servlets 241
- session data 22
- session information 18–19, 347
- session pools 22, 323
- shared memory 17–20, 22–23, 26, 31, 34, 36, 40, 283–285, 296, 298, 301–302, 323, 340–341, 352
- shared memory connection 22
- shared pool 18–19
- shared-memory header 20
- shared-memory structures 36
- simple large objects 120
- singleton connect 229
- smallfile tablespace 33
- smart large objects 21, 35, 120
- SMP 10
- snowflake 8
- sort pools 22
- sorting pool 22
- spatial 4
- Spatial DataBlade 14
- SPL routine cache 22
- SPL routine caches 22
- SPL statements 108
- SQL 7–8, 10–13
- SQL Administration API 305
- SQL functions 241
- SQL load statement 197
- SQL query changes 224
- SQL statement cache 22, 37
- SQL statements 135
- SQL translator 62, 99
- SQL*Plus 38, 170, 195
- SQLCA (SQL Communication Area) 149, 237–238
- SQLj 240
- stack space 18
- stacks 22
- standards compliance 50
- start the migration 83
- starting the MTK 72
- static embedded SQL 227
- statistical information 11
- storage-space backup 30, 299
- stored procedure 247
- stored procedure calls 243
- stored procedure compiler 152
- stored procedure result set 266
- stored procedures 2, 56, 236, 241
- stream pipe 40
- synchronous 9
- synonyms 131
- syntactic errors 137
- sysadmin database 307
- sysmaster database 307
- system catalog 15, 22, 35–36, 137–138, 161, 322, 348
- system catalog tables 36
- System Global Area 18

system monitor 24
system monitoring interface 36, 296

T

table fragmentation 117
table partitioning 11, 117
tables 14
tablespaces 28, 32–33, 62, 68, 89, 91, 113, 322
Tcl/TK programming environment 218
template files 280
temporary dbspace 34
temporary segments 32
temporary tables 123
terminology 15, 41
test phase 52
testing 53
testing considerations 55
testing methodology 53
testing process 53
text search 13
Text Search DataBlade 13
thread 11, 21, 25, 27, 243, 297
thread data 22
time planning 167
time-based data types 192
time-series 4
TimeSeries DataBlade 13
TimeSeries Real-Time Loader 13
TNS 38
tool selection 46
total cost of ownership 4
trace log file 29
trace log files 28–29
transaction logging 123
transitioning 2
translating tables and indexes 68
translation file 106
translation Information 105
translation ratio 105
translator error 85
translator information 85
translator information messages 87
translator warning 85
Transparent Network Substrate 38
triggers 56, 74, 101
TRUNCATE statement 146

U

UDF (User-Defined Function) 56, 68, 74, 237, 241, 333
UDR (User-Defined Routine) 11, 126, 147, 229, 328, 349
unbuffered logging 112
Unique Constraint 115
UNIX 5
unloading blobs 195
UPDATE 11
update concurrency 158
user authentication 229
user defined dbspace 34
user defined types 12
user education 225
user interfaces 64
user-built applications 229
user-defined C structures 232
user-defined data 4
user-defined data type 35, 214
user-defined types 121

V

value unit 41
variable declaration 151
variable definitions 151
variables 231
video 4, 14
Video Foundation DataBlade 14
views 73
virtual processors 10, 301, 305, 342
virtual shared memory 22
volume testing 55
VP (virtual processor) 10, 17, 20, 24–26, 42, 305, 343

W

WHENEVER statement 237
Windows 5, 7–8

X

XMI 67
XML 4

Archived



Migrating from Oracle... to IBM Informix Dynamic Server on Linux, Unix, and Windows

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



Migrating from Oracle . . . to IBM Informix Dynamic Server on Linux, Unix, and Windows



Developing a Data and Applications Migration Methodology

In this IBM Redbooks publication, we discuss considerations, and describe a methodology, for migrating from Oracle 10g to the Informix Dynamic Server (IDS). We focus primarily on the basic topic areas of data, applications, and administration, providing information about the differences in features and functionality in areas such as data types, DML, DDL, and Stored Procedures. Understanding the features and functionality of the two products will better enable your decisions as you develop your migration plan.

Understanding IDS and Oracle DBMS Functionality

We provide a migration methodology and discuss the processes for installing and using the IBM Migration Toolkit (MTK) to migrate the database objects and data from Oracle to IDS. We also illustrate, with examples, how to convert stored procedures, functions, and triggers. Application programming and conversion considerations are also discussed. The laboratory examples are performed under Oracle 10g and IDS Version 11.5. However, the processes and examples can also be applied to Oracle 7, 8, and 9i.

Using the IBM Migration Tool Kit as Your Guide

With this information, you can gather and document your conversion requirements, develop your required migration methodology, and then plan and execute the migration activities in an orderly and cost effective manner.

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