



IBM Informix Guide to SQL: Reference



IBM Informix Guide to SQL: Reference

Note!

Before using this information and the product it supports, read the information in “Notices” on page E-1.

First Edition (December 2004)

This document contains proprietary information of IBM. It is provided under a license agreement and is protected by copyright law. The information contained in this publication does not include any product warranties, and any statements provided in this manual should not be interpreted as such.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1996, 2004. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Introduction.	ix
About This Manual	ix
Types of Users	x
Software Dependencies	x
Assumptions About Your Locale	x
Demonstration Database	xi
Documentation Conventions	xii
Typographical Conventions	xii
Feature, Product, and Platform	xii
Syntax Diagrams	xiii
Example Code Conventions	xvii
Additional Documentation	xviii
Installation Guides	xviii
Online Notes	xviii
Informix Error Messages	xx
Manuals	xxi
Online Help	xxi
Accessibility	xxi
IBM Informix Dynamic Server Version 10.0 and CSDK Version 2.90 Documentation Set	xxi
Compliance with Industry Standards	xxiv
IBM Welcomes Your Comments	xxv
Chapter 1. System Catalog Tables	1-1
Objects That the System Catalog Tables Track	1-2
Using the System Catalog	1-3
Structure of the System Catalog	1-10
SYSAGGREGATES (IDS)	1-12
SYSAMS (IDS)	1-12
SYSATTRYPES (IDS)	1-15
SYSBLOBS	1-16
SYSCASTS (IDS)	1-17
SYSCHECKS	1-18
SYSCHECKUDRDEP (IDS)	1-18
SYSCOLATTRIBS (IDS)	1-19
SYSCOLAUTH	1-20
SYSCOLDEPEND	1-20
SYSCOLUMNS	1-22
SYSCONSTRAINTS	1-26
SYSDEFAULTS	1-27
SYSDEPEND	1-27
SYSDIRECTIVES (IDS)	1-28
SYSDISTRIB	1-28
SYSDOMAINS (IDS)	1-30
SYSERRORS (IDS)	1-30
SYSEXTCOLS (XPS)	1-31

SYSEXTDFILES (XPS)	1-31
SYSEXTERNAL (XPS)	1-32
SYSFRAGAUTH (IDS)	1-33
SYSFRAGMENTS	1-33
SYSINDEXES	1-36
SYSINDICES (IDS)	1-38
SYSINHERITS (IDS)	1-39
SYSLANGAUTH (IDS)	1-39
SYSLOGMAP (IDS)	1-40
SYSNEWDEPEND (XPS)	1-40
SYSOBJSTATE (IDS)	1-41
SYSOPCLASSES (IDS)	1-41
SYSOPCLSTR	1-42
SYSPROCAUTH	1-44
SYSPROCEDURE	1-44
SYSPROCEDURES	1-45
SYSPROCPLAN	1-48
SYSREFERENCES	1-49
SYSREPOSITORY (XPS)	1-49
SYSROLEAUTH	1-50
SYSROUTINELANGS (IDS)	1-50
SYSSEQUENCES (IDS)	1-51
SYSSYNONYMS	1-51
SYSSYNTABLE	1-51
SYSTABAMDATA (IDS)	1-52
SYSTABAUTH	1-53
SYSTABLES	1-54
SYSTRACECLASSES (IDS)	1-56
SYSTRACEMSGS (IDS)	1-56
SYSTRIGBODY	1-57
SYSTRIGGERS	1-58
SYSUSERS	1-59
SYSVIEWS	1-59
SYSVIOLATIONS	1-60
SYSXTDDESC (IDS)	1-61
SYSXTDTYPEAUTH (IDS)	1-61
SYSXTDTYPES (IDS)	1-61
Information Schema (IDS)	1-63
Generating the Information Schema Views	1-63
Accessing the Information Schema Views	1-64
Structure of the Information Schema Views	1-64
Chapter 2. Data Types	2-1
Summary of Data Types	2-2
Description of Data Types	2-6
BIGINT (XPS)	2-6
BLOB (IDS)	2-6
BOOLEAN (IDS)	2-7
BYTE	2-7
CHAR(n)	2-8

CHARACTER(n)	2-9
CHARACTER VARYING(m,r)	2-9
CLOB (IDS)	2-10
DATE	2-11
DATETIME	2-11
DEC	2-15
DECIMAL	2-15
Distinct (IDS)	2-17
DOUBLE PRECISION	2-18
FLOAT(n)	2-18
INT	2-19
INT8	2-19
INTEGER	2-19
INTERVAL	2-19
LIST(e) (IDS)	2-22
LVARCHAR(m) (IDS)	2-23
MONEY(p,s)	2-24
MULTISET(e) (IDS)	2-25
NCHAR(n)	2-26
NUMERIC(p,s)	2-26
NVARCHAR(m,r)	2-26
Opaque (IDS)	2-26
REAL	2-27
ROW, Named (IDS)	2-27
ROW, Unnamed (IDS)	2-28
SERIAL(n)	2-30
SERIAL8(n)	2-31
SET(e) (IDS)	2-32
SMALLFLOAT	2-33
SMALLINT	2-34
TEXT	2-34
VARCHAR(m,r)	2-36
Built-In Data Types	2-38
Large-Object Data Types	2-38
Time Data Types	2-40
Extended Data Types (IDS)	2-46
Complex Data Types	2-46
Distinct Data Types	2-49
Opaque Data Types	2-49
Data Type Casting and Conversion	2-50
Using Built-in Casts	2-50
Using User-Defined Casts	2-53
Determining Which Cast to Apply	2-54
Casts for Distinct Types	2-54
What Extended Data Types Can Be Cast?	2-56
Operator Precedence	2-56
Chapter 3. Environment Variables	3-1
Types of Environment Variables	3-3
Where to Set Environment Variables on UNIX	3-4

Where to Set Environment Variables on Windows	3-5
Using Environment Variables on UNIX	3-6
Setting Environment Variables in a Configuration File	3-6
Setting Environment Variables at Login Time	3-6
Syntax for Setting Environment Variables	3-7
Unsetting Environment Variables	3-7
Modifying an Environment-Variable Setting	3-8
Viewing Your Environment-Variable Settings.	3-8
Checking Environment Variables with the chkenv Utility	3-9
Rules of Precedence	3-9
Using Environment Variables on Windows	3-10
Environment Settings for Native Windows Applications	3-10
Environment Settings for Command-Prompt Utilities	3-11
Rules of Precedence	3-13
List of Environment Variables	3-14
Environment Variables.	3-18
AC_CONFIG	3-18
AFDEBUG.	3-18
ANSIOWNER (IDS)	3-18
BIG_FET_BUF_SIZE (XPS)	3-19
CPFIRST	3-20
DBACCNOIGN	3-20
DBANSIWARN	3-21
DBBLOBBUF	3-22
DBCENTURY.	3-22
DBDATE	3-25
DBDELIMITER	3-28
DBEDIT	3-28
DBFLTMASK	3-29
DBLANG	3-29
DBMONEY	3-30
DBNLS (IDS)	3-32
DBONPLOAD (IDS)	3-33
DBPATH	3-33
DBPRINT	3-35
DBREMOTECMD (UNIX)	3-36
DBSPACETEMP	3-36
DBTEMP (IDS)	3-38
DBTIME	3-39
DBUPSPACE	3-41
DEFAULT_ATTACH	3-42
DELIMIDENT	3-42
ENVIGNORE (UNIX)	3-43
FET_BUF_SIZE	3-44
GLOBAL_DETACH_INFORM (XPS)	3-45
IBM_XPS_PARAMS (XPS)	3-45
IFMX_CART_ALRM (XPS)	3-46
IFMX_HISTORY_SIZE (XPS)	3-47
IFMX_OPT_FACT_TABS (XPS)	3-47
IFMX_OPT_NON_DIM_TABS (XPS)	3-48

IFX_DEF_TABLE_LOCKMODE (IDS)	3-48
IFX_DIRECTIVES	3-49
IFX_EXTDIRECTIVES	3-50
IFX_LONGID	3-51
IFX_NETBUF_PVTPPOOL_SIZE (UNIX)	3-52
IFX_NETBUF_SIZE	3-52
IFX_NO_TIMELIMIT_WARNING	3-52
IFX_PAD_VARCHAR (IDS)	3-53
IFX_UPDDESC (IDS)	3-53
IFX_XASTDCOMPLIANCE_XAEND	3-53
IMCADMIN	3-54
IMCCONFIG	3-55
IMCSERVER	3-55
INFORMIXC (UNIX)	3-55
INFORMIXCONCSMCFG (IDS)	3-56
INFORMIXCONRETRY	3-56
INFORMIXCONTIME	3-57
INFORMIXCPPMAP (IDS)	3-58
INFORMIXDIR	3-58
INFORMIXKEYTAB (UNIX)	3-59
INFORMIXOPCACHE (IDS)	3-59
INFORMIXSERVER	3-60
INFORMIXSHMBASE (UNIX)	3-60
INFORMIXSQLHOSTS	3-61
INFORMIXSTACKSIZE	3-62
INFORMIXTERM (UNIX)	3-62
INF_ROLE_SEP (IDS)	3-63
INTERACTIVE_DESKTOP_OFF (Windows)	3-64
ISM_COMPRESSION	3-64
ISM_DEBUG_FILE	3-64
ISM_DEBUG_LEVEL	3-65
ISM_ENCRYPTION	3-65
ISM_MAXLOGSIZE	3-65
ISM_MAXLOGVERS	3-66
JAR_TEMP_PATH (IDS)	3-66
JAVA_COMPILER (IDS)	3-66
JVM_MAX_HEAP_SIZE (IDS)	3-66
LD_LIBRARY_PATH (UNIX)	3-67
LIBERAL_MATCH (XPS)	3-67
LIBPATH (UNIX)	3-68
NODEFDAC	3-68
ONCONFIG	3-68
OPTCOMPIND	3-69
OPTMSG	3-70
OPTOFC	3-70
OPT_GOAL (IDS, UNIX)	3-71
PATH	3-71
PDQPRIORITY	3-72
PLCONFIG (IDS)	3-74
PLOAD_LO_PATH (IDS)	3-74

PLOAD_SHMBASE (IDS)	3-74
PSORT_DBTEMP	3-75
PSORT_NPROCS	3-76
RTREE_COST_ADJUST_VALUE (IDS)	3-77
SHLIB_PATH (UNIX)	3-77
STMT_CACHE (IDS)	3-77
TERM (UNIX)	3-78
TERMCAP (UNIX)	3-78
TERMINFO (UNIX)	3-79
THREADLIB (UNIX)	3-79
TOBIGINT (XPS)	3-80
USETABLEAME (IDS)	3-80
XFER_CONFIG (XPS)	3-81
Index of Environment Variables	3-81
Appendix A. The stores_demo Database	A-1
Appendix B. The sales_demo and superstores_demo Databases	B-1
Appendix C. Accessibility	C-1
Glossary	D-1
Notices	E-1
Index	X-1

Introduction

About This Manual	ix
Types of Users	x
Software Dependencies	x
Assumptions About Your Locale	x
Demonstration Database	xi
Documentation Conventions	xii
Typographical Conventions.	xii
Feature, Product, and Platform	xii
Syntax Diagrams	xiii
How to Read a Command-Line Syntax Diagram	xv
Keywords and Punctuation	xvi
Identifiers and Names	xvi
Example Code Conventions	xvii
Additional Documentation	xviii
Installation Guides	xviii
Online Notes	xviii
Locating Online Notes	xix
Online Notes Filenames	xx
Informix Error Messages.	xx
Manuals	xxi
Online Manuals	xxi
Printed Manuals	xxi
Online Help	xxi
Accessibility	xxi
IBM Informix Dynamic Server Version 10.0 and CSDK Version 2.90 Documentation Set	xxi
Compliance with Industry Standards	xxiv
IBM Welcomes Your Comments	xxv

About This Manual

This manual includes information about system catalog tables, data types, and environment variables that IBM Informix products use. It also includes a glossary that contains definitions of common terms found in IBM Informix documentation and a description of the demonstration databases that Version 10.0 of IBM Informix Dynamic Server and Version 8.5 of IBM Informix Extended Parallel Server provide.

This manual is one of a series of manuals that discusses the Informix implementation of SQL. The *IBM Informix: Guide to SQL Syntax* contains all the syntax descriptions for SQL and stored procedure language (SPL). The *IBM Informix: Guide to SQL Tutorial* shows how to use basic and advanced SQL and SPL routines to access and manipulate the data in your databases.

The *IBM Informix: Database Design and Implementation Guide* shows how to use SQL to implement and manage your databases.

See the documentation notes files, which are described in the section “Online Notes” on page xviii of this Introduction, for a list of the manuals in the documentation set of your Informix database server.

Types of Users

This manual is written for the following users:

- Database users
- Database administrators
- Database server administrators
- Database-application programmers
- Performance engineers

This manual assumes that you have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational databases or exposure to database concepts
- Some experience with computer programming
- Some experience with database server administration, operating-system administration, or network administration

If you have limited experience with relational databases, SQL, or your operating system, refer to the *IBM Informix: Getting Started Guide* for your database server for a list of supplementary titles.

Software Dependencies

This manual is written with the assumption that you are using one of the following database servers:

- IBM Informix Dynamic Server, Version 10.0
- IBM Informix Extended Parallel Server, Version 8.50

Assumptions About Your Locale

IBM Informix products can support many languages, cultures, and code sets. All the information related to character set, collation, and representation of numeric data, currency, date, and time is brought together in a single environment, called a Global Language Support (GLS) locale.

This manual assumes that your database uses the default locale. This default is **en_us.8859-1** (ISO 8859-1) on UNIX platforms or **en_us.1252** (Microsoft 1252) in Windows environments. This locale supports U.S. English format conventions for displaying and entering date, time, number, and currency

values. It also supports the ISO 8859-1 (on UNIX and Linux) or Microsoft 1252 (on Windows) code set, which includes the ASCII code set plus many 8-bit characters such as é, è, and ñ.

If you plan to use nondefault characters in your data or in SQL identifiers, or if you plan to use other collation rules for sorting character data, you need to specify the appropriate nondefault locale.

For instructions on how to specify a nondefault locale, and for additional syntax and other considerations related to GLS locales, see the *IBM Informix: GLS User's Guide*.

Demonstration Database

The DB–Access utility, which is provided with the database server products, includes one or more of the following demonstration databases:

- The **stores_demo** database illustrates a relational schema with information about a fictitious wholesale sporting-goods distributor. Many examples in IBM Informix manuals are based on the **stores_demo** database.

Extended Parallel Server

- The **sales_demo** database illustrates a dimensional schema for data-warehousing applications. For conceptual information about dimensional data modeling, see the *IBM Informix: Database Design and Implementation Guide*.

End of Extended Parallel Server

Dynamic Server

- The **superstores_demo** database illustrates an object-relational schema. The **superstores_demo** database contains examples of extended data types, type and table inheritance, and user-defined routines.

End of Dynamic Server

For information about how to create and populate the demonstration databases, see the *IBM Informix: DB–Access User's Guide*. For descriptions of the databases and their contents, see Appendix A, “The stores_demo Database,” on page A-1 and Appendix B, “The sales_demo and superstores_demo Databases,” on page B-1.

The scripts that you use to install the demonstration databases reside in the **\$INFORMIXDIR/bin** directory on UNIX platforms and in the **%INFORMIXDIR%\bin** directory in Windows environments.

Documentation Conventions

This section describes the conventions that this manual uses. These conventions make it easier to gather information from this and other volumes in the documentation set.

The following conventions are discussed:

- Typographical conventions
- Other conventions
- Syntax diagrams
- Command-line conventions
- Example code conventions

Typographical Conventions

This manual uses the following conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth.

Convention	Meaning
KEYWORD	All primary elements in a programming language statement (keywords) appear in uppercase letters in a serif font.
<i>italics</i> <i>italics</i> <i>italics</i>	Within text, new terms and emphasized words appear in italics. Within syntax and code examples, variable values that you are to specify appear in italics.
boldface boldface	Names of program entities (such as classes, events, and tables), environment variables, file and pathnames, and interface elements (such as icons, menu items, and buttons) appear in boldface.
monospace <i>monospace</i>	Information that the product displays and information that you enter appear in a monospace typeface.
KEYSTROKE	Keys that you are to press appear in uppercase letters in a sans serif font.
>	This symbol indicates a menu item. For example, “Choose Tools > Options ” means choose the Options item from the Tools menu.

Tip: When you are instructed to “enter” characters or to “execute” a command, immediately press RETURN after the entry. When you are instructed to “type” the text or to “press” other keys, no RETURN is required.

Feature, Product, and Platform

Feature, product, and platform markup identifies paragraphs that contain feature-specific, product-specific, or platform-specific information. Some

examples of this markup follow:

Dynamic Server

Identifies information that is specific to IBM Informix Dynamic Server

End of Dynamic Server

Extended Parallel Server

Identifies information that is specific to IBM Informix Extended Parallel Server

End of Extended Parallel Server

UNIX Only

Identifies information that is specific to UNIX platforms

End of UNIX Only

Windows Only

Identifies information that is specific to the Windows environment

End of Windows Only

This markup can apply to one or more paragraphs within a section. When an entire section applies to a particular product or platform, this is noted as part of the heading text, for example:

Table Sorting (Linux Only)

Syntax Diagrams

This guide uses syntax diagrams built with the following components to describe the syntax for statements and all commands other than system-level commands.

Note: Starting in 2004, syntax diagrams have been reformatted to conform to the IBM standard.

Syntax diagrams depicting SQL and command-line statements have changed in the following ways:

- The symbols at the beginning and end of statements are now double arrows instead of a vertical line at the end.
- The symbols at the beginning and end of syntax segment diagrams are now vertical lines instead of arrows.

- How many times a loop can be repeated is now explained in a diagram footnote instead of a number in a gate symbol.
- Syntax statements that are longer than one line now continue on the next line instead of looping down with a continuous line.
- Product or condition-specific paths are now explained in diagram footnotes instead of icons.

The following table describes syntax diagram components.

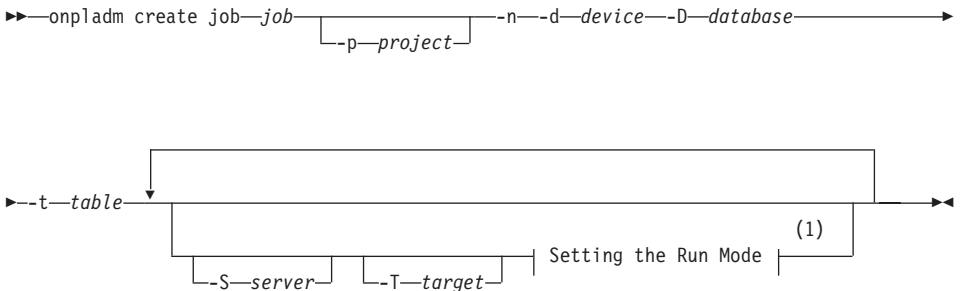
Component represented in PDF	Component represented in HTML	Meaning
	>>-----	Statement begins.
	----->	Statement continues on next line.
	>-----	Statement continues from previous line.
	-----><	Statement ends.
	-----SELECT-----	Required item.
	---+-----+--- '-----LOCAL-----'	Optional item.
	---+-----+--- +--DISTINCT--+ '---UNIQUE-----'	Required item with choice. One and only one item must be present.
	---+-----+--- +--FOR UPDATE--+ '--FOR READ ONLY--'	Optional items with choice are shown below the main line, one of which you might specify.
	.---NEXT-----. ---+-----+--- +--PRIOR-----+ '---PREVIOUS-----'	The values below the main line are optional, one of which you might specify. If you do not specify an item, the value above the line will be used as the default.

Component represented in PDF	Component represented in HTML	Meaning
	<pre> .v -----,----- +---index_name---+ '---table_name---' </pre>	Optional items. Several items are allowed; a comma must precede each repetition.
	<pre>>>- Table Reference -<<</pre>	Reference to a syntax segment.
<p>Table Reference</p>	<pre> ---+-----view-----+--- +-----table-----+ '-----synonym-----' </pre>	Syntax segment.

How to Read a Command-Line Syntax Diagram

The following command-line syntax diagram uses some of the elements listed in the table in the previous section.

Creating a No-Conversion Job

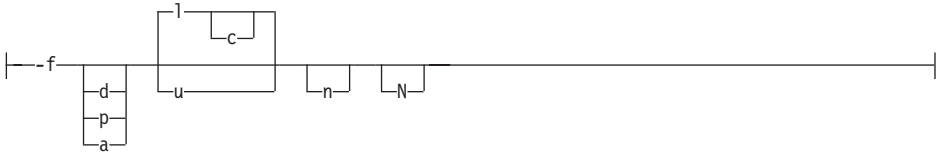


Notes:

- 1 See page 17-4

The second line in this diagram has a segment named “Setting the Run Mode,” which according to the diagram footnote, is on page 17-4. This segment is shown in the following segment diagram (the diagram uses segment start and end components).

Setting the Run Mode:



To construct a command correctly, start at the top left with the command. Follow the diagram to the right, including the elements that you want. The elements in the diagram are case sensitive.

The Creating a No-Conversion Job diagram illustrates the following steps:

1. Type **onpladm create job** and then the name of the job.
2. Optionally, type **-p** and then the name of the project.
3. Type the following required elements:
 - **-n**
 - **-d** and the name of the device
 - **-D** and the name of the database
 - **-t** and the name of the table
4. Optionally, you can choose one or more of the following elements and repeat them an arbitrary number of times:
 - **-S** and the server name
 - **-T** and the target server name
 - The run mode. To set the run mode, follow the Setting the Run Mode segment diagram to type **-f**, optionally type **d**, **p**, or **a**, and then optionally type **l** or **u**.
5. Follow the diagram to the terminator.

Your diagram is complete.

Keywords and Punctuation

Keywords are words reserved for statements and all commands except system-level commands. When a keyword appears in a syntax diagram, it is shown in uppercase letters. When you use a keyword in a command, you can write it in uppercase or lowercase letters, but you must spell the keyword exactly as it appears in the syntax diagram.

You must also use any punctuation in your statements and commands exactly as shown in the syntax diagrams.

Identifiers and Names

Variables serve as placeholders for identifiers and names in the syntax diagrams and examples. You can replace a variable with an arbitrary name,

identifier, or literal, depending on the context. Variables are also used to represent complex syntax elements that are expanded in additional syntax diagrams. When a variable appears in a syntax diagram, an example, or text, it is shown in *lowercase italic*.

The following syntax diagram uses variables to illustrate the general form of a simple SELECT statement.

►—SELECT—*column_name*—FROM—*table_name*—►

When you write a SELECT statement of this form, you replace the variables *column_name* and *table_name* with the name of a specific column and table.

Example Code Conventions

Examples of SQL code occur throughout this manual. Except as noted, the code is not specific to any single IBM Informix application development tool.

If only SQL statements are listed in the example, they are not delimited by semicolons. For instance, you might see the code in the following example:

```
CONNECT TO stores_demo
...

DELETE FROM customer
    WHERE customer_num = 121
...

COMMIT WORK
DISCONNECT CURRENT
```

To use this SQL code for a specific product, you must apply the syntax rules for that product. For example, if you are using DB–Access, you must delimit multiple statements with semicolons. If you are using an SQL API, you must use EXEC SQL at the start of each statement and a semicolon (or other appropriate delimiter) at the end of the statement.

Tip: Ellipsis points in a code example indicate that more code would be added in a full application, but it is not necessary to show it to describe the concept being discussed.

For detailed directions on using SQL statements for a particular application development tool or SQL API, see the manual for your product.

Additional Documentation

For additional information, refer to the following types of documentation:

- Installation guides
- Online notes
- Informix error messages
- Manuals
- Online help

Installation Guides

Installation guides are located in the `/doc` directory of the product CD or in the `/doc` directory of the product's compressed file if you downloaded it from the IBM Web site. Alternatively, you can obtain installation guides from the IBM Informix Online Documentation site at <http://www.ibm.com/software/data/informix/pubs/library/>.

Online Notes

The following sections describe the online files that supplement the information in this manual. Please examine these files before you begin using your IBM Informix product. They contain vital information about application and performance issues.

Online File	Description	Format
TOC Notes	The TOC (Table of Contents) notes file provides a comprehensive directory of hyperlinks to the release notes, the fixed and known defects file, and all the documentation notes files for individual manual titles.	HTML
Documentation Notes	The documentation notes file for each manual contains important information and corrections that supplement the information in the manual or information that was modified since publication.	HTML, text
Release Notes	The release notes file describes feature differences from earlier versions of IBM Informix products and how these differences might affect current products. For some products, this file also contains information about any known problems and their workarounds.	HTML, text
Machine Notes	(Non-Windows platforms only) The machine notes file describes any platform-specific actions that you must take to configure and use IBM Informix products on your computer.	text
Fixed and Known Defects File	This text file lists issues that have been identified with the current version. It also lists customer-reported defects that have been fixed in both the current version and in previous versions.	text

Locating Online Notes

Online notes are available from the IBM Informix Online Documentation site at <http://www.ibm.com/software/data/informix/pubs/library/>. Additionally you can locate these files before or after installation as described below.

Before Installation

All online notes are located in the `/doc` directory of the product CD. The easiest way to access the documentation notes, the release notes, and the fixed and known defects file is through the hyperlinks from the TOC notes file.

The machine notes file and the fixed and known defects file are only provided in text format.

After Installation

On UNIX platforms in the default locale, the documentation notes, release notes, and machine notes files appear under the `$INFORMIXDIR/release/en_us/0333` directory.

Dynamic Server

On Windows the documentation and release notes files appear in the **Informix** folder. To display this folder, choose **Start > Programs > IBM Informix Dynamic Server *version* > Documentation Notes** or **Release Notes** from the taskbar.

Machine notes do not apply to Windows platforms.

End of Dynamic Server

Online Notes Filenames

Online notes have the following file formats:

Online File	File Format	Examples
TOC Notes	<i>prod_os_tocnotes_version.html</i>	ids_win_tocnotes_10.0.html
Documentation Notes	<i>prod_bookname_docnotes_version.html/txt</i>	ids_hpl_docnotes_10.0.html
Release Notes	<i>prod_os_relnotes_version.html/txt</i>	ids_unix_relnotes_10.0.txt
Machine Notes	<i>prod_machine_notes_version.txt</i>	ids_machine_notes_10.0.txt
Fixed and Known Defects File	<i>prod_defects_version.txt</i>	ids_defects_10.0.txt client_defects_2.90.txt
	<i>ids_win_fixed_and_known_defects_version.txt</i>	ids_win_fixed_and_known_defects_10.0.txt

Informix Error Messages

This file is a comprehensive index of error messages and their corrective actions for the Informix products and version numbers.

On UNIX platforms, use the **finderr** command to read the error messages and their corrective actions.

Dynamic Server

On Windows, use the Informix Error Messages utility to read error messages and their corrective actions. To display this utility, choose **Start > Programs > IBM Informix Dynamic Server *version* > Informix Error Messages** from the taskbar.

End of Dynamic Server

You can also access these files from the IBM Informix Online Documentation site at <http://www.ibm.com/software/data/informix/pubs/library/>.

Manuals

Online Manuals

A CD that contains your manuals in electronic format is provided with your IBM Informix products. You can install the documentation or access it directly from the CD. For information about how to install, read, and print online manuals, see the installation insert that accompanies your CD. You can also obtain the same online manuals from the IBM Informix Online Documentation site at <http://www.ibm.com/software/data/informix/pubs/library/>.

Printed Manuals

To order hardcopy manuals, contact your sales representative or visit the IBM Publications Center Web site at <http://www.ibm.com/software/howtobuy/data.html>.

Online Help

IBM Informix online help, provided with each graphical user interface (GUI), displays information about those interfaces and the functions that they perform. Use the help facilities that each GUI provides to display the online help.

Accessibility

IBM is committed to making our documentation accessible to persons with disabilities. Our books are available in HTML format so that they can be accessed with assistive technology such as screen reader software. The syntax diagrams in our manuals are available in dotted decimal format, which is an accessible format that is available only if you are using a screen reader. For more information about the dotted decimal format, see the Accessibility appendix.

IBM Informix Dynamic Server Version 10.0 and CSDK Version 2.90 Documentation Set

The following tables list the manuals that are part of the IBM Informix Dynamic Server, Version 10.0 and the CSDK Version 2.90, documentation set. PDF and HTML versions of these manuals are available at <http://www.ibm.com/software/data/informix/pubs/library/>. You can order hardcopy versions of these manuals from the IBM Publications Center at <http://www.ibm.com/software/howtobuy/data.html>.

Table 1. Database Server Manuals

Manual	Subject
Administrator's Guide	Understanding, configuring, and administering your database server.
Administrator's Reference	Reference material for Informix Dynamic Server, such as the syntax of database server utilities onmode and onstat , and descriptions of configuration parameters, the sysmasters tables, and logical-log records.
Backup and Restore Guide	The concepts and methods you need to understand when you use the ON-Bar and ontape utilities to back up and restore data.
DB-Access User's Guide	Using the DB-Access utility to access, modify, and retrieve data from Informix databases.
DataBlade API Function Reference	The DataBlade API functions and the subset of ESQL/C functions that the DataBlade API supports. You can use the DataBlade API to develop client LIBMI applications and C user-defined routines that access data in Informix databases.
DataBlade API Programmer's Guide	The DataBlade API, which is the C-language application-programming interface provided with Dynamic Server. You use the DataBlade API to develop client and server applications that access data stored in Informix databases.
Database Design and Implementation Guide	Designing, implementing, and managing your Informix databases.
Enterprise Replication Guide	How to design, implement, and manage an Enterprise Replication system to replicate data between multiple database servers.
Error Messages file	Causes and solutions for numbered error messages you might receive when you work with IBM Informix products.
Getting Started Guide	Describes the products bundled with IBM Informix Dynamic Server and interoperability with other IBM products. Summarizes important features of Dynamic Server and the new features for each version.
Guide to SQL: Reference	Information about Informix databases, data types, system catalog tables, environment variables, and the stores_demo demonstration database.
Guide to SQL: Syntax	Detailed descriptions of the syntax for all Informix SQL and SPL statements.
Guide to SQL: Tutorial	A tutorial on SQL, as implemented by Informix products, that describes the basic ideas and terms that are used when you work with a relational database.
High-Performance Loader User's Guide	Accessing and using the High-Performance Loader (HPL), to load and unload large quantities of data to and from Informix databases.
Installation Guide for Microsoft Windows	Instructions for installing IBM Informix Dynamic Server on Windows.
Installation Guide for UNIX and Linux	Instructions for installing IBM Informix Dynamic Server on UNIX and Linux.

Table 1. Database Server Manuals (continued)

Manual	Subject
J/Foundation Developer's Guide	Writing user-defined routines (UDRs) in the Java programming language for Informix Dynamic Server with J/Foundation.
Large Object Locator DataBlade Module User's Guide	Using the Large Object Locator, a foundation DataBlade module that can be used by other modules that create or store large-object data. The Large Object Locator enables you to create a single consistent interface to large objects and extends the concept of large objects to include data stored outside the database.
Migration Guide	Conversion to and reversion from the latest versions of Informix database servers. Migration between different Informix database servers.
Optical Subsystem Guide	The Optical Subsystem, a utility that supports the storage of BYTE and TEXT data on optical disk.
Performance Guide	Configuring and operating IBM Informix Dynamic Server to achieve optimum performance.
R-Tree Index User's Guide	Creating R-tree indexes on appropriate data types, creating new operator classes that use the R-tree access method, and managing databases that use the R-tree secondary access method.
SNMP Subagent Guide	The IBM Informix subagent that allows a Simple Network Management Protocol (SNMP) network manager to monitor the status of Informix servers.
Storage Manager Administrator's Guide	Informix Storage Manager (ISM), which manages storage devices and media for your Informix database server.
Trusted Facility Guide	The secure-auditing capabilities of Dynamic Server, including the creation and maintenance of audit logs.
User-Defined Routines and Data Types Developer's Guide	How to define new data types and enable user-defined routines (UDRs) to extend IBM Informix Dynamic Server.
Virtual-Index Interface Programmer's Guide	Creating a secondary access method (index) with the Virtual-Index Interface (VII) to extend the built-in indexing schemes of IBM Informix Dynamic Server. Typically used with a DataBlade module.
Virtual-Table Interface Programmer's Guide	Creating a primary access method with the Virtual-Table Interface (VTI) so that users have a single SQL interface to Informix tables and to data that does not conform to the storage scheme of Informix Dynamic Server.

Table 2. Client/Connectivity Manuals

Manual	Subject
Client Products Installation Guide	Installing IBM Informix Client Software Developer's Kit (Client SDK) and IBM Informix Connect on computers that use UNIX, Linux, and Windows.
Embedded SQLJ User's Guide	Using IBM Informix Embedded SQLJ to embed SQL statements in Java programs.

Table 2. Client/Connectivity Manuals (continued)

Manual	Subject
ESQL/C Programmer's Manual	The IBM Informix implementation of embedded SQL for C.
GLS User's Guide	The Global Language Support (GLS) feature, which allows IBM Informix APIs and database servers to handle different languages, cultural conventions, and code sets.
JDBC Driver Programmer's Guide	Installing and using Informix JDBC Driver to connect to an Informix database from within a Java application or applet.
.NET Provider Reference Guide	Using Informix .NET Provider to enable .NET client applications to access and manipulate data in Informix databases.
ODBC Driver Programmer's Manual	Using the Informix ODBC Driver API to access an Informix database and interact with the Informix database server.
OLE DB Provider Programmer's Guide	Installing and configuring 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.
Object Interface for C++ Programmer's Guide	The architecture of the C++ object interface and a complete class reference.

Table 3. DataBlade Developer's Kit Manuals

Manual	Subject
DataBlade Developer's Kit User's Guide	Developing and packaging DataBlade modules using BladeSmith and BladePack.
DataBlade Module Development Overview	Basic orientation for developing DataBlade modules. Includes an example illustrating the development of a DataBlade module.
DataBlade Module Installation and Registration Guide	Installing DataBlade modules and using BladeManager to manage DataBlade modules in Informix databases.

Compliance with Industry Standards

The American National Standards Institute (ANSI) and the International Organization of Standardization (ISO) have jointly established a set of industry standards for the Structured Query Language (SQL). IBM Informix SQL-based products are fully compliant with SQL-92 Entry Level (published as ANSI X3.135-1992), which is identical to ISO 9075:1992. In addition, many features of IBM Informix database servers comply with the SQL-92 Intermediate and Full Level and X/Open SQL Common Applications Environment (CAE) standards.

IBM Welcomes Your Comments

We want to know about any corrections or clarifications that you would find useful in our manuals, which will help us improve future versions. Include the following information:

- The name and version of the manual that you are using
- Section and page number
- Your suggestions about the manual

Send your comments to us at the following email address:

`docinf@us.ibm.com`

This email address is reserved for reporting errors and omissions in our documentation. For immediate help with a technical problem, contact IBM Technical Support.

We appreciate your suggestions.

Chapter 1. System Catalog Tables

Objects That the System Catalog Tables Track	1-2
Using the System Catalog	1-3
Structure of the System Catalog.	1-10
SYSAGGREGATES (IDS)	1-12
SYSAMS (IDS)	1-12
SYSATTRTYPES (IDS)	1-15
SYSBLOBS.	1-16
SYSCASTS (IDS).	1-17
SYSCHECKS	1-18
SYSCHECKUDRDEP (IDS)	1-18
SYSCOLATTRIBS (IDS)	1-19
SYSCOLAUTH	1-20
SYSCOLDEPEND	1-20
SYSCOLUMNS	1-22
SYSCONSTRAINTS.	1-26
SYSDEFAULTS	1-27
SYSDEPEND	1-27
SYSDIRECTIVES (IDS).	1-28
SYSDISTRIB	1-28
SYSDOMAINS (IDS)	1-30
SYSERRORS (IDS)	1-30
SYSEXTCOLS (XPS)	1-31
SYSEXTDFILES (XPS)	1-31
SYSEXTERNAL (XPS)	1-32
SYSFRAGAUTH (IDS).	1-33
SYSFRAGMENTS	1-33
SYSINDEXES	1-36
SYSINDICES (IDS)	1-38
SYSINHERITS (IDS)	1-39
SYSLANGAUTH (IDS)	1-39
SYSLOGMAP (IDS).	1-40
SYSNEWDEPEND (XPS)	1-40
SYSOBJSTATE (IDS)	1-41
SYSOPCLASSES (IDS)	1-41
SYSOPCLSTR.	1-42
SYSPROCAUTH.	1-44
SYSPROCEDURE	1-44
SYSPROCEDURES	1-45
SYSPROCPLAN	1-48
SYSREFERENCES	1-49
SYSREPOSITORY (XPS)	1-49
SYSROLEAUTH	1-50
SYSROUTINELANGS (IDS)	1-50
SYSSEQUENCES (IDS)	1-51

SYSSYNONYMS.	1-51
SYSSYNTABLE	1-51
SYSTABAMDATA (IDS)	1-52
SYSTABAUTH	1-53
SYSTABLES	1-54
SYSTRACECLASSES (IDS)	1-56
SYSTRACEMSGS (IDS)	1-56
SYSTRIGBODY	1-57
SYSTRIGGERS	1-58
SYSUSERS.	1-59
SYSVIEWS.	1-59
SYSVIOLATIONS	1-60
SYSXTDESC (IDS)	1-61
SYSXTDTYPEAUTH (IDS)	1-61
SYSXTDTYPES (IDS)	1-61
Information Schema (IDS)	1-63
Generating the Information Schema Views	1-63
Accessing the Information Schema Views	1-64
Structure of the Information Schema Views	1-64

In This Chapter

The *system catalog* consists of tables that describe the structure of the database. Sometimes called the *data dictionary*, these tables contain everything that the database knows about itself. Each system catalog table contains specific information about elements in the database.

This chapter provides information about the structure, content, and use of the system catalog tables. It also discusses the Information Schema, which provides information about the tables, views, and columns on the current database server.

Objects That the System Catalog Tables Track

The system catalog tables maintain information about the database, including the following categories of database objects:

- Tables, views, and synonyms
- Columns, constraints, indexes, and fragments
- Triggers
- Procedures, functions, routines, and associated messages
- Authorized users, roles, and privileges to access database objects
- Data types and casts (IDS)
- Aggregate functions (IDS)
- Access methods and operator classes (IDS)
- Sequence objects (IDS)

- External optimizer directives (IDS)
- Inheritance relationships (IDS)

Using the System Catalog

Informix database servers automatically generate the system catalog tables when you create a database. You can query the system catalog tables as you would query any other table in the database. The system catalog tables for a newly created database reside in a common area of the disk called a *dbspace*. Every database has its own system catalog tables. All tables and views in the system catalog have the prefix **sys** (for example, the **systables** system catalog table).

Not all tables with the prefix **sys** are true system catalog tables. For example, the **syscdr** database supports the Enterprise Replication feature. Non-catalog tables, however, have a **tabid** \geq 100. System catalog tables all have a **tabid** $<$ 100. See later in this section and “SYSTABLES” on page 1-54 for more information about **tabid** numbers that the database server assigns to tables, views, synonyms, and (in Dynamic Server) sequence objects.

Tip: Do not confuse the system catalog tables of a database with the tables in the **sysmaster**, **sysutils**, **syscdr**, or (for Dynamic Server) the **sysuser** databases. The names of tables in those databases also have the **sys** prefix, but they contain information about an entire database server, which might manage multiple databases. Information in the **sysmaster**, **sysutils**, **syscdr**, and **sysuser** tables is primarily useful for database system administrators (DBSAs). See also the *IBM Informix: Administrator’s Guide* and *IBM Informix: Administrator’s Reference*.

The database server accesses the system catalog constantly. Each time an SQL statement is processed, the database server accesses the system catalog to determine system privileges, add or verify table or column names, and so on.

For example, the following CREATE SCHEMA block adds the **customer** table, with its indexes and privileges, to the **stores_demo** database. This block also adds a view, **california**, which restricts the data of the **customer** table to only the first and last names of the customer, the company name, and the telephone number for all customers who reside in California.

```
CREATE SCHEMA AUTHORIZATION mary1
CREATE TABLE customer (customer_num SERIAL(101), fname CHAR(15),
    lname CHAR(15), company CHAR(20), address1 CHAR(20), address2 CHAR(20),
    city CHAR(15), state CHAR(2), zipcode CHAR(5), phone CHAR(18))
GRANT ALTER, ALL ON customer TO cath1 WITH GRANT OPTION AS mary1
GRANT SELECT ON customer TO public
GRANT UPDATE (fname, lname, phone) ON customer TO nhowe
```

```
CREATE VIEW california AS
  SELECT fname, lname, company, phone FROM customer WHERE state = 'CA'
CREATE UNIQUE INDEX c_num_ix ON customer (customer_num)
CREATE INDEX state_ix ON customer (state)
```

To process this CREATE SCHEMA block, the database server first accesses the system catalog to verify the following information:

- The new table and view names do not already exist in the database. (If the database is ANSI-compliant, the database server verifies that the new names do not already exist for the specified owners.)
- The user has permission to create tables and grant user privileges.
- The column names in the CREATE VIEW and CREATE INDEX statements exist in the **customer** table.

In addition to verifying this information and creating two new tables, the database server adds new rows to the following system catalog tables:

- **systables**
- **syscolumns**
- **sysviews**
- **systabauth**
- **syscolauth**
- **sysindexes**
- **sysindices** (IDS)

The following two new rows of information are added to the **systables** system catalog table after the CREATE SCHEMA block is run.

Column Name	First Row	Second Row
tablename	customer	california
owner	maryl	maryl
partnum	16778361	0
tabid	101	102
rowsize	134	134
ncols	10	4
nindexes	2	0
nrows	0	0
created	01/26/1999	01/26/1999
version	1	0
tabtype	T	V
locklevel	P	B
npused	0	0
fextsize	16	0
nextsize	16	0
flags	0	0
site		
dbname		

Each table recorded in the **sysables** system catalog table is assigned a **tabid**, a system-assigned sequential number that uniquely identifies each table in the database. The system catalog tables receive 2-digit **tabid** numbers, and the user-created tables receive sequential **tabid** numbers that begin with 100.

The CREATE SCHEMA block adds 14 rows to the **syscolumns** system catalog table. These rows correspond to the columns in the table **customer** and the view **california**, as the following example shows.

colname	tabid	colno	coltype	collength	colmin	colmax
customer_num	101	1	262	4		
fname	101	2	0	15		
lname	101	3	0	15		
company	101	4	0	20		
address1	101	5	0	20		
address2	101	6	0	20		
city	101	7	0	15		
state	101	8	0	2		
zipcode	101	9	0	5		
phone	101	10	0	18		
fname	102	1	0	15		
lname	102	2	0	15		
company	102	3	0	20		
phone	102	4	0	18		

In the **syscolumns** table, each column within a table is assigned a sequential column number, **colno**, that uniquely identifies the column within its table. In the **colno** column, the **fname** column of the **customer** table is assigned the value 2 and the **fname** column of the view **california** is assigned the value 1.

The **colmin** and **colmax** columns are empty. These columns contain values when a column is the first key (or the only key) in an index, and has no NULL or duplicate values, and the UPDATE STATISTICS statement has been run.

The database server also adds rows to the **sysviews** system catalog table, whose **viewtext** column contains each line of the CREATE VIEW statement that defines the view. In that column, the **x0** that precedes the column names in the statement (for example, **x0.fname**) operates as an alias that distinguishes among the same columns that are used in a self-join.

The CREATE SCHEMA block also adds rows to the **systabauth** system catalog table. These rows correspond to the user privileges granted on **customer** and **california** tables, as the following example shows.

grantor	grantee	tabid	tabauth
maryl	public	101	su-idx--
maryl	cathl	101	SU-IDXAR
maryl	nhowe	101	--*-----
	maryl	102	SU-ID---

The **tabauth** column specifies the table-level privileges granted to users on the **customer** and **california** tables. This column uses an 8-byte pattern, such as s (Select), u (Update), * (column-level privilege), i (Insert), d (Delete), x (Index), a (Alter), and r (References), to identify the type of privilege. In this example, the user **nhowe** has column-level privileges on the **customer** table. Where a hyphen (-) appears, the user has not been granted the privilege whose position the hyphen occupies within the **tabauth** value.

If the **tabauth** privilege code appears in uppercase (for example, S for Select), the user has this privilege and can also grant it to others; but if the privilege code is lowercase (for example, s for Select), the user cannot grant it to others.

In addition, three rows are added to the **syscolauth** system catalog table. These rows correspond to the user privileges that are granted on specific columns in the **customer** table, as the following example shows.

grantor	grantee	tabid	colno	colauth
maryl	nhowe	101	2	-u-
maryl	nhowe	101	3	-u-
maryl	nhowe	101	10	-u-

The **colauth** column specifies the column-level privileges that are granted on the **customer** table. This column uses a 3-byte pattern, such as s (Select), u (Update), and r (References), to identify the type of privilege. For example, the user **nhowe** has Update privileges on the second column (because the **colno** value is 2) of the **customer** table (indicated by **tabid** value of 101).

The CREATE SCHEMA block adds two rows to the **sysindexes** system catalog table (the **sysindices** table for Dynamic Server). These rows correspond to the indexes created on the **customer** table, as the following example shows.

idxname	c_num_ix	state_ix
owner	maryl	maryl
tabid	101	101
idxtype	U	D
clustered		
part1	1	8
part2	0	0
part3	0	0
part4	0	0
part5	0	0
part6	0	0
part7	0	0
part8	0	0
part9	0	0
part10	0	0
part11	0	0
part12	0	0
part13	0	0
part14	0	0
part15	0	0
part16	0	0
levels		
leaves		
nunique		
clust		
idxflags		

In this table, the **idxtype** column identifies whether the created index requires unique values (U) or accepts duplicate values (D). For example, the **c_num_ix** index on the **customer.customer_num** column is unique.

Accessing the System Catalog

Normal user access to the system catalog is read-only. Users with Connect or Resource privileges cannot alter the catalog, but they can access data in the system catalog tables on a read-only basis using standard SELECT statements.

For example, the following `SELECT` statement displays all the table names and corresponding **tabid** codes of user-created tables in the database:

```
SELECT tabname, tabid FROM systables WHERE tabid > 99
```

When you use DB–Access, only the tables that you created are displayed. To display the system catalog tables, enter the following statement:

```
SELECT tabname, tabid FROM systables WHERE tabid < 100
```

You can use the **SUBSTR** or **SUBSTRING** function to select only part of a source string. To display the list of tables in columns, enter the following statement:

```
SELECT SUBSTR(tabname, 1, 18), tabid FROM systables
```

Although user **informix** and DBAs can modify most system catalog tables (only user **informix** can modify **systables**), it is strongly recommended that you do not update, delete, or insert any rows in them. Modifying system catalog tables can destroy the integrity of the database. The `ALTER TABLE` statement cannot modify the size of the next extent of system catalog tables.

For certain catalog tables of Dynamic Server, however, it is valid to add entries to the system catalog tables. For instance, in the case of the **syserrors** system catalog table and the **systracemsgs** system catalog table, a DataBlade module developer can directly insert entries that appear in these system catalog tables.

Updating System Catalog Data

In Informix database servers, the optimizer determines the most efficient strategy for executing SQL queries. The optimizer allows you to query the database without having to consider fully which tables to search first in a join or which indexes to use. The optimizer uses information from the system catalog to determine the best query strategy.

If you use the `UPDATE STATISTICS` statement to update the system catalog before executing a query, you can ensure that the information provided to the optimizer is current. When you delete or modify a table, the database server does not automatically update the related statistical data in the system catalog. For example, if you delete one or more rows in a table with the `DELETE` statement, the **nrows** column in the **systables** system catalog table, which holds the number of rows for that table, is not updated automatically.

The `UPDATE STATISTICS` statement causes the database server to recalculate data in the **systables**, **sysdistrib**, **syscolumns**, and **sysindexes** (**sysindices** for Dynamic Server) system catalog tables. After you run `UPDATE STATISTICS`, the **systables** system catalog table holds the correct value in the **nrows**

column. If you specify MEDIUM or HIGH mode when you run UPDATE STATISTICS, the **sysdistrib** system catalog table holds the updated data-distribution data.

Whenever you modify a data table extensively, use the UPDATE STATISTICS statement to update data in the system catalog. For more information on the UPDATE STATISTICS statement, see the *IBM Informix: Guide to SQL Syntax*.

Structure of the System Catalog

The following system catalog tables describe the structure of an Informix database. Here X indicates whether IDS, or XPS, or both support the table.

System Catalog Table	XPS	IDS	Page
sysaggregates		X	1-12
sysams		X	1-12
sysattrtypes		X	1-15
sysblobs	X	X	1-16
syscasts		X	1-17
syschecks	X	X	1-18
syscheckudrdep		X	1-18
syscolattribs		X	1-19
syscolauth	X	X	1-20
syscoldepend	X	X	1-20
syscolumns	X	X	1-22
sysconstraints	X	X	1-26
sysdefaults	X	X	1-27
sysdepend	X	X	1-27
sysdirectives		X	1-28
sysdistrib	X	X	1-28
sysdomains		X	1-30
syserrors		X	1-30
sysextcols	X		1-31
sysextdfiles	X		1-31
sysexternal	X		1-32
sysfragauth		X	1-33
sysfragments	X	X	1-33
sysindexes	X	X	1-36

System Catalog Table	XPS	IDS	Page
sysindices		X	1-38
sysinherits		X	1-39
syslangauth		X	1-39
syslogmap		X	1-40
sysnewdepend	X		1-40
sysobjstate		X	1-41
sysopclasses		X	1-41
sysopclstr	X	X	1-42
sysprocauth	X	X	1-44
sysprocbody	X	X	1-44
sysprocedures	X	X	1-45
sysprocplan	X	X	1-48
sysreferences	X	X	1-49
sysrepository	X		1-49
sysroleauth	X	X	1-50
sysroutinelangs		X	1-50
syssequences		X	1-51
sys synonyms	X	X	1-51
sys syntable	X	X	1-51
systabamdata		X	1-52
systabauth	X	X	1-53
systables	X	X	1-54
sys traceclasses		X	1-56
sys tracemsgs		X	1-56
sys trigbody	X	X	1-57
sys triggers	X	X	1-58
sys users	X	X	1-59
sys views	X	X	1-59
sys violations	X	X	1-60
sys xtddesc		X	1-61
sys xtypeauth		X	1-61
sys xtypeypes		X	1-61

In the default database locale (U. S. English, ISO 8859-1 codeset), character column types in these tables are CHAR and VARCHAR. For all other locales, character column types are national character types, NCHAR and NVARCHAR. For information about collation order of data types, see the *IBM Informix: GLS User's Guide*. See also Chapter 2 of this manual.

SYSAGGREGATES (IDS)

The **sysaggregates** system catalog table records user-defined aggregates (UDAs). The **sysaggregates** table has the following columns.

Column	Type	Explanation
name	VARCHAR(128)	Name of the aggregate
owner	CHAR(32)	Name of the owner of the aggregate
aggid	SERIAL	Unique code identifying the aggregate
init_func	VARCHAR(128)	Name of initialization UDR
iter_func	VARCHAR(128)	Name of iterator UDR
combine_func	VARCHAR(128)	Name of combine UDR
final_func	VARCHAR(128)	Name of finalization UDR
handlesnulls	BOOLEAN	NULL-handling indicator: t = handles NULLs f = does not handle NULLs

Each user-defined aggregate has one entry in **sysaggregates** that is uniquely identified by its identifying code (the **aggid** value). Only user-defined aggregates (aggregates that are not built in) have entries in **sysaggregates**.

Both a simple index on the **aggid** column and a composite index on the **name** and **owner** columns require unique values.

SYSAMS (IDS)

The **sysams** system catalog table contains information that is needed to use built-in access methods as well as those created by the CREATE ACCESS METHOD statement of SQL that is described in the *IBM Informix: Guide to SQL Syntax*. The **sysams** table has the following columns.

Column	Type	Explanation
am_name	VARCHAR(128)	Name of the access method
am_owner	CHAR(32)	Name of the owner of the access method

Column	Type	Explanation
am_id	INTEGER	Unique identifying code for an access method This corresponds to the am_id columns in the systables , sysindices , and sysopclasses tables.
am_type	CHAR(1)	Type of access method: P = Primary S = Secondary
am_sptype	CHAR(3)	Types of spaces where the access method can exist: A or a = all types: extspaces, dbspaces, and sbspaces. If the access method is not user-defined (that is, if it is built in or registered during database creation by the server), it supports dbspaces. D or d = dbspaces only S or s = sbspaces only (smart-large-object space) X or x = extspaces only
am_defopclass	INTEGER	Unique identifying code for default-operator class Value is the opclassid from the entry for this operator class in the sysopclasses table.
am_keyscan	INTEGER	Whether a secondary access method supports a key scan (An access method supports a key scan if it can return a key as well as a rowid from a call to the am_getnext function.) (0 = FALSE; Non-zero = TRUE)
am_unique	INTEGER	Whether a secondary access method can support unique keys (0 = FALSE; Non-zero = TRUE)
am_cluster	INTEGER	Whether a primary access method supports clustering (0 = FALSE; Non-zero = TRUE)
am_rowids	INTEGER	Whether a primary access method supports rowids (0 = FALSE; Non-zero = TRUE)
am_readwrite	INTEGER	Whether a primary access method can both read and write 0 = access method is read-only Non-zero = access method is read/write

Column	Type	Explanation
am_parallel	INTEGER	Whether an access method supports parallel execution (0 = FALSE; Non-zero = TRUE)
am_costfactor	SMALLFLOAT	The value to be multiplied by the cost of a scan in order to normalize it to costing done for built-in access methods The scan cost is the output of the am_scancost function.
am_create	INTEGER	The routine specified for the AM_CREATE purpose for this access method Value = procid for the routine in the sysprocedures table.
am_drop	INTEGER	The routine specified for the AM_DROP purpose function for this access method
am_open	INTEGER	The routine specified for the AM_OPEN purpose function for this access method
am_close	INTEGER	The routine specified for the AM_CLOSE purpose function for this access method
am_insert	INTEGER	The routine specified for the AM_INSERT purpose function for this access method
am_delete	INTEGER	The routine specified for the AM_DELETE purpose function for this access method
am_update	INTEGER	The routine specified for the AM_UPDATE purpose function for this access method
am_stats	INTEGER	The routine specified for the AM_STATS purpose function for this access method
am_scancost	INTEGER	The routine specified for the AM_SCANCOST purpose function for this access method
am_check	INTEGER	The routine specified for the AM_CHECK purpose function for this access method
am_beginscan	INTEGER	Routine specified for the AM_BEGINSCAN purpose function for this access method
am_endscan	INTEGER	The routine specified for the AM_ENDSCAN purpose function for this access method
am_rescan	INTEGER	The routine specified for the AM_RESCAN purpose function for this access method
am_getnext	INTEGER	The routine specified for the AM_GETNEXT purpose function for this access method

Column	Type	Explanation
am_getbyid	INTEGER	The routine specified for the AM_GETBYID purpose function for this access method
am_build	INTEGER	The routine specified for the AM_BUILD purpose function for this access method
am_init	INTEGER	The routine specified for the AM_INIT purpose function for this access method
am_truncate	INTEGER	The routine specified for the AM_TRUNCATE purpose function for this access method

For each of the nearly 20 columns that follow **am_costfactor**, the value is the **sysprocedures.procid** value for the corresponding routine.

The **am_sptype** column can have multiple entries. For example:

- A means the access method supports extspaces and sbspaces. If the access method is built-in, such as a B-tree, it also supports dbspaces.
- DS means the access method supports dbspaces and sbspaces.
- sx means the access method supports sbspaces and extspaces.

A composite index on the **am_name** and **am_owner** columns in this table allows only unique values. The **am_id** column has a unique index.

For information about access method functions, refer to the documentation of your access method.

SYSATTRTYPES (IDS)

The **sysattrtypes** system catalog table contains information about members of a complex data type. Each row of **sysattrtypes** contains information about elements of a collection data type or fields of a row data type.

The **sysattrtypes** table has the following columns.

Column	Type	Explanation
extended_id	INTEGER	Identifying code of an extended data type Value is the same as in the sysxdtypes table (page 1-62).
seqno	SMALLINT	Identifying code of an entry having extended_id type
levelno	SMALLINT	Position of member in collection hierarchy
parent_no	SMALLINT	Value in the seqno column of the complex data type that contains this member
fieldname	VARCHAR(128)	Name of the field in a row type Null for other complex data types
fieldno	SMALLINT	Field number sequentially assigned by system (from left to right within each row type)
type	SMALLINT	Code for the data type See the description of syscolumns.coltype (page 1-22).
length	SMALLINT	Length (in bytes) of the member
xtd_type_id	INTEGER	Code identifying this data type See the description of sysxdtypes.extended_id (page 1-62).

Two indexes on the **extended_id** column and the **xtd_type_id** column allow duplicate values. A composite index on the **extended_id** and **seqno** columns allows only unique values.

SYSBLOBS

The **sysblobs** system catalog table specifies the storage location of BYTE and TEXT column values. Its name is based on a legacy term for BYTE and TEXT columns, blobs (also known as *simple large objects*), and does not refer to the BLOB data type of Dynamic Server. The **sysblobs** table contains one row for each BYTE or TEXT column, and has the following columns.

Column	Type	Explanation
spacename	VARCHAR(128)	Name of partition, dbspace, or family
type	CHAR(1)	Code identifying the type of storage media: M = Magnetic O = Optical (IDS)
tabid	INTEGER	Code identifying the table
colno	SMALLINT	Column number within its table

A composite index on **tabid** and **colno** allows only unique values.

For information about the location and size of chunks of blobspaces, dbspaces, and sbspaces for TEXT, BYTE, BLOB, and CLOB columns, see the *IBM Informix: Administrator's Guide* and the *IBM Informix: Administrator's Reference*.

SYSCASTS (IDS)

The **syscasts** system catalog table describes the casts in the database. It contains one row for each built-in cast, each implicit cast, and each explicit cast that a user defines. The **syscasts** table has the following columns.

Column	Type	Explanation
owner	CHAR(32)	Owner of cast (user informix for built-in casts and <i>user</i> name for implicit and explicit casts)
argument_type	SMALLINT	Source data type on which the cast operates
argument_xid	INTEGER	Code for the source data type specified in the argument_type column
result_type	SMALLINT	Code for the data type returned by the cast
result_xid	INTEGER	Data type code of the data type named in the result_type column
routine_name	VARCHAR(128)	Function or procedure implementing the cast
routine_owner	CHAR(32)	Name of owner of the function or procedure specified in the routine_name column
class	CHAR(1)	Type of cast: E = Explicit cast I = Implicit cast S = Built-in cast

If **routine_name** and **routine_owner** have NULL values, this indicates that the cast is defined without a routine. This can occur if both of the data types

specified in the **argument_type** and **result_type** columns have the same length and alignment, and are passed by reference, or passed by value.

A composite index on columns **argument_type**, **argument_xid**, **result_type**, and **result_xid** allows only unique values. A composite index on columns **result_type** and **result_xid** allows duplicate values.

SYSCHECKS

The **syschecks** system catalog table describes each check constraint defined in the database. Because the **syschecks** table stores both the ASCII text and a binary encoded form of the check constraint, it contains multiple rows for each check constraint. The **syschecks** table has the following columns.

Column	Type	Explanation
constrid	INTEGER	Unique code identifying the constraint
type	CHAR(1)	Form in which the check constraint is stored: B = Binary encoded s = Select T = Text
seqno	SMALLINT	Line number of the check constraint
checktext	CHAR(32)	Text of the check constraint

The text in the **checktext** column associated with B type in the **type** column is in computer-readable format. To view the text associated with a particular check constraint, use the following query with the appropriate **constrid** code:

```
SELECT * FROM syschecks WHERE constrid=10 AND type='T'
```

Each check constraint described in the **syschecks** table also has its own row in the **sysconstraints** table.

A composite index on the **constrid**, **type**, and **seqno** columns allows only unique values.

SYSCHECKUDRDEP (IDS)

The **syscheckudrdep** system catalog table describes each check constraint that is referenced by a user-defined routine (UDR) in the database. The **syscheckudrdep** table has the following columns.

Column	Type	Explanation
udr_id	INTEGER	Unique code identifying the UDR
constraint_id	INTEGER	Unique code identifying the check constraint

Each check constraint described in the **syscheckudrdep** table also has its own row in the **sysconstraints** system catalog table, where the **constrid** column has the same value as the **constraint_id** column of **syscheckudrdep**.

A composite index on the **udr_id** and **constraint_id** columns requires that combinations of these values be unique.

SYSCOLATTRIBS (IDS)

The **syscolattribs** system catalog table describes the characteristics of smart large objects, namely CLOB and BLOB data types. It contains one row for each sbspace listed in the PUT clause of the CREATE TABLE statement.

Column	Type	Explanation	
tabid	INTEGER	Code uniquely identifying the table	
colno	SMALLINT	Number of the column that contains the smart large object	
extentsize	INTEGER	Pages in smart-large-object extent, expressed in kilobytes	
flags	INTEGER	Integer representation of the combination (by addition) of hexadecimal values of the following parameters:	
		LO_NOLOG (0x00000001 = 1)	The smart large object is not logged.
		LO_LOG (0x00000010 = 2)	Logging of smart large objects conforms to current log mode of the database.
		LO_KEEP_LASTACCESS_TIME (0x00000100 = 4)	A record is kept of the most recent access of this smart-large-object column by a user.
		LO_NOKEEP_LASTACCESS_TIME (0x00001000 = 8)	No record is kept of the most recent access of this smart-large-object column by a user.
		HI_INTEG (0x00010000= 16)	Data pages have headers and footers to detect incomplete writes and data corruption.
		MODERATE_INTEG (Not available at this time)	Data pages do not have headers and footers.
flags1	INTEGER	Reserved for future use	
sbspace	VARCHAR(128)	Name of the sbspace	

A composite index on the **tabid**, **colno**, and **sbspace** columns allows only unique combinations of these values.

SYSCOLAUTH

The **syscolauth** system catalog table describes each set of privileges granted on a column. It contains one row for each set of column privileges granted in the database. The **syscolauth** table has the following columns.

Column	Type	Explanation
grantor	VARCHAR(32)	Name of the grantor of privilege
grantee	VARCHAR(32)	Name of the grantee of privilege
tabid	INTEGER	Code uniquely identifying the table
colno	SMALLINT	Column number within its table
colauth	CHAR(3)	3-byte pattern that specifies column privileges: s or S = Select u or U = Update r or R = References

If the **colauth** privilege code is uppercase (for example, S for Select), a user who has this privilege can also grant it to others. If the **colauth** privilege code is lowercase (for example, s for Select), the user who has this privilege cannot grant it to others. A hyphen (-) indicates the absence of the privilege corresponding to that position within the **colauth** pattern.

A composite index on the **tabid**, **grantor**, **grantee**, and **colno** columns allows only unique values. A composite index on the **tabid** and **grantee** columns allows duplicate values.

SYSCOLDEPEND

The **syscoldepend** system catalog table tracks the table columns specified in check and NOT NULL constraints. Because a check constraint can involve more than one column in a table, the **syscoldepend** table can contain multiple rows for each check constraint; one row is created for each column involved in the constraint. The **syscoldepend** table has the following columns.

Column	Type	Explanation
constrid	INTEGER	Code uniquely identifying the constraint
tabid	INTEGER	Code uniquely identifying the table
colno	SMALLINT	Column number within the table

A composite index on the **constrid**, **tabid**, and **colno** columns allows only unique values. A composite index on the **tabid** and **colno** columns allows duplicate values.

See also the **syscheckudrdep** system catalog table in “SYSCHECKUDRDEP (IDS)” on page 1-18, which lists every check constraint that is referenced by a user-defined routine.

See also the **sysnewdepend** table in “SYSNEWDEPEND (XPS)” on page 1-40, which describes the column dependencies of generalized-key indexes.

See also the **sysreferences** table in “SYSREFERENCES” on page 1-49, which describes dependencies of referential constraints.

SYSCOLUMNS

The **syscolumns** system catalog table describes each column in the database. One row exists for each column that is defined in a table or view.

Column	Type	Explanation																										
colname	VARCHAR(128)	Column name																										
tabid	INTEGER	Identifying code of table containing the column																										
colno	SMALLINT	Column number The system sequentially assigns this (from left to right within each table).																										
coltype	SMALLINT	Code indicating the data type of the column: <table border="1"><tbody><tr><td>0 = CHAR</td><td>13 = VARCHAR</td></tr><tr><td>1 = SMALLINT</td><td>14 = INTERVAL</td></tr><tr><td>2 = INTEGER</td><td>15 = NCHAR</td></tr><tr><td>3 = FLOAT</td><td>16 = NVARCHAR</td></tr><tr><td>4 = SMALLFLOAT</td><td>17 = INT8</td></tr><tr><td>5 = DECIMAL</td><td>18 = SERIAL8 *</td></tr><tr><td>6 = SERIAL *</td><td>19 = SET</td></tr><tr><td>7 = DATE</td><td>20 = MULTISSET</td></tr><tr><td>8 = MONEY</td><td>21 = LIST</td></tr><tr><td>9 = NULL</td><td>22 = Unnamed ROW</td></tr><tr><td>10 = DATETIME</td><td>40 = Variable-length opaque type</td></tr><tr><td>11 = BYTE</td><td>4118 = Named ROW</td></tr><tr><td>12 = TEXT</td><td></td></tr></tbody></table>	0 = CHAR	13 = VARCHAR	1 = SMALLINT	14 = INTERVAL	2 = INTEGER	15 = NCHAR	3 = FLOAT	16 = NVARCHAR	4 = SMALLFLOAT	17 = INT8	5 = DECIMAL	18 = SERIAL8 *	6 = SERIAL *	19 = SET	7 = DATE	20 = MULTISSET	8 = MONEY	21 = LIST	9 = NULL	22 = Unnamed ROW	10 = DATETIME	40 = Variable-length opaque type	11 = BYTE	4118 = Named ROW	12 = TEXT	
0 = CHAR	13 = VARCHAR																											
1 = SMALLINT	14 = INTERVAL																											
2 = INTEGER	15 = NCHAR																											
3 = FLOAT	16 = NVARCHAR																											
4 = SMALLFLOAT	17 = INT8																											
5 = DECIMAL	18 = SERIAL8 *																											
6 = SERIAL *	19 = SET																											
7 = DATE	20 = MULTISSET																											
8 = MONEY	21 = LIST																											
9 = NULL	22 = Unnamed ROW																											
10 = DATETIME	40 = Variable-length opaque type																											
11 = BYTE	4118 = Named ROW																											
12 = TEXT																												
collength	SMALLINT	Column length (in bytes)																										
colmin	INTEGER	Minimum column length (in bytes)																										
colmax	INTEGER	Maximum column length (in bytes)																										
extended_id (IDS)	INTEGER	Data type code, from the sysxdtypes table, of the data type specified in the coltype column																										

* In DB–Access, an offset value of 256 is always added to these **coltype** codes because DB–Access sets SERIAL and SERIAL8 columns to NOT NULL.

Extended Parallel Server does not support opaque data types, nor the complex data types SET, MULTISSET, LIST, unnamed and named ROW.

A composite index on **tabid** and **colno** allows only unique values.

The **coltype** codes listed on the previous page can be incremented by bitmaps showing the following features of the column.

Bit Value	Significance When Bit Is Set
0x0100	NULL values are not allowed

0x0200	Value is from a host variable
0x0400	Float-to-decimal for networked database server
0x0800	DISTINCT data type
0x1000	Named ROW type
0x2000	DISTINCT type from LVARCHAR base type
0x4000	DISTINCT type from BOOLEAN base type
0x8000	Collection is processed on client system

For example, the **coltype** value 4118 for named row types is the decimal representation of the hexadecimal value 0x1016, which is the same as the hexadecimal **coltype** value for an unnamed row type (0 x 016), with the named-row-type bit set. (The file `$INFORMIXDIR/incl/esql/sqltypes.h` contains additional information about **syscolumns.coltype** codes.)

NOT NULL Constraints

Similarly, the **coltype** value is incremented by 256 if the column does not allow NULL values. To determine the data type for such columns, subtract 256 from the value and evaluate the remainder, based on the possible **coltype** values. For example, if the **coltype** value is 262, subtracting 256 leaves a remainder of 6, indicating that the column has a SERIAL data type.

Storing the Column Data Type

The database server stores the **coltype** value as bitmap, as listed in “SYSCOLUMNS” on page 1-22.

Opaque Data Types (IDS)

The BOOLEAN, BLOB, CLOB, and LVARCHAR data types are implemented by the database server as *built-in opaque* data types.

A built-in opaque data type is one for which the database server provides the type definition. Because these data types are built-in opaque types, they do not have a unique **coltype** value. Instead, they have one of the **coltype** values for opaque types: 41 (fixed-length opaque type), or 40 (varying-length opaque type). The different fixed-length opaque types are distinguished by the **extended_id** column in the **sysxtotypes** system catalog table.

The following are the **coltype** values for the built-in opaque data types.

Predefined Data Type	Value for coltype Column
BLOB	41
CLOB	41
BOOLEAN	41
LVARCHAR	40

Storing Column Length

The **collength** column value depends on the data type of the column.

Integer-Based Data Types

A **collength** value for a BIGINT, DATE, INTEGER, INT8, SERIAL, SERIAL8, or SMALLINT column is machine-independent. The database server uses the following lengths for these integer-based data types of the SQL language.

Integer-Based Data Types	Length (in Bytes)
SMALLINT	2
DATE	4
INTEGER	4
SERIAL	4
BIGINT, INT8	8 (XPS)
SERIAL8	8 (XPS)
INT8	10 (IDS)
SERIAL8	10 (IDS)

Varying-Length Character Data Types

For Dynamic Server columns of the LVARCHAR type, **collength** has the value of *max* from the data type declaration, or 2048 if no maximum was specified.

For VARCHAR, or NVARCHAR columns, the *max_size* and *min_space* values are encoded in the **collength** column using one of these formulas:

- If the **collength** value is positive:
$$\text{collength} = (\text{min_space} * 256) + \text{max_size}$$
- If the **collength** value is negative:
$$\text{collength} + 65536 = (\text{min_space} * 256) + \text{max_size}$$

Time Data Types

As noted previously, DATE columns have a value of 4 in the **collength** column.

For columns of type DATETIME or INTERVAL, **collength** is determined using the following formula:

$$(\text{length} * 256) + (\text{first_qualifier} * 16) + \text{last_qualifier}$$

The length is the physical length of the DATETIME or INTERVAL field, and *first_qualifier* and *last_qualifier* have values that the following table shows.

Field Qualifier	Value		Field Qualifier	Value
YEAR	0		FRACTION(1)	11
MONTH	2		FRACTION(2)	12
DAY	4		FRACTION(3)	13
HOUR	6		FRACTION(4)	14
MINUTE	8		FRACTION(5)	15
SECOND	10			

For example, if a DATETIME YEAR TO MINUTE column has a length of 12 (such as YYYY:DD:MO:HH:MI), a *first_qualifier* value of 0 (for YEAR), and a *last_qualifier* value of 8 (for MINUTE), then the **collength** value is 3080 (from $(256 * 12) + (0 * 16) + 8$).

Fixed-Point Data Types

The **collength** value for a MONEY or DECIMAL (*p*, *s*) column can be calculated using the following formula:

$$(precision * 256) + scale$$

Simple-Large-Object Data Types

If the data type of the column is BYTE or TEXT, **collength** holds the length of the descriptor.

Storing Maximum and Minimum Values

The **colmin** and **colmax** column values hold the second-smallest and second-largest data values in the column, respectively. For example, if the values in an indexed column are 1, 2, 3, 4, and 5, the **colmin** value is 2 and the **colmax** value is 4. Storing the second-smallest and second-largest data values lets the database server make assumptions about the range of values in a given column and, in turn, further optimize search strategies.

The **colmin** and **colmax** columns contain values only if the column is indexed and you have run the UPDATE STATISTICS statement. If you store BYTE or TEXT data in the tblspace, the **colmin** value is -1.

The **colmin** and **colmax** columns are valid only for data types that fit into four bytes: SMALLFLOAT, SMALLINT, INTEGER, and the first four bytes of CHAR. The values for all other noninteger column types are the initial four bytes of the maximum or minimum value, which are treated as integers.

It is better to use UPDATE STATISTICS MEDIUM than to depend on **colmin** and **colmax** values. UPDATE STATISTICS MEDIUM gives better information and is valid for all data types.

Dynamic Server does not calculate **colmin** and **colmax** values for user-defined data types. These columns, however, have values for user-defined data types if a user-defined secondary access method supplies them.

SYSCONSTRAINTS

The **sysconstraints** system catalog table lists the constraints placed on the columns in each database table. An entry is also placed in the **sysindexes** system catalog table (or **sysindices** view for Dynamic Server) for each unique, primary key, NOT NULL, or referential constraint that does not already have a corresponding entry in **sysindexes** or **sysindices**. Because indexes can be shared, more than one constraint can be associated with an index. The **sysconstraints** table has the following columns.

Column	Type	Explanation
constrid	SERIAL	Code uniquely identifying the constraint
constrname	VARCHAR(128)	Name of the constraint
owner	VARCHAR(32)	Name of the owner of the constraint
tabid	INTEGER	Code uniquely identifying the table
constrtype	CHAR(1)	Code identifying the constraint type: C = Check constraint N = Not NULL P = Primary key R = Referential T = Table U = Unique
idxname	VARCHAR(128)	Name of index corresponding to constraint
collation (IDS)	CHAR(32)	Collating order at the time when the constraint was created.

A composite index on the **constrname** and **owner** columns allows only unique values. An index on the **tabid** column allows duplicate values, and an index on the **constrid** column allows only unique values.

For check constraints (where **constrtype** = C), the **idxname** is always NULL. Additional information about each check constraint is contained in the **syschecks** and **syscoldepend** system catalog tables.

SYSDEFAULTS

The **sysdefaults** system catalog table lists the user-defined defaults that are placed on each column in the database. One row exists for each user-defined default value. The **sysdefaults** table has the following columns.

Column	Type	Explanation
tabid	INTEGER	Code uniquely identifying a table
colno	SMALLINT	Code uniquely identifying a column
type	CHAR(1)	Code identifying the type of default value: C = Current L = Literal value N = NULL S = Dbservername or Sitename T = Today U = User
default	CHAR(256)	If sysdefaults.type = L, a literal default value
class (IDS)	CHAR(1)	Code identifying what kind of column: T = table t = ROW type

If no default is specified explicitly in the CREATE TABLE or the ALTER TABLE statement, then no entry exists for that column in the **sysdefaults** table.

If you specify a literal for the default value, it is stored in the **default** column as ASCII text. If the literal value is not of one of the data types listed in the next paragraph, the **default** column consists of two parts. The first part is the 6-bit representation of the binary value of the default value structure. The second part is the default value in ASCII text. A blank space separates the two parts.

If the data type of the column is not CHAR, NCHAR, NVARCHAR, or VARCHAR, or (for Dynamic Server) BOOLEAN or LVARCHAR, a binary representation of the default value is encoded in the **default** column.

A composite index on the **tabid**, **colno**, and **class** columns allows only unique values. (For Extended Parallel Server, this index omits the **class** column.)

SYSDEPEND

The **sysdepend** system catalog table describes how each view or table depends on other views or tables. One row exists in this table for each dependency, so a view based on three tables has three rows. The **sysdepend** table has the following columns.

Column	Type	Explanation
btabid	INTEGER	Code uniquely identifying the base table or view
btype	CHAR(1)	Base object type: T = Table V = View
dtabid	INTEGER	Code uniquely identifying a dependent table or view
dtype	CHAR(1)	Code for the type of dependent object; currently, only view (V = View) is implemented

The **btabid** and **dtabid** columns are indexed and allow duplicate values.

SYSDIRECTIVES (IDS)

The **sysdirectives** table stores external optimizer directives that can be applied to queries. Whether queries in client applications can use these optimizer directives depends on the setting of the **IFX_EXTDIRECTIVES** environment variable on the client system, as described in Chapter 3, and on the **EXT_DIRECTIVES** setting in the configuration file of the database server.

The **sysdirectives** table has the following columns.

Column	Type	Explanation
id	SERIAL	Unique code identifying the optimizer directive
query	TEXT	Text of the query as it appears in the application
directives	TEXT	Text of the optimizer directive, without comments
active	SMALLINT	Integer code that identifies whether this entry is active (= 1) or test only (= 2)
hash_code	SMALLINT	For internal use only

NULL values are not valid in the **query** column. There is a unique index on the **id** column.

SYSDISTRIB

The **sysdistrib** system catalog table stores data-distribution information for the database server to use. Data distributions provide detailed table and column information to the optimizer to improve the choice of execution paths of SELECT statements. The **sysdistrib** table has the following columns.

Column	Type		Explanation
tabid	INTEGER		Code uniquely identifying the table where data values were gathered
colno	SMALLINT		Column number in the source table
seqno	INTEGER		Ordinal number for multiple entries
constructed	DATE		Date when the data distribution was created
mode	CHAR(1)		Optimization level: M = Medium H = High
resolution	SMALLFLOAT		Specified in the UPDATE STATISTICS statement
confidence	SMALLFLOAT		Specified in the UPDATE STATISTICS statement
encdat	STAT CHAR(256)	IDS XPS	Statistics information
type (IDS)	CHAR(1)		Type of statistics: A = encdat has ASCII-encoded histogram in fixed-length character field S = encdat has user-defined statistics

Information is stored in the **sysdistrib** table when an UPDATE STATISTICS statement with mode MEDIUM or HIGH is executed for a table. (UPDATE STATISTICS LOW does not insert a value into the **mode** column.)

Only user **informix** can select the **encdat** column.

Each row in the **sysdistrib** system catalog table is keyed by the **tabid** and **colno** for which the statistics are collected.

For built-in data type columns, the **type** field is set to A. The **encdat** column stores an ASCII-encoded histogram that is broken down into multiple rows, each of which contains 256 bytes.

In Dynamic Server, for columns of user-defined data types, the **type** field is set to S. The **encdat** column stores the statistics collected by the **statcollect** user-defined routine in multirepresentational form. Only one row is stored for each **tabid** and **colno** pair. A composite index on the **tabid**, **colno**, and **seqno** columns requires unique combinations of values.

SYSDOMAINS (IDS)

The **sysdomains** view is not used. It displays columns of other system catalog tables. It has the following columns.

Column	Type	Explanation
id	SERIAL	Unique code identifying the domain
owner	CHAR(32)	Name of the owner of the domain
name	VARCHAR(128)	Name of the domain
type	SMALLINT	Code identifying the type of domain

There is no index on this view.

SYSEERRORS (IDS)

The **syserrors** system catalog table stores information about error, warning, and informational messages returned by DataBlade modules and user-defined routines using the **mi_db_error_raise()** DataBlade API function.

The **syserrors** table has the following columns.

Column	Type	Explanation
sqlstate	CHAR(5)	SQLSTATE value associated with the error For more information about SQLSTATE values and their meanings, see the GET DIAGNOSTICS statement in the <i>IBM Informix: Guide to SQL Syntax</i> .
locale	CHAR(36)	The locale with which this version of the message is associated (for example, en_us.8859-1)
level	SMALLINT	Reserved for future use
seqno	SMALLINT	Reserved for future use
message	VARCHAR(255)	Message text

To create a new message, insert a row directly into the **syserrors** table. By default, all users can view this table, but only users with the DBA privilege can modify it.

A composite index on the **sqlstate**, **locale**, **level**, and **seqno** columns allows only unique values.

SYSEXTCOLS (XPS)

The **sysextcols** system catalog table contains a row that describes each of the internal columns in external table **tabid** of format type (**fmttype**) FIXED. The **sysextcols** table has the following columns.

Column	Type	Explanation
tabid	INTEGER	Unique identifying code of a table
colno	SMALLINT	Code identifying the column
exttype	SMALLINT	Code identifying an external column type
extstart	SMALLINT	Starting position of column in the external data file
extlength	SMALLINT	External column length (in bytes)
nullstr	CHAR(256)	Represents NULL in external data
picture	CHAR(256)	Reserved for future use
decimal	SMALLINT	Precision for external decimals
extstype	VARCHAR(128)	External type name

No entries are stored in **sysextcols** for DELIMITED or Informix format external files.

You can use the DBSCHEMA utility to write out the description of the external tables. To query these system catalog tables about an external table, use the **tabid** as stored in **systables** with **tabtype** = 'E'.

An index on the **tabid** column allows duplicate values.

SYSEXTDFILES (XPS)

For each external table, at least one row exists in the **sysextfiles** system catalog table, which has the following columns.

Column	Type	Explanation
tabid	INTEGER	Unique identifying code of an external table
dfentry	CHAR(152)	Data file entry

You can use DBSCHEMA to write out the description of the external tables. To query these system catalog tables about an external table, use the **tabid** as stored in **systables** with **tabtype** = 'E'.

An index on the **tabid** column allows duplicate values.

SYSEXTERNAL (XPS)

For each external table, a single row exists in the **sysexternal** system catalog table. The **tabid** column associates the external table in this system catalog table with an entry in **systables**.

Column	Type	Explanation
tabid	INTEGER	Unique identifying code of an external table
fmttype	CHAR(1)	Type of format: D = (delimited) F = (fixed) I = (Informix)
codeset	VARCHAR(128)	ASCII, EBCDIC
recdelim	CHAR(4)	The record delimiter
flddelim	CHAR(4)	The field delimiter
datefmt	CHAR(8)	Reserved for future use
moneyfmt	CHAR(20)	Reserved for future use
maxerrors	INTEGER	Number of errors to allow per coserver
rejectfile	CHAR(128)	Name of reject file
flags	INTEGER	Optional load flags
ndfiles	INTEGER	Number of data files in sysextdfiles

You can use DBSCHEMA to write out the description of the external tables. To query these system catalog tables about an external table, use the **tabid** as stored in **systables** with **tabtype = 'E'**.

An index on the **tabid** column allows only unique values.

SYSFRAGAUTH (IDS)

The **sysfragauth** system catalog table stores information about the privileges that are granted on table fragments. This table has the following columns.

Column	Type	Explanation
grantor	CHAR(32)	Name of the grantor of privilege
grantee	CHAR(32)	Name of the grantee of privilege
tabid	INTEGER	Identifying code of the fragmented table
fragment	VARCHAR(128)	Name of dbspace where fragment is stored
fragauth	CHAR(6)	A 6-byte pattern specifying fragment privileges (including 3 bytes reserved for future use): u or U = Update i or I = Insert d or D = Delete

In the **fragauth** column, an uppercase code (such as U for Update) means that the grantee can grant the privilege to other users; a lowercase (for example, u for Update) means the user cannot grant the privilege to others. Hyphen (-) indicates the absence of the privilege for that position within the pattern.

A composite index on the **tabid**, **grantor**, **grantee**, and **fragment** columns allows only unique values. A composite index on the **tabid** and **grantee** columns allows duplicate values.

The following example displays the fragment-level privileges for one base table, as they appear in the **sysfragauth** table. In this example, the grantee **ted** can grant the Update, Delete, and Insert privileges to other users.

grantor	grantee	tabid	fragment	fragauth
dba	dick	101	dbsp1	-ui---
dba	jane	101	dbsp3	--i---
dba	mary	101	dbsp4	--id--
dba	ted	101	dbsp2	-UID--

SYSFRAGMENTS

The **sysfragments** system catalog table stores fragmentation information for tables and indexes. One row exists for each table or index fragment.

The **sysfragments** table has the following columns.

Column	Type	Explanation
fragtype	CHAR(1)	Code indicating the type of fragmented object: I = Original index fragment i = Duplicated index fragment (XPS) T = Original table fragment t = Duplicated table fragment (XPS) B = TEXT or BYTE data (XPS) i = Index fragments of a duplicated table (XPS) d = Data fragments of a duplicated table (XPS)
tabid	INTEGER	Unique identifying code of table
indexname	VARCHAR(128)	Name of index
colno	INTEGER	Identifying code of TEXT or BYTE column (IDS) Identifying code of replica identifier (XPS)
partn	INTEGER	Identifying code of physical location
strategy	CHAR(1)	Code for type of fragment distribution strategy: R = Round-robin fragmentation strategy E = Expression-based fragmentation strategy I = IN DBSPACE clause specifies a specific location as part of fragmentation strategy T = Table-based fragmentation strategy H = Table is a subtable within a table hierarchy (IDS) Hash-based fragmentation strategy (XPS)
location	CHAR(1)	Reserved for future use; shows L for local
servername	VARCHAR(128)	Reserved for future use
evalpos	INTEGER	Position of fragment in the fragmentation list
exprtext	TEXT	Expression for fragmentation strategy (IDS) Contains names of the columns that are hashed and composite information for hybrid fragmentation strategies; shows hashed columns followed by the fragmentation expression of the dbslice. (XPS)
exprbin	BYTE	Binary version of expression
exprarr	BYTE	Range-partitioning data to optimize expression in range-expression fragmentation strategy

Column	Type	Explanation
flags	INTEGER	Used internally (IDS) Bitmap indicating a hybrid fragmentation strategy (value = 0x00000010) (XPS) Also, an additional flag (value = 0x00000020) is set on the first fragment of a globally detached index. (XPS)
dbspace	VARCHAR(128)	Name of dbspace for fragment
levels	SMALLINT	Number of B+ tree index levels
npused	INTEGER	For table-fragmentation strategy, npused is the number of data pages; for index-fragmentation strategy, npused is the number of leaf pages.
nrows	INTEGER	For tables, nrows represents the number of rows in the fragment; for indexes, nrows represents the number of unique keys.
clust	INTEGER	Degree of index clustering; smaller numbers correspond to greater clustering.
hybdpos	INTEGER	Contains the relative position of the hybrid fragment within a dbslice or list of dbspaces associated with a particular expression. The hybrid fragmentation strategy (and the set of fragments against which the hybrid strategy is applied) determine the relative position. The first fragment has a hybdpos value of zero (0). (XPS)
partname	VARCHAR(128)	Name of partition within dbspace for fragment (IDS)

In certain situations, you can duplicate selected tables across coservers to improve performance. If you have a duplicate copy of a small table on each coserver, then the database server can execute some small queries (queries that do not need rows from a table fragment on any other coserver and that do not require more than 128 kilobytes of memory per operator) as serial plans instead of as parallel plans that the Resource Grant Manager (RGM) manages. This performance feature applies only to OLTP-type transactions.

The following query returns the owner and name for each of the duplicated tables in the current database:

```
SELECT DISTINCT st.owner, st.tabname
FROM systables st, sysfragments sf
WHERE st.tabid = sf.tabid AND sf.fragtype = 't'
```

For more information about duplicating tables, refer to the description of the CREATE DUPLICATE statement in *IBM Informix: Guide to SQL Syntax*.

The **strategy** type T is used for attached indexes. (This is a fragmented index whose fragmentation is the same as the table fragmentation.)

In Dynamic Server, a composite index on the **fragtype**, **tabid**, **indexname**, and **evalpos** columns allows duplicate values.

In Extended Parallel Server, a composite index on the **fragtype**, **tabid**, **indexname**, **evalpos**, and **hybdpos** columns allows duplicate values.

SYSINDEXES

The **sysindexes** table is a view on the **sysindices** table. It contains one row for each index in the database. The **sysindexes** table has the following columns.

Column	Type	Explanation
idxname	VARCHAR(128)	Index name
owner	VARCHAR(32)	Owner of index (user informix for system catalog tables and <i>username</i> for database tables) (IDS)
tabid	INTEGER	Unique identifying code of table
idxtype	CHAR(1)	Index type: U = Unique D = Duplicates allowed G = Nonbitmap generalized-key index (XPS) g = Bitmap generalized-key index u = unique, bitmap d = nonunique, bitmap
clustered	CHAR(1)	Clustered or nonclustered index (C = Clustered)
part1	SMALLINT	Column number (colno) of a single index or the 1st component of a composite index
part2	SMALLINT	2nd component of a composite index
part3	SMALLINT	3rd component of a composite index
part4	SMALLINT	4th component of a composite index
part5	SMALLINT	5th component of a composite index
part6	SMALLINT	6th component of a composite index
part7	SMALLINT	7th component of a composite index
part8	SMALLINT	8th component of a composite index
part9	SMALLINT	9th component of a composite index
part10	SMALLINT	10th component of a composite index

Column	Type	Explanation
part11	SMALLINT	11th component of a composite index
part12	SMALLINT	12th component of a composite index
part13	SMALLINT	13th component of a composite index
part14	SMALLINT	14th component of a composite index
part15	SMALLINT	15th component of a composite index
part16	SMALLINT	16th component of a composite index
levels	SMALLINT	Number of B-tree levels
leaves	INTEGER	Number of leaves
nunique	INTEGER	Number of unique keys in the first column
clust	INTEGER	Degree of clustering; smaller numbers correspond to greater clustering
idxflags	INTEGER	Bitmap storing the current locking mode of the index Normal = 0x00000001 (XPS) Coarse = 0x00000002 (XPS)

As with most system catalog tables, changes that affect existing indexes are reflected in this table only after you run the UPDATE STATISTICS statement.

Each **part1** through **part16** column in this table holds the column number (**colno**) of one of the 16 possible parts of a composite index. If the component is ordered in descending order, the **colno** is entered as a negative value. The columns are filled in for B-tree indexes that do not use user-defined data types or functional indexes. For generic B-trees and all other access methods, the **part1** through **part16** columns all contain zeros.

The **clust** column is blank until the UPDATE STATISTICS statement is run on the table. The maximum value is the number of rows in the table, and the minimum value is the number of data pages in the table.

In Extended Parallel Server, the **tabid** column is indexed and allows duplicate values. A composite index on the **idxname**, **owner**, and **tabid** columns allows only unique values.

SYSINDICES (IDS)

The **sysindices** system catalog table describes the indexes in the database. It contains one row for each index that is defined in the database.

Column	Type	Explanation
idxname	VARCHAR(128)	Name of index
owner	VARCHAR(32)	Name of owner of index (user informix for system catalog tables and <i>username</i> for database tables)
tabid	INTEGER	Unique identifying code of table
idxtype	CHAR(1)	Index type: U = Unique D = Duplicates allowed
clustered	CHAR(1)	Clustered or nonclustered index (C = Clustered)
levels	SMALLINT	Number of tree levels
leaves	INTEGER	Number of leaves
nunique	INTEGER	Number of unique keys in the first column
clust	INTEGER	Degree of clustering; smaller numbers correspond to greater clustering. The maximum value is the number of rows in the table, and the minimum value is the number of data pages in the table. This column is blank until UPDATE STATISTICS is run on the table.
nrows	FLOAT	Estimated number of rows in the table (zero until UPDATE STATISTICS is run on the table).
indexkeys	INDEX-KEYARRAY	Column can have up to three fields, in the format: procid , (<i>col1</i> , ... , <i>coln</i>), opclassid where $1 < n < 341$
amid	INTEGER	Unique identifying code of the access method that implements this index. (Value = am_id for that access method in the sysams table.)
amparam	LVARCHAR	List of parameters used to customize the behavior of this access method.
collation	CHAR(32)	Collating order at the time of index creation.

Tip: This system catalog table is changed from Version 7.2 of Informix database servers. The earlier schema of this system catalog table is still available as a view and can be accessed under its original name: **sysindexes**.

Changes that affect existing indexes are reflected in this system catalog table only after you run the UPDATE STATISTICS statement.

The fields within the **indexkeys** columns have the following significance:

- The **procid** (as in **sysprocedures**) appears only for a functional index on return values of a function defined on columns of the table.
- The list of columns (*col1, ... , coln*) in the second field identifies the columns on which the index is defined. The maximum is language-dependent: up to 341 for an SPL or Java UDR; up to 102 for a C UDR.
- The **opclassid** identifies the secondary access method that the database server used to build and to search the index. This is the same as the **sysopclasses.opclassid** value for the access method.

The **tabid** column is indexed and allows duplicate values. A composite index on the **idxname**, **owner**, and **tabid** columns allows only unique values.

SYsinHERITS (IDS)

The **sysinherits** system catalog table stores information about table and named ROW type inheritance. Every supertype, subtype, supertable, and subtable in the database has a corresponding row in the **sysinherits** table.

Column	Type	Explanation
child	INTEGER	Identifying code of the subtable or subtype
parent	INTEGER	Identifying code of the supertable or supertype
class	CHAR(1)	Inheritance class: t = named ROW type T = table

The **child** and **parent** values are from **sysxdtypes.extended_id** for named ROW types, or from **sysables.tabid** for tables. Simple indexes on the **child** and **parent** columns allow duplicate values.

SYSLANGAUTH (IDS)

The **syslangauth** system catalog table contains the authorization information on computer languages that are used to write user-defined routines (UDRs).

Column	Type	Explanation
grantor	VARCHAR(32)	Name of the grantor of the language authorization
grantee	VARCHAR(32)	Name of the grantee of the language authorization
langid	INTEGER	Identifying code of language in sysroutinelangs table
langauth	CHAR(1)	The language authorization: u = Usage privilege granted U = Usage privilege granted WITH GRANT OPTION

A composite index on the **langid**, **grantor**, and **grantee** columns allows only unique values. A composite index on the **langid** and **grantee** columns allows duplicate values.

SYSLOGMAP (IDS)

The **syslogmap** system catalog table contains fragmentation information.

Column	Type	Explanation
tabloc	INTEGER	Code for the location of an external table
tabid	INTEGER	Unique identifying code of the table
fragid	INTEGER	Identifying code of the fragment
flags	INTEGER	Bitmap of modifiers from declaration of fragment

A simple index on the **tabloc** column and a composite index on the **tabid** and **fragid** columns do not allow duplicate values.

SYSNEWDEPEND (XPS)

The **sysnewdepend** system catalog table contains information about generalized-key indexes that are not available in the **sysindexes** table. The dependencies between a generalized-key index and the tables in the FROM clause of the CREATE INDEX statement are stored in the **sysnewdepend** table, which has the following columns.

Column	Type	Explanation
scrid1	VARCHAR(128)	Name of the generalized-key index
scrid2	INTEGER	Unique identifying code (= tabid) of the indexed table
type	INTEGER	Code for the type of generalized-key index
destid1	INTEGER	The systables.tabid value for the table on which the generalized-key index depends
destid2	INTEGER	The column number within the destid1 table

A composite index on the **scrid1**, **scrid2**, and **type** columns allows duplicate values. Another composite index on the **destid1**, **destid2**, and **type** columns also allows duplicate values.

SYSOBJSTATE (IDS)

The **sysobjstate** system catalog table stores information about the state (object mode) of database objects. The types of database objects that are listed in this table are indexes, triggers, and constraints.

Every index, trigger, and constraint in the database has a corresponding row in the **sysobjstate** table if a user creates the object. Indexes that the database server creates on the system catalog tables are not listed in the **sysobjstate** table because their object mode cannot be changed.

The **sysobjstate** table has the following columns.

Column	Type	Explanation
objtype	CHAR(1)	Code for the type of database object: C = Constraint I = Index T = Trigger
owner	VARCHAR(32)	Name of the owner of the database object
name	VARCHAR(128)	Name of the database object
tabid	INTEGER	Identifying code of table on which the object is defined
state	CHAR(1)	The current state (object mode) of the database object. This value can be one of the following codes: D = Disabled E = Enabled F = Filtering with no integrity-violation errors G = Filtering with integrity-violation errors

A composite index on the **objtype**, **name**, **owner**, and **tabid** columns allows only unique combinations of values. A simple index on the **tabid** column allows duplicate values.

SYSOPCLASSES (IDS)

The **sysopclasses** system catalog table contains information about operator classes associated with secondary access methods. It contains one row for each operator class that has been defined in the database. The **sysopclasses** table has the following columns.

Column	Type	Explanation
opclassname	VARCHAR(128)	Name of the operator class
owner	VARCHAR(32)	Name of the owner of the operator class
amid	INTEGER	Identifying code of the secondary access method associated with this operator class
opclassid	SERIAL	Identifying code of the operator class
ops	LVARCHAR	List of names of the operators that belong to this operator class
support	LVARCHAR	List of names of support functions defined for this operator class

The **opclassid** value corresponds to the **sysams.am_defopclass** value that specifies the default operator class for the secondary access method that the **amid** column specifies.

The **sysopclasses** table has a composite index on the **opclassname** and **owner** columns and an index on **opclassid** column. Both indexes allow only unique values.

SYSOPCLSTR

The **sysopclstr** system catalog table defines each optical cluster in the database. It contains one row for each optical cluster. The **sysopclstr** table has the following columns.

Column	Type	Explanation
owner	VARCHAR(32)	Name of the owner of the optical cluster
clstrname	VARCHAR(128)	Name of the optical cluster
clstrsize	INTEGER	Size of the optical cluster
tabid	INTEGER	Unique identifying code for the table
blobcol1	SMALLINT	BYTE or TEXT column number 1
blobcol2	SMALLINT	BYTE or TEXT column number 2
blobcol3	SMALLINT	BYTE or TEXT column number 3
blobcol4	SMALLINT	BYTE or TEXT column number 4
blobcol5	SMALLINT	BYTE or TEXT column number 5
blobcol6	SMALLINT	BYTE or TEXT column number 6
blobcol7	SMALLINT	BYTE or TEXT column number 7
blobcol8	SMALLINT	BYTE or TEXT column number 8
blobcol9	SMALLINT	BYTE or TEXT column number 9

Column	Type	Explanation
blobcol10	SMALLINT	BYTE or TEXT column number 10
blobcol11	SMALLINT	BYTE or TEXT column number 11
blobcol12	SMALLINT	BYTE or TEXT column number 12
blobcol13	SMALLINT	BYTE or TEXT column number 13
blobcol14	SMALLINT	BYTE or TEXT column number 14
blobcol15	SMALLINT	BYTE or TEXT column number 15
blobcol16	SMALLINT	BYTE or TEXT column number 16
clstrkey1	SMALLINT	Cluster key number 1
clstrkey2	SMALLINT	Cluster key number 2
clstrkey3	SMALLINT	Cluster key number 3
clstrkey4	SMALLINT	Cluster key number 4
clstrkey5	SMALLINT	Cluster key number 5
clstrkey6	SMALLINT	Cluster key number 6
clstrkey7	SMALLINT	Cluster key number 7
clstrkey8	SMALLINT	Cluster key number 8
clstrkey9	SMALLINT	Cluster key number 9
clstrkey10	SMALLINT	Cluster key number 10
clstrkey11	SMALLINT	Cluster key number 11
clstrkey12	SMALLINT	Cluster key number 12
clstrkey13	SMALLINT	Cluster key number 13
clstrkey14	SMALLINT	Cluster key number 14
clstrkey15	SMALLINT	Cluster key number 15
clstrkey16	SMALLINT	Cluster key number 16

The contents of this table are sensitive to CREATE OPTICAL CLUSTER, ALTER OPTICAL CLUSTER, and DROP OPTICAL CLUSTER statements that have been executed on databases that support optical cluster subsystems. Changes that affect existing optical clusters are reflected in this table only after you run the UPDATE STATISTICS statement.

A composite index on the **clstrname** and **owner** columns allows only unique values. A simple index on the **tabid** column allows duplicate values.

SYSPROCAUTH

The **sysprocauth** system catalog table describes the privileges granted on a procedure or function. It contains one row for each set of privileges that is granted. The **sysprocauth** table has the following columns.

Column	Type	Explanation
grantor	VARCHAR(32)	Name of grantor of privileges to access the routine
grantee	VARCHAR(32)	Name of grantee of privileges to access the routine
procid	INTEGER	Unique identifying code of the routine
procauth	CHAR(1)	Type of privilege granted on the routine: e = Execute privilege on routine E = Execute privilege WITH GRANT OPTION

A composite index on the **procid**, **grantor**, and **grantee** columns allows only unique values. A composite index on the **procid** and **grantee** columns allows duplicate values.

SYSPROCBODY

The **sysprocbody** system catalog table describes the compiled version of each procedure or function in the database. Because the **sysprocbody** table stores the text of the routine, each routine can have multiple rows. The **sysprocbody** table has the following columns.

Column	Type	Explanation
procid	INTEGER	Unique identifying code for the routine
datakey	CHAR(1)	Type of information in the data column: D = User document text E = Creation time information L = Literal value (that is, literal number or quoted string) P = Interpreter instruction code (p-code) R = Return value type list S = Routine symbol table T = Actual routine source
seqno	INTEGER	Line number within the routine
data	CHAR(256)	Actual text of the routine

The **data** column contains actual data, which can be in one of these formats:

- Encoded return values list
- Encoded symbol table

- Literal data
- P-code for the routine
- Compiled code for the routine
- Text of the routine and its documentation

A composite index on the **procid**, **datakey**, and **seqno** columns allows only unique values.

SYSPROCEDURES

The **sysprocedures** system catalog table lists the characteristics for each function and procedure in the database. It contains one row for each routine.

Each function in **sysprocedures** has a unique value, **procid**, called a *routine identifier*. Throughout the system catalog, a function is identified by its routine identifier, not by its name.

For Extended Parallel Server, **sysprocedures** has the following columns.

Column	Type	Explanation
procname	VARCHAR(128)	Name of routine
owner	VARCHAR(32)	Name of owner
procid	SERIAL	Unique identifying code for the routine
mode	CHAR(1)	Mode type: D or d = DBA O or o = Owner P or p = Protected R or r = Restricted
retsize	INTEGER	Compiled size (in bytes) of values
symsize	INTEGER	Compiled size (in bytes) of symbol table
datasize	INTEGER	Compiled size (in bytes) of constant data
codesize	INTEGER	Compiled size (in bytes) of routine instruction code
numargs	INTEGER	Number of arguments to routine

A composite index on **procname** and **owner** requires unique values.

For Dynamic Server, **sysprocedures** has the following columns.

Column	Type	Explanation
procname	VARCHAR(128)	Name of routine
owner	VARCHAR(32)	Name of owner

Column	Type	Explanation
procid	SERIAL	Unique identifying code for the routine
mode	CHAR(1)	Mode type: D or d = DBA O or o = Owner P or p = Protected R or r = Restricted
retsize	INTEGER	Compiled size (in bytes) of returned values
symsize	INTEGER	Compiled size (in bytes) of symbol table
datasize	INTEGER	Compiled size (in bytes) of constant data
codesize	INTEGER	Compiled size (in bytes) of routine code
numargs	INTEGER	Number of arguments to routine
isproc	CHAR(1)	Whether routine is a procedure or a function: t = procedure f = function
specificname	VARCHAR(128)	Specific name for the routine
externalname	VARCHAR(255)	Location of the external routine. This item is language-specific in content and format.
paramstyle	CHAR(1)	Parameter style: I = Informix
langid	INTEGER	Language code (in sysroutinelangs table)
paramtypes	RTNPARAMTYPES	Information describing returned parameters
variant	BOOLEAN	Whether the routine is VARIANT or not: t = is VARIANT f = is not VARIANT
client	BOOLEAN	Reserved for future use
handlesnulls	BOOLEAN	NULL handling indicator: t = handles NULLs f = does not handle NULLs
percallcost	INTEGER	Amount of CPU per call Integer cost to execute UDR: cost/call - 0 -(2 ³¹ -1)
commutator	VARCHAR(128)	Name of commutator function
negator	VARCHAR(128)	Name of negator function
selfunc	VARCHAR(128)	Name of function to estimate selectivity of UDR
internal	BOOLEAN	Whether the routine can be called from SQL: t = routine is internal, not callable from SQL f = routine is external, callable from SQL
class	CHAR(18)	CPU class in which routine should be executed

Column	Type	Explanation
stack	INTEGER	Stack size in bytes required per invocation
parallelizable	BOOLEAN	Parallelization indicator for UDR: t = parallelizable f = not parallelizable
costfunc	VARCHAR(128)	Name of cost function for UDR
selconst	SMALLFLOAT	Selectivity constant for UDR

In the **mode** column, the R mode is a special case of the O mode. A routine is in restricted (R) mode if it was created with a specified owner who is different from the routine creator. If routine statements involving a remote database are executed, the database server uses the permissions of the user who executes the routine instead of the permissions of the routine owner. In all other scenarios, R-mode routines behave the same as O-mode routines.

You cannot use the DROP FUNCTION, DROP ROUTINE, or DROP PROCEDURE statements to delete a protected routine. Protected routines are indicated by lowercase in the **mode** column. In earlier versions, protected SPL routines (which cannot be deleted) were indicated by a p. Starting with Version 9.0, protected SPL routines are treated as DBA routines and cannot be Owner routines. Thus D and O indicate DBA and Owner routines, and d and o indicate protected DBA and protected Owner routines.

A database server can create protected SPL routines for internal use. These protected SPL routines have p in the **mode** column. You cannot modify, drop, nor display protected SPL routines.

Important: After you issue the SET SESSION AUTHORIZATION statement, the database server assigns a restricted mode to all owner routines that you created while using the new identity.

The database server can create protected routines for internal use. The **sysprocedures** table identifies these protected routines with the letter P or p in the **mode** column. You cannot modify or drop protected routines, nor can you display them through DBSCHEMA.

A unique index is on the **procid** column. A composite index on the **procname**, **isproc**, **numargs**, and **owner** columns allows duplicate values, as does a composite index on the **specificname** and **owner** columns.

SYSPROCPLAN

The **sysprocplan** system catalog table describes the query-execution plans and dependency lists for data-manipulation statements within each routine. Because different parts of a routine plan can be created on different dates, this table can contain multiple rows for each routine.

Column	Type	Explanation
procid	INTEGER	Identifying code for the routine
planid	INTEGER	Identifying code for the plan
datakey	CHAR(1)	Type of information stored in data column: D = Dependency list I = Information record Q = Execution plan
seqno	INTEGER	Line number within the plan
created	DATE	Date when plan was created
datasize	INTEGER	Size (in bytes) of the list or plan
data	CHAR(256)	Encoded (compiled) list or plan (IDS) Text of the SPL routine (XPS)
collation	CHAR(32)	Collating order at the time when routine was created

Before a routine is run, its dependency list in the **data** column is examined. If the major version number of a table accessed by the plan has changed, or if any object that the routine uses has been modified since the plan was optimized (for example, if an index has been dropped), then the plan is optimized again. When **datakey** is I, the **data** column stores information about UPDATE STATISTICS and PDQPRIORITY.

It is possible to delete all the plans for a given routine by using the DELETE statement on **sysprocplan**. When the routine is subsequently executed, new plans are automatically generated and recorded in **sysprocplan**. The UPDATE STATISTICS FOR PROCEDURE statement also updates this table.

A composite index on the **procid**, **planid**, **datakey**, and **seqno** columns allows only unique values.

SYSREFERENCES

The **sysreferences** system catalog table lists all referential constraints on columns. It contains a row for each referential constraint in the database.

Column	Type	Explanation
constrid	INTEGER	Code uniquely identifying the constraint
primary	INTEGER	Identifying code of the corresponding primary key
ptabid	INTEGER	Identifying code of the table that is the primary key
updrule	CHAR(1)	Reserved for future use; displays an R
delrule	CHAR(1)	Whether constraint uses cascading delete or restrict rule: C = Cascading delete R = Restrict (default)
matchtype	CHAR(1)	Reserved for future use; displays an N
pendant	CHAR(1)	Reserved for future use; displays an N

The **constrid** column is indexed and allows only unique values. The **primary** column is indexed and allows duplicate values.

SYSREPOSITORY (XPS)

The **sysrepository** system catalog table contains data about generalized-key indexes that the **sysindexes** system catalog table does not provide.

Column	Type	Explanation
id1	VARCHAR(128)	Index from the generalized-key (GK) index
id2	INTEGER	Tabid of table with the generalized-key index
type	INTEGER	Integer code for type of object In this release, the only value that can appear is 1, indicating a GK index type.
seqid	SERIAL	Reserved for future use (This value is not related to syssequences.seqid .)
desc	TEXT	The CREATE INDEX statement of a GK index
bin	BYTE	Internal representation of the generalized-key index

The contents of the **sysrepository** table are useful when a generalized-key index has to be rebuilt during a recovery or if a user wants to see the CREATE statement for a specific generalized-key index.

The **desc** column contains the CREATE statement for each generalized-key index in the database.

An index on the **seqid** column allows duplicate values. A composite index on the **id1**, **id2**, and **type** columns requires unique combinations of values.

SYSROLEAUTH

The **sysroleauth** system catalog table describes the roles that are granted to users. It contains one row for each role that is granted to a user in the database. The **sysroleauth** table has the following columns.

Column	Type	Explanation
rolename	VARCHAR(32)	Name of the role
grantee	VARCHAR(32)	Name of the grantee of the role
is_grantable	CHAR(1)	Specifies whether the role is grantable: Y = Grantable N = Not grantable

The **is_grantable** column indicates whether the role was granted with the WITH GRANT OPTION of the GRANT statement.

A composite index on the **rolename** and **grantee** columns allows only unique values.

SYSROUTINELANGS (IDS)

The **sysroutinelangs** system catalog table lists the supported programming languages for user-defined routines (UDRs). It has these columns.

Column	Type	Explanation
langid	SERIAL	Code uniquely identifying a supported language
langname	CHAR(30)	Name of the language, such as C or SPL
langinitfunc	VARCHAR(128)	Name of initialization function for the language
langpath	CHAR(255)	Directory path for the UDR language
langclass	CHAR(18)	Name of the class of the UDR language

An index on the **langname** column allows duplicate values.

SYSSEQUENCES (IDS)

The **syssequences** system catalog table lists the sequence objects that exist in the database. The **syssequences** table has the following columns.

Column	Type	Explanation
seqid	SERIAL	Code uniquely identifying the sequence object
tabid	INTEGER	Identifying code of the sequence as a table object
start_val	INT8	Starting value of the sequence
inc_val	INT8	Value of the increment between successive values
max_val	INT8	Largest possible value of the sequence
min_val	INT8	Smallest possible value of the sequence
cycle	CHAR(1)	Zero means NOCYCLE, 1 means CYCLE
cache	INTEGER	Number of preallocated values in sequence cache
order	CHAR(1)	Zero means NOORDER, 1 means ORDER

SYSSYNONYMS

The **sys synonyms** system catalog table lists the synonyms for each table or view. Except for database servers that have migrated from certain interim releases of Version 1.10 Informix database servers, only the **sysstable** table describes synonyms, and the **sys synonyms** table is unused. It has the following columns.

Column	Type	Explanation
owner	VARCHAR(32)	Name of the owner of the synonym
synname	VARCHAR(128)	Name of the synonym
created	DATE	Date when the synonym was created
tabid	INTEGER	Identifying code of a table, sequence, or view

A composite index on the **owner** and **synonym** columns allows only unique values. The **tabid** column is indexed and allows duplicate values.

SYSSYNTABLE

The **sysstable** system catalog table outlines the mapping between each public or private synonym and the database object (table, sequence, or view) that it represents. It contains one row for each entry in the **sysstable** table that has a **tabtype** value of Por S. The **sysstable** table has the following columns.

Column	Type	Explanation
tabid	INTEGER	Identifying code of the public synonym
servername	VARCHAR(128)	Name of an external database server
dbname	VARCHAR(128)	Name of an external database
owner	VARCHAR(32)	Name of the owner of an external object
tablename	VARCHAR(128)	Name of an external table or view
btabid	INTEGER	Identifying code of a base table, sequence, or view

ANSI-compliant databases do not support public synonyms; their **syssynbtable** tables can describe only synonyms whose **syssynbtable.tabtype** value is P.

If you define a synonym for an object that is in your current database, only the **tabid** and **btabid** columns are used. If you define a synonym for a table that is external to your current database, the **btabid** column is not used, but the **tabid**, **servername**, **dbname**, **owner**, and **tablename** columns are used.

The **tabid** column maps to **systable.tabid**. With the **tabid** information, you can determine additional facts about the synonym from **systable**.

An index on the **tabid** column allows only unique values. The **btabid** column is indexed to allow duplicate values.

SYSTABAMDATA (IDS)

The **systabamdata** system catalog table stores the table-specific hashing parameters of tables that were created with a primary access method.

The **systabamdata** table has the following columns.

Column	Type	Explanation
tabid	INTEGER	Identifying code of the table
am_param	CHAR(256)	Access method parameter choices
am_space	VARCHAR(128)	Name of the storage space holding the data values

The **am_param** column stores configuration parameters that determine how a primary access method accesses a given table. Each configuration parameter in the **am_param** list has the format *keyword=value* or *keyword*.

The **am_space** column specifies the location of the table. It might reside in a cooked file, a different database, or an sbspace within the database server.

The **tabid** column is the primary key to the **systables** table. This column is indexed and must contain unique values.

SYSTABAUTH

The **systabauth** system catalog table describes each set of privileges that are granted on a table, view, sequence, or synonym. It contains one row for each set of table privileges that are granted in the database; the REVOKE statement can modify a row. The **systabauth** table has the following columns.

Column	Type		Explanation
grantor	VARCHAR(32)		Name of the grantor of privilege
grantee	VARCHAR(32)		Name of the grantee of privilege
tabid	INTEGER		Value from systables.tabid for database object
tabauth	CHAR(9) CHAR(8)	IDS XPS	Pattern that specifies privileges on the table, view, synonym, or (IDS) sequence: s or S = Select u or U = Update * = Column-level privilege i or I = Insert d or D = Delete x or X = Index a or A = Alter r or R = References n or N = Under privilege (IDS)

If the **tabauth** column shows a privilege code in uppercase (for example, S for Select), this indicates that the user also has the option to grant that privilege to others. Privilege codes listed in lowercase (for example, s for select) indicate that the user has the specified privilege, but cannot grant it to others.

A hyphen (-) indicates the absence of the privilege corresponding to that position within the **tabauth** pattern.

A **tabauth** value with an asterisk (*) means column-level privileges exist; see also **syscolauth** (page 1-36). (In DB–Access, the **Privileges** option of the **Info** command for a specified table can display the column-level privileges on that table.)

A composite index on **tabid**, **grantor**, and **grantee** allows only unique values. A composite index on **tabid** and **grantee** allows duplicate values.

SYSTABLES

The **systables** system catalog table contains a row for each table object (a table, view, synonym, or in Dynamic Server, a sequence) that has been defined in the database, including the tables and views of the system catalog.

Column	Type	Explanation
tabname	VARCHAR(128)	Name of table, view, synonym, or (for IDS) sequence
owner	VARCHAR(32)	Owner of table (user informix for system catalog tables and <i>username</i> for database tables)
partnum	INTEGER	Physical location code
tabid	SERIAL	System-assigned sequential identifying number
rowsize	SMALLINT	Row size
ncols	SMALLINT	Number of columns in the table
nindexes	SMALLINT	Number of indexes on the table
nrows	INTEGER	Number of rows in the table
created	DATE	Date when the table was created
version	INTEGER	Number that changes when table is altered
tabtype	CHAR(1)	Code indicating the type of object: T = Table E = External Table V = View Q = Sequence (IDS) P = Private synonym S = Public synonym (Type S is not available in an ANSI-compliant database.)
locklevel	CHAR(1)	Lock mode for the table: B = Page P = Page R = Row T = Table (XPS)
npused	INTEGER	Number of data pages that have ever been initialized in the tablespace by the database server
fextsize	INTEGER	Size of initial extent (in kilobytes)
nextsize	INTEGER	Size of all subsequent extents (in kilobytes)

Column	Type		Explanation
flags	SMALLINT		Codes for classifying permanent tables: ST_RAW (= 0x00000010) (IDS) RAW (= 0x00000002) (XPS) STATIC (= 0x00000004) (XPS) OPERATIONAL (= 0x00000008) (XPS) STANDARD (= 0x00000010) (XPS) EXTERNAL (= 0x00000020) (XPS)
site	VARCHAR(128)		Reserved for future use
dbname	VARCHAR(128)		Reserved for future use
type_xid (IDS)	INTEGER		Code from sysxtotypes.extended_id for typed tables, or 0 for untyped tables
am_id (IDS)	INTEGER		Access method code (key to sysams table) NULL or 0 indicates built-in storage manager
minrowsize	SMALLINT	XPS	Minimum row size

Each table, view, sequence, and synonym recorded in the **systables** table is assigned a **tabid**, which is a system-assigned SERIAL value that uniquely identifies the object. The first 99 **tabid** values are reserved for the system catalog. The **tabid** of the first user-defined table object in a database is always 100.

The **tabid** column is indexed and contains only unique values. A composite index on the **tablename** and **owner** columns also requires unique values.

The version column contains an encoded number that is stored in **systables** when a new table is created. Portions of this value are incremented when data-definition statements, such as ALTER INDEX, ALTER TABLE, DROP INDEX, and CREATE INDEX, are performed on the table.

In the **flags** column, ST_RAW represents a nonlogging permanent table in a database that supports transaction logging.

When a prepared statement that references a database table is executed, the version value is checked to make sure that nothing has changed since the statement was prepared. If the version value has changed, the prepared statement is not executed, and you must prepare the statement again.

The **npused** column does not reflect the number of pages used for BYTE or TEXT data, nor the number of pages that are freed in DELETE operations.

The **systables** table has two rows that store information about the database locale: GL_COLLATE with a **tabid** of 90 and GL_CTYPE with a **tabid** of 91. To view these rows, enter the following SELECT statement:

```
SELECT * FROM systables WHERE tabid=90 OR tabid=91
```

SYSTRACECLASSES (IDS)

The **systraceclasses** system catalog table contains the names and identifiers of trace classes. The **systraceclasses** table has the following columns.

Column	Type	Explanation
name	CHAR(18)	Name of the class of trace messages
classid	SERIAL	Identifying code of the trace class

A *trace class* is a category of trace messages that you can use in the development and testing of new DataBlade modules and user-defined routines. Developers use the tracing facility by calling the appropriate DataBlade API routines within their code.

To create a new trace class, insert a row directly into the **systraceclasses** table. By default, all users can view this table, but only users with the DBA privilege can modify it.

The database cannot support tracing unless the MITRACE_OFF configuration parameter is undefined.

A unique index on the **name** column requires each trace class to have a unique name. The database server assigns to each class a unique sequential code. The index on this **classid** column also allows only unique values.

SYSTRACEMSGS (IDS)

The **systracemsgs** system catalog table stores internationalized trace messages that you can use in debugging user-defined routines.

The **sysracemsgs** table has the following columns.

Column	Type	Explanation
name	VARCHAR(128)	Name of the message
msgid	SERIAL	Identifying code of the message template
locale	CHAR(36)	Locale with which this version of the message is associated (for example, en_us.8859-1)
seqno	SMALLINT	Reserved for future use
message	VARCHAR(255)	The message text

DataBlade module developers create a trace message by inserting a row directly into the **sysracemsgs** table. Once a message is created, the development team can specify it either by name or by **msgid** code, using trace statements that the DataBlade API provides.

To create a trace message, you must specify its name, locale, and text. By default, all users can view the **sysracemsgs** table, but only users with the DBA privilege can modify it.

The database cannot support tracing unless the MITRACE_OFF configuration parameter is undefined.

A unique composite index is defined on the **name** and **locale** columns. Another unique index is defined on the **msgid** column.

SYSTRIGBODY

The **systrigbody** system catalog table contains the ASCII text of the trigger definition and the linearized code for the trigger. *Linearized code* is binary data and code that is represented in ASCII format.

Important: The database server uses the linearized code that is stored in **systrigbody**. You must not alter the content of rows that contain linearized code.

The **systrigbody** table has the following columns.

Column	Type	Explanation
trigid	INTEGER	Identifying code of the trigger
datakey	CHAR(1)	Code specifying the type of data: A = ASCII text for the body, triggered actions B = Linearized code for the body D = English text for the header, trigger definition H = Linearized code for the header S = Linearized code for the symbol table
seqno	INTEGER	Page number of this data segment
data	CHAR(256)	English text or linearized code
collation	CHAR(32)	Collating order at the time when trigger was created

A composite index on the **trigid**, **datakey**, and **seqno** columns allows only unique values.

SYSTRIGGERS

The **systriggers** system catalog table contains information about the SQL triggers in the database. This information includes the triggering event and the correlated reference specification for the trigger. The **systriggers** table has the following columns.

Column	Type	Explanation
trigid	SERIAL	Identifying code of the trigger
trigname	VARCHAR(128)	Name of the trigger
owner	VARCHAR(32)	Name of the owner of the trigger
tabid	INTEGER	Identifying code of the triggering table
event	CHAR(1)	Code for the type of triggering event: D = Delete trigger I = Insert trigger U = Update trigger S = Select trigger d = INSTEAD OF Delete trigger i = INSTEAD OF Insert trigger u = INSTEAD OF Update trigger (IDS)
old	VARCHAR(128)	Name of value before update
new	VARCHAR(128)	Name of value after update
mode	CHAR(1)	Reserved for future use

A composite index on the **trigname** and **owner** columns allows only unique values. An index on the **trigid** column also requires unique values. An index on the **tabid** column allows duplicate values.

SYSUSERS

The **sysusers** system catalog table describes each set of privileges that are granted on the database. It contains one row for each user or role that has privileges on the database. This system catalog table has the following columns.

Column	Type	Explanation
username	VARCHAR(32)	Name of the database user or role
usertype	CHAR(1)	Code specifying database-level privileges: C = Connect (work within existing tables) D = DBA (all privileges) G = Role R = Resource (create permanent tables, user-defined data types, and indexes)
priority	SMALLINT	Reserved for future use
password	CHAR(16)	Reserved for future use
defrole	VARCHAR(32)	Name of the default role

An index on **username** allows only unique values. The **username** value can be the login name of a user or the name of a role.

SYSVIEWS

The **sysviews** system catalog table describes each view in the database. Because it stores the **SELECT** statement that created the view, **sysviews** can contain multiple rows for each view. It has the following columns.

Column	Type	Explanation
tabid	INTEGER	Identifying code of the view
seqno	SMALLINT	Line number of the SELECT statement
viewtext	CHAR(64)	Actual SELECT statement used to create the view

A composite index on **tabid** and **seqno** allows only unique values.

SYSVIOLATIONS

The **sysviolations** system catalog table stores information about the constraint violations for base tables. Every table in the database that has a violations table and a diagnostics table associated with it has a corresponding row in the **sysviolations** table, which has the following columns.

Column	Type	Explanation
targettid	INTEGER	Identifying code of the <i>target table</i> (the base table on which the violations table and the diagnostic table are defined)
viotid	INTEGER	Identifying code of the violations table
diatid	INTEGER	Identifying code of the diagnostics table
maxrows	INTEGER	Maximum number of rows that can be inserted into the diagnostics table by a single insert, update, or delete operation on a target table that has a filtering mode object defined on it (IDS) The maximum number of rows allowed in the violations table for each coserver (XPS)

The **maxrows** column also signifies the maximum number of rows that can be inserted in the diagnostics table during a single operation that enables a disabled object or that sets a disabled object to filtering mode (provided that a diagnostics table exists for the target table). If no maximum is specified for the diagnostics or violations table, then **maxrows** contains a NULL value.

Extended Parallel Server does not use the diagnostic table when a constraint violation occurs. Rather, the database server stores additional information in the violations table. The violations table contains the data that the transaction refused and an indication of the cause.

The primary key of this table is the **targettid** column. An additional unique index is also defined on the **viotid** column.

Dynamic Server also has a unique index on the **diatid** column.

SYSXTDDESC (IDS)

The **sysxtddesc** system catalog table provides a text description of each UDT defined in the database. The **sysxtddesc** table has the following columns.

Column	Type	Explanation
extended_id	INTEGER	Code uniquely identifying the extended data types
seqno	SMALLINT	Value to order and identify one line of the description of the UDT A new line is created only if the remaining text string is larger than 255 bytes.
description	CHAR(256)	Textual description of the extended data type

A composite index on **extended_id** and **seqno** allows duplicate values.

SYSXTDTYPEAUTH (IDS)

The **sysxtdtypeauth** system catalog table identifies the privileges for each UDT (user-defined data type). The **sysxtdtypeauth** table contains one row for each set of privileges granted and has the following columns.

Column	Type	Explanation
grantor	VARCHAR(32)	Name of grantor of privilege
grantee	VARCHAR(32)	Name of grantee of privilege
type	INTEGER	Code identifying the UDT
auth	CHAR(2)	Code identifying privileges on the UDT: n or N = Under privilege u or U = Usage privilege

If the privilege code in the **auth** column is uppercase (for example, 'U' for usage), a user who has this privilege can also grant it to others. If the code is in lowercase, a user who has the privilege cannot grant it to others.

A composite index on **type**, **grantor**, and **grantee** allows only unique values. A composite index on the **type** and **grantee** columns allows duplicate values.

SYSXTDTYPES (IDS)

The **sysxtdtype** system catalog table has an entry for each UDT (user-defined data type), including opaque and distinct data types and complex data types (named ROW type, unnamed ROW type, and COLLECTION type), that is defined in the database. The **sysxtdtypes** table has the following columns.

Column	Type	Explanation
extended_id	SERIAL	Unique identifying code for extended data type
domain	CHAR(1)	Code for the domain of the UDT
mode	CHAR(1)	Code classifying the UDT: B = Base (opaque) type C = Collection type or unnamed ROW type D = Distinct type R = Named ROW type ' ' (blank) = Built-in type
owner	VARCHAR(32)	Name of the owner of the UDT
name	VARCHAR(128)	Name of the UDT
type	SMALLINT	Code classifying the UDT
source	INTEGER	The sysxdtypes reference (for distinct types only) Zero (0) indicates that a distinct UDT was created from a built-in data type.
maxlen	INTEGER	The maximum length for variable-length data types Zero indicates a fixed-length UDT.
length	INTEGER	The length in bytes for fixed-length data types Zero indicates a variable-length UDT.
byvalue	CHAR(1)	'T' = UDT is passed by value 'F' = UDT is not passed by value
cannothash	CHAR(1)	'T' = UDT is hashable by default hash function 'F' = UDT is not hashable by default function
align	SMALLINT	Alignment (= 1, 2, 4, or 8) for this UDT
locator	INTEGER	Locator key for unnamed ROW type

Each extended data type is characterized by a unique identifier, called an extended identifier (**extended_id**), a data type identifier (**type**), and the length and description of the data type.

For distinct types created from built-in data types, the **type** column codes correspond to the value of the **syscolumns.coltype** column (indicating the source type) as listed on page 1-22, but incremented by the hexadecimal value 0x0000800. The file **\$INFORMIXDIR/incl/esql/sqltypes.h** contains information about **sysxdtypes.type** and **syscolumns.coltype** codes.

An index on the **extended_id** column allows only unique values. An index on the **locator** column allows duplicate values, as does a composite indexes on the **name** and **owner** columns. A composite index on the **type** and **source** columns also allows duplicate values.

Information Schema (IDS)

The Information Schema consists of read-only views that provide information about all the tables, views, and columns in the current database server to which you have access. These views also provide information about SQL dialects (such as Informix, Oracle, or Sybase) and SQL standards. Note that unlike a system catalog, whose tables describes an individual database, these views describe the Dynamic Server instance, rather than a single database.

This version of the Information Schema views is an X/Open CAE standard. These standards are provided so that applications developed on other database systems can obtain Informix system catalog information without accessing the Informix system catalog tables directly.

Important: Because the X/Open CAE standards Information Schema views differ from ANSI-compliant Information Schema views, it is recommended that you do not install the X/Open CAE Information Schema views on ANSI-compliant databases.

The following Information Schema views are available:

- **tables**
- **columns**
- **sql_languages**
- **server_info**

Sections that follow contain information about how to generate and access Information Schema views as well as information about their structure.

Generating the Information Schema Views

The Information Schema views are generated automatically when you, as DBA, run the following DB–Access command:

```
dbaccess database-name $INFORMIXDIR/etc/xpg4_is.sql
```

The views display data from the system catalog tables. If tables, views, or routines exist with any of the same names as the Information Schema views, you must either rename those database objects or rename the views in the script before you can install the views. You can drop the views with the DROP VIEW statement on each view. To re-create the views, rerun the script.

Important: In addition to the columns specified for each Information Schema view, individual vendors might include additional columns or change the order of the columns. It is recommended that applications not use the forms `SELECT *` or `SELECT table-name*` to access an Information Schema view.

Accessing the Information Schema Views

All Information Schema views have the Select privilege granted to PUBLIC WITH GRANT OPTION so that all users can query the views. Because no other privileges are granted on the Information Schema views, they cannot be updated.

You can query the Information Schema views as you would query any other table or view in the database.

Structure of the Information Schema Views

The following Information Schema views are described in this section:

- **tables**
- **columns**
- **sql_languages**
- **server_info**

In order to accept long identifier names, most of the columns in the views are defined as VARCHAR data types with large maximum sizes.

The tables Information Schema View

The **tables** Information Schema view contains one row for each table to which you have access. It contains the following columns.

Column	Data Type	Explanation
table_schema	VARCHAR(32)	Name of owner of table
table_name	VARCHAR(128)	Name of table or view
table_type	VARCHAR(128)	BASE TABLE for table or VIEW for view
remarks	VARCHAR(255)	Reserved for future use

The visible rows in the **tables** view depend on your privileges. For example, if you have one or more privileges on a table (such as Insert, Delete, Select, References, Alter, Index, or Update on one or more columns), or if privileges are granted to PUBLIC, you see the row that describes that table.

The columns Information Schema View

The **columns** Information Schema view contains one row for each accessible column. It contains the following columns.

Column	Data Type	Explanation
table_schema	VARCHAR(128)	Name of owner of table
table_name	VARCHAR(128)	Name of table or view
column_name	VARCHAR(128)	Name of the column in the table or view
ordinal_position	INTEGER	Position of the column within its table The ordinal_position value is a sequential number that starts at 1 for the first column. This is an Informix extension to XPG4.
data_type	VARCHAR(254)	Name of the data type of the column, such as CHARACTER or DECIMAL
char_max_length	INTEGER	Maximum length (in bytes) for character data types; NULL otherwise
numeric_precision	INTEGER	Uses one of the following values: <ul style="list-style-type: none"> • Total number of digits for exact numeric data types (DECIMAL, INTEGER, MONEY, SMALLINT) • Number of digits of mantissa precision (machine-dependent) for approximate data types (FLOAT, SMALLFLOAT) • NULL for all other data types.
numeric_prec_radix	INTEGER	Uses one of the following values: <ul style="list-style-type: none"> • 2 = Approximate data types (FLOAT and SMALLFLOAT) • 10 = Exact numeric data types (DECIMAL, INTEGER, MONEY, and SMALLINT) • NULL for all other data types
numeric_scale	INTEGER	Number of significant digits to the right of the decimal point for DECIMAL and MONEY data types 0 for INTEGER and SMALLINT types NULL for all other data types
datetime_precision	INTEGER	Number of digits in the fractional part of the seconds for DATE and DATETIME columns; NULL otherwise This column is an Informix extension to XPG4.
is_nullable	VARCHAR(3)	Indicates whether a column allows NULL values; either YES or NO

Column	Data Type	Explanation
remarks	VARCHAR(254)	Reserved for future use

The `sql_languages` Information Schema View

The `sql_languages` Information Schema view contains a row for each instance of conformance to standards that the current database server supports. The `sql_languages` view contains the following columns.

Column	Data Type	Explanation
source	VARCHAR(254)	Organization defining this SQL version
source_year	VARCHAR(254)	Year the source document was approved
conformance	VARCHAR(254)	Standard to which the server conforms
integrity	VARCHAR(254)	Indication of whether this is an integrity enhancement feature; either YES or NO
implementation	VARCHAR(254)	Identification of the SQL product of the vendor
binding_style	VARCHAR(254)	Direct, module, or other binding style
programming_lang	VARCHAR(254)	Host language for which binding style is adapted

The `sql_languages` view is completely visible to all users.

The `server_info` Information Schema View

The `server_info` Information Schema view describes the database server to which the application is currently connected. It contains two columns.

Column	Data Type	Explanation
server_attribute	VARCHAR(254)	An attribute of the database server
attribute_value	VARCHAR(254)	Value of the <code>server_attribute</code> as it applies to the current database server

Each row in this view provides information about one attribute. X/Open-compliant databases must provide applications with certain required information about the database server.

The **server_info** view includes the following **server_attribute** information.

server_attribute	Explanation
identifier_length	Maximum number of bytes for a user-defined identifier
row_length	Maximum number of bytes in a row
userid_length	Maximum number of bytes in a user name
txn_isolation	Initial transaction isolation level for the database server: Read Uncommitted(= Default isolation level for databases with no transaction logging; also called Dirty Read) Read Committed(= Default isolation level for databases that are not ANSI-compliant but that support explicit transaction logging) Serializable(= Default isolation level for ANSI-compliant databases; also called Repeatable Read)
collation_seq	Assumed ordering of the character set for the database server The following values are possible: ISO 8859-1EBCDIC The default Informix representation shows ISO 8859-1.

The **server_info** view is completely visible to all users.

Chapter 2. Data Types

Summary of Data Types	2-2
Description of Data Types	2-6
BIGINT (XPS)	2-6
BLOB (IDS)	2-6
BOOLEAN (IDS)	2-7
BYTE	2-7
CHAR(n)	2-8
CHARACTER(n)	2-9
CHARACTER VARYING(m,r)	2-9
CLOB (IDS)	2-10
DATE	2-11
DATETIME	2-11
DEC	2-15
DECIMAL	2-15
Distinct (IDS)	2-17
DOUBLE PRECISION	2-18
FLOAT(n)	2-18
INT	2-19
INT8	2-19
INTEGER	2-19
INTERVAL	2-19
LIST(e) (IDS)	2-22
LVARCHAR(m) (IDS)	2-23
MONEY(p,s)	2-24
MULTISET(e) (IDS)	2-25
NCHAR(n)	2-26
NUMERIC(p,s)	2-26
NVARCHAR(m,r)	2-26
Opaque (IDS)	2-26
REAL	2-27
ROW, Named (IDS)	2-27
ROW, Unnamed (IDS)	2-28
SERIAL(n)	2-30
SERIAL8(n)	2-31
SET(e) (IDS)	2-32
SMALLFLOAT	2-33
SMALLINT	2-34
TEXT	2-34
VARCHAR(m,r)	2-36
Built-In Data Types	2-38
Large-Object Data Types	2-38
Simple Large Objects	2-39
Smart Large Objects (IDS)	2-39
Time Data Types	2-40

Extended Data Types (IDS)	2-46
Complex Data Types	2-46
Collection Data Types	2-47
ROW Data Types	2-48
Distinct Data Types	2-49
Opaque Data Types	2-49
Data Type Casting and Conversion	2-50
Using Built-in Casts	2-50
Converting from Number to Number	2-51
Converting Between Number and Character	2-52
Converting Between INTEGER and DATE	2-52
Converting Between DATE and DATETIME	2-53
Using User-Defined Casts	2-53
Implicit Casts	2-53
Explicit Casts	2-53
Determining Which Cast to Apply	2-54
Casts for Distinct Types	2-54
What Extended Data Types Can Be Cast?	2-56
Operator Precedence	2-56

In This Chapter

Every column in a table in a database is assigned a data type. The data type precisely defines the kinds of values that you can store in that column.

This chapter describes built-in and extended data types, casting between two data types, and operator precedence.

Summary of Data Types

Figure 2-1 shows the logical categories of data types that Informix database servers support. Shaded categories are for Dynamic Server only.

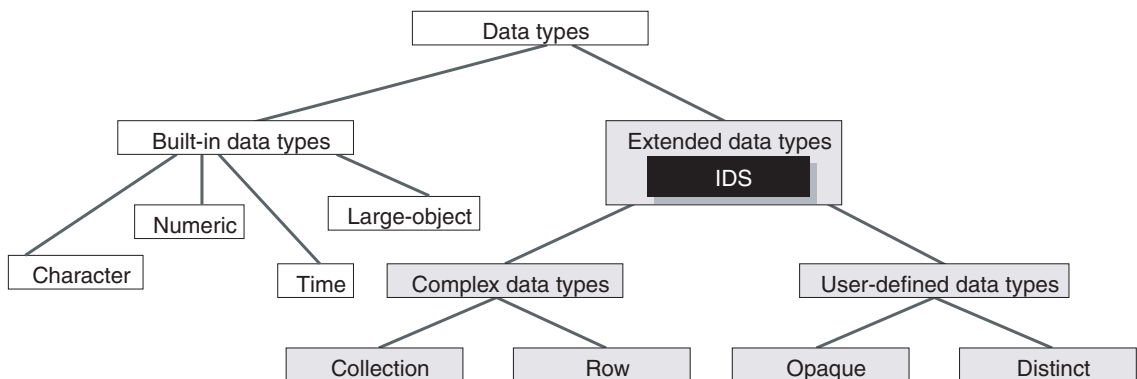


Figure 2-1. Overview of Supported Data Types

This diagram is simplified; some built-in types are implemented as opaque types, and are only supported on Dynamic Server. That is, *opaque* and *built-in* are not disjunct categories, though most built-in data types are not opaque.

Built-in data types (which are system-defined) and *extended* data types (which you can define) share the following characteristics. You can:

- Use them to create columns within database tables.
- Declare them as arguments and as returned types of routines.
- Use them as base types from which to create DISTINCT data types.
- Cast them to other data types.
- Declare and access host variables of these types in SPL and ESQL/C.

For exceptions, see the description of each data type. For an overview, see “Built-In Data Types” on page 2-38 and “Extended Data Types (IDS)” on page 2-46.

You assign data types to columns with the CREATE TABLE statement and change them with the ALTER TABLE statement. When you change an existing column data type, all data is converted to the new data type, if possible.

For information on the ALTER TABLE and CREATE TABLE statements, on SQL statements that create specific data types, that create and drop casts, and on other data type topics, refer to the *IBM Informix: Guide to SQL Syntax*.

For information about how to create and use complex data types of Dynamic Server, see the *IBM Informix: Database Design and Implementation Guide*. For information about how to create user-defined data types, see *IBM Informix: User-Defined Routines and Data Types Developer’s Guide*.

All Informix database servers support the data types that Table 2-1 lists. This chapter describes each of these built-in data types.

Table 2-1. Data Types That All Informix Database Servers Support

Data Type	Explanation	Page
BIGINT	Is a synonym (in XPS only) for INT8	2-19
BYTE	Stores any kind of binary data, up to 2 ³¹ bytes in length	2-7
CHAR(<i>n</i>)	Stores character strings; collation is in code-set order	2-8
CHARACTER(<i>n</i>)	Is a synonym for CHAR	2-9
CHARACTER VARYING(<i>m,r</i>)	Stores character strings of varying length (ANSI compliant); collation is in code-set order	2-9
DATE	Stores calendar dates	2-11
DATETIME	Stores calendar date combined with time of day	2-11

Table 2-1. Data Types That All Informix Database Servers Support (continued)

Data Type	Explanation	Page
DEC	Is a synonym for DECIMAL	2-15
DECIMAL(<i>p</i>)	Stores floating-point numbers with definable precision; if database is ANSI-compliant, the scale is zero.	2-15
DECIMAL(<i>p, s</i>)	Stores fixed-point numbers of defined scale and precision	2-16
DOUBLE PRECISION	Synonym for FLOAT	2-18
FLOAT(<i>n</i>)	Stores double-precision floating-point numbers corresponding to the double data type in C	2-18
INT	Is a synonym for INTEGER	2-19
INT8	Stores 8-byte integer values in range $-(2^{63} - 1)$ to $2^{63} - 1$	2-19
INTEGER	Stores whole numbers in a range from -2,147,483,647 to +2,147,483,647	2-19
INTERVAL (Year Month)	Stores a span of time (or level of effort) in units of <i>years</i> and <i>months</i> .	2-19
INTERVAL (Day Fraction)	Stores a span of time in a contiguous set of units of <i>days, hours, minutes, seconds, and fractions of a second</i> .	2-19
MONEY(<i>p,s</i>)	Stores currency amounts	2-24
NCHAR(<i>n</i>)	Same as CHAR, but can support localized collation	2-26
NUMERIC(<i>p,s</i>)	Synonym for DECIMAL(<i>p,s</i>)	2-26
NVARCHAR(<i>m,r</i>)	Same as VARCHAR, but can support localized collation	2-26
REAL	Is a synonym for SMALLFLOAT	2-27
SERIAL	Stores sequential integers (> 0) in positive range of INT	2-30
SERIAL8	Stores sequential integers (> 0) in positive range of INT8	2-31
SMALLFLOAT	Stores single-precision floating-point numbers corresponding to the float data type of the C language	2-33
SMALLINT	Stores whole numbers from -32,767 to +32,767	2-34
TEXT	Stores any kind of text data, up to 2^{31} bytes in length	2-34
VARCHAR(<i>m,r</i>)	Stores character strings of varying length (up to 255 bytes); collation is in code-set order.	2-36

For the character data types (CHAR, CHAR VARYING, LVARCHAR, NCHAR, NVARCHAR, and VARCHAR), a data string can include letters, digits, punctuation, whitespace, diacritical marks, ligatures, and other printable

symbols from the code set of the database locale. (For some East Asian locales, multibyte characters are supported within data strings.)

Dynamic Server also supports additional data types that Table 2-2 lists.

Table 2-2. Additional Data Types That Dynamic Server Supports

Data Type	Explanation	Page
BLOB	Stores binary data in random-access chunks	2-6
BOOLEAN	Stores Boolean values true and false	2-7
CLOB	Stores text data in random-access chunks	2-10
Distinct	Stores data in a user-defined type that has the same format as a source type on which it is based, but its casts and functions can differ from those on the source type	2-17
LIST(<i>e</i>)	Stores a sequentially ordered collection of elements, all of the same data type, <i>e</i> ; allows duplicate values	2-22
LVARCHAR(<i>m</i>)	Stores variable-length strings of up to 32,739 bytes	2-23
MULTISET(<i>e</i>)	Stores a non-ordered collection of values, with elements all of the same data type, <i>e</i> ; allows duplicate values.	2-25
Opaque	Stores a user-defined data type whose internal structure is inaccessible to the database server	2-26
ROW, Named	Stores a named ROW type	2-27
ROW, Unnamed	Stores an unnamed ROW type	2-28
SET(<i>e</i>)	Stores a non-ordered collection of elements, all of the same data type, <i>e</i> ; does not allow duplicate values	2-32

Distributed DML operations and function calls that access databases of other database servers cannot return these extended data types of Dynamic Server, which are individually described in this chapter. (Cross-database operations on other databases of the same Dynamic Server instance, however, can access BOOLEAN, BLOB, CLOB, and LVARCHAR data types, which are implemented as built-in opaque types. Such operations can also access DISTINCT types whose base types are built-in types, and user-defined types (UDTs), if the UDTs and DISTINCT types are explicitly cast to built-in types, and if all of the UDTs, casts, and DISTINCT types are defined in all the participating databases.)

For information about Informix internal data types that SQL statements support (such as IMPEX, IMPEXBIN, and SENDRECV), see *IBM Informix: User-Defined Routines and Data Types Developer's Guide*.

Description of Data Types

This section describes the data types that Informix database servers support.

BIGINT (XPS)

The BIGINT data type of Extended Parallel Server is a synonym for INT8. (See the description of INT8 on page 2-19.)

BLOB (IDS)

The BLOB data type stores any kind of binary data in random-access chunks, called sbspaces. Binary data typically consists of saved spreadsheets, program-load modules, digitized voice patterns, and so on. The database server performs no interpretation of the contents of a BLOB column. A BLOB column can be up to 4 terabytes (4×2^{40} bytes) in length, though your system resources might impose a lower practical limit.

The term *smart large object* refers to BLOB and CLOB data types. Use CLOB data types (see page 2-10) for random access to text data. For general information about BLOB and CLOB data types, see “Smart Large Objects (IDS)” on page 2-39.

You can use these SQL functions to perform operations on a BLOB column:

- **FILETOBLOB** copies a file into a BLOB column.
- **LOTOFILE** copies a BLOB (or CLOB) value into an operating-system file.
- **LOCOPY** copies an existing smart large object to a new smart large object.

For more information on these SQL functions, see the *IBM Informix: Guide to SQL Syntax*.

Within SQL, you are limited to the equality (=) comparison operation and the encryption and decryption functions for BLOB data. (The encryption and decryption functions are described in the *IBM Informix: Guide to SQL Syntax*.) To perform additional operations, you must use one of the application programming interfaces (APIs) from within your client application.

You can insert data into BLOB columns in the following ways:

- With the **dbload** or **onload** utilities
- With the LOAD statement (DB–Access)
- With the **FILETOBLOB** function
- From BLOB (**ifx_lo_t**) host variables (IBM Informix ESQL/C)

If you select a BLOB column using DB–Access, only the string <SBl ob va lue> is returned; no actual value is displayed.

BOOLEAN (IDS)

The BOOLEAN data type stores TRUE or FALSE data values as a single byte. This table shows internal and literal representations of the BOOLEAN data type.

Logical Value	Internal Representation	Literal Representation
TRUE	\0	't'
FALSE	\1	'f'
NULL	Internal Use Only	NULL

You can compare two BOOLEAN values to test for equality or inequality. You can also compare a BOOLEAN value to the Boolean literals 't' and 'f'. BOOLEAN values are case insensitive; 't' is equivalent to 'T' and 'f' to 'F'.

You can use a BOOLEAN column to store what a Boolean expression returns. In the following example, the value of **boolean_column** is 't' if **column1** is less than **column2**, 'f' if **column1** is greater than or equal to **column2**, and NULL if the value of either **column1** or **column2** is unknown:

```
UPDATE my_table SET boolean_column = lessthan(column1, column2)
```

BYTE

The BYTE data type stores any kind of binary data in an undifferentiated byte stream. Binary data typically consists of digitized information, such as spreadsheets, program load modules, digitized voice patterns, and so on. The term *simple large object* refers to BYTE and text data types. No more than 195 columns of the same table can be declared as BYTE and text data types.

The BYTE data type has no maximum size. A BYTE column has a theoretical limit of 2^{31} bytes and a practical limit that your disk capacity determines.

You can store, retrieve, update, or delete the contents of a BYTE column. You cannot, however, use BYTE operands in arithmetic or string operations, nor assign literals to BYTE columns with the SET clause of the UPDATE statement. You also cannot use BYTE items in any of the following ways:

- With aggregate functions
- With the IN clause
- With the MATCHES or LIKE clauses
- With the GROUP BY clause
- With the ORDER BY clause

BYTE operands are valid in Boolean expressions only when you are testing for NULL values with the IS NULL or IS NOT NULL operators.

You can insert data into BYTE columns in the following ways:

- With the **dbload** or **onload** utilities
- With the LOAD statement (DB–Access)
- From BYTE host variables (IBM Informix ESQL/C)

You cannot use a quoted text string, number, or any other actual value to insert or update BYTE columns.

When you select a BYTE column, you can choose to receive all or part of it. To retrieve it all, use the regular syntax for selecting a column. You can also select any part of a BYTE column by using subscripts, as the next example, which reads the first 75 bytes of the **cat_picture** column associated with the catalog number 10001:

```
SELECT cat_picture [1,75] FROM catalog WHERE catalog_num = 10001
```

A built-in cast converts BYTE values to BLOB values. For more information, see the *IBM Informix: Database Design and Implementation Guide*.

If you select a BYTE column using the DB–Access Interactive Schema Editor, only the string "<BYTE value>" is returned; no data value is displayed.

Important: If you try to return a BYTE column from a subquery, an error results, even if the column is not used in a Boolean expression nor with an aggregate.

CHAR(n)

The CHAR data type stores any string of letters, numbers, and symbols. It can store single-byte and multibyte characters, based on the database locale. (For more information on East Asian locales that support multibyte code sets, see "Multibyte Characters with VARCHAR" on page 2-37.)

A CHAR(*n*) column has a length of *n* bytes, where $1 \leq n \leq 32,767$. If you do not specify *n*, CHAR(1) is the default length. Character columns typically store alphanumeric strings, such as names, addresses, phone numbers, and so on. When a value is retrieved or stored as CHAR(*n*), exactly *n* bytes of data are transferred. If the string is shorter than *n* bytes, the string is extended with blank spaces up to the declared length. If the data value is longer than *n* bytes, a data string of length *n* that has been truncated from the right is inserted or retrieved, without the database server raising an exception.

This does not create partial characters in multibyte locales. In right-to-left locales, such as Arabic, Hebrew, or Farsi, the truncation is from the left.

Treating CHAR Values as Numeric Values

If you plan to perform calculations on numbers stored in a column, you should assign a number data type to that column. Although you can store numbers in CHAR columns, you might not be able to use them in some arithmetic operations. For example, if you insert a sum into a CHAR column, you might experience overflow problems if the CHAR column is too small to hold the value. In this case, the insert fails. Numbers that have leading zeros (such as some zip codes) have the zeros stripped if they are stored as number types INTEGER or SMALLINT. Instead, store these numbers in CHAR columns.

Sorting and Relational Comparisons

In general, the collating order for sorting CHAR values is the order of characters in the code set. (An exception is the MATCHES operator with ranges; see “Collating VARCHAR Values” on page 2-37.) For more information about collation order, see the *IBM Informix: GLS User’s Guide*.

For multibyte locales, the database supports any multibyte characters in the code set. When storing multibyte characters in a CHAR data type, make sure to calculate the number of bytes needed. For more information on multibyte characters and locales, see the *IBM Informix: GLS User’s Guide*.

CHAR values are compared to other CHAR values by padding the shorter value on the right with blank spaces until the values have equal length, and then comparing the two values, using the code-set order for collation.

Nonprintable Characters with CHAR

A CHAR value can include tab, newline, whitespace, and nonprintable characters. You must, however, use an application to insert nonprintable characters into host variables and the host variables into your database. After passing nonprintable characters to the database server, you can store or retrieve them. After you select nonprintable characters, fetch them into host variables and display them with your own display mechanism.

If you try to display nonprintable characters with DB–Access, your screen returns inconsistent results. (Which characters are nonprintable is locale-dependent. For more information, see the discussion of code-set conversion between the client and the database server in the *IBM Informix: GLS User’s Guide*.)

CHARACTER(n)

The CHARACTER data type is a synonym for CHAR.

CHARACTER VARYING(m,r)

The CHARACTER VARYING data type stores a string of letters, digits, and symbols of varying length, where *m* is the maximum size of the column (in

bytes) and r is the minimum number of bytes reserved for that column. The CHARACTER VARYING data type complies with ANSI/ISO standard for SQL; the non-ANSI VARCHAR data type supports the same functionality. For more information, see the description of the VARCHAR type in “VARCHAR(m,r)” on page 2-36.

CLOB (IDS)

The CLOB data type stores any kind of text data in random-access chunks, called sbspaces. Text data can include text-formatting information, as long as this information is also textual, such as PostScript, Hypertext Markup Language (HTML), Standard Graphic Markup Language (SGML), or Extensible Markup Language (XML) data.

The term *smart large object* refers to CLOB and BLOB data types. The CLOB data type supports special operations for character strings that are inappropriate for BLOB values. A CLOB value can be up to 4 terabytes (4×2^{40} bytes) in length.

Use the BLOB data type (see “BLOB (IDS)” on page 2-6) for random access to binary data. For general information about the CLOB and BLOB data types, see “Smart Large Objects (IDS)” on page 2-39.

The following SQL functions can perform operations on a CLOB column:

- **FILETOCLOB** copies a file into a CLOB column.
- **LOTOFILE** copies a CLOB (or BLOB) value into a file.
- **LOCOPY** copies a CLOB (or BLOB) value to a new smart large object.
- **ENCRYPT_DES** or **ENCRYPT_TDES** creates an encrypted BLOB value from a plain-text CLOB argument.
- **DECRYPT_BINAR** or **DECRYPT_CHAR** returns an unencrypted BLOB value from an encrypted BLOB argument (that **ENCRYPT_DES** or **ENCRYPT_TDES** created from a plain-text CLOB value).

For more information on these SQL functions, see the *IBM Informix: Guide to SQL Syntax*.

No casts exist for CLOB data. Therefore, the database server cannot convert data of the CLOB type to any other data type, except by using these encryption and decryption functions to return a BLOB. Within SQL, you are limited to the equality (=) comparison operation for CLOB data. To perform additional operations, you must use one of the application programming interfaces from within your client application.

Multibyte Characters with CLOB

You can insert data into CLOB columns in the following ways:

- With the **dbload** or **onload** utilities

- With the LOAD statement (DB–Access)
- From CLOB (`ifx_lo_t`) host variables (ESQL/C).

For examples of CLOB types, see the *IBM Informix: Guide to SQL Tutorial* and the *IBM Informix: Database Design and Implementation Guide*.

With GLS, the following rules apply:

- Multibyte CLOB characters must be defined in the database locale.
- The CLOB data type is collated in code-set order.
- The database server handles code-set conversions for CLOB data.

For more information on database locales, collation order, and code-set conversion, see the *IBM Informix: GLS User's Guide*.

DATE

The DATE data type stores the calendar date. DATE data types require four bytes. A calendar date is stored internally as an integer value equal to the number of days since December 31, 1899.

Because DATE values are stored as integers, you can use them in arithmetic expressions. For example, you can subtract a DATE value from another DATE value. The result, a positive or negative INTEGER value, indicates the number of days that elapsed between the two dates. (You can use a UNITS DAY expression to convert the result to an INTERVAL DAY TO DAY data type.)

The following example shows the default display format of a DATE column:

```
mm/dd/yyyy
```

In this example, *mm* is the month (1-12), *dd* is the day of the month (1-31), and *yyyy* is the year (0001-9999). You can specify a different order of time units and a different time-unit separator than / (or no separator) by setting the **DBDATE** environment variable. For more information, see “DBDATE” on page 3-25.

In non-default locales, you can display dates in culture-specific formats. The locale and the **GL_DATE** and **DBDATE** environment variables (as described in the next chapter) affect the display formatting of DATE values. They do not, however, affect the internal storage format for DATE columns in the database. For more information, see the *IBM Informix: GLS User's Guide*.

DATETIME

The DATETIME data type stores an instant in time expressed as a calendar date and time of day. You choose how precisely a DATETIME value is stored; its precision can range from a year to a fraction of a second.

DATETIME stores a data value as a contiguous series of fields that represents each time unit (*year, month, day*, and so forth) in the data type declaration.

Field qualifiers to specify a DATETIME data type have this format:

DATETIME *largest_qualifier* TO *smallest_qualifier*

This resembles an INTERVAL field qualifier (see “INTERVAL” on page 2-19), but DATETIME represents a point in time, rather than (like INTERVAL) a span of time. These differences exist between DATETIME and INTERVAL qualifiers:

- The DATETIME keyword replaces the INTERVAL keyword.
- DATETIME field qualifiers cannot specify a non-default precision for the *largest_qualifier* time unit.
- A DATETIME value that includes YEAR and/or MONTH time units can also include smaller time units, whereas an INTERVAL data type that stores days (or smaller time units) cannot store months or years.

The *largest_qualifier* and *smallest_qualifier* of a DATETIME data type can be any of the fields that Table 2-3 lists, provided that *smallest_qualifier* does not specify a larger time unit than *largest_qualifier*. (The largest and smallest time units can be the same; for example, DATETIME YEAR TO YEAR.)

Table 2-3. DATETIME Field Qualifiers

Qualifier Field	Valid Entries
YEAR	A year numbered from 1 to 9,999 (A.D.)
MONTH	A month numbered from 1 to 12
DAY	A day numbered from 1 to 31, as appropriate to the month
HOUR	An hour numbered from 0 (midnight) to 23
MINUTE	A minute numbered from 0 to 59
SECOND	A second numbered from 0 to 59
FRACTION	A decimal fraction-of-a-second with up to 5 digits of scale. The default scale is 3 digits (a thousandth of a second). For <i>smallest_qualifier</i> to specify another scale, write FRACTION(<i>n</i>), where <i>n</i> is the desired number of digits from 1 to 5.

The declaration of a DATETIME column need not include the full YEAR to FRACTION range of time units. It can include any contiguous subset of these time units, or even only a single time unit.

For example, you can enter a MONTH TO HOUR value in a column declared as YEAR TO MINUTE, as long as each entered value contains information for

a contiguous series of time units. You cannot, however, enter a value for only the MONTH and HOUR; the entry must also include a value for DAY.

If you use the DB–Access TABLE menu, and you do not specify the DATETIME qualifiers, a default DATETIME qualifier, YEAR TO YEAR, is assigned.

A valid DATETIME literal must include the DATETIME keyword, the values to be entered, and the field qualifiers. You must include these qualifiers because, as noted earlier, the value that you enter can contain fewer fields than were declared for that column. Acceptable qualifiers for the first and last fields are identical to the list of valid DATETIME fields that Table 2-3 on page 2-12 lists.

Write values for the field qualifiers as integers and separate them with delimiters. Table 2-4 lists the delimiters that are used with DATETIME values in the default U.S. English locale. (These are a superset of the delimiters that are used in INTERVAL values; see Table 2-6 on page 2-21.)

Table 2-4. Delimiters Used with DATETIME

Delimiter	Placement in DATETIME Literal
Hyphen (-)	Between the YEAR, MONTH, and DAY time-unit values
Blank space ()	Between the DAY and HOUR time-unit values
Colon (:)	Between the HOUR, MINUTE, and SECOND time-unit values
Decimal point (.)	Between the SECOND and FRACTION time-unit values

Figure 2-2 shows a DATETIME YEAR TO FRACTION(3) value with delimiters.

2003-09-23 12:42:06.001



↑ ↑ ↑ ↑ ↑ ↑ ↑
year Month Day Hour Minute Second Fraction

Figure 2-2. Example DATETIME Value with Delimiters

When you enter a value with fewer time-unit fields than in the column, the value that you enter is expanded automatically to fill all the declared time-unit fields. If you leave out any more significant fields, that is, time units larger than any that you include, those fields are filled automatically with the

current values for those time units from the system clock calendar. If you leave out any less-significant fields, those fields are filled with zeros (or with 1 for MONTH and DAY) in your entry.

You can also enter DATETIME values as character strings. The character string must include information for each field defined in the DATETIME column. The INSERT statement in the following example shows a DATETIME value entered as a character string:

```
INSERT INTO cust_calls (customer_num, call_dtime, user_id,  
                      call_code, call_descr)  
VALUES (101, '2001-01-14 08:45', 'maryj', 'D',  
       'Order late - placed 6/1/00')
```

If **call_dtime** is declared as DATETIME YEAR TO MINUTE, the character string must include values for the *year*, *month*, *day*, *hour*, and *minute* fields.

If the character string does not contain information for all the declared fields (or if it adds additional fields), then the database server returns an error.

All fields of a DATETIME column are two-digit numbers except for the *year* and *fraction* fields. The *year* field is stored as four digits. When you enter a two-digit value in the year field, how the abbreviated year is expanded to four digits depends on the setting of the **DBCENTURY** environment variable.

For example, if you enter 02 as the *year* value, whether the year is interpreted as 1902, 2002, or 2102 depends on the setting of **DBCENTURY** and on the value of the system clock calendar at execution time. If you do not set **DBCENTURY**, then the leading digits of the current year are appended by default. For information about setting **DBCENTURY**, see “DBCENTURY” on page 3-22.

The *fraction* field requires n digits where $1 \leq n \leq 5$, rounded up to an even number. You can use the following formula (rounded up to a whole number of bytes) to calculate the number of bytes that a DATETIME value requires:
 $(total\ number\ of\ digits\ for\ all\ fields) / 2 + 1$

For example, a YEAR TO DAY qualifier requires a total of eight digits (four for *year*, two for *month*, and two for *day*). According to the formula, this data value requires 5, or $(8/2) + 1$, bytes of storage.

For information on how to use DATETIME values in arithmetic and relational expressions, see “Manipulating DATE with DATETIME and INTERVAL Values” on page 2-43. For more information on the DATETIME data type, see the *IBM Informix: Guide to SQL Syntax*.

If you specify a locale other than U.S. English, the locale defines the culture-specific display formats for DATETIME values. To change the default display format, change the setting of the **GL_DATETIME** environment variable.

With an ESQL API, the **DBTIME** environment variable also affects DATETIME formatting. Non-default locales and settings of the **GL_DATE** and **DBDATE** environment variables also affect the display of datetime data. They do not, however, affect the internal storage format of a DATETIME column.

The USEOSTIME configuration parameter can affect the subsecond granularity when the database server obtains the current time from the operating system in SQL statements; for details, see the *IBM Informix: Administrator's Reference*.

For more information on **DBTIME**, see “DBTIME” on page 3-39. For more information on **DBCENTURY**, see “DBCENTURY” on page 3-22. For more information on locales and GLS environment variables that can specify end-user DATETIME formats, see the *IBM Informix: GLS User's Guide*.

DEC

The DEC data type is a synonym for DECIMAL.

DECIMAL

The DECIMAL data type can take two forms: DECIMAL(*p*) floating point and DECIMAL(*p,s*) fixed point. In an ANSI-compliant database, however, all DECIMAL numbers are fixed point. By default, literal numbers that include a decimal (.) point are interpreted by the database server as DECIMAL values.

DECIMAL(*p*) Floating Point

The DECIMAL data type stores decimal floating-point numbers up to a maximum of 32 significant digits, where *p* is the total number of significant digits (the *precision*).

Specifying precision is optional. If you specify no precision (*p*), DECIMAL is treated as DECIMAL(16), a floating-point decimal with a precision of 16 places. DECIMAL(*p*) has an absolute exponent range between 10^{-130} and 10^{124} .

If you declare a DECIMAL(*p*) column in an ANSI-compliant database, the scale defaults to DECIMAL(*p*, 0), meaning that only integer values can be stored in this data type.

In a database that is not ANSI-compliant, a DECIMAL(*p*) is a floating-point data type of a scale large enough to store the exponential notation for a value.

For example, the following calculation shows how many bytes of storage a DECIMAL(5) column requires in the default locale (where the decimal point occupies a single byte):

1 byte for the sign of the data value
1 byte for the first digit
1 byte for the decimal point
4 bytes for the rest of the digits in the declared precision of (5) - 1
1 byte for the 'e' symbol
1 byte for the sign of the exponent
3 bytes for the exponent

12 bytes (Total)

Thus, "12345" in a DECIMAL(5) column is displayed as "12345.00000" (that is, with a scale of 6) in a database that is not ANSI-compliant.

DECIMAL (p,s) Fixed Point

In fixed-point numbers, DECIMAL(*p,s*), the decimal point is fixed at a specific place, regardless of the value of the number. When you specify a column of this type, you declare its precision (*p*) as the total number of digits that it can store, from 1 to 32. You declare its *scale* (*s*) as the total number of digits in the fractional part (that is, to the right of the decimal point).

All numbers with an absolute value less than $0.5 * 10^{-s}$ have the value zero. The largest absolute value of a DECIMAL(*p,s*) data type that you can store without an overflow error is $10^{p-s} - 10^{-s}$. A DECIMAL column typically stores numbers with fractional parts that must be stored and displayed exactly (for example, rates or percentages). In an ANSI-compliant database, all DECIMAL numbers must have absolute values in the range 10^{-32} to 10^{+31} .

DECIMAL Storage

The database server uses one byte of disk storage to store two digits of a decimal number, plus an additional byte to store the exponent and sign, with the first byte representing a sign bit and a 7-bit exponent in excess-65 format. The rest of the bytes express the mantissa as base-100 digits. The significant digits to the left of the decimal and the significant digits to the right of the decimal are stored in separate groups of bytes. At the maximum *precision* specification, DECIMAL(32,s) data types can store s-1 decimal digits to the right of the decimal point, if s is an odd number.

How the database server stores decimal numbers is illustrated in the following example. If you specify DECIMAL(6,3), the data type consists of three significant digits in the integral part and three significant digits in the fractional part (for instance, 123.456). The three digits to the left of the decimal

are stored on 2 bytes (where one of the bytes only holds a single digit) and the three digits to the right of the decimal are stored on another 2 bytes, as Figure 2-3 illustrates.

(The exponent byte is not shown.) With the additional byte required for the exponent and sign, DECIMAL(6,3) requires a total of 5 bytes of storage.

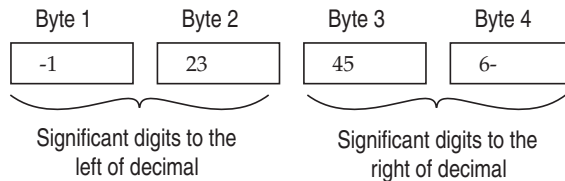


Figure 2-3. Schematic That Illustrates the Storage of Digits in a Decimal (*p,s*) Value

You can use the following formulas (rounded down to a whole number of bytes) to calculate the byte storage (*N*) for a DECIMAL(*p,s*) data type (where *N* includes the byte that is required to store the exponent and the sign):

If the *scale* is odd: $N = (\textit{precision} + 4) / 2$

If the *scale* is even: $N = (\textit{precision} + 3) / 2$

For example, the data type DECIMAL(5,3) requires 4 bytes of storage (9/2 rounded down equals 4).

There is one caveat to these formulas. The maximum number of bytes the database server uses to store a decimal value is 17. One byte is used to store the exponent and sign, leaving 16 bytes to store up to 32 digits of precision. If you specify a precision of 32 and an *odd* scale, however, you lose 1 digit of precision. Consider, for example, the data type DECIMAL(32,31). This decimal is defined as 1 digit to the left of the decimal and 31 digits to the right. The 1 digit to the left of the decimal requires 1 byte of storage. This leaves only 15 bytes of storage for the digits to the right of the decimal. The 15 bytes can accommodate only 30 digits, so 1 digit of precision is lost.

Distinct (IDS)

A *distinct* type is a data type that is derived from one of the following source types (called the *base type*):

- A built-in type
- An existing distinct type
- An existing named ROW type
- An existing opaque type

A distinct type inherits from its source type the length and alignment on the disk. A distinct type thus makes efficient use of the preexisting functionality of the database server.

When you create a distinct data type, the database server automatically creates two explicit casts: one cast from the distinct type to its source type and one cast from the source type to the distinct type. A distinct type based on a built-in source type does not inherit the built-in casts that are provided for the built-in type. A distinct type does inherit, however, any user-defined casts that have been defined on the source type.

A distinct type cannot be compared directly to its source type. To compare the two types, you must first explicitly cast one type to the other.

You must define a distinct type in the database. Definitions of distinct types are stored in the **sysxdtypes** system catalog table. The following SQL statements maintain the definitions of distinct types in the database:

- The CREATE DISTINCT TYPE statement adds a distinct type to the database.
- The DROP TYPE statement removes a previously defined distinct type from the database.

For more information about the SQL statements mentioned above, see the *IBM Informix: Guide to SQL Syntax*. For information about casting distinct data types, see “Casts for Distinct Types” on page 2-54. For examples that show how to create and register cast functions for a distinct type, see the *IBM Informix: Database Design and Implementation Guide*.

DOUBLE PRECISION

The DOUBLE PRECISION keywords are a synonym for the FLOAT keyword.

FLOAT(*n*)

The FLOAT data type stores double-precision floating-point numbers with up to 17 significant digits. FLOAT corresponds to IEEE 4-byte floating-point, and to the **double** data type in C. The range of values for the FLOAT data type is the same as the range of the C **double** data type on your computer.

You can use *n* to specify the precision of a FLOAT data type, but SQL ignores the precision. The value *n* must be a whole number between 1 and 14.

A column with the FLOAT data type typically stores scientific numbers that can be calculated only approximately. Because floating-point numbers retain only their most significant digits, the number that you enter in this type of column and the number the database server displays can differ slightly.

The difference between the two values depends on how your computer stores floating-point numbers internally. For example, you might enter a value of 1.1000001 into a FLOAT field and, after processing the SQL statement, the database server might display this value as 1.1. This situation occurs when a

value has more digits than the floating-point number can store. In this case, the value is stored in its approximate form with the least significant digits treated as zeros.

FLOAT data types usually require 8 bytes of storage per value. Conversion of a FLOAT value to a DECIMAL value results in 17 digits of precision.

INT

The INT data type is a synonym for INTEGER.

INT8

The INT8 data type stores whole numbers that can range in value from $-9,223,372,036,854,775,807$ to $9,223,372,036,854,775,807$ [or $-(2^{63}-1)$ to $2^{63}-1$], for 18 or 19 digits of precision. The number $-9,223,372,036,854,775,808$ is a reserved value that cannot be used. The INT8 data type is typically used to store large counts, quantities, and so on.

Dynamic Server stores INT8 data in internal format that can require up to 10 bytes of storage. Extended Parallel Server stores INT8 values as 8 bytes.

Arithmetic operations and sort comparisons are performed more efficiently on integer data than on floating-point or fixed-point decimal data, but INT8 cannot store data with absolute values beyond $|2^{63}-1|$. If a value exceeds the numeric range of INT8, the database server does not store the value.

INTEGER

The INTEGER data type stores whole numbers that range from $-2,147,483,647$ to $2,147,483,647$, for 9 or 10 digits of precision. The number $2,147,483,648$ is a reserved value and cannot be used. The INTEGER value is stored as a signed binary integer and is typically used to store counts, quantities, and so on.

Arithmetic operations and sort comparisons are performed more efficiently on integer data than on float or decimal data. INTEGER columns, however, cannot store absolute values beyond $(2^{31}-1)$. If a data value lies outside the numeric range of INTEGER, the database server does not store the value.

INTEGER data types require 4 bytes of storage per value.

INTERVAL

The INTERVAL data type stores a value that represents a span of time. INTERVAL types are divided into two classes: *year-month intervals* and *day-time intervals*. A year-month interval can represent a span of years and months, and a day-time interval can represent a span of days, hours, minutes, seconds, and fractions of a second.

An INTERVAL value is always composed of one value or a series of values that represents time units. Within a data-definition statement such as CREATE

TABLE or ALTER TABLE that defines the precision of an INTERVAL data type, the qualifiers must have the following format:

INTERVAL *largest_qualifier*(*n*) TO *smallest_qualifier*

Here the *largest_qualifier* and *smallest_qualifier* keywords are taken from one of the two INTERVAL classes, as shown in Table 2-5 on page 2-20.

If SECOND (or a larger time unit) is the *largest_qualifier*, the declaration of an INTERVAL data type can optionally specify *n*, the precision of the largest time unit (for *n* ranging from 1 to 9); this is not a feature of DATETIME data types.

If *smallest_qualifier* is FRACTION, you can also specify a scale in the range from 1 to 5. For FRACTION TO FRACTION qualifiers, the upper limit of *n* is 5, rather than 9. There are two incommensurable classes of INTERVAL data types:

- Those with a *smallest_qualifier* larger than DAY
- Those with a *largest_qualifier* smaller than MONTH

Table 2-5. Interval Classes

Interval Class	Time Units	Valid Entry
YEAR-MONTH INTERVAL	YEAR	A number of years
	MONTH	A number of months
DAY-TIME INTERVAL	DAY	A number of days
	HOURL	A number of hours
	MINUTE	A number of minutes
	SECOND	A number of seconds
	FRACTION	A decimal fraction of a second, with up to 5 digits. The default scale is 3 digits (thousandth of a second). To specify a non-default scale, write FRACTION(<i>n</i>), where $1 \leq n \leq 5$.

As with DATETIME data types, you can define an INTERVAL to include only the subset of time units that you need. But because the construct of “month” (as used in calendar dates) is not a time unit that has a fixed number of days, a single INTERVAL value cannot combine months and days; arithmetic that involves operands of the two different INTERVAL classes is not supported.

A value entered into an INTERVAL column need not include the full range of time units that were specified in the data-type declaration of the column. For example, you can enter a value of HOUR TO SECOND precision into a column defined as DAY TO SECOND. A value must always consist, however,

of contiguous time units. In the previous example, you cannot enter only the HOUR and SECOND values; you must also include MINUTE values.

A valid INTERVAL literal contains the INTERVAL keyword, the values to be entered, and the field qualifiers. (See the discussion of literal intervals in the *IBM Informix: Guide to SQL Syntax*.) When a value contains only one field, the largest and smallest fields are the same.

When you enter a value in an INTERVAL column, you must specify the largest and smallest fields in the value, just as you do for DATETIME values. In addition, you can optionally specify the precision of the first field (and the scale of the last field if it is a FRACTION). If the largest and smallest field qualifiers are both FRACTION, you can specify only the scale in the last field.

Acceptable qualifiers for the largest and smallest fields are identical to the list of INTERVAL fields that Table 2-5 on page 2-20 displays.

If you use the DB–Access **TABLE** menu, but you specify no INTERVAL field qualifiers, then a default INTERVAL qualifier, YEAR TO YEAR, is assigned.

The *largest_qualifier* in an INTERVAL value can be up to nine digits (except for FRACTION, which cannot be more than five digits), but if the value that you want to enter is greater than the default number of digits allowed for that field, you must explicitly identify the number of significant digits in the value that you enter. For example, to define an INTERVAL of DAY TO HOUR that can store up to 999 days, you could specify it the following way:

```
INTERVAL DAY(3) TO HOUR
```

INTERVAL literals use the same delimiters as DATETIME literals (except that MONTH and DAY time units are not valid within the same INTERVAL value). Table 2-6 shows the INTERVAL delimiters.

Table 2-6. INTERVAL Delimiters

Delimiter	Placement in an INTERVAL Literal
Hyphen	Between the YEAR and MONTH portions of the value
Blank space	Between the DAY and HOUR portions of the value
Colon	Between the HOUR, MINUTE, and SECOND portions of the value
Decimal point	Between the SECOND and FRACTION portions of the value

You can also enter INTERVAL values as character strings. The character string must include information for the same time units that were specified in the data-type declaration for the column. The INSERT statement in the following example shows an INTERVAL value entered as a character string:

```
INSERT INTO manufact (manu_code, manu_name, lead_time)
VALUES ('BR0', 'Ball-Racquet Originals', '160')
```

Because the **lead_time** column is defined as INTERVAL DAY(3) TO DAY, this INTERVAL value requires only one field, the span of days required for lead time. If the character string does not contain information for all fields (or adds additional fields), the database server returns an error. For additional information on entering INTERVAL values as character strings, see the *IBM Informix: Guide to SQL Syntax*.

By default, all fields of an INTERVAL column are two-digit numbers, except for the year and fraction fields. The year field is stored as four digits. The fraction field requires n digits where $1 \leq n \leq 5$, rounded up to an even number. You can use the following formula (rounded up to a whole number of bytes) to calculate the number of bytes required for an INTERVAL value: $(total\ number\ of\ digits\ for\ all\ fields) / 2 + 1$

For example, INTERVAL YEAR TO MONTH requires six digits (four for *year* and two for *month*), and requires 4, or $(6/2) + 1$, bytes of storage.

For information on using INTERVAL data in arithmetic and relational operations, see “Manipulating DATE with DATETIME and INTERVAL Values” on page 2-43. For information on using INTERVAL as a constant expression, see the description of the INTERVAL Field Qualifier in the *IBM Informix: Guide to SQL Syntax*.

LIST(e) (IDS)

The LIST data type is a collection type that stores ordered, non-unique elements; that is, it allows duplicate element values. The elements of a LIST have ordinal positions; that is, the list has a first element, a second element, and so on. (For a collection type with no ordinal positions, see “MULTISET(e) (IDS)” on page 2-25 and “SET(e) (IDS)” on page 2-32.)

No more than 97 columns of the same table can be declared as LIST data types. (The same restriction applies to SET and MULTISET collection types.)

By default, the database server inserts LIST elements at the end of the list. To support the ordinal position of a LIST, the INSERT statement provides the AT clause. This clause allows you to specify the position at which you want to insert a list-element value. For more information, see the INSERT statement in the *IBM Informix: Guide to SQL Syntax*.

All elements in a LIST have the same element type. To specify the element type, use the following syntax:

```
LIST(element_type NOT NULL)
```

The *element_type* of a LIST can be any of the following data types:

- A built-in type, except SERIAL, SERIAL8, BYTE, and TEXT
- A distinct type
- An unnamed or named row type
- Another collection type
- An opaque type

You must specify the NOT NULL constraint for LIST elements. No other constraints are valid for LIST columns. For more information on the syntax of the LIST data type, see the *IBM Informix: Guide to SQL Syntax*.

You can use LIST where any other data type is valid. For example:

- After the IN predicate in the WHERE clause of a SELECT statement to search for matching LIST values
- As an argument to the CARDINALITY or `mi_collection_card()` function to determine the number of elements in a LIST column

You *cannot* use LIST values as arguments to an aggregate function such as AVG, MAX, MIN, or SUM.

Two list values are equal if they have the same elements in the same order. The following examples both are list values but are not equal:

```
LIST{"blue", "green", "yellow"}  
LIST{"yellow", "blue", "green"}
```

The above statements are not equal because the values are not in the same order. To be equal, the second statement would have to be:

```
LIST{"blue", "green", "yellow"}
```

LVARCHAR(m) (IDS)

You can use the LVARCHAR data type to create a column for storing variable-length character strings whose upper limit (*m*) can be up to 32,739 bytes. (You can use the VARCHAR data type for strings no longer than 255 bytes.)

By default, the database server interprets quoted strings as LVARCHAR types. It also uses LVARCHAR for input and output casts for opaque data types.

The LVARCHAR data type stores opaque data types in the string (external) format. Each opaque type has an input support function and cast, which convert it from LVARCHAR to a form that database servers can manipulate. Each opaque type also has an output support function and cast, which convert it from its internal representation to LVARCHAR.

Important: When LVARCHAR is declared (with no size specification) as the data type of a column in a database table, the default maximum size is 2 kilobytes (2048 bytes), but you can specify an explicit maximum length of up to 32,739 bytes. When LVARCHAR is used in I/O operations on an opaque data type, however, the maximum size is limited only by the operating system.

LVARCHAR is implemented as a built-in opaque UDT. Only a subset of the string operations on CHAR and VARCHAR values are valid for LVARCHAR, and like other opaque types, LVARCHAR columns of remote tables are not accessible in distributed queries. For more information about LVARCHAR, see *IBM Informix: User-Defined Routines and Data Types Developer's Guide*.

MONEY(p,s)

The MONEY data type stores currency amounts. Like the DECIMAL(*p,s*) data type, MONEY can store fixed-point numbers up to a maximum of 32 significant digits, where *p* is the total number of significant digits (the precision) and *s* is the number of digits to the right of the decimal point (the scale).

Unlike the DECIMAL data type, the MONEY data type is always treated as a fixed-point decimal number. The database server defines the data type MONEY(*p*) as DECIMAL(*p*,2). If the precision and scale are not specified, the database server defines a MONEY column as DECIMAL(16,2).

You can use the following formula (rounded down to a whole number of bytes) to calculate the byte storage for a MONEY data type:

If the *scale* is odd: $N = (\textit{precision} + 4) / 2$

If the *scale* is even: $N = (\textit{precision} + 3) / 2$

For example, a MONEY data type with a precision of 16 and a scale of 2 (MONEY(16,2)) requires 10 or $(16 + 3)/2$, bytes of storage.

In the default locale, client applications format values from MONEY columns with the following currency notation:

- A currency symbol: a dollar sign (\$) at the front of the value
- A thousands separator: a comma (,) that separates every three digits in the integer part of the value
- A decimal point: a period (.) between the integer and fractional parts of the value

To change the format for MONEY values, change the **DBMONEY** environment variable. For valid **DBMONEY** settings, see “DBMONEY” on page 3-30.

The default value that the database server uses for scale is locale-dependent. The default locale specifies a default scale of two. For non-default locales, if the scale is omitted from the declaration, the database server creates MONEY values with a locale-specific scale.

The currency notation that client applications use is locale-dependent. If you specify a nondefault locale, the client uses a culture-specific format for MONEY values that might differ from the default U.S. English format in the leading (or trailing) currency symbol, thousands separator, and decimal separator, depending on what the locale files specify. For more information on locale dependency, see the *IBM Informix: GLS User's Guide*.

MULTISET(e) (IDS)

The MULTISET data type is a collection type that stores a non-ordered set that can include duplicate element values. The elements in a MULTISET have no ordinal position. That is, there is no concept of a first, second, or third element in a MULTISET. (For a collection type with ordinal positions for elements, see the LIST data type on page 2-22.)

All elements in a MULTISET have the same element type. To specify the element type, use the following syntax:

```
MULTISET(element_type NOT NULL)
```

The *element_type* of a collection can be any of the following types:

- Any built-in type, except SERIAL, SERIAL8, BYTE, and TEXT
- An unnamed or a named ROW type
- Another collection type or opaque type

You can use MULTISET anywhere that you use any other data type, unless otherwise indicated. For example:

- After the IN predicate in the WHERE clause of a SELECT statement to search for matching MULTISET values
- As an argument to the CARDINALITY or **mi_collection_card()** function to determine the number of elements in a MULTISET column

You *cannot* use MULTISET values as arguments to an aggregate function such as AVG, MAX, MIN, or SUM.

You must specify the NOT NULL constraint for MULTISET elements. No other constraints are valid for MULTISET columns. For more information on the MULTISET collection type, see the *IBM Informix: Guide to SQL Syntax*.

Two multiset data values are equal if they have the same elements, even if the elements are in different positions within the set. The following examples are both multiset values but are not equal:

```
MULTISET {"blue", "green", "yellow"}  
MULTISET {"blue", "green", "yellow", "blue"}
```

The following multiset values are equal:

```
MULTISET {"blue", "green", "blue", "yellow"}  
MULTISET {"blue", "green", "yellow", "blue"}
```

No more than 97 columns of the same table can be declared as **MULTISET** data types. (The same restriction applies to **SET** and **LIST** collection types.)

Named **ROW**

See “**ROW**, Named (IDS)” on page 2-27.

NCHAR(n)

The **NCHAR** data type stores fixed-length character data. The data can be a string of single-byte or multibyte letters, digits, and other symbols that are supported by the code set of the database locale. The main difference between **CHAR** and **NCHAR** data types is the collating order.

The collation order of the **CHAR** data type follows the code-set order, but the collating order of the **NCHAR** data type can be a localized order, if **DB_LOCALE** (or **SET COLLATION**) specifies a localized collation. For more information about **NCHAR**, see the description of “**DBNLS (IDS)**” on page 3-32.

NUMERIC(p,s)

The **NUMERIC** data type is a synonym for fixed-point **DECIMAL**.

NVARCHAR(m,r)

The **NVARCHAR** data type stores strings of varying lengths. The string can include digits, symbols, and single-byte and (in some locales) multibyte characters. The main difference between **VARCHAR** and **NVARCHAR** data types is the collation order. Collation of **VARCHAR** data follows code-set order, but **NVARCHAR** collation can be locale specific, if **DB_LOCALE** (or **SET COLLATION**) has specified a localized collation. (The section “Collating **VARCHAR** Values” on page 2-37 describes an exception.)

A column declared as **NVARCHAR**, without parentheses or parameters, has a maximum size of one byte, and a reserved size of zero.

No more than 195 columns of the same table can be **NVARCHAR** data types.

Opaque (IDS)

An opaque type is a data type for which you must provide the following information to the database server:

- A data structure for how the data values are stored on disk

- Support functions to determine how to convert between the disk storage format and the user format for data entry and display
- Secondary access methods that determine how the index on this data type is built, used, and manipulated
- User functions that use the data type
- A system catalog entry to register the opaque type in the database

The internal structure of an opaque type is not visible to the database server and can only be accessed through user-defined routines. Definitions for opaque types are stored in the **sysxdtypes** system catalog table. These SQL statements maintain the definitions of opaque types in the database:

- The CREATE OPAQUE TYPE statement registers a new opaque type in the database.
- The DROP TYPE statement removes a previously defined opaque type from the database.

For more information on the above-mentioned SQL statements, see the *IBM Informix: Guide to SQL Syntax*. For information on how to create opaque types and an example of an opaque type, see *IBM Informix: User-Defined Routines and Data Types Developer's Guide*.

REAL

The REAL data type is a synonym for SMALLFLOAT.

ROW, Named (IDS)

A named ROW type is declared by its name. That identifier must be unique within the schema. An unnamed ROW type is a ROW type that contains fields but has no user-defined name. Only named ROW types support data type inheritance. For more information, see “ROW Data Types” on page 2-48.

Defining Named ROW Types

You must declare and register in the database a new named ROW type by using the CREATE ROW TYPE statement of SQL. Definitions for named ROW types are stored in the **sysxdtypes** system catalog table.

The fields of a ROW data type can be any built-in data type or UDT, but TEXT or BYTE fields of a ROW type are valid in typed tables only. If you want to assign a ROW type to a column, its elements cannot be TEXT or BYTE data types.

In general, the data type of a field of a ROW type can be any of these types:

- A built-in type (except for the TEXT or BYTE data types)
- A collection type (LIST, MULTISSET, or SET)
- A distinct type

- Another named or unnamed ROW type
- An opaque type

These SQL statements maintain the definitions of named ROW data types:

- The CREATE ROW TYPE statement adds a named ROW type to the database.
- The DROP ROW TYPE statement removes a previously defined named ROW type from the database.

No more than 195 columns of the same table can be named ROW types.

For details about these SQL syntax statements, see the *IBM Informix: Guide to SQL Syntax*. For examples of how to create and use named ROW types, see the *IBM Informix: Database Design and Implementation Guide*.

Equivalence and Named ROW Types

No two named ROW types can be equal, even if they have identical structures, because they have different names. For example, the following named ROW types have the same structure (the same number of fields and the same order of data types of fields within the row) but are not equal:

```
name_t (lname CHAR(15), initial CHAR(1), fname CHAR(15))
emp_t (lname CHAR(15), initial CHAR(1), fname CHAR(15))
```

Named ROW Types and Inheritance

Named ROW types can be part of a type-inheritance hierarchy. One named ROW type can be the parent (or supertype) of another named ROW type. A subtype in a hierarchy inherits all the properties of its supertype. Type inheritance is discussed in the CREATE ROW TYPE statement in the *IBM Informix: Guide to SQL Syntax* and in the *IBM Informix: Database Design and Implementation Guide*.

Typed Tables

Tables that are part of an inheritance hierarchy must be typed tables. Typed tables are tables that have been assigned a named ROW type. For the syntax you use to create typed tables, see the CREATE TABLE statement in the *IBM Informix: Guide to SQL Syntax*. Table inheritance and its relation to type inheritance is also discussed in that section. For information about how to create and use typed tables, see the *IBM Informix: Database Design and Implementation Guide*.

ROW, Unnamed (IDS)

An unnamed ROW type contains fields but has no user-declared name. An unnamed ROW type is defined by its structure. Two unnamed ROW types are equal if they have the same structure (meaning the ordered list of the data types of the fields). If two unnamed ROW types have the same number of

fields, and if the order of the data type of each field in one ROW type matches the order of data types of the corresponding fields in the other ROW data type, then the two unnamed ROW data types are equal.

For example, the following unnamed ROW types are equal:

```
ROW (lname char(15), initial char(1) fname char(15))
ROW (dept char(15), rating char(1) name char(15))
```

The following ROW types have the same number of fields and the same data types, but are not equal, because their fields are not in the same order:

```
ROW (x integer, y varchar(20), z real)
ROW (x integer, z real, y varchar(20))
```

A field of an unnamed ROW type can be any of the following data types:

- A built-in type
- A collection type
- A distinct type
- Another ROW type
- An opaque type

Unnamed ROW types cannot be used in typed tables or in type inheritance hierarchies. For more information on unnamed ROW types, see the *IBM Informix: Guide to SQL Syntax* and the *IBM Informix: Database Design and Implementation Guide*.

Creating Unnamed ROW Types

You can create an unnamed ROW type in several ways:

- You can declare an unnamed ROW type using the ROW keyword. Each field in a ROW can have a different field type. To specify the field type, use the following syntax:

```
ROW(field_name field_type, ...)
```

The *field_name* must conform to the rules for SQL identifiers. (See the Identifier section in the *IBM Informix: Guide to SQL Syntax*.)

- To generate an unnamed ROW type, use the ROW keyword as a constructor with a series of values. A corresponding unnamed ROW type is created, using the default data types of the specified values.

For example, the following declaration:

```
ROW(1, 'abc', 5.30)
```

defines this unnamed ROW data type:

```
ROW (x INTEGER, y VARCHAR, z DECIMAL)
```

- You can create an unnamed ROW type by an implicit or explicit cast from a named ROW type or from another unnamed ROW type.

- The rows of any table (except a table defined on a named ROW type) are unnamed ROW types.

No more than 195 columns of the same table can be unnamed ROW types.

Inserting Values into Unnamed ROW Type Columns

When you specify field values for an unnamed ROW type, list the field values after the constructor and between parentheses. For example, suppose you have an unnamed ROW-type column. The following INSERT statement adds one group of field values to this ROW column:

```
INSERT INTO table1 VALUES (ROW(4, 'abc'))
```

You can specify a ROW column in the IN predicate in the WHERE clause of a SELECT statement to search for matching ROW values. For more information, see the Condition section in the *IBM Informix: Guide to SQL Syntax*.

SERIAL(n)

The SERIAL data type stores a sequential integer, in the positive range of the INT8 data type, that is automatically assigned by the database server when a new row is inserted. A table can have no more than one SERIAL column, but it can have one SERIAL and one SERIAL8 column.

SERIAL values in a column are not automatically unique. You must apply a unique index or primary key constraint to this column to prevent duplicate serial numbers. If you use the interactive schema editor in DB–Access to define the table, a unique index is applied automatically to a SERIAL column.

SERIAL numbers might not be consecutive, because of concurrent users, rollbacks, and other factors.

The DEFINE *variable* LIKE *column* syntax of SPL for indirect typing declares a variable of the INTEGER data type if *column* is a SERIAL data type.

The default serial starting number is 1, but you can assign a non-default initial value, *n*, when you create or alter the table. Any number greater than 0 can be your starting number. The maximum SERIAL is 2,147,483,647. If you assign a number greater than 2,147,483,647, you receive a syntax error. (Use the SERIAL8 data type, rather than SERIAL, if you need a larger range.)

After a nonzero number is assigned, it cannot be changed. You can insert a value into a SERIAL column (using the INSERT statement) or reset a serial column (using the ALTER TABLE statement), if the new value does not duplicate any existing value in the column. To insert into a SERIAL column, your database server increments by one the previous value (or the reset value, if that is larger) and assigns the result as the entered value. If ALTER TABLE

has reset the next value of a SERIAL column to a value smaller than values already in that column, however, the next value follows this formula:

(maximum existing value in SERIAL column) + 1

For example, if you reset the serial value of **customer.customer_num** to 50, when the largest existing value is 128, the next assigned number will be 129. For more details on SERIAL data entry, see the *IBM Informix: Guide to SQL Syntax*.

A SERIAL column can store unique codes (for example, order, invoice, or customer numbers). SERIAL data values require four bytes of storage, and have the same precision as the INTEGER data type. For details of another way to assign unique whole numbers to each row of a database table, see the CREATE SEQUENCE statement in *IBM Informix: Guide to SQL Syntax*.

SERIAL8(n)

The SERIAL8 data type stores a sequential integer, in the positive range of the INT8 data type, that is assigned automatically by the database server when a new row is inserted. It behaves like the SERIAL data type, but with a larger range. (For more information on how to insert values into SERIAL8 columns, see the *IBM Informix: Guide to SQL Syntax*.)

A SERIAL8 data column is commonly used to store large, unique numeric codes (for example, order, invoice, or customer numbers). SERIAL8 data values have the same precision and storage requirements as INT8 values (page 2-19). The following restrictions apply to SERIAL8 columns:

- You can define only one SERIAL8 column in a table.
A table, however, can have one SERIAL8 and one SERIAL column.
- SERIAL8 column values are not automatically unique.
You must apply a unique index or primary key constraint to this column to prevent duplicate SERIAL8 numbers.
- The SERIAL8 data type does not allow a negative, zero, or NULL value.

The DEFINE *variable* LIKE *column* syntax of SPL for indirect typing declares a variable of the INT8 data type if *column* is a SERIAL8 data type.

Assigning a Starting Value for SERIAL8

The default serial starting number is 1, but you can assign an initial value, *n*, when you create or alter the table. To start the values at 1 in a SERIAL8 column of a table, give the value 0 for the SERIAL8 column when you insert rows into that table. The database server will assign the value 1 to the SERIAL8 column of the first row of the table. The largest SERIAL8 value that you can assign is $2^{63}-1$ (9,223,372,036,854,775,807). If you assign a value greater

than this, you receive a syntax error. When the database server generates a SERIAL8 value of this maximum number, it wraps around and starts generating values beginning at 1.

After a nonzero SERIAL8 number is assigned, it cannot be changed. You can, however, insert a value into a SERIAL8 column (using the INSERT statement) or reset the SERIAL8 value *n* (using the ALTER TABLE statement), as long as that value does not duplicate any existing values in the column.

When you insert a number into a SERIAL8 column or reset the next value of a SERIAL8 column, your database server assigns the next number in sequence to the number entered. If you reset the next value of a SERIAL8 column to a value that is less than the values already in that column, however, the next value is computed using the following formula:

maximum existing value in SERIAL8 column + 1

For example, if you reset the SERIAL8 value of the **customer_num** column in the **customer** table to 50, when the highest-assigned customer number is 128, the next customer number assigned is 129.

Using SERIAL8 with INT8

All the arithmetic operators that are valid for INT8 (such as +, -, *, and /) and all the SQL functions that are valid for INT8 (such as ABS, MOD, POW, and so on) are also valid for SERIAL8 values. Data conversion rules that apply to INT8 also apply to SERIAL8, but with a NOT NULL constraint on SERIAL8.

The value of a SERIAL8 column of one table can be stored in an INT8 column of another table. In the second table, however, the INT8 values are not subject to the constraints on the original SERIAL8 column.

SET(e) (IDS)

The SET data type is an unordered collection type that stores unique elements; duplicate element values are not valid. (For a collection type that supports duplicate values, see the description of MULTISET in “MULTISET(e) (IDS)” on page 2-25.)

No more than 97 columns of the same table can be declared as SET data types. (The same restriction also applies to MULTISET and LIST collection types.)

The elements in a SET have no ordinal position. That is, no construct of a first, second, or third element in a SET exists. (For a collection type with ordinal positions for elements, see “LIST(e) (IDS)” on page 2-22.) All elements in a SET have the same element type. To specify the element type, use this syntax:

```
SET(element_type NOT NULL)
```


The *element_type* of a collection can be any of the following types:

- A built-in type, except SERIAL, SERIAL8, BYTE, and TEXT
- A named or unnamed ROW type
- Another collection type
- An opaque type

You must specify the NOT NULL constraint for SET elements. No other constraints are valid for SET columns. For more information on the syntax of the SET collection type, see the *IBM Informix: Guide to SQL Syntax*.

You can use SET anywhere that you use any other data type, unless otherwise indicated. For example:

- After the IN predicate in the WHERE clause of a SELECT statement to search for matching SET values
- As an argument to the CARDINALITY or **mi_collection_card()** function to determine the number of elements in a SET column

SET values are not valid as arguments to an aggregate function such as AVG, MAX, MIN, or SUM. For more information, see the Condition and Expression sections in the *IBM Informix: Guide to SQL Syntax*.

The following examples declare two sets. The first statement declares a set of integers and the second declares a set of character elements.

```
SET(INTEGER NOT NULL)
SET(CHAR(20) NOT NULL)
```

The following examples construct the same sets from value lists:

```
SET{1, 5, 13}
SET{"Oakland", "Menlo Park", "Portland", "Lenexa"}
```

In the following example, a SET constructor function is part of a CREATE TABLE statement:

```
CREATE TABLE tab
(
  c CHAR(5),
  s SET(INTEGER NOT NULL)
);
```

The following set values are equal:

```
SET{"blue", "green", "yellow"}
SET{"yellow", "blue", "green"}
```

SMALLFLOAT

The SMALLFLOAT data type stores single-precision floating-point numbers with approximately nine significant digits. SMALLFLOAT corresponds to the

float data type in C. The range of values for a SMALLFLOAT data type is the same as the range of values for the C **float** data type on your computer.

A SMALLFLOAT data type column typically stores scientific numbers that can be calculated only approximately. Because floating-point numbers retain only their most significant digits, the number that you enter in this type of column and the number the database displays might differ slightly depending on how your computer stores floating-point numbers internally.

For example, you might enter a value of 1.1000001 in a SMALLFLOAT field and, after processing the SQL statement, the application might display this value as 1.1. This difference occurs when a value has more digits than the floating-point number can store. In this case, the value is stored in its approximate form with the least significant digits treated as zeros.

SMALLFLOAT data types usually require 4 bytes of storage. Conversion of a SMALLFLOAT value to a DECIMAL value results in 9 digits of precision.

SMALLINT

The SMALLINT data type stores small whole numbers that range from $-32,767$ to $32,767$. The maximum negative number, $-32,768$, is a reserved value and cannot be used. The SMALLINT value is stored as a signed binary integer.

Integer columns typically store counts, quantities, and so on. Because the SMALLINT data type requires only two bytes per value, arithmetic operations are performed efficiently. SMALLINT, however, stores only a limited range of values, compared to other built-in numeric data types. If a number is outside the range of the minimum and maximum SMALLINT values, the database server does not store the data value, but instead issues an error message.

TEXT

The TEXT data type stores any kind of text data. It can contain both single-byte and multibyte characters that the locale supports. The term *simple large object* refers to the TEXT and BYTE data types.

A TEXT column has a theoretical limit of 2^{31} bytes (two gigabytes) and a practical limit that your available disk storage determines.

Important: An error results if you try to return a TEXT column from a subquery, even if no TEXT column is used in a comparison condition or with the IN predicate.

No more than 195 columns of the same table can be declared as TEXT data types. (The same restriction also applies to BYTE data types.)

You can store, retrieve, update, or delete the values in a TEXT column. You cannot, however, use TEXT operands in arithmetic or string expressions, nor can you assign literals to TEXT columns in the SET clause of the UPDATE statement. You also cannot use TEXT values in any of the following ways:

- With aggregate functions
- With the IN clause
- With the MATCHES or LIKE clauses
- With the GROUP BY clause
- With the ORDER BY clause

You can use TEXT operands in Boolean expressions only when you are testing for NULL values with the IS NULL or IS NOT NULL operators.

You can insert data into TEXT columns in the following ways:

- With the **dbload** or **onload** utilities
- With the LOAD statement (DB–Access)
- From TEXT host variables (ESQL)

You cannot use a quoted text string, number, or any other actual value to insert or update TEXT columns.

When you select a TEXT column, you can choose to receive all or part of it. To retrieve it all, use the regular syntax for selecting a column. You can also select any part of a TEXT column by using subscripts, as this example shows:

```
SELECT cat_descr [1,75] FROM catalog WHERE catalog_num = 10001
```

This statement reads the first 75 bytes of the **cat_descr** column associated with the **catalog_num** value 10001.

A built-in cast exists to convert TEXT objects to CLOB objects. For more information, see the *IBM Informix: Database Design and Implementation Guide*.

Strings of the TEXT data type are collated in code-set order. For more information on collating orders, see the *IBM Informix: GLS User's Guide*.

Nonprintable Characters in TEXT Values

TEXT columns typically store documents, program source files, and so on. In the default U.S. English locale, data objects of type TEXT can contain a combination of printable ASCII characters and the following control characters:

- Tab (CTRL-I)
- New line (CTRL-J)
- New page (CTRL-L)

Both printable and nonprintable characters can be inserted in text columns. IBM Informix products do not do any checking of data values that are inserted in a column of the TEXT data type. (Applications may have difficulty, however, in displaying TEXT values that include non-printable characters.) For detailed information on entering and displaying nonprintable characters, refer to “Nonprintable Characters with CHAR” on page 2-9.

Unnamed ROW

See “ROW, Unnamed (IDS)” on page 2-28.

VARCHAR(*m,r*)

The VARCHAR data type stores character strings of varying length that contain single-byte and (if the locale supports them) multibyte characters, where *m* is the maximum size (in bytes) of the column and *r* is the minimum number of bytes reserved for that column. A column declared as VARCHAR without parentheses or parameters has a maximum size of one byte, and a reserved size of zero.

The VARCHAR data type is the Informix implementation of a character varying data type. The ANSI standard data type for varying-length character strings is CHARACTER VARYING.

The size of the maximum size (*m*) parameter of a VARCHAR column can range from 1 to 255 bytes. If you are placing an index on a VARCHAR column, the maximum size is 254 bytes. You can store character strings that are shorter, but not longer, than the *m* value that you specify.

Specifying the minimum reserved space (*r*) parameter is optional. This value can range from 0 to 255 bytes but must be less than the maximum size (*m*) of the VARCHAR column. If you do not specify any minimum value, it defaults to 0. You should specify this parameter when you initially intend to insert rows with short or NULL character strings in the column but later expect the data to be updated with longer values.

For variable-length strings longer than 255 bytes, you can use the LVARCHAR data type, whose upper limit is 32,739 bytes, instead of VARCHAR. Because LVARCHAR is implemented as a built-in opaque data type, however, you cannot access LVARCHAR columns in distributed queries of remote tables.

In an index based on a VARCHAR column (or on a NVARCHAR column), each index key has a length that is based on the data values that are actually entered, rather than on the declared maximum size of the column. (See, however, “IFX_PAD_VARCHAR (IDS)” on page 3-53 for information on how you can configure the effective size of VARCHAR and NVARCHAR data strings that Dynamic Server sends or receives.)

When you store a string in an VARCHAR column, only the actual data characters are stored. The database server does not strip a VARCHAR string of any user-entered trailing blanks, nor pad a VARCHAR value to the declared length of the column. If you specify a reserved space (*r*), but some data strings are shorter than *r* bytes, some space reserved for rows goes unused.

VARCHAR values are compared to other VARCHAR values (and to other character-string data types) in the same way that CHAR values are compared. The shorter value is padded on the right with blank spaces until the values have equal lengths; then they are compared for the full length.

No more than 195 columns of the same table can be VARCHAR data types.

Nonprintable Characters with VARCHAR

Nonprintable VARCHAR characters are entered, displayed, and treated in the same way that nonprintable characters in CHAR values are treated. For details, see the section “Nonprintable Characters with CHAR” on page 2-9.

Storing Numeric Values in a VARCHAR Column

When you insert a numeric value in a VARCHAR column, the stored value does not get padded with trailing blanks to the maximum length of the column. The number of digits in a numeric VARCHAR value is the number of characters that you need to store that value. For example, in the next example, the value stored in table **mytab** is 1.

```
create table mytab (col1 varchar(10));
insert into mytab values (1);
```

Tip: VARCHAR treats C *null* (binary 0) and string terminators as termination characters for nonprintable characters.

Multibyte Characters with VARCHAR

In some East Asian locales, VARCHAR data types can store multibyte characters if the database locale supports a multibyte code set. If you store multibyte characters, make sure to calculate the number of bytes needed. For more information, see the *IBM Informix: GLS User's Guide*.

Collating VARCHAR Values

The main difference between the NVARCHAR and the VARCHAR data types (like the difference between CHAR and NCHAR) is the difference in collating order. In general, collation of VARCHAR (like CHAR and LVARCHAR) values is in the order of the characters as they appear in the code set.

An exception is the MATCHES operator, which applies a localized collation to NVARCHAR and VARCHAR values (as well as to CHAR, LVARCHAR, and NCHAR values) if you use bracket ([]) symbols to define ranges when

DB_LOCALE (or SET COLLATION) has specified a localized collating order. For more information, see the *IBM Informix: GLS User's Guide*.

Built-In Data Types

Informix database servers support the following built-in data types.

Category	Data Types
Character	CHAR, CHARACTER VARYING, LVARCHAR, NCHAR, NVARCHAR, VARCHAR
Numeric	DECIMAL, FLOAT, INT8, INTEGER, MONEY, SERIAL, SERIAL8, SMALLFLOAT, SMALLINT
Large-object	Simple-large-object types: BYTE, TEXT Smart-large-object types: BLOB, CLOB
Time	DATE, DATETIME, INTERVAL
Miscellaneous	BOOLEAN

Extended Parallel Server does not support BLOB, CLOB, or LVARCHAR. For a description of character, numeric, and miscellaneous data types, refer to the appropriate entry in “Description of Data Types” on page 2-6. Page references are in the alphabetical list in Table 2-1 on page 2-3.

Sections that follow provide additional information on large-object and time data types.

Large-Object Data Types

A large object is a data object that is logically stored in a table column but physically stored independent of the column. Large objects are stored separate from the table because they typically store a large amount of data. Separation of this data from the table can increase performance.

Figure 2-4 shows the large-object data types.

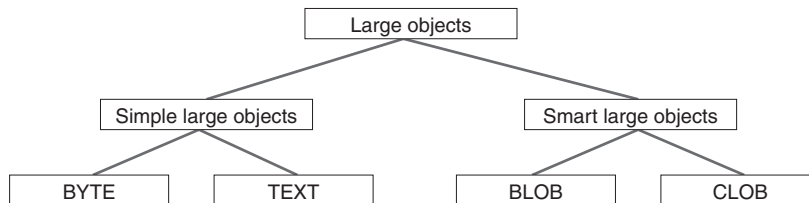


Figure 2-4. Large-Object Data Types

Only Dynamic Server supports BLOB and CLOB data types.

For the relative advantages and disadvantages of simple and smart large objects, see the *IBM Informix: Database Design and Implementation Guide*.

Simple Large Objects

Simple large objects are a category of large objects that have a theoretical size limit of 2^{31} bytes and a practical limit that your disk capacity determines.

Informix database servers support these simple-large-object data types:

- | | |
|------|--|
| BYTE | Stores binary data. For more detailed information about this data type, see the description on page 2-7. |
| TEXT | Stores text data. For more detailed information about this data type, see the description on page 2-34. |

No more than 195 columns of the same table can be declared as BYTE or TEXT data types. Unlike smart large objects, simple large objects do not support random access to the data. When you transfer a simple large object between a client application and the database server, you must transfer the entire BYTE or TEXT value. If the data cannot fit into memory, you must store the data value in an operating-system file and then retrieve it from that file.

The database server stores simple large objects in *blobspaces*. A *blobspace* is a logical storage area that contains one or more chunks that only store BYTE and TEXT data. For information on how to define blobspaces, see your *IBM Informix: Administrator's Guide*.

Smart Large Objects (IDS)

Smart large objects are a category of large objects that support random access to the data and are generally recoverable. The random access feature allows you to seek and read through the smart large object as if it were an operating-system file.

Smart large objects are also useful for opaque data types with large storage requirements. (See the description of opaque data types in “Opaque Data Types” on page 2-49.) They have a theoretical size limit of 2^{42} bytes and a practical limit that your disk capacity determines.

Dynamic Server supports the following smart-large-object data types:

- | | |
|------|---|
| BLOB | Stores binary data. For more information about this data type, see the description on page 2-6. |
| CLOB | Stores text data. For more information about this data type, see the description on page 2-10. |

Dynamic Server stores smart large objects in *sbspaces*. An *sbspace* is a logical storage area that contains one or more chunks that store only BLOB and CLOB data. For information on how to define *sbspaces*, see your *IBM Informix: Performance Guide*.

When you define a BLOB or CLOB column, you can determine the following large-object characteristics:

- LOG and NOLOG: whether the database server should log the smart large object in accordance with the current database log mode
- KEEP ACCESS TIME and NO KEEP ACCESS TIME: whether the database server should keep track of the last time the smart large object was accessed
- HIGH INTEG and MODERATE INTEG: whether the database server should use page headers to detect data corruption

Use of these characteristics can affect performance. For information, see your *IBM Informix: Performance Guide*.

When an SQL statement accesses a smart-large-object, the database server does not send the actual BLOB or CLOB data. Instead, it establishes a pointer to the data and returns this pointer. The client application can then use this pointer in open, read, or write operations on the smart large object.

To access a BLOB or CLOB column from within a client application, use one of the following application programming interfaces (APIs):

- From within an IBM Informix ESQL/C program, use the smart-large-object API. (For more information, see the *IBM Informix: ESQL/C Programmer's Manual*.)
- From within a DataBlade module, use the Client and Server API. (For more information, see the *IBM Informix: DataBlade API Programmer's Guide*.)

For information on smart large objects, see the *IBM Informix: Guide to SQL Syntax* and *IBM Informix: Database Design and Implementation Guide*.

Time Data Types

DATE and DATETIME data values represent zero-dimensional points in time; INTERVAL data values represent 1-dimensional spans of time, with positive or negative values. DATE precision is always an integer count of days, but various field qualifiers can define the DATETIME and INTERVAL precision. You can use DATE, DATETIME, and INTERVAL data in arithmetic and relational expressions. You can manipulate a DATETIME value with another DATETIME value, an INTERVAL value, the current time (specified by the keyword CURRENT), or some unit of time (using the keyword UNITS).

You can use a DATE value in most contexts where a DATETIME value is valid, and vice versa. You also can use an INTERVAL operand in arithmetic

operations where a DATETIME value is valid. In addition, you can add two INTERVAL values and multiply or divide an INTERVAL value by a number.

An INTERVAL column can hold a value that represents the difference between two DATETIME values or the difference between (or sum of) two INTERVAL values. In either case, the result is a span of time, which is an INTERVAL value. Conversely, if you add or subtract an INTERVAL from a DATETIME value, another DATETIME value is produced, because the result is a specific time.

Table 2-7 lists the binary arithmetic operations that you can perform on DATE, DATETIME, and INTERVAL operands, as well as the data type that is returned by the arithmetic expression.

Table 2-7. Arithmetic Operations on DATE, DATETIME, and INTERVAL Values

Operand 1	Operator	Operand 2	Result
DATE	-	DATETIME	INTERVAL
DATETIME	-	DATE	INTERVAL
DATE	+ or -	INTERVAL	DATETIME
DATETIME	-	DATETIME	INTERVAL
DATETIME	+ or -	INTERVAL	DATETIME
INTERVAL	+	DATETIME	DATETIME
INTERVAL	+ or -	INTERVAL	INTERVAL
DATETIME	-	CURRENT	INTERVAL
CURRENT	-	DATETIME	INTERVAL
INTERVAL	+	CURRENT	DATETIME
CURRENT	+ or -	INTERVAL	DATETIME
DATETIME	+ or -	UNITS	DATETIME
INTERVAL	+ or -	UNITS	INTERVAL
INTERVAL	* or /	NUMBER	INTERVAL

No other combinations are allowed. You cannot add two DATETIME values because this operation does not produce either a specific time or a span of time. For example, you cannot add December 25 and January 1, but you can subtract one from the other to find the time span between them.

Manipulating DATETIME Values

You can subtract most DATETIME values from each other. Dates can be in any order and the result is either a positive or a negative INTERVAL value.

The first DATETIME value determines the precision of the result, which includes the same time units as the first operand.

If the second DATETIME value has fewer fields than the first, the precision of the second operand is increased automatically to match the first.

In the following example, subtracting the DATETIME YEAR TO HOUR value from the DATETIME YEAR TO MINUTE value results in a positive interval value of 60 days, 1 hour, and 30 minutes. Because minutes were not included in the second operand, the database server sets the minutes value for the second operand to 0 before performing the subtraction.

```
DATETIME (2003-9-30 12:30) YEAR TO MINUTE  
- DATETIME (2003-8-1 11) YEAR TO HOUR
```

```
Result: INTERVAL (60 01:30) DAY TO MINUTE
```

If the second DATETIME operand has more fields than the first (regardless of whether the precision of the extra fields is larger or smaller than those in the first operand), the additional time unit fields in the second value are ignored in the calculation.

In the next expression (and its result), the year is not included for the second operand. Therefore, the year is set automatically to the current year (from the system clock-calendar), in this example 2005, and the resulting INTERVAL is negative, which indicates that the second date is later than the first.

```
DATETIME (2005-9-30) YEAR TO DAY  
- DATETIME (10-1) MONTH TO DAY
```

```
Result: INTERVAL (-1) DAY TO DAY [assuming that the current  
year is 2005]
```

Manipulating DATETIME with INTERVAL Values

INTERVAL values can be added to or subtracted from DATETIME values. In either case, the result is a DATETIME value. If you are adding an INTERVAL value to a DATETIME value, the order of values is unimportant; however, if you are subtracting, the DATETIME value must come first. Adding or subtracting a positive INTERVAL value simply moves the DATETIME result forward or backward in time. The expression shown in the following example moves the date ahead by three years and five months:

```
DATETIME (2000-8-1) YEAR TO DAY  
+ INTERVAL (3-5) YEAR TO MONTH
```

```
Result: DATETIME (2004-01-01) YEAR TO DAY
```

Important: Evaluate the logic of your addition or subtraction. Remember that months can have 28, 29, 30, or 31 days and that years can have 365 or 366 days.

In most situations, the database server automatically adjusts the calculation when the operands do not have the same precision. In certain contexts, however, you must explicitly adjust the precision of one value to perform the calculation. If the INTERVAL value you are adding or subtracting has fields that are not included in the DATETIME value, you must use the EXTEND function to increase the precision of the DATETIME value. (For more information on the EXTEND function, see the Expression segment in the *IBM Informix: Guide to SQL Syntax*.)

For example, you cannot subtract an INTERVAL MINUTE TO MINUTE value from the DATETIME value in the previous example that has a YEAR TO DAY field qualifier. You can, however, use the EXTEND function to perform this calculation, as the following example shows:

```
EXTEND (DATETIME (2003-8-1) YEAR TO DAY, YEAR TO MINUTE)
      - INTERVAL (720) MINUTE(3) TO MINUTE
```

Result: DATETIME (2003-07-31 12:00) YEAR TO MINUTE

The EXTEND function allows you to explicitly increase the DATETIME precision from YEAR TO DAY to YEAR TO MINUTE. This allows the database server to perform the calculation, with the resulting extended precision of YEAR TO MINUTE.

Manipulating DATE with DATETIME and INTERVAL Values

You can use DATE operands in some arithmetic expressions with DATETIME or INTERVAL operands by writing expressions to do the manipulating, as Table 2-8 shows.

Table 2-8. Results of Expressions That Manipulate DATE with DATETIME or INTERVAL Values

Expression	Result
DATE - DATETIME	INTERVAL
DATETIME - DATE	INTERVAL
DATE + or - INTERVAL	DATETIME

In the cases that Table 2-8 shows, DATE values are first converted to their corresponding DATETIME equivalents, and then the expression is evaluated by the rules of arithmetic.

Although you can interchange DATE and DATETIME values in many situations, you must indicate whether a value is a DATE or a DATETIME data type. A DATE value can come from the following sources:

- A column or program variable of type DATE
- The TODAY keyword

- The DATE() function
- The MDY function
- A DATE literal

A DATETIME value can come from the following sources:

- A column or program variable of type DATETIME
- The CURRENT keyword
- The EXTEND function
- A DATETIME literal

The database locale defines the default DATE and DATETIME formats. For the default locale, U.S. English, these formats are *'mm/dd/yy'* for DATE values and *'yyyy-mm-dd hh:MM:ss'* for DATETIME values.

To represent DATE and DATETIME values as character strings, the fields in the strings must be in proper order. In other words, when a DATE value is expected, the string must be in DATE format and when a DATETIME value is expected, the string must be in DATETIME format. For example, you can use the string 10/30/2003 as a DATE string but not as a DATETIME string. Instead, you must use 2003-10-30 or 03-10-30 as the DATETIME string.

In a nondefault locale, literal DATE and DATETIME strings must match the formats that the locale defines. For more information, see the *IBM Informix: GLS User's Guide*.

You can customize the DATE format that the database server expects with the **DBDATE** and **GL_DATE** environment variables. You can customize the DATETIME format that the database server expects with the **DBTIME** and **GL_DATETIME** environment variables. For more information, see “DBDATE” on page 3-25 and “DBTIME” on page 3-39. For more information on all these environment variables, see the *IBM Informix: GLS User's Guide*.

You can also subtract one DATE value from another DATE value, but the result is a positive or negative INTEGER count of days, rather than an INTERVAL value. If an INTERVAL value is required, you can either use the UNITS DAY operator to convert the INTEGER value into an INTERVAL DAY TO DAY value, or else use EXTEND to convert one of the DATE values into a DATETIME value before subtracting.

For example, the following expression uses the **DATE()** function to convert character string constants to DATE values, calculates their difference, and then uses the UNITS DAY keywords to convert the INTEGER result into an INTERVAL value:

```
(DATE ('5/2/1994') - DATE ('4/6/1955')) UNITS DAY
```

```
Result: INTERVAL (12810) DAY(5) TO DAY
```

Important: Because of the high precedence of UNITS relative to other SQL operators, you should generally enclose any arithmetic expression that is the operand of UNITS within parentheses, as in the preceding example.

If you need YEAR TO MONTH precision, you can use the EXTEND function on the first DATE operand, as the following example shows:

```
EXTEND (DATE ('5/2/1994'), YEAR TO MONTH) - DATE ('4/6/1955')
```

```
Result: INTERVAL (39-01) YEAR TO MONTH
```

The resulting INTERVAL precision is YEAR TO MONTH, because the DATETIME value came first. If the DATE value had come first, the resulting INTERVAL precision would have been DAY(5) TO DAY.

Manipulating INTERVAL Values

You can add or subtract INTERVAL values only if both values are from the same class; that is, if both are year-month or both are day-time. In the following example, a SECOND TO FRACTION value is subtracted from a MINUTE TO FRACTION value:

```
INTERVAL (100:30.0005) MINUTE(3) TO FRACTION(4)  
- INTERVAL (120.01) SECOND(3) TO FRACTION
```

```
Result: INTERVAL (98:29.9905) MINUTE TO FRACTION(4)
```

The use of numeric qualifiers alerts the database server that the MINUTE and FRACTION in the first value and the SECOND in the second value exceed the default number of digits.

When you add or subtract INTERVAL values, the second value cannot have a field with greater precision than the first. The second INTERVAL, however, can have a field of smaller precision than the first. For example, the second INTERVAL can be HOUR TO SECOND when the first is DAY TO HOUR. The additional fields (in this case MINUTE and SECOND) in the second INTERVAL value are ignored in the calculation.

Multiplying or Dividing INTERVAL Values

You can multiply or divide INTERVAL values by numbers. Any remainder from the calculation is ignored, however, and the result is truncated to the precision of the INTERVAL. The following expression multiplies an INTERVAL value by a literal number that has a fractional part:

```
INTERVAL (15:30.0002) MINUTE TO FRACTION(4) * 2.5
```

```
Result: INTERVAL (38:45.0005) MINUTE TO FRACTION(4)
```

In this example, $15 * 2.5 = 37.5$ minutes, $30 * 2.5 = 75$ seconds, and $2 * 2.5 = 5$ FRACTION (4). The 0.5 minute is converted into 30 seconds and 60 seconds are converted into 1 minute, which produces the final result of 38 minutes, 45 seconds, and 0.0005 of a second. The result of any calculation has the same precision as the original INTERVAL operand.

Extended Data Types (IDS)

Dynamic Server enables you to create *extended data types* to characterize data that cannot easily be represented with the built-in data types. (You cannot, however, use extended data types in distributed transactions that query external tables.) You can create these categories of extended data types:

- Complex data types
- Distinct data types
- Opaque data types

Sections that follow provide an overview of each of these data types.

For more information about extended data types, see the *IBM Informix: Database Design and Implementation Guide* and *IBM Informix: User-Defined Routines and Data Types Developer's Guide*.

Complex Data Types

A *complex data type* can store one or more values of other built-in and extended data types. Figure 2-5 shows the complex types that Dynamic Server supports.

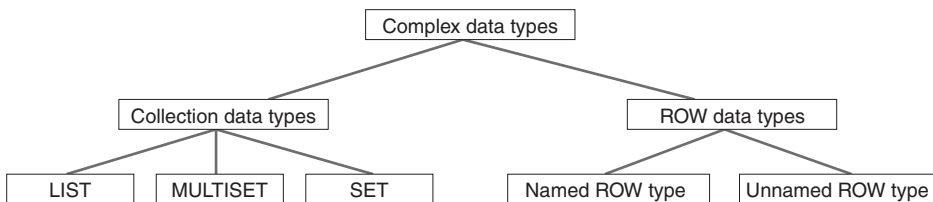


Figure 2-5. Complex Data Types of Dynamic Server

The following table summarizes the structure of the complex data types.

Data Type		Description
Collection types:		Complex data types that are made up of elements, each of which is of the same data type.
	LIST	A group of ordered elements, each of which need not be unique within the group.
	MULTISET	A group of elements, each of which need not be unique. The order of the elements is ignored.
	SET	A group of elements, each of which is unique. The order of the elements is ignored.
ROW types:		Complex data types that are made up of fields.
	Named ROW type	Row types that are identified by their name.
	Unnamed ROW type	Row types that are identified by their structure.

Complex data types can be nested. For example, you can construct a ROW type whose fields include one or more sets, multisets, ROW types, and lists. Likewise, a collection type can have elements whose data type is a ROW type or a collection type.

Complex types that include opaque types inherit the following support functions.

input	export	LO_handles
output	import_binary	hash
send	export_binary	lessthan
recv	assign	equal
import	destroy	lessthan (ROW only)

Sections that follow summarize the complex data types. For more information, see the *IBM Informix: Database Design and Implementation Guide*.

Collection Data Types

A collection data type is a complex type that is made up of one or more elements, all of the same data type. A collection element can be of any data type (including other complex types) except BYTE, TEXT, SERIAL, or SERIAL8.

Important: An element cannot have a *NULL* value. You must specify the *NOT NULL* constraint for collection elements. No other constraints are valid for collections.

Dynamic Server supports three kinds of built-in collection types: LIST, SET, and MULTISET. The keywords used to declare these collections are the names of the *type constructors* or just *constructors*. For the syntax of collection types, see the *IBM Informix: Guide to SQL Syntax*. No more than 97 columns of the same table can be declared as collection data types.

When you specify element values for a collection, list the element values after the constructor and between braces ({ }). For example, suppose you have a collection column with the following MULTISET data type:

```
CREATE TABLE table1
(
    mset_col MULTISET(INTEGER NOT NULL)
)
```

The next INSERT statement adds one group of element values to this column. (The word MULTISET in these two examples is the MULTISET constructor.)

```
INSERT INTO table1 VALUES (MULTISET{5, 9, 7, 5})
```

You can leave the braces empty to indicate an empty set:

```
INSERT INTO table1 VALUE (MULTISET{})
```

An empty collection is not equivalent to a NULL value for the column.

Accessing Collection Data: To access the elements of a collection column, you must fetch the collection into a collection variable and modify the contents of the collection variable. Collection variables can be either of the following types:

- Variables in an SPL routine

For more information, see the *IBM Informix: Guide to SQL Tutorial*.

- Host variables in an IBM Informix ESQL/C program

For more information, see the *IBM Informix: ESQL/C Programmer's Manual*.

You can also use nested dot notation to access collection data. For more about accessing elements of a collection, see the *IBM Informix: Guide to SQL Tutorial*.

Important: Collection data types are not valid as arguments to functions that are used for functional indexes.

ROW Data Types

A ROW data type is an ordered collection of one or more elements, called *fields*. Each field has a name and a data type. The fields of a ROW are comparable to the columns of a table, but with important differences:

- A field has no default clause.
- You cannot define constraints on a field.

- You can only use fields with row types, not with tables.

Two kinds of ROW data types exist:

- *Named ROW data types* are identified by their names.
- *Unnamed ROW data types* are identified by their structure.

The *structure* of an unnamed ROW data type is the number (and the order of data types) of its fields.

No more than 195 columns of the same table can be declared as ROW data types. For more information about ROW data types, see “ROW, Named (IDS)” on page 2-27 and “ROW, Unnamed (IDS)” on page 2-28.

You can cast between named and unnamed ROW data types; this is described in the *IBM Informix: Database Design and Implementation Guide*.

Distinct Data Types

A distinct data type has the same internal structure as some other source data type in the database. The source type can be a built-in or extended data type. What distinguishes a distinct type from its source type are support functions that are defined on the distinct type.

No more than 195 columns in the same table can be DISTINCT types that are based on BYTE, TEXT, ROW, LVARCHAR, NVARCHAR, or VARCHAR source types, and no more than 97 can have collection source types. For more information, see the section “Distinct (IDS)” on page 2-17. See also *IBM Informix: User-Defined Routines and Data Types Developer’s Guide*.

Opaque Data Types

An opaque data type is a user-defined data type that is fully encapsulated. That is, its internal structure is unknown to the database server. User-defined types (UDTs) that are not DISTINCT types whose source types are built-in types are opaque.

The built-in data types BLOB, BOOLEAN, CLOB, and LVARCHAR are implemented as opaque data types. You cannot access these *built-in opaque* data types in cross-server distributed operations, but you can access them in other databases of the same Dynamic Server instance.

For more information, see the section “Opaque (IDS)” on page 2-26. See also *IBM Informix: User-Defined Routines and Data Types Developer’s Guide*.

Data Type Casting and Conversion

Occasionally, the data type that was assigned to a column with the CREATE TABLE statement is inappropriate. You might wish to change the data type of a column when you need to store larger values than the current data type can accommodate. The database server allows you to change the data type of the column or to cast its values to a different data type with either of the following methods:

- Use the ALTER TABLE statement to modify the data type of a column.
For example, if you create a SMALLINT column and later find that you need to store integers larger than 32,767, you must change the data type of that column to store the larger value. You can use ALTER TABLE to change the data type to INTEGER. The conversion changes the data type of all values that currently exist in the column as well as any new values that might be added.
- Use the CAST AS keywords or the double colon (::) cast operator to cast a value to a different data type.

Casting does not permanently alter the data type of a value; it expresses the value in a more convenient form. Casting user-defined data types into built-in types allows client programs to manipulate data types without knowledge of their internal structure.

If you change data types, the new data type must be able to store all of the old value.

Both data-type conversion and casting depend on casts registered in the **syscasts** system catalog table. For information about **syscasts**, see “SYSCASTS (IDS)” on page 1-17.

A cast is either built-in or user defined. Guidelines exist for casting distinct and extended data types. For more information about casting opaque data types, see *IBM Informix: User-Defined Routines and Data Types Developer's Guide*. For information about casting other extended data types, see the *IBM Informix: Database Design and Implementation Guide*.

Using Built-in Casts

User **informix** owns built-in casts. They govern conversions from one built-in data type to another. Built-in casts allow the database server to attempt the following data-type conversions:

- A character type to any other character type
- A character type to or from another built-in type
- A numeric type to any other numeric type

The database server automatically invokes appropriate built-in casts when required. For time data types, conversion between DATE and DATETIME data

types requires explicit casts with the EXTEND function, and explicit casts with the UNITS operator are required for number-to-INTERVAL conversion. Built-in casts are not available for converting large (BYTE, BLOB, CLOB, and TEXT) built-in types to other built-in data types.

When you convert a column from one built-in data type to another, the database server applies the appropriate built-in casts to each value already in the column. If the new data type cannot store any of the resulting values, the ALTER TABLE statement fails.

For example, if you try to convert a column from the INTEGER data type to the SMALLINT data type and the following values exist in the INTEGER column, the database server does not change the data type, because SMALLINT columns cannot accommodate numbers greater than 32,767:

100	400	700	50000	700
-----	-----	-----	-------	-----

The same situation might occur if you attempt to transfer data from FLOAT or SMALLFLOAT columns to INTEGER, SMALLINT, or DECIMAL columns. Errors of overflow, underflow, or truncation can occur during data type conversion.

Sections that follow describe database server behavior during certain types of casts and conversions.

Converting from Number to Number

When you convert data from one number data type to another, you occasionally find rounding errors. The following table indicates which numeric data type conversions are acceptable and what kinds of errors you can encounter when you convert between certain numeric data types.

Target Type	SMALL INT	INTEGER	INT8	SMALL FLOAT	FLOAT	DECIMAL
SMALLINT	OK	OK	OK	OK	OK	OK
INTEGER	E	OK	OK	E	OK	P
INT8	E	E	OK	D	E	P
SMALLFLOAT	E	E	E	OK	OK	P
FLOAT	E	E	E	D	OK	P
DECIMAL	E	E	E	D	D	P

Legend:

OK No error

P An error can occur, depending on the precision of the decimal

E An error can occur, depending on the data value

D No error, but less significant digits might be lost

For example, if you convert a FLOAT value to DECIMAL(4,2), your database server rounds off the floating-point number before storing it as DECIMAL.

This conversion can result in an error depending on the precision assigned to the DECIMAL column.

Converting Between Number and Character

You can convert a character column (of a data type such as CHAR, NCHAR, NVARCHAR, or VARCHAR) to a numeric column. If a data string, however, contains any characters that are not valid in a number column (for example, the letter *l* instead of the number *1*), the database server returns an error.

You can also convert a numeric column to a character column. If the character column is not large enough to receive the number, however, the database server generates an error. If the database server generates an error, it cannot complete the ALTER TABLE statement or cast, and leaves the column values as characters. You receive an error message and the statement is rolled back automatically (regardless of whether you are in a transaction).

Converting Between INTEGER and DATE

You can convert an integer column (SMALLINT, INTEGER, or INT8) to a DATE value. The database server interprets the integer as a value in the internal format of the DATE column. You can also convert a DATE column to an integer column. The database server stores the internal format of the DATE column as an integer representing a Julian date.

Converting Between DATE and DATETIME

You can convert DATE columns to DATETIME columns. If the DATETIME column contains more fields than the DATE column, however, the database server either ignores the fields or fills them with zeros. The illustrations in the following list show how these two data types are converted (assuming that the default date format is *mm/dd/yyyy*):

- If you convert DATE to DATETIME YEAR TO DAY, the database server converts the existing DATE values to DATETIME values. For example, the value 08/15/2002 becomes 2002-08-15.
- If you convert DATETIME YEAR TO DAY to the DATE format, the value 2002-08-15 becomes 08/15/2002.
- If you convert DATE to DATETIME YEAR TO SECOND, the database server converts existing DATE values to DATETIME values and fills in the additional DATETIME fields with zeros. For example, 08/15/2002 becomes 2002-08-15 00:00:00.
- If you convert DATETIME YEAR TO SECOND to DATE, the database server converts existing DATETIME to DATE values but drops fields for time units smaller than DAY. For example, 2002-08-15 12:15:37 becomes 08/15/2002.

Using User-Defined Casts

Implicit and explicit casts are owned by the users who create them. They govern casts and conversions between user-defined data types and other data types. Developers of user-defined data types must create certain implicit and explicit casts and the functions that are used to implement them. The casts allow user-defined types to be expressed in a form that clients can manipulate.

For information on how to register and use implicit and explicit casts, see the CREATE CAST statement in the *IBM Informix: Guide to SQL Syntax* and the *IBM Informix: Database Design and Implementation Guide*.

Implicit Casts

Implicit casts allow you to convert a user-defined data type to a built-in type or vice versa. The database server automatically invokes a single implicit cast when needed to evaluate and compare expressions or pass arguments. Operations that require more than one implicit cast fail.

Users can explicitly invoke an implicit cast using the CAST AS keywords or the double colon (::) cast operator.

Explicit Casts

Explicit casts, unlike implicit casts or built-in casts, are *never* invoked automatically by the database server. Users must invoke them explicitly with the CAST AS keywords or with the double colon (::) cast operator.

Determining Which Cast to Apply

The database server uses the following rules to determine which cast to apply in a particular situation:

- To compare two built-in types, the database server automatically invokes the appropriate built-in casts.
- The database server applies only one implicit cast per operand. If two or more casts are needed to convert the operand to the desired type, the user must explicitly invoke the additional casts.

In the following example, the literal value 5.55 is implicitly cast to DECIMAL, and is then explicitly cast to MONEY, and finally to yen:

```
CREATE DISTINCT TYPE yen AS MONEY
. . .
INSERT INTO currency_tab
VALUES (5.55::MONEY::yen)
```

- To compare a distinct type to its source type, the user must explicitly cast one type to the other.
- To compare a distinct type to a type other than its source, the database server looks for an implicit cast between the source type and the desired type.

If neither cast is registered, the user must invoke an explicit cast between the distinct type and the desired type. If this cast is not registered, the database server automatically invokes a cast from the source type to the desired type.

If none of these casts is defined, the comparison fails.

- To compare an opaque type to a built-in type, the user must explicitly cast the opaque type to a data type that the database server understands (such as LVARCHAR, SENDRECV, IMPEX, or IMPEXBIN). The database server then invokes built-in casts to convert the results to the desired built-in type.
- To compare two opaque types, the user must explicitly cast one opaque type to a form that the database server understands (such as LVARCHAR, SENDRECV, IMPEX, or IMPEXBIN) and then explicitly cast this type to the second opaque type.

For information about casting and the IMPEX, IMPEXBIN, LVARCHAR, and SENDRECV types, see *IBM Informix: User-Defined Routines and Data Types Developer's Guide*.

Casts for Distinct Types

You define a distinct type based on a built-in type or an existing opaque type or ROW type. Although data of the distinct type has the same length and alignment and is passed in the same way as data of the source type, the two cannot be compared directly. To compare a distinct type and its source type, you must explicitly cast one type to the other.

When you create a new distinct type, the database server automatically registers two explicit casts:

- A cast from the distinct type to its source type
- A cast from the source type to the distinct type

You can create an implicit cast between a distinct type and its source type. To create an implicit cast, however, you must first drop the default explicit cast between the distinct type and its source type.

You also can use all casts that have been registered for the source type without modification on the distinct type. You can also create and register new casts and support functions that apply *only* to the distinct type.

For examples that show how to create a cast function for a distinct type and register the function as cast, see the *IBM Informix: Database Design and Implementation Guide*.

Important: For releases of Dynamic Server earlier than Version 9.21, distinct data types inherited the built-in casts that are provided for the source type. The built-in casts of the source type are not inherited by distinct data types in this release.

What Extended Data Types Can Be Cast?

The next table shows the extended data type combinations that you can cast.

Target Type	Opaque Type	Distinct Type	Named ROW Type	Unnamed ROW Type	Collection Type	Built-in Type
Opaque Type	Explicit or implicit	Explicit	Explicit	Not Valid	Not Valid	Explicit or implicit ³
Distinct Type	Explicit ³	Explicit	Explicit	Not Valid	Not Valid	Explicit or implicit
Named ROW Type	Explicit ³	Explicit	Explicit ³	Explicit ¹	Not Valid	Not Valid
Unnamed ROW Type	Not Valid	Not Valid	Explicit ¹	Implicit ¹	Not Valid	Not Valid
Collection Type	Not Valid	Not Valid	Not Valid	Not Valid	Explicit ²	Not Valid
Built-in Type	Explicit or implicit ³	Explicit or implicit	Not Valid	Not Valid	Not Valid	System defined (implicit)

¹ Applies when two ROW types are structurally equivalent or casts exist to handle data conversions where corresponding field types are not the same.² Applies when a cast exists to convert between the element types of the respective collection types.³ Applies when a user-defined cast exists to convert between the two data types.

The table shows only whether or not a cast between a source type and a target type are possible. In some cases, you must first create a user-defined cast before you can perform a conversion between two data types. In other cases, the database server provides either an implicit cast or a built-in cast that you must explicitly invoke.

Operator Precedence

An *operator* is a symbol or keyword that can appear in an SQL expression. Most SQL operators are restricted in the data types of their operands and returned values. Some operators only support operands of built-in data types; others can support built-in and extended data types as operands.

The following table shows the precedence of the operators that Informix database servers support, in descending (highest to lowest) order of precedence. Operators with the same precedence are listed in the same row.

Operator Precedence	Example in Expression
. (<i>membership</i>) [] (<i>substring</i>)	customer.phone [1, 3]
UNITS	x UNITS DAY
+ - (<i>unary</i>)	- y
:: (<i>cast</i>)	NULL::TEXT
* /	x / y
+ - (<i>binary</i>)	x -y
(<i>concatenation</i>)	customer.fname customer.lname
ANY ALL SOME	orders.ship_date > SOME (SELECT paid_date FROM orders)
NOT	NOT y
< <= = > >= != <>	x >= y
IN BETWEEN ... AND LIKE MATCHES	customer.fname MATCHES y
AND	x AND y
OR	x OR y

See the *IBM Informix: Guide to SQL Syntax* for the syntax and semantics of these SQL operators.

Chapter 3. Environment Variables

Types of Environment Variables	3-3
Where to Set Environment Variables on UNIX	3-4
Where to Set Environment Variables on Windows	3-5
Using Environment Variables on UNIX	3-6
Setting Environment Variables in a Configuration File	3-6
Setting Environment Variables at Login Time	3-6
Syntax for Setting Environment Variables	3-7
Unsetting Environment Variables	3-7
Modifying an Environment-Variable Setting	3-8
Viewing Your Environment-Variable Settings.	3-8
Checking Environment Variables with the chkenv Utility	3-9
Rules of Precedence	3-9
Using Environment Variables on Windows	3-10
Environment Settings for Native Windows Applications	3-10
Environment Settings for Command-Prompt Utilities	3-11
Using the System Applet to Work with Environment Variables	3-11
Using the Command Prompt to Work with Environment Variables	3-12
Using dbservername.cmd to Initialize a Command-Prompt Environment	3-13
Rules of Precedence	3-13
List of Environment Variables	3-14
Environment Variables.	3-18
AC_CONFIG	3-18
AFDEBUG.	3-18
ANSIOWNER (IDS)	3-18
BIG_FET_BUF_SIZE (XPS)	3-19
CPFIRST	3-20
DBACCNOIGN	3-20
DBANSIWARN	3-21
DBBLOBBUF	3-22
DBCENTURY.	3-22
DBDATE	3-25
DBDELIMITER	3-28
DBEDIT	3-28
DBFLTMASK	3-29
DBLANG	3-29
DBMONEY	3-30
DBNLS (IDS)	3-32
DBONPLOAD (IDS)	3-33
DBPATH	3-33
DBPRINT	3-35
DBREMOTECMD (UNIX)	3-36
DBSPACETEMP	3-36
DBTEMP (IDS)	3-38
DBTIME	3-39

DBUPSPACE	3-41
DEFAULT_ATTACH	3-42
DELIMIDENT	3-42
ENVIGNORE (UNIX)	3-43
FET_BUF_SIZE	3-44
GLOBAL_DETACH_INFORM (XPS)	3-45
IBM_XPS_PARAMS (XPS)	3-45
IFMX_CART_ALRM (XPS)	3-46
IFMX_HISTORY_SIZE (XPS)	3-47
IFMX_OPT_FACT_TABS (XPS)	3-47
IFMX_OPT_NON_DIM_TABS (XPS)	3-48
IFX_DEF_TABLE_LOCKMODE (IDS)	3-48
IFX_DIRECTIVES	3-49
IFX_EXTDIRECTIVES	3-50
IFX_LONGID	3-51
IFX_NETBUF_PVTPOOL_SIZE (UNIX)	3-52
IFX_NETBUF_SIZE	3-52
IFX_NO_TIMELIMIT_WARNING	3-52
IFX_PAD_VARCHAR (IDS)	3-53
IFX_UPDDDESC (IDS)	3-53
IFX_XASTDCOMPLIANCE_XAEND	3-53
IMCADMIN	3-54
IMCCONFIG	3-55
IMCSERVER	3-55
INFORMIXC (UNIX)	3-55
INFORMIXCONCSMCFG (IDS)	3-56
INFORMIXCONRETRY	3-56
INFORMIXCONTIME	3-57
INFORMIXCPPMAP (IDS)	3-58
INFORMIXDIR	3-58
INFORMIXKEYTAB (UNIX)	3-59
INFORMIXOPCACHE (IDS)	3-59
INFORMIXSERVER	3-60
INFORMIXSHMBASE (UNIX)	3-60
INFORMIXSQLHOSTS	3-61
INFORMIXSTACKSIZE	3-62
INFORMIXTERM (UNIX)	3-62
INF_ROLE_SEP (IDS)	3-63
INTERACTIVE_DESKTOP_OFF (Windows)	3-64
ISM_COMPRESSION	3-64
ISM_DEBUG_FILE	3-64
ISM_DEBUG_LEVEL	3-65
ISM_ENCRYPTION	3-65
ISM_MAXLOGSIZE	3-65
ISM_MAXLOGVERS	3-66
JAR_TEMP_PATH (IDS)	3-66
JAVA_COMPILER (IDS)	3-66
JVM_MAX_HEAP_SIZE (IDS)	3-66
LD_LIBRARY_PATH (UNIX)	3-67
LIBERAL_MATCH (XPS)	3-67

LIBPATH (UNIX)	3-68
NODEFDAC	3-68
ONCONFIG	3-68
OPTCOMPIND	3-69
OPTMSG	3-70
OPTOFC	3-70
OPT_GOAL (IDS, UNIX)	3-71
PATH	3-71
PDQPRIORITY	3-72
Using PDQPRIORITY with Dynamic Server	3-73
Using PDQPRIORITY with Extended Parallel Server.	3-73
PLCONFIG (IDS)	3-74
PLOAD_LO_PATH (IDS)	3-74
PLOAD_SHMBASE (IDS)	3-74
PSORT_DBTEMP	3-75
PSORT_NPROCS	3-76
RTREE_COST_ADJUST_VALUE (IDS)	3-77
SHLIB_PATH (UNIX)	3-77
STMT_CACHE (IDS)	3-77
TERM (UNIX)	3-78
TERMCAP (UNIX)	3-78
TERMINFO (UNIX)	3-79
THREADLIB (UNIX)	3-79
TOBIGINT (XPS)	3-80
USETABLEAME (IDS)	3-80
XFER_CONFIG (XPS)	3-81
Index of Environment Variables.	3-81

In This Chapter

Various *environment variables* affect the functionality of your IBM Informix products. You can set environment variables that identify your terminal, specify the location of your software, and define other parameters.

Some environment variables are required; others are optional. You must either set or accept the default setting for required environment variables.

This chapter describes how to use the environment variables that apply to one or more IBM Informix products and shows how to set them.

Types of Environment Variables

Two types of environment variables are discussed in this chapter:

- Informix-specific environment variables
 - Set Informix environment variables when you want to work with IBM Informix products. Each IBM Informix product manual specifies the environment variables that you must set to use that product.

- Operating-system-specific environment variables

IBM Informix products rely on the correct setting of certain standard operating system environment variables. For example, you must always set the **PATH** environment variable.

In a UNIX environment, you might also need to set the **TERMCAP** or **TERMINFO** environment variable to use some products effectively.

The GLS environment variables that support nondefault locales are described in the *IBM Informix: GLS User's Guide*. The GLS variables are included in the list of environment variables in Table 3-1 on page 3-14 and in the topic index in Table 3-4 on page 3-81, but are not discussed in this manual.

The database server uses the environment variables that were in effect at the time when the database server was initialized.

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

Tip: Additional environment variables that are specific to your client application or SQL API might be discussed in the manual for that product.

Important: Do not set any environment variable in the home directory of user **informix** (nor in the file **.informix** in that directory) while initializing the database and creating the **sysmaster** database.

Where to Set Environment Variables on UNIX

You can set environment variables on UNIX in the following places:

- At the system prompt on the command line
When you set an environment variable at the system prompt, you must reassign it the next time you log into the system. See also “Using Environment Variables on UNIX” on page 3-6.
- In an environment-configuration file
An environment-configuration file is a common or private file where you can set all the environment variables that IBM Informix products use. The use of such files reduces the number of environment variables that you must set at the command line or in a shell file.
- In a login file
Values of environment variables set in your **.login**, **.cshrc**, or **.profile** file are assigned automatically every time you log into the system.
- In the SET ENVIRONMENT statement of SQL
Values of some environment variables can be reset by the SET ENVIRONMENT statement. The scope of the new settings is generally the

routine that executed the SET ENVIRONMENT statement, but it is the current session for the **OPTCOMPIND** environment variable of Dynamic Server, as described in the section “OPTCOMPIND” on page 3-69, and for environment variables of Extended Parallel Server that the **sysdbopen()** or **sysdbclose()** SPL routines can set. For more information on these routines and on the SET ENVIRONMENT statement, see the *IBM Informix: Guide to SQL Syntax*.

In IBM Informix ESQL/C, you can set supported environment variables within an application with the **putenv()** system call and retrieve values with the **getenv()** system call, if your UNIX system supports these functions. For more information on **putenv()** and **getenv()**, see the *IBM Informix: ESQL/C Programmer's Manual* and your C documentation.

Where to Set Environment Variables on Windows

You can set environment variables in several places on Windows, depending on which IBM Informix application you use.

For native Windows IBM Informix applications, such as the database server, environment variables can be set only in the Windows registry. Environment variables set in the registry cannot be modified elsewhere.

For utilities that run in a command-prompt session, such as **dbaccess**, environment variables can be set in several ways, as described in “Environment Settings for Command-Prompt Utilities” on page 3-11.

The SET ENVIRONMENT statement of SQL can set certain routine-specific environment options. For more information, refer to the description of SET ENVIRONMENT in the *IBM Informix: Guide to SQL Syntax*.

To use client applications such as ESQL/C or the Schema Tools on Windows environment, use the **Setnet32** utility to set environment variables. For information about the **Setnet32** utility, see the *IBM Informix: Client Products Installation Guide* for your operating system.

In IBM Informix ESQL/C, you can set supported environment variables within an application with the **ifx_putenv()** function and retrieve values with the **ifx_getenv()** function, if your Windows system supports them. For more information on **ifx_putenv()** and **ifx_getenv()**, see the *IBM Informix: ESQL/C Programmer's Manual*.

Using Environment Variables on UNIX

The following sections discuss setting, unsetting, modifying, and viewing environment variables. If you already use an IBM Informix product, some or all of the appropriate environment variables might be set.

Setting Environment Variables in a Configuration File

The common (shared) environment-configuration file that is provided with IBM Informix products resides in `$INFORMIXDIR/etc/informix.rc`. Permissions for this shared file must be set to 644.

A user can override the system or shared environment variables by setting variables in a private environment-configuration file. This file must have all of the following characteristics:

- Stored in the user's home directory
- Named `.informix`
- Permissions set to readable by the user

An environment-configuration file can contain comment lines (preceded by the `#` comment indicator) and variable definition lines that set values (separated by blank spaces or tabs), as the following example shows:

```
# This is an example of an environment-configuration file
#
DBDATE DMY4-
#
# These are ESQL/C environment variable settings
#
INFORMIXC gcc
CPFIRST TRUE
```

You can use the **ENVIGNORE** environment variable, described in “**ENVIGNORE (UNIX)**” on page 3-43, to override one or more entries in an environment-configuration file. Use the Informix **chkenv** utility, described in “**Checking Environment Variables with the chkenv Utility**” on page 3-9, to perform a sanity check on the contents of an environment-configuration file. The **chkenv** utility returns an error message if the file contains a bad environment variable or if the file is too large.

The first time you set an environment variable in a shell file or environment-configuration file, you must tell the shell process to read your entry before you work with your IBM Informix product. If you use a C shell, **source** the file; if you use a Bourne or Korn shell, use a period (`.`) to execute the file.

Setting Environment Variables at Login Time

Add commands that set your environment variables to the appropriate login file:

For C shell `.login` or `.cshrc`

For Bourne shell or Korn shell
`.profile`

Syntax for Setting Environment Variables

Use standard UNIX commands to set environment variables. The examples in the following table show how to set the ABCD environment variable to *value* for the C shell, Bourne shell, and Korn shell. The Korn shell also supports a shortcut, as the last row indicates. Environment variables are case sensitive.

Shell	Command
C	<code>setenv ABCD value</code>
Bourne	<code>ABCD=value</code> <code>export ABCD</code>
Korn	<code>ABCD=value</code> <code>export ABCD</code>
Korn	<code>export ABCD=value</code>

The following diagram shows how the syntax for setting an environment variable is represented throughout this chapter. These diagrams indicate the setting for the C shell; for the Bourne or Korn shells, use the syntax illustrated in the preceding table.

▶▶—`setenv—ABCD—value`—▶▶

For more information on how to read syntax diagrams, see “Syntax Diagrams” on page xiii of the Introduction.

Unsetting Environment Variables

To unset an environment variable, enter the following command.

Shell	Command
C	<code>unsetenv ABCD</code>
Bourne or Korn	<code>unset ABCD</code>

Modifying an Environment-Variable Setting

Sometimes you must add information to an environment variable that is already set. For example, the **PATH** environment variable is always set on UNIX. When you use an IBM Informix product, you must add to the **PATH** setting the name of the directory where the executable files for the IBM Informix products are stored.

In the following example, the **INFORMIXDIR** is **/usr/informix**. (That is, during installation, the IBM Informix products were installed in the **/usr/informix** directory.) The executable files are in the **bin** subdirectory, **/usr/informix/bin**. To add this directory to the front of the C shell **PATH** environment variable, use the following command:

```
setenv PATH /usr/informix/bin:$PATH
```

Rather than entering an explicit pathname, you can use the value of the **INFORMIXDIR** environment variable (represented as **\$INFORMIXDIR**), as the following example shows:

```
setenv INFORMIXDIR /usr/informix
setenv PATH $INFORMIXDIR/bin:$PATH
```

You might prefer to use this version to ensure that your **PATH** entry does not conflict with the search path that was set in **INFORMIXDIR**, and so that you do not have to reset **PATH** whenever you change **INFORMIXDIR**. If you set the **PATH** environment variable on the C shell command line, you might need to include braces (**{ }**) with the existing **INFORMIXDIR** and **PATH**, as the following command shows:

```
setenv PATH ${INFORMIXDIR}/bin:${PATH}
```

For more information about how to set and modify environment variables, refer to the manuals for your operating system.

Viewing Your Environment-Variable Settings

After you install one or more IBM Informix products, enter the following command at the system prompt to view your current environment settings.

UNIX Version	Command
BSD UNIX	env
UNIX System V	printenv

Checking Environment Variables with the `chkenv` Utility

The `chkenv` utility checks the validity of shared or private environment-configuration files. It validates the names of the environment variables in the file, but not their values. Use `chkenv` to provide debugging information when you define, in an environment-configuration file, all the environment variables that your IBM Informix products use.

```
▶—chkenv—┬──────────┴──filename──▶
```

filename is the name of the environment-configuration file to be debugged.

pathname is the full directory path in which the environment variable file is located.

File `$INFORMIXDIR/etc/informix.rc` is the shared environment-configuration file. A private environment-configuration file is stored as `.informix` in the home directory of the user. If you specify no *pathname* for `chkenv`, the utility checks both the shared and private environment configuration files. If you provide a *pathname*, `chkenv` checks only the specified file.

Issue the following command to check the contents of the shared environment-configuration file:

```
chkenv informix.rc
```

The `chkenv` utility returns an error message if it finds a bad environment-variable name in the file or if the file is too large. You can modify the file and rerun the utility to check the modified environment-variable names.

IBM Informix products ignore all lines in the environment-configuration file, starting at the point of the error, if the `chkenv` utility returns the following message:

```
-33523    filename: Bad environment variable on line number.
```

If you want the product to ignore specified environment-variables in the file, you can also set the `ENVIGNORE` environment variable. For a discussion of the use and format of environment-configuration files and the `ENVIGNORE` environment variable, see page 3-43.

Rules of Precedence

When an IBM Informix product accesses an environment variable, normally the following rules of precedence apply:

1. Of highest precedence is the value that is defined in the environment (shell) by explicitly setting the value at the shell prompt.

2. The second highest precedence goes to the value that is defined in the private environment-configuration file in the home directory of the user (`~/informix`).
3. The next highest precedence goes to the value that is defined in the common environment-configuration file (`$INFORMIXDIR/etc/informix.rc`).
4. The lowest precedence goes to the default value, if one exists.

For precedence information about GLS environment variables, see the *IBM Informix: GLS User's Guide*.

Important: If you set one or more environment variables before you start the database server, and you do not explicitly set the same environment variables for your client products, the clients will adopt the original settings.

Using Environment Variables on Windows

The following sections discuss setting, viewing, unsetting, and modifying environment variables for native Windows applications and command-prompt utilities.

Environment Settings for Native Windows Applications

IBM Informix native Windows applications, such as the database server itself, store their configuration information in the Windows registry. To modify this information, you must use the Registry Editor, `regedt32.exe`.

Important: In order to use the Registry Editor to change database server environment variables, you must belong to either the Administrators or Informix-Admin groups. For information on assigning users to groups, see your operating-system documentation.

To manipulate environment variables with the Registry Editor:

1. Launch the Registry Editor, `regedt32.exe`, and choose the window titled `HKEY_LOCAL_MACHINE`.
2. In the left pane, double-click the SOFTWARE registry key (shown as a small, yellow file folder icon).

The SOFTWARE registry key expands to show several subkeys, one of which is Informix. Continue down the tree in the following sequence:

OnLine, `dbservername`, Environment.

Substitute the name of your database server for `dbservername`.

3. With the Environment registry key selected in the left pane, you should see a list of environment variables and their values in the right pane (for example, `CLIENT_LOCALE:REG_SZ:EN_US.CP1252`).

4. Change existing environment variables, if needed.
 - a. Double-click the environment variable.
 - b. Type the new value in the String Editor dialog box.
 - c. Click OK to accept the value.
5. Add new environment variables, if needed.
 - a. Choose **Edit > Add Value** in the Registry Editor.
 - b. Enter the name of the environment variable in the Value Name edit box and choose REG_SZ as the data type.
 - c. Click OK and type a value for the environment variable in the String Editor dialog box.
6. Delete an environment variable, if needed.
 - a. Select the variable name.
 - b. Choose **Edit > Delete** in the Registry Editor.

Environment Settings for Command-Prompt Utilities

You can set environment variables for command-prompt utilities in the following ways:

- With the System applet in the Control Panel
- In a command-line session

Using the System Applet to Work with Environment Variables

The System applet provides a graphical interface to create, modify, and delete system-wide and user-specific variables. Environment variables that are set with the System applet are visible to all command-prompt sessions.

To change environment variables with the System applet in the control panel:

1. Double-click the System applet icon from the Control Panel window.

Click the Environment tab near the top of the window. Two list boxes display System Environment Variables and User Environment Variables. System Environment Variables apply to an entire system, and User Environment Variables apply only to the sessions of the individual user.
2. To change the value of an existing variable, select that variable.

The name of the variable and its current value appear in the boxes at the bottom of the window.
3. Highlight the existing value and type the new value.
4. To add a new variable, highlight an existing variable and type the new variable name in the box at the bottom of the window.
5. Next, enter the value for the new variable at the bottom of the window and click the **Set** button.
6. To delete a variable, select the variable and click the **Delete** button.

Important: In order to use the System applet to change System environment variables, you must belong to the Administrators group. For information on assigning users to groups, see your operating-system documentation.

Using the Command Prompt to Work with Environment Variables

The following diagram shows the syntax for setting an environment variable at a command prompt in Windows.

►—set—ABCD—=—value—◄◄

If no *value* is specified, the environment variable is unset, as if it did not exist.

For more information on how to read syntax diagrams, see “Syntax Diagrams” on page xiii of the introduction.

To view your current settings after one or more IBM Informix products are installed, enter the following command at the command prompt.

►—set—◄◄

Sometimes you must add information to an environment variable that is already set. For example, the **PATH** environment variable is always set in Windows environments. When you use an IBM Informix product, you must add the name of the directory where the executable files for the IBM Informix products are stored to the **PATH**.

In the following example, **INFORMIXDIR** is **d:\informix** (that is, during installation, IBM Informix products were installed in the **d:\informix** directory). The executable files are in the **bin** subdirectory, **d:\informix\bin**. To add this directory at the beginning of the **PATH** environment-variable value, use the following command:

```
set PATH=d:\informix\bin;%PATH%
```

Rather than entering an explicit pathname, you can use the value of the **INFORMIXDIR** environment variable (represented as **%INFORMIXDIR%**), as the following example shows:

```
set INFORMIXDIR=d:\informix
set PATH=%INFORMIXDIR%\bin;%PATH%
```

You might prefer to use this version to ensure that your **PATH** entry does not contradict the search path that was set in **INFORMIXDIR** and to avoid the need to reset **PATH** whenever you change **INFORMIXDIR**.

For more information about setting and modifying environment variables, refer to your operating-system manuals.

Using `dbservername.cmd` to Initialize a Command-Prompt Environment

Each time that you open a Windows command prompt, it acts as an independent environment. Therefore, environment variables that you set within it are valid only for that particular command-prompt instance.

For example, if you open one command window and set the variable, **INFORMIXDIR**, and then open another command window and type `set` to check your environment, you will find that **INFORMIXDIR** is not set in the new command-prompt session.

The database server installation program creates a *batch file* that you can use to configure command-prompt utilities, ensuring that your command-prompt environment is initialized correctly each time that you run a command-prompt session. The batch file, **dbservername.cmd**, is located in **%INFORMIXDIR%**, and is a plain text file that you can modify with any text editor. If you have more than one database server installed in **%INFORMIXDIR%**, there will be more than one batch file with the **.cmd** extension, each bearing the name of the database server with which it is associated.

To run **dbservername.cmd** from a command prompt, type **dbservername** or configure a command prompt so that it runs **dbservername.cmd** automatically at start up.

Rules of Precedence

When an IBM Informix product accesses an environment variable, normally the following rules of precedence apply:

1. The highest precedence goes to the value that is defined in the environment by explicitly setting the value at the command prompt.
2. The second highest precedence goes to the value that is defined in the System control panel as a User Environment Variable.
3. The third highest precedence goes to the value that is defined in the System control panel as a System Environment Variable.
4. The lowest precedence goes to the default value.

Important: Because Windows services access only environment variables that are set in the registry, the preceding rules of precedence do not apply for IBM Informix native Windows applications. For native Windows applications, the highest precedence goes to variables that are explicitly defined in the registry, and the lowest precedence goes to the default value. In addition, if you set one or more environment variables before you start the database server,

and you do not explicitly set the same environment variables for client products, the clients will adopt the original settings.

List of Environment Variables

Table 3-1 contains an alphabetical list of the environment variables that you can set for an Informix database server and SQL API products. Most of these environment variables are described in this chapter on the pages listed in the “Page” column. The • symbol indicates that XPS or Dynamic Server (or both, if both columns are so marked) support the environment variable.

The notation ERG in the Page column indicates an environment variable that must be set with the CDR_ENV configuration parameter and that is described in the appendix on configuration parameters and environment variables of the *IBM Informix: Dynamic Server Enterprise Replication Guide*.

The notation GLS in the Page column indicates a GLS environment variable that is valid in nondefault locales and that is described in the GLS environment variables chapter of *IBM Informix: GLS User's Guide*.

Table 3-1. Alphabetical List Of Environment Variables

Environment Variable	XPS	IDS	Restrictions	Page
AC_CONFIG	X	X	ON-Bar	3-18
AFDEBUG			JVM	3-18
ANSIOWNER		X	None	3-18
BIG_FET_BUF_SIZE	X		SQL APIs and DB-Access only	3-19
CC8BITLEVEL			ESQL/C only	GLS
CDRSITES_731		X	ER only	ERG
CDRSITES_92X		X	ER only	ERG
CDR_LOGDELTA		X	ER only	ERG
CDR_PERFLOG		X	ER only	ERG
CDR_ROUTER		X	ER only	ERG
CDR_RMSCALEFACT		X	ER only	ERG
CLIENT_LOCALE	X	X	None	GLS
CPFIRST	X	X	ESQL/C only	3-20
DBACCNOIGN	X	X	DB-Access only	3-20
DBANSIWARN	X	X	None	3-21
DBBLOBBUF	X	X	UNLOAD only	3-22
DBCENTURY			SQL APIs only	3-22

Table 3-1. Alphabetical List Of Environment Variables (continued)

Environment Variable	XPS	IDS	Restrictions	Page
DBDATE	X	X	None	3-25; GLS
DBDELIMITER	X	X	None	3-28
DBEDIT	X	X	None	3-28
DBFLTMASK	X	X	DB-Access only	3-29
DBLANG	X	X	None	3-29; GLS
DBMONEY	X	X	None	3-30; GLS
DBNLS		X		3-32
DBONPLOAD		X	HPL only	3-33
DBPATH	X	X	None	3-33
DBPRINT	X	X	UNIX only	3-35
DBREMOTECMD	X	X	UNIX only	3-36
DBSPACETEMP	X	X	None	3-36
DBTEMP		X	DB-Access, Gateways	3-38
DBTIME			SQL APIs only	3-39; GLS
DBUPSPACE	X	X	None	3-41
DB_LOCALE	X	X	None	GLS
DEFAULT_ATTACH		X	Deprecated	3-42
DELIMIDENT	X	X	None	3-42
ENVIGNORE	X	X	UNIX only	3-43
ESQLMF	X	X	ESQL/C only	GLS
FET_BUF_SIZE	X	X	SQL APIs, DB-Access only	3-44
GLOBAL_DETACH_INFORM	X		None	3-45
GLS8BITFSYS	X	X	None	GLS
GL_DATE	X	X	None	GLS
GL_DATETIME	X	X	None	GLS
IBM_XPS_PARAMS	X		None	3-45
IFMX_CART_ALRM	X		None	3-46
IFMX_HISTORY_SIZE	X		DB-Access	3-47
IFMX_OPT_FACT_TABS	X		None	3-47
IFMX_OPT_NON_DIM_TABS	X		None	3-48
IFX_DEF_TABLE_LOCKMODE		X	None	3-48
IFX_DIRECTIVES	X	X	None	3-49

Table 3-1. Alphabetical List Of Environment Variables (continued)

Environment Variable	XPS	IDS	Restrictions	Page
IFX_EXTDIRECTIVES	X	X	Set on the client only	3-49
IFX_LONGID	X	X	None	3-51
IFX_NETBUF_PVTPPOOL_SIZE	X	X	UNIX only	3-52
IFX_NETBUF_SIZE	X	X	None	3-53
IFX_NO_TIMELIMIT_WARNING	X	X	None	3-52
IFX_PAD_VARCHAR		X	None	3-53
IFX_UPDDESC		X	None	3-53
IFX_XASTDCOMPLIANCE_XAEND		X	None	3-53
IMCADMIN		X		3-54
IMCCONFIG		X		3-55
IMCSERVER		X		3-55
INFORMIXC			ESQL/C, UNIX only	3-55
INFORMIXCONCSMCFG		X	None	3-56
INFORMIXCONRETRY	X	X	None	3-56
INFORMIXCONTIME	X	X	None	3-56
INFORMIXCPPMAP		X	None	3-58
INFORMIXDIR	X	X	None	3-58
INFORMIXKEYTAB	X	X	UNIX only	3-59
INFORMIXOPCACHE		X	Optical Subsystem only	3-59
INFORMIXSERVER	X	X	None	3-60
INFORMIXSHMBASE	X	X	UNIX only	3-60
INFORMIXSQLHOSTS	X	X	None	3-61
INFORMIXSTACKSIZE	X	X	None	3-62
INFORMIXTERM	X	X	DB-Access, UNIX only	3-62
INF_ROLE_SEP		X	None	3-63
INTERACTIVE_DESKTOP_OFF		X	Windows only	3-64
ISM_COMPRESSION	X	X	ISM, ON-Bar only	3-64
ISM_DEBUG_FILE	X	X	ISM only	3-64
ISM_DEBUG_LEVEL	X	X	ISM, ON-Bar only	3-65
ISM_ENCRYPTION	X	X	ISM, ON-Bar only	3-65
ISM_MAXLOGSIZE	X	X	ISM only	3-65
ISM_MAXLOGVERS	X	X	ISM only	3-66

Table 3-1. Alphabetical List Of Environment Variables (continued)

Environment Variable	XPS	IDS	Restrictions	Page
JAR_TEMP_PATH		X	JVM	3-66
JAVA_COMPILER		X	JVM	3-66
JVM_MAX_HEAP_SIZE		X	JVM	3-66
LD_LIBRARY_PATH			SQL APIs, UNIX only	3-67
LIBERAL_MATCH	X		None	3-67
LIBPATH			SQL APIs, UNIX only	3-68
NODEFDAC	X	X	None	3-68
ONCONFIG	X	X	None	3-68
OPTCOMPIND	X	X	None	3-69
OPTMSG			ESQL/C only	3-70
OPTOFC			ESQL/C only	3-70
OPT_GOAL		X	UNIX only	3-71
PATH	X	X	None	3-71
PDQPRIORITY	X	X	None	3-72
PLCONFIG		X	HPL only	3-74
PLOAD_LO_PATH		X	HPL only	3-74
PLOAD_SHMBASE		X	HPL only	3-74
PSORT_DBTEMP	X	X	None	3-75
PSORT_NPROCS	X	X	None	3-76
RTREE_COST_ADJUST_VALUE		X	None	3-77
SERVER_LOCALE	X	X	None	GLS
SHLIB_PATH	X	X	UNIX only	3-77
STMT_CACHE		X	None	3-77
TERM	X	X	UNIX only	3-78
TERMCAP	X	X	UNIX only	3-78
TERMINFO	X	X	UNIX only	3-79
THREADLIB			ESQL/C, UNIX only	3-80
TOBIGINT	X		dbschema only	3-80
USETABLENAME		X	None	3-80
XFER_CONFIG	X		None	3-81

Tip: You might encounter references to environment variables that are not listed in Table 3-1 on page 3-14. Most likely, these environment variables

are not supported in this release or are used to maintain backward compatibility with certain earlier product versions. For information, refer to an earlier version of your IBM Informix documentation.

Environment Variables

Sections that follow discuss (in alphabetical order) environment variables that IBM Informix database server products and their utilities use.

Important: The descriptions of the following environment variables include the syntax for setting the environment variable on UNIX. For a general description of how to set these environment variables on Windows, see “Environment Settings for Native Windows Applications” on page 3-10 and “Environment Settings for Command-Prompt Utilities” on page 3-11.

AC_CONFIG

You can set the **AC_CONFIG** environment variable to specify the path for the **ac_config.std** configuration file for the **archecker** utility, which checks the validity and completeness of an ON-Bar storage-space backup. The **ac_config.std** file contains default **archecker** configuration parameters.

►►setenv—AC_CONFIG—*pathname*—◄◄

pathname is the location of the **ac_config.std** configuration file in **\$INFORMIXDIR/etc** or **%INFORMIXDIR%\etc**.

For information on **archecker**, see your *IBM Informix: Backup and Restore Guide*.

AFDEBUG

You can create files to hold verbose messages from the Java virtual machine (JVM) about releasing memory that had been allocated to objects by setting the **AFDEBUG** environment variable.

►►setenv—AFDEBUG—◄◄

No value is required. You can also set the configuration parameter **AFCRASH** to **0x00000010** to achieve the same result.

ANSIOWNER (IDS)

In an ANSI-compliant database, you can prevent the default behavior of upshifting lowercase letters in owner names that are not delimited by quotation marks by setting the **ANSIOWNER** environment variable to 1.

To prevent upshifting of lowercase letters in owner names in an ANSI-compliant database, you must set **ANSIOWNER** before you initialize Dynamic Server.

The following table shows how an ANSI-compliant database of Dynamic Server stores or reads the specified name of a database object called **oblong** if you were the owner of **oblong** and your **userid** (in all lowercase letters) were **owen**:

Table 3-2. Lettercase of implicit, unquoted, and quoted owner names, with and without ANSIOWNER

Owner Format	Specification	ANSIOWNER = 1	ANSIOWNER Not Set
Implicit:	oblong	owen.oblong	OWEN.oblong
Unquoted:	owen.oblong	owen.oblong	OWEN.oblong
Quoted:	'owen'.oblong	owen.oblong	owen.oblong

Because they do not match the lettercase of your **userid**, any SQL statements that specified the formats that are stored as **OWEN.oblong** would fail with errors.

BIG_FET_BUF_SIZE (XPS)

The **BIG_FET_BUF_SIZE** environment variable functions the same as the **FET_BUF_SIZE** environment variable, but supports a larger cursor buffer.

▶▶—setenv—BIG_FET_BUF_SIZE—*size*—◀◀

size is a positive integer that is larger than the default buffer size.

The *size* can be no greater than 4 gigabytes and specifies the size (in bytes) of the fetch buffer that holds data retrieved by a query. For example, to set a buffer size to 5,000 bytes on a UNIX system that uses the C shell, set the **BIG_FET_BUF_SIZE** environment variable with the following command:

```
setenv BIG_FET_BUF_SIZE 5000
```

When **BIG_FET_BUF_SIZE** is set to a valid value, the new value overrides the default value (or any previously set value of **BIG_FET_BUF_SIZE**). The default setting for the fetch buffer is dependent on row size. The processing of **BYTE** and **TEXT** values is not affected by **BIG_FET_BUF_SIZE**.

No error is raised if **BIG_FET_BUF_SIZE** is set to a value less than the default size or out of the range of **SMALLINT** values. In these cases, however, the invalid fetch buffer size is ignored and the default size is in effect.

If you set **BIG_FET_BUF_SIZE** to a valid value, that value is in effect for the local database server as well as for any remote database server from which you retrieve rows through a distributed query in which the local server is the coordinator and the remote server is subordinate. The greater the size of the buffer, the more rows can be returned and the less frequently the client application must wait for returned rows. A large buffer can improve performance by reducing the overhead of filling the client-side buffer.

CPFIRST

Set the **CPFIRST** environment variable to specify the default compilation order for all ESQL/C source files in your programming environment.

```
►—setenv—CPFIRST—TRUE—►
                    └─FALSE─┘
```

When you compile an ESQL/C program with **CPFIRST** not set, the ESQL/C preprocessor runs first, by default, on the program source file and then passes the resulting file to the C language preprocessor and compiler. You can, however, compile an ESQL/C program source file in the following order:

1. Run the C preprocessor
2. Run the ESQL/C preprocessor
3. Run the C compiler and linker

To use a nondefault compilation order for a specific program, you can either give the program source file a **.ecp** extension, run the **-cp** option with the **esql** command on a program source file with a **.ec** extension, or set **CPFIRST**.

Set **CPFIRST** to **TRUE** (uppercase only) to run the C preprocessor before the ESQL/C preprocessor on all ESQL/C source files in your environment, irrespective of whether the **-cp** option is passed to the **esql** command or the source files have the **.ec** or the **.ecp** extension.

To restore the default order on a system where the **CPFIRST** environment variable has been set to **TRUE**, you can set **CPFIRST** to **FALSE**. On UNIX systems that support the C shell, the following command has the same effect:

```
unsetenv CPFIRST
```

DBACCNOIGN

The **DBACCNOIGN** environment variable affects the behavior of the DB–Access utility if an error occurs under one of the following circumstances:

- You run DB–Access in nonmenu mode.
- In Dynamic Server only, you execute the **LOAD** command with DB–Access in menu mode.

Set the **DBACCNOIGN** environment variable to 1 to roll back an incomplete transaction if an error occurs while you run the DB–Access utility under either of the preceding conditions.

▶▶—setenv—DBACCNOIGN—1—————▶▶

For example, assume DB–Access runs the following SQL commands:

```
DATABASE mystore
BEGIN WORK

INSERT INTO receipts VALUES (cust1, 10)
INSERT INTO receipt VALUES (cust1, 20)
INSERT INTO receipts VALUES (cust1, 30)

UPDATE customer
  SET balance =
    (SELECT (balance-60)
     FROM customer WHERE custid = 'cust1')
  WHERE custid = 'cust1'
COMMIT WORK
```

Here, one statement has a misspelled table name: the **receipt** table does not exist. If **DBACCNOIGN** is not set in your environment, DB–Access inserts two records into the **receipts** table and updates the **customer** table. Now, the decrease in the **customer** balance exceeds the sum of the inserted receipts.

But if **DBACCNOIGN** is set to 1, messages appear that indicate that DB–Access rolled back all the INSERT and UPDATE statements. The messages also identify the cause of the error so that you can resolve the problem.

LOAD Statement Example

You can set **DBACCNOIGN** to protect data integrity during a LOAD statement, even if DB–Access runs the LOAD statement in menu mode.

Assume you execute the LOAD statement from the DB–Access SQL menu. Forty-nine rows of data load correctly, but the 50th row contains an invalid value that causes an error. If you set **DBACCNOIGN** to 1, the database server does not insert the forty-nine previous rows into the database. If **DBACCNOIGN** is not set, the database server inserts the first forty-nine rows.

DBANSIWARN

Setting the **DBANSIWARN** environment variable indicates that you want to check for Informix extensions to ANSI-standard SQL syntax. Unlike most environment variables, you do not need to set **DBANSIWARN** to a value. You can set it to any value or to no value.

Running DB–Access with **DBANSIWARN** set is functionally equivalent to including the **-ansi** flag when you invoke DB–Access (or any IBM Informix product that recognizes the **-ansi** flag) from the command line. If you set **DBANSIWARN** before you run DB–Access, any syntax-extension warnings are displayed on the screen within the SQL menu.

At runtime, the **DBANSIWARN** environment variable causes the sixth character of the **sqlwarn** array in the SQL Communication Area (SQLCA) to be set to **W** when a statement is executed that is recognized as including any Informix extension to the ANSI/ISO standard for SQL syntax.

For details on SQLCA, see the *IBM Informix: ESQL/C Programmer's Manual*.

After you set **DBANSIWARN**, Informix extension checking is automatic until you log out or unset **DBANSIWARN**. To turn off Informix extension checking, you can disable **DBANSIWARN** with this command:

```
unsetenv DBANSIWARN
```

DBBLOBBUF

The **DBBLOBBUF** environment variable controls whether TEXT or BYTE values are stored temporarily in memory or in a file while being processed by the UNLOAD statement. **DBBLOBBUF** affects only the UNLOAD statement.

▶—setenv—DBBLOBBUF—*size*—▶

size represents the maximum size of TEXT or BYTE data in kilobytes.

If the TEXT or BYTE data size is smaller than the default of 10 kilobytes (or the setting of **DBBLOBBUF**), the TEXT or BYTE value is temporarily stored in memory. If the data size is larger than the default or the **DBBLOBBUF** setting, the data value is written to a temporary file. For instance, to set a buffer size of 15 kilobytes, set **DBBLOBBUF** as in the following example:

```
setenv DBBLOBBUF 15
```

Here any TEXT or BYTE value smaller than 15 kilobytes is stored temporarily in memory. Values larger than 15 kilobytes are stored temporarily in a file.

DBCENTURY

To avoid problems in expanding abbreviated years, applications should require entry of 4-digit years, and should always display years as four digits. The **DBCENTURY** environment variable specifies how to expand literal DATE and DATETIME values that are entered with abbreviated year values.



When **DBCENTURY** is not set (or is set to R), the first two digits of the current year are used to expand 2-digit year values. For example, if today's date is 09/30/2003, then the abbreviated date 12/31/99 expands to 12/31/2099, and the abbreviated date 12/31/00 expands to 12/31/2000.

The R, P, F, and C settings choose algorithms for expanding two-digit years.

Setting	Algorithm
R = Current	Use the first two digits of the current year to expand the year value.
P = Past	Expanded dates are created by prefixing the abbreviated year value with 19 and 20. Both dates are compared to the current date, and the most recent date that is earlier than the current date is used.
F = Future	Expanded dates are created by prefixing the abbreviated year value with 20 and 21. Both dates are compared to the current date, and the earliest date that is later than the current date is used.
C = Closest	Expanded dates are created by prefixing the abbreviated year value with 19, 20, and 21. These three dates are compared to the current date, and the date that is closest to the current date is used.

Settings are case sensitive, and no error is issued for invalid settings. If you enter f (for example), then the default (R) setting takes effect. The P and F settings cannot return the current date, which is not in the past or future.

Years entered as a single digit are prefixed with 0 and then expanded. Three-digit years are not expanded. Pad years earlier than 100 with leading zeros.

Examples of Expanding Year Values

The following examples illustrate how various settings of **DBCENTURY** cause abbreviated years to be expanded in **DATE** and **DATETIME** values.

DBCENTURY = P:

```
Example data type: DATE
Current date: 4/6/2003
User enters: 1/1/1
Prefix with "19" expansion : 1/1/1901
Prefix with "20" expansion: 1/1/2001
Analysis: Both are prior to current date, but 1/1/2001 is closer to
current date.
```

Important: The effect of **DBCENTURY** depends on the current date from the system clock-calendar. Thus, 1/1/1, the abbreviated date in this example, would instead be expanded to 1/1/1901 if the current date were 1/1/2001 and **DBCENTURY** = P.

DBCENTURY = F:

Example data type: DATETIME year to month
Current date: 5/7/2005
User enters: 1-1
Prefix with "20" expansion: 2001-1
Prefix with "21" expansion: 2101-1
Analysis: Only date 2101-1 is after the current date, so it is chosen.

DBCENTURY = C:

Example data type: DATE
Current date: 4/6/2000
User enters: 1/1/1
Prefix with "19" expansion : 1/1/1901
Prefix with "20" expansion: 1/1/2001
Prefix with "21" expansion: 1/1/2101
Analysis: Here 1/1/2001 is closest to the current date, so it is chosen.

DBCENTURY = R or DBCENTURY Not Set:

Example data type: DATETIME year to month
Current date: 4/6/2000
User enters: 1-1
Prefix with "20" expansion: 2001-1

Example data type: DATE
Current date: 4/6/2003
User enters: 0/1/1
Prefix with "20" expansion: 2000/1
Analysis: In both examples, the Prefix with "20" algorithm is used.

Setting **DBCENTURY** does not affect IBM Informix products when the locale specifies a non-Gregorian calendar, such as Hebrew or Islamic calendars. The leading digits of the current year are used for alternate calendar systems when the year is abbreviated.

Abbreviated Years and Expressions in Database Objects

When an expression in a database object (including a check constraint, fragmentation expression, SPL routine, trigger, or UDR) contains a literal date or DATETIME value in which the year has one or two digits, the database server evaluates the expression using the setting that **DBCENTURY** (and other relevant environment variables) had when the database object was created (or was last modified). If **DBCENTURY** has been reset to a new value, the new value is ignored when the abbreviated year is expanded.

For example, suppose a user creates a table and defines the following check constraint on a column named **birthdate**:

```
birthdate < '09/25/50'
```

The expression is interpreted according to the value of **DBCENTURY** when the constraint was defined. If the table that contains the **birthdate** column is created on 09/23/2000 and **DBCENTURY** =C, the check constraint expression is consistently interpreted as `birthdate < '09/25/1950'` when inserts or updates are performed on the **birthdate** column. Even if different values of **DBCENTURY** are set when users perform inserts or updates on the **birthdate** column, the constraint expression is interpreted according to the setting at the time when the check constraint was defined (or was last modified).

Database objects created on some earlier versions of Dynamic Server do not support the priority of creation-time settings.

For legacy objects to acquire this feature:

1. Drop the objects.
2. Re-create them (or for fragmentation expressions, detach them and then reattach them).

After the objects are redefined, date literals within expressions of the objects will be interpreted according to the environment at the time when the object was created or was last modified. Otherwise, their behavior will depend on the runtime environment and might become inconsistent if this changes.

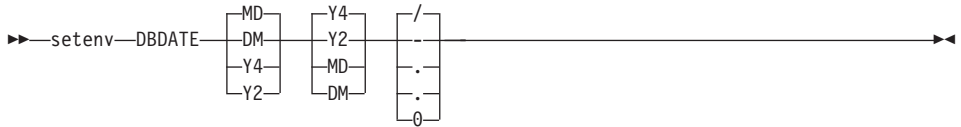
Administration of a database that includes a mix of legacy objects and new objects might become difficult because of differences between the new and the old behavior for evaluating date expressions. To avoid this, it is recommended that you redefine any legacy objects.

The value of **DBCENTURY** and the current date are not the only factors that determine how the database server interprets date and **DATETIME** values. The **DBDATE**, **DBTIME**, **GL_DATE**, and **GL_DATETIME** environment variables can also influence how dates are interpreted. For information about **GL_DATE** and **GL_DATETIME**, see the *IBM Informix: GLS User's Guide*.

Important: The behavior of **DBCENTURY** for Dynamic Server *and* Extended Parallel Server is not backwards compatible.

DBDATE

The **DBDATE** environment variable specifies the end-user formats of **DATE** values. On UNIX systems that use the C shell, set **DBDATE** with this syntax.



The following formatting symbols are valid in the **DBDATE** setting:

- ./ are characters that can appear as separators in a date format.
- 0 indicates that no separator is displayed between time units.
- D, M are characters that represent the day and the month.
- Y2, Y4 are characters that represent the year and the precision of the year.

Some East Asian locales support additional syntax for era-based dates. For details of era-based formats, see *IBM Informix: GLS User's Guide*.

DBDATE can specify the following attributes of the display format:

- The order of time units (the month, day, and year) in a date
- Whether the year appears with two digits (Y2) or four digits (Y4)
- The separator between the month, day, and year time units

For the U.S. English locale, the default for **DBDATE** is MDY4/, where M represents the month, D represents the day, Y4 represents a four-digit year, and slash (/) is the time-units separator (for example, 01/08/2002). Other valid characters for the separator are a hyphen (-), a period (.), or a zero (0). To indicate no separator, use the zero. The slash (/) is used by default if you attempt to specify a character other than a hyphen, period, or zero as a separator, or if you do not include any separator in the **DBDATE** specification.

If **DBDATE** is not set on the client, any **DBDATE** setting on the database server overrides the MDY4/ default on the client. If **DBDATE** is set on the client, that value (rather than the setting on the database server) is used by the client.

The following table shows some examples of valid **DBDATE** settings and their corresponding displays for the date 8 January, 2005:

DBDATE Setting	Representation of January 8, 2005:		DBDATE Setting	Representation of January 8, 2005:
MDY4/	01/08/2005		Y2DM.	05.08.01
DMY2-	08-01-05		MDY20	010805
MDY4	01/08/2005		Y4MD*	2005/01/08

Formats Y4MD* (because asterisk is not a valid separator) and MDY4 (with no separator defined) both display the default symbol (slash) as the separator.

Important: If you use the Y2 format, the setting of the **DBCENTURY** environment variable can also affect how literal DATE values are evaluated in data entry.

Also, certain routines that *IBM Informix ESQL/C* calls can use the **DBTIME** variable, rather than **DBDATE**, to set DATETIME formats to international specifications. For more information, see the discussion of the **DBTIME** environment variable in “DBTIME” on page 3-39 and in the *IBM Informix: ESQL/C Programmer’s Manual*.

The setting of the **DBDATE** variable takes precedence over that of the **GL_DATE** environment variable, as well as over any default DATE format that **CLIENT_LOCALE** specifies. For information about **GL_DATE** and **CLIENT_LOCALE**, see the *IBM Informix: GLS User’s Guide*.

End-user formats affect the following contexts:

- When you display DATE values, IBM Informix products use the **DBDATE** environment variable to format the output.
- During data entry of DATE values, IBM Informix products use the **DBDATE** environment variable to interpret the input.

For example, if you specify a literal DATE value in an INSERT statement, the database server expects this literal value to be compatible with the format that **DBDATE** specifies. Similarly, the database server interprets the date that you specify as the argument to the **DATE()** function to be in **DBDATE** format.

DATE Expressions in Database Objects

When an expression in a database object (including a check constraint, fragmentation expression, SPL routine, trigger, or UDR) contains a literal date value, the database server evaluates the expression using the setting that **DBDATE** (or other relevant environment variables) had when the database object was created (or was last modified). If **DBDATE** has been reset to a new value, the new value is ignored when the literal DATE is evaluated.

For example, suppose **DBDATE** is set to MDY2/ and a user creates a table with the following check constraint on the column **orderdate**:

```
orderdate < '06/25/98'
```

The date of the preceding expression is formatted according to the value of **DBDATE** when the constraint is defined. The check constraint expression is interpreted as `orderdate < '06/25/98'` regardless of the value of **DBDATE** during inserts or updates on the **orderdate** column. Suppose **DBDATE** is reset to DMY2/ when a user inserts the value '30/01/98' into the **orderdate** column. The date value inserted uses the date format DMY2/, whereas the check constraint expression uses the date format MDY2/.

See “Abbreviated Years and Expressions in Database Objects” on page 3-24 for a discussion of legacy objects from earlier versions of Informix database servers that are always evaluated according to the runtime environment. That section describes how to redefine objects so that dates are interpreted according to environment variable settings that were in effect when the object was defined (or when the object was last modified).

Important: The behavior of **DBDATE** for Dynamic Server *and* Extended Parallel Server is not backwards compatible.

DBDELIMITER

The **DBDELIMITER** environment variable specifies the field delimiter used with the **dbexport** utility and with the **LOAD** and **UNLOAD** statements.

```
►—setenv—DBDELIMITER—'delimiter'—◄
```

delimiter is the field delimiter for unloaded data files.

The *delimiter* can be any single character, except those in the following list:

- Hexadecimal digits (0 through 9, a through f, A through F)
- Newline or CTRL-J
- The backslash (\) symbol

The vertical bar (| = ASCII 124) is the default. To change the field delimiter to a plus (+) symbol, for example, you can set **DBDELIMITER** as follows:

```
setenv DBDELIMITER '+'
```

DBEDIT

The **DBEDIT** environment variable specifies the text editor to use with SQL statements and command files in DB-Access. If **DBEDIT** is set, the specified text editor is invoked automatically. If **DBEDIT** is not set, you are prompted to specify a text editor as the default for the rest of the session.

▶▶—setenv—DBEDIT—*editor*—————▶▶

editor is the name of the text editor you want to use.

For most UNIX systems, the default text editor is **vi**. If you use another text editor, be sure that it creates flat ASCII files. Some word processors in *document mode* introduce printer control characters that can interfere with the operation of your IBM Informix product.

To specify the EMACS text editor, set **DBEDIT** with the following command:

```
setenv DBEDIT emacs
```

DBFLTMASK

The DB–Access utility displays the floating-point values of data types **FLOAT**, **SMALLFLOAT**, and **DECIMAL(*p*)** within a 14-character buffer. By default, DB–Access displays as many digits to the right of the decimal point as will fit into this character buffer. Therefore, the actual number of decimal digits that DB–Access displays depends on the size of the floating-point value.

To reduce the number of digits displayed to the right of the decimal point in floating-point values, set **DBFLTMASK** to the desired number of digits.

▶▶—setenv—DBFLTMASK—*scale*—————▶▶

scale is the number of decimal digits that you want the IBM Informix client application to display in the floating-point values. Here *scale* must be smaller than 16, the default number of digits displayed.

If the floating-point value contains more digits to the right of the decimal than **DBFLTMASK** specifies, DB–Access rounds the value to the specified number of digits. If the floating-point value contains fewer digits to the right of the decimal, DB–Access pads the value with zeros. If you set **DBFLTMASK** to a value greater than can fit into the 14-character buffer, however, DB–Access rounds the value to the number of digits that can fit.

DBLANG

The **DBLANG** environment variable specifies the subdirectory of **\$INFORMIXDIR** or the full pathname of the directory that contains the compiled message files that an IBM Informix product uses.

▶▶—setenv—DBLANG—*relative_path*—————▶▶
 └─*full_path*—┘

relative_path is a subdirectory of **\$INFORMIXDIR**.

full_path is the pathname to the compiled message files.

By default, IBM Informix products put compiled messages in a locale-specific subdirectory of the `$INFORMIXDIR/msg` directory. These compiled message files have the file extension `.iem`. If you want to use a message directory other than `$INFORMIXDIR/msg`, where, for example, you can store message files that you create, you must perform the following steps:

To use a message directory other than `$INFORMIXDIR/msg`:

1. Use the `mkdir` command to create the appropriate directory for the message files.
You can make this directory under the directory `$INFORMIXDIR` or `$INFORMIXDIR/msg`, or you can make it under any other directory.
2. Set the owner and group of the new directory to `informix` and the access permission for this directory to 755.
3. Set the `DBLANG` environment variable to the new directory. If this is a subdirectory of `$INFORMIXDIR` or `$INFORMIXDIR/msg`, then you need only list the relative path to the new directory. Otherwise, you must specify the full pathname of the directory.
4. Copy the `.iem` files or the message files that you created to the new message directory that `$DBLANG` specifies.
All the files in the message directory should have the owner and group `informix` and access permission 644.

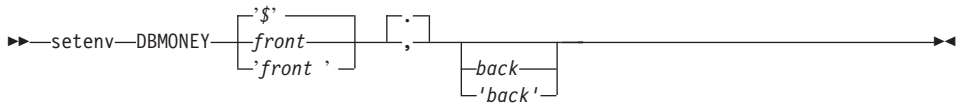
IBM Informix products that use the default U.S. English locale search for message files in the following order:

1. In `$DBLANG`, if `DBLANG` is set to a full pathname
2. In `$INFORMIXDIR/msg/$DBLANG`, if `DBLANG` is set to a relative pathname
3. In `$INFORMIXDIR/$DBLANG`, if `DBLANG` is set to a relative pathname
4. In `$INFORMIXDIR/msg/en_us/0333`
5. In `$INFORMIXDIR/msg/en_us.8859-1`
6. In `$INFORMIXDIR/msg`
7. In `$INFORMIXDIR/msg/english`

For more information on search paths for messages, see the description of `DBLANG` in the *IBM Informix: GLS User's Guide*.

DBMONEY

The `DBMONEY` environment variable specifies the display format of values in columns of `smallfloat`, `FLOAT`, `DECIMAL`, or `MONEY` data types, and of complex data types derived from any of these data types.



- \$ is a currency symbol that precedes MONEY values in the default locale if no other *front* symbol is specified, or if **DBMONEY** is not set.
- , or . is a comma or period (the default) that separates the integral part from the fractional part of the FLOAT, DECIMAL, or MONEY value. Whichever symbol you do not specify becomes the thousands separator.
- back* is a currency symbol that follows the MONEY value.
- front* is a currency symbol that precedes the MONEY value.

The *back* symbol can be up to seven characters and can contain any character that the locale supports, except a digit, a comma (,), or a period (.) symbol. The *front* symbol can be up to seven characters and can contain any character that the locale supports except a digit, a comma (,), or a period (.) symbol. If you specify any character that is not a letter of the alphabet for *front* or *back*, you must enclose the *front* or *back* setting between single quotation (') marks.

When you display MONEY values, IBM Informix products use the **DBMONEY** setting to format the output. **DBMONEY** has no effect, however, on the internal format of data values that are stored in columns of the database.

If you do not set **DBMONEY**, then MONEY values for the default locale, U.S. English, are formatted with a dollar sign (\$) that precedes the MONEY value, a period (.) that separates the integral from the fractional part of the MONEY value, and no *back* symbol. For example, 100.50 is formatted as \$100.50.

Suppose you want to represent MONEY values as DM (deutsche mark) units, using the currency symbol DM and comma (,) as the decimal separator. Enter the following command to set the **DBMONEY** environment variable:

```
setenv DBMONEY DM,
```

Here DM is the *front* currency symbol that precedes the MONEY value, and a comma separates the integral from the fractional part of the MONEY value. As a result, the value 100.50 is displayed as DM100,50.

For more information about how **DBMONEY** formats MONEY values in nondefault locales, see the *IBM Informix: GLS User's Guide*.

DBNLS (IDS)

The **DBNLS** environment variable specifies whether automatic data type conversion is supported between NCHAR and NVARCHAR database columns and CHAR and VARCHAR variables (respectively) of client systems.

Global Language Support (GLS) does not require the **DBNLS** environment variable. But Dynamic Server databases continue to support the legacy behavior of **DBNLS**, which supports applications that manipulate tables with NCHAR or NVARCHAR columns.

```
▶—setenv—DBNLS—▶
      ┌──'1'──┐
      └──'2'──┘
```

For UNIX systems that use the C shell, the following command line enables client applications such as DB-Access, IBM Informix SQL, IBM Informix 4GL, IBM Informix Dynamic 4GL, and embedded-SQL applications such as ESQL/C or ESQL/COBOL to convert automatically between CHAR and VARCHAR variables of the client application and NCHAR and NVARCHAR columns of the database:

```
setenv DBNLS 1
```

This setting also supports the automatic conversion of values retrieved from NCHAR columns into CHAR variables, and the conversion of NVARCHAR column values into VARCHAR variables.

Similarly, when **DBNLS** = 1, character strings stored as CHAR variables can be inserted into NCHAR columns, and character strings stored as VARCHAR variables can be inserted into NVARCHAR database columns.

To support these features, **DBNLS** must also be set to 1 on the client system. This setting also enables the client system to display dates, numbers, and currency values in formats specified on the client locale.

Conversely, each of the following command lines disables automatic conversion between CHAR and VARCHAR variables of the client application and NCHAR and NVARCHAR columns of the database and also prevents Dynamic Server from using the locale files of the client system:

```
setenv DBNLS
```

```
unsetenv DBNLS
```

On UNIX systems that use the C shell, either of these commands disables automatic conversion to and from NCHAR and NVARCHAR data values (by setting no value for **DBNLS**).

Another possible setting for **DBNLS** is 2. If you enter at the command line
setenv DBNLS 2

then automatic data type conversion between NCHAR and CHAR and between NVARCHAR and VARCHAR is supported (if the client system has **DBNLS** set to 1 or 2), but the database server can have a different locale from the client system.

DBONPLOAD (IDS)

The **DBONPLOAD** environment variable specifies the name of the database that the **onpload** utility of the High-Performance Loader (HPL) uses. If **DBONPLOAD** is set, **onpload** uses the specified name as the name of the database; otherwise, the default name of the database is **onpload**.

▶▶—setenv—DBONPLOAD—*dbname*—————▶▶

dbname specifies the name of the database that the **onpload** utility uses.

For example, to specify the name **load_db** as the name of the database, enter the following command:

```
setenv DBONPLOAD load_db
```

For more information, see the *IBM Informix: High-Performance Loader User's Guide*.

DBPATH

The **DBPATH** environment variable identifies database servers that contain databases. **DBPATH** can also specify a list of directories (in addition to the current directory) in which DB-Access looks for command scripts (**.sql** files).

The **CONNECT DATABASE**, **START DATABASE**, and **DROP DATABASE** statements use **DBPATH** to locate the database under two conditions:

- If the location of a database is not explicitly stated
- If the database cannot be located in the default server

The **CREATE DATABASE** statement does not use **DBPATH**.

To add a new **DBPATH** entry to existing entries, see “Modifying an Environment-Variable Setting” on page 3-8.

▶▶—setenv—DBPATH—*pathname*—————▶▶
┌: [16] ───────────────────────────────────┐
└─/ /—*servername*—/—*full_pathname*—┘
└─/ /—*servername*—┘

<i>full_pathname</i>	is the full path, from root , of a directory where .sql files are stored.
<i>pathname</i>	is the valid relative path of a directory where .sql files are stored.
<i>servername</i>	is the name of an Informix database server where databases are stored. You cannot reference database files with a <i>servername</i> .

DBPATH can contain up to 16 entries. Each entry must be less than 128 characters. In addition, the maximum length of **DBPATH** depends on the hardware platform on which you set **DBPATH**.

When you access a database with the **CONNECT**, **DATABASE**, **START DATABASE**, or **DROP DATABASE** statement, the search for the database is done first in the directory or database server specified in the statement. If no database server is specified, the default database server that was specified by the **INFORMIXSERVER** environment variable is used.

If the database is not located during the initial search, and if **DBPATH** is set, the database servers and directories in **DBPATH** are searched for in the specified database. These entries are searched in the same order in which they are listed in the **DBPATH** setting.

Using **DBPATH** with **DB-Access**

If you use **DB-Access** and select the **Choose** option from the **SQL** menu without having already selected a database, you see a list of all the **.sql** files in the directories listed in your **DBPATH**. Once you select a database, the **DBPATH** is not used to find the **.sql** files. Only the **.sql** files in the current working directory are displayed.

Searching Local Directories

Use a pathname without a database server name to search for **.sql** scripts on your local computer. In the following example, the **DBPATH** setting causes **DB-Access** to search for the database files in your current directory and then in the Joachim and Sonja directories on the local computer:

```
setenv DBPATH /usr/joachim:/usr/sonja
```

As the previous example shows, if the pathname specifies a directory name but not a database server name, the directory is sought on the computer that runs the default database server that the **INFORMIXSERVER** specifies; see “**INFORMIXSERVER**” on page 3-60. For instance, with the previous example, if **INFORMIXSERVER** is set to **quality**, the **DBPATH** value is *interpreted*, as the following example shows, where the double slash precedes the database server name:

```
setenv DBPATH //quality/usr/joachim://quality/usr/sonja
```

Searching Networked Computers for Databases

If you use more than one database server, you can set **DBPATH** explicitly to contain the database server and directory names that you want to search for databases. For example, if **INFORMIXSERVER** is set to **quality** but you also want to search the **marketing** database server for **/usr/joachim**, set **DBPATH** as the following example shows:

```
setenv DBPATH //marketing/usr/joachim:/usr/sonja
```

Specifying a Servername

You can set **DBPATH** to contain only database server names. This feature allows you to locate only databases; you cannot use it to locate command files.

The database administrator must include each database server mentioned by **DBPATH** in the **\$INFORMIXDIR/etc/sqlhosts** file. For information on communication-configuration files and dbservernames, see your *IBM Informix: Administrator's Guide* and the *IBM Informix: Administrator's Reference*.

For example, if **INFORMIXSERVER** is set to **quality**, you can search for a database first on the **quality** database server and then on the **marketing** database server by setting **DBPATH**, as the following example shows:

```
setenv DBPATH //marketing
```

If you use DB–Access in this example, the names of all the databases on the **quality** and **marketing** database servers are displayed with the **Select** option of the **DATABASE** menu.

DBPRINT

The **DBPRINT** environment variable specifies the default printing program.

```
►►—setenv—DBPRINT—program—————▶▶
```

program is any command, shell script, or UNIX utility that produces standard ASCII output.

If you do not set **DBPRINT**, the default *program* is found in one of two places:

- For most BSD UNIX systems, the default program is **lpr**.
- For UNIX System V, the default program is usually **lp**.

Enter the following command to set the **DBPRINT** environment variable to specify **myprint** as the print program:

```
setenv DBPRINT myprint
```

DBREMOTECMD (UNIX)

Set the **DBREMOTECMD** environment variable to override the default remote shell to perform remote tape operations with the database server. You can set **dbremotecmd** to a simple command or to a full pathname.

► setenv DBREMOTECMD command
pathname ►

command is a command to override the default remote shell.

pathname is a pathname to override the default remote shell.

If you do not specify the full pathname, the database server searches your **PATH** for the specified *command*. It is highly recommended that you use the full pathname syntax on interactive UNIX platforms to avoid problems with similarly named programs in other directories and possible confusion with the *restricted shell* (**/usr/bin/rsh**).

The following command sets **DBREMOTECMD** for a simple command name:

```
setenv DBREMOTECMD rcmd
```

The next command to set **DBREMOTECMD** specifies a full pathname:

```
setenv DBREMOTECMD /usr/bin/remsh
```

For more information on **DBREMOTECMD**, see the discussion in your *IBM Informix: Archive and Backup Guide* about how to use remote tape devices with your database server for archives, restores, and logical-log backups.

DBSPACETEMP

The **DBSPACETEMP** environment variable specifies the dbspaces in which temporary tables are built

You can list dbspaces, separated by colon (:) or comma (,) symbols to spread temporary space across any number of disks.

► setenv DBSPACETEMP temp_dspace ►

temp_dspace is the name of a valid existing temporary dbspace.

DBSPACETEMP overrides any default dbspaces that the **DBSPACETEMP** parameter specifies in the configuration file of the database server.

Important: The dbspaces that you list in **DBSPACETEMP** must be composed of chunks that are allocated as raw UNIX devices.

For example, the following command to set the **DBSPACETEMP** environment variable specifies three dbspaces for temporary tables:

```
setenv DBSPACETEMP sorttmp1:sorttmp2:sorttmp3
```

Separate the dspace entries with either colons or commas. The number of dbspaces is limited by the maximum size of the environment variable, as defined by your operating system. Your database server does not create a dspace specified by the environment variable if the dspace does not exist.

The two classes of temporary tables are *explicit* temporary tables that the user creates and *implicit* temporary tables that the database server creates. Use **DBSPACETEMP** to specify the dbspaces for both types of temporary tables.

If you create an explicit temporary table with the CREATE TEMP TABLE statement and do not specify a dspace for the table either in the IN *dbspace* clause or in the FRAGMENT BY clause, the database server uses the settings in **DBSPACETEMP** to determine where to create the table.

If you create an explicit temporary table with the SELECT INTO TEMP statement, the database server uses the settings in **DBSPACETEMP** to determine where to create the table.

If **DBSPACETEMP** is set, and the dbspaces that it lists include both logging and non-logging dbspaces, the database server stores temporary tables that implicitly or explicitly support transaction logging in a logged dspace, and non-logging temporary tables in a non-logging dspace.

The database server creates implicit temporary tables for its own use while executing join operations, SELECT statements with the GROUP BY clause, SELECT statements with the ORDER BY clause, and index builds.

When it creates explicit or implicit temporary tables, the database server uses disk space for writing the temporary data. If there are conflicts among settings or statement specifications for the location of a temporary table, these conflicts are resolved in this descending (highest to lowest) order of precedence:

1. What the IN or FRAGMENT BY clause of a DDL or DML statement specifies
2. For Extended Parallel Server, what a SET TEMP TABLE_SPACE statement specifies
3. On UNIX platforms, the operating-system directory or directories that the environment variable **PSORT_DBTEMP** specifies, if this is set
4. The dspace or dbspaces that the environment variable **DBSPACETEMP** specifies, if this is set
5. The dspace or dbspaces that the ONCONFIG parameter **DBSPACETEMP** specifies.

6. The dbspace or dbspaces that the ONCONFIG parameter TABLESPACE specifies.
7. The operating-system file space in **/tmp** (UNIX) or **%temp%** (Windows)
8. For Extended Parallel Server, in a non-critical space, if none of the above are specified
9. For Dynamic Server, in the space where the database was created, if none of the above are specified

Important: If the **DBSPACETEMP** environment variable is set to an invalid value, the database server defaults to the root dbspace for explicit temporary tables and to **/tmp** for implicit temporary tables, not to the **DBSPACETEMP** configuration parameter. In this situation, the database server might fill **/tmp** to the limit and eventually bring down the database server or kill the file system.

DBTEMP (IDS)

The **DBTEMP** environment variable is used by DB-Access and IBM Informix Enterprise Gateway products as well as by Dynamic Server and by earlier database servers. **DBTEMP** resembles **DBSPACETEMP**, specifying the directory in which to place temporary files and temporary tables.

►►—setenv—DBTEMP—*pathname*—◄◄

pathname is the full pathname of the directory for temporary files and tables.

For DB-Access to work correctly on Windows platforms, **DBTEMP** should be set to **\$INFORMIXDIR/infxtmp**.

The following example sets **DBTEMP** to the pathname **usr/magda/mytemp** for UNIX systems that use the C shell:

```
setenv DBTEMP usr/magda/mytemp
```

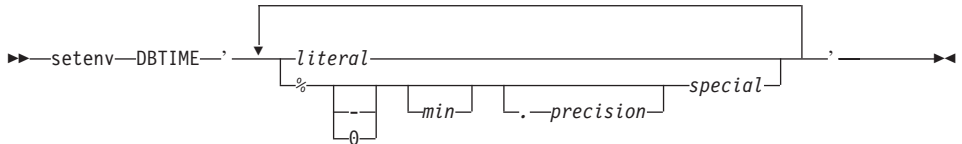
Important: **DBTEMP** can point to an NFS-mounted directory only if the vendor of that NFS device is certified by IBM. For information about NFS products for mounting storage space for an Informix database server, see the product compatibility data at <http://www.ibm.com/software/data/informix/pubs/smv/index.html>.

If **DBTEMP** is not set, the database server creates temporary files in the **/tmp** directory and temporary tables in the **DBSPACETEMP** directory. See “**DBSPACETEMP**” on page 3-36 for the default if **DBSPACETEMP** is not set. Similarly, if you do not set **DBTEMP** on the client system, temporary files (such as those created for scroll cursors) are created in the **/tmp** directory.

You might experience unexpected behavior or failure in operations on values of large or complex data types, such as BYTE or ROW, if DBTEMP is not set.

DBTIME

The **DBTIME** environment variable specifies a formatting mask for the display and data-entry format of DATETIME values.



- literal* is a literal white space or any printable character.
- min* is a literal integer, setting the minimum number of characters in the substring for the value that *special* specifies.
- precision* is the number of digits for the value of any time unit, or the maximum number of characters in the name of a month.
- special* is one of the placeholder characters that are listed following.

These terms and symbols are described in the pages that follow.

This quoted string can include literal characters as well as placeholders for the values of individual time units and other elements of a DATETIME value. **DBTIME** takes effect only when you call certain IBM Informix ESQL/C DATETIME routines. (For details, see the *IBM Informix: ESQL/C Programmer's Manual*.) If **DBTIME** is not set, the behavior of these routines is undefined, and "YYYY-MM-DD hh:mm:ss.fffff" is the default display and input format for DATETIME YEAR TO FRACTION(5) literal values in the default locale.

The percentage (%) symbol gives special significance to the *special* placeholder symbol that follows. Without a preceding % symbol, any character within the formatting mask is interpreted as a literal character, even if it is the same character as one of the placeholder characters in the following list. Note also that the *special* placeholder symbols are case sensitive.

The following characters within a **DBTIME** format string are placeholders for time units (or for other features) within a DATETIME value.

- %b** is replaced by the abbreviated month name.
- %B** is replaced by the full month name.
- %d** is replaced by the day of the month as a decimal number [01,31].

%Fn	is replaced by a fraction of a second with a scale that the integer <i>n</i> specifies. The default value of <i>n</i> is 2; the range of <i>n</i> is $0 \leq n \leq 5$.
%H	is replaced by the hour (24-hour clock).
%I	is replaced by the hour (12-hour clock).
%M	is replaced by the minute as a decimal number [00,59].
%m	is replaced by the month as a decimal number [01,12].
%p	is replaced by A.M. or P.M. (or the equivalent in the locale file).
%S	is replaced by the second as a decimal number [00,59].
%y	is replaced by the year as a four-digit decimal number.
%Y	is replaced by the year as a four-digit decimal number. User must enter a four-digit value.
%%	is replaced by % (to allow % in the format string).

For example, consider this display format for DATETIME YEAR TO SECOND:

```
Mar 21, 2001 at 16 h 30 m 28 s
```

If the user enters a two-digit year value, this value is expanded to 4 digits according to the **DBCENTURY** environment variable setting. If **DBCENTURY** is not set, then the string 19 is used by default for the first two digits.

Set **DBTIME** as the following command line (for the C shell) shows:

```
setenv DBTIME '%b %d, %Y at %H h %M m %S s'
```

The default **DBTIME** produces the following ANSI SQL string format:

```
2001-03-21 16:30:28
```

You can set the default **DBTIME** as the following example shows:

```
setenv DBTIME '%Y-%m-%d %H:%M:%S'
```

An optional field width and precision specification (*w.p*) can immediately follow the percent (%) character. It is interpreted as follows:

<i>w</i>	Specifies the minimum field width. The value is right-justified with blank spaces on the left.
<i>-w</i>	Specifies the minimum field width. The value is left-justified with blank spaces on the right.
<i>0w</i>	Specifies the minimum field width. The value is right-justified and padded with zeros on the left.

p Specifies the precision of d, H, I, m, M, S, y, and Y time unit values, or the maximum number of characters in b and B month names.

The following limitations apply to field-width and precision specifications:

- If the data value supplies fewer digits than *precision* specifies, the value is padded with leading zeros.
- If a data value supplies more characters than *precision* specifies, excess characters are truncated from the right.
- If no field width or precision is specified for d, H, I, m, M, S, or y placeholders, 0.2 is the default, or 0.4 for the Y placeholder.
- A *precision* specification is significant only when converting a DATETIME value to an ASCII string, but not vice versa.

The F placeholder does not support this field-width and precision syntax.

Like **DBDATE**, **GL_DATE**, or **GL_DATETIME**, the **DBTIME** setting controls only the character-string representation of data values; it cannot change the internal storage format of the DATETIME column. (For information about formatting DATE values, see the discussion of **DBDATE** on page 3-25.)

In East Asian locales that support era-based dates, **DBTIME** can also specify Japanese or Taiwanese eras. See *IBM Informix: GLS User's Guide* for details of additional placeholder symbols for setting **DBTIME** to display era-based DATETIME values, and for descriptions of the **GL_DATETIME** and **GL_DATE** environment variables.

DBUPSPACE

The **DBUPSPACE** environment variable lets you specify and constrain the amount of system disk space that the UPDATE STATISTICS statement can use when trying to simultaneously construct multiple column distributions.

►►—setenv—DBUPSPACE— *max*— :— *default*—◀◀

max is a positive integer, specifying the maximum disk space (in kilobytes) to allocate for sorting in UPDATE STATISTICS operations.

default is a positive integer, specifying the maximum amount of memory (in kilobytes) to allocate.

For example, to set **DBUPSPACE** to 2,500 kilobytes of disk space and 1,000 kilobytes of memory, enter this command:

```
setenv DBUPSPACE 2500:1
```

After you set this value, the database server can use no more than 2,500 kilobytes of disk space during the execution of an UPDATE STATISTICS statement. If a table requires 5 megabytes of disk space for sorting, then UPDATE STATISTICS accomplishes the task in two passes; the distributions for one half of the columns are constructed with each pass.

If you do not set **DBUPSPACE**, the default is a megabyte (1,024 kilobytes) for *max*, and the maximum amount of memory that is available without using PDQ for *default*. If you attempt to set **DBUPSPACE** to any value less than 1,024 kilobytes, it is automatically set to 1,024 kilobytes, but no error message is returned. If this value is not large enough to allow more than one distribution to be constructed at a time, at least one distribution is done, even if the amount of disk space required to do this is more than what **DBUPSPACE** specifies.

DEFAULT_ATTACH

The **DEFAULT_ATTACH** environment variable supports the legacy behavior of Version 7.x of Dynamic Server, which required that only nonfragmented B-tree indexes on nonfragmented tables can be attached.

►—setenv—DEFAULT_ATTACH—1—◄

If **DEFAULT_ATTACH** is set to 1, then all other indexes that you create, including R-trees and UDR functional indexes, must be detached, unless you specify the IN TABLE keywords as the Storage clause of the CREATE INDEX statement. (An *attached index* is one that has the same distribution scheme as the table on which it is built. Any index that does not is a *detached index*.)

If **DEFAULT_ATTACH** is not set, then any CREATE INDEX statement that does not specify IN TABLE as its Storage clause creates detached indexes by default. This release of Dynamic Server can support attached indexes that were created by Version 7.x of Dynamic Server.

Important: Future releases of Informix database servers might not continue to support **DEFAULT_ATTACH**. Developing new applications that depend on this deprecated feature is not recommended.

DELIMIDENT

The **DELIMIDENT** environment variable specifies that strings enclosed between double quotation (") marks are delimited database identifiers.

►—setenv—DELIMIDENT—◄

No value is required; **DELIMIDENT** takes effect if it exists, and it remains in effect while it is on the list of environment variables.

Delimited identifiers can include white space (such as the phrase "Vitamin E") or can be identical to SQL keywords, (such as "TABLE" or "USAGE"). You can also use them to declare database identifiers that contain characters outside the default character set for SQL identifiers (such as "Column #6"). In the default locale, this set consists of letters, digits, and the underscore (_) symbol.

Database identifiers (also called *SQL identifiers*) are names for database objects, such as tables and columns. *Storage identifiers* are names for storage objects, such as dbspaces, blobspaces, and sbspaces (smart blob spaces). You cannot use **DELIMITED** to declare storage identifiers that contain characters outside the default SQL character set.

Delimited identifiers are case sensitive. To use delimited identifiers, applications in ESQL/C must set **DELIMITED** at compile time and at runtime.

Warning: If **DELIMITED** is not already set, you should be aware that setting it can cause the failure of existing **.sql** scripts or client applications that use double (") quotation marks in contexts other than delimiting SQL identifiers, such as delimiters of string literals. You must use single (') rather than double quotation marks for delimited constructs that are not SQL identifiers if **DELIMITED** is set.

On UNIX systems that use the C shell and on which **DELIMITED** has been set, you can disable this feature (which causes anything between double quotes to be interpreted as an SQL identifier) by the command:

```
unsetenv DELIMITED
```

ENVIGNORE (UNIX)

The **ENVIGNORE** environment variable can deactivate specified environment variable settings in the common (shared) and private environment-configuration files, **informix.rc** and **.informix** respectively.



variable is the name of an environment variable to be deactivated.

Use colon (:) symbols between consecutive *variable* names. For example, to ignore the **DBPATH** and **DBMONEY** entries in the environment-configuration files, enter the following command:

```
setenv ENVIGNORE DBPATH:DBMONEY
```

The common environment-configuration file is stored in **\$INFORMIXDIR/etc/informix.rc**.

The private environment-configuration file is stored in the user's home directory as **.informix**.

For information on creating or modifying an environment-configuration file, see "Setting Environment Variables in a Configuration File" on page 3-6.

ENVIGNORE itself cannot be set in an environment-configuration file.

FET_BUF_SIZE

The **FET_BUF_SIZE** environment variable can override the default setting for the size of the fetch buffer for all data types except **BYTE** and **TEXT** values.

►—setenv—FET_BUF_SIZE—*size*—◄

size is a positive integer that is larger than the default buffer size, but no greater than 32,767, specifying the size (in bytes) of the fetch buffer that holds data retrieved by a query.

For example, to set a buffer size to 5,000 bytes on a UNIX system that uses the C shell, set **FET_BUF_SIZE** by entering the following command:

```
setenv FET_BUF_SIZE 5000
```

When **FET_BUF_SIZE** is set to a valid value, the new value overrides the default value (or any previously set value of **FET_BUF_SIZE**). The default setting for the fetch buffer is dependent on row size.

The processing of **BYTE** and **TEXT** values is not affected by **FET_BUF_SIZE**.

No error is raised if **FET_BUF_SIZE** is set to a value that is less than the default size or that is out of the range of **SMALLINT** values. In these cases, however, the invalid fetch buffer size is ignored, and the default size is in effect.

A valid **FET_BUF_SIZE** setting is in effect for the local database server as well as for any remote database server from which you retrieve rows through a distributed query in which the local server is the coordinator and the remote database is subordinate. The greater the size of the buffer, the more rows can be returned, and the less frequently the client application must wait while the database server returns rows. A large buffer can improve performance by reducing the overhead of filling the client-side buffer.

GLOBAL_DETACH_INFORM (XPS)

All indexes in IBM Informix Extended Parallel Server are detached. An XPS index is *locally detached* when every index fragment resides on the same coserver as its associated data fragment and an XPS index is *globally detached* if it is fragmented, with index items and their associated data rows residing on different coservers.

You should avoid using globally detached indexes because they are inherently less efficient than locally detached indexes.

The **GLOBAL_DETACH_INFORM** environment variable triggers an alarm if a globally detached index is created. The alarm has a severity of 3 (Attention), a Class ID of 10 (Performance Improvement Possible) and a Tag ID of 1 (Globally Detached Index Built).

To enable this alarm, set **GLOBAL_DETACH_INFORM** to any value before starting the server.

```
▶▶—setenv—GLOBAL_DETACH_INFORM—n—————▶▶
```

Alternatively, you can turn this variable on or off with the **onutil** SET command, as in the following example:

```
% onutil  
1> SET GLOBAL_DETACH_INFORM 1;  
Dynamic Configuration completed successfully
```

Because **GLOBAL_DETACH_INFORM** is an environment variable and not a configuration parameter, however, it cannot be made persistent with the **onutil** command. Add the variable to an environment-configuration file to avoid setting it each time the server is restarted.

IBM_XPS_PARAMS (XPS)

By default, the **CURRENT** and **TODAY** functions return values from the system clock-calendar, based on the location of the server. Use the **IBM_XPS_PARAMS** environment variable to specify a non-default time zone, as an offset from Greenwich Mean Time (GMT), for values returned by **CURRENT** and **TODAY**.

To specify an offset from GMT for the built-in **CURRENT** and **TODAY** functions, set **IBM_XPS_PARAMS** on the client system to the desired value before you start the client application, using the following syntax.

```
▶▶—setenv—IBM_XPS_PARAMS—'CLIENT_TZ— = — [ + ] — hours — : — [ - ] — minutes — '—————▶▶
```

hours is an integer in the range from 0 to 13 inclusive, specifying the absolute number of hours in the offset from GMT.

minutes is a 2-digit integer in the range from 00 to 59 inclusive, specifying any additional minutes in the offset from GMT.

A minus (-) sign before the *hours* specifies a time zone east of GMT, and a positive (*) sign, which is the default, specifies a time zone west of GMT.

This example specifies that CURRENT and TODAY return GMT values:

```
% setenv IBM_XPS_PARAMS 'CLIENT_TZ = 00:00'
```

The next example specifies the Atlantic time zone of eastern Canada:

```
% setenv IBM_XPS_PARAMS 'CLIENT_TZ = +4:00'
```

The **onstat -g ses** command can display the current offset from GMT.

The SET ENVIRONMENT CLIENT_TZ statement of SQL can override the **IBM_XPS_PARAMS** setting, but the default scope of this environment variable is all sessions, rather than only the session in which the SET ENVIRONMENT CLIENT_TZ statement is issued. Reset the GMT offset with the SQL statement for the current session if your application requires a different time zone.

IFMX_CART_ALRM (XPS)

The **IFMX_CART_ALRM** environment variable triggers an alarm if a query executes a Cartesian join. The alarm has a severity of 3 (Attention), a Class ID of 10 (Performance Improvement Possible) and a Tag ID of 2 (Cartesian Join Processing). The alarm message indicates the ID of the session executing the Cartesian join.

To enable this alarm, set **IFMX_CART_ALRM** to any value before starting the database server.

```
►►setenv—IFMX_CART_ALRM—n—————◄◄
```

Alternatively, you can turn this variable on or off with the **onutil SET** command:

```
% onutil
1> SET IFMX_CART_ALRM 1;
Dynamic Configuration completed successfully
```

Because **IFMX_CART_ALRM** is an environment variable and not a configuration parameter, it cannot be made persistent by the **onutil** command. Add the variable to an environment-configuration file to avoid setting it each time the server is restarted.

IFMX_HISTORY_SIZE (XPS)

The **IFMX_HISTORY_SIZE** environment variable determines the number of SQL commands that are logged in the DB-Access command history.

►►—setenv—IFMX_HISTORY_SIZE—*value*—◄◄

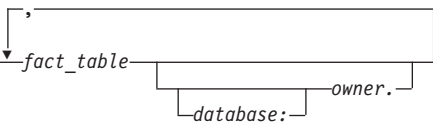
value the number of commands stored in the DB-Access history

The default value is 10. The maximum is 100. If a value greater or lower is specified, the default value is used. For more information on using the DB-Access history command, see the *IBM Informix: DB-Access User's Guide*.

IFMX_OPT_FACT_TABS (XPS)

The **IFMX_OPT_FACT_TABS** environment variable specifies a list of fact tables that should be used in push-down hash joins whenever possible.

►►—setenv—IFMX_OPT_FACT_TABS—*fact_table*—◄◄



database is name of the database.

fact_table is name of the fact table.

owner is name of the table owner.

If you do not specify a database name or owner, the fact table can be in any database or belong to any owner.

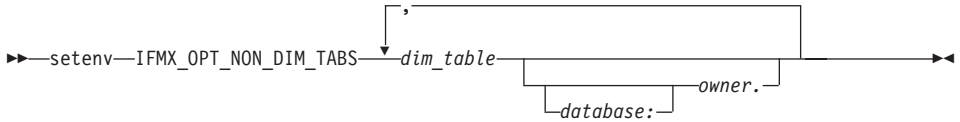
The environment variable lists fact tables for which you want to encourage the optimizer to choose push-down hash-join plans. If you do not specify the database name or owner, the table can be in any database or belong to any owner.

When this environment variable is set, push-down hash-join restrictions for the specified fact tables are relaxed to allow the optimizer to choose a push-down plan even when the fact table is not larger than the dimension table or when the dimension-table join columns are not unique.

You can use **IFMX_OPT_FACT_TABS** alone to increase the possibility of push-down hash joins. You can also use it in conjunction with the **IFMX_OPT_NON_DIM_TABS** environment variable to fine tune the use of push-down hash joins.

IFMX_OPT_NON_DIM_TABS (XPS)

The **IFMX_OPT_NON_DIM_TABS** environment variable specifies a list of dimension tables that *cannot* be used in push-down hash-join query plans. If the optimizer detects a fact-dimension table query that joins one of these dimension tables, it does not choose a push-down hash-join plan.



database is name of a database.

dim_table is name of a dimension table.

owner is name of table owner.

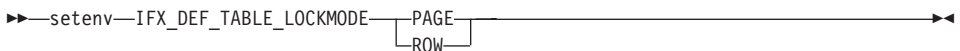
If the database name or owner is not specified, the table can be in any database or can belong to any owner.

When this environment variable is set, if a query joins one of the dimension tables in this list with any fact table, the optimizer never selects a push-down hash join for the query, even if the fact table is included in the **IFMX_OPT_FACT_TABS** list.

You can use the **IFMX_OPT_NON_DIM_TABS** environment variable alone to decrease the possibility of push-down hash joins. You can also use it in conjunction with the **IFMX_OPT_FACT_TABS** environment variable to fine tune the use of push-down hash joins.

IFX_DEF_TABLE_LOCKMODE (IDS)

The **IFX_DEF_TABLE_LOCKMODE** environment variable can specify the default lock mode for database tables that are subsequently created without explicitly specifying the **LOCKMODE PAGE** or **LOCKMODE ROW** keywords. This feature is convenient if you need to create several tables of the same lock mode. UNIX systems that use the C shell support the following syntax:



PAGE The default lock mode is page-level granularity.

ROW The default lock mode is row-level granularity.

Similar functionality is available by setting the **DEF_TABLE_LOCKMODE** parameter of the **ONCONFIG** file to **PAGE** or **ROW**. When a table is created

or modified, any conflicting lock mode specifications are resolved according to the following descending (highest to lowest) order of precedence:

1. Explicit LOCKMODE specification of CREATE TABLE or ALTER TABLE
2. **IFX_DEF_TABLE_LOCKMODE** environment variable setting
3. DEF_TABLE_LOCKMODE parameter setting in the ONCONFIG file
4. The system default lock mode (= page mode)

To make the DEF_TABLE_LOCKMODE setting the default mode (or to restore the system default if DEF_TABLE_LOCKMODE is not set) use the command:

```
unsetenv IFX_DEF_TABLE_LOCKMODE
```

If **IFX_DEF_TABLE_LOCKMODE** is set in the environment of the database server before starting **oninit**, then its scope is all sessions of the database server (just as if DEF_TABLE_LOCKMODE were set in the ONCONFIG file). If **IFX_DEF_TABLE_LOCKMODE** is set in the shell, or in the **\$HOME/.informix** or **\$INFORMIXDIR/etc/informix.rc** files, then the scope is restricted to the current session (if you set it in the shell) or to the individual user.

Important: This has no effect on existing tables. If you specify *ROW* as the lock mode, the database will use this to restore, recover, or copy data. For tables that were created in *PAGE* mode, this might cause lock-table overflow or performance degradation.

IFX_DIRECTIVES

The **IFX_DIRECTIVES** environment variable setting determines whether the optimizer allows query optimization directives from within a query. The **IFX_DIRECTIVES** environment variable is set on the client.

You can specify either ON and OFF or 1 and 0 to set the environment variable.

```
►►—setenv—IFX_DIRECTIVES—1
```

1 Optimizer directives accepted

0 Optimizer directives not accepted

The setting of the **IFX_DIRECTIVES** environment variable overrides the value of the DIRECTIVES configuration parameter that is set for the database server. If the **IFX_DIRECTIVES** environment variable is not set, however, then all client sessions will inherit the database server configuration for directives that the ONCONFIG parameter DIRECTIVES determines. The default setting for the **IFX_DIRECTIVES** environment variable is ON.

For more information about the **DIRECTIVES** parameter, see the *IBM Informix: Administrator's Reference*. For more information on the performance impact of directives, see your *IBM Informix: Performance Guide*.

IFX_EXTDIRECTIVES

The **IFX_EXTDIRECTIVES** environment variable specifies whether the query optimizer allows external query optimization directives from the **sysdirectives** system catalog table to be applied to queries in existing applications. The **IFX_EXTDIRECTIVES** environment variable is set on the client.

You can specify either ON and OFF or 1 and 0 to set the environment variable.

```
► setenv IFX_DIRECTIVES 1 ◀
```

- 1 External optimizer directives accepted
- 0 External optimizer directives not accepted

Queries within a given client application can use external directives if both the **EXT_DIRECTIVES** parameter in the configuration file of the database server and the **IFX_EXTDIRECTIVES** environment variable setting on the client system are both set to 1 or ON. If **IFX_EXTDIRECTIVES** is not set, external directives are supported only if the **ONCONFIG** parameter **EXT_DIRECTIVES** is set to 2. The following table summarizes the effect of valid **IFX_EXTDIRECTIVES** and **EXT_DIRECTIVES** settings on support for external optimizer directives.

Table 3-3. Effect of IFX_EXTDIRECTIVES and EXT_DIRECTIVES settings on external directives

	EXT_DIRECTIVES = 0	EXT_DIRECTIVES = 1	EXT_DIRECTIVES = 2
IFX_EXTDIRECTIVES No setting	OFF	OFF	ON
IFX_EXTDIRECTIVES 0 = OFF	OFF	OFF	OFF
IFX_EXTDIRECTIVES 1 = ON	OFF	ON	ON

The database server interprets any **EXT_DIRECTIVES** setting besides 1 or 2 (or no setting) as equivalent to OFF, disabling support for external directives. Any value of **IFX_EXTDIRECTIVES** other than 1 has the same effect for the client.

For information on how to define external optimizer directives, see the description of the **SAVE EXTERNAL DIRECTIVES** statement of SQL in the *IBM Informix: Guide to SQL Syntax*. For more information about the

EXT_DIRECTIVES configuration parameter, see the *IBM Informix: Administrator's Reference*. For more information on the performance impact of directives, see your *IBM Informix: Performance Guide*.

IFX_LONGID

The **IFX_LONGID** environment variable setting and the version number of the client application determine whether a given client application is capable of handling long identifiers. (Older versions of Informix databases restricted SQL identifiers to 18 or fewer bytes; *long identifiers* can have up to 128 bytes when **IFX_LONGID** is set.) Valid **IFX_LONGID** values are 1 and 0.

```
►—setenv—IFX_LONGID—1——►
                    |
                    |
                    0
```

- 1 Client supports long identifiers.
- 0 Client cannot support long identifiers.

When **IFX_LONGID** is set to zero, applications display only the first 18 bytes of long identifiers, without indicating (by +) that truncation has occurred.

If **IFX_LONGID** is unset or is set to a value other than 1 or 0, the determination is based on the internal version of the client application. If the (server-based) version is not less than 9.0304, or is in the (CSDK-based) range $2.90 \leq \text{version} < 4.0$, the client is considered capable of handling long identifiers. Otherwise, the client application is considered incapable.

The **IFX_LONGID** setting overrides the internal version of the client application. If the client cannot handle long identifiers despite a newer version number, set **IFX_LONGID** to 0. If the client version can handle long identifiers despite an older version number, set **IFX_LONGID** to 1.

If you set **IFX_LONGID** on the client, the setting affects only that client. If you bring up the database server with **IFX_LONGID** set, all client applications use that setting by default. If **IFX_LONGID** is set to different values on the client and on the database server, however, the client setting takes precedence.

Important: ESQL executables that have been built with the **-static** option using the **libos.a** library version that does not support long identifiers cannot use the **IFX_LONGID** environment variable. You must recompile such applications with the new **libos.a** library that includes support for long identifiers. Executables that use shared libraries (no **-static** option) can use **IFX_LONGID** without recompilation provided that they use the new **libifos.so** that provides support for long identifiers. For details, see your ESQL product manual.

IFX_NETBUF_PVTPOOL_SIZE (UNIX)

The **IFX_NETBUF_PVTPOOL_SIZE** environment variable specifies the maximum size of the free (unused) private network buffer pool for each database server session.

►—setenv—IFX_NETBUF_PVTPOOL_SIZE—*count*—◄

count is an integer specifying the number of units (buffers) in the pool.

The default size is 1 buffer. If **IFX_NETBUF_PVTPOOL_SIZE** is set to 0, then each session obtains buffers from the free global network buffer pool. You must specify the value in decimal form.

IFX_NETBUF_SIZE

The **IFX_NETBUF_SIZE** environment variable lets you configure the network buffers to the optimum size. It specifies the size of all network buffers in the free (unused) global pool and the private network buffer pool for each database server session.

►—setenv—IFX_NETBUF_SIZE—*size*—◄

size is the integer size (in bytes) for one network buffer.

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

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

IFX_NO_TIMELIMIT_WARNING

Trial or evaluation versions of IBM Informix software products, which cease to function when some time limit has elapsed since the software was installed, by default issue warning messages that tell users when the license will expire. If you set the **IFX_NO_TIMELIMIT_WARNING** environment variable, however, the time-limited software does not issue these warning messages.

►—setenv—IFX_NO_TIMELIMIT_WARNING—◄

For users who dislike viewing warning messages, this feature is an alternative to redirecting the error output. Setting **IFX_NO_TIMELIMIT_WARNING** has no effect, however, on when a time-limited license expires; the software ceases to function at the same point in time when it would if this environment variable had not been set. If you do set **IFX_NO_TIMELIMIT_WARNING**, users will not see potentially annoying warnings about the impending license

expiration, but some users might be annoyed at you when the database server (or whatever software has a time-limited license) ceases to function without any warning.

IFX_PAD_VARCHAR (IDS)

The **IFX_PAD_VARCHAR** environment variable setting controls how the database server sends and receives VARCHAR and NVARCHAR data values. Valid **IFX_PAD_VARCHAR** values are 1 and 0.

```
►►—setenv—IFX_PAD_VARCHAR—

|   |
|---|
| 1 |
| 0 |

—————►►
```

- 1 Transmit the entire structure, up to the declared *max* size.
- 0 Transmit only the portion of the structure containing data.

For example, to send the string "ABC" from a column declared as NVARCHAR(255) when **IFX_PAD_VARCHAR** is set to 0 would send 3 bytes.

If the setting were 1 in the previous example, however, the number of bytes sent would be 255 bytes.

The effect **IFX_PAD_VARCHAR** is context-sensitive. In a low-bandwidth network, a setting of 0 might improve performance by reducing the total volume of transmitted data. But in a high-bandwidth network, a setting of 1 might improve performance, if the CPU time required to process variable-length packets were greater than the time required to send the entire character stream.

IFX_UPDDESC (IDS)

You must set the **IFX_UPDDESC** environment variable at execution time before you can do a DESCRIBE of an UPDATE statement.

```
►►—setenv—IFX_UPDDESC—value—————►►
```

value is any non-NULL value.

A NULL value (here meaning that **IFX_UPDDESC** is not set) disables the describe-for-update feature. Any non-NULL value enables the feature.

IFX_XASTDCOMPLIANCE_XAEND

In earlier releases of IBM Informix database servers, an internal rollback of a global transaction freed the transaction. In releases later than XPS 8.40 and IDS 9.40, however, the default behavior after an internal rollback is not to free the global transaction until an explicit rollback, as required by the X/Open

XA standard. By setting the `DISABLE_B162428_XA_FIX` configuration parameter to 1, you can restore the legacy behavior as the default for all sessions.

The `IFX_XASTDCOMPLIANCE_XAEND` environment variable can override the configuration parameter for the current session, using the following syntax. Valid `IFX_XASTDCOMPLIANCE_XAEND` values are 1 and 0.

```
►►—setenv—IFX_XASTDCOMPLIANCE_XAEND—1—————►►  
                                          0
```

- | | |
|---|---|
| 0 | Frees global transactions only after an explicit rollback |
| 1 | Frees global transactions after any rollback |

This environment variable can be particularly useful when the server instance is disabled for new behaviour by the `DISABLE_B162428_XA_FIX` configuration parameter, but one client requires the new behaviour. Setting this environment variable to zero supports the new behaviour in the current session.

IMCADMIN

The `IMCADMIN` environment variable supports the `imcadmin` administrative tool by specifying the name of a database server through which `imcadmin` can connect to MaxConnect. For `imcadmin` to operate correctly, you must set `IMCADMIN` before you use an IBM Informix product.

```
►►—setenv—IMCADMIN—dbservername—————►►
```

dbservername is the name of a database server.

Here *dbservername* must be listed in the `sqlhosts` file on the computer where the MaxConnect runs. MaxConnect uses this setting to obtain the following connectivity information from the `sqlhosts` file:

- Where the administrative listener port must be established
- The network protocol that the specified database server uses
- The host name of the system where the specified database server resides

You cannot use the `imcadmin` tool unless `IMCADMIN` is set to a valid database server name.

For more information about using `IMCADMIN`, refer to *IBM Informix: MaxConnect User's Guide*.

IMCCONFIG

The **IMCCONFIG** environment variable specifies a nondefault filename, and optionally a pathname, for the MaxConnect configuration file. On UNIX systems that support the C shell, this variable can be set by the following command.

```
▶▶—setenv—IMCCONFIG—pathname—————▶▶
```

pathname is a full pathname or a simple filename.

When the setting is a filename that is not qualified by a full pathname, MaxConnect searches for the specified file in the **\$INFORMIXDIR/etc/** directory. Thus, if you set **IMCCONFIG** to **IMCconfig.imc2**, MaxConnect searches for **\$INFORMIXDIR/etc/IMCconfig.imc2** as its configuration file.

If the **IMCCONFIG** environment variable is not set, MaxConnect searches by default for **\$INFORMIXDIR/etc/IMCconfig** as its configuration file.

IMCSERVER

The **IMCSERVER** environment variable specifies the name of a database server entry in the **sqlhosts** file that contains information on connectivity.

The database server can be either local or remote. On UNIX systems that support the C shell, the **IMCSERVER** environment variable can be set by the command.

```
▶▶—setenv—IMCSERVER—dbservername—————▶▶
```

dbservername is the valid name of a database server.

Here *dbservername* must be the name of a database server in the **sqlhosts** file. For more information about **sqlhosts** settings with MaxConnect, see your *IBM Informix: Administrator's Guide*. You cannot use MaxConnect unless **IMCSERVER** is set to a valid database server name.

INFORMIXC (UNIX)

The **INFORMIXC** environment variable specifies the filename or pathname of the C compiler to be used to compile files that IBM Informix ESQ/C generates. The setting takes effect only during the C compilation stage.

If **INFORMIXC** is not set, the default compiler on most systems is **cc**.

Tip: On Windows, you pass either *-mcc* or *-bcc* options to the *esql* preprocessor to use either the Microsoft or Borland C compilers.

►—setenv—INFORMIXC—compiler
pathname—►

compiler is the filename of the C compiler.

pathname is the full pathname of the C compiler.

For example, to specify the GNU C compiler, enter the following command:

```
setenv INFORMIXC gcc
```

Important: If you use **gcc**, be aware that the database server assumes that strings are writable, so you need to compile using the **-fwritable-strings** option. Failure to do so can produce unpredictable results, *possibly including core dumps*.

INFORMIXCONCSMCFG (IDS)

The **INFORMIXCONCSMCFG** environment variable specifies the location of the **concsm.cfg** file that describes communications support modules.

►—setenv—INFORMIXCONCSMCFG—*pathname*—►

pathname specifies the full pathname of the **concsm.cfg** file.

The following command specifies that the **concsm.cfg** file is in **/usr/myfiles**:

```
setenv INFORMIXCONCSMCFG /usr/myfiles
```

You can also specify a different name for the file. The following example specifies a filename of **csconfig** in the same directory:

```
setenv INFORMIXCONCSMCFG /usr/myfiles/csmconfig
```

The default location of the **concsm.cfg** file is in **\$INFORMIXDIR/etc**. For more information about communications support modules and the contents of the **concsm.cfg** file, refer to the *IBM Informix: Administrator's Reference*.

INFORMIXCONRETRY

The **INFORMIXCONRETRY** environment variable sets the maximum number of *additional* connection attempts that should be made to each database server by the client during the time limit that **INFORMIXCONTIME** specifies.

►—setenv—INFORMIXCONRETRY—*count*—►

count is the number of additional attempts to connect to each database server.

For example, the following command sets **INFORMIXCONRETRY** to specify three additional connection attempts (after the initial attempt):

```
setenv INFORMIXCONRETRY 3
```

The default value for **INFORMIXCONRETRY** is one retry after the initial connection attempt. The **INFORMIXCONTIME** setting, described in the following section, takes precedence over the **INFORMIXCONRETRY** setting.

INFORMIXCONTIME

The **INFORMIXCONTIME** environment variable specifies for how many seconds the **CONNECT** statement continues each attempt to establish a connection to a database server before returning an error. If you set no value, the default of 60 seconds can typically support a few hundred concurrent client connections, but some systems might encounter very few connection errors with a value as low as 15. The total distance between nodes, hardware speed, the volume of traffic, and the concurrency level of the network can all affect what value you should set to optimize **INFORMIXCONTIME**.

The **INFORMIXCONTIME** and **INFORMIXCONRETRY** environment variables let you configure your client-side connection capability to retry the connection instead of returning a **-908** error.

►—setenv—INFORMIXCONTIME—seconds—◄

seconds represents the minimum number of seconds spent in attempts to establish a connection to a database server.

For example, enter this command to set **INFORMIXCONTIME** to 60 seconds:

```
setenv INFORMIXCONTIME 60
```

If **INFORMIXCONTIME** is set to 60 and **INFORMIXCONRETRY** is set to 3, attempts to connect to the database server (after the initial attempt at 0 seconds) are made at 20, 40, and 60 seconds, if necessary, before aborting. This 20-second interval is the result of **INFORMIXCONTIME** divided by **INFORMIXCONRETRY**. If you attempt to set **INFORMIXCONTIME** to zero, the database server automatically resets it to the default value of 60 seconds.

If execution of the **CONNECT** statement involves searching **DBPATH**, the following rules apply:

- All appropriate servers in the **DBPATH** setting are accessed at least once, even though the **INFORMIXCONTIME** value might be exceeded. Thus, the **CONNECT** statement might take longer than the **INFORMIXCONTIME** time limit to return an error that indicates connection failure or that the database was not found.
- **INFORMIXCONRETRY** specifies how many additional connection attempts should be made for each database server entry in **DBPATH**.

- The **INFORMIXCONTIME** value is divided among the number of database server entries specified in **DBPATH**. Thus, if **DBPATH** contains numerous servers, you should increase the **INFORMIXCONTIME** value accordingly. For example, if **DBPATH** contains three entries, to spend at least 30 seconds attempting each connection, set **INFORMIXCONTIME** to 90.

INFORMIXCONTIME takes precedence over the **INFORMIXCONRETRY** setting. Retry efforts could end after the **INFORMIXCONTIME** value is exceeded, but before the **INFORMIXCONRETRY** value is reached.

The **INFORMIXCONTIME** and **INFORMIXCONRETRY** environment variables can be modified with the **onutil** SET command, as in the following example:

```
% onutil
1> SET INFORMIXCONTIME 120;
Dynamic Configuration completed successfully
2> SET INFORMIXCONRETRY 10;
Dynamic Configuration completed successfully
```

INFORMIXCPPMAP (IDS)

Set the **INFORMIXCPPMAP** environment variable to specify the fully qualified pathname of the map file for C++ programs. Information in the map file includes the database server type, the name of the shared library that supports the database object or value object type, the library entry point for the object, and the C++ library for which an object was built.

```
►►—setenv—INFORMIXCPPMAP—pathname—◄◄
```

pathname is the directory path where the C++ map file is stored.

The map file is a text file that can have any filename. You can specify several map files, separated by colons (:) on UNIX or semicolons (;) on Windows.

On UNIX, the default map file is **\$INFORMIXDIR/etc/c++map**. On Windows, the default map file is **%INFORMIXDIR%\etc\c++map**.

INFORMIXDIR

The **INFORMIXDIR** environment variable specifies the directory that contains the subdirectories in which your product files are installed. You must always set **INFORMIXDIR**. Verify that **INFORMIXDIR** is set to the full pathname of the directory in which you installed your database server. If you have multiple versions of a database server, set **INFORMIXDIR** to the appropriate directory name for the version that you want to access. For information about when to set **INFORMIXDIR**, see your *IBM Informix: Installation Guide*.

```
►►—setenv—INFORMIXDIR\pathname—◄◄
```

pathname is the directory path where the product files are installed.

To set **INFORMIXDIR** to **usr/informix/**, for example, as the installation directory, enter the following command:

```
setenv INFORMIXDIR /usr/informix
```

INFORMIXKEYTAB (UNIX)

The **INFORMIXKEYTAB** environment variable specifies the location of the **keytab** file. The **keytab** file contains authentication information that database servers and clients access at connection time, if they use the DCE-GSS communications support module (CSM). It contains key tables that store keys, each of which contains a principal name (database server or user name), type, version, and value.

The database server uses the **keytab** file to find the key to register the database server and to acquire a credential for it. A client application uses the key if the user did not execute **dce_login** with the current operating-system user name (which is the same as the DCE principal name) or did not explicitly provide a credential.

```
►—setenv—INFORMIXKEYTAB—pathname—◄
```

pathname specifies the full path of the **keytab** file.

For example, the following command specifies that the name and location of the **keytab** file is **/usr/myfiles/mykeytab**:

```
setenv INFORMIXKEYTAB /usr/myfiles/mykeytab
```

For more information about the DCE-GSS communications support module, see the *IBM Informix: Administrator's Guide*.

INFORMIXOPCACHE (IDS)

The **INFORMIXOPCACHE** environment variable can specify the size of the memory cache for the staging-area blob space of the client application.

```
►—setenv—INFORMIXOPCACHE—kilobytes—◄
```

kilobytes specifies the value you set for the optical memory cache.

Set the **INFORMIXOPCACHE** environment variable by specifying the size of the memory cache in kilobytes. The specified size must be equal to or smaller than the size of the system-wide configuration parameter, **OPCACHEMAX**.

If you do not set **INFORMIXOPCACHE**, the default cache size is 128 kilobytes or the size specified in the configuration parameter **OPCACHEMAX**. The default for **OPCACHEMAX** is 128 kilobytes. If you set **INFORMIXOPCACHE** to a value of 0, Optical Subsystem does not use the cache.

INFORMIXSERVER

The **INFORMIXSERVER** environment variable specifies the default database server to which an explicit or implicit connection is made by an SQL API client, the DB–Access utility, or other IBM Informix products. This must be set before you can use IBM Informix client products. It has the following syntax.

```
►—setenv—INFORMIXSERVER—dbservername—◄
```

dbservername is the name of the default database server.

The value of **INFORMIXSERVER** can be a local or remote server, but must correspond to a valid *dbservername* entry in the **\$INFORMIXDIR/etc/sqlhosts** file on the computer running the application. The *dbservername* must begin with a lower-case letter and cannot exceed 128 bytes. It can include any printable characters except uppercase characters, field delimiters (blank space or tab), the newline character, and the hyphen (or minus) symbol.

For example, this command specifies the **coral** database server as the default:

```
setenv INFORMIXSERVER coral
```

INFORMIXSERVER specifies the database server to which an application connects if the **CONNECT DEFAULT** statement is executed. It also defines the database server to which an initial implicit connection is established if the first statement in an application is not a **CONNECT** statement.

Important: You must set **INFORMIXSERVER** even if the application or DB–Access does not use implicit or explicit default connections.

For Extended Parallel Server, the **INFORMIXSERVER** environment variable specifies the name of a *dbserver* group. To specify a *coserver* name, use the following format:

```
dbservername.coserver_number
```

Here *dbservername* is the value that you assigned to the **DBSERVERNAME** configuration parameter in the **ONCONFIG** configuration file and *coserver_number* is the value that you assigned to the **COSERVER** configuration parameter for the connection *coserver*.

Strictly speaking, **INFORMIXSERVER** is not required for initialization. If **INFORMIXSERVER** is not set, however, Extended Parallel Server does not build the **sysmaster** tables.

INFORMIXSHMBASE (UNIX)

The **INFORMIXSHMBASE** environment variable affects only client applications connected to Informix databases that use the interprocess communications (IPC) shared-memory (**ipcshm**) protocol.

Important: Resetting **INFORMIXSHMBASE** requires a thorough understanding of how the application uses memory. Normally you do not reset **INFORMIXSHMBASE**.

INFORMIXSHMBASE specifies where shared-memory communication segments are attached to the client process so that client applications can avoid collisions with other memory segments that it uses. If you do not set **INFORMIXSHMBASE**, the memory address of the communication segments defaults to an implementation-specific value such as 0x800000.

►—setenv—**INFORMIXSHMBASE**—*value*—►

value is an integer (in kilobytes) used to calculate the memory address.

The database server calculates the memory address where segments are attached by multiplying the value of **INFORMIXSHMBASE** by 1,024. For example, on a system that uses the C shell, you can set the memory address to the value 0x800000 by entering the following command:

```
setenv INFORMIXSHMBASE 8192
```

For more information, see your *IBM Informix: Administrator's Guide* and the *IBM Informix: Administrator's Reference*.

INFORMIXSQLHOSTS

The **INFORMIXSQLHOSTS** environment variable specifies where the SQL client or the database server can find connectivity information.

►—setenv—**INFORMIXSQLHOSTS**—*pathname*—►

pathname is the full pathname of the connectivity information file.

On UNIX systems, the default search path for the connectivity information file is **\$INFORMIXDIR/etc/sqlhosts**.

The following command overrides this default to specify the **mysqlhosts** file in the **/work/envt** directory:

```
setenv INFORMIXSQLHOSTS /work/envt/mysqlhosts
```

On Windows, **INFORMIXSQLHOSTS** points to the computer whose registry contains the **SQLHOSTS** subkey.

The next example specifies that the client or database server look for connectivity information on a computer named **arizona**:

```
set INFORMIXSQLHOSTS = \\arizona
```

For details of the information that **sqlhosts** (or a file with a non-default filename) can provide about connectivity, see your *IBM Informix: Administrator's Guide*.

INFORMIXSTACKSIZE

The **INFORMIXSTACKSIZE** environment variable specifies the stack size (in kilobytes) that the database server uses for the primary thread of a client session. You can use **INFORMIXSTACKSIZE** to override the value of the **ONCONFIG** parameter **STACKSIZE** for a given application or user.

►—setenv—**INFORMIXSTACKSIZE**—*size*—◄

size is an integer, setting the stack size (in kilobytes) for SQL client threads.

For example, to decrease the **INFORMIXSTACKSIZE** to 20 kilobytes, enter the following command:

```
setenv -STACKSIZE 20
```

If **INFORMIXSTACKSIZE** is not set, the stack size is taken from the database server configuration parameter **STACKSIZE** or else defaults to a platform-specific value. The default stack size value for the primary thread of an SQL client is 32 kilobytes for nonrecursive database activity.

Warning: For instructions on setting this value, see the *IBM Informix: Administrator's Reference*. If you incorrectly set the value of **INFORMIXSTACKSIZE**, it can cause the database server to fail.

INFORMIXTERM (UNIX)

The **INFORMIXTERM** environment variable specifies whether DB–Access should use the information in the **termcap** file or the **terminfo** directory.

On character-based systems, the **termcap** file and **terminfo** directory determine terminal-dependent keyboard and screen capabilities, such as the operation of function keys, color and intensity attributes in screen displays, and the definition of window borders and graphic characters.

►—setenv—**INFORMIXTERM**—

termcap
terminfo

—◄

If **INFORMIXTERM** is not set, the default setting is **termcap**. When DB–Access is installed on your system, a **termcap** file is placed in the **etc** subdirectory of **\$INFORMIXDIR**. This file is a superset of an operating-system **termcap** file.

You can use the **termcap** file that the database server supplies, the system **termcap** file, or a **termcap** file that you create. You must set the **TERMCAP** environment variable if you do not use the default **termcap** file. For information on setting the **TERMCAP** environment variable, see page 3-78.

The **terminfo** directory contains a file for each terminal name that has been defined. The **terminfo** setting for **INFORMIXTERM** is supported only on computers that provide full support for the UNIX System V **terminfo** library. For details, see the machine notes file for your product.

INF_ROLE_SEP (IDS)

The **INF_ROLE_SEP** environment variable configures the security feature of role separation when the database server is installed or reinstalled on UNIX systems. Role separation enforces separating administrative tasks by people who run and audit the database server. After the installation is complete, **INF_ROLE_SEP** has no effect. If **INF_ROLE_SEP** is not set, then user **informix** (the default) can perform all administrative tasks.

```
►►—setenv—INF_ROLE_SEP—n—————►►
```

n is any positive integer.

On Windows, the install process asks whether you want to enable role separation regardless of the setting of **INF_ROLE_SEP**. To enable role separation for database servers on Windows, choose the role-separation option during installation.

If **INF_ROLE_SEP** is set when Dynamic Server is installed on a UNIX platform, role separation is implemented and a separate group is specified to serve each of the following responsibilities:

- The Database Server Administrator (DBSA)
- The Audit Analysis Officer (AAO)
- The standard user

On UNIX, you can establish role separation manually by changing the group that owns the **aaodir**, **dbسادir**, or **etc** directories at any time after the installation is complete. You can disable role separation by resetting the group that owns these directories to **informix**. You can have role separation enabled, for example, for the Audit Analysis Officer (AAO) without having role separation enabled for the Database Server Administrator (DBSA).

For more information about the security feature of role separation, see the *IBM Informix: Trusted Facility Guide*. To learn how to configure role separation when you install your database server, see your *IBM Informix: Installation Guide*.

INTERACTIVE_DESKTOP_OFF (Windows)

This environment variable lets you prevent interaction with the Windows desktop when an SPL routine executes a SYSTEM command.

```
►►—setenv—INTERACTIVE_DESKTOP_OFF—1  
                                          0—◄◄
```

If **INTERACTIVE_DESKTOP_OFF** is 1 and an SPL routine attempts to interact with the desktop (for example, with the **notepad.exe** or **cmd.exe** program), the routine fails unless the user is a member of the **Administrators** group.

The valid settings (1 or 0) have the following effects:

- | | |
|---|---|
| 1 | Prevents the database server from acquiring desktop resources for the user executing the stored procedure |
| 0 | SYSTEM commands in a stored procedure can interact with the desktop. This is the default value. |

Setting **INTERACTIVE_DESKTOP_OFF** to 1 allows an SPL routine that does not interact with the desktop to execute more quickly. This setting also allows the database server to simultaneously call a greater number of SYSTEM commands because the command no longer depends on a limited operating-system resource (Desktop and WindowStation handles).

ISM_COMPRESSION

Set this environment variable in the ON-Bar environment to specify whether the IBM Informix Storage Manager (ISM) should use data compression.

```
►►—setenv—ISM_COMPRESSION—TRUE  
                                          FALSE—◄◄
```

If **ISM_COMPRESSION** is set to TRUE in the environment of the ON-Bar process that makes a request, the ISM server uses a data-compression algorithm to store or retrieve the requested data. If **ISM_COMPRESSION** is set to FALSE or is not set, the ISM server does not use compression.

ISM_DEBUG_FILE

Set the **ISM_DEBUG_FILE** environment variable in the IBM Informix Storage Manager server environment to specify where to write XBSA messages.

```
►►—setenv—ISM_DEBUG_FILE—pathname—◄◄
```

pathname specifies the location of the XBSA message log file.

If you do not set `ISM_DEBUG_FILE`, the XBSA message log is located in the `$INFORMIXDIR/ism/appllogs/xbsa.messages` directory on UNIX, or in the `c:\nsr\appllogs\xbsa.messages` directory on Windows systems.

ISM_DEBUG_LEVEL

Set the `ISM_DEBUG_LEVEL` environment variable in the ON-Bar environment to control the level of reporting detail recorded in the XBSA messages log. The XBSA shared library writes to this log.

►►—setenv—ISM_DEBUG_LEVEL—*value*—◄◄

value specifies the level of reporting detail, where $1 \leq \textit{value} \leq 9$.

If `ISM_DEBUG_LEVEL` is not set, has a null value, or has a value outside this range, the default detail level is 1. A detail level of 0 suppresses all XBSA debugging records. A detail level of 1 reports only XBSA failures.

ISM_ENCRYPTION

Set the `ISM_ENCRYPTION` environment variable in the ON-Bar environment to specify whether IBM Informix Storage Manager (ISM) uses data encryption.

►►—setenv—ISM_ENCRYPTION—

XOR
NONE
TRUE

—◄◄

Three settings of `ISM_ENCRYPTION` are supported:

XOR uses encryption.

NONE does not use encryption.

TRUE uses encryption.

If `ISM_ENCRYPTION` is set to `NONE` or is not set, the ISM server does not use encryption.

If the `ISM_ENCRYPTION` is set to `TRUE` or `XOR` in the environment of the ON-Bar process that makes a request, the ISM server uses encryption to store or retrieve the data specified in that request.

ISM_MAXLOGSIZE

Set the `ISM_MAXLOGSIZE` environment variable in the IBM Informix Storage Manager (ISM) server environment to specify the size threshold of the ISM activity log.

►►—setenv—ISM_MAXLOGSIZE—*size*—◄◄

size specifies the size threshold (in megabytes) of the activity log.

If **ISM_MAXLOGSIZE** is not set, then the default size limit is 1 megabyte. If **ISM_MAXLOGSIZE** is set to a null value, then the threshold is 0 bytes.

ISM_MAXLOGVERS

Set the **ISM_MAXLOGVERS** environment variable in the IBM Informix Storage Manager (ISM) server environment to specify the maximum number of activity-log files to be preserved by the ISM server.

►—setenv—ISM_MAXLOGVERS—*value*—◄◄

value specifies the number of files to be preserved.

If **ISM_MAXLOGVERS** is not set, then the default number of files is four. If the setting is a null value, then the ISM server preserves no activity log files.

JAR_TEMP_PATH (IDS)

Set the **JAR_TEMP_PATH** variable to specify a non-default local file system location where jar management procedures such as **install_jar()** and **replace_jar()** can store temporary **.jar** files of the Java virtual machine.

►—setenv—JAR_TEMP_PATH—*pathname*—◄◄

pathname specifies a local directory for temporary **.jar** files.

This directory must have read and write permissions for the user who brings up the database server. If the **JAR_TEMP_PATH** environment variable is not set, temporary copies of **.jar** files are stored in the **/tmp** directory of the local file system for the database server.

JAVA_COMPILER (IDS)

You can set the **JAVA_COMPILER** environment variable in the Java virtual machine environment to disable JIT compilation.

►—setenv—JAVA_COMPILER—

none
NONE

—◄◄

The **NONE** and **none** settings are equivalent. On UNIX systems that support the C shell and on which **JAVA_COMPILER** has been set to **NONE** or **none**, you can enable the JIT compiler for the JVM environment by the following command:

```
unset JAVA_COMPILER
```

JVM_MAX_HEAP_SIZE (IDS)

The **JVM_MAX_HEAP_SIZE** environment variable can set a non-default upper limit on the size of the heap for the Java virtual machine.

►►—setenv—JVM_MAX_HEAP_SIZE—*size*—◄◄

size is a positive integer that specifies the maximum size (in megabytes).

For example, the following command sets the maximum heap size at 12 MB:

```
set JVM_MAX_HEAP_SIZE 12
```

If you do not set **JVM_MAX_HEAP_SIZE**, 16 MB is the default maximum size.

LD_LIBRARY_PATH (UNIX)

The **LD_LIBRARY_PATH** environment variable tells the shell on Solaris systems which directories to search for client or shared Informix general libraries. You must specify the directory that contains your client libraries before you can use the product.

►►—setenv—LD_LIBRARY_PATH—\$PATH:—*pathname*—◄◄

pathname specifies the search path for the library.

For INTERSOLV DataDirect ODBC Driver on AIX, set **LIBPATH**. For INTERSOLV DataDirect ODBC Driver on HP-UX, set **SHLIB_PATH**.

The following example sets the **LD_LIBRARY_PATH** environment variable to the desired directory:

```
setenv LD_LIBRARY_PATH  
${INFORMIXDIR}/lib:${INFORMIXDIR}/lib/esql:$LD_LIBRARY_PATH
```

LIBERAL_MATCH (XPS)

The **LIBERAL_MATCH** environment variable allows the database server to ignore trailing blanks when the **LIKE** and **MATCHES** operators occur in SQL statements that compare two column values.

►►—setenv—LIBERAL_MATCH—◄◄

When this environment variable is set, the database server ignores trailing blanks in a **LIKE** or **MATCHES** condition. For example, if **LIBERAL_MATCH** is set, and you specify “**M LIKE P**” when **P** contains trailing blank spaces that do not occur in **M**, the result is **TRUE**. When this environment variable is not set, the database server returns **FALSE** for string comparisons like this that differ only in trailing blank characters.

This environment variable supports behavior consistent with that of the LIKE and MATCHES operators in Dynamic Server, Versions 7.x, 9.x, and 10.x. This behavior (like the MATCHES operator) is an extension to the ANSI/ISO standard for SQL.

For more information about the LIKE and MATCHES operators, refer to the *IBM Informix: Guide to SQL Syntax*.

LIBPATH (UNIX)

The **LIBPATH** environment variable tells the shell on AIX systems which directories to search for dynamic-link libraries for the INTERSOLV DataDirect ODBC Driver. You must specify the full pathname for the directory where you installed the product.

```
►►setenv—LIBPATH—pathname—————►►
```

pathname specifies the search path for the libraries.

On Solaris, set **LD_LIBRARY_PATH**. On HP-UX, set **SHLIB_PATH**.

NODEFDAC

When the **NODEFDAC** environment variable is set to **yes**, it prevents default table privileges (Select, Insert, Update, and Delete) from being granted to PUBLIC when a new table is created during the current session in a database that is not ANSI compliant.

```
►►setenv—NODEFDAC—yes—————►►
```

yes prevents default table privileges from being granted to PUBLIC on new tables in a database that is not ANSI compliant. This setting also prevents the Execute privilege for a new user-defined routine from being granted to PUBLIC by default when the routine is created in Owner mode.

The **yes** setting is case sensitive, and is also sensitive to leading and trailing blank spaces. Including uppercase letters or blank spaces in the setting is equivalent to leaving **NODEFDAC** unset. When **NODEFDAC** is not set, or if it is set to any value besides **yes**, default privileges on tables and Owner-mode UDRs are granted to PUBLIC by default when the table or UDR is created in a database that is not ANSI-compliant.

ONCONFIG

The **ONCONFIG** environment variable specifies the name of the active file that holds configuration parameters for the database server. This file is read as

input during the initialization procedure. After you prepare the ONCONFIG configuration file, set **ONCONFIG** to the name of this file.

►►—setenv—ONCONFIG—*filename*—►►

filename is the name of a file in **\$INFORMIXDIR/etc** that contains the configuration parameters for your database.

To prepare the ONCONFIG file, make a copy of the **onconfig.std** file and modify the copy. It is recommended that you name the ONCONFIG file so that it can easily be related to a specific database server. If you have multiple instances of a database server, each instance *must* have its own uniquely named ONCONFIG file.

To prepare the ONCONFIG file for Extended Parallel Server, make a copy of the **onconfig.std** file if you are using a single coserver configuration or make a copy of the **onconfig.xps** file if you are using a multiple coserver configuration. You can use the **onconfig.std** file for a multiple coserver configuration, but you would have to add additional keywords and configuration parameters such as END, NODE, and COSERVER, which are already provided for you in the **onconfig.xps** file.

If the **ONCONFIG** environment variable is not set, the database server uses configuration values from either the **\$ONCONFIG** file or the **\$INFORMIXDIR/etc/onconfig** file.

For more information on configuration parameters and the ONCONFIG file, see the *IBM Informix: Administrator's Reference*.

OPTCOMPIND

You can set the **OPTCOMPIND** environment variable so that the optimizer can select the appropriate join method.

►►—setenv—OPTCOMPIND—

2
1
0

—►►

- 0 A nested-loop join is preferred, where possible, over a sort-merge join or a hash join.
- 1 When the isolation level is *not* Repeatable Read, the optimizer behaves as in setting 2; otherwise, the optimizer behaves as in setting 0.
- 2 Nested-loop joins are not necessarily preferred. The optimizer bases its decision purely on costs, regardless of transaction isolation mode.

When **OPTCOMPIND** is not set, the database server uses the OPTCOMPIND value from the ONCONFIG configuration file. When neither the environment variable nor the configuration parameter is set, the default value is 2.

On Dynamic Server, the SET ENVIRONMENT OPTCOMPIND statement can set or reset **OPTCOMPIND** dynamically at runtime. This overrides the current **OPTCOMPIND** value (or the the ONCONFIG configuration parameter OPTCOMPIND) for the current user session only. For more information on the SET ENVIRONMENT OPCOMPIND statement of SQL see the *IBM Informix: Guide to SQL Syntax*.

For more information on the ONCONFIG configuration parameter OPTCOMPIND, see the *IBM Informix: Administrator's Reference*. For more information on the different join methods that the optimizer uses, see your *IBM Informix: Performance Guide*.

OPTMSG

Set the **OPTMSG** environment variable at runtime before you start an IBM Informix ESQL/C application to enable (or disable) optimized message transfers (message chaining) for all SQL statements in an application.



0 disables optimized message transfers.

1 enables optimized message transfers and implements the feature for any subsequent connection.


The default value is 0 (zero), which explicitly disables message chaining. You might wish, for example, to disable optimized message transfers for statements that require immediate replies, for debugging, or to ensure that the database server processes all messages before the application terminates.

When you set **OPTMSG** within an application, you can activate or deactivate optimized message transfers for each connection or within each thread. To enable optimized message transfers, you must set **OPTMSG** before you establish a connection.

For more information about setting **OPTMSG** and defining related global variables, see the *IBM Informix: ESQL/C Programmer's Manual*.

OPTOFC

Set the **OPTOFC** environment variable to enable optimize-OPEN-FETCH-CLOSE functionality in an IBM Informix ESQL/C application or other APIs (such as JDBC, ODBC, OLE DB, LIBDMI, and Lib C++) that use DECLARE and OPEN statements to execute a cursor.

►►—setenv—OPTOFC——

- 0 disables **OPTOFC** for all threads of the application.
- 1 enables **OPTOFC** for every cursor in every thread of the application.

The default value is 0 (zero).

The **OPTOFC** environment variable reduces the number of message requests between the application and the database server.

If you set **OPTOFC** from the shell, you must set it before you start the ESQL/C application. For more information about enabling **OPTOFC** and related features, see the *IBM Informix: ESQL/C Programmer's Manual*.

OPT_GOAL (IDS, UNIX)

Set the **OPT_GOAL** environment variable in the user environment, before you start an application, to specify the query performance goal for the optimizer.

►►—setenv—OPT_GOAL——

- 0 specifies user-response-time optimization.
- 1 specifies total-query-time optimization.

The default behavior is for the optimizer to choose query plans that optimize the total query time.

You can also specify the optimization goal for individual queries with optimizer directives or for a session with the SET OPTIMIZATION statement.

Both methods take precedence over the **OPT_GOAL** environment variable setting. You can also set the OPT_GOAL configuration parameter for the Dynamic Server system; this method has the lowest level of precedence.

For more information about optimizing queries for your database server, see your *IBM Informix: Performance Guide*. For information on the SET OPTIMIZATION statement, see the *IBM Informix: Guide to SQL Syntax*.

PATH

The UNIX **PATH** environment variable tells the shell which directories to search for executable programs. You must add the directory containing your IBM Informix product to your **PATH** setting before you can use the product.



pathname specifies the search path for the executables.

Include a colon (:) separator between the pathnames on UNIX systems. (Use the semicolon (;) separator between pathnames on Windows systems.)

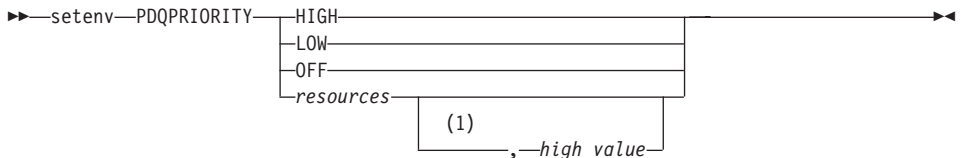
You can specify the search path in various ways. The **PATH** environment variable tells the operating system where to search for executable programs. You must include the directory that contains your IBM Informix product in your **path** setting before you can use the product. This directory should appear before **\$INFORMIXDIR/bin**, which you must also include.

For additional information about how to modify your path, see “Modifying an Environment-Variable Setting” on page 3-8.

PDQRIORITY

For Dynamic Server, the **PDQRIORITY** environment variable determines the degree of parallelism that the database server uses and affects how the database server allocates resources, including memory, processors, and disk reads.

For Extended Parallel Server, the **PDQRIORITY** environment variable determines only the allocation of memory resources.



Notes:

1 Extended Parallel Server only

resources Is an integer in the range 0 to 100. Value 1 is the same as LOW.

Value 0 is the same as OFF (for Dynamic Server only).

high_value Optional integer value that requests the maximum percentage of memory (for Extended Parallel Server only). When you specify this value after the *resources* value, you request a range of memory, expressed as a percentage.

Here the HIGH, LOW, and OFF keywords have the following effects:

HIGH	When the database server allocates resources among all users, it gives as many resources as possible to the query.
LOW	Data values are fetched from fragmented tables in parallel.
OFF	PDQ processing is turned off (for Dynamic Server only).

Usually, the more resources a database server uses, the better its performance for a given query. If the server uses too many resources, however, contention for the resources can take resources away from other queries, resulting in degraded performance. For more information on performance considerations for **PDQPRIORITY**, refer to your *IBM Informix: Performance Guide*.

An application can override the setting of this environment variable when it issues the SQL statement **SET PDQPRIORITY**, as the *IBM Informix: Guide to SQL Syntax* describes.

Using **PDQPRIORITY** with Dynamic Server

The *resources* value specifies the query priority level and the amount of resources that the database server uses to process the query. When you specify **LOW**, Dynamic Server uses no forms of parallelism.

When **PDQPRIORITY** is not set, the default value is **OFF**.

When **PDQPRIORITY** is set to **HIGH**, Dynamic Server determines an appropriate value to use for **PDQPRIORITY** based on several criteria. These include the number of available processors, the fragmentation of tables queried, the complexity of the query, and additional factors.

Using **PDQPRIORITY** with Extended Parallel Server

The *resources* value establishes the minimum percentage of memory when you also specify *high_value* to request a range of memory allocation. Other parallel operations can occur when the **PDQPRIORITY** setting is **LOW**.

When the **PDQPRIORITY** environment variable is not set, the default value is the value of the **PDQPRIORITY** configuration parameter.

When **PDQPRIORITY** is set to 0, Extended Parallel Server can execute a query in parallel, depending on the number of available processors, the fragmentation of tables queried, the complexity of the query, and other factors. **PDQPRIORITY** does not affect the degree of parallelism in Extended Parallel Server.

An application can prevent changes to the **PDQPRIORITY** setting with the **SET PDQPRIORITY IMMUTABLE** or **SET ALL_MUTABLES** statements of SQL. You can also override the setting of this environment variable by issuing

the SQL statement SET ENVIRONMENT to change the IMPLICIT_PDQ or BOUNT_IMPL_PDQ options, as the *IBM Informix: Guide to SQL Syntax* describes.

PLCONFIG (IDS)

The **PLCONFIG** environment variable specifies the name of the configuration file that the High-Performance Loader (HPL) uses. This file must reside in the **\$INFORMIXDIR/etc** directory. If the **PLCONFIG** environment variable is not set, then **\$INFORMIXDIR/etc/plconfig** is the default configuration file.

►—setenv—PLCONFIG—*filename*—◄

filename specifies the simple filename of the configuration file that the High-Performance Loader uses.

For example, to specify the **\$INFORMIXDIR/etc/custom.cfg** file as the configuration file for the High-Performance Loader, enter the following command:

```
setenv PLCONFIG custom.cfg
```

For more information, see the *IBM Informix: High-Performance Loader User's Guide*.

PLOAD_LO_PATH (IDS)

The **PLOAD_LO_PATH** environment variable lets you specify the pathname for smart-large-object handles (which identify the location of smart large objects such as BLOB and CLOB data types).

►—setenv—PLOAD_LO_PATH—*pathname*—◄

pathname specifies the directory for the smart-large-object handles.

If **PLOAD_LO_PATH** is not set, the default directory is **/tmp**.

For more information, see the *IBM Informix: High-Performance Loader User's Guide*.

PLOAD_SHMBASE (IDS)

The **PLOAD_SHMBASE** environment variable lets you specify the shared-memory address at which the High-Performance Loader (HPL) **onpload** processes will attach. If **PLOAD_SHMBASE** is not set, the HPL determines which shared-memory address to use.

►—setenv—PLOAD_SHMBASE—*value*—◄

value is used to calculate the shared-memory address.

If the **onpload** utility cannot attach, an error appears and you must specify a new value.

The **onpload** utility tries to determine at which address to attach, as follows in the following (descending) order:

1. Attach at the same address (SHMBASE) as the database server.
2. Attach beyond the database server segments.
3. Attach at the address specified in **PLOAD_SHMBASE**.

Tip: It is recommended that you let the HPL decide where to attach and that you set **PLOAD_SHMBASE** only if necessary to avoid shared-memory collisions between **onpload** and the database server.

For more information, see the *IBM Informix: High-Performance Loader User's Guide*.

PSORT_DBTEMP

The **PSORT_DBTEMP** environment variable specifies where the database server writes the temporary files it uses when it performs a sort.



pathname is the name of the UNIX directory used for intermediate writes during a sort.

To set the **PSORT_DBTEMP** environment variable to specify the directory (for example, `/usr/leif/tempsort`), enter the following command:

```
setenv PSORT_DBTEMP /usr/leif/tempsort
```

For maximum performance, specify directories that reside in file systems on different disks.

You might also want to consider setting the environment variable **DBSPACETEMP** to place temporary files used in sorting in dbspaces rather than operating-system files. See the discussion of the **DBSPACETEMP** environment variable in “**DBSPACETEMP**” on page 3-36.

The database server uses the directory that **PSORT_DBTEMP** specifies, even if the environment variable **PSORT_NPROCS** is not set. For additional information about the **PSORT_DBTEMP** environment variable, see your *IBM Informix: Administrator's Guide* and your *IBM Informix: Performance Guide*.

PSORT_NPROCS

The **PSORT_NPROCS** environment variable enables the database server to improve the performance of the parallel-process sorting package by allocating more threads for sorting.

PSORT_NPROCS does not necessarily improve sorting speed for Extended Parallel Server, because the database server sorts in parallel whether this environment variable is set or not.

Before the sorting package performs a parallel sort, make sure that the database server has enough memory for the sort.

►—setenv—PSORT_NPROCS—*threads*—◄

threads is an integer, specifying the maximum number of threads to be used to sort a query. This value cannot be greater than 10.

The following command sets **PSORT_NPROCS** to 4:

```
setenv PSORT_NPROCS 4
```

To disable parallel sorting, enter the following command:

```
unsetenv PSORT_NPROCS
```

It is recommended that you initially set **PSORT_NPROCS** to 2 when your computer has multiple CPUs. If subsequent CPU activity is lower than I/O activity, you can increase the value of **PSORT_NPROCS**.

Tip: If the **PDQPRIORITY** environment variable is not set, the database server allocates the minimum amount of memory to sorting. This minimum memory is insufficient to start even two sort threads. If you have not set **PDQPRIORITY**, check the available memory before you perform a large-scale sort (such as an index build) to make sure that you have enough memory.

Default Values for Detached Indexes

If the **PSORT_NPROCS** environment variable is set, the database server uses the specified number of sort threads as an upper limit for ordinary sorts. If **PSORT_NPROCS** is not set, parallel sorting does not take place. The database server uses one thread for the sort. If **PSORT_NPROCS** is set to 0, the database server uses three threads for the sort.

Default Values for Attached Indexes

The default number of threads is different for attached indexes.

If the **PSORT_NPROCS** environment variable is set, you get the specified number of sort threads for each fragment of the index that is being built.

If **PSORT_NPROCS** is not set, or if it is set to 0, you get two sort threads for each fragment of the index unless you have a single-CPU virtual processor. If you have a single-CPU virtual processor, you get one sort thread for each fragment of the index.

For additional information about the **PSORT_NPROCS** environment variable, see your *IBM Informix: Administrator's Guide* and your *IBM Informix: Performance Guide*.

RTREE_COST_ADJUST_VALUE (IDS)

The **RTREE_COST_ADJUST_VALUE** environment variable specifies a coefficient that support functions of user-defined data types can use to estimate the cost of an R-tree index for queries on UDT columns.

►►—setenv—RTREE_COST_ADJUST_VALUE—*value*—►►

value is a floating-point number, where $1 \leq \textit{value} \leq 1000$, specifying a multiplier for estimating the cost of using an index on a UDT column.

For spatial queries, the I/O overhead tends to exceed by far the CPU cost, so by multiplying the uncorrected estimated cost by an appropriate *value* from this setting, the database server can make better cost-based decisions on how to implement queries on UDT columns for which an R-tree index exists.

SHLIB_PATH (UNIX)

The **SHLIB_PATH** environment variable tells the shell on HP-UX systems which directories to search for dynamic-link libraries. This is used, for example, with the INTERSOLV DataDirect ODBC Driver. You must specify the full pathname for the directory where you installed the product.

►►—setenv—SHLIB_PATH—\$PATH:—*pathname*—►►

pathname specifies the search path for the libraries.

On Solaris systems, set **LD_LIBRARY_PATH**. On AIX systems, set **LIBPATH**.

STMT_CACHE (IDS)

Use the **STMT_CACHE** environment variable to control the use of the shared-statement cache on a session. This feature can reduce memory consumption and can speed query processing among different user sessions. Valid **STMT_CACHE** values are 1 and 0.

►►—setenv—STMT_CACHE—1
0—————►►

- 1 enables the SQL statement cache.
- 0 disables the SQL statement cache.

Set the **STMT_CACHE** environment variable for applications that do not use the SET STMT_CACHE statement to control the use of the SQL statement cache. By default, a statement cache of 512 kilobytes is enabled, but this feature can be disabled or set to a non-default size through the STMT_CACHE parameter of the **onconfig.std** file or by the SET STMT_CACHE statement.

This environment variable has no effect if the SQL statement cache is disabled through the configuration file setting. Values set by the SET STMT_CACHE statement in the application override the **STMT_CACHE** setting.

TERM (UNIX)

The **TERM** environment variable is used for terminal handling. It lets DB–Access (and other character-based applications) recognize and communicate with the terminal that you are using.

►►—setenv—TERM—*type*—————►►

type specifies the terminal type.

The terminal type specified in the **TERM** setting must correspond to an entry in the **termcap** file or **terminfo** directory.

Before you can set the **TERM** environment variable, you must obtain the code for your terminal from the database administrator.

For example, to specify the vt100 terminal, set the **TERM** environment variable by entering the following command:

```
setenv TERM vt100
```

TERMCAP (UNIX)

The **TERMCAP** environment variable is used for terminal handling. It tells DB–Access (and other character-based applications) to communicate with the **termcap** file instead of the **terminfo** directory.

►►—setenv—TERMCAP—*pathname*—————►►

pathname specifies the location of the **termcap** file.

The **termcap** file contains a list of various types of terminals and their characteristics. For example, to provide DB–Access terminal-handling information, which is specified in the `/usr/informix/etc/termcap` file, enter the following command:

```
setenv TERMCAP /usr/informix/etc/termcap
```

You can use set **TERMCAP** in any of the following ways. If several **termcap** files exist, they have the following (descending) order of precedence:

1. The **termcap** file that you create
2. The **termcap** file that the database server supplies (that is, `$INFORMIXDIR/etc/termcap`)
3. The operating-system **termcap** file (that is, `/etc/termcap`)

If you set the **TERMCAP** environment variable, be sure that the **INFORMIXTERM** environment variable is set to the default, **termcap**.

If you do not set the **TERMCAP** environment variable, the system file (that is, `/etc/termcap`) is used by default.

TERMINFO (UNIX)

The **TERMINFO** environment variable is used for terminal handling.

The environment variable is supported only on platforms that provide full support for the **terminfo** libraries that System V and Solaris UNIX systems provide.

```
►—setenv—TERMINFO—/usr/lib/terminfo—◄
```

TERMINFO tells DB–Access to communicate with the **terminfo** directory instead of the **termcap** file. The **terminfo** directory has subdirectories that contain files that pertain to terminals and their characteristics.

To set **TERMINFO**, enter the following command:

```
setenv  TERMINFO  /usr/lib/terminfo
```

If you set the **TERMINFO** environment variable, you must also set the **INFORMIXTERM** environment variable to **terminfo**.

THREADLIB (UNIX)

Use the **THREADLIB** environment variable to compile multithreaded ESQL/C applications. A multithreaded ESQL/C application lets you establish as many connections to one or more databases as there are threads. These connections can remain active while the application program executes.

The **THREADLIB** environment variable indicates which thread package to use when you compile an application. Currently only the Distributed Computing Environment (DCE) is supported.

►►—setenv—THREADLIB—DCE—◄◄

The **THREADLIB** environment variable is checked when the **-thread** option is passed to the ESQL/C script when you compile a multithreaded ESQL/C application. When you use the **-thread** option while compiling, the ESQL/C script generates an error if **THREADLIB** is not set, or if **THREADLIB** is set to an unsupported thread package.

TOBIGINT (XPS)

You can use the **TOBIGINT** environment variable to change the default **INT8** label that the **dbschema** utility displays in its output for columns of the **INT8** data type to the string **BIGINT**.

►►—setenv—TOBIGINT—1—◄◄

Set **TOBIGINT** to 1 to enable, and unset **TOBIGINT** to disable this **dbschema** functionality. The name **BIGINT** is the identifier of a built-in 8-byte integer data type of DB2 database servers of IBM. See the Migration Guide for additional information about the **TOBIGINT** environment variable.

USETABLENAME (IDS)

The **USETABLENAME** environment variable can prevent users from using a synonym to specify the *table* in ALTER TABLE or DROP TABLE statements. Unlike most environment variables, **USETABLENAME** does not need to be set to a value. It takes effect if you set it to any value, or to no value.

►►—setenv—USETABLENAME—◄◄

By default, ALTER TABLE or DROP TABLE statements accept a valid synonym for the name of the *table* to be altered or dropped. (In contrast, RENAME TABLE issues an error if you specify a synonym, as do the ALTER SEQUENCE, DROP SEQUENCE, and RENAME SEQUENCE statements, if you attempt to substitute a synonym for the *sequence* name in those statements.)

If you set **USETABLENAME**, an error results if a synonym appears in ALTER TABLE or DROP TABLE statements. Setting **USETABLENAME** has no effect on the DROP VIEW statement, which accepts a valid synonym for the view.

XFER_CONFIG (XPS)

The **XFER_CONFIG** environment variable specifies the location of the **xfer_config** configuration file.

►—setenv—XFER_CONFIG—*pathname*—◄

pathname specifies the location of the **xfer_config** file.

The **xfer_config** file works with the **onxfer** utility to help users migrate from Version 7.x to Version 8.x. It contains various configuration parameter settings that users can modify and a list of tables that users can select to be transferred.

The default **xfer_config** file is located in the **\$INFORMIXDIR/etc** directory on UNIX systems or in the **%INFORMIXDIR%\etc** directory in Windows.

Index of Environment Variables

Table 3-4 on page 3-81 provides an overview of the uses for the various Informix and UNIX environment variables that Version 8.5 and Version 10.0 support. This serves as an index to general topics and lists the related environment variables and the pages where the environment variables are introduced. Where the **Topic** column is empty, the entry refers to the previously listed topic.

The term *GLS Guide* in the **Page** column in Table 3-4 indicates environment variables that are described in the *IBM Informix: GLS User's Guide*.

The term *ER Guide* in the **Page** column in Table 3-4 indicates environment variables that are described in the *IBM Informix: Dynamic Server Enterprise Replication Guide*.

Table 3-4. Uses for Environment Variables

Topic	Environment Variable	Page
Abbreviated year values	DBCENTURY	3-22
Alarms for SQL operations		
Globally detached indexes	GLOBAL_DETACH_INFRM	3-45
Cartesian joins	IFMX_CART_ALARM	3-46
ANSI/ISO SQL compliance		
Lettercase of owner names	ANSIOWNER	3-18
Informix syntax extensions	DBANSIWARN	3-21
default table privileges	NODEFDAC	3-68

Table 3-4. Uses for Environment Variables (continued)

Topic	Environment Variable	Page
archecker utility	AC_CONFIG	3-18
Buffer: fetch size	FET_BUF_SIZE	3-44
network size	IFX_NETBUF_SIZE	3-53
network pool size	IFX_NETBUF_PVTPOOL_SIZE	3-52
BYTE or TEXT data buffer	DBBLOBBUF	3-22
Cache: enabling	STMT_CACHE	3-77
size for Optical Subsystem	INFORMIXOPCACHE	3-59
Client/server:		
default server	INFORMIXSERVER	3-60
shared memory segments	INFORMIXSHMBASE	3-60
stacksize for client session	INFORMIXSTACKSIZE	3-62
locale of client, server	CLIENT_LOCALE, DBLOCALE	GLS Guide
locale for file I/O	SERVER_LOCALE	GLS Guide
Code-set conversion		
code set of client, server	CLIENT_LOCALE, DB_LOCALE	GLS Guide
character-string conversion	DBNLS	3-32
Communication Support Module: DCE-GSS	INFORMIXKEYTAB	3-59
concsm.cfg file	INFORMIXCONCSMCFG	3-56
Compiler:	INFORMIXC	3-55
multibyte characters	CC8BITLEVEL	GLS Guide
C++	INFORMIXCPPMAP	3-58
ESQL/C	THREADLIB	3-80
Configuration file:		
database server	ONCONFIG	3-68
ignore environment variables	ENVIGNORE	3-43
Configuration parameter: COSERVER	INFORMIXSERVER	3-60
DBSERVERNAME	INFORMIXSERVER	3-60
DBSPACETEMP	DBSPACETEMP	3-36
DIRECTIVES	IFX_DIRECTIVES IFX_EXTDIRECTIVES	3-49 3-50
OPCACHEMAX	INFORMIXOPCACHE	3-59

Table 3-4. Uses for Environment Variables (continued)

Topic	Environment Variable	Page	
	OPTCOMPIND	3-69	
	OPT_GOAL	3-71	
	PDQPRIORITY	3-72	
	STACKSIZE	3-62	
Connecting	INFORMIXCONRETRY	3-56	
	INFORMIXCONTIME	3-56	
	INFORMIXSERVER	3-60	
	INFORMIXSQLHOSTS	3-61	
Data distributions	DBUPSPACE	3-41	
Database locale	DB_LOCALE	GLS Guide	
Database server	INFORMIXSERVER	3-60	
	locale for file I/O	SERVER_LOCALE	GLS Guide
	configuration file	ONCONFIG	3-68
	parallel sorting	PSORT_DBTEMP PSORT_NPROCS	3-75 3-76
	parallelism	PDQPRIORITY	3-72
	role separation	INF_ROLE_SEP	3-63
	shared memory	INFORMIXSHMBASE	3-60
	stacksize	INFORMIXSTACKSIZE	3-62
	temporary tables	DBSPACETEMP DBTEMP PSORT_DBTEMP	3-36 3-38 3-75
	variable-length packets	IFX_PAD_VARCHAR	3-53
Date and time values, formats	DBCENTURY	3-22	
	DBDATE	3-25:	
	GL_DATE	GLS Guide	
	DBTIME	3-39:	
	GL_DATETIME	GLS Guide	
	IBM_XPS_PARAMS	3-45	

Table 3-4. Uses for Environment Variables (continued)

Topic	Environment Variable	Page
DB-Access utility	DBANSIWARN	3-21
	DBDELIMITER	3-28
	DBEDIT	3-28
	DBFLTMASK	3-29
	DBNLS	3-32
	DBPATH	3-33
	FET_BUF_SIZE	3-44
	INFORMIXSERVER	3-60
	INFORMIXTERM	3-62
	TERM	3-78
	TERMCAP	3-78
TERMINFO	3-79	
dbexport utility	DBDELIMITER	3-28
dbschema utility	TOBIGINT	3-80
Delimited identifiers	DELIMIDENT	3-42
Disk space	DBUPSPACE	3-41
Editor	DBEDIT	3-28
Enterprise Replication	CDR_LOGDELTA CDR_PERFLOG CDR_ROUTER CDR_RMSCALEFACT CDRSITES_731 CDRSITES_92X	ER Guide
ESQL/C: ANSI compliance	DBANSIWARN	3-21
C compiler	INFORMIXC	3-55
DATEIME formatting	DBTIME	3-39; GLS Guide
delimited identifiers	DELIMIDENT	3-42
multibyte characters	CLIENT_LOCALE, ESQLMF	GLS Guide
multithreaded applications	THREADLIB	3-80
C preprocessor	CPFIRST	3-20
Executable programs	PATH	3-71
Fetch buffer size	FET_BUF_SIZE	3-44
Filenames: multibyte	GLS8BITSYS	GLS Guide
Files: field delimiter	DBDELIMITER	3-28
Files: installation	INFORMIXDIR	3-58

Table 3-4. Uses for Environment Variables (continued)

Topic	Environment Variable	Page
Files: locale	CLIENT_LOCALE DB_LOCALE SERVER_LOCALE	GLS Guide
Files: map for C++	INFORMIXCPPMAP	3-58
Files: message	DBLANG	3-29
Files: temporary	DBSPACETEMP	3-36
Files: temporary, for Gateways	DBTEMP	3-38
Files: temporary sorting	PSORT_DBTEMP	3-75
Files: termcap , terminfo	INFORMIXTERM TERM TERMCAP TERMINFO	3-62 3-78 3-78 3-79
Formats: date and time	DBDATE GL_DATE DBTIME GL_DATETIME	3-25; GLS Guide 3-39; GLS Guide
Format: money	DBMONEY	3-30, GLS Guide
Gateways	DBTEMP	3-38
High-Performance Loader	DBONPLOAD PLCONFIG PLOAD_LO_PATH PLOAD_SHMBASE	3-33 3-74 3-74 3-74
Identifiers: delimited	DELIMIDENT	3-42
Identifiers: longer than 18 bytes	IFX_LONGID	3-51
Identifiers: multibyte characters	CLIENT_LOCALE,ESQLMF	GLS Guide
IBM Informix Storage Manager	ISM_COMPRESSION ISM_DEBUG_FILE ISM_DEBUG_LEVEL ISM_ENCRYPTION	3-64 3-64 3-65 3-65
IBM Informix Storage Manager	ISM_MAXLOGSIZE ISM_MAXLOGVERS	3-65 3-66
Installation	INFORMIXDIR PATH	3-58 3-71
Language environment	DBLANG See also "Nondefault Locale"	3-29,; GLS Guide

Table 3-4. Uses for Environment Variables (continued)

Topic	Environment Variable	Page
Libraries	LD_LIBRARY_PATH LIBPATH SHLIB_PATH	3-67 3-68 3-77
Locale	CLIENT_LOCALE DB_LOCALE SERVER_LOCALE	GLS Guide
Lock Mode	IFX_DEF_TABLE_LOCKMODE	3-48
Long Identifiers	IFX_LONGID	3-51
Map file for C++	INFORMIXCPPMAP	3-58
Message chaining	OPTMSG	3-70
Message files	DBLANG	3-29;; GLS Guide
Money format	DBMONEY	3-30;; GLS Guide
Multibyte characters	CLIENT_LOCALE DB_LOCALE SERVER_LOCALE	GLS Guide
Multibyte filter	ESQLMF	GLS Guide
Multithreaded applications	THREADLIB	3-80
Network	DBPATH	3-33
Nondefault locale	DBNLS CLIENT_LOCALE DB_LOCALE SERVER_LOCALE	3-32;; GLS Guide
ON-Bar utility	ISM_COMPRESSION ISM_DEBUG_LEVEL ISM_ENCRYPTION	3-64 3-65 3-65
ONCONFIG parameters	See "Configuration parameter"	3-82
Optical Subsystem	INFORMIXOPCACHE	3-59
Optimization: directives	IFX_DIRECTIVES IFX_EXTDIRECTIVES	3-49 3-50
Optimization: message transfers	OPTMSG	3-70
Optimization: join method	OPTCOMPIND	3-69
Optimization: performance goal	OPT_GOAL	3-71
OPTOFC feature	OPTOFC	3-70
Parameters	See "Configuration parameter"	3-82 to 3-83
Pathname: archecker config file	AC_CONFIG	3-18

Table 3-4. Uses for Environment Variables (continued)

Topic	Environment Variable	Page
Pathname: C compiler	INFORMIXC	3-55
Pathname: database files	DBPATH	3-33
Pathname: executable programs	PATH	3-71
Pathname: HPL sblob handles	PLOAD_LO_PATH	3-74
Pathname: installation	INFORMIXDIR	3-58
Pathname: libraries	LD_LIBRARY_PATH LIBPATH SHLIB_PATH	3-67 3-68 3-77
Pathname: message files	DBLANG	3-29;; GLS Guide
Pathname: parallel sorting	PSORT_DBTEMP	3-75
Pathname: remote shell	DBREMOTECMD	3-36
Pathname: xfer_config file	XFER_CONFIG	3-81
Preserve owner name lettercase	ANSIOWNER	3-18
Printing	DBPRINT	3-35
Privileges	NODEFDAC	3-63
Query: optimization	IFX_DIRECTIVES IFX_EXTDIRECTIVES IFMX_OPT_FACT_TABS IFMX_OPT_NON_DIM_TABS OPTCOMPIND OPT_GOAL RTREE_COST_ADJUST_VALUE	3-49 3-50 3-47 3-48 3-69 3-71 3-77
Query: prioritization	PDQPRIORITY	3-72
Remote shell	DBREMOTECMD	3-36
Role separation	INF_ROLE_SEP	3-63
Rolled-back transactions	DBACCNOIGN, IFX_ XASTDCOMPLIANCE_XAEND	3-20 3-53
Routine: DATETIME formatting	DBTIME	3-39;; GLS Guide
Server	See "Database Server"	3-83
Server locale	SERVER_LOCALE	GLS Guide
Shared memory	INFORMIXSHMBASE PLOAD_SHMBASE	3-60 3-74
Shell: remote	DBREMOTECMD	3-36
Shell: search path	PATH	3-71

Table 3-4. Uses for Environment Variables (continued)

Topic	Environment Variable	Page
Sorting	PSORT_DBTEMP	3-75
	PSORT_NPROCS	3-76
SQL statements:caching	STMT_CACHE	3-77
CONNECT	INFORMIXCONTIME	3-56
	INFORMIXSERVER	3-60
CREATE TEMP TABLE	DBSPACETEMP	3-36
DESCRIBE FOR UPDATE	IFX_UPDESC	3-53
LOAD, UNLOAD	DBDELIMITER	3-28
LOAD, UNLOAD	DBBLOBBUF	3-22
SELECT INTO TEMP	DBSPACETEMP	3-36
SET PDQPRIORITY	PDQPRIORITY	3-72
SET STMT_CACHE	STMT_CACHE	3-77
UPDATE STATISTICS	DBUPSPACE	3-41
Stacksize	INFORMIXSTACKSIZE	3-62
String search: trailing blanks	LIBERAL_MATCH	3-67
Temporary tables	DBSPACETEMP	3-36
	DBTEMP	3-38
	PSORT_DBTEMP	3-75
Terminal handling	INFORMIXTERM	3-62
	TERM	3-78
	TERMCAP	3-78
	TERMINFO	3-79
Time-limited software license	IFX_NO_TIMELIMIT_WARNING	3-52
Time zone, specifying	IBM_XPS_PARAMS	3-45
Utilities: DB-Access	DBANSIWARN	3-21
	DBDELIMITER	3-28
	DBEDIT	3-28
	DBFLTMASK	3-29
	DBNLS	3-32
	DBPATH	3-33
	FET_BUF_SIZE	3-44
	IFMX_HISTORY_SIZER	3-47
	INFORMIXSERVER	3-60
	INFORMIXTERM	3-62
	TERM	3-78
	TERMCAP	3-78
	TERMINFO	3-79
Utilities: dbexport	DBDELIMITER	3-28

Table 3-4. Uses for Environment Variables (continued)

Topic	Environment Variable	Page
Utilities: ON-Bar	ISM_COMPRESSION	3-64
	ISM_DEBUG_LEVEL	3-65
	ISM_ENCRYPTION	3-65
Variables: overriding	ENVIGNORE	3-43
Year values (abbreviated)	DBCENTURY	3-22

Appendix A. The `stores_demo` Database

The `stores_demo` database contains a set of tables that describe an imaginary business. The examples in the *IBM Informix: Guide to SQL Syntax*, the *IBM Informix: Guide to SQL Tutorial*, and other IBM Informix manuals are based on this demonstration database. The `stores_demo` database uses the default (U.S. English) locale and is not ANSI compliant.

This appendix contains the following sections:

- The first section describes the structure of the tables in the `stores_demo` database. It identifies the primary key of each table, lists the name and data type of each column, and indicates whether the column has a default value or check constraint. Indexes on columns are also identified and classified as unique, allowing duplicate values.
- The second section (“The `stores_demo` Database Map” on page A-6) shows a map of the tables in the `stores_demo` database and indicates the relationships among columns.
- The third section (“Primary-Foreign Key Relationships” on page A-8) describes the primary-foreign key relationships among columns in tables.
- The final section (“Data in the `stores_demo` Database” on page A-13) lists the data contained in each table of the `stores_demo` database.

For information on how to create and populate the `stores_demo` database, see the *IBM Informix: DB–Access User’s Guide*. For information on how to design and implement a relational database, see the *IBM Informix: Database Design and Implementation Guide*.

Structure of the Tables

The `stores_demo` database contains information about a fictitious sporting-goods distributor that services stores in the western United States. This database includes the following tables:

- `customer` (page A-2)
- `orders` (page A-2)
- `items` (page A-3)
- `stock` (page A-3)
- `catalog` (page A-4)
- `cust_calls` (page A-4)
- `call_type` (page A-5)
- `manufact` (page A-5)

- **state** (page A-5)

Sections that follow describe each table. The unique identifying value for each table (primary key) is shaded.

The customer Table

The **customer** table contains information about the retail stores that place orders from the distributor. Table A-1 shows the columns of the **customer** table.

The **zipcode** column in Table A-1 is indexed and allows duplicate values.

Table A-1. The customer Table

Column Name	Data Type	Description
customer_num	SERIAL(101)	System-generated customer number
fname	CHAR(15)	First name of store representative
lname	CHAR(15)	Last name of store representative
company	CHAR(20)	Name of store
address1	CHAR(20)	First line of store address
address2	CHAR(20)	Second line of store address
city	CHAR(15)	City
state	CHAR(2)	State (foreign key to state table)
zipcode	CHAR(5)	Zipcode
phone	CHAR(18)	Telephone number

The orders Table

The **orders** table contains information about orders placed by the customers of the distributor. Table A-2 on page A-3 shows the columns of the **orders** table.

Table A-2. The orders Table

Column Name	Data Type	Description
order_num	SERIAL(1001)	System-generated order number
order_date	DATE	Date order entered
customer_num	INTEGER	Customer number (foreign key to customer table)
ship_instruct	CHAR(40)	Special shipping instructions
backlog	CHAR(1)	Indicates order cannot be filled because the item is backlogged: y = yes n = no
po_num	CHAR(10)	Customer purchase order number
ship_date	DATE	Shipping date
ship_weight	DECIMAL(8,2)	Shipping weight
ship_charge	MONEY(6)	Shipping charge
paid_date	DATE	Date order paid

The items Table

An order can include one or more items. One row exists in the **items** table for each item in an order. Table A-3 shows the columns of the **items** table.

Table A-3. The items Table

Column Name	Data Type	Description
item_num	SMALLINT	Sequentially assigned item number for an order
order_num	INTEGER	Order number (foreign key to orders table)
stock_num	SMALLINT	Stock number for item (foreign key to stock table)
manu_code	CHAR(3)	Manufacturer code for item ordered (foreign key to manufact table)
quantity	SMALLINT	Quantity ordered (value must be > 1)
total_price	MONEY(8)	Quantity ordered * unit price = total price of item

The stock Table

The distributor carries 41 types of sporting goods from various manufacturers. More than one manufacturer can supply an item. For example, the distributor offers racing goggles from two manufacturers and running shoes from six manufacturers.

The **stock** table is a catalog of the items sold by the distributor. Table A-4 on page A-4 shows the columns of the **stock** table.

Table A-4. The stock Table

Column Name	Data Type	Description
stock_num	SMALLINT	Stock number that identifies type of item
manu_code	CHAR(3)	Manufacturer code (foreign key to manufact table)
description	CHAR(15)	Description of item
unit_price	MONEY(6,2)	Unit price
unit	CHAR(4)	Unit by which item is ordered: <ul style="list-style-type: none"> • Each • Pair • Case • Box
unit_descr	CHAR(15)	Description of unit

The catalog Table

The **catalog** table describes each item in stock. Retail stores use this table when placing orders with the distributor. Table A-5 shows the columns of the **catalog** table.

Table A-5. The catalog Table

Column Name	Data Type	Description
catalog_num	SERIAL(10001)	System-generated catalog number
stock_num	SMALLINT	Distributor stock number (foreign key to stock table)
manu_code	CHAR(3)	Manufacturer code (foreign key to manufact table)
cat_descr	TEXT	Description of item
cat_picture	BYTE	Picture of item (binary data)
cat_advert	VARCHAR(255, 65)	Tag line underneath picture

The cust_calls Table

All customer calls for information on orders, shipments, or complaints are logged. The **cust_calls** table contains information about these types of customer calls. Table A-6 on page A-5 shows the columns of the **cust_calls** table.

Table A-6. The *cust_calls* Table

Column Name	Data Type	Description
customer_num	INTEGER	Customer number (foreign key to customer table)
call_dtime	DATETIME YEAR TO MINUTE	Date and time when call was received
user_id	CHAR(18)	Name of person logging call (default is user login name)
call_code	CHAR(1)	Type of call (foreign key to call_type table)
call_descr	CHAR(240)	Description of call
res_dtime	DATETIME YEAR TO MINUTE	Date and time when call was resolved
res_descr	CHAR(240)	Description of how call was resolved

The **call_type** Table

The call codes associated with customer calls are stored in the **call_type** table. Table A-7 shows the columns of the **call_type** table.

Table A-7. The *call_type* Table

Column Name	Data Type	Description
call_code	CHAR(1)	Call code
code_descr	CHAR (30)	Description of call type

The **manufact** Table

Information about the nine manufacturers whose sporting goods are handled by the distributor is stored in the **manufact** table. Table A-8 shows the columns of the **manufact** table.

Table A-8. The *manufact* Table

Column Name	Data Type	Description
manu_code	CHAR(3)	Manufacturer code
manu_name	CHAR(15)	Name of manufacturer
lead_time	INTERVAL DAY(3) TO DAY	Lead time for shipment of orders

The **state** Table

The **state** table contains the names and postal abbreviations for the 50 states of the United States. Table A-9 on page A-6 shows the columns of the **state** table.

Table A-9. The state Table

Column Name	Data Type	Description
code	CHAR(2)	State code
sname	CHAR(15)	State name

The stores_demo Database Map

Figure A-1 on page A-7 displays the joins in the **stores_demo** database. The gray shading that connects a column in one table to a column with the same name in another table indicates the relationships, or *joins*, between tables.

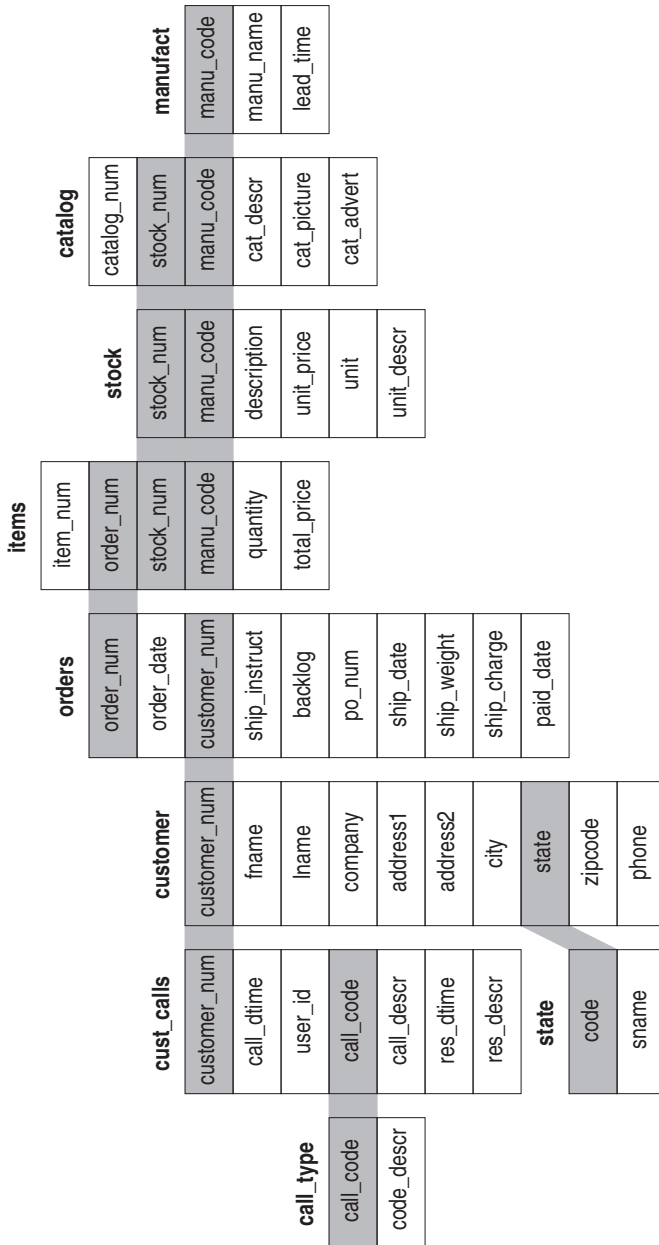


Figure A-1. Joins in the `stores_demo` Database

Primary-Foreign Key Relationships

The tables of the **stores_demo** database are linked by the primary-foreign key relationships that Figure A-1 on page A-7 shows and are identified in this section. This type of relationship is called a *referential constraint* because a foreign key in one table *references* the primary key in another table. Figure A-2 through Figure A-9 on page A-13 show the relationships among tables and how information stored in one table supplements information stored in others.

The customer and orders Tables

The **customer** table contains a **customer_num** column that holds a number that identifies a customer and columns for the customer name, company, address, and telephone number. For example, the row with information about Anthony Higgins contains the number 104 in the **customer_num** column. The **orders** table also contains a **customer_num** column that stores the number of the customer who placed a particular order. In the **orders** table, the **customer_num** column is a foreign key that references the **customer_num** column in the **customer** table. Figure A-2 shows this relationship.

customer Table (detail)		
customer_num	fname	lname
101	Ludwig	Pauli
102	Carole	Sadler
103	Philip	Currie
104	Anthony	Higgins

orders Table (detail)		
order_num	order_date	customer_num
1001	05/20/1998	104
1002	05/21/1998	101
1003	05/22/1998	104
1004	05/22/1998	106

Figure A-2. Tables That the *customer_num* Column Joins

According to Figure A-2, customer 104 (Anthony Higgins) has placed two orders, as his customer number appears in two rows of the **orders** table. Because the customer number is a foreign key in the **orders** table, you can retrieve Anthony Higgins's name, address, and information about his orders at the same time.

The orders and items Tables

The **orders** and **items** tables are linked by an **order_num** column that contains an identification number for each order. If an order includes several items, the same order number appears in several rows of the **items** table. In the **items**

table, the **order_num** column is a foreign key that references the **order_num** column in the **orders** table. Figure A-3 shows this relationship.

orders Table (detail)

order_num	order_date	customer_num
1001	05/20/1998	104
1002	05/21/1998	101
1003	05/22/1998	104

items Table (detail)

item_num	order_num	stock_num	manu_code
1	1001	1	HRO
4	1002	4	HSK
3	1002	3	HSK
9	1003	9	ANZ
8	1003	8	ANZ
5	1003	5	ANZ

Figure A-3. Tables That the `order_num` Column Joins

The items and stock Tables

The **items** table and the **stock** table are joined by two columns: the **stock_num** column, which stores a stock number for an item, and the **manu_code** column, which stores a code that identifies the manufacturer. You need both the stock number and the manufacturer code to uniquely identify an item. For example, the item with the stock number 1 and the manufacturer code HRO is a Hero baseball glove; the item with the stock number 1 and the manufacturer code HSK is a Husky baseball glove.

The same stock number and manufacturer code can appear in more than one row of the **items** table, if the same item belongs to separate orders. In the **items** table, the **stock_num** and **manu_code** columns are foreign keys that reference the **stock_num** and **manu_code** columns in the **stock** table. Figure A-4 on page A-10 shows this relationship.

items Table (detail)

item_num	order_num	stock_num	manu_code
1	1001	1	HRO
1	1002	4	HSK
2	1002	3	HSK
1	1003	9	ANZ
2	1003	8	ANZ
3	1003	5	ANZ
1	1004	1	HRO

stock Table (detail)

stock_num	manu_code	Description
1	HRO	baseball gloves
1	HSK	baseball gloves
1	SMT	baseball gloves

Figure A-4. Tables That the `stock_num` and `manu_code` Columns Join

The stock and catalog Tables

The **stock** table and **catalog** table are joined by two columns: the **stock_num** column, which stores a stock number for an item, and the **manu_code** column, which stores a code that identifies the manufacturer. You need both columns to uniquely identify an item. In the **catalog** table, the **stock_num** and **manu_code** columns are foreign keys that reference the **stock_num** and **manu_code** columns in the **stock** table. Figure A-5 shows this relationship.

stock Table (detail)

stock_num	manu_code	Description
1	HRO	baseball gloves
1	HSK	baseball gloves
1	SMT	baseball gloves

catalog Table (detail)

catalog_num	stock_num	manu_code
10001	1	HRO
10002	1	HSK
10003	1	SMT
10004	2	HRO

Figure A-5. Tables That the `stock_num` and `manu_code` Columns Join

The stock and manufact Tables

The **stock** table and the **manufact** table are joined by the **manu_code** column. The same manufacturer code can appear in more than one row of the **stock** table if the manufacturer produces more than one piece of equipment. In the **stock** table, the **manu_code** column is a foreign key that references the **manu_code** column in the **manufact** table. Figure A-6 shows this relationship.

stock Table (detail)

stock_num	manu_code	Description
1	HRO	baseball gloves
1	HSK	baseball gloves
1	SMT	baseball gloves

manufact Table (detail)

manu_code	manu_name
NRG	Norge
HSK	Husky
HRO	Hero

Figure A-6. Tables That the manu_code Column Joins

The cust_calls and customer Tables

The **cust_calls** table and the **customer** table are joined by the **customer_num** column. The same customer number can appear in more than one row of the **cust_calls** table if the customer calls the distributor more than once with a problem or question. In the **cust_calls** table, the **customer_num** column is a foreign key that references the **customer_num** column in the **customer** table. Figure A-7 on page A-12 shows this relationship.

customer Table (detail)

customer_num	fname	lname
101	Ludwig	Pauli
102	Carole	Sadler
103	Philip	Currie
104	Anthony	Higgins
105	Raymond	Vector
106	George	Watson

cust_calls Table (detail)

customer_num	call_dtime	user_id
106	1998-06-12 08:20	maryj
127	1998-07-31 14:30	maryj
116	1997-11-28 13:34	mannyh
116	1997-12-21 11:24	mannyh

Figure A-7. Tables That the customer_num Column Joins

The call_type and cust_calls Tables

The **call_type** and **cust_calls** tables are joined by the **call_code** column. The same call code can appear in more than one row of the **cust_calls** table because many customers can have the same *type* of problem. In the **cust_calls** table, the **call_code** column is a foreign key that references the **call_code** column in the **call_type** table. Figure A-8 shows this relationship.

call_type Table (detail)

call_code	code_descr
B	Billing error
D	Damaged goods
I	Incorrect merchandise sent
L	Late shipment
O	Other

cust_calls Table (detail)

customer_num	call_dtime	call_code
106	1998-06-12 08:20	D
127	1998-07-31 14:30	I
116	1997-11-28 13:34	I
116	1997-12-21 11:24	I

Figure A-8. Tables That the call_code Column Joins

The state and customer Tables

The **state** table and the **customer** table are joined by a column that contains the state code. This column is called **code** in the **state** table and **state** in the **customer** table. If several customers live in the same state, the same state code appears in several rows of the table. In the **customer** table, the **state** column is a foreign key that references the **code** column in the **state** table. Figure A-9 shows this relationship.

customer Table (detail)					
customer_num	fname	lname	---	state	
101	Ludwig	Pauli	---	CA	
102	Carole	Sadler	---	CA	
103	Philip	Currie	---	CA	

state Table (detail)	
code	sname
AK	Alaska
AL	Alabama
AR	Arkansas
AZ	Arizona
CA	California

Figure A-9. Relationship Between the state Column and the code Column

Data in the stores_demo Database

The following tables display the data in the **stores_demo** database.

customer Table

customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone
101	Ludwig	Pauli	All Sports Supplies	213 Erstwild Court		Sunnyvale	CA	94086	408-789-8075
102	Carole	Sadler	Sports Spot	785 Geary Street		San Francisco	CA	94117	415-822-1289
103	Philip	Currie	Phil's Sports	654 Poplar	P. O. Box 3498	Palo Alto	CA	94303	650-328-4543

customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone
104	Anthony	Higgins	Play Ball!	East Shopping Center	422 Bay Road	Redwood City	CA	94026	650-368-1100
105	Raymond	Vector	Los Altos Sports	1899 La Loma Drive		Los Altos	CA	94022	650-776-3249
106	George	Watson	Watson & Son	1143 Carver Place		Mountain View	CA	94063	650-389-8789
107	Charles	Ream	Athletic Supplies	41 Jordan Avenue		Palo Alto	CA	94304	650-356-9876
108	Donald	Quinn	Quinn's Sports	587 Alvarado		Redwood City	CA	94063	650-544-8729
109	Jane	Miller	Sport Stuff	Mayfair Mart	7345 Ross Blvd.	Sunnyvale	CA	94086	408-723-8789
110	Roy	Jaeger	AA Athletics	520 Topaz Way		Redwood City	CA	94062	650-743-3611
111	Frances	Keyes	Sports Center	3199 Sterling Court		Sunnyvale	CA	94085	408-277-7245
112	Margaret	Lawson	Runners & Others	234 Wyandotte Way		Los Altos	CA	94022	650-887-7235
113	Lana	Beatty	Sportstown	654 Oak Grove		Menlo Park	CA	94025	650-356-9982
114	Frank	Albertson	Sporting Place	947 Waverly Place		Redwood City	CA	94062	650-886-6677
115	Alfred	Grant	Gold Medal Sports	776 Gary Avenue		Menlo Park	CA	94025	650-356-1123

customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone
116	Jean	Parmelee	Olympic City	1104 Spinosa Drive		Mountain View	CA	94040	650-534-8822
117	Arnold	Sipes	Kids Korner	850 Lytton Court		Redwood City	CA	94063	650-245-4578
118	Dick	Baxter	Blue Ribbon Sports	5427 College		Oakland	CA	94609	650-655-0011
119	Bob	Shorter	The Triathletes Club	2405 Kings Highway		Cherry Hill	NJ	08002	609-663-6079
120	Fred	Jewell	Century Pro Shop	6627 N. 17th Way		Phoenix	AZ	85016	602-265-8754
121	Jason	Wallack	City Sports	Lake Biltmore Mall	350 W. 23rd Street	Wilmington	DE	19898	302-366-7511
122	Cathy	O'Brian	The Sporting Life	543 Nassau Street		Princeton	NJ	08540	609-342-0054
123	Marvin	Hanlon	Bay Sports	10100 Bay Meadows Road	Suite 1020	Jacksonville	FL	32256	904-823-4239
124	Chris	Putnum	Putnum's Putters	4715 S.E. Adams Blvd	Suite 909C	Bartlesville	OK	74006	918-355-2074
125	James	Henry	Total Fitness Sports	1450 Commonwealth Avenue		Brighton	MA	02135	617-232-4159
126	Eileen	Neelie	Neelie's Discount Sports	2539 South Utica Street		Denver	CO	80219	303-936-7731
127	Kim	Satifer	Big Blue Bike Shop	Blue Island Square	12222 Gregory Street	Blue Island	NY	60406	312-944-5691

customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone
128	Frank	Lessor	Phoenix University	Athletic Department	1817 N. Thomas Road	Phoenix	AZ	85008	602-533-1817

items Table

item_num	order_num	stock_num	manu_code	quantity	total_price
1	1001	1	HRO	1	250.00
1	1002	4	HSK	1	960.00
2	1002	3	HSK	1	240.00
1	1003	9	ANZ	1	20.00
2	1003	8	ANZ	1	840.00
3	1003	5	ANZ	5	99.00
1	1004	1	HRO	1	250.00
2	1004	2	HRO	1	126.00
3	1004	3	HSK	1	240.00
4	1004	1	HSK	1	800.00
1	1005	5	NRG	10	280.00
2	1005	5	ANZ	10	198.00
3	1005	6	SMT	1	36.00
4	1005	6	ANZ	1	48.00
1	1006	5	SMT	5	125.00
2	1006	5	NRG	5	140.00
3	1006	5	ANZ	5	99.00
4	1006	6	SMT	1	36.00
5	1006	6	ANZ	1	48.00
1	1007	1	HRO	1	250.00
2	1007	2	HRO	1	126.00
3	1007	3	HSK	1	240.00
4	1007	4	HRO	1	480.00
5	1007	7	HRO	1	600.00

item_num	order_num	stock_num	manu_code	quantity	total_price
1	1008	8	ANZ	1	840.00
2	1008	9	ANZ	5	100.00
1	1009	1	SMT	1	450.00
1	1010	6	SMT	1	36.00
2	1010	6	ANZ	1	48.00
1	1011	5	ANZ	5	99.00
1	1012	8	ANZ	1	840.00
2	1012	9	ANZ	10	200.00
1	1013	5	ANZ	1	19.80
2	1013	6	SMT	1	36.00
3	1013	6	ANZ	1	48.00
4	1013	9	ANZ	2	40.00
1	1014	4	HSK	1	960.00
2	1014	4	HRO	1	480.00
1	1015	1	SMT	1	450.00
1	1016	101	SHM	2	136.00
2	1016	109	PRC	3	90.00
3	1016	110	HSK	1	308.00
4	1016	114	PRC	1	120.00
1	1017	201	NKL	4	150.00
2	1017	202	KAR	1	230.00
3	1017	301	SHM	2	204.00
1	1018	307	PRC	2	500.00
2	1018	302	KAR	3	15.00
3	1018	110	PRC	1	236.00
4	1018	5	SMT	4	100.00
5	1018	304	HRO	1	280.00
1	1019	111	SHM	3	1499.97
1	1020	204	KAR	2	90.00
2	1020	301	KAR	4	348.00
1	1021	201	NKL	2	75.00
2	1021	201	ANZ	3	225.00
3	1021	202	KAR	3	690.00

item_num	order_num	stock_num	manu_code	quantity	total_price
4	1021	205	ANZ	2	624.00
1	1022	309	HRO	1	40.00
2	1022	303	PRC	2	96.00
3	1022	6	ANZ	2	96.00
1	1023	103	PRC	2	40.00
2	1023	104	PRC	2	116.00
3	1023	105	SHM	1	80.00
4	1023	110	SHM	1	228.00
5	1023	304	ANZ	1	170.00
6	1023	306	SHM	1	190.00

call_type Table

call_code	code_descr
B	billing error
D	damaged goods
I	incorrect merchandise sent
L	late shipment
O	other

orders Table

order_num	order_date	customer_num	ship_instruct	backlog	po_num	ship_date	ship_weight	ship_charge	paid_date
1001	05/20/1998	104	express	n	B77836	06/01/1998	20.40	10.00	07/22/1998
1002	05/21/1998	101	PO on box; deliver back door only	n	9270	05/26/1998	50.60	15.30	06/03/1998
1003	05/22/1998	104	express	n	B77890	05/23/1998	35.60	10.80	06/14/1998
1004	05/22/1998	106	ring bell twice	y	8006	05/30/1998	95.80	19.20	
1005	05/24/1998	116	call before delivery	n	2865	06/09/1998	80.80	16.20	06/21/1998
1006	05/30/1998	112	after 10AM	y	Q13557		70.80	14.20	

order_num	order_date	customer_num	ship_instruct	backlog	po_num	ship_date	ship_weight	ship_charge	paid_date
1007	05/31/1998	117		n	278693	06/05/1998	125.90	25.20	
1008	06/07/1998	110	closed Monday	y	LZ230	07/06/1998	45.60	13.80	07/21/1998
1009	06/14/1998	111	door next to grocery	n	4745	06/21/1998	20.40	10.00	08/21/1998
1010	06/17/1998	115	deliver 776 King St. if no answer	n	429Q	06/29/1998	40.60	12.30	08/22/1998
1011	06/18/1998	104	express	n	B77897	07/03/1998	10.40	5.00	08/29/1998
1012	06/18/1998	117		n	278701	06/29/1998	70.80	14.20	
1013	06/22/1998	104	express	n	B77930	07/10/1998	60.80	12.20	07/31/1998
1014	06/25/1998	106	ring bell, kick door loudly	n	8052	07/03/1998	40.60	12.30	07/10/1998
1015	06/27/1998	110	closed Mondays	n	MA003	07/16/1998	20.60	6.30	08/31/1998
1016	06/29/1998	119	delivery entrance off Camp St.	n	PC6782	07/12/1998	35.00	11.80	
1017	07/09/1998	120	North side of clubhouse	n	DM3543 31	07/13/1998	60.00	18.00	
1018	07/10/1998	121	SW corner of Biltmore Mall	n	S22942	07/13/1998	70.50	20.00	08/06/1998
1019	07/11/1998	122	closed til noon Mondays	n	Z55709	07/16/1998	90.00	23.00	08/06/1998
1020	07/11/1998	123	express	n	W2286	07/16/1998	14.00	8.50	09/20/1998
1021	07/23/1998	124	ask for Elaine	n	C3288	07/25/1998	40.00	12.00	08/22/1998
1022	07/24/1998	126	express	n	W9925	07/30/1998	15.00	13.00	09/02/1998
1023	07/24/1998	127	no deliveries after 3 p.m.	n	KF2961	07/30/1998	60.00	18.00	08/22/1998

stock Table

stock_num	manu_code	description	unit_rice	unit	unit_descr
1	HRO	baseball gloves	250.00	case	10 gloves/case
1	HSK	baseball gloves	800.00	case	10 gloves/case

stock_num	manu_code	description	unit_rice	unit	unit_descr
1	SMT	baseball gloves	450.00	case	10 gloves/case
2	HRO	baseball	126.00	case	24/case
3	HSK	baseball bat	240.00	case	12/case
3	SHM	baseball bat	280.00	case	12/case
4	HSK	football	960.00	case	24/case
4	HRO	football	480.00	case	24/case
5	NRG	tennis racquet	28.00	each	each
5	SMT	tennis racquet	25.00	each	each
5	ANZ	tennis racquet	19.80	each	each
6	SMT	tennis ball	36.00	case	24 cans/case
6	ANZ	tennis ball	48.00	case	24 cans/case
7	HRO	basketball	600.00	case	24/case
8	ANZ	volleyball	840.00	case	24/case
9	ANZ	volleyball net	20.00	each	each
101	PRC	bicycle tires	88.00	box	4/box
101	SHM	bicycle tires	68.00	box	4/box
102	SHM	bicycle brakes	220.00	case	4 sets/case
102	PRC	bicycle brakes	480.00	case	4 sets/case
103	PRC	front derailleur	20.00	each	each
104	PRC	rear derailleur	58.00	each	each
105	PRC	bicycle wheels	53.00	pair	pair
105	SHM	bicycle wheels	80.00	pair	pair
106	PRC	bicycle stem	23.00	each	each
107	PRC	bicycle saddle	70.00	pair	pair
108	SHM	crankset	45.00	each	each
109	PRC	pedal binding	30.00	case	6 pairs/case
109	SHM	pedal binding	200.00	case	4 pairs/case
110	PRC	helmet	236.00	case	4/case
110	ANZ	helmet	244.00	case	4/case
110	SHM	helmet	228.00	case	4/case
110	HRO	helmet	260.00	case	4/case
110	HSK	helmet	308.00	case	4/case
111	SHM	10-spd, assmbld	499.99	each	each

stock_num	manu_code	description	unit_rice	unit	unit_descr
112	SHM	12-spd, assmbld	549.00	each	each
113	SHM	18-spd, assmbld	685.90	each	each
114	PRC	bicycle gloves	120.00	case	10 pairs/case
201	NKL	golf shoes	37.50	each	each
201	ANZ	golf shoes	75.00	each	each
201	KAR	golf shoes	90.00	each	each
202	NKL	metal woods	174.00	case	2 sets/case
202	KAR	std woods	230.00	case	2 sets/case
203	NKL	irons/wedges	670.00	case	2 sets/case
204	KAR	putter	45.00	each	each
205	NKL	3 golf balls	312.00	case	24/case
205	ANZ	3 golf balls	312.00	case	24/case
205	HRO	3 golf balls	312.00	case	24/case
301	NKL	running shoes	97.00	each	each
301	HRO	running shoes	42.50	each	each
301	SHM	running shoes	102.00	each	each
301	PRC	running shoes	75.00	each	each
301	KAR	running shoes	87.00	each	each
301	ANZ	running shoes	95.00	each	each
302	HRO	ice pack	4.50	each	each
302	KAR	ice pack	5.00	each	each
303	PRC	socks	48.00	box	24 pairs/box
303	KAR	socks	36.00	box	24 pair/box
304	ANZ	watch	170.00	box	10/box
304	HRO	watch	280.00	box	10/box
305	HRO	first-aid kit	48.00	case	4/case
306	PRC	tandem adapter	160.00	each	each
306	SHM	tandem adapter	190.00	each	each
307	PRC	infant jogger	250.00	each	each
308	PRC	twin jogger	280.00	each	each
309	HRO	ear drops	40.00	case	20/case
309	SHM	ear drops	40.00	case	20/case
310	SHM	kick board	80.00	case	10/case

stock_num	manu_code	description	unit_rice	unit	unit_descr
310	ANZ	kick board	89.00	case	12/case
311	SHM	water gloves	48.00	box	4 pairs/box
312	SHM	racer goggles	96.00	box	12/box
312	HRO	racer goggles	72.00	box	12/box
313	SHM	swim cap	72.00	box	12/box
313	ANZ	swim cap	60.00	box	12/box

catalog Table

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10001	1	HRO	Brown leather. Specify first baseman's or infield/outfield style. Specify right- or left-handed.	<BYTE value>	Your First Season's Baseball Glove
10002	1	HSK	Babe Ruth signature glove. Black leather. Infield/outfield style. Specify right- or left-handed.	<BYTE value>	All-Leather, Hand-Stitched, Deep-Pockets, Sturdy Webbing that Won't Let Go
10003	1	SMT	Catcher's mitt. Brown leather. Specify right- or left-handed.	<BYTE value>	A Sturdy Catcher's Mitt With the Perfect Pocket
10004	2	HRO	Jackie Robinson signature glove. Highest Professional quality, used by National League.	<BYTE value>	Highest Quality Ball Available, from the Hand-Stitching to the Robinson Signature
10005	3	HSK	Pro-style wood. Available in sizes: 31, 32, 33, 34, 35.	<BYTE value>	High-Technology Design Expands the Sweet Spot
10006	3	SHM	Aluminum. Blue with black tape. 31", 20 oz or 22 oz; 32", 21 oz or 23 oz; 33", 22 oz or 24 oz.	<BYTE value>	Durable Aluminum for High School and Collegiate Athletes
10007	4	HSK	Norm Van Brocklin signature style.	<BYTE value>	Quality Pigskin with Norm Van Brocklin Signature
10008	4	HRO	NFL-Style pigskin.	<BYTE value>	Highest Quality Football for High School and Collegiate Competitions

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10009	5	NRG	Graphite frame. Synthetic strings.	<BYTE value>	Wide Body Amplifies Your Natural Abilities by Providing More Power Through Aerodynamic Design
10010	5	SMT	Aluminum frame. Synthetic strings.	<BYTE value>	Mid-Sized Racquet for the Improving Player
10011	5	ANZ	Wood frame, cat-gut strings.	<BYTE value>	Antique Replica of Classic Wooden Racquet Built with Cat-Gut Strings
10012	6	SMT	Soft yellow color for easy visibility in sunlight or artificial light.	<BYTE value>	High-Visibility Tennis, Day or Night
10013	6	ANZ	Pro-core. Available in neon yellow, green, and pink.	<BYTE value>	Durable Construction Coupled with the Brightest Colors Available
10014	7	HRO	Indoor. Classic NBA style. Brown leather.	<BYTE value>	Long-Life Basketballs for Indoor Gymnasiums
10015	8	ANZ	Indoor. Finest leather. Professional quality.	<BYTE value>	Professional Volleyballs for Indoor Competitions
10016	9	ANZ	Steel eyelets. Nylon cording. Double-stitched. Sanctioned by the National Athletic Congress.	<BYTE value>	Sanctioned Volleyball Netting for Indoor Professional and Collegiate Competition
10017	101	PRC	Reinforced, hand-finished tubular. Polyurethane belted. Effective against punctures. Mixed tread for super wear and road grip.	<BYTE value>	Ultimate in Puncture Protection, Tires Designed for In-City Riding
10018	101	SHM	Durable nylon casing with butyl tube for superior air retention. Center-ribbed tread with herringbone side. Coated sidewalls resist abrasion.	<BYTE value>	The Perfect Tire for Club Rides or Training
10019	102	SHM	Thrust bearing and coated pivot washer/ spring sleeve for smooth action. Slotted levers with soft gum hoods. Two-tone paint treatment. Set includes calipers, levers, and cables.	<BYTE value>	Thrust-Bearing and Spring-Sleeve Brake Set Guarantees Smooth Action

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10020	102	PRC	Computer-aided design with low-profile pads. Cold-forged alloy calipers and beefy caliper bushing. Aero levers. Set includes calipers, levers, and cables.	<BYTE value>	Computer Design Delivers Rigid Yet Vibration-Free Brakes
10021	103	PRC	Compact leading-action design enhances shifting. Deep cage for super-small granny gears. Extra strong construction to resist off-road abuse.	<BYTE value>	Climb Any Mountain: ProCycle's Front Derailleur Adds Finesse to Your ATB
10022	104	PRC	Floating trapezoid geometry with extra thick parallelogram arms. 100-tooth capacity. Optimum alignment with any freewheel.	<BYTE value>	Computer-Aided Design Engineers 100-Tooth Capacity Into ProCycle's Rear Derailleur
10023	105	PRC	Front wheels laced with 15g spokes in a 3-cross pattern. Rear wheels laced with 14g spikes in a 3-cross pattern.	<BYTE value>	Durable Training Wheels That Hold True Under Toughest Conditions
10024	105	SHM	Polished alloy. Sealed-bearing, quick-release hubs. Double-butted. Front wheels are laced 15g/2-cross. Rear wheels are laced 15g/3-cross.	<BYTE value>	Extra Lightweight Wheels for Training or High-Performance Touring
10025	106	PRC	Hard anodized alloy with pearl finish. 6mm hex bolt hardware. Available in lengths of 90-140mm in 10mm increments.	<BYTE value>	ProCycle Stem with Pearl Finish
10026	107	PRC	Available in three styles: Men's racing; Men's touring; and Women's. Anatomical gel construction with lycra cover. Black or black/hot pink.	<BYTE value>	The Ultimate In Riding Comfort, Lightweight With Anatomical Support
10027	108	SHM	Double or triple crankset with choice of chainrings. For double crankset, chainrings from 38-54 teeth. For triple crankset, chainrings from 24-48 teeth.	<BYTE value>	Customize Your Mountain Bike With Extra-Durable Crankset

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10028	109	PRC	Steel toe clips with nylon strap. Extra wide at buckle to reduce pressure.	<BYTE value>	Classic Toeclip Improved to Prevent Soreness at Clip Buckle
10029	109	SHM	Ingenious new design combines button on sole of shoe with slot on a pedal plate to give riders new options in riding efficiency. Choose full or partial locking. Four plates mean both top and bottom of pedals are slotted—no fishing around when you want to engage full power. Fast unlocking ensures safety when maneuverability is paramount.	<BYTE value>	Ingenious Pedal/Clip Design Delivers Maximum Power and Fast Unlocking
10030	110	PRC	Super-lightweight. Meets both ANSI and Snell standards for impact protection. 7.5 oz. Quick-release shadow buckle.	<BYTE value>	Feather-Light, Quick-Release, Maximum Protection Helmet
10031	110	ANZ	No buckle so no plastic touches your chin. Meets both ANSI and Snell standards for impact protection. 7.5 oz. Lycra cover.	<BYTE value>	Minimum Chin Contact, Feather-Light, Maximum Protection Helmet
10032	110	SHM	Dense outer layer combines with softer inner layer to eliminate the mesh cover, no snagging on brush. Meets both ANSI and Snell standards for impact protection. 8.0 oz.	<BYTE value>	Mountain Bike Helmet: Smooth Cover Eliminates the Worry of Brush Snags But Delivers Maximum Protection
10033	110	HRO	Newest ultralight helmet uses plastic shell. Largest ventilation channels of any helmet on the market. 8.5 oz.	<BYTE value>	Lightweight Plastic with Vents Assures Cool Comfort Without Sacrificing Protection
10034	110	HSK	Aerodynamic (teardrop) helmet covered with anti-drag fabric. Credited with shaving 2 seconds/mile from winner's time in Tour de France time-trial. 7.5 oz.	<BYTE value>	Teardrop Design Used by Yellow Jerseys, You Can Time the Difference

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10035	111	SHM	Light-action shifting 10 speed. Designed for the city commuter with shock-absorbing front fork and drilled eyelets for carry-all racks or bicycle trailers. Internal wiring for generator lights. 33 lbs.	<BYTE value>	Fully Equipped Bicycle Designed for the Serious Commuter Who Mixes Business With Pleasure
10036	112	SHM	Created for the beginner enthusiast. Ideal for club rides and light touring. Sophisticated triple-buttet frame construction. Precise index shifting. 28 lbs.	<BYTE value>	We Selected the Ideal Combination of Touring Bike Equipment, then Turned It Into This Package Deal: High-Performance on the Roads, Maximum Pleasure Everywhere
10037	113	SHM	Ultra-lightweight. Racing frame geometry built for aerodynamic handlebars. Cantilever brakes. Index shifting. High-performance gearing. Quick-release hubs. Disk wheels. Bladed spokes.	<BYTE value>	Designed for the Serious Competitor, The Complete Racing Machine
10038	114	PRC	Padded leather palm and stretch mesh merged with terry back; Available in tan, black, and cream. Sizes S, M, L, XL.	<BYTE value>	Riding Gloves for Comfort and Protection
10039	201	NKL	Designed for comfort and stability. Available in white & blue or white & brown. Specify size.	<BYTE value>	Full-Comfort, Long-Wearing Golf Shoes for Men and Women
10040	201	ANZ	Guaranteed waterproof. Full leather upper. Available in white, bone, brown, green, and blue. Specify size.	<BYTE value>	Waterproof Protection Ensures Maximum Comfort and Durability In All Climates
10041	201	KAR	Leather and leather mesh for maximum ventilation. Waterproof lining to keep feet dry. Available in white and gray or white and ivory. Specify size.	<BYTE value>	Karsten's Top Quality Shoe Combines Leather and Leather Mesh

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10042	202	NKL	Complete starter set utilizes gold shafts. Balanced for power.	<BYTE value>	Starter Set of Woods, Ideal for High School and Collegiate Classes
10043	202	KAR	Full set of woods designed for precision control and power performance.	<BYTE value>	High-Quality Woods Appropriate for High School Competitions or Serious Amateurs
10044	203	NKL	Set of eight irons includes 3 through 9 irons and pitching wedge. Originally priced at \$489.00.	<BYTE value>	Set of Irons Available From Factory at Tremendous Savings: Discontinued Line
10045	204	KAR	Ideally balanced for optimum control. Nylon-covered shaft.	<BYTE value>	High-Quality Beginning Set of Irons Appropriate for High School Competitions
10046	205	NKL	Fluorescent yellow.	<BYTE value>	Long Drive Golf Balls: Fluorescent Yellow
10047	205	ANZ	White only.	<BYTE value>	Long Drive Golf Balls: White
10048	205	HRO	Combination fluorescent yellow and standard white.	<BYTE value>	HiFlier Golf Balls: Case Includes Fluorescent Yellow and Standard White
10049	301	NKL	Super shock-absorbing gel pads disperse vertical energy into a horizontal plane for extraordinary cushioned comfort. Great motion control. Men's only. Specify size.	<BYTE value>	Maximum Protection For High-Mileage Runners
10050	301	HRO	Engineered for serious training with exceptional stability. Fabulous shock absorption. Great durability. Specify men's/women's, size.	<BYTE value>	Pronators and Supinators Take Heart: A Serious Training Shoe For Runners Who Need Motion Control

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10051	301	SHM	For runners who log heavy miles and need a durable, supportive, stable platform. Mesh/synthetic upper gives excellent moisture dissipation. Stability system uses rear antipronation platform and forefoot control plate for extended protection during high-intensity training. Specify men's/women's size.	<BYTE value>	The Training Shoe Engineered for Marathoners and Ultra-Distance Runners
10052	301	PRC	Supportive, stable racing flat. Plenty of forefoot cushioning with added motion control. Women's only. D widths available. Specify size.	<BYTE value>	A Woman's Racing Flat That Combines Extra Forefoot Protection With a Slender Heel
10053	301	KAR	Anatomical last holds your foot firmly in place. Feather-weight cushioning delivers the responsiveness of a racing flat. Specify men's/women's size.	<BYTE value>	Durable Training Flat That Can Carry You Through Marathon Miles
10054	301	ANZ	Cantilever sole provides shock absorption and energy rebound. Positive traction shoe with ample toe box. Ideal for runners who need a wide shoe. Available in men's and women's. Specify size.	<BYTE value>	Motion Control, Protection, and Extra Toebox Room
10055	302	KAR	Reusable ice pack with velcro strap. For general use. Velcro strap allows easy application to arms or legs.	<BYTE value>	Finally, an Ice Pack for Achilles Injuries and Shin Splints That You Can Take to the Office
10056	303	PRC	Neon nylon. Perfect for running or aerobics. Indicate color: Fluorescent pink, yellow, green, and orange.	<BYTE value>	Knock Their Socks Off With YOUR Socks
10057	303	KAR	100% nylon blend for optimal wicking and comfort. We've taken out the cotton to eliminate the risk of blisters and reduce the opportunity for infection. Specify men's or women's.	<BYTE value>	100% Nylon Blend Socks - No Cotton

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10058	304	ANZ	Provides time, date, dual display of lap/cumulative splits, 4-lap memory, 10 hr count-down timer, event timer, alarm, hour chime, waterproof to 50m, velcro band.	<BYTE value>	Athletic Watch w/4-Lap Memory
10059	304	HRO	Split timer, waterproof to 50m. Indicate color: Hot pink, mint green, space black.	<BYTE value>	Waterproof Triathlete Watch In Competition Colors
10060	305	HRO	Contains ace bandage, anti-bacterial cream, alcohol cleansing pads, adhesive bandages of assorted sizes, and instant-cold pack.	<BYTE value>	Comprehensive First-Aid Kit Essential for Team Practices, Team Traveling
10061	306	PRC	Converts a standard tandem bike into an adult/child bike. User-tested assembly instructions	<BYTE value>	Enjoy Bicycling With Your Child on a Tandem; Make Your Family Outing Safer
10062	306	SHM	Converts a standard tandem bike into an adult/child bike. Lightweight model.	<BYTE value>	Consider a Touring Vacation for the Entire Family: A Lightweight, Touring Tandem for Parent and Child
10063	307	PRC	Allows mom or dad to take the baby out too. Fits children up to 21 pounds. Navy blue with black trim.	<BYTE value>	Infant Jogger Keeps A Running Family Together
10064	308	PRC	Allows mom or dad to take both children! Rated for children up to 18 pounds.	<BYTE value>	As Your Family Grows, Infant Jogger Grows With You
10065	309	HRO	Prevents swimmer's ear.	<BYTE value>	Swimmers Can Prevent Ear Infection All Season Long
10066	309	SHM	Extra-gentle formula. Can be used every day for prevention or treatment of swimmer's ear.	<BYTE value>	Swimmer's Ear Drops Specially Formulated for Children
10067	310	SHM	Blue heavy-duty foam board with Shimara or team logo.	<BYTE value>	Exceptionally Durable, Compact Kickboard for Team Practice
10068	310	ANZ	White. Standard size.	<BYTE value>	High-Quality Kickboard

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10069	311	SHM	Swim gloves. Webbing between fingers promotes strengthening of arms. Cannot be used in competition.	<BYTE value>	Hot Training Tool - Webbed Swim Gloves Build Arm Strength and Endurance
10070	312	SHM	Hydrodynamic egg-shaped lens. Ground-in anti-fog elements; Available in blue or smoke.	<BYTE value>	Anti-Fog Swimmer's Goggles: Quantity Discount
10071	312	HRO	Durable competition-style goggles. Available in blue, grey, or white.	<BYTE value>	Swim Goggles: Traditional Rounded Lens For Greater Comfort
10072	313	SHM	Silicone swim cap. One size. Available in white, silver, or navy. Team Logo Imprinting Available.	<BYTE value>	Team Logo Silicone Swim Cap
10073	314	ANZ	Silicone swim cap. Squared-off top. One size. White	<BYTE value>	Durable Squared-off Silicone Swim Cap
10074	315	HRO	Re-usable ice pack. Store in the freezer for instant first-aid. Extra capacity to accommodate water and ice.	<BYTE value>	Water Compartment Combines With Ice to Provide Optimal Orthopedic Treatment

cust_calls Table

customer_num	call_dtime	user_id	call_code	call_descr	res_dtime	res_descr
106	1998-06-12 8:20	maryj	D	Order was received, but two of the cans of ANZ tennis balls within the case were empty.	1998-06-12 8:25	Authorized credit for two cans to customer, issued apology. Called ANZ buyer to report the QA problem.
110	1998-07-07 10:24	richc	L	Order placed one month ago (6/7) not received.	1998-07-07 10:30	Checked with shipping (Ed Smith). Order sent yesterday- we were waiting for goods from ANZ. Next time will call with delay if necessary.
119	1998-07-01 15:00	richc	B	Bill does not reflect credit from previous order.	1998-07-02 8:21	Spoke with Jane Akant in Finance. She found the error and is sending new bill to customer.

customer_num	call_dtime	user_id	call_code	call_descr	res_dtime	res_descr
121	1998-07-10 14:05	maryj	O	Customer likes our merchandise. Requests that we stock more types of infant joggers. Will call back to place order.	1998-07-10 14:06	Sent note to marketing group of interest in infant joggers.
127	1998-07-31 14:30	maryj	I	Received Hero watches (item # 304) instead of ANZ watches.		Sent memo to shipping to send ANZ item 304 to customer and pickup HRO watches. Should be done tomorrow, 8/1.
116	1997-11-28 13:34	mannyn	I	Received plain white swim caps (313 ANZ) instead of navy with team logo (313 SHM).	1997-11-28 16:47	Shipping found correct case in warehouse and express mailed it in time for swim meet.
116	1997-12-21 11:24	mannyn	I	Second complaint from this customer! Received two cases right-handed outfielder gloves (1 HRO) instead of one case lefties.	1997-12-27 08:19	Memo to shipping (Ava Brown) to send case of left-handed gloves, pick up wrong case; memo to billing requesting 5% discount to placate customer due to second offense and lateness of resolution because of holiday.

manufact Table

manu_code	manu_name	lead_time
ANZ	Anza	5
HSK	Husky	5
HRO	Hero	4
NRG	Norge	7
SMT	Smith	3
SHM	Shimara	30
KAR	Karsten	21
NKL	Nikolus	8
PRC	ProCycle	9

state Table

code	sname	code	sname
AK	Alaska	MT	Montana
AL	Alabama	NE	Nebraska
AR	Arkansas	NC	North Carolina
AZ	Arizona	ND	North Dakota
CA	California	NH	New Hampshire
CT	Connecticut	NJ	New Jersey
CO	Colorado	NM	New Mexico
DC	Washington, D.C.	NV	Nevada
DE	Delaware	NY	New York
FL	Florida	OH	Ohio
GA	Georgia	OK	Oklahoma
HI	Hawaii	OR	Oregon
IA	Iowa	PA	Pennsylvania
ID	Idaho	PR	Puerto Rico
IL	Illinois	RI	Rhode Island
IN	Indiana	SC	South Carolina
KY	Kentucky	TN	Tennessee
LA	Louisiana	TX	Texas
MA	Massachusetts	UT	Utah
MD	Maryland	VA	Virginia
ME	Maine	VT	Vermont
MI	Michigan	WA	Washington
MN	Minnesota	WI	Wisconsin
MO	Missouri	WV	West Virginia
MS	Mississippi	WY	Wyoming

Appendix B. The sales_demo and superstores_demo Databases

In addition to the **stores_demo** database that is described in detail in Appendix A, IBM Informix products include the following demonstration databases:

Extended Parallel Server

- The **sales_demo** database illustrates a dimensional schema for data-warehousing applications.

End of Extended Parallel Server

Dynamic Server

- The **superstores_demo** database illustrates an object-relational schema.

End of Dynamic Server

This appendix discusses the structures of these two demonstration databases.

For information on how to create and populate the demonstration databases, including relevant SQL files, see the *IBM Informix: DB–Access User’s Guide*. For conceptual information about demonstration databases, see the *IBM Informix: Database Design and Implementation Guide*.

The sales_demo Database (XPS)

Your database server product contains SQL scripts for the **sales_demo** dimensional database. The **sales_demo** database provides an example of a simple data-warehousing environment and works in conjunction with the **stores_demo** database. The scripts for the **sales_demo** database create new tables and add extra rows to the **items** and **orders** tables of **stores_demo**.

To create the **sales_demo** database, you must first create the **stores_demo** database with the logging option. Once you create the **stores_demo** database, you can execute the scripts that create and load the **sales_demo** database from DB–Access. The files are named **createdw.sql** and **loaddw.sql**.

Dimensional Model of the sales_demo Database

Figure B-1 gives an overview of the tables in the **sales_demo** database.

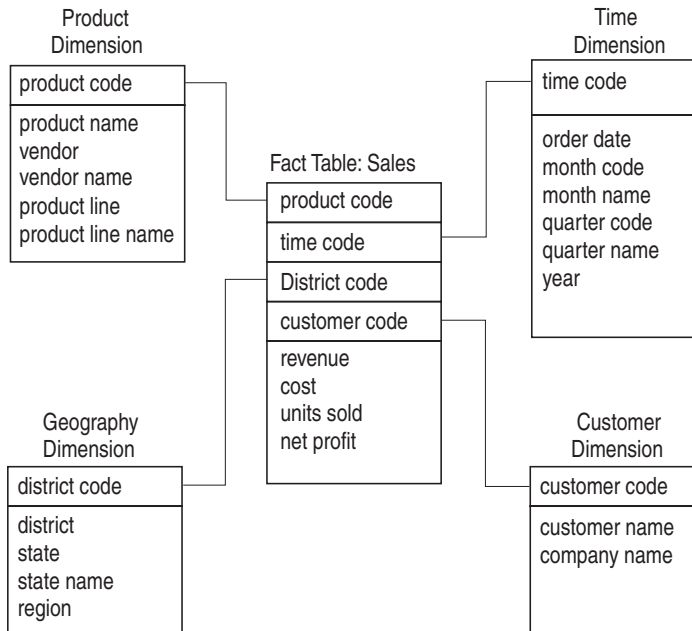


Figure B-1. The sales_demo Dimensional Data Model

For information on how to create and populate the **sales_demo** database, see the *IBM Informix: DB–Access User’s Guide*. For information on how to design and implement dimensional databases, see the *IBM Informix: Database Design and Implementation Guide*. For information on the **stores_demo** database, see Appendix A

The next section describes the schema of these five tables, and identifies the column in the fact table that logically associates each dimension table to the fact table.

Structure of the sales_demo Tables

The **sales_demo** database includes the following tables:

- **customer**
- **geography**
- **product**
- **sales**
- **time**

The tables are listed alphabetically, not in the order in which they are created. The **customer**, **geography**, **product**, and **time** tables are the dimensions for the **sales** fact table.

The **sales_demo** database is not an ANSI-compliant database.

The following sections describe the column names, data types, and column descriptions for each table. A SERIAL field serves as the primary key for the **district_code** column of the **geography** table. The primary and foreign key relationships that exist between the fact (**sales**) table and its dimension tables are not defined, however, because data-loading performance improves dramatically when the database server does not enforce constraint checking.

The customer Table

The **customer** table contains information about sales customers. Table B-1 shows the columns of the **customer** table.

Table B-1. The customer Table

Name	Type	Description
customer_code	INTEGER	Customer code
customer_name	CHAR(31)	Customer name
company_name	CHAR(20)	Company name

The geography Table

The **geography** table contains information about the sales district and region. Table B-2 shows the columns of the **geography** table.

Table B-2. The geography Table

Name	Type	Description
district_code	SERIAL	District code
district_name	CHAR(15)	District name
state_code	CHAR(2)	State code
state_name	CHAR(18)	State name
region	SMALLINT	Region name

The product Table

The **product** table contains information about the products sold through the data warehouse. Table B-3 on page B-4 shows the columns of the **product** table.

Table B-3. The product Table

Name	Type	Description
product_code	INTEGER	Product code
product_name	CHAR(31)	Product name
vendor_code	CHAR(3)	Vendor code
vendor_name	CHAR(15)	Vendor name
product_line_code	SMALLINT	Product line code
product_line_name	CHAR(15)	Name of product line

The sales Table

The **sales** fact table contains information about product sales and has a pointer to each dimension table. For example, the **customer_code** column references the **customer** table, the **district_code** column references the **geography** table, and so on. The **sales** table also contains the measures for the units sold, revenue, cost, and net profit. Table B-4 shows the columns of the **sales** table.

Table B-4. The sales Table

Name	Type	Description
customer_code	INTEGER	Customer code (references customer)
district_code	SMALLINT	District code (references geography)
time_code	INTEGER	Time code (references time)
product_code	INTEGER	Product code (references product)
units_sold	SMALLINT	Number of units sold
revenue	MONEY(8,2)	Amount of sales revenue
cost	MONEY(8,2)	Cost of sale
net_profit	MONEY(8,2)	Net profit of sale

The time Table

The **time** table contains time information about the sale. Table B-5 on page B-5 shows the columns of the **time** table.

Table B-5. The time Table

Name	Type	Description
time_code	INTEGER	Time code
order_date	DATE	Order date
month_code	SMALLINT	Month code
month_name	CHAR(10)	Name of month
quarter_code	SMALLINT	Quarter code
quarter_name	CHAR(10)	Name of quarter
year	INTEGER	Year

The superstores_demo Database (IDS)

SQL files and user-defined routines (UDRs) that are provided with DB–Access let you derive the **superstores_demo** object-relational database.

The **superstores_demo** database uses the default locale and is not ANSI compliant.

This section provides the following **superstores_demo** information:

- The structure of all the tables in the **superstores_demo** database
- A list and definition of the extended data types that **superstores_demo** uses
- A map of table hierarchies
- The primary-foreign key relationships among the columns in the database tables

For information on how to create and populate the **superstores_demo** database, see the *IBM Informix: DB–Access User’s Guide*. For information on how to work with object-relational databases, see the *IBM Informix: Database Design and Implementation Guide*. For information on the **stores_demo** database on which **superstores_demo** is based, see Appendix A.

Structure of the superstores_demo Tables

The **superstores_demo** database includes the following tables. Although many tables have the same name as **stores_demo** tables, they are different. The tables are listed alphabetically, not in the order in which they are created.

- **call_type**
- **catalog**
- **cust_calls**
- **customer**
 - **retail_customer** (*new*)

- **whlsale_customer** (*new*)
- **items**
- **location** (*new*)
 - **location_non_us** (*new*)
 - **location_us** (*new*)
- **manufact**
- **orders**
- **region** (*new*)
- **sales_rep** (*new*)
- **state**
- **stock**
- **stock_discount** (*new*)
- **units** (*new*)

This section lists the names, data types, and descriptions of the columns for each table in the **superstores_demo** database. The unique identifying value for each table (primary key) is shaded. Columns that represent extended data types are discussed in “User-Defined Routines and Extended Data Types” on page B-16. Primary-foreign key relationships between the tables are outlined in “Referential Relationships” on page B-19.

The call_type Table

The call codes associated with customer calls are stored in the **call_type** table. Table B-6 shows the columns of the **call_type** table.

Table B-6. The call_type Table

Name	Type	Description
call_code	CHAR(1)	Call code
codel_descr	CHAR (30)	Description of call code

The catalog Table

The **catalog** table describes each item in stock. Retail stores use this table when placing orders with the distributor. Table B-7 on page B-7 shows the columns of the **catalog** table.

Table B-7. The catalog Table

Name	Type	Description
catalog_num	SERIAL(1001)	System-generated catalog number
stock_num	SMALLINT	Distributor stock number (foreign key to stock table)
manu_code	CHAR(3)	Manufacturer code (foreign key to stock table)
unit	CHAR(4)	Unit by which item is ordered (foreign key to stock table)
advert	ROW (picture BLOB, caption LVARCHAR)	Picture of item and caption
advert_descr	CLOB	Tag line underneath picture

The cust_calls Table

All customer calls for information on orders, shipments, or complaints are logged. The **cust_calls** table contains information about these types of customer calls. Table B-8 shows the columns of the **cust_calls** table.

Table B-8. The cust_calls Table

Name	Type	Description
customer_num	INTEGER	Customer number (foreign key to customer table)
call_dtime	DATETIME YEAR TO MINUTE	Date and time call received
user_id	CHAR(18)	Name of person logging call (default is user login name)
call_code	CHAR(1)	Type of call (foreign key to call_type table)
call_descr	CHAR(240)	Description of call
res_dtime	DATETIME YEAR TO MINUTE	Date and time call resolved
res_descr	CHAR(240)	Description of how call was resolved

The customer, retail_customer, and whlsale_customer Tables

In this hierarchy, **retail_customer** and **whlsale_customer** are subtables that are created under the **customer** supertable, as Figure B-2 on page B-18 shows.

For information about table hierarchies, see the *IBM Informix: Database Design and Implementation Guide*.

The customer Table: The **customer** table contains information about the retail stores that place orders from the distributor. Table B-9 shows the columns of the **customer** table.

Table B-9. The customer Table

Name	Type	Description
customer_num	SERIAL	Unique customer identifier
customer_type	CHAR(1)	Code to indicate type of customer: R = retail W = wholesale
customer_name	name_t	Name of customer
customer_loc	INTEGER	Location of customer (foreign key to location table)
contact_dates	LIST(DATETIME YEAR TO DAY NOT NULL)	Dates of contact with customer
cust_discount	percent	Customer discount
credit_status	CHAR(1)	Customer credit status: D = deadbeat L = lost N = new P = preferred R = regular

The retail_customer Table: The **retail_customer** table contains general information about retail customers. Table B-10 on page B-9 shows the columns of the **retail_customer** table.

Table B-10. The retail_customer Table

Name	Type	Description
customer_num	SERIAL	Unique customer identifier
customer_type	CHAR(1)	Code to indicate type of customer: R = retail W = wholesale
customer_name	name_t	Name of customer
customer_loc	INTEGER	Location of customer
contact_dates	LIST(DATETIME YEAR TO DAY NOT NULL)	Dates of contact with customer
cust_discount	percent	Customer discount
credit_status	CHAR(1)	Customer credit status: D = deadbeat L = lost N = new P = preferred R = regular
credit_num	CHAR(19)	Credit card number
expiration	DATE	Expiration data of credit card

The whlsale_customer Table: The **whlsale_customer** table contains general information about wholesale customers. Table B-11 on page B-10 shows the columns of the **whlsale_customer** table.

Table B-11. The *whlsale_customer* Table

Name	Type	Description
customer_num	SERIAL	Unique customer identifier
customer_type	CHAR(1)	Code to indicate type of customer: R = retail W = wholesale
customer_name	name_t	Name of customer
customer_loc	INTEGER	Location of customer
contact_dates	LIST(DATETIME YEAR TO DAY NOT NULL)	Dates of contact with customer
cust_discount	percent	Customer discount
credit_status	CHAR(1)	Customer credit status: D = deadbeat L = lost N = new P = preferred R = regular
resale_license	CHAR(15)	Resale license number
terms_net	SMALLINT	Net term in days

The items Table

An order can include one or more items. One row exists in the **items** table for each item in an order. Table B-12 shows the columns of the **items** table.

Table B-12. The *items* Table

Name	Type	Description
item_num	SMALLINT	Sequentially assigned item number for an order
order_num	INT8	Order number (foreign key to orders table)
stock_num	SMALLINT	Stock number for item (foreign key to stock table)
manu_code	CHAR(3)	Manufacturer code for item ordered (foreign key to stock table)
unit	CHAR(4)	Unit by which item is ordered (foreign key to stock table)
quantity	SMALLINT	Quantity ordered (value must be > 1)
item_subtotal	MONEY(8,2)	Quantity ordered * unit price = total price of item

The location, location_non_us, and location_us Tables

In this hierarchy, **location_non_us** and **location_us** are subtables that are created under the **location** supertable, as shown in the diagram in “Table Hierarchies” on page B-18. For information about table hierarchies, see the *IBM Informix: Database Design and Implementation Guide*.

The location Table

The **location** table contains general information about the locations (addresses) that the database tracks. Table B-13 shows the columns of the **location** table.

Table B-13. The location Table

Name	Type	Description
location_id	SERIAL	Unique identifier for location
loc_type	CHAR(2)	Code to indicate type of location
company	VARCHAR(20)	Name of company
street_addr	LIST(VARCHAR(25) NOT NULL)	Street address
city	VARCHAR(25)	City for address
country	VARCHAR(25)	Country for address

The location_non_us Table

The **location_non_us** table contains specific address information for locations (addresses) that are outside the United States. Table B-14 shows the columns of the **location_non_us** table.

Table B-14. The location_non_us Table

Name	Type	Description
location_id	SERIAL	Unique identifier for location
loc_type	CHAR(2)	Code to indicate type of location
company	VARCHAR(20)	Name of company
street_addr	LIST(VARCHAR(25) NOT NULL)	Street address
city	VARCHAR(25)	City for address
country	VARCHAR(25)	Country for address
province_code	CHAR(2)	Province code
zipcode	CHAR(9)	Zip code
phone	CHAR(15)	Phone number

The location_us Table

The **location_us** table contains specific address information for locations (addresses) that are in the United States. Table B-15 shows the columns of the **location_us** table.

Table B-15. The location_us Table

Name	Type	Description
location_id	SERIAL	Unique identifier for location
loc_type	CHAR(2)	Code to indicate type of location
company	VARCHAR(20)	Name of company
street_addr	LIST(VARCHAR(25) NOT NULL)	Street address
city	VARCHAR(25)	City for address
country	VARCHAR(25)	Country for address
state_code	CHAR(2)	State code (foreign key to state table)
zip	CHAR(9)	Zip code
phone	CHAR(15)	Phone number

The manufact Table

Information about the manufacturers whose sporting goods are handled by the distributor is stored in the **manufact** table. Table B-16 shows the columns of the **manufact** table.

Table B-16. The manufact Table

Name	Type	Description
manu_code	CHAR(3)	Manufacturer code
manu_name	VARCHAR(15)	Name of manufacturer
lead_time	INTERVAL DAY(3) TO DAY	Lead time for shipment of orders
manu_loc	INTEGER	Manufacturer location (foreign key to location table)
manu_account	CHAR(32)	Distributor account number with manufacturer
account_status	CHAR(1)	Status of account with manufacturer
terms_net	SMALLINT	Distributor terms with manufacturer (in days)
discount	percent	Distributor volume discount with manufacturer

The orders Table

The **orders** table contains information about orders placed by the customers of the distributor. Table B-17 shows the columns of the **orders** table.

Table B-17. The orders Table

Name	Type	Description
order_num	SERIAL8(1001)	System-generated order number
order_date	DATE	Date order entered
customer_num	INTEGER	Customer number (foreign key to customer table)
shipping	ship_t	Special shipping instructions
backlog	BOOLEAN	Indicates order cannot be filled because the item is back ordered
po_num	CHAR(10)	Customer purchase order number
paid_date	DATE	Date order paid

The region Table

The **region** table contains information about the sales regions for the distributor. Table B-18 shows the columns of the **region** table.

Table B-18. The region Table

Name	Type	Description
region_num	SERIAL	System-generated region number
region_name	VARCHAR(20) UNIQUE	Name of sales region
region_loc	INTEGER	Location of region office (foreign key to location table)

The sales_rep Table

The **sales_rep** table contains information about the sales representatives for the distributor. Table B-19 on page B-14 shows the columns of the **sales_rep** table.

Table B-19. The sales_rep Table

Name	Type	Description
rep_num	SERIAL(101)	System-generated sales rep number
name	name_t	Name of sales rep
region_num	INTEGER	Region in which sales rep works (foreign key to the region table)
home_office	BOOLEAN	Home office location of sales rep
sales	SET(ROW (month DATETIME YEAR TO MONTH, amount MONEY) NOT NULL)	Amount of monthly sales for rep
commission	percent	Commission rate for sales rep

The state Table

The **state** table contains the names and postal abbreviations, as well as sales tax information, for the 50 states of the United States. Table B-20 shows the columns of the **state** table.

Table B-20. The state Table

Name	Type	Description
code	CHAR(2)	State code
sname	CHAR(15)	State name
sales_tax	percent	State sales tax

The stock Table

The **stock** table is a catalog of the items sold by the distributor. Table B-21 on page B-15 shows the columns of the **stock** table.

Table B-21. The stock Table

Name	Type	Description
stock_num	SMALLINT	Stock number that identifies type of item
manu_code	CHAR(3)	Manufacturer code (foreign key to manufact)
unit	CHAR(4)	Unit by which item is ordered
description	VARCHAR(15)	Description of item
unit_price	MONEY(6,2)	Unit price
min_reord_qty	SMALLINT	Minimum reorder quantity
min_inv_qty	SMALLINT	Quantity of stock below which item should be reordered
manu_item_num	CHAR(20)	Manufacturer item number
unit_cost	MONEY(6,2)	Distributor cost per unit of item from manufacturer
status	CHAR(1)	Status of item: A = active D = discontinued N = no order
bin_num	INTEGER	Bin number
qty_on_hand	SMALLINT	Quantity in stock
bigger_unit	CHAR(4)	Stock unit for next larger unit (for same stock_num and manu_code)
per_bigger_unit	SMALLINT	How many of this item in bigger_unit

The stock_discount Table

The **stock_discount** table contains information about stock discounts. (There is no primary key). Table B-22 on page B-16 shows the columns of the **stock_discount** table.

Table B-22. The *stock_discount* Table

Name	Type	Description
discount_id	SERIAL	System-generated discount identifier
stock_num	SMALLINT	Distributor stock number (part of foreign key to stock table)
manu_code	CHAR(3)	Manufacturer code (part of foreign key to stock table)
unit	CHAR(4)	Unit by which item is ordered (each, pair, case, and so on) (foreign key to units table; part of foreign key to stock table)
unit_discount	percent	Unit discount during sale period
start_date	DATE	Discount start date
end_date	DATE	Discount end date

The **units** Table

The **units** table contains information about the units in which the inventory items can be ordered. Each item in the **stock** table is available in one or more types of container. Table B-23 shows the columns of the **units** table.

Table B-23. The *units* Table

Name	Type	Description
unit_name	CHAR(4)	Units by which an item is ordered (each, pair, case, box)
unit_descr	VARCHAR(15)	Description of units

User-Defined Routines and Extended Data Types

The **superstores_demo** database uses *user-defined routines* (UDRs) and extended data types.

A UDR is a routine that you define that can be invoked within an SQL statement or another UDR. A UDR can either return values or not.

The data type system of Dynamic Server is an extensible and flexible system that supports the creation of following kinds of data types:

- Extensions of existing data types, by redefining some of the behavior for data types that the database server provides
- Definitions of customized data types by a user

This section lists the extended data types and UDRs created for the **superstores_demo** database. For information about creating and using UDRs and extended data types, see *IBM Informix: User-Defined Routines and Data Types Developer's Guide*.

The **superstores_demo** database creates the *distinct* data type, percent, in a UDR, as follows:

```
CREATE DISTINCT TYPE percent AS DECIMAL(5,5);
DROP CAST (DECIMAL(5,5) AS percent);
CREATE IMPLICIT CAST (DECIMAL(5,5) AS percent);
```

The **superstores_demo** database creates the following *named row types*:

- **location** hierarchy:
 - **location_t**
 - **loc_us_t**
 - **loc_non_us_t**
- **customer** hierarchy:
 - **name_t**
 - **customer_t**
 - **retail_t**
 - **whlsale_t**
- **orders** table
 - **ship_t**

location_t definition

location_id	SERIAL
loc_type	CHAR(2)
company	VARCHAR(20)
street_addr	LIST(VARCHAR(25) NOT NULL)
city	VARCHAR(25)
country	VARCHAR(25)

loc_us_t definition

state_code	CHAR(2)
zip	ROW(code INTEGER, suffix SMALLINT)
phone	CHAR(18)

loc_non_us_t definition

province_code	CHAR(2)
zipcode	CHAR(9)
phone	CHAR(15)

name_t definition

first	VARCHAR(15)
last	VARCHAR(15)

customer_t definition

customer_num	SERIAL
customer_type	CHAR(1)
customer_name	name_t
customer_loc	INTEGER
contact_dates	LIST(DATETIME YEAR TO DAY NOT NULL)
cust_discount	percent
credit_status	CHAR(1)

retail_t definition

credit_num	CHAR(19)
expiration	DATE

whlsale_t definition

resale_license	CHAR(15)
terms_net	SMALLINT

ship_t definition

date	DATE
weight	DECIMAL(8,2)
charge	MONEY(6,2)
instruct	VARCHAR(40)

Table Hierarchies

Figure B-2 shows how the hierarchical tables of the **superstores_demo** database are related. See “The customer and location Tables” on page B-20 for an explanation of the foreign key and primary relationships between those two tables, which are indicated by shaded arrows that point from the **customer.custnum** and **customer.loc** columns to the **location.location_id** columns in the following diagram.

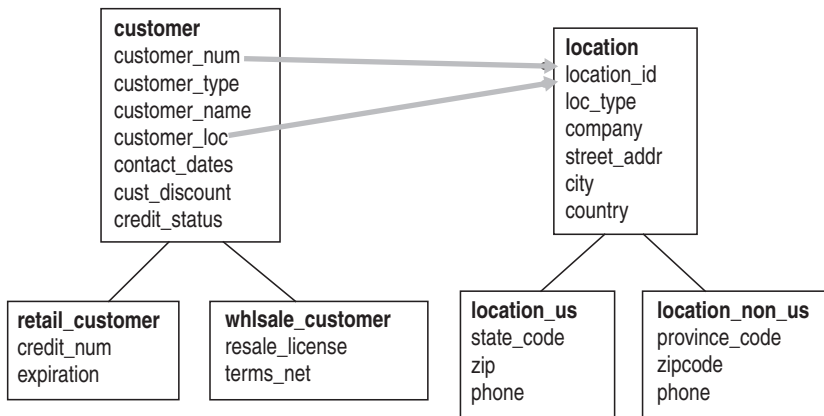


Figure B-2. Hierarchies of superstores_demo Tables

Referential Relationships

The tables of the **superstores_demo** database are linked by the primary-foreign key relationships that are identified in this section. This type of relationship is called a *referential constraint* because a foreign key in one table *references* the primary key in another table.

The customer and orders Tables

The **customer** table contains a **customer_num** column that holds a number that identifies a customer. The **orders** table also contains a **customer_num** column that stores the number of the customer who placed a particular order. In the **orders** table, the **customer_num** column is a foreign key that references the **customer_num** column in the **customer** table.

The orders and items Tables

The **orders** and **items** tables are linked by an **order_num** column that contains an identification number for each order. If an order includes several items, the same order number appears in several rows of the **items** table. In the **items** table, the **order_num** column is a foreign key that references the **order_num** column in the **orders** table.

The items and stock Tables

The **items** table and the **stock** table are joined by three columns: the **stock_num** column, which stores a stock number for an item, the **manu_code** column, which stores a code that identifies the manufacturer, and the **units** column, which identifies the types of unit in which the item can be ordered. You need the stock number, the manufacturer code, and the units to uniquely identify an item. The same stock number and manufacturer code can appear in more than one row of the **items** table, if the same item belongs to separate orders. In the **items** table, the **stock_num**, **manu_code**, and **unit** columns are foreign keys that reference the **stock_num**, **manu_code**, and **unit** columns in the **stock** table.

The stock and catalog Tables

The **stock** table and **catalog** table are joined by three columns: the **stock_num** column, which stores a stock number for an item, the **manu_code** column, which stores a code that identifies the manufacturer, and the **unit** column, which identifies the type of units in which the item can be ordered. You need all three columns to uniquely identify an item. In the **catalog** table, the **stock_num**, **manu_code**, and **unit** columns are foreign keys that reference the **stock_num**, **manu_code**, and **unit** columns in the **stock** table.

The stock and manufact Tables

The **stock** table and the **manufact** table are joined by the **manu_code** column. The same manufacturer code can appear in more than one row of the **stock** table if the manufacturer produces more than one piece of equipment. In the

stock table, the **manu_code** column is a foreign key that references the **manu_code** column in the **manufact** table.

The cust_calls and customer Tables

The **cust_calls** table and the **customer** table are joined by the **customer_num** column. The same customer number can appear in more than one row of the **cust_calls** table if the customer calls the distributor more than once with a problem or question. In the **cust_calls** table, the **customer_num** column is a foreign key that references the **customer_num** column in the **customer** table.

The call_type and cust_calls Tables

The **call_type** and **cust_calls** tables are joined by the **call_code** column. The same call code can appear in more than one row of the **cust_calls** table, because many customers can have the same type of problem. In the **cust_calls** table, the **call_code** column is a foreign key that references the **call_code** column in the **call_type** table.

The state and customer Tables

The **state** table and the **customer** table are joined by a column that contains the state code. This column is called **code** in the **state** table and **state** in the **customer** table. If several customers live in the same state, the same state code appears in several rows of the table. In the **customer** table, the **state** column is a foreign key that references the **code** column in the **state** table.

The customer and location Tables

In the **customer** table, the **customer_loc** column is a foreign key that references the **location_id** of the **location** table. The **customer_loc** and **location_id** columns each uniquely identify the customer location.

The manufact and location Tables

The **manu_loc** column in the **manufact** table is a foreign key that references the **location_id** column, which is the primary key in the **location** table. Both **manu_loc** and **location_id** uniquely identify the manufacturer location.

The state and location_us Tables

The **state** and **location_us** tables are joined by the column that contains the state code. The **state_code** column in the **location_us** table is a foreign key that references the **code** column in the **state** table.

The sales_rep and region Tables

The **region_num** column is the primary key in the **region** table. It is a system-generated region number. The **region_num** column in the **sales_rep** table is a foreign key that references and joins the **region_num** column in the **region** table.

The region and location Tables

The **region_loc** column in the **region** table identifies the regional office location. It is a foreign key that references the **location_id** column in the **location** table, which is a unique identifier for location.

The stock and stock_discount Tables

The **stock** table and the **stock_discount** table are joined by three columns: **stock_num**, **manu_code**, and **unit**. These columns form the primary key for the **stock** table. The **stock_discount** table has no primary key and references the **stock** table.

The stock and units Tables

The **unit_name** column of the **units** table is a primary key that identifies the kinds of units that can be ordered, such as case, pair, box, and so on. The **unit** column of the **stock** table joins the **unit_name** column of the **units** table.

Appendix C. Accessibility

The syntax diagrams in the HTML version of this manual are available in dotted decimal syntax format, which is an accessible format that is available only if you are using a screen reader.

Dotted Decimal Syntax Diagrams

In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), the elements can appear on the same line, because they can be considered as a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read punctuation. All syntax elements that have the same dotted decimal number (for example, all syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, the word or symbol is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is read as 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol that provides information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements

must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, this identifies a reference that is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you should refer to a separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

- ? Specifies an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element (for example, 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.
- ! Specifies a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicates that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP only applies to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.
- * Specifies a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data-area, you know that you can include more than one data area or

you can include none. If you hear the lines 3*, 3 HOST, and 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you could write HOST STATE, but you could not write HOST HOST.
3. The * symbol is equivalent to a loop-back line in a railroad syntax diagram.

+ Specifies a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times. For example, if you hear the line 6.1+ data-area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. As for the * symbol, you can only repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back line in a railroad syntax diagram.

Glossary

Tip: For additional product-specific information, refer to the glossaries provided in the *IBM Informix: Storage Manager Administrator's Guide*, the *IBM Informix: SNMP Subagent Guide*, and other manuals in the IBM Informix documentation set.

8-bit character. A single-byte character that consists of eight bits, which means that the code point is in the range 128 through 255. Examples from the ISO8859-1 code set include the non-English é, ñ, and ö characters. They can be interpreted correctly only by software that is *8-bit clean*. See also non-ASCII character.

8-bit clean. An attribute of software that can process character data that contains *8-bit characters*. The operating system or the database server reads the eighth bit as part of the code value. In other words, it does not ignore the eighth bit nor make its own interpretation of the eighth bit.

16-bit code set. Any *code set* that represents each character by two bytes, so that approximately 65,000 distinct characters can be encoded. Also called *double-byte code set*. An example is JIS X0208.

access method. Any of a group of routines that access or manipulate a table or an index. In the output of a SET EXPLAIN statement, *access method* refers to the mode of table access in a query (for example, SEQUENTIAL SCAN as opposed to INDEX PATH). See also primary access method and secondary access method.

access privileges. The types of operations that a user has permission to perform in a specific database or database object. The Informix database, table, table-fragment, index, and column-access privileges are independent of operating-system access permissions.

active set. The collection of rows that satisfies a query associated with a cursor.

aggregate function. An SQL function that returns one value for a group of retrieved rows; for example, the frequency, sum, average, maximum, or minimum of an expression in a query or report. See also user-defined aggregate.

aggregate support function. One of a group of *user-defined functions* that the database server uses to calculate a user-defined aggregate.

alias. In an SQL query or in a form-specification file, an *alias* is a single-word temporary alternative name that is used in place of a qualified table name (for example, **t1** as an alias for *owner.table_name*). Aliases are often used in complex subqueries and are required for a self-join.

ALS. Legacy acronym for *Asian Language Support* (ALS), a feature for multibyte East Asian locales. Supplanted by *Global Language Support* (GLS).

ANSI. Acronym for the American National Standards Institute. This group sets standards in many areas, including the information technology industry and the SQL language. Some ANSI standards are also standards of the International Standards Organization (ISO).

ANSI compliant. For databases, conforming to ANSI/ISO standards for the SQL language. An IBM Informix database can be created as either ANSI compliant or not ANSI compliant. An ANSI-compliant database enforces certain ANSI requirements, such as implicit transactions, explicit owner naming, no public synonyms, and unbuffered logging, which are not enforced in databases that are not ANSI compliant.

API. See application programming interface (API).

application development tool. Software, such as IBM Informix 4GL, which you can use to create and maintain a database. The software allows a user to send instructions and data to (and receive information from) the database server.

application process. The process that manages an ESQL or other program at runtime. It executes the program logic and initiates SQL requests. Memory that is allocated for program variables, program data, and the fetch buffer is part of this process. *See also* database server process.

application-productivity tools. Software tools that are used to write application programs.

application program. An executable file or a logically related set of files.

application programming interface (API). A set of related software components, such as those provided by IBM, that a developer uses to create applications that communicate with a third-party product. An API can include a *library of functions*, *header files*, graphic interfaces, and application programs. *See also* SQL API and DataBlade API.

arbitrary rule. Expressions that define a fragmentation strategy. Unlike a *range rule*, an arbitrary rule can specify any relational and logical operators of SQL to define expressions (such as the OR operator to group data).

archiving. To copy all the data and indexes of a database to a different physical device from that which stores the database. Archived material can be used by the Database Administrator for recovery from a failure. *See also* backup.

argument. A value that is passed to a *routine* or command. *Compare with* parameter.

array. An ordered set of items of the same data type. An individual member of the array is referred to as an element and usually is identified by an integer index that gives the position of the element within the array.

ASCII. Acronym for American Standards Committee for Information Interchange, a code set of letters, digits, punctuation, and certain

other nonprintable and printable characters used in computers and telecommunication. It contains 128 characters, each of which can be represented with 7 bits of information. The code set of every Informix locale includes the ASCII characters as a subset. *See also* single-byte character.

ASF. Acronym for Associated Services Facility. The ASF code in IBM Informix products controls the connections between client applications and database servers. Software architects use this term; most users of IBM Informix products see this term only in occasional error messages.

Asian Language Support (ALS). A legacy technology to support the multibyte code sets of certain East Asian languages. The functionality of ALS is replaced by *Global Language Support* (GLS) in current IBM Informix products. For more information about ALS and GLS, see the *IBM Informix: Migration Guide*.

attached index. An index that has the same distribution scheme as its table. To create an attached index, specify no distribution scheme (no FRAGMENT BY clause) and no storage option (no IN clause), or else specify IN TABLE as the storage option in CREATE INDEX or ALTER FRAGMENT ON INDEX statements. For Dynamic Server, index pages for user indexes reside in separate tablespaces, but in the same tablespaces as the data pages to which they refer. Only system indexes reside in the same tablespace as the corresponding data pages. For Extended Parallel Server, user index and system index pages reside in separate tablespaces, but in the same tablespaces as the corresponding data pages. *See also* detached index, system index, user index.

audit event. Any database server activity or operation that can potentially access or alter data. Audit events can be recorded and monitored by the database *secure auditing* facility. Examples of audit events include accessing tables, altering indexes, dropping chunks, granting database access, updating the current row, running database utilities, and so forth. (For a complete list of audit events, see the *IBM Informix: Trusted Facility Guide*.)

audit file. A file that contains records of audit events and resides in the specified audit directory. Audit files provide an audit trail of information that can be extracted by the database *secure auditing facility* for analysis by the *database system security officer* (DSSO).

audit mask. A structure that specifies which audit events should be logged (or excluded from logging) by the database *secure auditing facility*.

auxiliary statements. A residual category of SQL statements, including statements like INFO, OUTPUT, and GET DIAGNOSTICS that can display information about the database.

B-tree. A model (with "simple graph" topology) for defining the logical structure of an index.

B-tree index. A type of index that uses a balanced tree structure for efficient record retrieval. A B-tree index is balanced when the leaf nodes are all at the same level from the root node. B-tree indexes store a list of rowids for any duplicate key value data in ascending or descending order. *See also* bitmap index and R-tree index.

backup. A duplicate of a computer file on another device or tape to preserve existing work, in case of a computer failure or other mishap. A *backup* refers to duplicating logical-log files, while *archiving* refers to duplicating data.

base table. (1) A table referenced in the definition of a view, containing data that the view displays. A *multitable view* has more than one base table. (2) Within a table hierarchy of Dynamic Server, a *parent table* from whose schema another table (the *child table*) is derived. *See also* table hierarchy.

base type. *See* opaque data type.

before-image. The image of a row, page, or other item before any changes are made to it.

bidirectional. Of a locale, supporting both right-to-left and left-to-right text.

big-endian. A hardware-determined storage method in which the most-significant byte of a multibyte number has the lowest address. *See also* little-endian.

bitmap index. A type of index that stores a bitmap for any highly duplicate key value. The bitmap indicates which rows have the duplicate key value. You create a bitmap index with the USING BITMAP keywords in the CREATE INDEX statement. *See also* B-tree index.

blob. A legacy term for a large object that is now known as a simple large object and includes TEXT or BYTE data types. These data objects effectively have no maximum size (theoretically as large as 2^{31} bytes). *See also* simple large object.

BLOB. Acronym for binary large object. A data type for a *smart large object* that stores any type of binary data, including images. These objects can be stored and retrieved in pieces and have database properties such as recovery and transaction rollback. *See also* CLOB.

blobpage. (Not for Extended Parallel Server) The unit of disk allocation within a blobspace. The database server administrator determines the size of a blobpage. The size can vary, depending on the size of the TEXT or BYTE data that the user inserts.

blobspace. (Not for Extended Parallel Server) A logical collection of *chunks* that is used to store TEXT and BYTE data.

Boolean. A variable or an expression that can take on the logical values TRUE (1), FALSE (0), or UNKNOWN (if NULL values are involved).

BOOLEAN. A built-in data type that supports single-byte true/false values. TRUE is represented internally as 0 and externally as t. FALSE is represented internally as 1 and externally as f. A NULL value is represented as NULL.

Boolean function. A function that returns a Boolean value (true or false). A Boolean function can act as a *filter*.

branch node. An index page that contains pointers to a leaf node or other branch nodes. The database server creates a branch node when the root node and subsequent leaf nodes become full.

buffer. A portion of computer memory where a program temporarily stores data. Data typically is read into or written out from buffers to disk.

buffered disk I/O. Disk I/O that is controlled by the operating system instead of by an application. With buffered disk I/O, the operating system stores data in the kernel portion of memory before periodically writing the data to disk. *See also* unbuffered disk I/O and disk I/O.

buffered logging. A type of logging that holds transactions in a memory buffer until the buffer is full, regardless of when the transaction is committed or rolled back. Informix database servers provide this option to speed up operations by reducing the number of disk writes.

built-in. Provided by the database server, usually in the system catalog; not defined by the user.

built-in data type. A predefined data type that the database server supports; for example, INTEGER, CHAR, or SERIAL.

byte. An physical computer storage unit of 8 bits. A character is not necessarily one byte. In a multibyte code set, a character can require more than one byte.

BYTE. A data type for a simple large object that stores any type of binary data and can be as large as 2^{31} bytes. *See also* TEXT.

Cartesian product. The set that results when you pair each and every member of one set with each and every member of another set. A Cartesian product results from a multiple-table query when you do not specify the joining conditions among tables. *See also* join.

cascading deletes. Deletion of rows from a child table that were associated by foreign key to a

row that is deleted from the parent table. When any rows are deleted from the primary key column of a table, cascading deletes, if enabled, delete identical information from any foreign-key column in a related table.

case sensitivity. The condition of distinguishing between uppercase and lowercase letters. Case-sensitive commands and command options react differently to the same letters entered as uppercase or lowercase characters. The locale files specify which characters (if any) of a code set are uppercase or lowercase.

cast. A database object that converts one data type to another. Most built-in data types have built-in casts (that is, system-defined casts) to compatible data types. *See also* user-defined cast, explicit cast, and implicit cast.

cast function. A user-defined function that implements a cast. The function must be registered with the CREATE CAST statement before it can be used.

character. A logical unit of storage for a code point. A character is equal to one or more bytes and can be numeric, alphabetic, or a nonprintable character (such as a control character). *See also* multibyte character and single-byte character.

character set. One or more natural-language alphabets, together with additional symbols for digits, punctuation, ligatures, diacritical marks, and whitespace. (A *natural language* is a written language that human beings use to communicate with each other, such as English, Chinese, or German.) *See also* code set.

character special device. *See* unbuffered disk I/O.

check constraint. A logical condition that must be satisfied before data values can be entered into a column of a database table during an INSERT or UPDATE statement.

checkpoint. A point during a database server operation when the pages on disk are

synchronized with the pages in the shared memory buffer pool. It can be a *full checkpoint* or a *fuzzy checkpoint*.

child table. The referencing table in a referential constraint. *See also* parent table.

chunk. The largest contiguous section of disk space available for a database server. A specified set of chunks defines a *dbspace* or *blobspace*. A database server administrator allocates a chunk to a *dbspace* or *blobspace* when that *dbspace* or *blobspace* approaches full capacity. A chunk contains a certain number of pages. (Extended Parallel Server does not support *blobspace* chunks.)

client application. A program that requests services from a server program, typically a file server or a database server. For the GLS feature, the term *client application* includes database server utilities.

client computer. The computer on which a client application runs.

client locale. The locale that a client application uses to perform read and write operations on the client computer. The **CLIENT_LOCALE** environment variable can specify a nondefault locale. *See also* server locale.

client/server architecture. A hardware and software design that allows the user interface and database server to reside on separate nodes or platforms on a single computer or over a network. *See also* ASE, client application, and server-processing locale.

client/server connection statements. The SQL statements that can make connections with databases. These statements include **CONNECT**, **DISCONNECT**, and **SET CONNECTION**.

CLOB. Acronym for character large object. A data type for a smart large object that stores large text items, such as PostScript or HTML files. These objects can be stored and retrieved in pieces and have database properties such as recovery and transaction rollback. *See also* BLOB.

close a cursor. To drop the association between a cursor and the active set of rows that results from a query.

close a database. To deactivate the connection between a client application and a database. Only one database can be active at a time.

close a file. To deactivate the association between a file and a program.

cluster an index. To rearrange the physical data of a table according to a specific index.

cluster key. The column in a table that logically relates a group of simple large objects or smart large objects that are stored in an optical cluster.

clustersize. The amount of space, specified in kilobytes, that is allocated to an optical cluster on an optical volume.

code point. A bit pattern that represents one character in a code set. For example, in the ASCII code set, the uppercase *A* character has a code point of 0x41.

code set. The representation of a character set that specifies how to map each element of a character set to a unique code point. For example, ASCII, ISO8859-1, Microsoft 1252, and EBCDIC are code sets to represent the English language. A locale name specifies a code set.

code-set conversion. The process of converting character data from one code set (the *source* code set) to another (the *target* code set). Code-set conversion is useful when the client and server computers use different code sets to represent the same character data.

code-set order. The serialized order of characters within a code set. For example, in the ASCII code set, uppercase characters (*A* through *Z*) are ordered before lowercase characters (*a* through *z*). *See also* collation order and localized order.

cogroup. A named group of coservers. At initialization, the database server creates a cogroup that is named **cogroup_all** from all configured *coservers*.

collating sequence. See collation order, code-set order, and localized order.

collation. The process of sorting character strings according to some order. The term is sometimes used as a synonym for collation order or for localized order.

collation order. The logical order in which the character-string values in a database are sorted and indexed. This is based on either the order of the code set or else some locale-specific order. Collating order is also known as collating sequence.

collection. An instance of a collection data type; a group of *elements* of the same *data type* stored in a SET, MULTISET, or LIST.

collection cursor. A database cursor that has an IBM Informix ESQL/C collection variable associated with it and provides access to the individual *elements* of a column whose data type is a *collection data type*.

collection data type. A complex data type whose instances are groups of *elements* of the same *data type*, which can be any *opaque data type*, *distinct data type*, *built-in data type*, *collection data type*, or *row type*.

collection-derived table. A table that IBM Informix ESQL/C or SPL creates for a collection column when it encounters the TABLE keyword in an INSERT, DELETE, UPDATE, or SELECT statement. ESQL/C and SPL store this table in a collection variable to access *elements* of the collection as rows of the collection-derived table.

collection subquery. A query that takes the result of a subquery and turns it into a expression by using the MULTISET keyword to convert returned values into a MULTISET collection.

collection variable. An IBM Informix ESQL/C *host variable* or *SPL variable* that holds an entire *collection* and provides access, through a *collection cursor*, to the individual *elements* of the collection.

collocated join. A join that occurs locally on the coserver where the data resides. The local coserver sends the data to the other coservers after the join is complete.

column. A data element that contains a specified type of information that occurs in every row of the table. Also known as a *display label* or a *field*. See also *row*.

column expression. An expression that includes a column name and optionally uses *column subscripts* to define a *column substring*.

column subscript. A subscript (an integer enclosed between brackets) in a column expression, showing the ordinal position of a byte within the column value. Two integers, separated by a comma, as in `col2[3,9]`, define a column substring.

column substring. A substring of a character column. For example, the column expression `col2[3,9]` specifies a 7-byte substring of `col2` that begins with the 3rd byte.

command file. A system file that contains one or more statements or commands, such as SQL statements.

comment. Information in a program file that is not processed by the computer but that documents the program. Special characters such as the sharp sign (`#`), braces (`{ }`), slash mark followed by asterisk (`/ *`), or a double hyphen (`--`) can be used to identify comments, depending on the programming context.

commit work. To complete a transaction by accepting all changes to the database since the beginning of the transaction. See also *roll back*.

Committed Read. An Informix level of isolation in which the user can view only rows that are currently committed at the moment when the query is requested; the user cannot view rows that were changed as a part of a currently uncommitted transaction. Committed Read is available through a database server and set with the SET ISOLATION statement. It is the default level of isolation for databases that are not ANSI compliant. See also *Read Committed*.

compatible data types. Two data types for which casts exist in the database. *See also* implicit cast.

compile. To translate source code (in a high-level language) into executable code. *Compare with* execute and link. *See also* source file.

compile-time error. An error that occurs when you *compile* the program source code. This type of error indicates syntax errors in the source code. *Compare with* runtime error.

complex data type. A *data type* that is built from a combination of other data types using an SQL type constructor and whose components can be accessed through SQL statements. *See also* row type and collection data type.

component. In the High-Performance Loader (HPL), the information required to load or unload data is organized in several *components*. The components are format, map, filter, query, project, device array, load job, and unload job.

composite data type. *See* row type.

composite index. An index constructed on two or more columns of a table. The ordering imposed by the composite index varies least frequently on the first-named column and most frequently on the last-named column.

composite join. A join between two or more tables based on the relationship among two or more columns in each table. *See also* simple join.

compressed bitmap. An indexing method that identifies records through a fragment identifier and a record identifier.

concatenate. To append a second string to the end of a first string.

concatenation operator. An operator whose notation is composed of two pipe symbols (||); this is used in expressions to indicate the joining of two strings.

concurrency. Access by two or more processes to the same database simultaneously.

configuration management (CM) coserver. A coserver designated to run CM software and store CM data.

configuration file. A file read during database server disk or shared-memory initialization that contains the parameters that specify values for configurable behavior. Database server and its archiving tool use configuration files.

connection. A logical association between two applications or between an application and a database environment, created by a CONNECT or DATABASE statement. Database servers can also have connections to one another. *See also* explicit connection, implicit connection, and multiplexed connection.

connection coserver. The coserver to which a client is directly connected. *See also* coserver and participating coserver.

connection redirector. An Extended Parallel Server feature, enabled by a setting in the **sqlhosts** file, whereby the database server attempts to establish a client connection with each coserver in a dserver group until a connection succeeds.

constant. A value that cannot change during the execution of a program or command. In some programming languages, a constant has a name that can be referenced. *Compare with* variable. *See also* literal.

constraint. A restriction on what kinds of data can be inserted or updated in tables. *See also* check constraint, primary-key constraint, referential constraint, not-NULL constraint, and unique constraint.

constructed data type. *See* complex data type.

constructor. *See* type constructor.

control character. A character whose occurrence in a particular context initiates, modifies, or stops a control function (an operation to control a device, for example, in moving a visual cursor or in reading data). In a program, you can define actions that use the CTRL key with another key to execute some programming action (for

example, entering CTRL-W to obtain online Help in IBM Informix products). A control character is sometimes referred to as a *control key*. Compare with printable character.

cooked files. See buffered disk I/O.

coordinating server. In a query that spans multiple database servers, the server in which the query is initiated is called the *coordinator* or *coordinating server*. This server is also sometimes called the *local server* because it is the local server to the client initiating the query. To respond to the query, the coordinating server starts sessions on the other servers involved in the query. See also distributed query, subordinate servers, and remote servers.

correlated subquery. A subquery (or inner SELECT) that depends on a value produced by the outer SELECT statement that contains it. Also a nested subquery whose WHERE clause refers to an attribute of a relation that is declared in an outer SELECT. Correlated subqueries reference one or more columns from one or more tables of a parent query and need to be evaluated once for each row in the parent query. See also independent subquery and subquery.

correlation name. The prefix used with a column name in a triggered action to refer to an old (before triggering statement) or a new (after triggering statement) column value. The associated column must be in the triggering table. See also trigger.

corrupted database. A database whose tables or indexes contain incomplete, inconsistent, or invalid data.

corrupted index. An index that does not correspond exactly to the data in its table.

coserver. The functional equivalent of a database server that operates on a single node. See also connection coserver and participating coserver.

current row. The most recently retrieved row of the active set of a query.

cross-server query. See distributed query.

cursor. In SQL, an identifier associated with a group of rows or with a collection data type. Conceptually, the pointer to the current row or collection element. You can use cursors for SELECT statements or EXECUTE PROCEDURE statements (associating the cursor with the rows returned by a query) or INSERT statements (associating the cursor with a buffer to insert multiple rows as a group). A select cursor is declared for sequential only (regular cursor) or nonsequential (scroll cursor) retrieval of rows. In addition, you can declare a select cursor for update (initiating locking control for updated and deleted rows) or WITH HOLD (so that completing the current transaction does not close the cursor). In ESQL/C, a cursor can be dynamic, meaning that it can be referenced by an identifier or by a character variable.

cursor function. A user-defined routine (UDR) that returns one or more rows of data and therefore requires a cursor to execute. An *SPL routine* is a cursor function when its RETURN statement contains the WITH RESUME keywords. An *external function* is a cursor function when it is defined as an iterator function. Compare with noncursor function.

cursor manipulation statements. The SQL statements that control cursors; specifically, the CLOSE, DECLARE, FETCH, FLUSH, OPEN, and PUT statements.

Cursor Stability. An Informix level of isolation available through the SET ISOLATION statement in which the database server must secure a shared lock on a fetched row before the row can be viewed. The database server retains the lock until it receives a request to fetch a new row. See also isolation.

data access statements. The subset of SQL statements that you can use to grant and revoke permissions and to lock tables.

data definition statements. The subset of SQL statements (sometimes called *data definition language*, or DDL) to create, alter, drop, and rename data objects, including databases, tables, views, synonyms, triggers, sequences, and user-defined routines.

data dictionary. The set of tables that keeps track of the structure of the database and the inventory of database objects. This is also called the system catalog. Each database that a database server supports has its own system catalog.

data distribution. A mapping of the data values within a column into a set of categories that are equivalent to a histogram or to a frequency distribution. You can use the UPDATE STATISTICS statement (specifying the MEDIUM or HIGH keyword options) to create data distributions.

data integrity. The process of ensuring that data corruption does not occur when multiple users simultaneously try to alter the same data. Locking, constraints, and transaction logging are used to control data integrity.

data integrity statements. SQL statements that you use to control transactions and audits. Data integrity statements also include statements for repairing and recovering tables.

data manipulation statements. The SQL statements that can query tables, insert into tables, delete from tables, or update tables (select, insert, delete, update). The load and unload utilities also are sometimes called data manipulation statements.

data partitioning . *See* table fragmentation.

data replication. The ability to allow database objects to have more than one representation at more than one distinct site.

data restriction. Synonym for *constraint*.

data type. A descriptor assigned to each column in a table or program variable, which indicates the type of data the column or program variable is intended to hold. Informix data types are discussed in Chapter 2, "Data Types." Informix data types for Global Language Support are discussed in the *IBM Informix: GLS User's Guide*. *See also* built-in data type, complex data type, distinct data type, opaque data type, and user-defined data type.

database. A collection of information (contained in tables) that is used for a specific purpose. *See also* relational database.

Database Administrator. *See* DBA.

database application. A program that applies database management techniques to implement specific data manipulation and reporting tasks.

database environment. Used in the CONNECT statement, either the database server or the database server-and-database to which a user connects.

database locale. The locale that defines the code set, collation, and date, time, number, and currency display conventions of a database server. The **DB_LOCALE** environment variable can specify this locale.

database management system. *See* DBMS.

database object. An SQL entity that is recorded in a system catalog table, such as a table, column, constraint, access method, *default value*, *cast*, *index*, *operator class*, *prepared statement*, *privilege*, *role*, *sequence*, *synonym*, *trigger*, *user-defined aggregate*, *user-defined cast*, *user-defined data type*, *user-defined routine*, or *view*.

database server. A software package that manages access to one or more databases for one or more client applications. *See also* relational database server.

database server process. A virtual processor that functions similarly to a CPU in a computer. *See also* application process.

database server utility. A program that performs a specific task. For example, **DB-Access**, **dbexport**, and **onmode** are Informix database server utilities.

DataBlade API. An application programming interface (API) that allows a C *user-defined routine* access to the client application.

DataBlade module. A group of *database objects* and supporting code that extends an object-relational database to manage new kinds

of data or add new features. A DataBlade module can include new data types, *routines*, *casts*, *aggregates*, access methods, SQL code, client code, and installation programs.

DBA. Acronym for *Database Administrator*. The DBA is responsible for the contents and use of a database, whereas the database server administrator (DBSA) is responsible for managing one or more database servers. Also a level of privilege, typically for operations that most users are not authorized to perform.

DBA-privileged. A class of SPL routines that only a user with DBA database privileges creates.

DBMS. Acronym for *database management system*. These are all the components necessary to create and maintain a database, including the application development tools and the database server.

dbserver group. A collection of coservers defined and named by entries in the `sqlhosts` file. Dbserver groups make multiple coservers into a single logical entity for establishing or changing client/server connections.

dbslice. A named set of dbspaces that can span multiple coservers. A dbslice is managed as a single storage object. *See also* logslice, physslice, and rootslice.

dbspace. A logical collection of one or more chunks. Because chunks represent specific regions of disk space, the creators of databases and tables can control where their data is physically located by placing databases or tables in specific dbspaces. A dbspace provides a link between logical (such as tables) and physical units (such as chunks) of storage. *See also* root dbspace.

DDL. Acronym for data definition language, a subset of the Structured Query Language (SQL) for declaring, modifying, and dropping database objects (such as tables, constraints, or indexes). *See also* data definition statements.

deadlock. A situation in which two or more threads cannot proceed because each is waiting for data locked by the other (or another) thread.

The database server monitors and prevents potential deadlock situations by sending an error message to the application if a request for a lock might result in a deadlock.

debug file. A file that receives output used for debugging purposes.

decision-support application. An application that provides information that is used for strategic planning, decision-making, and reporting. It typically executes in a batch environment in a sequential scan fashion and returns a large fraction of the rows scanned. Decision-support queries typically scan the entire database. *See also* online transaction processing application.

decision-support query. A query that a decision-support application generates. It often requires multiple joins, temporary tables, and extensive calculations, and can benefit significantly from PDQ. *See also* online transaction processing queries.

declaration statement. A programming language statement that describes or defines objects; for example, defining a program variable. *Compare with* procedure. *See also* data definition statements.

default. Values or behavior that take effect unless the user explicitly specifies another value or action.

default locale. The locale that a product uses unless you specify a different (nondefault) locale. For IBM Informix products, U.S. English is the default locale.

default value. A value that is used when no explicit value is specified. For example, you can assign default values to columns with the ALTER TABLE and CREATE TABLE statements and to variables in *SPL routines*.

delete. To remove any row or combination of rows with the DELETE statement.

delimited identifier. If the DELIMITED environment variable is set, this is an SQL identifier enclosed between double (") quotation

marks. This supports identifiers that are SQL-reserved keywords or that contain whitespace or other characters outside the default SQL character set for identifiers.

delimiter. A character used as a boundary on a field or as the terminator for a column or row. Some files and prepared objects require semicolon (;), comma (,), blank space (), or tab delimiters between statements. SQL statements can have semicolon or other delimiters in some programming contexts.

deluxe mode. A method of loading or unloading data that uses regular inserts.

descriptor. A quoted string or variable that identifies an allocated system-descriptor area or an sqllda structure. It is used for the Informix SQL APIs. *See also* identifier.

detached index. An index whose distribution scheme (specified by the FRAGMENT BY clause) and storage option (specified by the IN clause) of the CREATE INDEX or ALTER FRAGMENT ON INDEX statement differ from the distribution scheme of the underlying table. Index pages reside in separate dbspaces from the corresponding data pages. *Compare with* attached index.

device array. A list of I/O devices. *See also* component.

diagnostic area. A data structure (sometimes called sqllda) that stores diagnostic information about an executed SQL statement.

diagnostics table. A special table that holds information about the integrity violations caused by each row in a violations table. You use the START VIOLATIONS TABLE statement to create violations tables and diagnostics tables and associate them with a base table.

Dirty Read. An Informix isolation level set with the SET ISOLATION statement that disregards locks and allows viewing of any existing rows, even rows that currently can be altered from inside an uncommitted transaction. Dirty Read is

the lowest level of isolation (no isolation at all), and is thus the most efficient. *See also* Read Uncommitted.

disabled mode. The object mode in which a database object is disabled. When a constraint, index, or trigger is in the disabled mode, the database server acts as if the object does not exist and does not take it into consideration during the execution of data manipulation statements.

disk configuration. The organization of data on a disk; also refers to the process of preparing a disk to store data.

disk I/O. The process of transferring data between memory and disk. The I/O refers to input/output.

display label. A temporary name for a column or expression in a query.

distinct data type. A *data type* that you declare with the CREATE DISTINCT TYPE statement. A distinct data type has the same internal storage representation as its *source type* (an existing *opaque data type*, *built-in data type*, *named row type*, or *distinct data type*) but a different name, and can have different casts and routines. To compare a distinct data type with its source type requires an *explicit cast*. A distinct data type inherits all routines that are defined on its source type.

distributed query. A query that accesses data from a database other than the current database.

distribution. *See* data distribution.

distribution scheme. *See* table fragmentation.

DLL. *See* dynamic link library (DLL).

DML. Acronym for data manipulation language. *See also* data manipulation statements.

dominant table. *See* outer join.

DRDA. Acronym for Distributed Relational Database Architecture. DRDA is an IBM-defined set of protocols that software manufacturers can

follow to develop connectivity solutions between heterogeneous relational database management environments.

DSS. Acronym for Decision Support System. *See also* decision-support application.

duplicate index. An index that allows duplicate values in the indexed column.

dynamic link library (DLL). A *shared-object file* on a Windows system. *See also* shared library.

dynamic management statements. The SQL statements that describe, execute, and prepare other statements.

dynamic routine-name specification. The execution of a *user-defined routine* whose name is determined at runtime through an *SPL variable* in the EXECUTE PROCEDURE, EXECUTE ROUTINE, or EXECUTE FUNCTION statement.

Dynamic Server instance. The set of processes, storage spaces, and shared memory that together comprise a complete database server. A single Dynamic Server instance can support more than one database.

dynamic SQL. The statements and structures that allow a program to form an SQL statement at runtime, so that portions of the statement can be determined by user input.

dynamic statements. The SQL statements that are specified at runtime (when the program is executed), rather than when the program is compiled. You can use the PREPARE statement to create dynamic SQL statements.

EBCDIC. Acronym for *Extended Binary Coded Decimal Interchange Code*, a 256-element 8-bit character set, originally designed for IBM mainframe computers.

element. A member of a *collection*, such as a LIST, MULTISSET, or SET data type. An element can be a value of any *built-in data type*, *opaque data type*, *distinct data type*, *named row type*, *unnamed row type*, or *collection data type*.

element type. The *data type* of the *elements* in a *collection*.

embedded SQL. The SQL statements that are placed within some other host language. For example, Informix supports embedded SQL in C.

enabled mode. The default object mode of database objects. When a constraint, index, or trigger is in this mode, the database server recognizes the existence of the object and takes the object into consideration while executing data manipulation statements. *See also* object mode.

end-user format. The format in which data appears within a client application as literal strings or character variables. End-user formats are useful for data types whose database format is different from the format to which users are accustomed.

end-user routine. A *user-defined routine* (UDR) that performs a task within an SQL statement that the existing *built-in* routines do not perform. Examples of tasks include encapsulating multiple SQL statements, creating trigger actions, and restricting who can access *database objects*.

environment variable. A variable that the operating system maintains for each user and makes available to all programs that the user runs.

error log. A file that receives error information whenever a program runs.

error message. A message that is associated with a (usually negative) number. IBM Informix applications display error messages on the screen or write them to files.

error trapping. *See* exception handling.

escape character. A character that indicates that the next character, normally interpreted by the program as having special significance, is to be processed as a literal character instead. The escape character precedes the special character (such as a wildcard or delimiter) to “escape” (that is, ignore) the special significance.

escape key. The physical key of a keyboard, usually marked ESC, that is used to terminate one mode and start another mode in most UNIX and DOS systems.

ESQL/C Smart Large-Object API. An API of C routines that an IBM Informix ESQL/C client application can use to access *smart large objects* as operating-system files. The ESQL/C Smart Large-Object API is part of the IBM Informix ESQL/C *SQL API*. You can also access smart large objects with a set of functions in the *DataBlade API*.

exception. An error or warning that the database server returns, or a state that a SPL statement initiates.

exception handling. The code in a program that anticipates and reacts to runtime errors and warnings. Also referred to as error handling or error trapping.

exclusive access. Sole access to a database or table by a user. Other users are prevented from using it.

exclusive lock. A lock on an object (row, page, table, or database) that is held by a single thread that prevents other processes from acquiring a lock of any kind on the same object.

executable file. A file that contains code that can be executed directly. A C-language object file can be an executable file; it contains the machine-level instructions that correspond to the C-language *source file*.

execute. To run a statement, program, *routine*, or a set of instructions. *See also* executable file.

explicit cast. A *user-defined cast* that a user explicitly invokes with the CAST AS keyword or cast operator (::). *See also* implicit cast.

explicit connection. A connection made to a database environment that uses the CONNECT statement. *See also* implicit connection.

explicit projection list. In a SELECT statement, a projection list that does not use the asterisk (*) notation, but explicitly lists the names of the

columns from which the query returns values. (The projection list is sometimes called the *select list*.)

explicit transaction. A transaction that is initiated by the BEGIN WORK statement. This type of transaction is available only in non-ANSI compliant databases that support logging. *See also* implicit transaction and singleton implicit transaction.

exponent. The power to which a value is to be raised.

express mode. An Extended Parallel Server method of loading or unloading data that uses light appends.

expression. A specification that the database server can evaluate. This can include literal values, constants, column values, functions, quoted strings, operators, and parentheses (as delimiters). A Boolean expression evaluates as TRUE, FALSE, or UNKNOWN. An arithmetic expression can contain the arithmetic operators (+, -, ¥, /, and so on) and returns a number.

expression-based fragmentation. A distribution scheme that distributes rows to fragments according to a user-specified expression that is defined in the WHERE clause of an SQL statement.

extended data type. A term used to refer to data types that are not built in; namely *complex data types*, *opaque data types*, and *distinct data types*.

extent. A continuous segment of disk space that a database server allocated to a tblspace (a table). The user can specify both the initial extent size for a table and the size of all subsequent extents that a database server allocates to the table.

external function. An *external routine* that returns a single value.

external procedure. An *external routine* that does not return a value.

external routine. A *user-defined routine* that is written in an external language that the database

supports. These external languages include the C and Java languages. The routine names, parameters, and other information are registered in the system catalog tables of a database. However, the executable code of an external routine is stored outside the database. An external routine can be an *external function* or an *external procedure*.

external space. Storage space that a user-defined access method manages rather than the database server. The IN clause of the CREATE TABLE and CREATE INDEX statements can specify the name of an external space instead of a dbspace.

external table. A database table that is not in the current database. It might or might not be in a database that the same database server manages.

extspace. (Not for Extended Parallel Server) A logical name associated with an arbitrary string that signifies the location of external data. Access its contents with a user-defined access method.

family name. A quoted string constant that specifies a family name in the optical family. *See also* optical family.

fault tolerance. *See* high availability.

fetch. The action of moving a cursor to a new row and retrieving the row values into memory.

fetch buffer. A buffer in the application process that the database server uses to send fetched row data (except TEXT and BYTE data) to the application.

field. A component of a *named row type* or *unnamed row type* that contains a name and a *data type* and can be accessed in an SQL statement by using dot notation in the form *row type name.field name*. *See also* column.

file. A collection of related information stored together on a system, such as the words in a letter or report, a computer program, or a listing of data.

file server. A network node that manages a set of disks and provides storage services to computers on the network.

filename extension. The part of a filename following the period. For example, DB–Access appends the extension *.sql* to command files.

filter. A set of conditions (sometimes called a *predicate*) for selecting rows or records. In an SQL query, the conditional expression in the WHERE clause is a filter that controls the active set of the query. The High-Performance Loader (HPL) uses a filter component to screen data before loading it into a database.

filtering mode. An object mode of a database object, causing bad rows to be written to the violations table during DML operations. During DML operations, the database server enforces requirements of a constraint or of a unique index that is in this mode and identifies any rows that would violate the requirement.

fixchar. A character data type in ESQL/C programs, for fixed-length character strings that are padded (as needed) with trailing blanks, and not NULL-terminated.

fixed-point number. A number where the decimal point is fixed at a specific place regardless of the value of the number.

flag. A command-line option, usually indicated by a minus (-) sign in UNIX systems. For example, in DB–Access the *-e* flag echoes input to the screen.

flexible temporary table. An explicit temporary table that Extended Parallel Server automatically fragments using a round-robin distribution scheme.

floating-point number. A number with fixed precision (total number of digits) and undefined scale (number of digits to the right of the decimal point). The decimal point *floats* as appropriate to represent an assigned value.

foreign key. A column or set of columns that references a unique or primary key in a table. For every entry in a foreign-key column

containing only non-NULL values, there must exist a matching entry in the unique or primary column.

format. A description of the organization of a data file. *See also* component.

formatting character. For XPS, a percent sign (%) followed by a letter (c, n, o, or r). In a command line, Extended Parallel Server expands the formatting character to designate multiple coserver numbers (%c), multiple nodes (%n), multiple ordinal numbers designating dbspaces (%d), or a range of dbspaces (%r).

fragment. *See* index fragment and table fragment.

fragment elimination. The process of applying a filter predicate to the fragmentation strategy of a table or index and removing the fragments that do not apply to the operation.

fragmentation. The process of defining groups of rows within a table based on a rule and then storing these groups, or fragments, in separate dbspaces that you specify when you create a table or index fragmentation strategy.

full checkpoint. A type of checkpoint where the pages on disk are synchronized with the pages in the shared-memory buffer pool.

function. A *routine* that returns one or more values. *See also* user-defined function.

function cursor. A cursor that is associated with an EXECUTE FUNCTION statement, which executes routines that return values. *See also* cursor function.

function overloading. *See* routine overloading.

fuzzy checkpoint. A type of checkpoint where only certain pages on disk are synchronized with the pages in the shared-memory buffer pool, and the logical log is used to synchronize the rest of the pages during fast recovery.

gateway. A device that establishes data communications between networks.

generalized-key (GK) index. A type of index for static tables with Extended Parallel Server that can speed certain queries by storing the result of an expression as a key in a B+ tree or bitmap index. Three types of GK index are *selective*, *virtual column*, and *join*.

gigabyte. A unit of storage, equal to 1024 megabytes or 1024³ bytes.

Global Language Support (GLS). A feature that enables Informix APIs and database servers to support different languages, cultural conventions, and code sets. For information about the GLS feature, see the *IBM Informix: GLS User's Guide*.

global variable. A *variable* or *identifier* whose scope of reference is all modules of a program. *Compare with* local variable.

globally detached index. For Extended Parallel Server, a type of index that has a fragmentation strategy that is independent of the table fragmentation and where the database server cannot verify that each index row resides on the same coserver as the referenced data row. You can use an expression, system-defined hash, or hybrid distribution scheme to create globally detached indexes for any table. *See also* locally detached index.

GLS. *See* Global Language Support (GLS).

GLS API. A legacy acronym for IBM Informix GLS. An API of C routines that a C-language *external routine* can use to access IBM Informix GLS *locales*. This API also includes functions that obtain culture-specific collation, and time, date, number, and currency formats and functions that provide a uniform way of accessing character data, regardless of whether the locale is left-to-right or bidirectional or supports *single-byte characters* or *multibyte characters*.

hash fragmentation. *See* system-defined hash fragmentation.

hash rule. A user-defined algorithm that maps each row in a table to a set of hash values and that is used to determine the fragment in which a row is stored.

header file. A *source file* that contains *declarations* for *variables*, *constants*, and *macros* that a particular group of modules or programs share.

help message. Online text displayed automatically or at the request of the user to assist the user in interactive programs. Such messages are stored in help files.

heterogeneous commit. A protocol governing a group of database servers, of which at least one is a *gateway* participant. It ensures the all-or-nothing basis of distributed transactions in a heterogeneous environment. *See also* two-phase commit.

hierarchy. A tree-like data structure in which some groups of data are subordinate to others such that only one group (called **root**) exists at the highest level, and each group except root is related to only one parent group on a higher level.

high availability. The ability of a system to resist failure and data loss. High availability includes features such as fast recovery and mirroring. It is sometimes referred to as *fault tolerance*.

High-Performance Loader. The High-Performance Loader (HPL) utility is part of Dynamic Server. The HPL loads and unloads data using parallel access to devices. *See also* external table.

highlight. A rectangular inverse-video area marking your place on the screen. A highlight can indicate the current option on a menu or the current character in an editing session. If a terminal cannot display highlighting, the current option can appear in angle (< >) brackets, with the current character underlined.

hold cursor. A cursor that is created using the WITH HOLD keywords. A hold cursor remains open past the end of a transaction. It allows uninterrupted access to a set of rows across multiple transactions.

home page. The page that contains the first byte of the data row, specified by the rowid. Even if a data row outgrows its original storage location,

the home page does not change. The home page contains a forward pointer to the new location of the data row. *See also* remainder page.

host variable. An SQL API program variable that you use in an embedded statement to transfer information between the SQL API program and the database.

HPL. *See* High-Performance Loader.

hybrid fragmentation. A distribution scheme that lets the user specify two fragmentation methods. Usually one method is used globally and one method is used locally.

identifier. In the default locale, a sequence of letters, digits, and underscores (`_`) that is the unqualified name of a database, storage, or program object. (Additional characters are valid in other locales or if the **DELIMITED** variable is set.)

implicit transaction. A transaction that begins implicitly after the COMMIT WORK or ROLLBACK WORK statement. This is the only type of transaction that ANSI-compliant databases support, but it is also available for other databases that support logging. *See also* explicit transaction and singleton implicit transaction.

implicit cast. A *built-in* or *user-defined* cast that the database server automatically invokes to perform data-type conversion. *See also* explicit cast.

implicit connection. A connection that is made using the DATABASE, CREATE DATABASE, START DATABASE, or DROP DATABASE statement. *See also* explicit connection.

implicit projection list. In a SELECT statement, using the asterisk (`*`) wildcard symbol so that a query returns values from all columns of the table. The projection list is sometimes called the "select list," because it immediately follows the SELECT keyword.

incremental archiving. A system of archiving that allows the option to archive only those parts of the data that have changed since the last archive was created.

independent subquery. A subquery that has no relationship to or dependency on any of its parent queries. It needs to be evaluated only once and the results can be used thereafter. In independent subqueries, both the parent and subquery are parallelized. *See also* correlated subquery and subquery.

index. A structure of entries, each of which contains a value or values and a pointer to the corresponding location in a table or smart large object. An index might improve the performance of database queries by ordering a table according to key column values or by providing access to data inside of large objects.

index fragment. Consists of zero or more index items grouped together, which can be stored in the same dbspace as the associated table fragment or in a separate dbspace. An index fragment also might occupy an sbspace or an *extspace*.

Informix user ID. A login user ID (login *user name* or *authorization identifier*) that must be valid on all computer systems (operating systems) involved in the client's database access. Often referred to as the *client's user ID*. This need not refer to a fully functional user account on the computer system; only the user identity components of the user account information are significant to Informix database servers. A given user typically has the same Informix user ID on all networked computer systems involved in the database access. The authorization identifier **informix** is required for some database objects and operations.

Informix user password. A user ID password that must be valid on all computer systems (operating systems) involved in the client's database access. When the client specifies an explicit user ID, most computer systems require the Informix user password.

inheritance. The process that allows an object to acquire the properties of another object. Inheritance allows for incremental modification, so that an object can inherit a general set of properties and add properties that are specific to itself. *See also* type inheritance and table inheritance.

initialize. To prepare for execution. To initialize a *variable*, you assign it a starting value. To initialize the database server, you start its operation.

inmigration. The process by which Optical Subsystem migrates TEXT and BYTE data from the optical storage subsystem into the Dynamic Server environment.

inner join. A query that symmetrically combines rows from two or more tables. *See also* simple inner join.

input. The information that is received from an external source (for example, from the keyboard, a file, or another program) and processed by a program.

input parameter. In a prepared SQL statement, a value, represented by a "?" placeholder symbol, that must be provided when the prepared statement is executed.

insert cursor. A cursor for insert operations, associated with an INSERT statement. Allows bulk insert data to be buffered in memory and written to disk.

installation. The loading of software from some medium (tape, cartridge, removable disk, or CD) onto a computer, and preparing the software for use.

interactive. An aspect of a program that accepts input from the user, processes the input, and then produces output on the screen, in a file, or on a printer.

internationalization. The process of making IBM Informix products easily adaptable to the conventions of various cultures and natural languages. Among other features, internationalized software can support

culturally-specific collation and date, time, and money formats. *See also IBM Informix Guide to GLS Functionality.*

interquery parallelization. The ability to process multiple queries simultaneously to avoid a performance delay when multiple independent queries access the same table. *See also* intraquery parallelization.

interrupt. A signal from a user or another process that can stop the current process temporarily or permanently. *See also* signal.

interrupt key. A key used to cancel or abort a program or to leave a current menu and return to the menu one level above. On many systems, the interrupt key is CONTROL-C; on some others, the interrupt key is DEL or CONTROL-Break.

intraquery parallelization. Breaking of a single query into subqueries by a database server using PDQ and then processing the subqueries in parallel. Parallel processing of this type has important implications when each subquery retrieves data from a fragment of a table. Because each partial query operates on a smaller amount of data, the retrieval time is significantly reduced and performance is improved. *See also* interquery parallelization.

IPX/SPX. Acronym for Internetwork Packet Exchange/Sequenced Packet Exchange. It refers to the NetWare network protocol by Novell.

ISAM. Acronym for *Indexed Sequential Access Method*. This allows you to find information in a specific order or to find specific items of information quickly through an index. *See also* access method.

ISAM error. Operating system or file access error.

ISO. Acronym for the *International Standards Organization*. ISO sets worldwide standards for the computer industry, including standards for character input and manipulation, code sets, and SQL syntax.

ISO8859-1. A code set that contains 256 single-byte characters. Characters 0 through 127 are the ASCII characters. Characters 128 through 255 are mostly non-English characters from European languages; for example, é, ñ and ö.

isolation. When multiple users attempt to access common data, the level of independence specifically relating to the locking strategy for read-only SQL requests. The various levels of isolation are distinguished primarily by the length of time that shared locks are (or can be) acquired and held. You can set the isolation level with the SET ISOLATION or SET TRANSACTION statement.

iterator function. A cursor function that returns a data set by successive calls. It can be written in C or Java or can be an SPL routine that includes RETURN WITH RESUME.

jagged rows. A query result in which rows differ in the number and type of columns they contain because the query applies to more than one table in a table hierarchy.

join. The process of combining information from two or more tables based on some common domain of information. Rows from one table are paired with rows from another table when information in the corresponding rows match on the joining criterion. For example, if a **customer_num** column exists in the **customer** and the **orders** tables, you can construct a query that pairs each **customer** row with all the associated **orders** rows based on the common **customer_num**. *See also* Cartesian product and outer join.

join index. A type of generalized-key (GK) index that contains keys that are the result of a query that joins multiple tables.

jukebox. A physical storage device that consists of one or more optical-disc drives, slots for platters, optical platters, and a robotic arm to transfer platters between the slots and the drives. A jukebox is also known as an *autochanger*.

kernel. Operating system component that controls processes and resource allocation.

key. The items of information that are used to locate a row of data. A key defines the pieces of information for which you want to search as well as the order in which you want to process information in a table. For example, you can index the **last_name** column in a **customer** table to find specific customers or to process the customers in alphabetical order (or in reverse alphabetical order) by their last names (when **last_name** serves as the key).

keyword. A word that has a predefined meaning in a programming language. For example, the word SELECT is a keyword in SQL.

kilobyte. A unit of storage that consists of 1024 bytes.

Language Supplement. An IBM Informix product that provides the locale files and error messages to support one or more languages. The International Language Supplement supports several European languages. IBM Informix provides separate Language Supplements for several Asian languages.

large object. A data object that is logically stored in a column of a table, but physically stored independently of the column, due to its size. Large objects can be *simple large objects* (TEXT, BYTE) or *smart large objects* (BLOB, CLOB).

leaf node. Index page containing index items and horizontal pointers to other leaf nodes. A database server creates leaf nodes when the root node becomes full.

level of isolation . *See* isolation.

library. A group of precompiled *routines* designed to perform tasks that are common to a given kind of application. An application programming interface (API) can include a library of routines that you can call from application programs. *See also* dynamic link library (DLL), shared library, and shared-object file.

light append. An unbuffered, unlogged insert operation.

link. A way of combining separately compiled program modules, usually into an executable program. *Compare with* compile and execute.

LIST. A *collection data type* created with the LIST constructor in which *elements* are ordered and duplicates are allowed.

literal. The representation of a data type value in a format that the database server accepts in data-entry operations. For example, 234 is a literal integer and "abcd" is a literal character.

little-endian. A hardware-determined storage method in which the least-significant byte of a multibyte number has the lowest address. *See also* big-endian.

load job. The information required to load data into a relational database using the HPL. This information includes the format, map, filter, device array, project, and special options.

local copy. For Extended Parallel Server, a replica of a table on a local coserver that is copied to multiple coservers. This allows faster access to the data for OLTP transactions connected to those coservers because you do not have to send the data across the communication links between coservers.

local loopback. A connection between the client application and database server that uses a network connection even though the client application and the database server are on the same computer.

local variable. A *variable* or *identifier* whose scope of reference is only within the *routine* in which it is defined. *Compare with* global variable.

locale. A set of Informix files that specify the linguistic conventions for a country, region, culture, or language. IBM Informix products provide predefined locales that customers cannot modify. A locale provides the name of the code set that the application data uses, the collation order to use for character data, and the end-user format. *See also* client locale, database locale, default locale, server locale, and server-processing locale.

localized order. A collation order other than code-set order, if defined for a locale. Only NCHAR and NVARCHAR data values are collated in a localized order. Database objects collate in their creation-time order, if this is not the runtime order.

locally-detached index. For Extended Parallel Server, a type of index that has a fragmentation strategy that is independent of the table fragmentation but where the database server recognizes that each index row resides on the same co-server as the referenced data row. You can use an expression, system-defined hash, or hybrid distribution scheme to create locally detached indexes for any table. *See also* globally-detached index.

lock coupling. A locking feature that holds a lock on the child node until a lock is obtained on the parent node during upward movement when updating an R-tree index. Lock coupling is used if the bounding box of a leaf node has changed. You must propagate the change to the parent node by moving upwards in the tree until you reach a parent node that needs no change.

lock mode. An option to specify whether a user who requests a lock on an already locked object waits until the object is released to receive the lock, or immediately receives an error, or waits for some span of time before receiving an error.

locking. A concurrency feature temporarily limiting access to an object (database, table, page, or row) to prevent conflicting interactions among concurrent processes. An exclusive lock restricts read and write access to only one user; a shared lock allows read-only access to other users. An update lock begins as a shared lock, but is upgraded to an exclusive lock after a row is changed.

locking granularity. The size of a locked object. The size can be a database, table, page, or row.

logical log. An allocation of disk space that the database server manages that contains records of all changes that were performed on a database during the period when the log was active. It is used to roll back transactions, recover from

system failures, and restore databases from archives. *See also* physical log.

login. The process of identifying oneself to a computer.

login password. *See* Informix user password.

login user ID. *See* Informix user ID.

logslice. A dbslice whose contents are comprised solely of logical-log files. The logical-log files in the logslice can be owned by multiple coservers, one log file per dbspace. *See also* dbslice, rootslice, and physslice.

LVARCHAR. A built-in data type that stores varying-length character data of up to 32 kilobytes.

macro. A named set of instructions that the computer substitutes when it encounters the name in source code.

mantissa. The significant digits in a floating-point number.

map. A description of the relation between the records of a data file and the columns of a relational database. *See also* component.

massively parallel processing system. A system composed of multiple computers that are connected to a single high-speed communication subsystem. MPP computers can be partitioned into nodes. *Compare with* symmetric multiprocessing system.

megabyte. A unit of storage that equals 1024 kilobytes or 1024² bytes.

Memory Grant Manager (MGM). (Not for Extended Parallel Server) A database server component that coordinates the use of memory and I/O bandwidth for decision-support queries. MGM uses the DS_MAX_QUERIES, DS_TOTAL_MEMORY, DS_MAX_SCANS, and PDQPRIORITY configuration parameters to determine what resources can or cannot be granted to a decision-support query.

menu. A screen display that allows you to choose the commands that you want the computer to perform.

MGM. Acronym for Memory Grant Manager.

mirroring. Storing the same data on two chunks simultaneously. If one chunk fails, the data values are still usable on the other chunk in the mirrored pair. The database server administrator handles this data storage option.

MODE ANSI. The keywords specified on the CREATE DATABASE or START DATABASE statement to make a database ANSI compliant.

monochrome. A term that describes a monitor that can display only one color.

MPP. Acronym for *massively parallel processing system*.

multibyte character. A character that might require from two to four bytes of storage. If a language contains more than 256 characters, the code set must contain multibyte characters. With a multibyte code set, an application cannot assume that one character requires only one byte of storage. *See also* single-byte character.

multiplexed connection. A single network connection between a database server and a client application that handled multiple database connections from the client.

MULTISET. A *collection data type* created with the MULTISET constructor in which *elements* are not ordered and duplicates are allowed.

multithreading. Running of multiple threads within the same process. *See* thread.

named row type. A *row type* created with the CREATE ROW TYPE statement that has a declared name and *inheritance* properties and can be used to construct a *typed table*.

national character. In National Language Support (NLS), a character from the written form of a natural language that can be stored in an NCHAR or NVARCHAR column. Sometimes called a *native character*.

native character. *See* national character.

National Language Support (NLS). A feature that supports single-byte code sets, using NCHAR and NVARCHAR columns to store non-English character strings. This technology has been superseded on IBM Informix databases by Global Language Support (GLS).

NLS. *See* National Language Support (NLS).

node. (1) In the context of an index for a database, a node is an ordered group of key values having a fixed number of elements. (A key is a value from a data record.) A B-tree for example, is a set of nodes that contain keys and pointers arranged in a hierarchy. *See also* branch node, leaf node, and root node. (2) In the context of a MPP system, a node is an individual computer. *See also* massively parallel processing system. (3) In the context of a SMP system, a node can either be the entire SMP computer or a fully functioning subsystem that uses a portion of the hardware resources of that SMP system. *See also* symmetric multiprocessing system. (4) For Extended Parallel Server, a node is an individual computer with one or more CPUs that runs a single instance of an operating system within a parallel-processing platform. A node can be a uniprocessor, a cluster of stand-alone computers, an SMP computer, or an independent subsystem configured within an SMP computer.

non-ASCII character. A character that is not match any of the 128 code points in the ASCII character set. Non-ASCII characters include 8-bit characters and multibyte characters.

noncursor function. A *user-defined function* that returns a single group of values (one row of data) and therefore does not require a cursor when it is executed. *Compare with* cursor function.

nonvariant function. A user-defined function that always returns the same value when passed the same arguments. A nonvariant function must not contain SQL statements. *Compare with* variant function.

NOT NULL constraint. A constraint on a column that specifies that the column cannot contain NULL values.

NULL. A keyword of SQL, indicating that a value is unknown, missing, or not applicable. (A NULL value is not the same as a value of zero or blank.)

object. See database object.

object mode. The state of a database object as recorded in the **sysobjstate** system catalog table. A constraint or unique index can be in enabled, disabled, or filtering mode. A trigger or duplicate index can be in enabled or disabled mode. You use SET statements to change the mode of an object.

object-relational database. A database that adds object-oriented features to a *relational database*, including support for *user-defined data types*, *user-defined routines*, *user-defined casts*, *user-defined access methods*, and *inheritance*.

OLTP. Acronym for Online Transaction Processing. See also online transaction processing application.

online transaction processing application. Characterized by quick, indexed access to a small number of data items. The applications are typically multiuser, and response times are measured in fractions of seconds. See also decision-support application.

online transaction processing queries. The transactions that OLTP applications handle are usually simple and predefined. A typical OLTP system is an order-entry system where only a limited number of rows are accessed by a single transaction many times. See also decision-support query.

opaque data type. A *data type* whose inner structure is not visible to the database server. Opaque types that are not built-in need *user-defined routines* and *user-defined operators* that work on them. Synonym for *base type* and *user-defined base type*.

opaque-type support function. One of a group of *user-defined functions* that the database server uses to perform operations on *opaque data types* (such as converting between the internal and external representations of the type).

open. The process of making a resource available, such as preparing a file for access, activating a column, or initiating a window.

operational table. A logging permanent table that uses light appends for fast update operations. Operational tables do not perform record-by-record logging.

operator. In an SQL statement, a keyword (such as UNITS or UNION) or a symbol (such as =, >, <, +, -, or *) that invokes an *operator function*. The operands to the operator are arguments to the *operator function*.

operator binding. The implicit invocation of an *operator function* when an *operator* is used in an SQL statement.

operator class. An association of *operator-class functions* with a *secondary access method*. The database server uses an operator class to optimize queries and build an index of that secondary access method.

operator-class function. One of the operator-class support functions or *operator-class strategy functions* that constitute an *operator class*. For user-defined operator classes, the operator-class functions are *user-defined functions*.

operator-class strategy function. An *operator-class function* that can appear as a *filter* in a query. The query optimizer uses the strategy functions to determine if an *index* of a particular *secondary access method* can be used to process the filter. You register operator-class strategy functions in the STRATEGIES clause of the CREATE OPCLASS statement.

operator-class support function. An *operator-class function* that a *secondary access method* uses to build or search an *index*. You register operator-class support functions in the SUPPORT clause of the CREATE OPCLASS statement.

operator function. A *function* that processes one or more arguments (its operands) and returns a value. Many operator functions have corresponding operators, such as **plus()** and **+**. You can overload an operator function so that it handles a *user-defined data type*. *See also* routine overloading.

optical cluster. An amount of space on an optical disc that is reserved for storing a group of logically related simple large objects or smart large objects.

optical family. A group of optical discs, theoretically unlimited in number.

optical platter. A removable optical disc that stores data in an optical storage subsystem.

optical statements. The SQL statements that you use to control optical clustering.

optical volume. One side of a removable Write-Once-Read-Many (WORM) optical disc.

outer join. An asymmetric joining of a dominant (*outer*) table and one or more “subservient” tables in a query, where the values for the outer table are preserved even if no matching rows exist in the subservient table. In the result set, any outer-table row that has no matching row in the subservient table has NULL values in the columns selected from the subservient table.

outmigration. The process by which Optical Subsystem migrates TEXT or BYTE data from the Dynamic Server environment to an optical storage subsystem.

output. The result that the computer produces in response to a query or a request for a report.

overloading. *See* routine overloading.

owner-privileged. A class of SPL routines that any user can create who has Resource database privileges.

packed decimal. A storage format that represents either two decimal digits or a sign and one decimal digit in each byte.

pad. Usually, to fill empty places at the beginning or end of a line, string, or field with spaces or blanks when the input is shorter than the field.

page. The physical unit of disk storage and basic unit of memory storage that the database server uses to read from and write to Informix databases. Page size is fixed for a particular operating system and platform. A page is always entirely contained within a chunk. *See also* home page and remainder page.

parallel database query. The execution of SQL queries in parallel rather than sequential order. The tasks a query requires are distributed across several processors. This type of distribution enhances database performance.

parallel-processing platform. A parallel-processing platform is a set of independent computers that operate in parallel and communicate over a high-speed network, bus, or interconnect. *See also* symmetric multiprocessing system and massively parallel processing system.

parallelism. Ability of an Informix database server to process a task in parallel by breaking the task into subtasks and processing the subtasks simultaneously, thus improving performance.

parameter. A variable that is given a value for a specified application. In the signature of a user-defined routine, a parameter serves as a placeholder for an argument. The parameter specifies the *data type* of the value that the user-defined routine expects when it receives the associated argument at runtime. *See also* configuration file, input parameter, and routine signature.

parent-child relationship. *See* referential constraint.

parent table. The referenced table in a referential constraint. *See also* child table.

participating coserver. A coserver that controls one or more fragments of a table that Extended Parallel Server accesses. *See also* coserver and connection coserver.

partition. *See* table fragment.

pattern. An identifiable or repeatable series of characters or symbols.

PDQ. Acronym for *parallel database query*.

PDQ priority. Determines the amount of resources that a database server allocates to process a query in parallel. These resources include memory, threads (such as scan threads), and sort space. The level of parallelism is established by using the **PDQPRIORITY** environment variable or various database server configuration parameters (including **PDQPRIORITY** and **MAX_PDQPRIORITY**) or dynamically through the **SET PDQPRIORITY** statement.

permission. On some operating systems, the right to access files and directories.

phantom row. A row of a table that is initially modified or inserted during a transaction but is subsequently rolled back. Another user can see a phantom row if the isolation level is Informix Dirty Read or ANSI compliant Read Uncommitted. No other isolation level allows a user to see a changed but uncommitted row.

physical log. A set of contiguous disk pages in shared memory where the database server stores an unmodified copy (before-image) of pages before the changed pages are recorded. The pages in the physical log can be any database server page except a blob space blobpage.

physslice. A dbslice that contains the physical log. *See also* dbslice, logslice, and rootslice.

pointer. A value that specifies the address in memory of the data or *variable*, rather than the contents of the data or variable.

polymorphism. *See* routine overloading and type substitutability.

precision. The total number of significant digits in a real number, both to the right and left of the decimal point. For example, the number 1437.2305 has a precision of 8. *See also* scale.

predefined opaque data type. An opaque data type for which the database server provides the data type definition. *See also* BLOB, BOOLEAN, CLOB, LVARCHAR, and pointer.

predicate. *See* filter.

predicate lock. A lock held on index keys that qualifies for a predicate. In a predicate lock, exclusive predicates consist of a single key value, and shared predicates consist of a query rectangle and a scan operation such as inclusion or overlap.

prepared statement. An SQL statement that is generated by the **PREPARE** statement from a character string or from a variable that contains a character string. This feature allows you to form your request while the program is executing without having to modify and recompile the program.

preprocessor. A program that takes high-level programs and produces code that a standard language compiler such as C can compile.

primary access method. An access method whose *routines* access a *table* with such operations as inserting, deleting, updating, and scanning. *See also* secondary access method.

primary key. The information from a column or set of columns that uniquely identifies each row in a table. The primary key sometimes is called a *unique key*.

primary-key constraint. Specifies that each entry in a column or set of columns contains a unique non-NULL value.

printable character. A character that can be displayed on a terminal, screen, or printer. Printable characters include A-Z, a-z, 0-9, and punctuation marks. *Compare with* control character.

privilege. The right to use or change the contents of a database, table, table fragment, or column. *See also* access privileges.

procedure. A *routine* that does not return values. *See also* user-defined procedure.

procedure overloading . *See* routine overloading.

process. A discrete task, generally a program, that the operating system executes.

project. A group of related components that the High-Performance Loader (HPL) uses. *See also* component.

projection. Taking a subset of the columns in a table. Projection is implemented through the Projection clause of the SELECT statement and returns some of the columns and all the qualifying rows of a table. The set of columns in the Projection clause is sometimes called the "select list." *See also* selection and join.

promotable lock . A lock that can be changed from a shared lock to an exclusive lock. *See also* update lock.

protocol. A set of conventions that governs communication among computers. These conventions govern format, timing, sequencing, and error control.

query. A request to the database to retrieve data that meet certain criteria, usually made with the SELECT statement.

query optimization information statements. The SQL statements that are used to optimize the performance of queries. These statements include SET EXPLAIN, SET OPTIMIZATION, and UPDATE STATISTICS.

query unnesting. An execution strategy for nested SQL subqueries whereby Extended Parallel Server rewrites such subqueries to use modified joins rather than iteration mechanisms. The **sqexplain.out** file reflects the query plan that has been selected after subquery unnesting has occurred.

R-tree index. (Not for Extended Parallel Server) A type of index that uses a tree structure based on overlapping bounding rectangles to speed access to spatial and multidimensional data types. *See also* bitmap index and B-tree index.

range fragmentation. A distribution scheme that distributes data in table fragments that contain a specified key range. This technique can eliminate scans of table fragments that do not contain the required rows, making queries faster.

range rule. A user-defined algorithm for *expression-based fragmentation*. It defines the boundaries of each fragment in a table using SQL relational and logical operators. Expressions in a range rule can use the following restricted set of operators: >, <, >=, <=, and the logical operator AND.

raw device. *See* unbuffered disk I/O.

raw disk. *See* unbuffered disk I/O.

raw table. A nonlogged permanent table that uses *light appends*.

Read Committed. A level of isolation that the SET TRANSACTION statement can specify, in which a user can view rows that are currently committed at the moment of the query request, but cannot view rows that were changed as part of a currently uncommitted transaction. This is the default isolation level for databases that are not ANSI compliant. *See also* Committed Read.

Read Uncommitted. An ANSI-compliant level of isolation, set with the SET TRANSACTION statement, that does not account for locks. This allows a user to view any existing rows, even rows that can be altered within currently uncommitted transactions. Read Uncommitted is the lowest level of isolation (no isolation at all), and is thus the most efficient. *See also* Dirty Read.

real user ID. *See* Informix user ID.

record. *See* row.

Record-ID. A four-byte RSAM entity, also known as RID, that describes the logical position of the record within a fragment. Not the same as rowid.

recover a database. To restore a database to a former condition after a system failure or other destructive event. The recovery restores the database as it existed immediately before the failure.

referential constraint. The relationship between columns within a table or between tables; also known as a *parent-child relationship*. Referencing columns are also known as *foreign keys*.

registering. In a database, the process of storing information about a *database object* in the *system catalog tables* of a database. Most SQL data definition statements perform some type of registration. For example, the CREATE FUNCTION and CREATE PROCEDURE statements register a *user-defined routine* in a database.

relation. *See* table.

relational database. A database that uses table structures to store data. Data in a relational database is divided across tables in such a way that additions and modifications to the data can be made easily without loss of information.

relational database server. A database server that manages data values that are stored in rows and columns.

remainder page. A page that accommodates subsequent bytes of a long data row. If the trailing portion of a data row is less than a full page, it is stored on a remainder page. After the database server creates a remainder page for a long row, it can use the remaining space in the page to store other rows. Each full page that follows the home page is referred to as a big-remainder page.

remote connection. A connection that requires a network.

remote routine. A routine in a databases of a remote server. *See* subordinate server.

remote server. *See* subordinate server.

remote table. In a distributed query, a table in a database of a server that is not the local database server. *See also* coordinating server, subordinate server.

Repeatable Read. An Informix and ANSI level of isolation available with the Informix SET ISOLATION statement or the ANSI compliant SET TRANSACTION statement, which ensures that all data values read during a transaction are not modified during the entire transaction. Transactions under ANSI Repeatable Read are also known as Serializable. Informix Repeatable Read is the default level of isolation for ANSI compliant databases. *See also* isolation and Serializable.

reserved pages. The first 12 pages of the initial chunk of the root dbspace. Each reserved page stores specific control and tracking information that the database server uses.

reserved word. A word in a statement or command that you cannot use in any other context of the language or program without receiving a warning or error message.

restore a database. *See* recover a database.

role. A classification or work task, such as **payroll**, that the DBA assigns. Assignment of roles makes management of privileges convenient.

role separation. (Not for Extended Parallel Server) A database server installation option that allows different users to perform different administrative tasks.

roll back. The process that reverses an action or series of actions on a database. The database is returned to the condition that existed before the actions were executed. *See also* transaction and commit work.

root dbspace. The initial *dbspace* that the database server creates. It contains reserved pages and internal tables that describe and track all other dbspaces, blobspaces, sbspaces, tblspaces, chunks, and databases.

root node. A single index page that contains node pointers to *branch nodes*. The database server allocates the root node when you create an index for an empty table.

root supertype. The *named row type* at the top of a *type hierarchy*. A root supertype has no *supertype* above it.

rootslice. A dbslice that contains the root dbspaces for all coservers for Extended Parallel Server. *See also* dbslice, logslice, and physslice.

round-robin fragmentation. A distribution scheme in which the database server distributes rows sequentially and evenly across specified dbspaces.

routine. A group of program statements that perform a particular task. A routine can be a *function* or a *procedure*. All routines can accept arguments. *See also* built-in and user-defined routine.

routine modifier. A keyword in the WITH clause of a CREATE FUNCTION, CREATE PROCEDURE, ALTER FUNCTION, ALTER PROCEDURE, or ALTER ROUTINE statement that specifies a particular attribute or usage of a *user-defined routine*.

routine overloading. The ability to assign one name to multiple *user-defined routines* and specify *parameters* of different data types on which each routine can operate. An overloaded routine is uniquely defined by its *routine signature*.

routine resolution. The process that the database server uses to determine which *user-defined routine* to execute, based on the *routine signature*. *See also* routine overloading.

routine signature. The information that the database server uses to uniquely identify a *user-defined routine*. The signature includes the type of routine (function or procedure); the routine name; and the number, order, and data types of the parameters. *See also* routine overloading and specific name.

row. A group of related items of information about a single entity across all columns in a

database table. In a table of customer information, for example, a row contains information about a single customer. A row is sometimes referred to as a *record* or *tuple*. In an object-relational model, each row of a table stands for one *instance* of the subject of the table, which is one particular example of that entity. In a screen form, a row can refer to a line of the screen.

row type. A *complex data type* that contains one or more related data *fields*, of any *data type*, that form a template for a record. The data in a row type can be stored in a row or column. *See also* named row type and unnamed row type.

row variable. An IBM Informix ESQL/C *host variable* or *SPL variable* that holds an entire *row type* and provides access to the individual *fields* of the row.

rowid. In nonfragmented tables, rowid refers to an integer that defines the physical location of a row. Rowids must be explicitly created to be used in fragmented tables, and they do not define a physical location for a row. Rowids in fragmented tables are accessed by an index that is created when the rowid is created; this access method is slow. It is recommended that users creating new applications move toward using primary keys as a method of row identification instead of using rowids.

rule. How a database server or a user determines into which fragment rows are placed. The database server determines the rule for *round-robin fragmentation* and *system-defined hash fragmentation*. The user determines the rule for *expression-based fragmentation* and *hybrid fragmentation*. *See also* arbitrary rule and range rule.

runtime environment. The hardware and operating-system services available at the time a program runs.

runtime error. An error that occurs during program execution. *Compare with* compile-time error.

subspace. (Not for Extended Parallel Server) A logical storage area that contains one or more chunks that store only BLOB and CLOB data.

scale. The number of digits to the right of the decimal place in DECIMAL notation. The number 14.2350 has a scale of 4 (four digits to the right of the decimal point). *See also* precision.

scale up. The ability to compensate for an increase in query size by adding a corresponding amount of computer resources so that processing time does not also increase.

scan thread. A database server thread that is assigned the task of reading rows from a table. When a query is executed in parallel, the database server allocates multiple scan threads to perform the query in parallel.

schema. The structure of a database or a table. The schema for a table lists the names of the columns, their data types, and (where applicable) the lengths, indexing, and other information about the structure of the table.

scope of reference. The portion of a *routine* or application program where an *identifier* can be accessed. Three possible scopes exist: local (applies only in a single statement block), modular (applies throughout a single module), and global (applies throughout the entire program). *See also* local variable and global variable.

scroll cursor. A cursor created with the SCROLL keyword that allows you to fetch rows of the active set in any sequence.

secondary access method. An access method whose *routines* access an *index* with such operations as inserting, deleting, updating, and scanning. *See also* operator class and primary access method.

secure auditing. A facility of Informix database servers that lets a database system security officer (DSSO) monitor unusual or potentially harmful user activity. Use the **onaudit** utility to enable auditing of events and to create audit masks. Use the **onshowaudit** utility to extract

audit event data for analysis. (For more details of secure auditing, see the *IBM Informix: Trusted Facility Guide*.)

select. *See* query.

select cursor. A cursor that is associated with a SELECT statement, which lets you scan multiple rows of data, moving data row by row into a set of receiving variables.

selection. A horizontal subset of the rows of a single table that satisfies a specified condition. Selection is implemented through the WHERE clause of a SELECT statement and returns some of the rows and all of the columns in a table. *See also* projection and join.

selective index. A type of *generalized-key index* that contains keys for only a subset of a table.

selectivity. The proportion of rows within the table that a query filter can pass.

self-join. A join between a table and itself. A self-join occurs when a table is used two or more times in a SELECT statement (with different aliases) and joined to itself.

semaphore. An operating-system communication device that signals a process to awaken.

sequence. A database object (sometimes called a *sequence generator*) that can generate unique integer values in the INT8 range.

sequential cursor. A cursor that can fetch only the next row in sequence. A sequential cursor can read through a table only once each time the sequential cursor is opened.

Serializable. An ANSI-compliant level of isolation set with the SET TRANSACTION statement, ensuring all data read during a transaction is not modified during the entire transaction. *See also* isolation and Repeatable Read.

server locale. The locale that a database server uses when it performs its own read and write

operations. The **SERVER_LOCALE** environment variable can specify a nondefault locale. *See also* client locale and locale.

server name. The unique name of a database server, assigned by the database server administrator, that an application uses to select a database server.

server number. A unique number between 0 and 255, inclusive, that a database server administrator assigns when a database server is initialized.

server-processing locale. The locale that a database server determines dynamically for a given connection between a client application and a database. *See also* locale.

session. The structure that is created for an application using the database server.

SET. A *collection data type* created with the SET type constructor, in which *elements* are not ordered and duplicate values can be inserted.

shared library. A *shared-object file* on a UNIX system. *See also* dynamic link library (DLL).

shared lock. A lock that more than one thread can acquire on the same object. Shared locks allow for greater concurrency with multiple users; if two users have shared locks on a row, a third user cannot change the contents of that row until both users (not just the first) release the lock. Shared-locking strategies are used in all levels of isolation except Informix Dirty Read and ANSI-compliant Read Uncommitted.

shared memory. A portion of main memory that is accessible to multiple processes. Shared memory allows multiple processes to communicate and access a common data space in memory. Common data does not have to be reread from disk for each process, reducing disk I/O and improving performance. Also used as an Inter-Process Communication (IPC) mechanism to communicate between two processes running on the same computer.

shared-object file. A *library* that is not linked to an application at compile time but instead is

loaded into memory by the operating system as needed. Several applications can share access to the loaded shared-object file. *See also* dynamic link library (DLL) *and* shared library.

shelf. The location of an optical platter that is neither on an optical drive nor in a jukebox slot.

shuffling. Shuffling refers to the process that occurs when a database server moves rows or key values from one fragment to another. Shuffling occurs in a variety of circumstances including when you attach, detach, or drop a fragment.

signal. A means of asynchronous communication between two processes. For example, signals are sent when a user or a program attempts to interrupt or suspend the execution of a process.

signature. *See* routine signature.

simple inner join. A join that combines information from two or more tables based on the relationship between one column in each table. Rows that do not satisfy the join criteria are discarded from the result. *See also* composite join.

simple large object. A *large object* that is stored in a *blob space* or *dbspace* is not recoverable and does not obey transaction isolation modes. Simple large objects include TEXT and BYTE data types. Extended Parallel Server does not support simple large objects that are stored in a *blob space*.

simple predicate. A search condition in the WHERE clause that has one of the following forms: **f(column, constant)**, **f(constant, column)**, or **f(column)**, where **f** is a binary or unary function that returns a Boolean value (TRUE, FALSE, or UNKNOWN).

single-byte character. A character that uses one byte of storage. Because a single byte can store values in the range of 0 to 255, it can uniquely identify 256 characters. With these code sets, an application can assume that one character is always stored in one byte. *See also* 8-bit character and multibyte character.

singleton implicit transaction. A single-statement transaction that does not require either a BEGIN WORK or a COMMIT WORK statement. This type of transaction can occur only in a database that is not ANSI compliant, but that supports transaction logging. *See also* explicit transaction and implicit transaction.

singleton select . A SELECT statement that returns a single row.

smart large object. A *large object* that is stored in an *sbspace*, which has read, write, and seek properties similar to a UNIX file, is recoverable, obeys transaction isolation modes, and can be retrieved in segments by an application. Smart large objects include BLOB and CLOB data types.

SMI. Acronym for *system-monitoring interface*.

SMP. *See* symmetric multiprocessing system.

source file. A text file that contains instructions in a high-level language, such as C. A C source file is *compiled* into an *executable file* called an object file. An SPL source file is compiled into its own executable format. *See also* compile.

source type . The data type from which a DISTINCT type is derived.

specific name. A name that you can assign to an overloaded *user-defined routine* to uniquely identify a particular signature of the user-defined routine. *See also* routine overloading and routine signature.

speed up. The ability to add computing hardware to achieve correspondingly faster performance for a DSS query or OLTP operation of a given volume.

SPL. *See* Stored Procedure Language (SPL).

SPL function. An *SPL routine* that returns one or more values.

SPL procedure. An *SPL routine* that does not return a value.

SPL routine. A *user-defined routine* that is written in Stored Procedure Language (SPL). Its

name, parameters, executable format, and other information are stored in the system catalog tables of a database. An SPL routine can be an *SPL procedure* or an *SPL function*.

SPL variable. A *variable* that is declared with the DEFINE statement in an *SPL routine*.

SQL. Acronym for *Structured Query Language*. SQL is a database query language that was developed by IBM and standardized by ANSI. Informix relational database management products are based on an extended implementation of ANSI-standard SQL.

SQL API. An *application programming interface* that allows you to embed SQL statements directly in an application. The embedded-language product IBM Informix ESQL/C is an example of an SQL API. *See also* host variable.

SQLCA. Acronym for *SQL Communications Area*. The SQLCA is a data structure that stores information about the most recently executed SQL statement. The result code returned by the database server to the SQLCA is used for error handling by Informix SQL APIs.

sqlda. Acronym for *SQL descriptor area*. A dynamic SQL management structure that can be used with the DESCRIBE statement to store information about database columns or host variables used in dynamic SQL statements. The **sqlda** structure is an Informix-specific structure for handling dynamic columns. It is available only within an IBM Informix ESQL/C program. *See also* descriptor and system-descriptor area.

sqlhosts. A file that identifies the types of connections the database server supports.

stack operator. Operators that allow programs to manipulate values that are on the stack.

staging-area blobspace. (Not for Extended Parallel Server) The blobspace where a database server temporarily stores TEXT or BYTE data that is being outmigrated to an optical storage subsystem.

statement. A line or set of lines of program code that describes a single action (for example, a SELECT statement or an UPDATE statement).

statement block. A unit of SPL program code that performs a particular task and is usually marked by the keywords begin and end. The statement block of an *SPL routine* is the smallest scope of reference for program variables.

statement identifier. An embedded variable name or SQL statement identifier that represents a data structure defined in a PREPARE statement. It is used for dynamic SQL statement management by Informix SQL APIs.

static table. A nonlogging, read-only permanent table.

status variable. A program variable that indicates the status of some aspect of program execution. Status variables often store error numbers or act as flags to indicate that an error has occurred.

storage space. A *dbspace*, *blobspace*, or *sbspace* that is used to hold data.

stored procedure. A legacy term for an *SPL routine*.

Stored Procedure Language (SPL). An Informix extension to SQL that provides flow-control features such as sequencing, branching, and looping. *See also* SPL routine.

strategy function. *See* operator-class strategy function.

string. A sequence of characters (typically alphanumeric) that is manipulated as a single unit. A string might consist of a word (such as 'Smith'), a set of digits representing a number (such as '19543'), or any other collection of characters. Strings generally are delimited by single quotes. *String* is also a character data type, available in IBM Informix ESQL/C programs, in which the character string is stripped of trailing blanks and is NULL terminated.

subordinate server. Any database server in a distributed query that did not initiate the query. Sometimes called remote server. *See also* coordinating server.

subordinate table. *See* outer join.

subquery. A query that is embedded as part of another SQL statement. For example, an INSERT statement can contain a subquery in which a SELECT statement supplies the inserted values in place of a VALUES clause; an UPDATE statement can contain a subquery in which a SELECT statement supplies the updating values; or a SELECT statement can contain a subquery in which a second SELECT statement supplies the qualifying conditions of a WHERE clause for the first SELECT statement. (Parentheses always delimit a subquery, unless you are referring to a CREATE VIEW statement or unions.) Subqueries are always parallelized. *See also* correlated subquery and independent subquery.

subscript. A subscript is an integer-valued offset into an array. Subscripts can be used to indicate the start or end position in a character data-type variable.

substring. A portion of a character string.

subtable. A *typed table* that inherits properties (column definitions, constraints, triggers) from a *supertable* above it in the *table hierarchy* and can add additional properties.

subtype. A *named row type* that inherits all representation (data *fields*) and behavior (*routines*) from a *supertype* above it in the *type hierarchy* and can add additional fields and routines. The number of fields in a subtype is always greater than or equal to the number of fields in its supertype.

supertable. A *typed table* whose properties (constraints, storage options, triggers) are inherited by a *subtable* beneath it in the *table hierarchy*. The scope of a query on a supertable is the supertable and its subtables.

supertype. A named row type whose representation (data *fields*) and behavior (*routines*) is inherited by a *subtype* below it in the *type hierarchy*.

support function. See aggregate support function, opaque-type support function, and operator-class support function.

support routine. See support function.

symmetric multiprocessing system. A system composed of multiple computers that are connected to a single high-speed communication subsystem. An SMP has fewer computers than an MPP system and cannot be partitioned into nodes. *Compare with* massively parallel processing system.

synonym. A name that is assigned to a table, view, or sequence and that can be used in place of the original name. A synonym does not replace the original name; instead, it acts as an alias for the table, view, or sequence.

sysmaster database. A database on each database server that holds the ON-Archive catalog tables and *system-monitoring interface* (SMI) tables that contain information about the state of the database server. The database server creates the **sysmaster** database when it initializes disk space.

system call. A routine in an operating-system *library* that programs call to obtain information from the operating system.

system catalog. A group of database tables that contain information about the database itself, such as the names of tables or columns in the database, the number of rows in a table, the information about indexes and database privileges, and so on. *See also* data dictionary.

system-defined cast. A pre-defined *cast* that is known to the database server. Each built-in cast performs automatic conversion between two different *built-in data types*.

system-defined hash fragmentation. An Extended Parallel Server-defined distribution scheme that maps each row in a table to a set of

integers and uses a system-defined algorithm to distribute data evenly by hashing a specified key.

system-descriptor area. A dynamic SQL management structure that is used with the ALLOCATE DESCRIPTOR, DEALLOCATE DESCRIPTOR, DESCRIBE, GET DESCRIPTOR, and SET DESCRIPTOR statements to store information about database columns or host variables used in dynamic SQL statements. The structure contains an item descriptor for each column; each item descriptor provides information such as the name, data type, length, scale, and precision of the column. The system-descriptor area is the X/Open standard for handling dynamic columns. *See also* descriptor and sqlda.

system index. An index that the database server creates to implement a *unique constraint* or a *referential constraint*. A system index is distinct from a *user index*, which a user creates explicitly.

system-monitoring interface. A collection of tables and pseudo-tables in the **sysmaster** database that maintains dynamically updated information about the operation of the database server. The tables are constructed in memory but are not recorded on disk. Users can query the SMI tables with the SELECT statement of SQL.

table. A rectangular array of data in which each row describes a single entity and each column contains the values for each category of description. For example, a table can contain the names and addresses of customers. Each row corresponds to a different customer and the columns correspond to the name and address items. A table is sometimes referred to as a *base table* to distinguish it from the views, indexes, and other objects defined on the underlying table or associated with it.

table fragment. Zero or more rows that are grouped together and stored in a *dbspace* that you specify when you create the fragment. A virtual table fragment might reside in an *sbspace* or an *extspace*.

table fragmentation. A method of separating a table into potentially balanced fragments to

distribute the workload and optimize the efficiency of the database operations. Also known as data partitioning. Table-fragmentation methods (also known as distribution schemes) include *expression-based*, *hybrid*, *range*, *round-robin*, and *system-defined hash*.

table hierarchy. A relationship you can define among *typed tables* in which *subtables* inherit the behavior (constraints, triggers, storage options) from *supertables*. Subtables can add additional constraint definitions, storage options, and triggers.

table inheritance. The property that allows a *typed table* to inherit the behavior (constraints, storage options, triggers) from a *typed table* above it in the *table hierarchy*.

target table. The underlying base table that a violations table and diagnostics table are associated with. You use the START VIOLATIONS TABLE statement to create the association between the target table and the violations and diagnostics tables.

tblspace. The logical collection of *extents* that are assigned to a table. It contains all the disk space that is allocated to a given table or table fragment and includes pages allocated to data and to indexes, pages that store TEXT or BYTE data in the dbspace, and bitmap pages that track page use within the extents.

TCP/IP. The specific name of a particular standard transport layer protocol (TCP) and network layer protocol (IP). A popular network protocol used in DOS, UNIX, and other environments.

temporary. An attribute of any file, index, or table that is used only during a single session. Temporary files or resources are typically removed or freed when program execution terminates or an online session ends.

terabyte. A unit of storage, equal to 1024 gigabytes or 1024^4 bytes.

TEXT. A *data type* for a *simple large object* that stores text and can be as large as 2^{31} bytes. *See also* BYTE.

thread. A piece of work or task for a *virtual processor* just as a virtual processor is a task for a CPU. A virtual processor is a task that the operating system schedules for execution on the CPU; a database server thread is a task that a virtual processor schedules internally for processing. Threads are sometimes called *lightweight processes* because they are like processes but make fewer demands on the operating system. *See also* multithreading and user thread.

TLLI. Acronym for Transport Layer Interface. It is the interface designed for use by application programs that are independent of a network protocol.

trace. To keep a running list of the values of program variables, arguments, expressions, and so on, in a program or SPL routine.

transaction. A collection of one or more SQL statements that is treated as a single unit of work. If one statement in a transaction fails, the entire transaction can be *rolled back* (canceled). If the transaction is successful, the work is *committed* and all changes to the database from the transaction are accepted. *See also* explicit transaction, implicit transaction, and singleton implicit transaction.

transaction lock. A lock on an *R-tree index* that is obtained at the beginning of a transaction and held until the end of the transaction.

transaction logging. The process of keeping records of transactions. *See also* logical log.

transaction mode. The method by which constraints are checked during transactions. You use the SET statement to specify whether constraints are checked at the end of each data manipulation statement or after the transaction is committed.

trigger. A database object that executes a set of actions if a DML event manipulates a specified table. (An INSTEAD OF trigger substitutes a set of actions for a DML event that attempts to manipulate a specified view.)

tuple. *See* row.

two-phase commit. A protocol that ensures that transactions are uniformly committed or rolled back across multiple database servers. It governs the order in which commit transactions are performed and provides a recovery mechanism in case a transaction does not execute. *See also* heterogeneous commit.

type constructor. An SQL keyword that indicates to the database server the type of complex data to create (for example, *LIST*, *MULTISET*, *ROW*, *SET*).

type hierarchy. A relationship that you define among *named row types* in which *subtypes* inherit representation (data *fields*) and behavior (*routines*) from *supertypes* and can add more fields and routines.

type inheritance. The property that allows a *named row type* or *typed table* to inherit representation (data *fields*, columns) and behavior (*routines*, operators, rules) from a named row type above it in the *type hierarchy*.

type substitutability. The ability to use an instance of a subtype when an instance of its supertype is expected.

typed collection variable. An ESQL/C *collection variable* or *SPL variable* that has a defined *collection data type* associated with it and can only hold a *collection* of its defined type. *See also* untyped collection variable.

typed table. A table that is constructed from a *named row type* and whose rows contain instances of that *row type*. A typed table can be used as part of a *table hierarchy*. The columns of a typed table correspond to the *fields* of the named row type.

UDA. *See* user-defined aggregate.

UDF. *See* user-defined function.

UDR. *See* user-defined routine.

UDT. *See* user-defined data type.

unbuffered disk I/O. Disk I/O that is controlled directly by the database server instead

of the operating system. This direct control helps improve performance and reliability for updates to data. Unbuffered I/O is supported by character-special files on UNIX and by both unbuffered files and the raw disk interface on Windows.

Uncommitted Read. *See* Read Uncommitted.

uncorrelated subquery. *See* independent subquery.

unique constraint. Specifies that each entry in a column or set of columns has a unique value.

unique index. An index that prevents duplicate values in the indexed column.

unique key. *See* primary key.

UNIX real user ID. *See* Informix user ID.

unload job. The information required to unload data from a relational database using the HPL. This information includes format, map, query, device array, project, and special options.

unlock. To free an object (database, table, page, or row) that has been locked. For example, a locked table prevents others from adding, removing, updating, or (in the case of an exclusive lock) viewing rows in that table as long as it is locked. When the user or program unlocks the table, others are permitted access again.

unnamed row type. A *row type* created with the ROW constructor that has no defined name and no inheritance properties. Two unnamed row types are equivalent if they have the same number of *fields* and if corresponding fields have the same *data type*, even if the fields have different names.

untyped collection variable. A generic ESQL/C *collection variable* or *SPL variable* that can hold a *collection* of any *collection data type* and takes on the data type of the last collection assigned to it. *See also* typed collection variable.

updatable view. A view whose underlying table can be modified by inserting values into the view. Only an INSTEAD OF trigger can update a multi-table view.

update. The process of changing the contents of one or more columns in one or more existing rows of a table.

update lock. A promotable lock that is acquired during a SELECT...FOR UPDATE. An update lock behaves like a shared lock until the update actually occurs, and it then becomes an exclusive lock. It differs from a shared lock in that only one update lock can be acquired on an object at a time.

user-defined aggregate. An aggregate function that is not provided by the database server (built in) that includes extensions to built-in aggregates and newly defined aggregates. The database server manages all aggregates.

user-defined base type. See opaque data type.

user-defined cast. A cast that a user creates with the CREATE CAST statement. A user-defined cast typically requires a *cast function*. A user-defined cast can be an *explicit cast* or an *implicit cast*.

user-defined data type. A *data type* that you define for use in a relational database. You can define opaque data types and distinct data types.

user-defined function. A *user-defined routine* that returns at least one value. You can write a user-defined function in SPL (*SPL function*) or in an external language that the database server supports (*external function*).

user-defined procedure. A *user-defined routine* that does not return a value. You can write a user-defined procedure in SPL (*SPL procedure*) or in an external language that the database server supports (*external procedure*).

user-defined routine. A *routine* that you write and register in the system catalog tables of a database, and that an SQL statement or another routine can invoke. You can write a user-defined

routine in SPL (*SPL routine*) or in an external language (*external routine*) that the database server supports.

user ID. Also called *authorization identifier*. See also Informix user ID.

user ID password. See Informix user password.

user index. An index that a user creates explicitly with the CREATE INDEX statement. Compare with system index.

user name. See Informix user ID.

user password. See Informix user password.

user thread. User threads include session threads (called **sqlexec** threads) that are the primary threads that the database server runs to service client applications. User threads also include a thread to service requests from the **onmode** utility, threads for recovery, and page-cleaner threads. See *thread*.

variable. The *identifier* for a location in memory that stores the value of a program object whose value can change during program execution. Compare with constant, macro, and pointer.

variant function. A *user-defined function* that might return different values when passed the same arguments. A variant function can contain SQL statements. Compare with nonvariant function.

view. A dynamically controlled subset of the columns of one or more database tables. A view can give the programmer control over what information the user sees and manipulates and represents a virtual table that holds the results of a specified SELECT statement.

violations table. A special table that holds rows that fail to satisfy constraints and unique index requirements during data manipulation operations on base tables. You use the START VIOLATIONS TABLE statement to create a violations table and associate it with a base table.

virtual column. A derived column of information, created with an SQL statement that

is not stored in the database. For example, you can create virtual columns in a SELECT statement by arithmetically manipulating a single column, such as multiplying existing values by a constant, or by combining multiple columns, such as adding the values from two columns.

virtual-column index. A type of *generalized-key index* that contains keys that are the result of an expression.

virtual processor. A multithreaded process that makes up the database server and is similar to the hardware processors in the computer. It can serve multiple clients and, where necessary, run multiple threads to work in parallel for a single query.

virtual table. A table created to access data in an external file, external DBMS, smart large object, or in the result set of an iterator function in a query. The database server does not manage external data or directly manipulate data within a smart large object. The Virtual-Table Interface allows users to access the external data in a virtual table using SQL DML statements and join the external data with Dynamic Server table data.

VLDB. Acronym for very large database.

warning. A message or other indicator about a condition that software (such as the database server or compiler) detects. A condition that results in a warning does not necessarily affect the ability of the code to run. *See also* compile-time error *and* runtime error.

white space. A series of one or more space characters. The GLS locale defines the characters that are considered to be space characters. For example, both the TAB and blank might be defined as space characters in one locale, but certain combinations of the CTRL key and another character might be defined as space characters in a different locale.

wide character. A form of a code set that involves normalizing the size of each multibyte character so that each character is the same size. This size must be equal to or greater than the

largest character that an operating system can support, and it must match the size of an integer data type that the C compiler can scale. Some examples of an integer data type that the C compiler can scale are short integer (**short int**), integer (**int**), or long integer (**long int**).

wildcard. A special symbol representing any character or any string of zero or more characters. In SQL, for example, you can use the asterisk (*), question mark (?), percent sign (%), and underscore (_) as wildcard characters in some contexts. (Asterisk and question mark are also UNIX wildcards.)

window. A rectangular area on the screen in which you can take actions without leaving the context of the background program.

WORM. Acronym for *Write-Once-Read-Many* optical media. When a bit of data is written to a WORM platter, a permanent mark is made on that optical platter.

X/Open. An independent consortium that produces and develops specifications and standards for open-systems products and technology, such as dynamic SQL.

X/Open Portability Guide (XPG). A set of specifications that vendors and users can use to build portable software. Any vendor that carries the XPG brand on a given software product is guaranteeing that the software correctly implements the X/Open Common Applications Environment (CAE) specifications. There are CAE specifications for SQL, XA, ISAM, RDA, and so on.

zoned decimal. (1) A data representation that uses the low-order four bits of each byte to designate a decimal digit (0 through 9) and the high-order four bits to designate the sign of the digit. (2)

Notices

IBM may not offer the products, services, or features discussed in this document in all 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.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

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. 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 IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. (enter the year or years).
All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

AIX; DB2; DB2 Universal Database; Distributed Relational Database Architecture; NUMA-Q; OS/2, OS/390, and OS/400; IBM Informix®; C-ISAM®; Foundation.2000™; IBM Informix® 4GL; IBM Informix® DataBlade® Module; Client SDK™; Cloudscape™; Cloudsync™; IBM Informix® Connect; IBM Informix® Driver for JDBC; Dynamic Connect™; IBM Informix® Dynamic Scalable Architecture™ (DSA); IBM Informix® Dynamic Server™; IBM Informix® Enterprise Gateway Manager (Enterprise Gateway Manager); IBM Informix® Extended Parallel Server™; i.Financial Services™; J/Foundation™; MaxConnect™; Object Translator™; Red Brick™; IBM Informix® SE; IBM Informix® SQL; InformiXML™; RedBack®; SystemBuilder™; U2™; UniData®; UniVerse®; wintegrate® are trademarks or registered trademarks of International Business Machines Corporation.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Windows, Windows NT, and Excel are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names used in this publication may be trademarks or service marks of others.

Index

Special characters

- (_), underscore
 - in SQL identifiers 3-43
- (;), semicolon
 - list separator 3-58, 3-72
- (:), colon
 - cast (::) operator 2-53, 2-57
 - DATETIME delimiter 2-13
 - INTERVAL delimiter 2-21
 - list separator 3-36, 3-43, 3-58, 3-67, 3-72
- (!=), not equal to
 - relational operator 2-57
- (/), slash
 - DATE separator 2-11, 2-44, 3-26
 - division operator 2-41, 2-57
 - pathname delimiter 3-10, 3-34, 3-67
- ('), single quotes
 - string delimiter 3-31, 3-43
- ("), double quotes
 - delimited SQL identifiers 3-43
 - string delimiter 2-1, 2-23, 2-26, 2-33
- ((), parentheses
 - delimiters in expressions 2-45
- (\$), dollar sign
 - currency symbol 2-24, 3-31
 - pathname indicator 3-18, 3-71
- (\), backslash
 - invalid as delimiter 3-28
 - pathname delimiter 3-12, 3-61
- ([]), brackets
 - MATCHES range delimiters 2-38
 - substring operator 2-8, 2-35, 2-57
- (%), percentage
 - DBTIME escape symbol 3-39
 - pathname indicator 3-18, 3-38
- (>), greater than
 - angle (< >) brackets 2-8, A-22
 - relational operator 1-9, 2-57
- (<), less than
 - angle (< >) brackets 2-8, A-22
 - relational operator 2-57, 3-28
- (|), vertical bar
 - absolute value delimiter 2-19
 - concatenation (||) operator 2-57
 - field delimiter 3-28
- (#), sharp
 - comment indicator 3-6
- ({ }), braces
 - collection delimiters 2-23, 2-26, 2-33, 2-48
 - ({ }), braces (*continued*)
 - pathname delimiters 3-8
- (-), hyphen
 - DATE separator 3-26
 - DATETIME delimiter 2-13
 - INTERVAL delimiter 2-21
 - subtraction operator 2-41, 2-57
 - symbol in syscolauth 1-7, 1-20
 - symbol in sysfragauth 1-33
 - symbol in systabauth 1-53
 - unary operator 2-42, 2-57
- (,), comma
 - decimal point 3-31
 - list separator 2-25, 2-29, 3-36
 - thousands separator 2-24
- (.), period
 - DATE separator 3-26, 3-27
 - DATETIME delimiter 2-13
 - decimal point 2-17, 2-24, 3-31
 - execution symbol 3-6
 - INTERVAL delimiter 2-21
 - membership operator 2-57
 - nested dot notation 2-48
- (), blank space
 - DATETIME delimiter 2-13
 - INTERVAL delimiter 2-21
 - padding CHAR values 2-9
 - padding VARCHAR values 2-37
- (*), asterisk
 - multiplication operator 2-32, 2-41, 2-45, 2-57
 - systabauth value 1-7, 1-53
 - wildcard symbol 1-18, 1-64
- (+), plus sign
 - addition operator 2-41, 2-57
 - truncation indicator 3-51
 - unary operator 2-57
- (=), equality
 - assignment operator 3-12
 - relational operator 1-18, 2-6, 2-10, 2-57
- (~), tilde
 - pathname indicator 3-10

A

- Abbreviated year values 2-14, 3-22, 3-24, 3-27, 3-40
- AC_CONFIG environment variable 3-18
- ac_config.std file 3-18
- ACCESS keyword 1-12, 2-40
- Access method
 - B-tree 1-15, 1-37, 3-42
 - built-in 1-12, 1-15

Access method (*continued*)

- primary 1-13, 1-52
- R-tree 3-42
- secondary 1-12, 1-26, 1-39, 2-27
- sysams data 1-12
- sysindices data 1-39
- sysopclasses data 1-41
- systabamdata data 1-52

Access privilege.

See Privilege.

Accessibility xxi

- dotted decimal format of syntax diagrams C-1
- syntax diagrams, reading in a screen reader C-1

Activity-log files 3-66

Addition (+) operator 2-41, 2-57

Administrative listener port 3-54

AFCRASH configuration parameter 3-18

AFDEBUG environment variable 3-18

Aggregate function

- built-in 2-23, 2-25, 2-33
- no BYTE argument 2-7
- no collection arguments 2-23, 2-25, 2-33
- no TEXT argument 2-35
- sysaggregates data 1-12
- user-defined 1-12

AIX operating system 3-68

Alias of a table 1-6

Alignment of data type 1-18, 1-62

ALL operator 2-57

ALTER OPTICAL CLUSTER statement 1-43

Alter privilege 1-7, 1-53, 1-64

ALTER SEQUENCE statement 3-80

ALTER TABLE statement

- casting effects 2-51
- changing data types 2-3
- lock mode 3-49
- next extent size 1-9
- SERIAL columns 2-31
- SERIAL8 columns 2-32
- synonyms 3-80
- systables.version 1-55

am_beginscan() function 1-14

am_close() function 1-14

am_getnext() function 1-13

am_insert() function 1-14

am_open() function 1-14

AND operator 1-18, 2-57

ANSI compliance

- ansi flag 3-22
- DATETIME literals 3-40
- DBANSIWARN environment variable 3-21
- DECIMAL range 2-16
- DECIMAL(p) data type 2-15
- Information Schema views 1-63
- isolation level 1-67

ANSI compliance (*continued*)

- public synonyms 1-52, 1-54

ANSIOWNER environment variable 3-18

ANY operator 2-57

Arabic locales 2-8

archecker utility 3-18

Archiving

- setting DBREMOTECMD 3-36

Arithmetic

- DATE operands 2-11, 2-43
- DATETIME operands 2-42
- integer operands 2-19, 2-32, 2-34
- INTERVAL operands 2-20, 2-42
- operators 2-56
- string operands 2-9
- time operands 2-40

AS keyword 2-53, 2-54

ASCII code set 1-32

assign() support function 2-47

AT keyword 2-22

Attached indexes 1-36, 3-25, 3-76

Audit Analysis officer 3-63

See AAO

Authentication information file 3-59

Authorization identifier 1-59, 1-67

B

B-tree access method 1-15, 1-37, 3-42

B+ tree index 1-35

Backslash (\) symbol 3-28

Bandwidth 3-53

BETWEEN operator 2-57

bin subdirectory 3-8

Binding style 1-66

Blank spaces 3-67

BLOB data type

- casting not available 2-6
- coltype code 1-23
- defined 2-6
- inserting data 2-6
- syscolattribs data 1-19

Blobspace

- defined 2-39
- memory cache for staging 3-59
- names 3-43
- sysblobs data 1-17

Boldface type xii

BOOLEAN data type

- coltype code 1-23
- defined 2-7

Boolean expression

- with BOOLEAN data type 2-7
- with BYTE data type 2-7
- with TEXT data type 2-35

Borland C compiler 3-55

- Bourne shell 3-6, 3-7
- Braces ({ }) symbols 2-48
- Bracket ([]) symbols 2-35
- Buffer
 - BYTE or TEXT storage (DBBLOBBUF) 3-22
 - fetch buffer (FET_BUFFER_SIZE) 3-19, 3-44
 - floating-point display (DBFLTMASK) 3-29
 - network buffer (IFX_NETBUF_SIZE) 3-52
 - private network buffer pool 3-52
- Built-in access method 1-13, 1-15
- Built-in aggregates 1-12, 2-23, 2-25, 2-33
- Built-in casts 1-17, 2-50
- Built-in data types
 - casts 2-50, 2-56
 - listed 2-38
 - syscolumns.coltype code 1-22
 - sysdistrib.type code 1-29
 - sysxdtypes data 1-62
- BY keyword 2-7, 2-35
- BYTE data type
 - casting to BLOB 2-8
 - defined 2-7
 - increasing buffer size 3-22
 - inserting values 2-8
 - restrictions
 - in Boolean expression 2-7
 - sysables.npused 1-55
 - with GROUP BY 2-7
 - with LIKE or MATCHES 2-7
 - with ORDER BY 2-7
 - selecting from BYTE columns 2-8
 - setting buffer size 3-22
 - sysblobs data 1-16
 - syscolumns data 1-25
 - sysfragments data 1-34
 - sysopclstr data 1-42

C

- C compiler
 - default name 3-55
 - INFORMIXC setting 3-55
 - thread package 3-80
- C shell 3-6
 - .cshrc file 3-7
 - .login file 3-7
- C++ map file 3-58
- call_type table in stores_demo database A-5
- call_type table in superstores_demo database B-6
- CARDINALITY() function 2-23, 2-25, 2-33
- Cartesian join 3-46
- Cascading delete 1-49
- Cast 2-50, 2-56
 - built-in 1-17, 2-50, 2-54
 - distinct data type 2-54
 - explicit 1-17, 2-53, 2-54

- Cast (*continued*)
 - from BYTE to BLOB 2-8
 - from TEXT to CLOB 2-35
 - implicit 1-17, 2-53, 2-54
 - rules of precedence 2-54
 - syscasts data 1-17
- Cast (::) operator 2-53, 2-57
- CAST AS keywords 2-53
- CHAR data type
 - built-in casts 2-52
 - collation 2-8, 2-9, 2-38
 - conversion to NCHAR 2-26, 3-32
 - defined 2-8
 - nonprintable characters 2-9
 - storing numeric values 2-9
- CHARACTER data type.
 - See CHAR data type.
- Character data types
 - Boolean comparisons 2-37
 - casting between 2-50
 - data strings 2-5
 - listed 2-38
 - syscolumns data 1-24
- Character string
 - CHAR data type 2-8
 - CHARACTER VARYING data type 2-10
 - CLOB data type 2-10
 - DATETIME literals 2-14, 2-44, 3-40
 - INTERVAL literals 2-21
 - LVARCHAR data type 2-23
 - NCHAR data type 2-26
 - NVARCHAR data type 2-26
 - TEXT data type 2-34
 - VARCHAR data type 2-36
 - with DELIMIDENT set 3-43
- CHARACTER VARYING data type
 - See also VARCHAR data type.
 - defined 2-9
 - length (syscolumns) 1-24
- Character-based applications 3-62, 3-78
- Check constraint
 - creation-time value 3-24, 3-27
 - syschecks data 1-18
 - syscheckudrdep data 1-18
 - syscoldepend data 1-20
 - sysconstraints data 1-26
- chkenv utility 3-6
 - error message 3-9
 - syntax 3-9
- Chunk 2-39, 3-37
- CLIENT_LOCALE environment variable 3-27
- Client/server
 - Datablade API 2-40
 - default database 3-60
 - INFORMIXSQLHOSTS environment variable 3-61

- Client/server (*continued*)
 - shared memory communication segments 3-60
 - stacksize for client session 3-62
- CLOB data type
 - casting not available 2-10
 - code-set conversion 2-11
 - collation 2-11
 - coltype code 1-23
 - defined 2-10
 - inserting data 2-10
 - multibyte characters 2-10
 - syscolattns data 1-19
- CLOSE statement 3-70
- Clustering 1-13, 1-35, 1-38
- Code set
 - ASCII 1-32
 - collation order 2-37
 - conversion 3-82
 - East Asian 2-9, 2-37, 3-41
 - EBCDIC 1-32, 1-67
 - ISO 8859-1 xi, 1-30
- Code, sample, conventions for xviii
- Collation
 - CHAR data type 2-8, 2-9
 - CLOB data type 2-11
 - GL_COLLATE table 1-56
 - NCHAR data type 2-26
 - server_attribute data 1-67
 - TEXT data type 2-35
 - VARCHAR data type 2-37
- Collection data type
 - casting matrix 2-56
 - defined 2-47
 - empty 2-48
 - LIST 2-22
 - MULTISET 2-25
 - SET 2-32
 - sysattrtypes data 1-15
 - sysxtddesc data 1-61
 - sysxtdtypes data 1-61
- Colon
 - cast (::) operator 2-53
 - DATETIME delimiter 2-13
 - INTERVAL delimiter 2-21
 - pathname separator 3-67
- Color and intensity screen attributes 3-62
- Column
 - changing data type 2-3, 2-50
 - constraints (sysconstraints) 1-26
 - default values (sysdefaults) 1-27
 - hashed 1-34
 - in sales_demo database B-3, B-4
 - in stores_demo database A-2, A-5
 - in superstores_demo database B-6, B-16
 - inserting BLOB data 2-6
- Column (*continued*)
 - range of values 1-25
 - referential constraints (sysreferences) 1-49
 - syscolumns data 1-22
- Column-level privileges
 - systabauth data 1-7
 - systabauth table 1-53
- Combine function 1-12
- Command-line conventions
 - how to read xv
 - sample diagram xv
- Comment indicator 3-6
- Comment lines 3-6
- Committed read 1-67
- Communications support module 3-56, 3-59
- Commutator function 1-46
- Compiling
 - ESQL/C programs 3-20
 - INFORMIXC setting 3-55
 - JAVA_COMPILER setting 3-66
 - multithreaded ESQL/C applications 3-79
- Complex data type 2-46, 2-49
 - collection types 2-47
 - ROW types 2-48
 - sysattrtypes data 1-15
- Compliance
 - ANSI/ISO standard for SQL 1-63, 3-21
 - sql_languages.conformance 1-66
 - with industry standards xxiv
 - X/Open CAE standards 1-63
 - XPG4 standard 1-65
- Composite index 1-37
- Concatenation (||) operator 2-57
- conscm.cfg file 3-56
- Configuration file
 - .cshrc file 3-7
 - .informix 3-6, 3-9, 3-44, 3-49
 - .login file 3-7
 - .profile file 3-7
 - for communications support module 3-56, 3-59
 - for connectivity 3-55, 3-60, 3-61
 - for database servers 3-43, 3-68
 - for High-Performance Loader 3-74
 - for MaxConnect 3-55
 - for ON-Bar utility 3-18
 - for onxfer utility 3-81
 - for terminal I/O 3-62, 3-79
- Configuration parameter
 - COSERVER 3-69
 - DBSPACETEMP 3-37
 - DEF_TABLE_LOCKMODE 3-49
 - DIRECTIVES 3-49
 - DISABLE_B162428_XA_FIX 3-54
 - END 3-69
 - EXT_DIRECTIVES 1-28, 3-50

Configuration parameter (*continued*)

MITRACE_OFF 1-56, 1-57
NODE 3-69
OPCACHEMAX 3-59
OPT_GOAL 3-71
OPTCOMPIND 3-70
STACKSIZE 3-62
STMT_CACHE 3-78
TABLESPACE 3-38
USEOSTIME 2-15

CONNECT DEFAULT statement 3-60

Connect privilege 1-8, 1-59

CONNECT statement 3-33, 3-57, 3-60

Connection

authentication 3-59
coserver 3-60
INFORMIXCONRETRY environment variable 3-56
INFORMIXCONTIME environment variable 3-57
INFORMIXSERVER environment variable 3-60

Connectivity information 3-54, 3-61

Constraint

check

creation-time value 3-27
loading performance B-3
syschecks data 1-18
syscheckudrdep data 1-18
syscoldepend data 1-20

column

sysconstraints data 1-26

not null

collection data types 2-23, 2-25, 2-33, 2-48
syscoldepend data 1-20
syscolumns data 1-23
sysconstraints data 1-26

object mode 1-41

primary key

sysconstraints data 1-26
sysreferences data 1-49
unique SERIAL values 2-30
unique SERIAL8 values 2-31

referential

stores_demo data A-8
superstores_demo data B-19
sysconstraints data 1-26
sysreferences data 1-49

table

sysconstraints data 1-26

unique

sysconstraints data 1-26

violations 1-60

Constructor 2-33, 2-48

Contact information xxv

Conventions

command-line xv
documentation xii

Conventions (*continued*)

sample-code xvii
syntax diagrams xiii
syntax notation xiii
typographical xii

Converting data types

CHAR and NCHAR 3-32
DATE and DATETIME 2-53
INTEGER and DATE 2-52
number and string 2-52
number to number 2-51
retyping a column 2-50
VARCHAR and NVARCHAR 3-32

Coserver

sysexternal data 1-32
sysviolations data 1-60

COSERVER configuration parameter 3-60, 3-69

CPFIRST environment variable 3-20

CPU cost 3-77

CREATE ACCESS METHOD statement 1-12

CREATE CAST statement 1-17, 2-53

CREATE DATABASE statement 3-33

CREATE DISTINCT TYPE statement 1-62, 2-18, B-17

CREATE DUPLICATE statement 1-36

CREATE EXTERNAL TABLE statement 1-31, 1-32

CREATE FUNCTION statement 1-50

CREATE IMPLICIT CAST statement B-17

CREATE INDEX statement 1-36, 1-38, 1-40, 1-49, 1-55, 3-42

storage options 3-42

CREATE OPAQUE TYPE statement 2-27

CREATE OPERATOR CLASS statement 1-42

CREATE OPTICAL CLUSTER statement 1-43

CREATE PROCEDURE statement 1-50, 3-68

CREATE ROLE statement 1-50

CREATE ROUTINE FROM statement 1-50, 3-68

CREATE ROW TYPE statement 1-23, 2-28

CREATE SCHEMA statement 1-3

CREATE SEQUENCE statement 1-51

CREATE SYNONYM statement 1-51

CREATE TABLE statement

assigning data types 2-3

default lock mode 3-49

default privileges 3-68

SET constructor 2-33

typed tables 2-28

CREATE TEMP TABLE statement 3-37

CREATE TRIGGER statement 1-58

CREATE VIEW statement 1-6, 1-59

Credential 3-59

Currency symbol 2-24, 3-31

Current date 1-27, 3-23

CURRENT keyword 2-41, 3-45

cust_calls table in stores_demo database A-4

cust_calls table in superstores_demo database B-7

customer table in sales_demo database B-3
customer table in stores_demo database A-2
customer table in superstores_demo database B-8, B-9

D

Data compression 3-64
Data corruption 1-9, 1-19
Data dependencies
 syscheckudrdep data 1-19
 syscoldepend data 1-20
 sysdepend data 1-27
 sysnewdepend data 1-40
Data dictionary 1-2
Data distributions 1-10, 1-28, 3-41
Data encryption 3-65
Data integrity 1-66
Data pages 1-19, 1-37, 1-54
Data type
 BLOB 2-6
 BOOLEAN 2-7
 BYTE 2-7
 CHAR 2-8
 CHARACTER 2-6, 2-9
 CHARACTER VARYING 2-9
 CLOB 2-10
 DATE 2-11
 DATETIME 2-11
 DEC 2-15
 DECIMAL 2-15
 DISTINCT 2-17
 DOUBLE PRECISION 2-18
 FLOAT 2-18
 INT 2-19
 INT8 2-19
 INTEGER 2-19
 INTERVAL 2-19
 LIST 2-22
 LVARCHAR 2-23
 MONEY 2-24
 MULTISET 2-25
 NCHAR 2-26
 NUMERIC 2-26
 NVARCHAR 2-26
 OPAQUE 2-26
 REAL 2-27
 ROW 2-27, 2-29
 SERIAL 2-30
 SERIAL8 2-31
 SET 2-32
 SMALLFLOAT 2-33
 SMALLINT 2-34
 TEXT 2-34
 VARCHAR 2-36
Data types
 approximate 1-65
Data types (*continued*)
 casting 2-50, 2-56
 classified by category 2-2
 collection 2-47
 complex 2-46
 conversion 2-50
 distinct 2-17, 2-49
 exact numeric 1-65
 extended 2-46
 fixed point 2-16
 floating-point 2-15, 2-19, 2-34
 inheritance 2-28
 internal 2-5
 named ROW 2-27
 opaque 2-26, 2-49
 sequential integer 2-31
 simple large object 2-39
 smart large object 2-39
 summary list 2-3
 unique numeric value 2-31
 unnamed ROW 2-28
Data warehousing B-1
Database
 data types 2-2
 dimensional B-2
 identifiers 3-42
 joins in stores_demo A-7
 object-relational B-1
 objects, sysobjectstate data 1-41
 privileges 1-59
 sales_demo B-1
 stores_demo A-1
 superstores_demo B-3, B-5
 syscrd 1-3
 sysmaster 1-3
 sysutils 1-3
 sysuuid 1-3
Database identifiers 3-43
Database server
 attributes in Information Schema view 1-66
 codeset 1-67
 coserver name 3-60
 default connection 3-60
 default isolation level 1-67
 optimizing queries 3-71
 pathname for 3-33
 remote 3-20, 3-44
 role separation 3-63
 server name 1-27, 3-34
Database Server Administrator (DBSA) 3-63
DATABASE statement 3-33
Database system administrator (DBSA) 1-3
DataBlade module
 Client and Server API 2-40
 data types (sysbuiltintypes) 1-3

- DataBlade module (*continued*)
 - trace messages (systracemsgs) 1-56, 1-57
 - user messages (syserrors) 1-30
- DATE data type
 - abbreviated year values 3-22
 - casting to integer 2-52
 - converting to DATETIME 2-53
 - defined 2-11
 - display format 3-25
 - in expressions 2-40, 2-43
 - international date formats 2-11
 - source data 2-43
- DATE() function 2-44, 3-27
- DATETIME data type
 - abbreviated year values 3-22
 - converting to DATE 2-52, 2-53
 - defined 2-11
 - display format 3-39
 - EXTEND function 2-43
 - extending precision 2-42
 - field qualifiers 2-12
 - in expressions 2-40, 2-45
 - international formats 2-13, 2-15, 2-21
 - length (syscolumns) 1-24
 - literal values 2-14
 - precision and size 2-12
 - source data 2-44
 - two-digit year values and DBDATE variable 2-14
 - year to fraction example 2-13
- DAY keyword
 - DATETIME qualifier 2-12
 - INTERVAL qualifier 2-20
 - UNITS operator 2-11, 2-44
- DB-Access utility 1-9, 1-63, 3-5, 3-29, 3-33, 3-38, 3-60
- DBA privilege 1-30, 1-56, 1-57, 1-59
- DBA routines 1-47
- DBACCNOIGN environment variable 3-20
- DBANSIWARN environment variable 3-21
- DBBLOBBUF environment variable 3-22
- DBCENTURY environment variable
 - defined 3-22
 - effect on functionality of DBDATE 3-27
 - expanding abbreviated years 2-14, 3-23
- DBDATE environment variable 2-11, 3-25
- DBDELIMITER environment variable 3-28
- DBEDIT environment variable 3-28
- dbexport utility 3-28
- DBFLTMASK environment variable 3-29
- DBLANG environment variable 3-29
- dbload utility 2-6, 2-8, 2-35, 3-28
- DBMONEY environment variable 2-24, 3-30
- DBNLS environment variable 3-32
- DBONPLOAD environment variable 3-33
- DBPATH environment variable 3-33
- DBPRINT environment variable 3-35
- DBREMOTECMD environment variable 3-36
- dbschema utility 3-80
- Dobserver group 3-60
- DBSERVERNAME configuration parameter 3-60
- dbservername.cmd batch file 3-13
- dbslice 1-34, 1-35
- dbspace
 - for BYTE or TEXT values 1-17
 - for system catalog 1-3
 - for table fragments 1-33
 - for temporary tables 3-36
 - name 3-43
- DBSPACE keyword 1-34
- DBSPACETEMP configuration parameter 3-36
- DBSPACETEMP environment variable 3-36
- DBTEMP environment variable 3-38
- DBTIME environment variable 2-15, 3-39
- DBUPSPACE environment variable 3-41
- DCE-GSS communications support module (CSM) 3-59
- DEC data type.
 - See* DECIMAL data type.
- DECIMAL data type
 - built-in casts 2-52
 - defined 2-15
 - disk storage 2-16
 - display format 3-29, 3-30
 - fixed point 2-16
 - floating point 2-15
 - length (syscolumns) 1-25
- Decimal digits, display of 3-29
- Decimal point
 - DBFLTMASK setting 3-29
 - DBMONEY setting 3-31
 - DECIMAL radix 2-17
- Decimal separator 3-31
- DECLARE statement 3-70
- DECRYPT_BINARY function 2-10
- DECRYPT_CHAR function 2-10
- DEF_TABLE_LOCKMODE configuration parameter 3-49
- Default
 - C compiler 3-55
 - century 3-23, 3-40
 - CHAR length 2-8
 - character set for SQL identifiers 3-43
 - compilation order 3-20
 - configuration file 3-69
 - connection 3-60
 - data type 2-29
 - database server 3-34, 3-60
 - DATE display format 2-11
 - DATE separator 3-26
 - DATETIME display format 2-15
 - DECIMAL precision 2-15

- Default (*continued*)
 - detached indexes 3-42
 - detail level 3-65
 - disk space for sorting 3-42
 - fetch buffer size 3-44
 - heap size 3-66
 - isolation level 1-67
 - join method 3-69
 - level of parallelism 3-73
 - lock mode 3-48
 - message directory 3-30
 - MONEY scale 2-25
 - operator class 1-13, 1-42
 - printing program 3-35
 - query optimizer goal 3-71
 - sysdefaults.default 1-27
 - table privileges 3-68
 - temporary dbspace 3-38
 - termcap file 3-79
 - text editor 3-28
 - Default locale xi
 - DEFAULT_ATTACH environment variable 3-42
 - DEFINE statement of SPL 2-30, 2-31
 - Delete privilege 1-33, 1-53, 3-68
 - DELETE statement 1-9, 1-60
 - Delete trigger 1-58
 - DELIMITED environment variable 3-42
 - DELIMITED files 1-31, 1-32
 - Delimited identifiers 3-42, 3-43
 - Delimiter
 - for DATETIME values 2-13
 - for fields 1-32, 3-28
 - for identifiers 3-42
 - for INTERVAL values 2-21
 - Demonstration database
 - tables A-2, A-5, B-5
 - Dependencies, software x
 - Descending index 1-37
 - DESCRIBE statement 3-53
 - Describe-for-updates 3-53
 - destroy() support function 2-47
 - Detached index 3-42
 - Deutsche mark (DM) currency symbol 3-31
 - Diagnostics table 1-60
 - Dimension tables, in push-down hash joins 3-48
 - DIRECTIVES configuration parameter 3-49
 - Directives for query optimization 3-49, 3-69, 3-71
 - Disabilities, visual
 - reading syntax diagrams C-1
 - Disabled object 1-60
 - Disk space
 - for data distributions 3-41
 - for temporary data 3-37
 - Distinct data type
 - casts 2-54
 - Distinct data type (*continued*)
 - defined 2-17
 - sysxtddesc data 1-61
 - sysxtdtypes data 1-61, 1-62, 2-18
 - Distributed Computing Environment (DCE) 3-80
 - Distributed query 2-46, 3-20, 3-44
 - Documentation conventions xii
 - Documentation Notes xix
 - Documentation set of all manuals xxi
 - Documentation, types of xviii
 - machine notes xix
 - online manuals xxi
 - printed manuals xxi
 - Dollar sign 2-24, 3-31
 - Dotted decimal format of syntax diagrams C-1
 - Double data type of C 2-18
 - DOUBLE PRECISION data type.
 - See FLOAT data type.
 - Double-precision floating-point number 2-18
 - DROP CAST statement B-17
 - DROP DATABASE statement 3-33
 - DROP FUNCTION statement 1-47
 - DROP INDEX statement 1-55
 - DROP OPTICAL CLUSTER statement 1-43
 - DROP PROCEDURE statement 1-47
 - DROP ROUTINE statement 1-47
 - DROP ROW TYPE statement 2-28
 - DROP SEQUENCE statement 3-80
 - DROP TABLE statement 3-80
 - DROP TYPE statement 2-18, 2-27
 - DROP VIEW statement 1-63, 3-80
- ## E
- EBCDIC collation 1-32, 1-67
 - Editor, DBEDIT setting 3-28
 - EMACS text editor 3-29
 - Empty set 2-48
 - en_us.8859-1 locale xi
 - ENCRYPT_DES function 2-10
 - ENCRYPT_TDES function 2-10
 - Encryption 3-65
 - END configuration parameter 3-69
 - Enterprise Replication 1-3
 - env utility 3-8
 - ENVIGNORE environment variable
 - defined 3-6, 3-43
 - relation to chkenv utility 3-9
 - Environment configuration file
 - debugging with chkenv 3-9
 - setting environment variables in UNIX 3-4, 3-6
 - Environment registry key 3-10
 - Environment variable
 - AC_CONFIG 3-18
 - AFDEBUB 3-18
 - ANSIOWNER 3-18

Environment variable (continued)

C8BITLEVEL 3-14
 CLIENT_LOCALE 3-14, 3-27
 CPFIRST 3-20
 DB_LOCALE 3-15
 DBACCNOIGN 3-20
 DBANSIWARN 3-21
 DBBLOBBUF 3-22
 DBCENTURY 3-22
 DBDATE 2-11, 3-25
 DBDELIMITER 3-28
 DBEDIT 3-28
 DBFLTMASK 3-29
 DBLANG 3-29
 DBMONEY 2-24, 3-30
 DBNLS 3-32
 DBONPLOAD 3-33
 DBPATH 3-33
 DBPRINT 3-35
 DBREMOTECMD 3-36
 DBSPACETEMP 3-36
 DBTEMP 3-38
 DBTIME 2-15, 3-39
 DBUPSPACE 3-41
 DEFAULT_ATTACH 3-42
 DELIMIDENT 3-42
 ENVIGNORE 3-43
 ESQLMF 3-15
 FET_BUF_SIZE 3-19, 3-44
 GL_DATE 2-11, 3-25
 GL_DATETIME 2-15, 3-25
 GLOBAL_DETACH_INFORM 3-15, 3-45
 GLS8BITSYS 3-15
 IBM_XPS_PARAMS 3-45
 IFMX_CART_ALRM 3-46
 IFMX_OPT_NON_DIM_TABS 3-48
 IFX_DEF_TABLE_LOCKMODE 3-49
 IFX_DIRECTIVES 3-49
 IFX_EXTDIRECTIVES 1-28, 3-50
 IFX_LONGID 3-51
 IFX_NETBUF_PVTPOOL_SIZE 3-52
 IFX_NETBUF_SIZE 3-52
 IFX_NO_TIMELIMIT_WARNING 3-52
 IFX_OPT_FACT_TABS 3-47
 IFX_PAD_VARCHAR 3-53
 IFX_UPDESC 3-53
 IFX_XASTDCOMPLIANCE_XAEND 3-53
 IMCADMIN 3-54
 IMCCONFIG 3-55
 IMCSERVER 3-55
 INF_ROLE_SEP 3-63
 INFORMIXC 3-55
 INFORMIXCONCSMCFG 3-56
 INFORMIXCONRETRY 3-56
 INFORMIXCONTIME 3-57

Environment variable (continued)

INFORMIXCPPMAP 3-58
 INFORMIXDIR 3-58
 INFORMIXKEYTAB 3-59
 INFORMIXOPCACHE 3-59
 INFORMIXSERVER 3-60
 INFORMIXSHMBASE 3-60
 INFORMIXSQLHOSTS 3-61
 INFORMIXSTACKSIZE 3-62
 INFORMIXTERM 3-62
 INTERACTIVE_DESKTOP_OFF 3-64
 ISM_COMPRESSION 3-64
 ISM_DEBUG_FILE 3-64
 ISM_DEBUG_LEVEL 3-65
 ISM_ENCRYPTION 3-65
 ISM_MAXLOGSIZE 3-65
 ISM_MAXLOGVERS 3-66
 JAR_TEMP_PATH 3-66
 JAVA_COMPILER 3-66
 JVM_MAX_HEAP_SIZE 3-66
 LD_LIBRARY_PATH 3-67
 LIBERAL_MATCH 3-67
 LIBPATH 3-68
 NODEFDAC 3-68
 ONCONFIG 3-68, 3-69
 OPT_GOAL 3-71
 OPTCOMPIND 3-69
 OPTMSG 3-70
 OPTOFC 3-70
 PATH 3-71, 3-72
 PDQPRIORITY 3-72
 PLCONFIG 3-74
 PLOAD_LO_PATH 3-74
 PLOAD_SHMBASE 3-74
 PSORT_DBTEMP 3-75
 PSORT_NPROCS 3-76
 RTREE_COST_ADJUST_VALUE 3-77
 SERVER_LOCALE 3-17
 SHLIB_PATH 3-77
 STMT_CACHE 3-77
 TERM 3-78
 TERMCAP 3-78
 TERMINFO 3-79
 THREADLIB 3-79
 TOBIGINT 3-80
 USETABLENAME 3-80
 XFER_CONFIG 3-81

Environment variables xii

command-line utilities 3-11
 displaying current settings 3-8, 3-12
 how to set

- in Bourne shell 3-7
- in C shell 3-7
- in Korn shell 3-7

 how to set in Bourne shell 3-7

- Environment variables (*continued*)
 - how to set in Korn shell 3-7
 - listed alphabetically 3-14
 - listed by topic 3-81
 - manipulating in Windows environments 3-10
 - modifying settings 3-8
 - overriding a setting 3-6, 3-43
 - rules of precedence in UNIX 3-9
 - rules of precedence in Windows 3-13
 - scope of reference 3-11
 - setting
 - at the command line 3-4
 - for native Windows applications 3-10
 - in a configuration file 3-4
 - in a login file 3-4
 - in a shell file 3-7
 - in Windows environments 3-5
 - with command-line utilities 3-11
 - with the Registry Editor 3-10
 - with the System applet 3-11
 - setting in autoexec.bat 3-12
 - standard UNIX system 3-4
 - types of 3-3
 - unsetting 3-7, 3-12, 3-43
 - view current setting 3-8
 - where to set 3-7
 - Equality (=) operator 2-10
 - Era-based dates 3-41
 - Error message files 3-29
 - Error messages xx
 - esql command 3-20, 3-55
 - ESQL/C
 - DATETIME routines 3-39
 - esqlc command 3-20
 - long identifiers 3-52
 - message chaining 3-70
 - multithreaded applications 3-79
 - program compilation order 3-20
 - Exact numeric data types 1-65
 - Executable programs 3-71
 - Execute privilege 1-44, 3-68
 - Explicit cast 1-17, 2-53
 - Explicit pathnames 3-12, 3-35
 - Explicit temporary tables 3-37
 - Exponent 2-17
 - Exponential notation 2-15
 - export utility 3-7
 - export_binary() support function 2-47
 - export() support function 2-47
 - Expression-based fragmentation 1-34, 3-24, 3-27
 - EXT_DIRECTIVES configuration parameter 1-28, 3-50
 - EXTEND function 2-43
 - Extended data types 1-61, 2-46, B-17
 - Extended Parallel Server (XPS) 1-10, 3-14, B-1
 - Extensible Markup Language (XML) 2-10
 - Extension checking (DBANSIWARN) 3-22
 - Extent, changing size 1-9
 - External database 1-52
 - External database server 1-52
 - External directives for query optimization 3-50
 - External routine 1-46
 - External table
 - sysextcols data 1-31
 - sysextdfiles data 1-31
 - sysextdata data 1-32
 - syssyntax data 1-52
 - systables data 1-54
 - External view 1-52
 - extspace 1-13
- ## F
- Fact table
 - dimensional example B-3
 - in push-down hash joins 3-47
 - FALSE setting
 - BOOLEAN value 2-7
 - CPFIRST 3-20
 - ISM_COMPRESSION 3-64
 - Farsi locales 2-8
 - FET_BUF_SIZE environment variable 3-19, 3-44
 - Fetch buffer 3-44
 - Fetch buffer size 3-19, 3-44
 - FETCH statement 3-70
 - Field delimiter
 - DBDELIMITER 3-28
 - Field of a ROW data type 2-48
 - Field qualifier
 - DATETIME values 2-12
 - EXTEND function 2-43
 - INTERVAL values 2-20
 - Fields of a ROW data type 2-48
 - File
 - environment configuration files 3-9
 - installation directory 3-58
 - permission settings 3-6
 - shell 3-6
 - temporary 3-36, 3-38, 3-75
 - temporary for SE 3-38
 - termcap, terminfo 3-62, 3-78, 3-79
 - File extensions
 - .a 3-52
 - .cfg 3-56
 - .cmd 3-13
 - .ec 3-20
 - .ecp 3-20
 - .iem 3-30
 - .jar 3-66
 - .rc 3-6, 3-10, 3-43, 3-49
 - .so 3-52
 - .sql 1-63, 3-33, 3-34, 3-43, B-1, B-5

- File extensions (*continued*)
 - .std 3-18, 3-69, 3-78
 - .xps 3-69
- FILETOBLOB function 2-6
- FILETOCLOB function 2-10
- Filtering mode 1-41, 1-60
- Finalization function 1-12
- Fixed and Known Defects File xix
- FIXED column format 1-31, 1-32
- Fixed point decimal 2-16, 2-24, 3-31
- Fixed-length UDT 1-62
- FLOAT data type
 - built-in casts 2-52
 - defined 2-18
 - display format 3-29, 3-30
- Floating-point decimal 2-15, 2-18, 2-34, 3-29
- Foreign key A-8, B-3
- Formatting
 - DATE values with DBDATE 3-26
 - DATE values with GL_DATE 3-41
 - DATETIME values with DBTIME 3-39
 - DATETIME values with GL_DATETIME 3-41
 - DECIMAL(p) values with DBFLTMASK 3-29
 - FLOAT values with DBFLTMASK 3-29
 - MONEY values with DBMONEY 3-30
 - SMALLFLOAT values with DBFLTMASK 3-29
- Formatting mask
 - with DBDATE 3-25
 - with DBFLTMASK 3-29
 - with DBMONEY 3-30
 - with DBTIME 3-39
 - with GL_DATE 3-41
 - with GL_DATETIME 3-41
- FRACTION keyword
 - DATETIME qualifier 2-12
 - INTERVAL qualifier 2-20
- FRAGMENT BY clause 3-37
- Fragmentation
 - distribution strategy 1-34
 - expression 1-34, 3-24, 3-27
 - list 1-34
 - PDQPRIORITY environment variable 3-73
 - PSORT_NPROCS environment variable 3-77
 - round robin 1-34
 - setting priority levels for PDQ 3-72
 - sysfragauth data 1-33
 - sysfragments data 1-33
- FROM keyword 1-9, 1-18
- Function
 - for BLOB columns 2-6
 - for CLOB columns 2-10
 - for MULTISSET columns 2-25
 - support for complex types 2-47
- Function keys 3-62
- Functional index 1-37, 2-48

- fwritable gcc option 3-56

G

- gcc compiler 3-56
- Generalized-key index
 - sysindexes data 1-36
 - sysnewdepend data 1-40
 - sysrepository data 1-49
- Generic B-trees 1-37
- geography table in sales_demo database B-3
- GET DIAGNOSTICS statement 1-30
- getenv utility 3-5
- GL_COLLATE table 1-56
- GL_CTYPE table 1-56
- GL_DATE environment variable 2-11, 3-25, 3-27
- GL_DATETIME environment variable 2-15, 3-25
- Global Language Support (GLS) x, 3-32
- Global network buffer pool 3-52
- GLOBAL_DETACH_INFORM environment variable 3-45
- Globally detached index 1-35
- GLS environment variables 3-10
- GNU C compiler 3-56
- GRANT statement 1-50
- Graphic characters 3-62
- Greenwich Mean Time (GMT) 3-45
- GROUP BY clause 2-7, 2-35, 3-37
- Group informix 3-30

H

- Hash-join 3-47, 3-48, 3-69
- Hashed columns 1-34
- Hashing parameters 1-52
- Heap size 3-67
- Hebrew locales 2-8
- Help xxi
- Hexadecimal digits 3-28
- HIGH INTEG keywords
 - ALTER TABLE statement 2-40
 - CREATE TABLE statement 2-40
- HIGH keyword
 - PDQPRIORITY 3-73
 - UPDATE STATISTICS 1-10, 1-29
- High-Performance Loader 3-33, 3-74
- Histogram 1-29
- HKEY_LOCAL_MACHINE window 3-10
- Host language 1-66
- Host variable 2-6, 2-8, 2-35, 2-48
- HOUR keyword
 - DATETIME qualifier 2-12
 - INTERVAL qualifier 2-20
- HP-UX operating system 3-77
- HTML (Hypertext Markup Language) 2-10
- Hybrid fragmentation strategy 1-35

Hyphen

- DATETIME delimiter 2-13
- INTERVAL delimiter 2-21

I

I/O overhead 3-77

- IBM Informix Dynamic Server (IDS) 1-10
- IBM Informix ESQL/C 3-20, 3-27, 3-39, 3-52, 3-70
- IBM Informix Extended Parallel Server (XPS) 1-10, 3-14
- IBM Informix Storage Manager (ISM) 3-64, 3-66
- IBM_XPS_PARAMS environment variable 3-45
- IDS (Informix Dynamic Server) 1-10
- IFMX_CART_ALARM environment variable 3-46
- IFMX_OPT_FACT_TABS environment variable 3-47
- IFMX_OPT_NON_DIM_TABS environment variable 3-48
- IFX_DEF_TABLE_LOCKMODE environment variable 3-48
- IFX_DIRECTIVES environment variable 3-49
- IFX_EXTDIRECTIVES environment variable 1-28, 3-50
- IFX_LONGID environment variable 3-51
- IFX_NETBUF_PVTPOOL_SIZE environment variable 3-52
- IFX_NETBUF_SIZE environment variable 3-52
- IFX_NO_TIMELIMIT_WARNING environment variable 3-52
- IFX_PAD_VARCHAR environment variable 3-53
- IFX_UPDESC environment variable 3-53
- IFX_XASTDCOMPLIANCE_XAEND environment variable 3-53
- imcadmin administrative tool 3-54
- IMCADMIN environment variable 3-54
- IMCONFIG environment variable 3-55
- IMCSERVER environment variable 3-55
- IMPEX data type 2-54
- IMPEXBIN data type 2-54
- Implicit cast 1-17, 2-53
- Implicit connection 3-60
- Implicit temporary tables 3-37
- import_binary() support function 2-47
- import() support function 2-47
- IN clause 3-37
- IN keyword 1-34, 2-7, 2-25, 2-30, 2-33, 2-35, 2-57
- IN TABLE storage option 3-42

Index

- attached 1-36, 3-25, 3-42, 3-76
- B-tree 1-37, 3-42
- clustered 1-36, 1-38
- composite 1-36, 1-37
- default values for attached 3-76
- descending 1-37
- detached 3-42
- distribution scheme 3-42
- fragmented 1-33

Index (continued)

- functional 1-37, 2-48
 - generalized-key 1-36, 1-40, 1-49
 - globally detached 1-35, 3-45
 - nonfragmented 3-42
 - of data types 2-3
 - of environment variables 3-81
 - of system catalog 1-10
 - R-tree 3-42
 - sysindexes data 1-36
 - sysindices data 1-38
 - sysobjstate data 1-41
 - threads for sorting 3-76
 - unique 1-26, 1-36, 2-30, 2-31
- Index privilege 1-53
- Indexkey structure 1-38
- Indirect typing 2-30, 2-31
- Industry standards
- See Compliance.
- Industry standards, compliance with xxiv, 1-66
- INF_ROLE_SEP environment variable 3-63
- Information Schema views
- accessing 1-64
 - columns 1-64
 - defined 1-63
 - generating 1-63
 - server_info 1-66
 - sql_languages 1-66
 - tables 1-64
- Informational messages 1-30
- Informix Dynamic Server documentation set xxi
- Informix extension checking (DBANSIWARN) 3-21
- informix owner name 1-9, 1-17, 1-29, 1-36, 1-38, 1-54, 3-30, 3-63
- Informix subkey 3-10
- informix.rc file 3-6, 3-10, 3-49
- INFORMIXC environment variable 3-55
- INFORMIXCONCSMCFG environment variable 3-56
- INFORMIXCONRETRY environment variable 3-56
- INFORMIXCONTIME environment variable 3-57
- INFORMIXCPPMAP environment variable 3-58
- INFORMIXDIR environment variable 3-58
- INFORMIXDIR/bin directory xi
- INFORMIXKEYTAB environment variable 3-59
- INFORMIXOPCACHE environment variable 3-59
- INFORMIXSERVER environment variable 3-60
- INFORMIXSHMBASE environment variable 3-60
- INFORMIXSTACKSIZE environment variable 3-62
- INFORMIXTERM environment variable 3-62
- Inheritance hierarchy 1-39, 2-29
- Initialization function 1-12, 1-50
- Input support function 2-23
- input() support function 2-47
- Insert privilege 1-33, 1-53, 3-68
- INSERT statement 1-56, 1-60, 2-14, 2-48, 3-21, 3-27

- Insert trigger 1-58
- Installation directory 3-59
- Installation Guides xviii
- INSTEAD OF trigger 1-58
- INT data type.
 - See INTEGER data type.
- INT8 data type
 - built-in casts 2-52
 - defined 2-19
 - using with SERIAL8 2-32
- INTEG keyword 2-40
- INTEGER data type
 - built-in casts 2-52
 - defined 2-19
 - length (syscolumns) 1-24
- Intensity attributes 3-62
- INTERACTIVE_DESKTOP_OFF environment variable 3-64
- Internationalized trace messages 1-56
- Interprocess communications (IPC) 3-60
- INTERVAL data type
 - defined 2-19
 - field delimiters 2-21
 - in expressions 2-40, 2-41, 2-45, 2-46
 - length (syscolumns) 1-24
- ipcshm protocol 3-60
- IS NULL operator 2-7, 2-35
- ISM_COMPRESSION environment variable 3-64
- ISM_DEBUG_FILE environment variable 3-64
- ISM_DEBUG_LEVEL environment variable 3-65
- ISM_ENCRYPTION environment variable 3-65
- ISM_MAXLOGSIZE environment variable 3-65
- ISM_MAXLOGVERS environment variable 3-66
- ISO 8859-1 code set xi, 1-67
- Isolation level 1-67, 3-69
- items table in stores_demo database A-3
- items table in superstores_demo database B-10
- Iterator function 1-12

J

- Japanese eras 3-41
- Jar management procedures 3-66
- JAR_TEMP_PATH environment variable 3-66
- Java virtual machine (JVM) 3-18, 3-66
- JAVA_COMPILER environment variable 3-66
- JIT compiler 3-66
- Join columns A-6, B-19
- Join methods 3-69
- Join operations 1-9, 3-37
- Join, Cartesian 3-46
- JVM_MAX_HEAP_SIZE environment variable 3-66

K

- KEEP ACCESS TIME keywords
 - ALTER TABLE statement 2-40

- KEEP ACCESS TIME keywords (*continued*)
 - CREATE TABLE statement 2-40

Key

- foreign A-8, B-3
- generalized 1-40, 1-49
- primary 1-26, 1-49, 1-60, A-8, B-6

- Key scan 1-13
- Key tables 3-59
- Keyboard I/O
 - INFORMIXTERM setting 3-62
 - TERM setting 3-78
 - TERMCAP setting 3-78
 - TERMINFO setting 3-79

- keytab file 3-59

Keywords

- in syntax diagrams xvi
- Korn shell 3-6, 3-7

L

Language

- See also Locale.
- C 1-50, 3-20, 3-55
- C++ 3-58
- CLIENT_LOCALE setting 3-10, 3-27
- DBLANG setting 3-29
- Extensible Markup Language (XML) 2-10
- Hypertext Markup Language (HTML) 2-10
- Informix ESQL/C 2-40, 2-48, 3-80
- Java 3-18, 3-66
- sql_languages information schema view 1-66
- Stored Procedure Language (SPL) 2-48, 3-24, 3-27
- syslangauth data 1-39
- sysroutinelangs data 1-50

Large-object data type

- defined 2-38
- listed 2-38

- LD_LIBRARY_PATH environment variable 3-67

- Leaf pages 1-35

- LIBERAL_MATCH environment variable 3-67

- libos.a library 3-51

- LIBPATH environment variable 3-68

- LIKE keyword of SPL 2-30, 2-31

- LIKE operator 2-7, 2-35, 2-57, 3-68

- Linearized code 1-57

List

- of data types 2-3
- of environment variables 3-14
- of environment variables, by topic 3-81
- of system catalog tables 1-10

- LIST data type, defined 2-22

- LOAD statement 2-6, 2-8, 2-35, 3-28

Locale x

- collation order 1-56, 2-38
- default xi
- en_us.8859-1 xi

- Locale (*continued*)
 - multibyte 2-9
 - of trace messages 1-57
 - right-to-left 2-8
 - specifying 3-82, 3-86
- Lock-table overflow 3-49
- LOCKMODE keyword 3-48
- LOCOPY function 2-6, 2-10
- LOG keyword
 - ALTER TABLE statement 2-40
 - CREATE TABLE statement 2-40
- Logging mode 1-19
- Long identifiers
 - client version 3-51
 - IFX_LONGID setting 3-51
 - Information Schema views 1-64
- LOTOFILE function 2-6, 2-10
- LOW keyword
 - PDQPRIORITY 3-73
 - UPDATE STATISTICS 1-29
- Lowercase mode codes 1-47
- Lowercase privilege codes 1-7, 1-20, 1-33, 1-53, 1-61
- LVARCHAR data type
 - casting opaque types 2-54
 - coltype code 1-23
 - defined 2-23

M

- Machine notes xix, 3-63
- Magnetic storage media 1-17
- Mantissa precision 1-65, 2-16
- manufact table in superstores_demo database B-12
- Map file for C++ programs 3-58
- MATCHES operator 2-7, 2-35, 2-38, 2-57, 3-68
- MaxConnect 3-54, 3-55
- MEDIUM keyword 1-10, 1-25, 1-29
- Membership operator 2-57
- Memory cache, for staging blobspace 3-59
- Message file
 - specifying subdirectory with DBLANG 3-30
 - XBSA 3-64
- Messages
 - chaining 3-70
 - error in syserrors 1-30
 - optimized transfers 3-70
 - reducing requests 3-71
 - trace message template 1-57
 - warning in syserrors 1-30
- mi_collection_card() function 2-23, 2-25, 2-33
- mi_db_error_raise() function 1-30
- Microsoft C compiler 3-55
- MINUTE keyword
 - DATETIME qualifier 2-12
 - INTERVAL qualifier 2-20
- MITRACE_OFF configuration parameter 1-56, 1-57

- mkdir utility 3-30
- MODERATE INTEG keywords
 - ALTER TABLE statement 2-40
 - CREATE TABLE statement 2-40
- Modifiers
 - CLASS 1-46
 - COSTFUNC 1-47
 - HANDLESNULLS 1-46
 - INTERNAL 1-46
 - NEGATOR 1-46
 - NOT VARIANT 1-46
 - PARALLELIZABLE 1-47
 - SELCONST 1-46
 - STACK 1-47
 - VARIANT 1-46
- MODIFY NEXT SIZE keywords 1-9
- MONEY data type
 - built-in casts 2-52
 - defined 2-24
 - display format 3-30
 - international money formats 2-25
 - length (syscolumnms) 1-25
- MONTH keyword
 - DATETIME qualifier 2-12
 - INTERVAL qualifier 2-20
- Multibyte characters
 - CLOB data type 2-10
 - VARCHAR data type 2-37
- MULTISET data type
 - constructor 2-48
 - defined 2-25

N

- N setting
 - sysroleauth.is_grantable 1-50
- Named ROW data type
 - See also* ROW type.
 - casting permitted 2-56
 - defined 2-27
 - defining 2-27
 - equivalence 2-28
 - inheritance 1-39, 2-28
 - typed tables 2-28
- NCHAR data type
 - collation order 2-26
 - conversion to CHAR 3-32
 - defined 2-26
 - multibyte characters 2-26
- Negator function 1-46
- Nested dot notation 2-48
- Nested-loop join 3-69
- Network buffers 3-52
- Network environment variable, DBPATH 3-33
- NFS directory 3-38

- NO KEEP ACCESS TIME keywords
 - ALTER TABLE statement 2-40
 - CREATE TABLE statement 2-40
- no setting of NODEFDAC 3-68
- NODE configuration parameter 3-69
- NODEFDAC environment variable 3-68
- NOLOG keyword
 - ALTER TABLE statement 2-40
 - CREATE TABLE statement 2-40
- NONE setting
 - ISM_ENCRYPTION 3-65
 - JAVA_COMPILER 3-66
- Nonprintable characters
 - CHAR data type 2-9
 - TEXT data type 2-35
 - VARCHAR data type 2-37
- NOT NULL constraint
 - collection elements 2-23, 2-25, 2-33, 2-48
 - syscoldepend data 1-20
 - sysconstraints data 1-26
- NOT NULL keywords 2-7, 2-22, 2-35
- NOT operator 2-57
- NULL value
 - allowed or not allowed 1-12, 1-23
 - BOOLEAN literal 2-7
 - BYTE data type 2-7
 - TEXT data type 2-35
- NUMERIC data type.
 - See DECIMAL data type.
- Numeric data types
 - casting between 2-51
 - casting to character types 2-52
 - listed 2-38
- NVARCHAR data type
 - collation order 2-26
 - conversion to VARCHAR 3-33
 - defined 2-26
 - length (syscolumns) 1-24
 - multibyte characters 2-26

O

- Object mode of database objects 1-41
- Object-relational schema B-1
- ODBC driver 3-67, 3-77
- OFF setting
 - IFX_DIRECTIVES 3-49, 3-50
 - PDQPRIORITY 3-73
- ON setting
 - IFX_DIRECTIVES 3-49, 3-50
- ON-Bar 3-65
- ONCONFIG environment variable 3-68, 3-69
- onconfig.std file 3-69, 3-78
- onconfig.xps file 3-69
- oninit command 3-49
- Online help xxi

- Online manuals xxi
- Online notes xviii, xix
- Online transaction processing (OLTP) 1-35
- onload utility 2-6, 2-8, 2-35
- onpload utility 3-33, 3-74
- onstat utility 3-4, 3-46
- onutils utility 3-45
- Opaque data type
 - cast matrix 2-56
 - comparing 2-54
 - defined 2-26
 - smart large objects 2-39
 - storage 2-23
 - sysxtddesc data 1-61
 - sysxtdtypes data 1-61
- OPCACHEMAX configuration parameter 3-59
- OPEN statement 3-70
- Operator class
 - sysams data 1-13
 - sysindices data 1-39
 - sysopclasses data 1-41
- Operator precedence 2-56
- OPT_GOAL configuration parameter 3-71
- OPT_GOAL environment variable 3-71
- OPTCOMPIND configuration parameter 3-70
- OPTCOMPIND environment variable 3-69
- Optical cluster
 - INFORMIXOPCACHE setting 3-59
 - sysblobs.type 1-17
 - sysopclstr data 1-42
- Optimizer
 - setting IFX_DIRECTIVES 3-49
 - setting IFX_EXTDIRECTIVES 3-50
 - setting OPT_GOAL 3-71
 - setting OPTCOMPIND 3-70
 - setting OPTOFC 3-71
 - sysdistrib data 1-29
- Optimizer directives
 - sysdirectives data 1-28
- OPTMSG environment variable 3-70
- OPTOFC environment variable 3-70
- OR operator 2-57
- ORDER BY clause 2-7, 2-35, 3-37
- orders table in superstores_demo database B-11, B-12, B-13
- Ordinal positions 2-22
- Output support function 2-23
- output() support function 2-47
- Overflow error 2-16
- Owner routines 1-47, 3-68

P

- PAGE lock mode 1-54, 3-49
- Parallel database query.
 - See PDQ.

- Parallel distributed queries, setting with
 - PDQPRIORITY 3-72
- Parallel sorting, setting with PSORT_NPROCS 3-75
- Partial characters 2-8
- PATH environment variable 3-71, 3-72
- Pathname
 - for C compiler 3-55
 - for C++ map file 3-58
 - for client or shared libraries 3-67
 - for conccsm.cfg file 3-56
 - for connectivity information 3-61
 - for database server 3-33
 - for dynamic-link libraries 3-68, 3-77
 - for environment-configuration file 3-9
 - for executable programs 3-71
 - for installation 3-58
 - for keytab file 3-59
 - for message files 3-29
 - for parallel sorting 3-75
 - for remote shell 3-36
 - for smart-large-object handles 3-74
 - for temporary .jar files 3-66
 - for termcap file 3-78
 - for terminfo directory 3-79
 - for XBSA messages 3-64
 - for xfer_config file 3-81
 - separator symbols 3-72
- PDQ
 - OPTCOMPIND environment variable 3-69
 - PDQPRIORITY environment variable 3-72
- PDQPRIORITY configuration parameter 3-73
- Percentage (%) symbol 3-39
- Period
 - DATE delimiter 3-26
 - DATETIME delimiter 2-13
 - INTERVAL delimiter 2-21
- Permissions 3-6, 3-30
- PLCONFIG environment variable 3-74
- plconfig file 3-74
- PLOAD_LO_PATH environment variable 3-74
- PLOAD_SHMBASE environment variable 3-74
- PostScript 2-10
- Precedence rules
 - for casts 2-54
 - for lock mode 3-49
 - for native Windows application 3-14
 - for SQL operators 2-56
 - for UNIX environment variables 3-9
 - for Windows environment variables 3-13
- Precision
 - of currency values 2-24
 - of numbers 1-65, 2-15, 2-18, 2-19, 2-34
 - of time values 2-11, 2-20, 2-42, 2-45
- PREPARE statement 1-55
- Prepared statement 1-55
- Primary access method 1-13, 1-52
- Primary key 1-26, 1-49, 1-60, 2-30, 2-31, A-2, B-6
- Primary thread 3-62
- Printed manuals xxi
- printenv utility 3-8
- Printing with DBPRINT 3-35
- Private environment-configuration file 3-9, 3-44
- Private network buffer pool 3-52
- Private synonym 1-54
- Privilege
 - default table privileges 3-68
 - on columns (syscolauth table) 1-20
 - on procedures and functions (sysprocauth table) 1-44
 - on table fragments (sysfragauth table) 1-33
 - on tables (systabauth table) 1-53
 - on the database (sysusers table) 1-59
 - on UDTs and named row types (sysxdttypeauth) 1-61
- product table in sales_demo database B-3
- Protected routines 1-47
- Pseudo-machine code (p-code) 1-44
- PSORT_DBTEMP environment variable 3-75
- PSORT_NPROCS environment variable 3-76
- Public synonym 1-51, 1-54
- public user name 1-64
- Purpose functions 1-13
- Push-down hash join
 - dimension tables 3-48
 - fact tables 3-47
- putenv utility 3-5

Q

- Qualifier field
 - DATETIME 2-12
 - EXTEND 2-45
 - INTERVAL 2-20
 - UNITS 2-44
- Query optimizer
 - defined 1-9
 - directives 3-49, 3-50
 - push-down hash-join plans 3-47, 3-48
 - sysprocplan data 1-48
- Quoted string
 - DATE and DATETIME literals 2-44
 - DELIMIDENT setting 3-43
 - INTERVAL literals 2-21
 - invalid with BYTE 2-8
 - invalid with TEXT 2-35
 - LVARCHAR data type 2-23

R

- R-tree index 3-42, 3-77
- Raw UNIX devices 3-37
- Read committed 1-67

- Read uncommitted 1-67
- REAL data type.
 - See SMALLFLOAT data type.
- recv() support function 2-47
- References privilege 1-20, 1-53
- Referential constraint 1-26, 1-49, 1-60, A-8, B-19
- regedt32.exe Registry Editor 3-10
- region table in superstores_demo database B-13
- Registry Editor 3-10
- Reject file 1-32
- Relational operators 2-9, 2-57
- Release Notes xix
- Remote database server 3-20, 3-44
- Remote shell 3-36
- Remote tape devices 3-36
- RENAME SEQUENCE statement 3-80
- Repeatable read 3-69
- Replica identifier 1-34
- Resource contention 3-73
- Resource Grant Manager (RGM) 1-35
- Resource privilege 1-8, 1-59
- REVOKE statement 1-53
- Right-to-left locales 2-8
- Role
 - default role 1-59
 - INF_ROLE_SEP setting 3-63
 - sysroleauth data 1-50
 - sysusers data 1-59
- Role separation 3-63
- Round-robin fragmentation 1-34
- Routine
 - See also User-defined routine.
 - DataBlade API routine 1-56
 - DATETIME formatting 3-39
 - identifier 1-45
 - owner 1-45
 - privileges 1-44
 - protected 1-47
 - Stored Procedure Language (SPL) 2-48
 - syserrors data 1-30
 - syslangauth data 1-39
 - sysprocauth data 1-44
 - sysprocbody data 1-44
 - sysprocedures data 1-45
 - sysprocplan data 1-48
 - sysroutinelangs data 1-50
 - systraceclasses data 1-56
 - systracemsgs data 1-56
- ROW lock mode 1-54, 3-49
- ROW type 2-48
 - casting permitted 2-56
 - equivalence 2-28
 - fields 1-15, 2-48
 - inheritance 1-39, 2-28
 - inserting values 2-30

- ROW type (*continued*)
 - named 2-27, 2-49
 - sysattrtypes data 1-15
 - sysxtddesc data 1-61
 - sysxtdtype data 1-61
 - unnamed 2-28, 2-49
- Rowids 1-13
- RTNPARAMTYPES data type 1-46
- RTREE_COST_ADJUST_VALUE environment variable 3-77
- Runtime
 - warnings (DBANSIWARN) 3-22

S

- sales table in sales_demo database B-4
- sales_demo database
 - customer table columns B-3
 - defined B-3
 - geography table columns B-3
 - product table columns B-3
 - sales table columns B-4
 - time table columns B-4
- sales_rep table in superstores_demo database B-13
- Sample-code conventions xvii
- SAVE EXTERNAL DIRECTIVES statement 3-51
- sbspace
 - defined 2-10, 2-40
 - name 3-43
 - sysams data 1-13
 - syscolattrs data 1-19
 - sysstabamdata data 1-52
- Scale of numbers 1-65, 2-16, 3-29
- Scan cost 1-14
- Schema Tools 3-5
- Screen reader
 - reading syntax diagrams C-1
- SECOND keyword
 - DATETIME qualifier 2-12
 - INTERVAL qualifier 2-20
- Secondary access method 1-13, 1-26, 1-39, 1-41, 2-27
- SELECT INTO TEMP statement 3-37
- Select privilege 1-20, 1-53, 1-64, 3-68
- SELECT statement 1-9, 1-28
- Select trigger 1-58
- Selectivity constant 1-47
- Self-join 1-6
- send() support function 2-47
- SENDRECV data type 2-54
- Sequence
 - syssequences data 1-51
 - sysynonyms data 1-51
 - sysyntable data 1-51
 - sysstabauth data 1-53
 - sysables data 1-54

- Sequential integers
 - aggid code 1-12
 - am_id code 1-13
 - classid code 1-56
 - constrid code 1-26
 - extended_id code 1-62
 - id code 1-28, 1-30
 - langid code 1-50
 - msgid code 1-57
 - opclassid code 1-42
 - planid code 1-48
 - procid code 1-45, 1-46
 - seqid code 1-51
 - SERIAL data type 2-30
 - SERIAL8 data type 2-31
 - tabid code 1-5, 1-51, 1-54
 - trigid code 1-58
 - udr_id code 1-18
- SERIAL data type
 - defined 2-30
 - inserting values 2-31
 - length (syscolumns) 1-24
 - resetting values 2-31
- SERIAL8 data type
 - assigning a starting value 2-31
 - defined 2-31
 - inserting values 2-32
 - length (syscolumns) 1-24
 - resetting values 2-32
 - using with INT8 2-32
- Serializable transactions 1-67
- SET ALL_MUTABLES statement 3-74
- SET data type, defined 2-32
- SET ENVIRONMENT statement 3-4, 3-5, 3-70, 3-74
- SET OPTIMIZATION statement 3-71
- SET PDQPRIORITY statement 3-73
- SET SESSION AUTHORIZATION statement 1-47
- SET STMT_CACHE statement 3-78
- SET TEMP TABLE_SPACE statement 3-37
- set utility 3-12
- setenv utility 3-8
- Setnet32 utility 3-5
- Setting environment variables
 - in UNIX 3-6
 - in Windows 3-10
- SGML (Standard Graphic Markup Language) 2-10
- Shared environment-configuration file 3-9
- Shared libraries 3-52
- Shared memory
 - INFORMIXSHMBASE 3-60
 - PLOAD_SHMBASE 3-74
- Shell
 - remote 3-36
 - search path 3-71
 - setting environment variables in a file 3-6
- Shell (*continued*)
 - specifying with DBREMOTECMD 3-36
 - SHLIB_PATH environment variable 3-77
- Simple large objects
 - defined 2-39
 - location (sysblobs) 1-16
- Single-precision floating-point number 2-27, 2-34
- SMALLFLOAT data type
 - built-in casts 2-52
 - defined 2-34
 - display format 3-29, 3-30
- SMALLINT data type
 - built-in casts 2-52
 - defined 2-34
 - length (syscolumns) 1-24
- Smart large objects
 - See also* sbspacee.
 - defined 2-39
 - syscolattribs data 1-19
- Smart-large-object handles 3-74
- Software dependencies x
- SOFTWARE registry key 3-10
- SOME operator 2-57
- Sort-merge join 3-69
- Sorting
 - See also* Collation.
 - DBSPACETEMP environment variable 3-36
 - PSORT_DBTEMP environment variable 3-75
 - PSORT_NPROCS environment variable 3-76
- Space
 - DATETIME delimiter 2-13
 - INTERVAL delimiter 2-21
- Spatial queries 3-77
- SPL routine 1-45, 2-48, 3-24, 3-27
- SPL variables 2-48
- SQL (Structured Query Language) 3-22
- SQL character set 3-43
- SQL code xvii
- SQL Communication Area (SQLCA) 3-22
- sqlhosts file 3-54, 3-60, 3-61
- SQLHOSTS subkey 3-61
- SQLSTATE value 1-30
- sqlwarn array 3-22
- Stack size 1-47, 3-62
- STACKSIZE configuration parameter 3-62
- Staging area blob space 3-59
- Standard Graphic Markup Language (SGML) 2-10
- START DATABASE statement 3-33
- STAT data type 1-29
- state table in stores_demo database A-5
- state table in superstores_demo database B-14
- Statement cache 3-77
- Statements of SQL
 - ALTER OPTICAL CLUSTER 1-43
 - ALTER SEQUENCE 3-80

Statements of SQL (*continued*)

ALTER TABLE 1-9, 1-55, 3-80
CLOSE 3-70
CONNECT 3-33, 3-34, 3-57, 3-60
CREATE ACCESS METHOD 1-12
CREATE AGGREGATE 1-12
CREATE CAST 1-17, 2-53
CREATE DATABASE 3-33
CREATE DISTINCT TYPE 1-61, 2-18, B-17
CREATE EXTERNAL TABLE 1-31, 1-32
CREATE FUNCTION 1-50, 3-68
CREATE IMPLICIT CAST B-17
CREATE INDEX 1-4, 1-36, 1-38, 1-40, 1-50, 1-55, 3-42
CREATE OPAQUE TYPE 1-61, 2-27
CREATE OPERATOR CLASS 1-41
CREATE OPTICAL CLUSTER 1-42, 1-43
CREATE PROCEDURE 1-44, 1-50
CREATE ROLE 1-50, 1-59
CREATE ROUTINE FROM 1-50, 1-51
CREATE ROW TYPE 1-61, 2-28
CREATE SCHEMA AUTHORIZATION 1-3
CREATE SEQUENCE 1-51
CREATE SYNONYM 1-51
CREATE TABLE 1-27, 1-49, 1-52
CREATE TRIGGER 1-58
CREATE VIEW 1-59
DATABASE 3-34
DECLARE 3-70
DELETE 1-9, 1-48, 1-60
DESCRIBE 3-53
DROP CAST B-17
DROP DATABASE 3-34
DROP FUNCTION 1-47
DROP INDEX 1-55
DROP OPTICAL CLUSTER 1-43
DROP PROCEDURE 1-47
DROP ROUTINE 1-47
DROP ROW TYPE 2-28
DROP SEQUENCE 3-80
DROP TYPE 2-18, 2-27
DROP VIEW 1-63, 3-80
FETCH 3-70
GET DIAGNOSTICS 1-30
GRANT 1-33, 1-50, 1-53, 1-64
INSERT 1-60, 2-48, 3-21, 3-27
LOAD 2-8, 2-35, 3-21, 3-28
OPEN 3-70
PREPARE 1-55
RENAME SEQUENCE 3-80
RENAME TABLE 3-80
REVOKE 1-53, 1-59
SELECT 1-9, 1-28, 1-48, 3-37
SET ALL_MUTABLES 3-74
SET ENVIRONMENT 3-70, 3-74

Statements of SQL (*continued*)

SET ENVIRONMENT CLIENT_TZ 3-45, 3-46
SET OPTIMIZATION 3-71
SET PDQPRIORITY 3-73
SET SESSION AUTHORIZATION 1-47
SET STMT_CACHE 3-78
SET TEMP TABLE_SPACE 3-37
START DATABASE 3-34
UNLOAD 3-22, 3-28
UPDATE 2-7, 2-35, 3-21
UPDATE STATISTICS 1-9, 3-42
UPDATE STATISTICS FOR PROCEDURE 1-48

Statements of SQL

DROP TABLE 3-80

static option of ESQL/C 3-51

STMT_CACHE configuration parameter 3-78

STMT_CACHE environment variable 3-77

STMT_CACHE keyword 3-78

stock table in stores_demo database A-3

stock table in superstores_demo database B-14

stock_discount table in superstores_demo database B-15

Storage identifiers 3-43

Stored procedure language (SPL) 1-45, 2-48, 3-24

stores_demo database xi

call_type table columns A-5

catalog table columns A-4

cust_calls table columns A-4

customer table columns A-2

data values A-13

defined A-1

items table columns A-3

join columns A-7

manufact table columns A-5

primary-foreign key relationships A-8

stock table columns A-3

structure of tables A-1

String Editor dialog box 3-11

strings option of gcc 3-56

Structured Query Language (SQL) 3-21

See Statements of SQL.

Subscripts 2-8, 2-35

SUBSTRING function 1-9

Subtable 1-34, 1-39, B-11, B-18

Subtype 1-39, 2-28

Summary

of data types 2-3

of environment variables, by topic 3-81

of environment variables, by type of server 3-14

of system catalog tables, by type of server 1-10

superstores_demo database xi

call_type table columns B-6

catalog table columns B-6

cust_calls table columns B-7

customer table columns B-8, B-9

- superstores_demo database *(continued)*
 - defined B-5
 - items table columns B-10
 - manufact table columns B-12
 - orders table columns B-11, B-12, B-13
 - primary-foreign key relationships B-19, B-21
 - sales_rep table columns B-13
 - stock table columns B-14
 - stock_discount table columns B-15
 - structure of tables B-5
- Supertable 1-39, B-11, B-18
- Supertype 1-39, 2-28
- Support function
 - DISTINCT data types 2-49
 - OPAQUE data types 2-27, 2-47
 - routine identifier 1-45
- Symbol table 1-45, 1-46
- Synonym
 - sys synonyms data 1-51
 - sys syntable data 1-51
 - sys tables data 1-54
 - USETABLENAME setting 3-80
- Syntax diagrams
 - conventions for xiii
 - keywords in xvi
 - reading in a screen reader C-1
 - variables in xvi
- Syntax segment xv
- sysaggregates system catalog table 1-12
- sysams system catalog table 1-12
- sysattrtypes system catalog table 1-15
- sysblobs system catalog table 1-16
- sysbuiltintypes table 1-3
- syscasts system catalog table 1-17, 2-50
- syschecks system catalog table 1-18
- syscheckudrdep system catalog table 1-18
- syscolattrs system catalog table 1-19
- syscolauth system catalog table 1-20
- syscoldepend system catalog table 1-20
- syscolumns system catalog table 1-22
- sysconstraints system catalog table 1-26
- syscrd database 1-3
- sysdbclose() routine 3-5
- sysdbopen() routine 3-5
- sysdefaults system catalog table 1-27
- sysdepend system catalog table 1-27
- sysdirectives system catalog table 1-28
- sysdistrib system catalog table 1-28
- sysdomains system catalog table 1-30
- syserrors system catalog table 1-30
- sysextcols system catalog table 1-31
- sysextdfiles system catalog table 1-31
- sysexternal system catalog table 1-32
- sysfragauth system catalog table 1-33
- sysfragments system catalog table 1-33
- sysindexes system catalog table 1-36
- sysindices system catalog table 1-38
- sysinherits system catalog table 1-39
- syslangauth system catalog table 1-39
- syslogmap system catalog table 1-40
- sysmaster database 1-3
 - initialization 3-4, 3-60
 - versus system catalog tables 1-3
- sysnewdepend system catalog table 1-40
- sysobjstate system catalog table 1-41
- sysopclasses system catalog table 1-41
- sysopclstr system catalog table 1-42
- sysprocauth system catalog table 1-44
- sysprocbody system catalog table 1-44
- sysprocedures system catalog table 1-45
- sysprocplan system catalog table 1-48
- sysreferences system catalog table 1-49
- sysrepository system catalog table 1-49
- sysroleauth system catalog table 1-50
- sysroutinelangs system catalog table 1-50
- syssequences system catalog table 1-51
- sys synonyms system catalog table 1-51
- sys syntable system catalog table 1-51
- sysstabamdata system catalog table 1-52
- sysstabauth system catalog table 1-53
- sys tables system catalog table 1-54
- System administrator (DBA) 1-3
- System applet 3-11
- System catalog
 - sysaggregates 1-12
 - sysams 1-12
 - sysattrtypes 1-15
 - sysblobs 1-16
 - syscasts 1-17
 - syschecks 1-18
 - syscheckudrdep 1-18
 - syscolattrs 1-19
 - syscolauth 1-20
 - syscoldepend 1-20
 - syscolumns 1-22
 - sysconstraints 1-26
 - sysdefaults 1-27
 - sysdepend 1-27
 - sysdirectives 1-28
 - sysdistrib 1-28
 - sysdomains 1-30
 - syserrors 1-30
 - sysextcols 1-31
 - sysextdfiles 1-31
 - sysexternal 1-32
 - sysfragauth 1-33
 - sysfragments 1-33
 - sysindexes 1-36
 - sysindices 1-38
 - sysinherits 1-39

System catalog *(continued)*

- syslangauth 1-39
- syslogmap 1-40
- sysnewdepend 1-40
- sysobjstate 1-41
- sysopclasses 1-41
- sysopclstr 1-42
- sysprocauth 1-44
- sysprobody 1-44
- sysprocedures 1-45
- sysproclan 1-48
- sysreferences 1-49
- sysrepository 1-49
- sysroleauth 1-50
- sysroutinelangs 1-50
- syssequences 1-51
- syssynonyms 1-51
- syssytable 1-51
- systabamdata 1-52
- systabauth 1-53
- systables 1-54
- systraceclasses 1-56
- systracemsgs 1-56
- systrigbody 1-57
- systriggers 1-58
- sysusers 1-59
- sysviews 1-59
- sysviolations 1-60
- sysxtddesc 1-61
- sysxtdtypeauth 1-61
- sysxtdtypes 1-61

System catalog tables

See also System catalog.

- access methods 1-12, 1-52
- accessing 1-9
- altering contents 1-9
- authorization identifiers 1-59
- casts 1-17
- columns 1-22
- complex data types 1-15, 1-61
- constraint violations 1-60
- constraints 1-18, 1-20, 1-26
- data distributions 1-28
- database tables 1-54
- default values 1-27
- defined 1-2
- dependencies 1-27, 1-40
- example
 - syscolauth 1-7
 - syscolumns 1-5
 - sysindexes 1-7
 - systabauth 1-6
 - systables 1-4
- external tables 1-31, 1-32
- fragmentation 1-33

System catalog tables *(continued)*

- indexes 1-36, 1-38, 1-50
- inheritance 1-39
- list of tables 1-10
- messages 1-30, 1-56
- operator classes 1-41
- optical clusters 1-42
- privileges 1-20, 1-33, 1-53, 1-59, 1-61
- programming languages 1-39, 1-50
- referential constraints 1-26, 1-49, 1-60
- roles 1-50
- routines 1-44, 1-45, 1-48
- sequence objects 1-51
- simple large objects 1-16
- smart large objects 1-19
- synonyms 1-51, 1-52
- trace classes 1-56
- trace messages 1-56
- triggers 1-57, 1-58
- updating 1-9, 1-10
- use by database server 1-4
- user-defined aggregates 1-12
- user-defined data types 1-61
- views 1-54, 1-59

System control panel 3-13

System environment variable 3-13

System requirements

- database x
- software x

SYSTEM() command, on NT 3-64

systraceclasses system catalog table 1-56

systracemsgs system catalog table 1-56

systrigbody system catalog table 1-57

systriggers system catalog table 1-58

sysusers system catalog table 1-59

sysutils database 1-3

sysuuid database 1-3

sysviews system catalog table 1-59

sysviolations systems catalog table 1-60

sysxtddesc system catalog table 1-61

sysxtdtypeauth system catalog table 1-61

sysxtdtypes system catalog table 1-61, 2-27

T

tabid 1-5, 1-55

Table

- changing a column data type 2-50
- dependencies, in sysdepend 1-27
- fragmented 1-33
- hashing parameters 1-52
- hierarchy 1-34, 1-39, 2-28, B-18
- inheritance, sysinherits data 1-39
- lock mode 3-48
- nonfragmented 3-42
- separate from large object storage 2-38

Table (*continued*)

- structure in superstores_demo database B-5
- synonyms in sysssyntax 1-51
- system catalog tables 1-12, 1-61
- temporary 3-37, 3-38
- temporary in SE 3-38
- typed, and named ROW type 2-28
- untyped, and unnamed ROW 2-30

Table-based fragmentation 1-34

Table-level privilege

- PUBLIC 1-64
- sysfragauth data 1-33
- sysstabauth data 1-7, 1-53

Tape management

- setting DBREMOTECMD 3-36

Temporary dbspace 3-36

Temporary files 3-38

- in SE, specifying directory with DBTEMP 3-38
- setting DBSPACETEMP 3-36
- setting PSORT_DBTEMP 3-75

Temporary tables 3-36

- in SE, specifying directory with DBTEMP 3-38
- specifying dbspace with DBSPACETEMP 3-36

TERM environment variable 3-78

TERMCAP environment variable 3-78

termcap file

- setting INFORMIXTERM 3-62
- setting TERMCAP 3-78

Terminal handling

- setting INFORMIXTERM 3-62
- setting TERM 3-78
- setting TERMCAP 3-78
- setting TERMINFO 3-79

terminfo directory 3-63, 3-79

TERMINFO environment variable 3-79

TEXT data type

- casting to CLOB 2-35
- collation 2-35
- defined 2-34
- increasing buffer size 3-22
- inserting values 2-35
- length (syscolumns) 1-25
- nonprintable characters 2-35
- queries 2-35
- restrictions
 - in Boolean expression 2-35
 - with GROUP BY 2-35
 - with LIKE or MATCHES 2-35
 - with ORDER BY 2-35
- setting buffer size 3-22
- sysblobs data 1-16
- sysfragments data 1-34
- with control characters 2-35

Text editor 3-28

Thousands separator 2-24

thread flag of ESQL/C 3-80

THREADLIB environment variable 3-79

Time data types

- arithmetic 2-41
- length (syscolumns) 1-24
- listed 2-38

time table in sales_demo database B-4

Time values

- DBCENTURY setting 3-22
- DBDATE setting 3-25
- DBTIME setting 3-39
- GL_DATETIME settings 3-41
- USEOSTIME parameter 2-15

Time zone, specifying 3-45, 3-46

Time-limited licenses

- (IFX_NO_TIMELIMIT_WARNING) 3-52

TO keyword

- DATETIME qualifier 2-12
- EXTEND function 2-43
- INTERVAL qualifier 2-20

TOBIGINT environment variable 3-80

TOC Notes xix

TODAY operator 1-27, 3-45

Trace class 1-56

Trace statements 1-57

Trailing blank spaces 3-67

Transaction isolation level 1-67, 3-69

Transaction logging 1-19, 1-67, B-1

Trigger

- creation-time value 3-24, 3-27
- sysobjstate data 1-41
- systrigbody data 1-57
- systriggers data 1-58

TRUE setting

- BOOLEAN values 2-7
- CPFIRST 3-20
- ISM_COMPRESSION 3-64
- ISM_ENCRYPTION 3-65
- sysams table 1-13, 1-14

Truncation 2-8

TYPE keyword 2-29

Typographical conventions xii

U

UDA

- See* User-defined aggregates.

UDR

- See* User-defined routine.

UDT

- See* User-defined data type.

UDT indexes 3-77

Unary arithmetic operators 2-57

Uncommitted read 1-67

Under privilege 1-53

Unique constraint 2-30, 2-31

- Unique index 1-36, 2-30
- Unique keys 1-13
- Unique numeric values
 - See also* Sequential integers.
 - SERIAL data type 2-31
 - SERIAL8 data type 2-31
- UNITS operator 2-11, 2-41, 2-44, 2-57
- units table in superstores_demo database B-16
- UNIX
 - BSD, default print utility 3-35
 - default locale for xi
 - environment variables 3-4
 - PATH environment variable 3-71
 - System V
 - default print utility 3-35
 - terminfo libraries 3-63, 3-79
 - temporary files 3-75
 - TERM environment variable 3-78
 - TERMCAP environment variable 3-78
 - TERMINFO environment variable 3-79
- UNLOAD statement 3-22, 3-28
- Unnamed ROW data type
 - See also* ROW type.
 - declaring 2-29
 - defined 2-28
 - inserting values 2-30
- unset utility 3-7
- unsetenv utility 3-7
- Unsetting an environment variable 3-7
- Untyped table 1-55
- Update privilege 1-20, 1-33, 1-53, 3-68
- UPDATE statement 1-60, 2-7, 2-35, 3-53
- UPDATE STATISTICS FOR PROCEDURE
 - statement 1-48
- UPDATE STATISTICS statement 1-38, 3-41
 - and DBUPSPACE environment variable 3-41
 - effect on sysdistrib table 1-29
 - sysindices (index statistics) 1-43
 - sysindices data 1-38
 - update system catalog 1-9
- Update trigger 1-58
- Uppercase mode codes 1-47
- Uppercase privilege codes 1-7, 1-20, 1-33, 1-53, 1-61
- Usage privilege 1-61
- USEOSTIME configuration parameter 2-15
- User environment variable 3-13
- User informix 1-9, 1-17, 2-50
- User name 1-67
- User privileges
 - syscolauth data 1-20
 - sysfragauth data 1-33
 - syslangauth data 1-39
 - sysprocauth data 1-44
 - sysstabauth data 1-53
 - sysusers data 1-59
- User privileges (*continued*)
 - sysxtdtypeauth data 1-61
- User-defined aggregates 1-12
- User-defined casts 1-17, 2-53
- User-defined data type
 - casting 2-53
 - casting into built-in type 2-50
 - opaque 2-49
 - sysxtddesc data 1-61
 - sysxtdtypes data 1-61
- User-defined routine
 - casts (syscasts) 1-17
 - check constraints (syscheckudrdep) 1-18
 - error messages (syserrors) 1-30
 - for opaque data types 2-27
 - functional index 3-42
 - language authorization (syslangauth) 1-39
 - privileges 1-44, 3-68
 - protected 1-47
 - secondary access method 1-26
 - sysprocedures data 1-45
- USETABLENAME environment variable 3-80
- Utility
 - archecker 3-18
 - chkenv 3-6, 3-9
 - DB-Access 1-9, 1-63, 3-5, 3-22, 3-29, 3-60, B-1
 - dbexport 3-28
 - dbload 2-6, 2-8, 2-35
 - dbschema 1-31, 1-32, 1-47, 3-80
 - dce_login 3-59
 - env 3-8
 - export 3-7
 - gcc 3-56
 - getenv 3-5
 - ifx_getenv 3-5
 - ifx_putenv 3-5
 - imcadmin 3-54
 - load 1-32
 - lp 3-35
 - lpr 3-35
 - MaxConnect 3-55
 - ON-Bar 3-64, 3-65
 - oninit 3-49
 - onload 2-6, 2-8, 2-35
 - onpload 3-33, 3-74
 - onstat 3-46
 - onutil 3-46
 - onutils 3-45
 - onxfer 3-81
 - printenv 3-8
 - putenv 3-5
 - regedt32.exe 3-10
 - set 3-12
 - setenv 3-8
 - Setnet32 3-5

Utility (*continued*)

- source 3-6
- unset 3-7
- unsetenv 3-7, 3-43
- vi 3-29

V

VARCHAR data type

- See also* CHARACTER VARYING data type.
- collation 2-37
- conversion to NVARCHAR 3-32, 3-33
- defined 2-36
- length (syscolumns) 1-24
- multibyte characters 2-37
- nonprintable characters 2-37
- storing numeric values 2-37

Variable-length packets 3-53

Variable-length UDT 1-62

Variables, in syntax diagrams xvi

VARIANT routine 1-46

vi text editor 3-29

View

- columns view 1-64
- Information Schema 1-63
- server_info view 1-66
- sql_languages view 1-66
- sysdepend data 1-27
- sysindexes view 1-38
- sys synonyms data 1-51
- sysstable data 1-51
- systabauth data 1-53
- systables data 1-54
- sysviews data 1-59
- tables view 1-64

Violations

- sysobjstate data 1-41
- sysviolations data 1-60

Virtual machine 3-18, 3-66

Virtual processor 3-77

Visual disabilities

- reading syntax diagrams C-1

W

Warning message 1-30, 3-22

WHERE keyword 1-9, 1-18, 2-35

Whitespace characters 3-67

Whitespace in identifiers 3-43

Window borders 3-62

Windows environments

- manipulating environment variables 3-10
- setting environment variables 3-5

Windows NT

- default locale for xi

Windows registry 3-10

X

X setting

- sysams.am_sptype 1-13
- systabauth.tabauth 1-53

X/Open

- compliance 1-66
- Information Schema views 1-63
- server_info view 1-66

XBSA

- debugging records 3-65
- message log file 3-65
- shared library 3-65

XFER_CONFIG environment variable 3-81

xfer_config file 3-81

XML (Extensible Markup Language) 2-10

XOR setting 3-65

XPG4 standard 1-65

XPS (Extended Parallel Server) 1-10, 3-14, B-1

Y

Y setting

- DBDATE 3-26
- DBTIME 3-40
- sysroleauth.is_grantable 1-50

Year 2000 3-23

YEAR keyword

- DATETIME qualifier 2-12
- EXTEND function 2-43
- INTERVAL qualifier 2-20

Year values, two and four digit 2-14, 3-22, 3-26, 3-40

yes setting

- NODEFDAC 3-68

YES setting

- columns.is_nullable 1-65
- sql_languages.integrity 1-66

Z

Zero (0)

- C null as terminator 2-37
- DBDATE separator 3-26
- DECIMAL scale 2-15
- hexadecimal digit 3-28
- IFX_DIRECTIVES setting 3-49, 3-50
- IFX_LONGID setting 3-51
- IFX_NETBUF_PVTPOOL_SIZE setting 3-52
- INFORMIXOPCACHE setting 3-59
- integer scale 1-65, 2-15
- ISM_DEBUG_LEVEL setting 3-65
- OPTCOMPIND setting 3-69
- OPTMSG setting 3-70
- OPTOFC setting 3-71
- padding of 1-digit years 3-23
- padding with DBFLTMASK 3-29
- padding with DBTIME 3-40
- PDQPRIORITY setting 3-72

Zero (0) *(continued)*

- PSORT_NPROCS setting 3-77
- STMT_CACHE setting 3-77
- sysams values 1-13, 1-14
- sysfragments.hybdpos 1-35
- sysindices.nrows 1-38
- systables.type_xid 1-55
- sysxdtypes values 1-62
- zip column B-12
- zipcode column A-2, B-11



Printed in USA

G251-2283-00



Spine information:

IBM DB2 IBM Informix **Version 10.0/8.5**

IBM Informix Guide to SQL: Reference

