Informix Guide to SQL

Reference

Informix Extended Parallel Server, Version 8.3
Informix Dynamic Server.2000, Version 9.2

December 1999
Part No. 000-6526

Published by Informix® Press

Informix Corporation
4100 Bohannon Drive
Menlo Park, CA 94025-1032

Documentation Team: Brian Deutscher, Mary Kraemer, Jennifer Leland

Table of Contents

Introduction

In This Introduction . . . 3
About This Manual . . . 3
- Types of Users . . . 3
- Software Dependencies . . . 4
- Assumptions About Your Locale . . . 4
- Demonstration Databases . . . 5

New Features . . . 5
- New Features in Version 8.3 . . . 6
- New Features in Version 9.2 . . . 6

Documentation Conventions . . . 6
- Typographical Conventions . . . 7
- Icon Conventions . . . 7
- Command-Line Conventions . . . 9
- Sample-Code Conventions . . . 12

Additional Documentation . . . 12
- On-Line Manuals . . . 13
- Printed Manuals . . . 13
- Error Message Documentation . . . 13
- Documentation Notes, Release Notes, Machine Notes . . . 14
- Related Reading . . . 15

Compliance with Industry Standards . . . 15
Informix Welcomes Your Comments . . . 16

Chapter 1 System Catalog

In This Chapter . . . 1-5
Objects That the System Catalog Tables Track . . . 1-5
Using the System Catalog . . . 1-6
Accessing the System Catalog . . . 1-12
Updating System Catalog Data . . . 1-13
Structure of the System Catalog . . . 1-14
SYSAGGREGATES . . . 1-16
SYSAMS. . . . 1-17
SYSATTRTYPES . . . 1-23
SYSBLOBS . . . 1-24
SYSCASTS . . . 1-25
SYSCHECKS . . . 1-26
SYSCOLATTRIBS . . . 1-27
SYSCOLAUTH . . . 1-28
SYSCOLDEPEND . . . 1-29
SYSCOLUMNS . . . 1-30
SYSCONSTRAINTS. . . . 1-36
SYSDEFAULTS . . . 1-37
SYSDEPEND . . . 1-38
SYSDISTRIB . . . 1-39
SYSERRORS . . . 1-41
SYSEXTCOLS . . . 1-42
SYSEXTDFILES . . . 1-43
SYSEXTERNAL . . . 1-44
SYSFRAGAUTH . . . 1-45
SYSFRAGMENTS . . . 1-46
SYSINDEXES . . . 1-49
SYSINDICES . . . 1-51
SYSINHERITS. . . . 1-53
SYSLANGAUTH. . . . 1-53
SYSLOGMAP . . . 1-54
SYSNEWDEPEND . . . 1-54
SYSOBJSTATE. . . . 1-55
SYSOPCLASSES . . . 1-56
SYSOPCLSTR . . . 1-57
SYSPROCAUTH . . . 1-59
SYSPROCBODY . . . 1-60
SYSPROCEDURES . . . 1-61

SYSPROCPLAN 1-65
SYSREFERENCES 1-66
SYSREPOSITORY 1-67
SYSROLEAUTH 1-68
SYSROUTINELANGS 1-68
SYSSYNONYMS 1-69
SYSSYNTABLE 1-70
SYSTABAMDATA 1-71
SYSTABAUTH 1-72
SYSTABLES 1-73
SYSTRACECLASSES 1-75
SYSTRACEMSGS 1-76
SYSTRIGBODY 1-77
SYSTRIGGERS 1-78
SYSUSERS 1-79
SYSVIEWS 1-80
SYSVIOLATIONS 1-81
SYSXTDDESC 1-82
SYSXTDTYPEAUTH 1-83
SYSXTDTYPES 1-84
Information Schema 1-86
Generating the Information Schema Views 1-86
Accessing the Information Schema Views 1-87
Structure of the Information Schema Views 1-87

Chapter 2 **Data Types**

In This Chapter 2-5
Summary of Data Types 2-5
Description of Data Types 2-9
BLOB 2-9
BOOLEAN 2-11
BYTE 2-12
CHAR(*n*) 2-13
CHARACTER(*n*) 2-14
CHARACTER VARYING(*m*,*r*) 2-15
CLOB 2-15
DATE 2-16
DATETIME 2-17
DEC. 2-21

DECIMAL 2-21
Distinct 2-23
DOUBLE PRECISION 2-24
FLOAT(*n*) 2-24
INT 2-25
INT8 2-25
INTEGER 2-26
INTERVAL 2-26
LIST(*e*) 2-30
LVARCHAR 2-31
MONEY(*p,s*) 2-32
MULTISET(*e*) 2-33
Named Row 2-34
NCHAR(*n*) 2-34
NUMERIC(*p,s*) 2-34
NVARCHAR(*m,r*) 2-34
Opaque 2-35
REAL 2-35
Row, Named 2-36
Row, Unnamed 2-37
SERIAL(*n*) 2-39
SERIAL8. 2-40
SET(*e*). 2-42
SMALLFLOAT 2-43
SMALLINT. 2-44
TEXT 2-44
Unnamed Row 2-46
VARCHAR(*m,r*) 2-46

Built-In Data Types 2-48
Large-Object Data Types 2-49
Time Data Types 2-52

Extended Data Types 2-58
Complex Data Types 2-59
Distinct Data Types 2-62
Opaque Data Types 2-62

Data Type Casting and Conversion 2-63
Using Built-in Casts 2-64
Using User-Defined Casts 2-67
Determining Which Cast to Apply 2-68
Casts for Distinct Types 2-69
What Extended Data Types Can Be Cast? 2-70
Operator Precedence 2-71

Chapter 3 **Environment Variables**

In This Chapter 3-5
Types of Environment Variables 3-6
Where to Set Environment Variables in UNIX 3-7
Where to Set Environment Variables in Windows NT 3-7
Using Environment Variables in UNIX 3-8
Setting Environment Variables in an Environment-Configuration File 3-8
Setting Environment Variables at Login Time 3-9
Syntax for Setting Environment Variables 3-10
Unsetting Environment Variables 3-10
Modifying an Environment-Variable Setting 3-11
Viewing Your Environment-Variable Settings 3-12
Checking Environment Variables with the chkenv Utility . . 3-12
Rules of Precedence 3-13
Using Environment Variables in Windows NT 3-14
Setting Environment Variables for Native Windows Applications 3-14
Setting Environment Variables for Command-Prompt Utilities 3-15
Rules of Precedence 3-18
List of Environment Variables 3-19
Environment Variables 3-24
AC_CONFIG 3-24
ARC_CONFIG 3-25
ARC_DEFAULT 3-25
ARC_KEYPAD 3-26
COCKPITSERVICE 3-27
CPFIRST 3-28
DBACCNOIGN. 3-29
DBANSIWARN 3-30
DBBLOBBUF 3-31

DBCENTURY 3-32
DBDATE 3-36
DBDELIMITER 3-39
DBEDIT 3-40
DBFLTMASK 3-41
DBLANG 3-42
DBMONEY. 3-44
DBONPLOAD. 3-45
DBPATH. 3-46
DBPRINT 3-48
DBREMOTECMD 3-49
DBSPACETEMP 3-50
DBTEMP 3-52
DBTIME. 3-53
DBUPSPACE 3-56
DELIMIDENT. 3-57
ENVIGNORE 3-58
FET_BUF_SIZE 3-59
IFMX_SMLTBL_BROADCAST_SIZE 3-60
IFX_DIRECTIVES 3-61
IFX_LONGID 3-62
IFX_NETBUF_PVTPOOL_SIZE. 3-63
IFX_NETBUF_SIZE 3-63
IFX_UPDDESC 3-64
INFORMIXC 3-64
INFORMIXCONCSMCFG 3-65
INFORMIXCONRETRY 3-66
INFORMIXCONTIME 3-66
INFORMIXCPPMAP 3-68
INFORMIXDIR 3-68
INFORMIXKEYTAB. 3-69
INFORMIXOPCACHE 3-70
INFORMIXSERVER. 3-70
INFORMIXSHMBASE 3-72
INFORMIXSQLHOSTS. 3-73
INFORMIXSTACKSIZE 3-74
INFORMIXTERM 3-75
INF_ROLE_SEP 3-76
ISM_COMPRESSION 3-77

ISM_DEBUG_FILE 3-77
ISM_DEBUG_LEVEL. 3-78
ISM_ENCRYPTION 3-79
ISM_MAXLOGSIZE 3-79
ISM_MAXLOGVERS 3-80
LD_LIBRARY_PATH 3-80
LIBPATH 3-81
NODEFDAC 3-81
ONCONFIG 3-82
OPTCOMPIND 3-83
OPTMSG 3-84
OPTOFC 3-85
OPT_GOAL 3-86
PATH 3-87
PDQPRIORITY 3-88
PLCONFIG 3-90
PLOAD_LO_PATH 3-91
PLOAD_SHMBASE 3-91
PSORT_DBTEMP 3-92
PSORT_NPROCS 3-93
SHLIB_PATH 3-94
STMT_CACHE 3-95
TERM 3-96
TERMCAP 3-96
TERMINFO 3-97
THREADLIB. 3-98
XFER_CONFIG 3-98
Index of Environment Variables 3-99

Appendix A **The stores_demo Database**

Appendix B **The sales_demo and superstores_demo Databases**

Glossary

Index

Introduction

In This Introduction 3

About This Manual. 3
Types of Users 3
Software Dependencies 4
Assumptions About Your Locale. 4
Demonstration Databases 5

New Features. 5
New Features in Version 8.3 6
New Features in Version 9.2 6
Year 2000 Compliance 6
Version 9.2 Features from Dynamic Server 7.30 6

Documentation Conventions 6
Typographical Conventions 7
Icon Conventions 7
Comment Icons 8
Feature, Product, and Platform Icons 8
Command-Line Conventions 9
How to Read a Command-Line Diagram 11
Sample-Code Conventions 12

Additional Documentation 12
On-Line Manuals 13
Printed Manuals 13
Error Message Documentation 13
Documentation Notes, Release Notes, Machine Notes 14
Related Reading 15

Compliance with Industry Standards. 15
Informix Welcomes Your Comments 16

In This Introduction

This Introduction provides an overview of the information in this manual and describes the conventions it uses.

About This Manual

This manual includes information about system catalog tables, data types, and environment variables that Informix products use. It also includes a glossary that contains definitions of common terms found in Informix documentation and a description of the demonstration databases that Version 9.2 of Informix Dynamic Server 2000 and Version 8.3 of Informix Extended Parallel Server provide.

This manual is one of a series of manuals that discusses the Informix implementation of SQL. The *Informix Guide to SQL: Syntax* contains all the syntax descriptions for SQL and stored procedure language (SPL). The *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 *Informix Guide to Database Design and Implementation* shows how to use SQL to implement and manage your databases.

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 *Getting Started* manual for your database server for a list of supplementary titles.

Software Dependencies

This manual assumes that you are using one of the following database servers:

- Informix Extended Parallel Server, Version 8.3
- Informix Dynamic Server 2000, Version 9.2

Assumptions About Your Locale

Informix products can support many languages, cultures, and code sets. All culture-specific information is brought together in a single environment, called a Global Language Support (GLS) locale.

This manual assumes that you use the U.S. 8859-1 English locale as the default locale. The default is **en_us.8859-1** (ISO 8859-1) on UNIX platforms or **en_us.1252** (Microsoft 1252) for Windows NT environments. This locale supports U.S. English format conventions for dates, times, and currency, and also supports the ISO 8859-1 or Microsoft 1252 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 your SQL identifiers, or if you want to conform to the nondefault collation rules of character data, you need to specify the appropriate nondefault locale.

For instructions on how to specify a nondefault locale, additional syntax, and other considerations related to GLS locales, see the *Informix Guide to GLS Functionality*.

Demonstration Databases

The DB-Access utility, which is provided with your Informix 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 Informix manuals are based on the **stores_demo** database.

XPS

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

IDS

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

For information about how to create and populate the demonstration databases, see the *DB-Access User's Manual*. For descriptions of the databases and their contents, see this manual.

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.

New Features

For a comprehensive list of new database server features, see the release notes. This section lists new features relevant to this manual.

New Features in Version 8.3

This manual describes the DBCENTURY environment variable, which is a Year 2000 compliance feature in Version 8.3 of Extended Parallel Server.

New Features in Version 9.2

This manual describes new features in Version 9.2 of Dynamic Server. The features fall into the following areas:

- Year 2000 Compliance
- Version 9.2 features from Version 7.30 of Dynamic Server

Year 2000 Compliance

This manual describes the DBCENTURY environment variable in Version 9.2 of Dynamic Server.

Version 9.2 Features from Dynamic Server 7.30

This manual also describes the **IFX_UPDDESC** environment variable, which was first released in Version 7.30.

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
- Icon conventions
- Command-line conventions
- Sample-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.
italics ***italics*** `italics`	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` `monospace`	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 the end of one or more product- or platform-specific paragraphs.
→	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.*

Icon Conventions

Throughout the documentation, you will find text that is identified by several different types of icons. This section describes these icons.

Comment Icons

Comment icons identify three types of information, as the following table describes. This information always appears in italics.

Icon	Label	Description
⚠	***Warning:***	Identifies paragraphs that contain vital instructions, cautions, or critical information
⇨	***Important:***	Identifies paragraphs that contain significant information about the feature or operation that is being described
💡	***Tip:***	Identifies paragraphs that offer additional details or shortcuts for the functionality that is being described

Feature, Product, and Platform Icons

Feature, product, and platform icons identify paragraphs that contain feature-specific, product-specific, or platform-specific information.

Icon	Description
GLS	Identifies information that relates to the Informix Global Language Support (GLS) feature
IDS	Identifies information that is specific to Informix Dynamic Server 2000

(1 of 2)

Icon	Description
UNIX	Identifies information that is specific to UNIX platforms
WIN NT	Identifies information that is specific to the Windows NT environment
XPS	Identifies information or syntax that is specific to Informix Extended Parallel Server

(2 of 2)

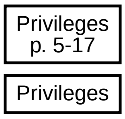
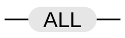
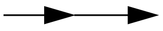
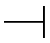
These icons can apply to an entire section or to one or more paragraphs within a section. If an icon appears next to a section heading, the information that applies to the indicated feature, product, or platform ends at the next heading at the same or higher level. A ♦ symbol indicates the end of feature-, product-, or platform-specific information that appears within one or more paragraphs within a section.

Command-Line Conventions

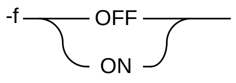
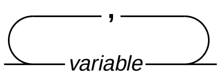
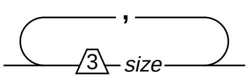
This section defines and illustrates the format of commands that are available in Informix products. These commands have their own conventions, which might include alternative forms of a command, required and optional parts of the command, and so forth.

Each diagram displays the sequences of required and optional elements that are valid in a command. A diagram begins at the upper-left corner with a command. It ends at the upper-right corner with a vertical line. Between these points, you can trace any path that does not stop or back up. Each path describes a valid form of the command. You must supply a value for words that are in italics.

You might encounter one or more of the following elements on a command-line path.

Element	**Description**
command	This required element is usually the product name or other short word that invokes the product or calls the compiler or preprocessor script for a compiled Informix product. It might appear alone or precede one or more options. You must spell a command exactly as shown and use lowercase letters.
variable	A word in italics represents a value that you must supply, such as a database, file, or program name. A table following the diagram explains the value.
-flag	A flag is usually an abbreviation for a function, menu, or option name, or for a compiler or preprocessor argument. You must enter a flag exactly as shown, including the preceding hyphen.
.ext	A filename extension, such as **.sql** or **.cob**, might follow a variable that represents a filename. Type this extension exactly as shown, immediately after the name of the file. The extension might be optional in certain products.
(. , ; + * - /)	Punctuation and mathematical notations are literal symbols that you must enter exactly as shown.
' '	Single quotes are literal symbols that you must enter as shown.
 Privileges p. 5-17 Privileges	A reference in a box represents a subdiagram. Imagine that the subdiagram is spliced into the main diagram at this point. When a page number is not specified, the subdiagram appears on the same page.
 ALL	A shaded option is the default action.
	Syntax within a pair of arrows indicates a subdiagram.
	The vertical line terminates the command.

(1 of 2)

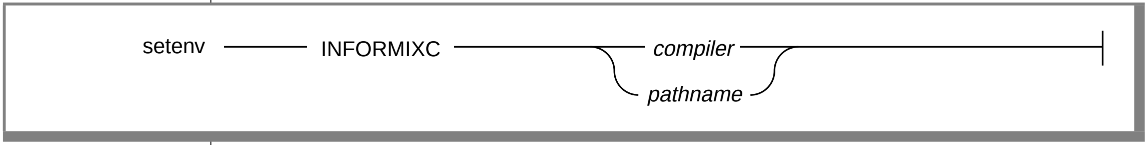
Element	Description
	A branch below the main path indicates an optional path. (Any term on the main path is required, unless a branch can circumvent it.)
	A loop indicates a path that you can repeat. Punctuation along the top of the loop indicates the separator symbol for list items.
	A gate (3) on a path indicates that you can only use that path the indicated number of times, even if it is part of a larger loop. You can specify *size* no more than three times within this statement segment.

(2 of 2)

How to Read a Command-Line Diagram

Figure 1 shows a command-line diagram that uses some of the elements that are listed in the previous table.

Figure 1
Example of a Command-Line Diagram



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.

Figure 1 illustrates the following steps:

1. Type `setenv`.
2. Type `INFORMIXC`.
3. Supply either a compiler name or a pathname.

   After you choose *compiler* or *pathname*, you come to the terminator. Your command is complete.
4. Press RETURN to execute the command.

Sample-Code Conventions

Examples of SQL code occur throughout this manual. Except where noted, the code is not specific to any single 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, you might want to refer to the following types of documentation:

- On-line manuals
- Printed manuals
- Error message documentation
- Documentation notes, release notes, and machine notes
- Related reading

On-Line Manuals

An Answers OnLine CD that contains Informix manuals in electronic format is provided with your Informix products. You can install the documentation or access it directly from the CD. For information about how to install, read, and print on-line manuals, see the installation insert that accompanies Answers OnLine.

Informix on-line manuals are also available on the following Web site:

```
www.informix.com/answers
```

Printed Manuals

To order printed manuals, call 1-800-331-1763 or send email to moreinfo@informix.com. Please provide the following information when you place your order:

- The documentation that you need
- The quantity that you need
- Your name, address, and telephone number

Error Message Documentation

Informix software products provide ASCII files that contain all of the Informix error messages and their corrective actions.

UNIX

To read error messages and corrective actions on UNIX, use one of the following utilities.

Utility	Description
finderr	Displays error messages on line
rofferr	Formats error messages for printing

♦

WIN NT

To read error messages and corrective actions in Windows environments, use the **Informix Find Error** utility. To display this utility, choose **Start→Programs→Informix** from the Task Bar. ♦

Instructions for using the preceding utilities are available in Answers OnLine. Answers OnLine also provides a listing of error messages and corrective actions in HTML format.

Documentation Notes, Release Notes, Machine Notes

In addition to printed documentation, the following sections describe the on-line files that supplement the information in this manual. Please examine these files before you begin using your database server. They contain vital information about application and performance issues.

UNIX

On UNIX platforms, the following on-line files appear in the **$INFORMIXDIR/release/en_us/0333** directory. Replace *x.y* in the filenames with the version number of your database server.

On-Line File	Purpose
SQLRDOC_*x.y*	The documentation notes file for your version of this manual describes topics that are not covered in the manual or that were modified since publication.
SERVERS_*x.y*	The release notes file describes feature differences from earlier versions of Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.
IDS_*x.y* or **XPS_*x.y***	The machine notes file describes any special actions that you must take to configure and use Informix products on your computer. Machine notes are named for the product described.

♦

WIN NT

The following items appear in the **Informix** folder. To display this folder, choose **Start→Programs→Informix** from the Task Bar.

Program Group Item	Description
Documentation Notes	This item includes additions or corrections to manuals and information about features that might not be covered in the manuals or that have been modified since publication.
Release Notes	This item describes feature differences from earlier versions of Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.

The machine notes do not apply to Windows environments. ♦

Related Reading

For a list of publications that provide an introduction to database servers and operating-system platforms, refer to your *Getting Started* manual.

Compliance with Industry Standards

The American National Standards Institute (ANSI) has established a set of industry standards for SQL. 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 Informix database servers comply with the SQL-92 Intermediate and Full Level and X/Open SQL CAE (common applications environment) standards.

Informix Welcomes Your Comments

Let us know what you like or dislike about our manuals. To help us with future versions of our manuals, we want to know about any corrections or clarifications that you would find useful. Include the following information:

- The name and version of the manual that you are using
- Any comments that you have about the manual
- Your name, address, and phone number

Send electronic mail to us at the following address:

doc@informix.com

The **doc** alias is reserved exclusively for reporting errors and omissions in our documentation.

We appreciate your suggestions.

Chapter

System Catalog

1

In This Chapter 1-5

Objects That the System Catalog Tables Track. 1-5

Using the System Catalog 1-6
Accessing the System Catalog. 1-12
Updating System Catalog Data 1-13

Structure of the System Catalog 1-14
SYSAGGREGATES 1-16
SYSAMS 1-17
SYSATTRTYPES 1-23
SYSBLOBS 1-24
SYSCASTS 1-25
SYSCHECKS 1-26
SYSCOLATTRIBS 1-27
SYSCOLAUTH 1-28
SYSCOLDEPEND 1-29
SYSCOLUMNS. 1-30
SYSCONSTRAINTS 1-36
SYSDEFAULTS 1-37
SYSDEPEND 1-38
SYSDISTRIB 1-39
SYSERRORS 1-41
SYSEXTCOLS 1-42
SYSEXTDFILES. 1-43
SYSEXTERNAL 1-44
SYSFRAGAUTH 1-45
SYSFRAGMENTS 1-46

SYSINDEXES 1-49
SYSINDICES. 1-51
SYSINHERITS 1-53
SYSLANGAUTH 1-53
SYSLOGMAP 1-54
SYSNEWDEPEND. 1-54
SYSOBJSTATE 1-55
SYSOPCLASSES 1-56
SYSOPCLSTR 1-57
SYSPROCAUTH 1-59
SYSPROCBODY 1-60
SYSPROCEDURES. 1-61
SYSPROCPLAN 1-65
SYSREFERENCES 1-66
SYSREPOSITORY 1-67
SYSROLEAUTH 1-68
SYSROUTINELANGS 1-68
SYSSYNONYMS 1-69
SYSSYNTABLE 1-70
SYSTABAMDATA 1-71
SYSTABAUTH 1-72
SYSTABLES 1-73
SYSTRACECLASSES 1-75
SYSTRACEMSGS 1-76
SYSTRIGBODY 1-77
SYSTRIGGERS 1-78
SYSUSERS 1-79
SYSVIEWS 1-80
SYSVIOLATIONS 1-81
SYSXTDDESC 1-82
SYSXTDTYPEAUTH 1-83
SYSXTDTYPES 1-84

Information Schema 1-86
- Generating the Information Schema Views 1-86
- Accessing the Information Schema Views 1-87
- Structure of the Information Schema Views. 1-87
  - The tables Information Schema View 1-88
  - The columns Information Schema View 1-88
  - The sql_languages Information Schema View 1-90
  - The server_info Information Schema View. 1-90

In This Chapter

The system catalog consists of tables that describe the structure of the database. Each system catalog table contains specific information about an element in the database.

This chapter details the following material:

- Objects that the system catalog tables track
- Using the system catalog (page 1-6)
- Structure of the system catalog (page 1-13)
- Information Schema (page 1-86)

Objects That the System Catalog Tables Track

The system catalog tables track the following objects:

- Tables and constraints
- Views
- Triggers
- Authorized users and privileges associated with every table that you create
- User-defined routines
- Data types
- Casts

- Access methods and operator classes
- Error, warning, and informational messages associated with user-defined routines

IDS

- Inheritance relationships ♦

Using the System Catalog

Informix database servers automatically generate the system catalog tables when you create a database. You can query them as you would query any other table in the database. For a newly created database, the system catalog tables for the database reside in a common area of the disk called a dbspace.

Every database has its own system catalog tables. All tables in the system catalog have the prefix **sys** (for example, the **systables** system catalog table).

Not all tables that have the prefix ***sys*** are true system catalog tables. For example, a common DataBlade builds a table called **sysbuiltintypes** that looks like a system table and contains similar information. However, it has an id > 99. System catalog tables all have an id < 99.

Tip: *Do not confuse the system catalog tables of a database with the tables in the* ***sysmaster*** *database. The* ***sysmaster*** *tables also have the* ***sys*** *prefix, but they contain information about an entire database server, which might manage many databases. The information in the* ***sysmaster*** *tables is primarily useful for database administrators (DBAs). For more information about the* ***sysmaster*** *tables, see the "Administrator's Guide."*

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

```
CREATE SCHEMA AUTHORIZATION maryl
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 cathl WITH GRANT OPTION AS maryl
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 table and view names do not already exist for the specified owners.)
- The user has permission to create the 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**

XPS
IDS

- **syscolauth**
- **sysindexes** ♦
- **sysindices** ♦

The following two new rows of information are added to the **systables** system catalog table after the CREATE SCHEMA block is run, as page 1-7 shows.

```
tabname              customer
owner                maryl
partnum              16778361
tabid                101
rowsize              134
ncols                10
nindexes             2
nrows                0
created              01/26/1999
version              1
tabtype              T
locklevel            P
npused               0
fextsize             16
nextsize             16
flags                0
site
dbname

tabname              california
owner                maryl
partnum              0
tabid                102
rowsize              134
ncols                4
nindexes             0
nrows                0
created              01/26/1999
version              0
tabtype              V
locklevel            B
npused               0
fextsize             0
nextsize             0
flags                0
site
dbname
```

Each table recorded in the **systables** system catalog table is assigned a **tabid**, a system-assigned sequential ID number that uniquely identifies each table in the database. The system catalog tables receive **tabid** numbers `1` through `24`, and the user-created tables receive **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** system catalog 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 contain no entries. These two columns contain values when a column is the first key in a composite index or is the only key in the index, has no null or duplicate values, and the UPDATE STATISTICS statement has been run.

The rows that the following example shows are added to the **sysviews** system catalog table. These rows correspond to the CREATE VIEW portion of the CREATE SCHEMA block.

```
tabid    seqview    text
102      1          ,address1, address2, city, state,zipcode,phone) a
                    s select x0.custom
102      2          er_num, x0.fname, x0.lname, x0.company,
                    x0.address1, x0.address2
102      3          ,x0.city, x0.state, x0.zipcode, x0.phone from
                    'maryl'.customer
102      4          x0 where (x0.state = 'CA');
```

The **sysviews** system catalog table contains the CREATE VIEW statement that creates the view. Each line of the CREATE VIEW statement in the current schema is stored in this table. In the **viewtext** column, the **x0** that precedes the column names in the statement (for example, **x0.fname**) operates as an alias name 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           1 01              SU-IDXAR
maryl             nhowe            101              --*-----
                 maryl             102              SU-ID---
```

The **tabauth** column of this table 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.

If the **tabauth** privilege code is uppercase (for example, `S` for select), the user who is granted this privilege can also grant it to others. If the **tabauth** privilege code is lowercase (for example, `s` for select), the user who has this privilege 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** system catalog table for Informix Dynamic Server 2000.) 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
```

In this table, the **idxtype** column identifies whether the created index is unique or a duplicate. For example, the index **c_num_ix** that is placed on the **customer_num** column of the **customer** table 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 system catalog. They can, however, 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 table ID numbers 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 SUBSTR or SUBSTRING function to select only part of a source string. To display the list of database tables in columns, enter the following statement:

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

Warning: *Although user **informix** and DBAs can modify most system catalog tables (only user **informix** can modify **systables**), Informix strongly recommends that you do not update, delete, or insert any rows in them. Modifying the system catalog tables can destroy the integrity of the database. Informix supports the use of the ALTER TABLE statement to modify the size of the next extent of system catalog tables.*

IDS

*However, in certain cases with Dynamic Server, 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 developer of DataBlade modules can add message entries that appear in these system catalog tables.* ♦

Updating System Catalog Data

The optimizer in Informix database servers determines the most efficient strategy for executing SQL queries. The optimizer allows you to query the database without having to fully consider 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, 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 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.

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 use the medium or high mode with the UPDATE STATISTICS statement, the **sysdistrib** system catalog table holds the updated data-distribution data after you run UPDATE STATISTICS.

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

Structure of the System Catalog

The following system catalog tables describe the structure of an Informix database.

System Catalog Table	XPS	IDS	Page
sysaggregates		✔	1-16
sysams		✔	1-17
sysattrtypes		✔	1-22
sysblobs	✔	✔	1-23
syscasts		✔	1-24
syschecks	✔	✔	1-25
syscolattribs		✔	1-26
syscolauth	✔	✔	1-28
syscoldepend	✔	✔	1-29
syscolumns	✔	✔	1-30
sysconstraints	✔	✔	1-36
sysdefaults	✔	✔	1-37
sysdepend	✔	✔	1-38
sysdistrib	✔	✔	1-39
syserrors		✔	1-41
sysextcols	✔		1-42
sysextdfiles	✔		1-43
sysexternal	✔		1-44
sysfragauth		✔	1-45
sysfragments	✔	✔	1-46

(1 of 3)

System Catalog Table	XPS	IDS	Page
sysindexes	✔		1-49
sysindices		✔	1-51
sysinherits		✔	1-53
syslangauth		✔	1-53
syslogmap			1-54
sysnewdepend	✔		1-54
sysobjstate	✔	✔	1-55
sysopclasses		✔	1-56
sysopclstr		✔	1-57
sysprocauth	✔	✔	1-59
sysprocbody	✔	✔	1-60
sysprocedures	✔	✔	1-61
sysprocplan	✔	✔	1-65
sysreferences	✔	✔	1-66
sysrepository	✔		1-67
sysroleauth		✔	1-68
sysroutinelangs		✔	1-68
syssynonyms			1-69
syssyntable	✔	✔	1-70
systabamdata		✔	1-71
systabauth	✔	✔	1-72
systables	✔	✔	1-73
systraceclasses		✔	1-75
systracemsgs		✔	1-76

(2 of 3)

System Catalog Table	XPS	IDS	Page
systrigbody		✔	1-77
systriggers		✔	1-78
sysusers	✔	✔	1-79
sysviews	✔	✔	1-80
sysviolations	✔	✔	1-81
sysxtddesc		✔	1-82
systdtypeauth		✔	1-83
sysxtdtypes		✔	1-84

(3 of 3)

GLS

In a database whose collation order is locale dependent, all character information in the system catalog tables is stored in NCHAR rather than CHAR columns. However, for those databases where the collation order is code-set dependent, all character information in the system catalog tables is stored in CHAR columns. For more information on collation orders and NCHAR and NVARCHAR data types, see the *Informix Guide to GLS Functionality*. For information about data types, see Chapter 2 of this manual. ♦

IDS

SYSAGGREGATES

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

Column Name	Type	Explanation
name	NVARCHAR(128)	Aggregate name
owner	NCHAR(32)	Aggregate owner
aggid	SERIAL	Aggregate identifier
init_func	NVARCHAR(128)	Name of initialization UDR
iter_func	NVARCHAR(128)	Name of iterator UDR

(1 of 2)

Column Name	Type	Explanation
combine_func	NVARCHAR(128)	Name of combine UDR
final_func	NVARCHAR(128)	Name of finalization UDR
handlesnulls	BOOLEAN	Whether nulls should be considered

(2 of 2)

Each UDA has one entry in **sysaggregates** that is uniquely identified by its aggregate identifier (the **aggid** column). Only new aggregates (aggregates that are not built in) have entries in **sysaggregates**.

IDS

SYSAMS

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 SQL statement that is described in the *Virtual-Table Interface Programmer's Manual*. The **sysams** table has the following columns.

Column Name	Type	Explanation
am_name	NVARCHAR(128)	Name of the access method
am_owner	NCHAR(32)	Owner of the access method
am_id	INTEGER	Unique identifier for the access method. This value corresponds to the *am_id* in the **systables** system catalog table and to the *am_id* in the **sysindices** and **sysopclasses** system catalog tables.

am_type	NCHAR(1)	Type of access method:
		P = Primary
		S = Secondary

(1 of 6)

Column Name	Type	Explanation
am_sptype	NCHAR(3)	Type(s) of space(s) in which the access method can live: D or d = dbspaces only X or x = extspaces only S or s = sbspaces only (smart-large-object space) A or a = all types: extspaces, dbspaces, or 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.
am_defopclass	INTEGER	Default-operator class identifier. The *opclassid* from the entry for this operator class in the **sysopclasses** system catalog 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 `Non-Zero` = access method supports key scan `Zero` = access method does not support key scan
am_unique	INTEGER	Whether a secondary access method can support unique keys `Non-Zero` = access method supports unique keys `Zero` = access method does not support unique keys

(2 of 6)

Column Name	Type	Explanation
am_cluster	INTEGER	Whether a primary access method supports clustering `Non-Zero` = access method supports clustering `Zero` = access method does not support clustering
am_rowids	INTEGER	Whether a primary access method supports rowids `Non-Zero` = access method supports rowids `Zero` = access method does not support rowids
am_readwrite	INTEGER	Whether a primary access method is read / write `Non-Zero` = access method is read / write `Zero` = access method is read only
am_parallel	INTEGER	Whether an access method supports parallel execution `Non-Zero` = access method supports parallel execution `Zero` = access method does not support parallel execution
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 The value of **am_create** is the *procid* for the routine in the **sysprocedures** system catalog table

(3 of 6)

Column Name	Type	Explanation
am_drop	INTEGER	The routine specified for the AM_DROP purpose for this access method The value of **am_drop** is the *procid* listed for the routine in the **sysprocedures** system catalog table
am_open	INTEGER	The routine specified for the AM_OPEN purpose for this access method The value of **am_open** is the *procid* listed for the routine in the **sysprocedures** system catalog table
am_close	INTEGER	The routine specified for the AM_CLOSE purpose for this access method The value of **am_close** is the *procid* listed for the routine in the **sysprocedures** system catalog table
am_insert	INTEGER	The routine specified for the AM_INSERT purpose for this access method The value of **am_insert** is the *procid* listed for the routine in the **sysprocedures** system catalog table
am_delete	INTEGER	The routine specified for the AM_DELETE purpose for this access method The value of **am_delete** is the *procid* listed for the routine in the **sysprocedures** system catalog table
am_update	INTEGER	The routine specified for the AM_UPDATE purpose for this access method The value of **am_update** is the *procid* listed for the routine in the **sysprocedures** system catalog table
am_stats	INTEGER	The routine specified for the AM_STATS purpose for this access method The value of **am_stats** is the *procid* listed for the routine in the **sysprocedures** system catalog table

(4 of 6)

Column Name	Type	Explanation
am_scancost	INTEGER	The routine specified for the AM_SCANCOST purpose for this access method The value of **am_scancost** is the *procid* listed for the routine in the **sysprocedures** system catalog table
am_check	INTEGER	The routine specified for the AM_CHECK purpose for this access method The value of **am_check** is the *procid* listed for the routine in the **sysprocedures** system catalog table
am_beginscan	INTEGER	The routine specified for the AM_BEGINSCAN purpose for this access method The value of **am_beginscan** is the *procid* listed for the routine in the **sysprocedures** system catalog table
am_endscan	INTEGER	The routine specified for the AM_ENDSCAN purpose for this access method The value of **am_endscan** is the *procid* listed for the routine in the **sysprocedures** system catalog table
am_rescan	INTEGER	The routine specified for the AM_RESCAN purpose for this access method The value of **am_rescan** is the *procid* listed for the routine in the **sysprocedures** system catalog table
am_getnext	INTEGER	The routine specified for the AMGETNEXT purpose for this access method The value of **am_getnext** is the *procid* listed for the routine in the **sysprocedures** system catalog table

(5 of 6)

Column Name	Type	Explanation
am_getbyid	INTEGER	The routine specified for the AM_GETBYID purpose for this access method The value of **am_getbyid** is the *procid* listed for the routine in the **sysprocedures** system catalog tables
am_build	Reserved for future use	
am_init	INTEGER	For internal use

(6 of 6)

The composite index for the **am_name** and **am_owner** columns in this table allows only unique values. The **am_id** column is indexed and must contain unique values.

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.

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

IDS

SYSATTRTYPES

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** system catalog table has the following columns.

Column Name	Type	Explanation
extended_id	INTEGER	Identifier for extended data types, same as in sysxtdtypes
seqno	SMALLINT	Value to order and identify entries for specific values of **extended_id**
levelno	SMALLINT	Position of member in collection hierarchy
parent_no	SMALLINT	Value in the **seqno** column of the complex type that contains this member
fieldname	NVARCHAR(128)	Name of the field in a row type. Null for other complex types
fieldno	SMALLINT	Field number sequentially assigned by the system (from left to right within each row type)
type	SMALLINT	Identifier of the data type. For a complete list of values associated with different data types, see the **coltype** column entries in the **syscolumns** system catalog table.
length	SMALLINT	Length of the data type
xtd_type_id	INTEGER	The identifier used for this data type in the *extended_id* column of the **sysxtdtypes** system catalog table

The two indexes on the **extended_id** column and the **xtd_type_id** column, respectively, allow duplicate values. The composite index on **extended_id** and **seqno** columns allows only unique values.

SYSBLOBS

The title **sysblobs** is a legacy name based on a term that was used to refer to BYTE and TEXT columns (also known as simple large objects).

The **sysblobs** system catalog table specifies the storage location of a BYTE or TEXT column. It contains one row for each BYTE or TEXT column in a table. The **sysblobs** system catalog table has the following columns.

Column Name	Type	Explanation
spacename	NVARCHAR(128)	Partition BYTE or TEXT data, dbspace, or family name

type	NCHAR(1)	Media type:
		M = Magnetic
		O = Optical. Only available for Dynamic Server

tabid	INTEGER	Table identifier
colno	SMALLINT	Column number

A composite index for the **tabid** and **colno** columns 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 your *Administrator's Guide* and the *Administrator's Reference*.

IDS

SYSCASTS

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

Column Name	Type	Explanation
owner	NCHAR(32)	Owner of cast (user **informix** for built-in casts and user name for implicit and explicit casts)
argument_type	SMALLINT	Source data type on which the cast operates
argument_xid	INTEGER	Data type identifier of the source data type named in the **argument_type** column
result_type	SMALLINT	Data type returned by the cast
result_xid	INTEGER	Data type identifier of the data type named in the **result_type** column
routine_name	NVARCHAR(128)	Function or procedure used to implement the cast (might be null if the data types named in the **argument_type** and **result_type** columns have the same length and alignment and are both passed either by reference or by value)
routine_owner	NCHAR(32)	User name of the owner of the function or procedure named in the **routine_name** column

class	NCHAR	Type of cast:
		E = Explicit cast
		I = Implicit cast
		S = Built-in cast

If **routine_name** and **routine_owner** have null values, it indicates that the cast is defined without a routine.

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

SYSCHECKS

The **syschecks** system catalog table describes each check constraint defined in the database. Because the **syschecks** system catalog 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** system catalog table has the following columns.

Column Name	Type	Explanation
constrid	INTEGER	Constraint identifier
type	NCHAR(1)	Form in which the check constraint is stored:
		B = Binary encoded
		T = ASCII text
seqno	SMALLINT	Line number of the check constraint
checktext	NCHAR(32)	Text of the check constraint

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

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 constraint ID:

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

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

IDS

SYSCOLATTRIBS

The **syscolattribs** system catalog table describes the characteristics of smart large objects, namely CLOB and BLOB data types. It contains one row for each characteristic. The **syscolattribs** system catalog table has the following columns.

Column Name	Type	Explanation	
tabid	INTEGER	Table identifier	
colno	SMALLINT	Column number	
extentsize	INTEGER	Pages in smart-large-object extent, expressed in kilobytes	
flags	INTEGER	An integer representation of the combination (addition) of hexadecimal values of the following parameters:	
		LO_NOLOG	The smart large object is not logged.
		LO_LOG	Logging of smart-large-object data is done in accordance with the current database log mode.
		LO_KEEP_LASTACCESS_TIME	A record is kept of the most recent access of this smart-large-object column by a user.
		LO_NOKEEP_LASTACCESS_TIME	No record is kept of the most recent access of this smart-large-object column by a user.

(1 of 2)

Column Name	Type	Explanation	
		HI_INTEG	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	NVARCHAR(128)	Name of sbspace	

(2 of 2)

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** system catalog table has the following columns.

Column Name	Type	Explanation
grantor	NCHAR(32)	Grantor of privilege
grantee	NCHAR(32)	Grantee of privilege
tabid	INTEGER	Table identifier
colno	SMALLINT	Column number
colauth	NCHAR(3)	3-byte pattern that specifies column privileges: s = Select; u = Update; 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 composite index for the **tabid**, **grantor**, **grantee**, and **colno** columns allows only unique values. A composite index for 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 in the **syscoldepend** table for each column involved in the constraint. The **syscoldepend** system catalog table has the following columns.

Column Name	Type	Explanation
constrid	INTEGER	Constraint identifier
tabid	INTEGER	Table identifier
colno	SMALLINT	Column number

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

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 Name	Type	Explanation	
colname	NVARCHAR(128)	Column name	
tabid	INTEGER	Table identifier	
colno	SMALLINT	Column number that the system sequentially assigns (from left to right within each table)	
coltype	SMALLINT	Code for column data type:	
		0 = CHAR	14 = INTERVAL
		1 = SMALLINT	15 = NCHAR
		2 = INTEGER	16 = NVARCHAR
		3 = FLOAT	17 = INT8
		4 = SMALLFLOAT	18 = SERIAL8 *
		5 = DECIMAL	19 = SET
		6 = SERIAL *	20 = MULTISET
		7 = DATE	21 = LIST
		8 = MONEY	22 = rOW (unnamed)
		9 = NULL	23 = COLLECTION
		10 = DATETIME	24 = ROWREF
		11 = BYTE	40 = Variable-length opaque type
		12 = TEXT	41 = Fixed-length opaque type
		13 = VARCHAR	4118 = Named row type

(1 of 2)

Column Name	Type	Explanation
collength	SMALLINT	Column length (in bytes)
colmin	INTEGER	Second minimum value
colmax	INTEGER	Second maximum value
minlen	INTEGER	Minimum column length (in bytes)
maxlen	INTEGER	Maximum column length (in bytes)
extended_id	INTEGER	Type identifier, from the **sysxtdtypes** system catalog table, of the data type named in the **coltype** column

* An offset value of 256 is added to these columns to indicate that they do not allow null values.

(2 of 2)

A composite index for the **tabid** and **colno** columns allows only unique values.

The **coltype** 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.

Null-Valued Columns

If the **coltype** column contains a value greater than 256, it does not allow null values. To determine the data type for a **coltype** column that contains a value greater than 256, subtract 256 from the value and evaluate the remainder, based on the possible **coltype** values. For example, if a column has a **coltype** value of 262, subtracting 256 from 262 leaves a remainder of 6, which indicates that this column uses a SERIAL data type.

The next sections provide the following additional information about information in the **syscolumns** system catalog table:

- How the **coltype** and **collength** columns encode the type and length values, respectively, for certain data types.
- How the **colmin** and **colmax** columns store column values.

Storing Column Data Type

The database server stores the column data type as an integer value. For a list of the column data type values, see the description of the **coltype** column in the preceding table. The following sections provide additional information on data type values.

IDS

The following data types are implemented by the database server as *built-in opaque* types:

- BLOB
- CLOB
- BOOLEAN
- LVARCHAR

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 **sysxtdtypes** system catalog table.

The following table summarizes the **coltype** values for the predefined data types.

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

♦

Storing Column Length

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

Length of Integer-Based Columns

A **collength** value for a SMALLINT, INTEGER, or INT8 column is *not* machine-dependent. The database server uses the following lengths for SQL integer-based data types.

Integer-Based Data Type	Length (in bytes)
SMALLINT	2
INTEGER	4
INT8	8

The database server stores a SERIAL data type as an INTEGER value and a SERIAL8 data type as an INT8 value. Therefore, SERIAL has the same length as INTEGER (4 bytes) and SERIAL8 has the same length as INT8 (8 bytes).

Length of Fixed-Point Columns

A **collength** value for a MONEY or DECIMAL column is determined using the following formula:

```
(precision * 256) + scale
```

Length of Varying-Length Character Columns

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

- If the **collength** value is positive:

  ```
  collength = (min_space * 256) + max_size
  ```

- If the **collength** value is negative:

  ```
  collength + 65536 = (min_space * 256) + max_size
  ```

GLS

The database server uses the preceding formulas to encode the **collength** column for an NVARCHAR data type. For more information about the NVARCHAR data type, see the *Informix Guide to GLS Functionality*. ♦

Length for Time Data Types

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

```
(length * 256) + (largest_qualifier_value * 16) + smallest_qualifier_value
```

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

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

For example, if a DATETIME YEAR TO MINUTE column has a length of `12` (such as *YYYY:DD:MM:HH:MM)*, a *largest_qualifier value* of `0` (for YEAR), and a *smallest_qualifier value* of `8` (for MINUTE), the **collength** value is `3080` or `(256 * 12) + (0 * 16) + 8`.

Length of Simple-Large-Object Columns

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 searching 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 values for all other noninteger column types are the initial 4 bytes of the maximum or minimum value, which are treated as an integer.

IDS

The database server does not calculate **colmin** and **colmax** values for user-defined data types. However, these columns 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** (**sysindices** for Dynamic Server) system catalog table for each unique, primary key, or referential constraint that you create, if the constraint does not already have a corresponding entry in the **sysindexes** or **sysindices** system catalog table. Because indexes can be shared, more than one constraint can be associated with an index. The **sysconstraints** system catalog table has the following columns.

Column Name	Type	Explanation
constrid	SERIAL	System-assigned sequential identifier
constrname	NVARCHAR(128)	Constraint name
owner	NCHAR(32)	User name of owner
tabid	INTEGER	Table identifier
constrtype	NCHAR(1)	Constraint type:
		C = Check constraint
		P = Primary key
		R = Referential
		U = Unique
		N = Not null
idxname	NVARCHAR(128)	Index name

A composite index for the **constrname** and **owner** columns allows only unique values. The index for the **tabid** column allows duplicate values, and the index for 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** system catalog table.

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. If a default is not explicitly specified in the CREATE TABLE statement, no entry exists in this table. The **sysdefaults** system catalog table has the following columns.

Column Name	Type	Explanation
tabid	INTEGER	Table identifier
colno	SMALLINT	Column identifier
type	NCHAR(1)	Default type:
		L = Literal default
		U = User
		C = Current
		N = Null
		T = Today
		S = Dbservername
default	NCHAR(256)	If default type = L, the literal default value
class	CHAR(1)	Type of column:
		T = table
		t = row type

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 type NCHAR, 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 English text. A space separates the two parts.

If the data type of the column is not NCHAR or NVARCHAR, a binary representation is encoded in the **default** column. A composite index for the **tabid**, **colno**, and **class** columns allows only unique values.

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** system catalog table has the following columns.

Column Name	Type	Explanation
btabid	INTEGER	Table identifier of base table or view

btype	NCHAR(1)	Base object type:
		T = Table
		V = View

dtabid	INTEGER	Table identifier of dependent table
dtype	NCHAR(1)	Dependent object type (V = View); currently, only view is implemented

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

SYSDISTRIB

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

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

The **sysdistrib** system catalog table has the following columns.

Column Name	Type	Explanation
tabid	INTEGER	Table identifier of the table where data was gathered
colno	SMALLINT	Column number in the source table
seqno	INTEGER	Sequence number for multiple entries
constructed	DATE	Date when the data distribution was created

mode	NCHAR(1)	Optimization level:
		M = Medium
		H = High

resolution	FLOAT	Specified in the UPDATE STATISTICS statement
confidence	FLOAT	Specified in the UPDATE STATISTICS statement
encdat	STAT	Statistics information
type	NCHAR(1)	Type of statistics: A = **encdat** has ASCII-encoded histogram in fixed-length character field S = **encdat** has-user defined statistics
udtstat	STAT	UDT statistics information

You can select any column from **sysdistrib** except **encdat** and **udtstat**. Only user **informix** can select the **encdat** and **udtstat** columns.

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.

For UDT columns, the **type** field is set to S. The **encdat** column stores the statistics collected by the statcollect UDR in multirepresentational form. Only one row is stored for each **tabid** and **colno** pair.

IDS

The **sysdistrib** system catalog table supports extensions for user-defined statistics in Dynamic Server only. ♦

IDS

SYSERRORS

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.

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

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

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

XPS

SYSEXTCOLS

The **sysextcols** system catalog table contains a row that describes each of the internal columns in external table tabid of format type (fmttype) FIXED. No entries are stored in **sysextcols** for DELIMITED or Informix-format external files.

Column	Type	Description
tabid	INTEGER	Table identifier
colno	SMALLINT	Column identifier
exttype	SMALLINT	External column type
extstart	SMALLINT	Starting position of column in the external data file
extlength	SMALLINT	External column length in bytes
nullstr	NCHAR(256)	Represents null in external data
picture	NCHAR(256)	Reserved for future use
decimal	SMALLINT	Precision for external decimals
extstype	NCHAR(18)	The external type name

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

XPS

SYSEXTDFILES

For each external table, at least one row exists in the **sysextdfiles** system catalog table.

Column	Type	Description
tabid	INTEGER	Table identifier
dfentry	NCHAR(152)	Data file entry

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

XPS

SYSEXTERNAL

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	Description
tabid	INTEGER	Table identifier
fmttype	NCHAR(1)	'D' (delimiter), 'F' (fixed), 'I' (Informix)
recdelim	NCHAR(4)	The record delimiter
flddelim	NCHAR(4)	The field delimiter
codeset	NCHAR(18)	ASCII, EBCDIC
datefmt	NCHAR(8)	Reserved for future use
moneyfmt	NCHAR(20)	Reserved for future use
maxerrors	INTEGER	Number of errors to allow per coserver
relectfile	NCHAR(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 catalogs about an external table, use the tabid as stored in **systables** with tabtype = 'E'.

IDS

SYSFRAGAUTH

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

The **sysfragauth** system catalog table has the following columns.

Column Name	Type	Explanation
grantor	NCHAR(32)	Grantor of privilege
grantee	NCHAR(32)	Grantee of privilege
tabid	INTEGER	Table identifier of the table that contains the fragment named in the **fragment** column.
fragment	NVARCHAR (128)	Name of dbspace where fragment is stored. Identifies the fragment on which privileges are granted.

fragauth	NCHAR(6)	A 6-byte pattern that specifies fragment-level privileges (including 3 bytes reserved for future use). This pattern contains one or more of the following codes:
		u = Update
		i = Insert
		d = Delete

If a code in the **fragauth** column is lowercase (such as u for Update), the grantee cannot grant the privilege to other users. If a code in the **fragauth** column is uppercase (such as U for Update), the grantee can grant the privilege to other users.

A composite index for 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** system catalog table. 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 Name	Type	Explanation

fragtype	NCHAR(1)	Fragment type:
		I = Index
		T = Table
		B = TEXT or BYTE data (Extended Parallel Server)
		i = Index fragments of a duplicated table (Extended Parallel Server)
		d = data fragments of a duplicated table (Extended Parallel Server)

tabid	INTEGER	Table identifier
indexname	NVARCHAR(128)	Index identifier

(1 of 3)

Column Name	Type	Explanation
colno	INTEGER	TEXT or BYTE column identifier Replica identifier (Extended Parallel Server)
partn	INTEGER	Physical location identifier
strategy	NCHAR(1)	Distribution scheme type: R = Round-robin strategy was used to distribute the fragments E = Expression-based strategy was used to distribute the fragments T = Table-based strategy was used to distribute the fragments I = IN DBSPACE clause specified a specific location as part of the fragmentation strategy H = hash-based strategy was used to distribute the fragments (Extended Parallel Server)
location	NCHAR(1)	Reserved for future use; shows L for local
servername	NVARCHAR(128)	Reserved for future use
evalpos	INTEGER	Position of fragment in the fragmentation list
exprtext	TEXT	Expression that was entered for fragmentation strategy For Extended Parallel Server, contains the names of the columns that are hashed. Contains composite information for hybrid fragmentation strategies; shows hashed columns followed by the fragmentation expression of the dbslice.
exprbin	BYTE	Binary version of expression
exprarr	BYTE	Range-partitioning data used to optimize expression in range-expression fragmentation strategy

(2 of 3)

Column Name	Type	Explanation
flags	INTEGER	Used internally For Extended Parallel Server, a bit value indicates the existence of a hybrid fragmentation strategy (value = 0x10). Also, an additional flag (value = 0x20) will be set on the first fragment of a globally detached index.
dbspace	NVARCHAR(128)	Dbspacename for fragment
levels	SMALLINT	Number of B+ tree index levels
npused	INTEGER	For table-fragmentation strategy, npused represents the number of data pages; for index-fragmentation strategy, npused represents 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 determines the relative position. The first fragment has a **hybdpos** value of zero (0). (Extended Parallel Server)

(3 of 3)

The **strategy** type `T` is used for attached indexes (where index fragmentation is the same as the table fragmentation).

The composite index on the **fragtype**, **tabid**, **indexname**, and **evalpos** columns allows duplicate values.

XPS

The **hybdpos** field is the last field of the composite key in the fraginfo index on the **SYSFRAGMENTS** table.

XPS

SYSINDEXES

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

Column Name	Type	Explanation
idxname	NVARCHAR(18)	Index name
owner	NCHAR(32)	Owner of index (user **informix** for system catalog tables and user name for database tables)
tabid	INTEGER	Table identifier
idxtype	NCHAR(1)	Index type: U = Unique; G = Nonbitmap generalized-key index (Extended Parallel Server only); D = Duplicates; g = Bitmap generalized-key index (Extended Parallel Server only); u = unique, bitmap (Extended Parallel Server only); d = nonunique, bitmap (Extended Parallel Server only)
clustered	NCHAR(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

(1 of 2)

Column Name	Type	Explanation
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
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	Stores the current locking mode of the index:
		Normal = 0x01
		Coarse = 0x02

(2 of 2)

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

Each **part***nth* column component of a composite index (the **part1** through **part16** columns in this table) holds the column number (**colno**) of each part 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 **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.

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

IDS

SYSINDICES

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

Column Name	Type	Explanation
idxname	NVARCHAR(128)	Index name
owner	NCHAR(32)	Owner of index (user **informix** for system catalog tables and user name for database tables)
tabid	INTEGER	Table identifier

idxtype	NCHAR(1)	Index type:
		U = Unique
		D = Duplicates

clustered	NCHAR(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 the UPDATE STATISTICS statement is run on the table.
nrows	INTEGER	Estimated number of rows in the table (zero until UPDATE STATISTICS is run on the table).

(1 of 2)

Column Name	Type	Explanation
indexkeys	INDEXKEYARRAY	This column has a maximum of three fields, displayed in the following form: <function id>(col1, ..., coln) [operator class id] The function id shows only if the index is on return values of a function defined over the columns of the table; that is, if it is a functional index. The function id is the same as the **procid** of the function showing in the **sysprocedures** system catalog table. The list of the columns (col1,..., coln) in the second field gives columns over which the index is defined. The operator class id signifies the particular secondary access method used to build and search the index.
amid	INTEGER	Identifier of the access method used to implement this index. It is the value of the *am_id* for that access method in the **sysams** system catalog table.
amparam	LVARCHAR	List of parameters used to customize the behavior of this access method.

(2 of 2)

Tip: *This system catalog table is changed from the system catalog table in the 7.2 version of the Informix database server. The previous version of this system catalog table is still available as a view and can be accessed under its original name:* ***sysindexes.*** *The columns* ***part1*** *through* ***part16*** *in* ***sysindexes*** *are filled in for B-tree indexes that do not use user-defined types or functional indexes. For generic B-trees and all other access methods, the* ***part1*** *to* ***part16*** *columns contain zeros.*

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

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

The system indexes **tabid** and **idxname** are used to index **sysindices**.

IDS

SYSINHERITS

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

Column Name	Type	Explanation
child	INTEGER	Identifier of the subtable or subtype in an inheritance relationship
parent	INTEGER	Identifier of the supertable or supertype in an inheritance relationship

class	NCHAR(1)	Inheritance class:
		t = named row type
		T = table

IDS

SYSLANGAUTH

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

Column Name	Type	Explanation
grantor	NCHAR(32)	The grantor of the language authorization
grantee	NCHAR(32)	The grantee of the language authorization
langid	INTEGER	The language id reference to the **sysroutinelangs** system catalog table

langauth	NCHAR(1)	The language authorization
		u = Usage permission granted
		U = Usage permission granted WITH GRANT OPTION

A nonunique index on the **langid** and **grantee** columns is created for faster access to the **syslangauth** table.

SYSLOGMAP

The **syslogmap** system catalog table is not implemented in this version.

Column Name	Type	Explanation
tabloc	Reserved for future use	
tabid	Reserved for future use	
fragid	Reserved for future use	
flags	Reserved for future use	

XPS

SYSNEWDEPEND

The **sysnewdepend** system catalog table contains information about generalized-key indexes that is not available in the **sysindexes** system catalog 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** system catalog table. The **sysnewdepend** system catalog table has the following columns.

Column Name	Type	Explanation
scrid1	CHAR	The name of the generalized-key index
scrid2	INTEGER	The tableid of the indexed table
type	INTEGER	The type of generalized-key index
destid1	INTEGER	The tableid of the table that the generalized-key index depends
destid2	INTEGER	The column name in the table that the generalized-key index depends

SYSOBJSTATE

The **sysobjstate** system catalog table stores information about the state (object mode) of database objects. The types of database objects 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** system catalog table has the following columns.

Column Name	Type	Explanation

objtype	NCHAR(1)	The type of database object. This column has one of the following codes:
		C = Constraint
		I = Index
		T = Trigger

owner	NCHAR(32)	The owner of the database object
name	NVARCHAR(128)	The name of the database object
tabid	INTEGER	Table identifier of the table on which the database object is defined

state	NCHAR(1)	The current state (object mode) of the database object. This column has 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 for the **objtype**, **name**, **owner**, and **tabid** columns allows only unique values.

IDS

SYSOPCLASSES

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. The **sysopclasses** system catalog table has the following columns.

Column Name	Type	Explanation
opclassname	NVARCHAR(128)	Name of the operator class
owner	NCHAR(32)	Owner of the operator class
amid	INTEGER	Identifier of the secondary access method associated with this operator class
opclassid	SERIAL	Identifier of the operator class. This identifier is used in the **sysams** system catalog table to specify the default operator class (**am_defopclass**) for the access method.
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 **sysopclasses** system catalog table has two indexes. There is a composite index on **opclassname** and **owner** columns and an index on **opclassid** column. Both indexes allow only unique values.

IDS

SYSOPCLSTR

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

The **sysopclstr** system catalog table has the following columns.

Column Name	Type	Explanation
owner	NCHAR(32)	Owner of the cluster
clstrname	NVARCHAR(128)	Name of the cluster
clstrsize	INTEGER	Size of the cluster
tabid	INTEGER	Table identifier
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
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

(1 of 2)

Column Name	Type	Explanation
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

(2 of 2)

A composite index for both the **clstrname** and **owner** columns allows only unique values. 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 are granted. The **sysprocauth** system catalog table has the following columns.

Column Name	Type	Explanation
grantor	NCHAR(32)	Grantor of routine
grantee	NCHAR(32)	Grantee of routine
procid	INTEGER	Routine identifier
procauth	NCHAR(1)	Type of routine permission granted: e = Execute permission on routine E = Execute permission and the ability to grant it to others

A composite index for the **procid**, **grantor**, and **grantee** columns allows only unique values. The composite index for 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** system catalog table stores the text of the routine, each routine can have multiple rows. The **sysprocbody** system catalog table has the following columns.

Column Name	Type	Explanation
procid	INTEGER	Routine identifier
datakey	NCHAR(1)	Data-descriptor type:
		D = User document text
		T = Actual routine source
		R = Return value type list
		S = Routine symbol table
		L = Constant routine data string (that is, literal numbers or quoted strings)
		P = Interpreter instruction code
seqno	INTEGER	Line number of the routine
data	NCHAR(256)	Actual text of the routine

Although the **datakey** column indicates the type of data that is stored, the **data** column contains the actual data, which can be one of the following types:

- Encoded return values list
- Encoded symbol table
- Constant data
- Compiled code for the routine
- Text of the routine and its documentation

A composite index for 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.

XPS

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

Column Name	Type	Explanation
procname	NVARCHAR(128)	Routine name
owner	NCHAR(32)	Owner name
procid	SERIAL	Routine identifier
mode	NCHAR(1)	Mode type: D, d = DBA O, o = Owner P, p = Protected R, 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) constant data
codesize	INTEGER	Compiled size (in bytes) of routine instruction code
numargs	INTEGER	Number of routine arguments

IDS

For Dynamic Server, the **sysprocedures** system catalog table has the following columns.

Column Name	Type	Explanation
procname	NVARCHAR(128)	Routine name
owner	NCHAR(32)	Owner name
procid	SERIAL	Routine identifier
mode	NCHAR(1)	Mode type:
		D, d = DBA
		O, o = Owner
		P, p = Protected
		R, 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) constant data
codesize	INTEGER	Compiled size (in bytes) of routine instruction code
numargs	INTEGER	Number of routine arguments
isproc	NCHAR(1)	Whether the routine is a procedure or a function
		`t` = procedure
		`f` = function
specificname	NVARCHAR(128)	The specific name for the routine
externalname	NVARCHAR(255)	Location of the external routine. This item is language-specific in content and format.
paramstyle	NCHAR(1)	Parameter style
		`I` = Informix

(1 of 3)

Column Name	Type	Explanation
langid	INTEGER	Language identifier (from the **sysroutinelangs** system catalog table)
paramtypes	rtnparamtypes	Data types of the parameters; *rtnparamtypes* indicates a built-in data type
variant	BOOLEAN	Indicates whether the routine is VARIANT or not `t` = is variant `f` = is not variant
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	NVARCHAR(128)	Field commutator
negator	NVARCHAR(128)	Negator function name
selfunc	NVARCHAR(128)	Function used to estimate function selectivity
iterator	BOOLEAN	Iterator routine
internal	BOOLEAN	Whether the routine can be called from SQL `t` = routine is internal, not callable from SQL `f` = routine is external, can be called from SQL
class	NCHAR(18)	CPU class in which the routine should be executed
stack	INTEGER	Stack size in bytes required per invocation
costfunc	NVARCHAR(128)	Name of cost function for UDR
selconst	INTEGER	Selectivity constant for UDR

(2 of 3)

Column Name	Type	Explanation

parallelizable	BOOLEAN	Parallelization indicator for UDR:
		t = Parallelizable
		f = Not parallelizable

(3 of 3)

♦

A unique index on the **procid** column indexes the routine id. A composite index on the **procname**, **isproc**, **numargs**, and **owner** columns maintains the uniqueness of the routines in the database. This index allows duplicate values. The index on the **specificname** and **owner** columns maintains the uniqueness of the routines with specific names. It allows duplicate values.

For the **sysprocedures** system catalog table, 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 that is different from the routine creator. If routine statements involving a remote database are executed, the database server uses the permissions of the user that 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.

Starting with Version 9.x, protected routines (which cannot be deleted) are indicated differently in the mode column. In earlier versions, protected routines were simply indicated by a P. Currently, protected 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.

Important: *After a SET SESSION AUTHORIZATION is done, all owner routines created while using the new identity are given a restricted mode.*

A 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 or display them through **dbschema**.

SYSPROCPLAN

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

It is possible to delete all the plans for a particular routine with the DELETE statement on **sysprocplan**. When the routine is executed, new plans are automatically generated and recorded in **sysprocplan**.

IDS

Only Dynamic Server stores plans in **sysprocplan**. ♦

The **sysprocplan** system catalog table has the following columns.

Column Name	Type	Explanation
procid	INTEGER	Routine identifier
planid	INTEGER	Plan identifier

datakey	NCHAR(1)	Identifier routine plan part:
		D = Dependency list
		Q = Execution plan

seqno	INTEGER	Line number of plan
created	DATE	Date plan created
datasize	INTEGER	Size (in bytes) of the list or plan
data	NCHAR(256)	Encoded (compiled) list or plan

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

SYSREFERENCES

The **sysreferences** system catalog table lists the referential constraints that are placed on columns in the database. It contains a row for each referential constraint in the database. The **sysreferences** system catalog table has the following columns.

Column Name	Type	Explanation
constrid	INTEGER	Constraint identifier
primary	INTEGER	Constraint identifier of the corresponding primary key
ptabid	INTEGER	Table identifier of the primary key
updrule	NCHAR(1)	Reserved for future use; displays an R
delrule	NCHAR(1)	Displays cascading delete or restrict rule:
		C = Cascading delete
		R = Restrict (default)
matchtype	NCHAR(1)	Reserved for future use; displays an N
pendant	NCHAR(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.

XPS

SYSREPOSITORY

The **sysrepository** system catalog table contains information about generalized-key indexes that is not available in the **sysindexes** system catalog table. The **sysrepository** system catalog table contains the CREATE statement for each generalized-key index in the database in its **desc** column. The contents of the **sysrepository** system catalog 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.

Column Name	Type	Explanation
id1	NCHAR	Index from the generalized-key index
id2	INTEGER	Tabid of table with the generalized-key index
type	INTEGER	Integer representing object type In this release, the only integer that shows is 1, indicating generalized-key index type.
seqid	INTEGER	For future use
desc	TEXT	The CREATE statement used for each generalized-key index in the database
bin	BYTE	Internal representation of the generalized-key index

IDS

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** system catalog table has the following columns.

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

The **rolename** and **grantee** columns are indexed and allow only unique values. The **is_grantable** column indicates whether the role was granted with the WITH GRANT OPTION on the GRANT statement.

IDS

SYSROUTINELANGS

The **sysroutinelangs** system catalog table contains the supported languages for writing user-defined routines (UDRs).

Column Name	Type	Explanation
langid	SERIAL	Identifies the supported language
langname	NCHAR(30)	The name of the language, such as C or SPL
langinitfunc	NVARCHAR(128)	The name of the initialization function for the language
langpath	NCHAR(255)	The path for the UDR language
langclass	NCHAR(18)	The class of the UDR language

SYSSYNONYMS

Important: *Version 4.0 or later Informix products no longer use this table; however, any* ***syssynonyms*** *entries made before Version 4.0 remain in this table. See the discussion of* ***syssyntable****.*

The **syssynonyms** system catalog table lists the synonyms for each table or view. It contains a row for every synonym defined in the database. The **syssynonyms** system catalog table has the following columns.

Column Name	Type	Explanation
owner	NCHAR(32)	User name of owner
synname	NVARCHAR(128)	Synonym identifier
created	DATE	Date synonym created
tabid	INTEGER	Table identifier

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

SYSSYNTABLE

The **syssyntable** system catalog table outlines the mapping between each synonym and the object that it represents. It contains one row for each entry in the **systables** table that has a **tabtype** of S. The **syssyntable** system catalog table has the following columns.

Column Name	Type	Explanation
tabid	INTEGER	Table identifier
servername	NVARCHAR(128)	Server name
dbname	NVARCHAR(128)	Database name
owner	NCHAR(32)	User name of owner
tabname	NVARCHAR(128)	Name of table
btabid	INTEGER	Table identifier of base table or view

If you define a synonym for a table 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 **tabname** columns are used.

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

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

IDS

SYSTABAMDATA

The **systabamdata** system catalog table stores the parameterization option choices (table-specific hashing parameters) that you make when you create a table using a primary access method. It stores configuration parameters that determine how a primary access method accesses a particular table. The table might reside in a cooked file, a different database, or an sbspace within the database server.

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

Column Name	Type	Explanation
tabid	INTEGER	Table identifier
am_param	NCHAR(256)	Access method parameterization option choices
am_space	NVARCHAR(128)	The name of the space where the table data is stored

The **tabid** column, the key to the **systables** system catalog table, is indexed and must contain unique values. Each configuration parameter in the **am_param** list has the format *keyword=value* or *keyword*.

SYSTABAUTH

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

Column Name	Type	Explanation
grantor	NCHAR(32)	Grantor of privilege
grantee	NCHAR(32)	Grantee of privilege
tabid	INTEGER	Table identifier
tabauth	NCHAR(9)	9-byte pattern that specifies table privileges:
		s = Select
		u = Update
		* = Column-level privilege
		i = Insert
		d = Delete
		x = Index
		a = Alter
		r = References
		n = Under privilege
		N = Under privilege with grant option

If the **tabauth** privilege code is uppercase (for example, `S` for select), a user who has this privilege can grant it to others. If the **tabauth** privilege code is lowercase (for example, `s` for select), the user who has this privilege cannot grant it to others.

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

SYSTABLES

The **systables** system catalog table describes each table in the database. It contains one row for each table, view, or synonym that is defined in the database. The information in the **systables** system catalog table includes all database tables and the system catalog tables. The **systables** system catalog table has the following columns

Column Name	Type	Explanation
tabname	NVARCHAR(128)	Name of table, view, or synonym
owner	NCHAR(32)	Owner of table (user **informix** for system catalog tables and user name for database tables)
partnum	INTEGER	Physical location identifier
tabid	SERIAL	System-assigned sequential ID number (system tables: 1-24, user tables: 100-nnn)
rowsize	SMALLINT	Row size
ncols	SMALLINT	Number of columns
nindexes	SMALLINT	Number of indexes
nrows	INTEGER	Number of rows
created	DATE	Date created
version	INTEGER	Number that changes when table is altered

tabtype	NCHAR(1)	Table type:
		T = Table
		V = View
		P = Private synonym
		S = Public synonym (not available in an ANSI-compliant database)

(1 of 2)

Column Name	Type	Explanation

locklevel	NCHAR(1)	Lock mode for a table:
		B = Page
		P = Page
		R = Row
		T = Table (Extended Parallel Server only)

npused	INTEGER	Number of data pages in use
fextsize	INTEGER	Size of initial extent (in kilobytes)
nextsize	INTEGER	Size of all subsequent extents (in kilobytes)
flags	SMALLINT	Has a unique value for the following types of permanent tables: ST_RAW represents a raw (nonlogging permanent) table in a logging database RAW 0x00000002 (Extended Parallel Server only) STATIC 0x00000004 (Extended Parallel Server only) OPERATIONAL 0x00000010 (Extended Parallel Server only) STANDARD 0x00000020 (Extended Parallel Server only) EXTERNAL 0x00000020 (Extended Parallel Server only)
site	NVARCHAR(128)	Reserved for future use (used to store database collation and C-type information)
dbname	NVARCHAR(128)	Reserved for future use
am_id	INTEGER	Access method ID (key to **sysams** table); null value or 0 indicates built-in storage manager used

(2 of 2)

Each table recorded in the **systables** system catalog table is assigned a **tabid**, which is a system-assigned sequential number that uniquely identifies each table in the database. The first **tabid** numbers up to 99 are reserved for system catalog tables. The user-created **tabid** numbers begin with 100.

The **tabid** column is indexed and must contain unique values. A composite index for the **tabname** and **owner** columns allows only unique values. The **version** column contains an encoded number that is put in the **systables** system catalog table when the table is created. Portions of the encoded value are incremented when data-definition statements, such as ALTER INDEX, ALTER TABLE, DROP INDEX, and CREATE INDEX, are performed.

When a prepared statement is executed, the **version** number is checked to make sure that nothing has changed since the statement was prepared. If the **version** number has changed, your statement does not execute, and you must prepare your statement again.

The **npused** column does not reflect BYTE or TEXT data used.

GLS

The **systables** system catalog table has two additional rows to store 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 tabname, tabid FROM systables
```

IDS

SYSTRACECLASSES

The **systraceclasses** system catalog table contains the names and identifiers of trace classes. 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** system catalog table. By default, all users can view this table, but only users with the DBA privilege can modify it.

A unique index on the **name** column ensures that each trace class has a unique name. The database server also assigns each class a sequential identifier. Therefore, the index on the **classid** column also allows only unique values.

The **systraceclasses** system catalog table has the following columns.

Column Name	Type	Explanation
name	NCHAR(18)	Name of the class of trace messages
classid	SERIAL	Trace class identifier

IDS

SYSTRACEMSGS

The **systracemsgs** system catalog table stores internationalized trace messages that you can use in debugging user-defined routines. DataBlade module developers create a trace message by inserting a row directly into the **systracemsgs** system catalog table. Once a message is created, the development team can specify it either by name or by ID, 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 **systracemsgs** table, but only users with the DBA privilege can modify it.

The **systracemsgs** system catalog table has the following columns.

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

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

IDS

SYSTRIGBODY

The **systrigbody** system catalog table contains the English 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.

Warning: *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** system catalog table has the following columns.

Column Name	Type	Explanation
trigid	INT	Trigger identifier
datakey	NCHAR	Type of data:
		D = English text for the header, trigger definition
		A = English text for the body, triggered actions
		H = Linearized code for the header
		S = Linearized code for the symbol table
		B = Linearized code for the body
seqno	INT	Sequence number
data	NCHAR(256)	English text or linearized code

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

IDS

SYSTRIGGERS

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

Column Name	Type	Explanation
trigid	SERIAL	Trigger identifier
trigname	NVARCHAR(128)	Trigger name
owner	NCHAR(32)	Owner of trigger
tabid	INT	ID of triggering table
event	NCHAR	Triggering event:
		I = Insert trigger
		U = Update trigger
		D = Delete trigger
		S = Select trigger
old	NVARCHAR(128)	Name of value before update
new	NVARCHAR(128)	Name of value after update
mode	NCHAR	Reserved for future use

A composite index for the **trigname** and **owner** columns allows only unique values. The **trigid** column is indexed and must contain unique values. An index for the **tabid** column allows duplicate values.

SYSUSERS

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

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

The **username** column is indexed and allows only unique values. The **username** can be the name of a role.

SYSVIEWS

The **sysviews** system catalog table describes each view that is defined in the database. Because the **sysviews** system catalog table stores the SELECT statement that you use to create the view, it can contain multiple rows for each view in the database. The **sysviews** system catalog table has the following columns.

Column Name	Type	Explanation
tabid	INTEGER	Table identifier
seqno	SMALLINT	Line number of the SELECT statement
viewtext	NCHAR(64)	Actual SELECT statement used to create the view

A composite index for the **tabid** and **seqno** columns 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. The **sysviolations** system catalog table has the following columns.

Column Name	Type	Explanation
targettid	INTEGER	Table identifier of the target table. The target table is the base table on which the violations table and the diagnostic table are defined.
viotid	INTEGER	Table identifier of the violations table
diatid	INTEGER	Table identifier of the diagnostics table
maxrows	INTEGER	Maximum number of rows that can be inserted in the diagnostics table during a single insert, update, or delete operation on a target table that has a filtering mode object defined on it.
		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 sets a disabled object to filtering mode (provided that a diagnostics table exists for the target table).
		For Extended Parallel Server, indicates the maximum number of rows that are allowed in the violations table for each coserver.
		If no maximum is specified for the diagnostics or violations table, this column contains a null value.

The primary key of the **sysviolations** table is the **targettid** column. Unique indexes are also defined on the **viotid** and **diatid** columns.

XPS

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. ♦

IDS

SYSXTDDESC

The **sysxtddesc** system catalog table provides a text description of each user-defined data type (opaque, distinct, and complex (named row types, unnamed row types, and collection types)) that you define in the database. The **sysxtddesc** system catalog table has the following columns.

Column Name	Type	Explanation
extended_id	SERIAL	Unique identifier for extended data types
seqno	SMALLINT	Value to order and identify one line of description for specific values of **extended_id**. A new sequence is created only if the text string is larger than the 255 limit of the text string.
description	NCHAR(256)	Textual description of the extended data type

IDS

SYSXTDTYPEAUTH

The **sysxtdtypeauth** system catalog table provides privileges for each user-defined data type (opaque and distinct) and for each named row type that you define in the database. The table contains one row for each set of privileges granted.

The **sysxtdtypeauth** system catalog table has the following columns.

Column Name	Type	Explanation
grantor	NCHAR(32)	Grantor of privilege
grantee	NCHAR(32)	Grantee of privilege
type	INTEGER	Identifier of the user-defined type
auth	NCHAR(2)	Privileges on the user-defined data type: n = Under privilege N = Under privilege with grant option u = Usage privilege U = Usage privilege with grant option

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

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

IDS

SYSXTDTYPES

The **sysxtdtype** system catalog table has an entry for each user-defined data type (opaque and distinct data types) and complex data type (named row type, unnamed row type, and collection type) that is defined in the database. Each extended data type has a unique id, called an extended id (**extended_id**), a data type identifier (**type**), and the length and description of the data type.

The **sysxtdtypes** system catalog table has the following columns.

Column Name	Type	Explanation
extended_id	SERIAL	Unique identifier for extended data types

mode	NCHAR(1)	Detailed description of the user-defined type. One of the following values:
		B = Base (opaque)
		C = Collection type as well as unnamed row type
		D = Distinct
		R = Named row type
		' ' (blank) = Built-in type

owner	NCHAR(32)	Owner of the data type
name	NVARCHAR(128)	Name of the data type
type	SMALLINT	The identifier of the data type. See the coltype column values of the syscolumns system catalog table for a complete list of identifiers associated with different data types. For distinct types created from built-in data types, the value in this column corresponds with the value of the **coltype** column (indicating the source type) in the **syscolumns** system catalog table. A value of 40 indicates a distinct data type created from a variable-length opaque type. A value of 41 indicates a distinct type created from a fixed-length opaque type.

(1 of 2)

Column Name	Type	Explanation
source	INTEGER	The sysxtdtypes reference for this type, if it is a distinct type. A value of 0 indicates that the distinct type was created from a built-in data type.
maxlen	INTEGER	The maximum length for variable-length data types. A value of 0 indicates a fixed-length data type.
length	INTEGER	The length in bytes for fixed-length data types. A value of 0 indicates a variable-length data type.
byvalue	NCHAR(1)	If the data type is passed by value 'T' = type is passed by value 'F' = type is not passed by value
cannothash	NCHAR(1)	Is data type hashable using the default bit-hashing function? 'T' = type is hashable 'F' = type is not hashable
align	SMALLINT	Alignment for this data type. One of the following values: 1, 2, 4, 8.
locator	INTEGER	Locator (key) for unnamed row types.

(2 of 2)

The index on the **extended_id** column allows only unique values. Similarly the index on columns **name** and **type** columns also allows only unique values. The index on the **source** and **maxlen** columns allow duplicate values.

IDS

Information Schema

The Information Schema consists of read-only views that provide information about all the tables, views, and columns on the current database server to which you have access. In addition, Information Schema views provide information about SQL dialects (such as Informix, Oracle, or Sybase) and SQL standards.

This version of the Information Schema views are X/Open CAE standards. Informix provides them so that applications developed on other database systems can obtain Informix system catalog information without having to use the Informix system catalogs directly.

Important: *Because the X/Open CAE standards Information Schema views differ from ANSI-compliant Information Schema views, Informix recommends 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**

The following sections 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
```

Data in the Informix system catalog tables populates the views. If tables, views, or routines exist with any of the same names as the Information Schema views, you need to either rename the 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. Informix recommends 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 views are described in this section:

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

Most of the columns in the views are defined as VARCHAR data types with large maximums to accept large names and in anticipation of long identifier names in future standards.

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 Name	Data Type	Explanation
table_schema	VARCHAR(128)	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

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 these privileges are granted to PUBLIC, you see one 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 Name	Data Type	Explanation
table_schema	VARCHAR(128)	Owner of table
table_name	VARCHAR(128)	Name of table or view
column_name	VARCHAR(128)	Name of the column of the table or view
ordinal_position	INTEGER	Ordinal position of the column. The ordinal_position of a column in a table is a sequential number that starts at 1 for the first column.This column is an Informix extension to XPG4.
data_type	VARCHAR(254)	Data type of the column, such as CHARACTER or DECIMAL

(1 of 2)

Column Name	Data Type	Explanation
char_max_length	INTEGER	Maximum length for character data types; null otherwise
numeric_precision	INTEGER	Total number of digits allowed for exact numeric data types (DECIMAL, INTEGER, MONEY, and SMALLINT), and the number of digits of mantissa precision for approximate data types (FLOAT and SMALLFLOAT), and null for all other data types. The value is machine dependent for FLOAT and SMALLFLOAT.
numeric_prec_radix	INTEGER	Uses one of the following values: 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 data 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 nulls; either YES or NO
remarks	VARCHAR(254)	Reserved

(2 of 2)

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** Information Schema view contains the following columns.

Column Name	Data Type	Explanation
source	VARCHAR(254)	Organization that defines this SQL version
source_year	VARCHAR(254)	Year the source document was approved
conformance	VARCHAR(254)	Which conformance is supported
integrity	VARCHAR(254)	Indicates whether this is an integrity enhancement feature; either YES or NO
implementation	VARCHAR(254)	Identifies the SQL product of the vendor
binding_style	VARCHAR(254)	Direct, module, or other bind style
programming_lang	VARCHAR(254)	Host language for which the binding style is adopted

The **sql_languages** Information Schema 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 the following columns.

Column Name	Data Type	Explanation
server_attribute	VARCHAR(254)	An attribute of the database server
attribute_value	VARCHAR(254)	Value of the **server_attribute** 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 information.

server_attribute	Description
identifier_length	Maximum number of characters for a user-defined name
row_length	Maximum length of a row
userid_length	Maximum number of characters of a user name (or "authorization identifier")
txn_isolation	Initial transaction isolation level that the database server assumes: Read Committed Default isolation level for databases created without logging Read Uncommitted Default isolation level for databases created with logging but not ANSI compliant Serializable Default isolation level for ANSI-compliant databases
collation_seq	Assumed ordering of the character set for the database server. The following values are possible: ISO 8859-1 EBCDIC The Informix representation shows ISO 8859-1.

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

Chapter

Data Types

2

In This Chapter 2-5

Summary of Data Types 2-5

Description of Data Types 2-9
- BLOB 2-9
- BOOLEAN 2-11
- BYTE 2-12
- CHAR(*n*) 2-13
- CHARACTER(*n*) 2-14
- CHARACTER VARYING(*m*,*r*) 2-15
- CLOB 2-15
- DATE 2-16
- DATETIME 2-17
- DEC 2-21
- DECIMAL 2-21
- Distinct 2-23
- DOUBLE PRECISION 2-24
- FLOAT(*n*). 2-24
- INT 2-25
- INT8 2-25
- INTEGER. 2-26
- INTERVAL 2-26
- LIST(*e*) 2-30
- LVARCHAR 2-31
- MONEY(*p*,*s*) 2-32
- MULTISET(*e*) 2-33
- Named Row 2-34
- NCHAR(*n*) 2-34

NUMERIC(*p,s*) 2-34
NVARCHAR(*m,r*) 2-34
Opaque 2-35
REAL 2-35
Row, Named 2-36
Row, Unnamed 2-37
SERIAL(*n*) 2-39
SERIAL8 2-40
SET(*e*) 2-42
SMALLFLOAT 2-43
SMALLINT 2-44
TEXT 2-44
Unnamed Row 2-46
VARCHAR(*m,r*). 2-46

Built-In Data Types 2-48
Large-Object Data Types. 2-49
Simple Large Objects 2-50
Smart Large Objects 2-50
Time Data Types 2-52
Manipulating DATETIME Values 2-53
Manipulating DATETIME with INTERVAL Values 2-54
Manipulating DATE with DATETIME and INTERVAL Values 2-55
Manipulating INTERVAL Values 2-57
Multiplying or Dividing INTERVAL Values. 2-58

Extended Data Types 2-58
Complex Data Types 2-59
Collection Data Types 2-60
Row Data Types 2-61
Distinct Data Types 2-62
Opaque Data Types 2-62

Data Type Casting and Conversion 2-63
Using Built-in Casts 2-64
Converting from Number to Number 2-65
Converting Between Number and CHAR 2-65
Converting Between INTEGER and DATE or DATETIME . . . 2-66
Converting Between DATE and DATETIME 2-66

Using User-Defined Casts 2-67
Implicit Casts. 2-67
Explicit Casts 2-68
Determining Which Cast to Apply 2-68
Casts for Distinct Types 2-69
What Extended Data Types Can Be Cast? 2-70

Operator Precedence 2-71

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.

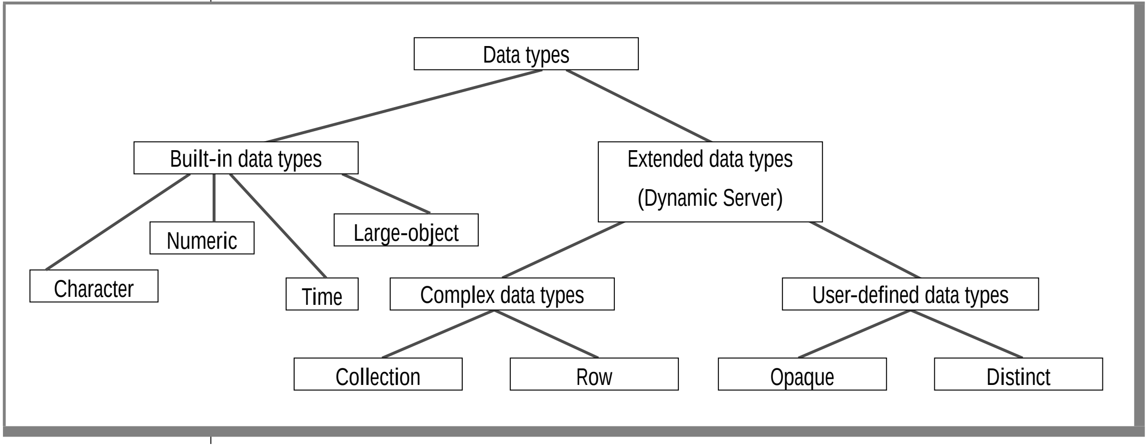
This chapter covers the following topics:

- Built-in data types
- IDS Extended data types ♦
- Casting or converting between two data types
- Operator precedence

Summary of Data Types

Figure 2-1 charts the categories of data types that Informix database servers support.

Figure 2-1 *Overview of Supported Data Types*



Built-in data types and extended data types share the following characteristics. You can:

- use them to create columns of tables.
- use them as arguments and as return types of functions.
- use them to create distinct types.
- cast them to other data types.
- declare and access them with SPL and ESQL/C.

Specific exceptions are mentioned in the description of each data type. For an overview, see "Built-In Data Types" on page 2-47 and "Extended Data Types" on page 2-57.

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, SQL statements that create specific data types and create and drop casts, and other data type syntax conventions, refer to the *Informix Guide to SQL: Syntax*.

IDS

For information about how to create and use complex data types, see the *Informix Guide to Database Design and Implementation*.

For information about how to create user-defined data types, see *Extending Informix Dynamic Server 2000*. ♦

All Informix database servers support the data types that Figure 2-2 lists. This chapter describes each of these data types.

Figure 2-2
Data Types That All Informix Database Servers Support

Data Type	Explanation	Page
BYTE	Stores any kind of binary data	2-12
CHAR(*n*)	Stores single-byte or multibyte sequences of characters, including letters, numbers, and symbols; collation is code-set dependent	2-13
CHARACTER(*n*)	Is a synonym for CHAR	2-14

(1 of 3)

Data Type	Explanation	Page
CHARACTER VARYING(*m*,*r*)	Stores single-byte and multibyte sequences of characters, including letters, numbers, and symbols of varying length (ANSI compliant); collation is code-set dependent	2-15
DATE	Stores calendar date	2-16
DATETIME	Stores calendar date combined with time of day	2-17
DEC	Is a synonym for DECIMAL	2-21
DECIMAL	Stores numbers with definable scale and precision	2-21
DOUBLE PRECISION	Behaves the same way as FLOAT	2-24
FLOAT(*n*)	Stores double-precision floating-point numbers corresponding to the **double** data type in C	2-24
INT	Is a synonym for INTEGER	2-25
INTEGER	Stores whole numbers from –2,147,483,647 to +2,147,483,647	2-25
INTERVAL	Stores a span of time	2-26
MONEY(*p*,*s*)	Stores currency amount	2-30
MULTISET(*e*)	Stores a collection (all elements of same element type, *e*) of values; allows duplicate values.	2-32
NCHAR(*n*)	Stores single-byte and multibyte sequences of characters, including letters, numbers, and symbols; collation is locale dependent	2-33
NUMERIC(*p*,*s*)	Is a synonym for DECIMAL	2-33
NVARCHAR(*m*,*r*)	Stores single-byte and multibyte sequences of characters, including letters, numbers, and symbols of varying length; collation is locale dependent	2-33
REAL	Is a synonym for SMALLFLOAT	2-34
Row, Named	Stores a named row type	2-35
SERIAL	Stores sequential integers	2-38

(2 of 3)

Data Type	Explanation	Page
SERIAL8	Stores large sequential integers; has same range as INT8	2-39
SMALLFLOAT	Stores single-precision floating-point numbers corresponding to the **float** data type in C	2-42
SMALLINT	Stores whole numbers from –32,767 to +32,767	2-43
TEXT	Stores any kind of text data.	2-43
VARCHAR(*m*,*r*)	Stores single-byte and multibyte strings of letters, numbers, and symbols of varying length; collation is code-set dependent.	2-45

(3 of 3)

IDS

Dynamic Server also supports the data types that Figure 2-3 lists. This chapter describes each of these data types.

Figure 2-3
Additional Data Types that Dynamic Server Supports

Data Type	Explanation	Page
BLOB	Stores binary data in random-access chunks	2-9
BOOLEAN	Stores Boolean values `true` and `false`	2-11
CLOB	Stores text data in random-access chunks	2-15
Distinct	Is a user-defined data type that is stored in the same way as the source data type on which it is based but has different casts and functions defined over it than those on the source type	2-23
INT8	Stores an 8-byte integer value. These whole numbers can be in the range $-(2^{63}-1)$ to $2^{63}-1$.	2-25
LIST(*e*)	Stores a collection of elements of the same element type, *e*; elements have an implicit order (first, second, and so on); allows duplicate values	2-29
LVARCHAR	Stores variable-length data that can be larger than 255 bytes	2-30

(1 of 2)

Data Type	Explanation	Page
MULTISET(*e*)	Stores a collection of elements of the same element type, *e*; allows duplicate values.	2-32
Opaque	Stores a user-defined data type whose internal structure is inaccessible to the database server	2-33
Row, Named	Stores a named row type	2-35
Row, Unnamed	Stores an unnamed row type	2-36
SERIAL8	Stores large sequential integers; has same range as INT8	2-39
SET(*e*)	Stores a collection of elements of the same element type, *e*; does not allow duplicate values	2-41

(2 of 2)

♦

For information about Informix internal data types that SQL statements support (IMPEX, IMPEXBIN, and SENDRECV), see *Extending Informix Dynamic Server 2000*.

Description of Data Types

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

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 on the contents of a BLOB column. A BLOB column can be up to 4 terabytes in length.

The term *smart large object* refers to BLOB and CLOB data types. Use the CLOB data type (see page 2-15) for random access to text data. For general information about BLOB and CLOB data types, see "Smart Large Objects" on page 2-49.

You can use the following SQL functions to perform some 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 *Informix Guide to SQL: Syntax*.

No casts exist for BLOB data. Therefore, the database server cannot convert data of type BLOB to any other data type. Within SQL, you are limited to the equality (=) comparison operation for BLOB data. 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 (Informix ESQL/C)

If you select a BLOB column using DB-Access, only the phrase *SBlob value* is returned; no actual value is displayed.

IDS

BOOLEAN

The BOOLEAN data type stores single-byte true/false type data. The following table gives internal and literal representations of the BOOLEAN data type.

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

You can compare two BOOLEAN values to determine whether they are equal or not equal. 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 capture the results of an expression. 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 = (column1 < column2)
```

BYTE

The BYTE data type stores any kind of binary data in an undifferentiated byte stream. Binary data typically consists of saved spreadsheets, program load modules, digitized voice patterns, and so on.

The term *simple large objec*t is used to refer to 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. However, you cannot use BYTE data items in arithmetic or string operations or assign literals to BYTE items 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

You can use BYTE objects in a Boolean expression only if you are testing for null values.

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 (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 following example shows:

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

This statement reads the first 75 bytes of the **cat_picture** column associated with the catalog number 10001.

The database server provides a cast to convert BYTE values to BLOB values. For more information, see the *Informix Guide to Database Design and Implementation*.

If you select a BYTE column using the DB-Access Interactive Schema Editor, only the phrase "BYTE value" is returned; no actual value is displayed.

Important: *If you try to return a BYTE column from a subquery, you get an error message even when the BYTE column is not used in a comparison condition or with the IN predicate.*

CHAR(*n*)

The CHAR data type stores any sequence of letters, numbers, and symbols. It can store single-byte and multibyte characters, based on what the chosen locale supports. For more information on multibyte CHARS, see "Multibyte Characters with CHAR" on page 2-14.

A character column has a maximum length *n* bytes, where 1 ≤ *n* ≤ 32,767. If you do not specify *n*, CHAR(1) is assumed. Character columns typically store names, addresses, phone numbers, and so on.

Because the length of this column is fixed, when a character value is retrieved or stored, exactly *n* bytes of data are transferred. If the character string is shorter than *n* bytes, the string is extended with spaces to make up the length. If the string value is longer than *n* bytes, the string is truncated, without the database server raising an exception.

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 the sum of values into a character column, you might experience overflow problems if the character column is too small to hold the value. In this case, the insert fails. However, 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.

CHAR values are compared to other CHAR values by taking the shorter value and padding it on the right with spaces until the values have equal length. Then the two values are compared for the full length. Comparisons use the code-set collation order.

Nonprintable Characters with CHAR

A CHAR value can include tabs, spaces, and other nonprintable characters. However, you must use an application to insert the nonprintable characters into host variables and to insert the host variables into your database. After passing nonprintable characters to the database server, you can store or retrieve the characters. When you select nonprintable characters, fetch them into host variables and display them with your own display mechanism.

The only nonprintable character that you can enter and display with DB-Access is a tab. If you try to display other nonprintable characters with DB-Access, your screen returns inconsistent results.

GLS

Collating CHAR Data

The collation order of the CHAR data type depends on the code set. That is, this data is sorted by the order of characters as they appear in the code set. For more information, see the *Informix Guide to GLS Functionality*.

GLS

Multibyte Characters with CHAR

The database locale must support the multibyte characters that a database uses. If you are storing multibyte characters, make sure to calculate the number of bytes needed. For more information on multibyte characters and locales, see the *Informix Guide to GLS Functionality*.

CHARACTER(*n*)

The CHARACTER data type is a synonym for CHAR.

CHARACTER VARYING(*m,r*)

The CHARACTER VARYING data type stores any multibyte string of letters, numbers, and symbols of varying length, where *m* is the maximum size of the column and *r* is the minimum amount of space reserved for that column.

The CHARACTER VARYING data type complies with ANSI standards; the non-ANSI Informix VARCHAR data type supports the same functionality. See the description of the VARCHAR data type on page 2-45.

IDS

CLOB

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), or Standard Graphic Markup Language (SGML) data.

The term *smart large object* refers to CLOB and BLOB data types. The CLOB data type includes special operations for character strings that are inappropriate for BLOB values. A CLOB column can be up to 4 terabytes in length.

Use the BLOB data type (see page 2-9) for random access to binary data. For general information about the CLOB and BLOB data types, see “Smart Large Objects” on page 2-49.

You can use the following SQL functions to perform some operations on a CLOB column:

- **FILETOCLOB** copies a file into a CLOB column.
- **LOTOFILE** copies a CLOB (or BLOB) 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 *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. 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 (Informix ESQL/C)

For more information and examples for using the CLOB data type, see the *Informix Guide to SQL: Tutorial* and the *Informix Guide to Database Design and Implementation*.

GLS

With GLS, the following rules apply:

- Multibyte CLOB characters must be supported by the database locale.
- The CLOB data type is collated in code-set order.
- For CLOB columns, the database server handles any required code-set conversions for the data.

For more information on database locales, collation order, and codeset conversion, see the *Informix Guide to GLS Functionality*. ♦

DATE

The DATE data type stores the calendar date. DATE data types require 4 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.

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). For the month, Informix products accept a number value `1` or `01` for January, and so on. For the day, Informix products accept a value `1` or `01` that corresponds to the first day of the month, and so on.

If you enter only a two-digit value for the year, how Informix products fill in the century digits depends on how you set the **DBCENTURY** environment variable. For example, if you enter the year value as `99`, whether that year value is interpreted as `1999` or `2099` depends on the setting of your **DBCENTURY** environment variable. If you do not set the **DBCENTURY** environment variable, then your Informix products consider the present century as the default. For information on how to set the **DBCENTURY** environment variable, refer to page 3-33.

GLS

If you specify a locale other than the default locale, you can display culture-specific formats for dates. The locales and the **GL_DATE** and **DBDATE** environment variables affect the display formatting of DATE values. They do not affect the internal format used in a DATE column of a database. To change the default DATE format, set the **DBDATE** or **GL_DATE** environment variable. GLS functionality permits the display of international DATE formats. For more information, see the *Informix Guide to GLS Functionality*. ♦

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.

The DATETIME data type is composed of a contiguous sequence of fields that represents each component of time that you want to record and uses the following syntax:

```
DATETIME largest_qualifier TO smallest_qualifier
```

The *largest_qualifier* and *smallest_qualifier* can be any one of the fields that Figure 2-4 on page 2-18 lists.

Figure 2-4
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 precision. The default precision is 3 digits (a thousandth of a second). To indicate explicitly other precisions, write FRACTION(*n*), where *n* is the desired number of digits from 1 to 5.

A DATETIME column does not need to include all fields from YEAR to FRACTION; it can include a subset of fields or even a single field. For example, you can enter a value of MONTH TO HOUR in a column that is defined as YEAR TO MINUTE, as long as each entered value contains information for a contiguous sequence of fields. You cannot, however, define a column for just MONTH and HOUR; this entry must also include a value for DAY.

If you use the DB-Access TABLE menu, and you do not specify the DATETIME qualifiers, the 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 you enter can contain fewer fields than defined for that column. Acceptable qualifiers for the first and last fields are identical to the list of valid DATETIME fields that Figure 2-4 lists.

Write values for the field qualifiers as integers and separate them with delimiters. Figure 2-5 lists the delimiters that are used with DATETIME values in the U.S. ASCII English locale.

Figure 2-5
Delimiters Used with DATETIME

Delimiter	Placement in DATETIME Expression
Hyphen	Between the YEAR, MONTH, and DAY portions of the value
Space	Between the DAY and HOUR portions of the value
Colon	Between the HOUR and MINUTE and the MINUTE and SECOND portions of the value
Decimal point	Between the SECOND and FRACTION portions of the value

Figure 2-6 shows a DATETIME YEAR TO FRACTION(3) value with delimiters.

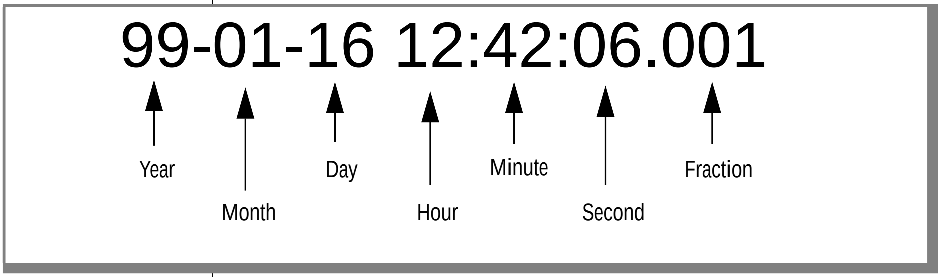


Figure 2-6
Example DATETIME Value with Delimiters

When you enter a value with fewer fields than the defined column, the value that you enter is expanded automatically to fill all the defined fields. If you leave out any more significant fields, that is, fields of larger magnitude than any value that you supply, those fields are filled automatically with the current date. If you leave out any less-significant fields, those fields are filled with zeros (or a 1 for MONTH and DAY) in your entry.

You can also enter DATETIME values as character strings. However, 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, '1999-01-14 08:45', 'maryj', 'D',
      'Order late - placed 6/1/98')
```

In this example, the **call_dtime** column is defined as DATETIME YEAR TO MINUTE. This character string must include values for the year, month, day, hour, and minute fields. If the character string does not contain information for all defined fields (or adds additional fields), 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 century digits are filled in and interpreted depends on the value that you assign to the **DBCENTURY** environment variable.

For example, if you enter `99` as the year value, whether the year is interpreted as `1999` or `2099` depends on the setting of the **DBCENTURY** environment variable. If you do not set the **DBCENTURY** environment variable, then your Informix products consider the present century to be the default. For information on how to set and use the **DBCENTURY** environment variable, see page 3-33.

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). This data value requires 5, or (8/2) + 1, bytes of storage.

For information on how to use DATETIME data in arithmetic and relational expressions, see "Manipulating DATE with DATETIME and INTERVAL Values" on page 2-53. For more information on the DATETIME data type, see the *Informix Guide to SQL: Syntax* and the *Informix Guide to GLS Functionality*.

GLS

If you specify a locale other than U.S. ASCII 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. Locales and the **GL_DATE** and **DBDATE** environment variables affect the display of datetime data. They do not affect the internal format used in a DATETIME column.

For information on how the USEOSTIME configuration parameter can affect the subsecond granularity when the database server obtains the current time from the operating system for SQL statements, see the *Administrator's Reference*.

For more information on **DBTIME**, see page 3-54. For more information on locales and GLS environment variables, see the *Informix Guide to GLS Functionality*. ♦

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, all DECIMAL numbers are fixed point.

DECIMAL 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 do not specify the precision (*p*), DECIMAL is treated as DECIMAL(16), a floating decimal with a precision of 16 places. DECIMAL(*p*) has an absolute exponent range between 10^{-130} and 10^{124}.

If you use an ANSI-compliant database and specify DECIMAL(*p*), the value defaults to DECIMAL(*p*, 0). For more information about fixed-point decimal values, see the following discussion.

DECIMAL 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 write its precision (*p*) as the total number of digits that it can store, from 1 to 32. You write its *scale* (*s*) as the total number of digits that fall 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 variable of this type that you can store without an error is $10^{p-s} - 10^{-s}$. A DECIMAL data type 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 be in the range 10^{-32} to 10^{+31}.

DECIMAL Storage

The database server uses 1 byte of disk storage to store two digits of a decimal number. The database server uses an additional byte to store the exponent and sign. The significant digits to the left of the decimal and the significant digits to the right of the decimal are stored on separate groups of bytes. The way the database server stores decimal numbers is best illustrated with an example.

If you specify DECIMAL(6,3), the data type consists of three significant digits to the left of the decimal and three significant digits to the right of the decimal (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 2-7 illustrates. (The exponent byte is not shown.) With the additional byte required for the exponent and sign, this data type requires a total of 5 bytes of storage.

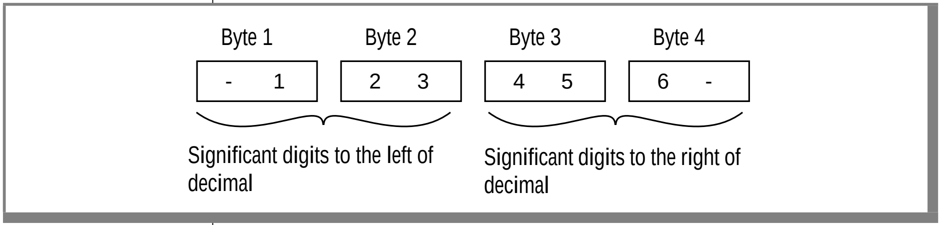

Byte 1
Byte 2
Byte 3
Byte 4
- 1
2 3
4 5
6 -
Significant digits to the left of decimal
Significant digits to the right of decimal

Figure 2-7
Schematic That Illustrates the Storage of Digits in a Decimal Value

You can use the following formulas (rounded down to a whole number of bytes) to calculate the byte storage (N) for a decimal data type (N includes the byte required to store the exponent and sign):

```
If the scale is odd: N = (precision + 4) / 2
If the scale is even: N = (precision + 3) / 2
```

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

One caveat to these formulas exists. 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.

IDS

Distinct

A distinct type is a data type that is based on one of the following source types:

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

A distinct type inherits the casts and functions of its source types as well as the length and alignment on the disk. A distinct type thus makes efficient use of the pre-existing 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 of a built-in type does not inherit the built-in casts that are provided for the built-in type. However, a distinct type does inherit any user-defined casts that have been defined on the source type.

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

You must define a distinct type in the database. Definitions for distinct types are stored in the **sysxtdtypes** 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 on the SQL statements mentioned above, see the *Informix Guide to SQL: Syntax*. For information about casting distinct data types, see "Casts for Distinct Types" on page 2-68. For examples that show how to create and register cast functions for a distinct type, see the *Informix Guide to Database Design and Implementation*.

DOUBLE PRECISION

Columns defined as DOUBLE PRECISION behave the same as those defined as FLOAT.

FLOAT(*n*)

The FLOAT data type stores double-precision floating-point numbers with up to 16 significant digits. FLOAT corresponds to the **double** data type in C. The range of values for the FLOAT data type is the same as the range of values for 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 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.

IDS

INT8

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

The way that the database server stores the INT8 data is platform dependent. On 64-bit platforms, INT8 is stored as a signed binary integer; the data type requires 8 bytes per value. On 32-bit platforms, the database server uses an internal format that consists of several integer values; the data type can require 10 bytes.

Arithmetic operations and sort comparisons are performed more efficiently on integer data than on float or decimal data. However, INT8 columns can store only a limited range of values. If the data value exceeds the numeric range, 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. The maximum negative number, –2,147,483,648, is a reserved value and cannot be used. The INTEGER data type 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. However, INTEGER columns can store only a limited range of values. If the data value exceeds the numeric range, the database server does not store the value.

INTEGER data types require 4 bytes 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 contiguous sequence of values, that represents a component of time. The following example defines an INTERVAL data type:

```
INTERVAL largest_qualifier(n) TO smallest_qualifier(n)
```

In this example, the *largest_qualifier* and *smallest_qualifier* fields are taken from one of the two INTERVAL classes, as Figure 2-8 shows, and *n* optionally specifies the precision of the largest field (and smallest field if it is a FRACTION).

Figure 2-8
Interval Classes

Interval Class	Qualifier Field	Valid Entry
YEAR-MONTH INTERVAL	YEAR	A number of years
	MONTH	A number of months
DAY-TIME INTERVAL	DAY	A number of days
	HOUR	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 of precision. The default precision is 3 digits (thousandth of a second). To indicate explicitly other precisions, write FRACTION(*n*), where *n* is the desired number of digits from 1 to 5.

As with a DATETIME column, you can define an INTERVAL column to include a subset of the fields that you need; however, because the INTERVAL data type represents a span of time that is independent of an actual date, you cannot combine the two INTERVAL classes. For example, because the number of days in a month depends on which month it is, a single INTERVAL data value cannot combine months and days.

A value entered into an INTERVAL column need not include all fields contained in the column. For example, you can enter a value of HOUR TO SECOND into a column defined as DAY TO SECOND. However, a value must always consist of a contiguous sequence of fields. In the previous example, you cannot enter just 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 the Literal Interval in the *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 use *n* optionally to specify the precision of the first field (and the last field if it is a FRACTION). If the largest and smallest field qualifiers are both FRACTIONS, you can specify only the precision in the last field. Acceptable qualifiers for the largest and smallest fields are identical to the list of INTERVAL fields that Figure 2-8 displays.

If you use the DB-Access TABLE menu, and you do not specify the INTERVAL field qualifiers, the 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 values use the same delimiters as DATETIME values. Figure 2-9 shows the delimiters.

Figure 2-9
INTERVAL Delimiters

Delimiter	Placement in DATETIME Expression
Hyphen	Between the YEAR and MONTH portions of the value
Space	Between the DAY and HOUR portions of the value
Colon	Between the HOUR and MINUTE and the 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. However, the character string must include information for the identical sequence of fields defined for that 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 ('BRO', '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 more information on entering INTERVAL values as character strings, see the *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, a YEAR TO MONTH qualifier requires a total of six digits (four for year and two for month). This data value 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 2-53. For information on using INTERVAL as a constant expression, see the description of the INTERVAL Field Qualifier in the *Informix Guide to SQL: Syntax*.

IDS

LIST(*e*)

The LIST data type is a collection type that stores ordered, nonunique elements; that is it allows duplicate element values. The elements of a LIST have ordinal positions; with a first, second, and third element in a LIST. (For a collection type with no ordinal positions, see the MULTISET data type on page 2-32 and the SET data type on page 2-41.)

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 *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 collection can be any of the following 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 collection type, see the *Informix Guide to SQL: Syntax*.

You can use LIST anywhere that you would use any other data type, 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 SQL CARDINALITY function to determine the number of elements in a LIST column

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

Two lists are equal if they have the same elements in the same order. The following examples are lists 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"}
```

IDS

LVARCHAR

The LVARCHAR data type is an SQL data type that you can use to create a column of variable-length character data types that are potentially larger than 255 bytes.

The LVARCHAR data type is also used 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 data is stored in a table column, the value is limited to 2 kilobytes (2Kb). When LVARCHAR is used in I/O operations on an opaque type, it has the theoretical size limit of 4 gigabytes (4Gb).*

The LVARCHAR data type supports only a subset of the string operations that you can perform on CHAR and VARCHAR data types.

For more information about LVARCHAR, see *Extending Informix Dynamic Server 2000*.

MONEY(*p*,*s*)

The MONEY data type stores currency amounts. As with the DECIMAL data type, the MONEY data type stores 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 = (precision + 4) / 2
If the scale is even: N = (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.

Client applications format values in 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 of the value
- A decimal point: a period (.)

To change the format for MONEY values, change the **DBMONEY** environment variable. For information on how to set the **DBMONEY** environment variable, see page 3-45.

GLS

The default value that the database server uses for scale is locale-dependent. The default locale specifies a default scale of two. For nondefault 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.

For more information on locale dependency, see the *Informix Guide to GLS Functionality*. ♦

IDS

MULTISET(*e*)

The MULTISET data type is a collection type that stores nonunique elements: it allows 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 2-29.)

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:

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

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

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 SQL **CARDINALITY** function to determine the number of elements in a MULTISET column

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

Two multisets are equal if they have the same elements, even if the elements are in different positions in the set. The following examples are multisets but are not equal:

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

The following multisets are equal:

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

Named Row

See "Row, Named" on page 2-35.

GLS

NCHAR(*n*)

The NCHAR data type stores fixed-length character data. This data can be a sequence of single-byte or multibyte letters, numbers, and symbols. The main difference between CHAR and NCHAR data types is the collation order. While the collation order of the CHAR data type is defined by the code-set order, the collation order of the NCHAR data type depends on the locale-specific localized order. For more information about the NCHAR data type, see the *Informix Guide to GLS Functionality*.

NUMERIC(*p*,*s*)

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

GLS

NVARCHAR(*m*,*r*)

The NVARCHAR data type stores character data of varying lengths. This data can be a sequence of single-byte or multibyte letters, numbers, and symbols. The main difference between VARCHAR and NVARCHAR data types is the collation order. While the collation order of the VARCHAR data type is defined by the code-set order, the collation order of the NVARCHAR data type depends on the locale-specific localized order. For more information about the NVARCHAR data type, see the *Informix Guide to GLS Functionality*.

IDS

Opaque

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

- A data structure for how the data is stored on disk
- Support functions to determine how to convert between the disk format and the user format
- 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 row in a system catalog table to register the opaque type in the database

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

- The CREATE OPAQUE TYPE statement adds an opaque type to 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 *Informix Guide to SQL: Syntax*. For information on how to create opaque types and an example of an opaque type, see *Extending Informix Dynamic Server 2000*.

REAL

The REAL data type is a synonym for SMALLFLOAT.

IDS

Row, Named

A named row type is defined by its name. That name must be unique within the schema. An unnamed row type is a row type that contains fields but has no user-defined name. Use a named row type if you want to use type inheritance. For more information, see "Row Data Types" on page 2-60.

Defining Named Row Types

You must define a named row type in the database. Definitions for named row types are stored in the **sysxtdtypes** system catalog table.

The fields of a row type can be any data type. The fields of a row type that are TEXT or BYTE type can be used in typed tables only. If you want to assign a row type to a column, then the elements of the row cannot be of TEXT and BYTE data types.

In general, the data type of the field of a row type can be any of the following types:

- A built-in type, except for the restriction against TEXT and BYTE mentioned above
- A collection type
- A distinct type
- A row type
- An opaque type

The following SQL statements maintain the definitions of named row types in the database:

- 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.

For details about the preceding SQL syntax statements, see the *Informix Guide to SQL: Syntax*. For examples of how to create and used named row types, see the *Informix Guide to Database Design and Implementation*.

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 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. That is, one named row type can be the parent (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 *Informix Guide to SQL: Syntax* and in the *Informix Guide to Database Design and Implementation*.

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 *Informix Guide to SQL: Syntax*. Table inheritance and how it relates to type inheritance is also discussed in that section. For information about how to create and use typed tables, see the *Informix Guide to Database Design and Implementation*.

IDS

Row, Unnamed

An unnamed row type contains fields but has no user-defined name. An unnamed row type is defined by its structure. Two unnamed row types are equal if they have the same structure. If two unnamed row types have the same number of fields, and if the data type of each field in one row type matches the data type of the corresponding field in the other row type, the two unnamed row 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 are not equivalent, even though they have the same number of fields and the same data types, because the fields are not in the same order:

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

The data type of the field of an unnamed row type can be any of the following types:

- A built-in type
- A collection type
- A distinct type
- A row type
- An opaque type

Unnamed row types cannot be used in type tables or in type inheritance hierarchies.

For more information on unnamed row types, see the *Informix Guide to SQL: Syntax* and the *Informix Guide to Database Design and Implementation*.

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. For more information, see the Identifier segment in the *Informix Guide to SQL: Syntax*.

- You can generate an unnamed row type using ROW as a constructor and a series of values. A corresponding unnamed row type is created, using the default data types of the specified values.

  For example, a declaration of the following row value:

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

  defines this row 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.

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 segment in the *Informix Guide to SQL: Syntax*.

SERIAL(*n*)

The SERIAL data type stores a sequential integer assigned automatically by the database server when a row is inserted. You can define only one SERIAL column in a table. For information on inserting values in SERIAL columns, see the *Informix Guide to SQL: Syntax*.

The SERIAL data type is not automatically a unique column. 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.

Also, serial numbers might not be contiguous due to such factors as multiuser systems and rollbacks.

The default serial starting number is 1, but you can assign an initial value, *n*, when you create or alter the table. You can assign any number greater than 0 as your starting number. The highest serial number that you can assign is 2,147,483,647. If you assign a number greater than 2,147,483,647, you receive a syntax error.

Once a nonzero number is assigned, it cannot be changed. You can, however, insert a value in a SERIAL column (using the INSERT statement) or reset the serial value *n* (using the ALTER TABLE statement), as long as that value does not duplicate any existing values in the table. When you insert a number in a SERIAL column or reset the next value of a SERIAL column, your database server assigns the next number in sequence to the number entered. However, if you reset the next value of a SERIAL column to a value that is less than the values already in that column, the next value is computed with the following formula:

```
    maximum existing value in SERIAL column + 1
```

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

A SERIAL data column is commonly used to store unique numeric codes (for example, order, invoice, or customer numbers). SERIAL data values require 4 bytes of storage.

IDS

SERIAL8

The SERIAL8 data type stores a sequential integer assigned automatically by the database server when a row is inserted. (For more information on how to insert values into SERIAL8 columns, see the *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 require 8 bytes of storage. The following restrictions apply to SERIAL8 columns:

- You can define only one SERIAL8 column in a table.

  However, a table can have one SERIAL8 and one SERIAL column.
- The SERIAL8 data type is not automatically a unique column.

  You must apply a unique index to this column to prevent duplicate SERIAL8 numbers.
- The SERIAL8 data type does not allow a null value.

If you are using the interactive schema editor in DB-Access to define the table, a unique index is applied automatically to a SERIAL8 column.

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 serial 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 serial column of the first row of the table. The highest serial number you can assign is 2^{63}-1 (9,223,372,036,854,775,807). If you assign a number greater than this value, 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.

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

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. However, if you reset the next value of a SERIAL8 column to a value that is less than the values already in that column, the next value is computed using the following formula:

```
    maximum existing value in SERIAL8 column + 1
```

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

Using SERIAL8 with INT8

The database server treats the SERIAL8 data type as a special case of the INT8 data type. Therefore, all the arithmetic operators that are legal for INT8 (such as +, –, *, and /) and all the SQL functions that are legal for INT8 (such as ABS, MOD, POW, and so on) are also legal for SERIAL8 values. All data conversion rules that apply to INT8 also apply to SERIAL8.

The value of a SERIAL8 column of a table can be stored in the columns of another table. However, when the values of the SERIAL8 column are put into the second table, their values lose the constraints imposed by their original SERIAL8 column and they are stored as INT8 values.

IDS

SET(*e*)

The SET data type is a collection type that stores unique elements; it does not allow duplicate element values. (For a collection type that does allow duplicate values, see the description of MULTISET on page 2-32.)

The elements in a SET have no ordinal position. That is, no concept of a first, second, or third element in a SET exists. (For a collection type with ordinal positions for elements, see the LIST data type on page 2-29.)

All elements in a SET have the same element type. To specify the element type, use the following 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 *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 SQL CARDINALITY function to determine the number of elements in a SET column

For more information, see the Condition and Expression segments in the *Informix Guide to SQL: Syntax*.

You *cannot* use the SET column with an aggregate function such as AVG, MAX, MIN, or SUM.

The following examples declare two sets. The first example 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 is part of a CREATE TABLE statement:

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

The following sets are equal:

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

SMALLFLOAT

The SMALLFLOAT data type stores single-precision floating-point numbers with approximately eight 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 development tool 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 per value.

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 takes up only 2 bytes per value, arithmetic operations are performed efficiently. However, this data type stores a limited range of values. If the values exceed the range between the minimum and maximum numbers, the database server does not store the value and provides you with an error message.

TEXT

The TEXT data type stores any kind of text data. It can contain both single and multibyte characters.

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

The term *simple large object* is used to refer to TEXT and BYTE data types.

TEXT columns typically store memos, manual chapters, business documents, program source files, and so on. In the default locale U.S. ASCII English, data object of type TEXT can contain a combination of printable ASCII characters and the following control characters:

- Tabs (CTRL-I)
- New lines (CTRL-J)
- New pages (CTRL-L)

You can store, retrieve, update, or delete the contents of a TEXT column. However, you cannot use TEXT data items in arithmetic or string operations or assign literals to TEXT items with the SET clause of the UPDATE statement. You also cannot use TEXT items in 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 objects in Boolean expressions only if you are testing for null values.

You can insert data in TEXT columns in the following ways:

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

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 see all of a column, use the regular syntax for selecting a column into a variable. You can also select any part of a TEXT column with subscripts, as the following 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 catalog number 10001.

The database server provides a cast to convert TEXT objects to CLOB objects. For more information, see the *Informix Guide to Database Design and Implementation*.

Important: *If you try to return a TEXT column from a subquery, you get an error message even when the TEXT column is not used in a comparison condition or with the IN predicate.*

Nonprintable Characters with TEXT

Both printable and nonprintable characters can be inserted in text columns. Informix products do not do any checking of the data that is inserted in a column with the TEXT data type. For detailed information on entering and displaying nonprintable characters, refer to "Nonprintable Characters with CHAR" on page 2-14.

Collating TEXT Data

The TEXT data type is collated in code-set order. For more information on collation orders, see the *Informix Guide to GLS Functionality*.

GLS

Multibyte Characters with TEXT

The database locale must support multibyte TEXT characters. For more information, see the *Informix Guide to GLS Functionality*.

Unnamed Row

See "Row, Unnamed" on page 2-36.

VARCHAR(*m*,*r*)

The VARCHAR data type stores character sequences that contain single-byte and multibyte character sequences of varying length, where *m* is the maximum byte size of the column and *r* is the minimum amount of byte space reserved for that column.

The VARCHAR data type is the Informix implementation of a character varying data type. The ANSI standard data type for varying character strings is CHARACTER VARYING and is described on page 2-15.

You must specify the maximum size (*m*) of the VARCHAR column. The size of this parameter 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 shorter, but not longer, character strings than the 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 a minimum space value, it defaults to 0. You should specify this parameter when you initially intend to insert rows with short or null data in this column, but later expect the data to be updated with longer values.

Although the use of VARCHAR economizes on space used in a table, it has no effect on the size of an index. In an index based on a VARCHAR column, each index key has length *m*, the maximum size of the column.

When you store a VARCHAR value in the database, only its defined characters are stored. The database server does not strip a VARCHAR object of any user-entered trailing blanks, nor does the database server pad the VARCHAR to the full length of the column. However, if you specify a minimum reserved space (*r*) and some data values are shorter than that amount, some space reserved for rows goes unused.

VARCHAR values are compared to other VARCHAR values and to character values in the same way that character values are compared. The shorter value is padded on the right with spaces until the values have equal lengths; then they are compared for the full length.

Nonprintable Characters with VARCHAR

Nonprintable VARCHAR characters are entered, displayed, and treated in the same way as nonprintable CHAR characters are. For detailed information on entering and displaying nonprintable characters, refer to "Nonprintable Characters with CHAR" on page 2-14.

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, given the following statement, the value that gets 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.*

GLS

Multibyte Characters with VARCHAR

The database locale must support multibyte VARCHAR characters. If you store multibyte characters, make sure to calculate the number of bytes needed. For more information, see the *Informix Guide to GLS Functionality*.

Collating VARCHAR

The main difference between the NVARCHAR and the VARCHAR data types is the difference in collation sequencing. Collation order of NVARCHAR characters depends on the GLS locale chosen, while collation of VARCHAR characters depends on the code set. For more information, see the *Informix Guide to GLS Functionality*.

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

For a description of character, numeric, and miscellaneous data types, refer to the appropriate entry in "Description of Data Types" on page 2-9. Page references are in the alphabetical list in Figure 2-2 on page 2-6.

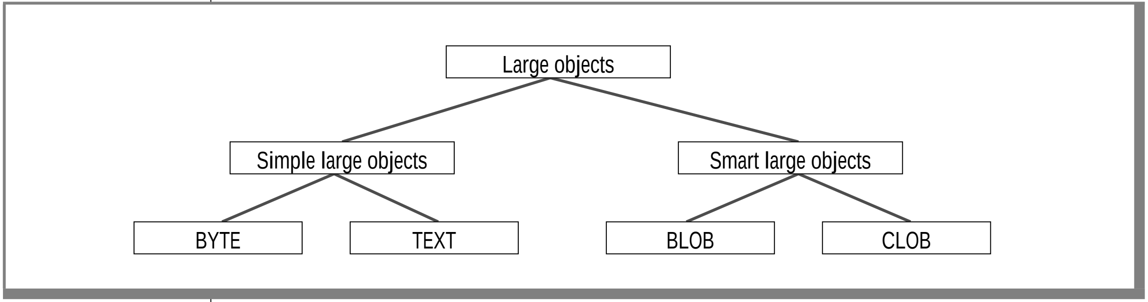
The following sections provide additional information on large-object and time data types.

Large-Object Data Types

A large object is a data object that is logically stored in a table column but physically stored independently of the column. Large objects are stored separately from the table because they typically store a large amount of data. Separation of this data from the table can increase performance.

Figure 2-10 shows the large-object data types.

Figure 2-10
Large-Object Data Types



IDS

Only Dynamic Server supports BLOB and CLOB data types. ♦

For the relative advantages and disadvantages of simple and smart large objects, see the *Informix Guide to Database Design and Implementation*.

Simple Large Objects

Simple large objects are a category of large objects that have a theoretical limit of 2^{31} bytes and a practical limit that your disk capacity determines. Informix database servers support the following simple-large-object data types:

BYTE	Stores binary data. For more detailed information about this data type, see the description on page 2-12.
TEXT	Stores text data. For more detailed information about this data type, see the description on page 2-43.

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 does not fit into memory, you must store it 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 *Administrator's Guide*.

IDS

Smart Large Objects

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 on page 2-61.) ♦

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 2-9.
CLOB	Stores text data. For more information about this data type, see the description on page 2-15.

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 *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 *Performance Guide*.

When you access a smart-large-object column with an SQL statement, 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 to perform the 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 Informix ESQL/C program, use the smart-large-object API.

  For more information, see the *Informix ESQL/C Programmer's Manual*.
- From within a DataBlade module, use the Client and Server API.

  For more information, see the *DataBlade API Programmer's Manual*.

For information on smart large objects, see the *Informix Guide to SQL: Syntax* and *Informix Guide to Database Design and Implementation*.

Time Data Types

You can use DATE, DATETIME, and INTERVAL data in a variety of arithmetic and relational expressions. You can manipulate a DATETIME value with another DATETIME value, an INTERVAL value, the current time (identified by the keyword CURRENT), or a specified unit of time (identified by the keyword UNITS).

In most situations, you can use a DATE value wherever it is appropriate to use a DATETIME value and vice versa. You also can manipulate an INTERVAL value with the same choices as a DATETIME value. In addition, you can 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. On the other hand, if you add or subtract an INTERVAL value from a DATETIME value, another DATETIME value is produced because the result is a specific time.

Figure 2-11 indicates the range of expressions that you can use with DATE, DATETIME, and INTERVAL data and the data type that results from each expression.

Figure 2-11
Range of Expressions for DATE, DATETIME, and INTERVAL

Data Type of Operand 1	Operator	Data Type of 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

(1 of 2)

Data Type of Operand 1	Operator	Data Type of Operand 2	Result
CURRENT	–	DATETIME	INTERVAL
INTERVAL	+	CURRENT	DATETIME
CURRENT	+ or –	INTERVAL	DATETIME
DATETIME	+ or –	UNITS	DATETIME
INTERVAL	+ or –	UNITS	INTERVAL
INTERVAL	* or /	NUMBER	INTERVAL

(2 of 2)

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 field precision of the result.

If the second DATETIME value has fewer fields than the first, the shorter value is extended automatically to match the longer one. (See the discussion of the EXTEND function in the Expression segment in the *Informix Guide to SQL: Syntax*.)

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 value, the database server sets the minutes for the result to 0.

```
DATETIME (1999-9-30 12:30) YEAR TO MINUTE
    - DATETIME (1999-8-1 11) YEAR TO HOUR

Result: INTERVAL (60 01:30) DAY TO MINUTE
```

If the second DATETIME value has more fields than the first (regardless of whether the precision of the extra fields is larger or smaller than those in the first value), the additional fields in the second value are ignored in the calculation.

In the following expression (and result), the year is not included for the second value. Therefore, the year is set automatically to the current year, in this case 1999, and the resulting INTERVAL is negative, which indicates that the second date is later than the first.

```
DATETIME (1999-9-30) YEAR TO DAY
    - DATETIME (10-1) MONTH TO DAY

Result: INTERVAL (1) DAY TO DAY [assuming current year
is 1999]
```

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 an INTERVAL value simply moves the DATETIME value forward or backward in time. The expression shown in the following example moves the date ahead three years and five months:

```
DATETIME (1994-8-1) YEAR TO DAY
    + INTERVAL (3-5) YEAR TO MONTH

Result: DATETIME (1998-01-01) YEAR TO DAY
```

Important: *Evaluate the logic of your addition or subtraction. Remember that months can be 28, 29, 30, or 31 days and that years can be 365 or 366 days.*

In most situations, the database server automatically adjusts the calculation when the initial values do not have the same precision. However, in certain situations, 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 explicitly extend the field qualifier of the DATETIME value. (For more information on the EXTEND function, see the Expression segment in the *Informix Guide to SQL: Syntax*.)

For example, you cannot subtract a minute INTERVAL 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 (1998-8-1) YEAR TO DAY, YEAR TO MINUTE)
    - INTERVAL (720) MINUTE(3) TO MINUTE

Result: DATETIME (1998-07-31 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 values in arithmetic expressions with DATETIME or INTERVAL values by writing expressions that allow the manipulations that Figure 2-12 shows.

Figure 2-12
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 Figure 2-12 shows, DATE values are first converted to their corresponding DATETIME equivalents, and then the expression is computed normally.

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.

When you represent DATE and DATETIME values as quoted 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 `'10/30/1999'` as a DATE string but not as a DATETIME string. Instead, you must use `'1999-10-30'` or `'99-10-30'` as the DATETIME string.

GLS

If you use a nondefault locale, the DATE and DATETIME strings must match the formats that your locale defines. For more information, see the *Informix Guide to GLS Functionality*.

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 3-37 and "DBTIME" on page 3-54. For more information on all these environment variables, see the *Informix Guide to GLS Functionality*. ♦

You can also subtract one DATE value from another DATE value, but the result is a positive or negative INTEGER value rather than an INTERVAL value. If an INTERVAL value is required, you can either convert the INTEGER value into an INTERVAL value or 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
```

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 as long as both values are from the same class; that is, 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 a number that can be an integer or a fraction. However, any remainder from the calculation is ignored and the result is truncated. The following expression multiplies an INTERVAL by a fraction:

```
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 results of any calculation include the same amount of precision as the original INTERVAL value.

IDS

Extended Data Types

Dynamic Server lets you create the following kinds of extended data types to characterize data that cannot be easily represented with the built-in data types:

- Complex data types
- Distinct data types
- Opaque data types

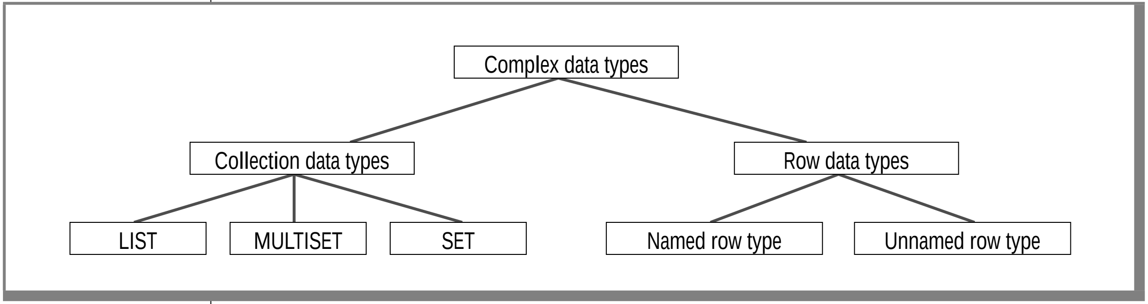
The following sections provide an overview of each of these data types.

For more information about extended data types, see the *Informix Guide to Database Design and Implementation* and *Extending Informix Dynamic Server 2000*.

Complex Data Types

A *complex data type* is a data type that you build from other data types (built-in and extended). Figure 2-13 shows the complex types that Dynamic Server supports. The table that follows briefly describes the structure of these data types.

Figure 2-13
Supported Complex Data Types



Data Type	Description
Collection types	Complex data types that are made up of elements, each of which is the same data type.
LIST	A group of ordered elements, each of which need not be unique.
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/or lists. Likewise, a collection type can have elements whose data type is a row type or a collection type.

All complex types inherit the following support functions:

input	assign
output	destroy
send	LO handles
recv	hash
import	lessthan
export	equal
import binary	lessthan (ROW only)
export binary	

The following sections summarize the complex types. For more information on complex types, see the *Informix Guide to Database Design and Implementation*.

Collection Data Types

A collection data type is a complex type that is made up of one or more elements. Every element in a collection has the same data type. A collection element can have 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 collection types: LIST, SET, and MULTISET. The keywords used to construct these collections are called *type constructors* or just *constructors*. For a description of each of these collection data types, see its entry in this chapter. For the syntax of collection types, see the *Informix Guide to SQL: Syntax*.

Using Complex Data Types in Table Columns

When you specify element values for a collection, list the element values after the constructor and between curly brackets. For example, suppose you have a collection column with the following type:

```
CREATE TABLE table1
(
    mset_col MULTISET(INTEGER NOT NULL)
)
```

The following INSERT statement adds one group of element values to this MULTISET column. The word MULTISET in the two examples is the MULTISET constructor.

```
INSERT INTO table1 VALUES (MULTISET{5, 9, 7, 5})
```

Leave the brackets 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 *Informix Guide to SQL: Tutorial*.
- Host variables in an Informix ESQL/C program

  For more information, see the *Informix ESQL/C Programmer's Manual*.

You can also use nested dot notation to access collection data. To learn more about accessing the elements of a collection, see the *Informix Guide to SQL: Tutorial*.

Row Data Types

A row type is a sequence 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, and you cannot use fields with tables, only with row types.

Two kinds of row types exist:

- Named row types are identified by their names.
- Unnamed row types are identified by their structure.

The structure of an unnamed row type consists of the number and data types of its fields. For more information about row types, see “Row, Named” on page 2-35 and “Row, Unnamed” on page 2-36.

You can cast between named and unnamed row types. For information about casting between row types, see the *Informix Guide to Database Design and Implementation*.

Distinct Data Types

A distinct data type has the same internal structure as some other source data type in the database. The source data type can be either a built-in type or an extended type. What distinguishes a distinct type from the source type are the functions defined on this type. For more information, see the description on page 2-23.

Opaque Data Types

An opaque data type is a user-defined data type that is fully encapsulated, that is, whose internal structure is unknown to the database server. For more information, see the description on page 2-33.

Data Type Casting and Conversion

Occasionally, the data type that was assigned to a column with the CREATE TABLE statement is inappropriate. You might want 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 the old values. 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 32,768:

```
100 400 700 50000700
```

The same situation might occur if you attempt to transfer data from FLOAT or SMALLFLOAT columns to INTEGER, SMALLINT, or DECIMAL columns.

Both data type conversion and casting depend on casts defined in the **syscasts** system catalog table. For information about syscasts, see “SYSCASTS” on page 1-25.

A cast is either built-in or user-defined. Guidelines exist for casting distinct and extended data types.

For more information about casting opaque types, see *Extending Informix Dynamic Server 2000*. For information about casting other extended types, see the *Informix Guide to Database Design and Implementation*.

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 convert:

- A character type to any other character type
- A character type to or from any other built-in type
- A numeric type to any other numeric type
- A time data type to or from a datetime type

The database server automatically invokes the appropriate built-in casts when required. An infinite number of built-in casts might be invoked to evaluate and compare expressions or to change a column from one built-in data type to another.

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 50000700
```

The same situation might occur if you attempt to transfer data from FLOAT or SMALLFLOAT columns to INTEGER, SMALLINT, or DECIMAL columns.

The following sections 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. Figure 2-14 on page 2-64 indicates which numeric data type conversions are acceptable and what kinds of errors you can encounter when you convert between certain numeric data types.

Figure 2-14
Numeric Data Type Conversion Chart

	TO					
FROM	**SMALLINT**	**INTEGER**	**INT8**	**SMALLFLOAT**	**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 data
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 numbers before storing them as decimal numbers. This conversion can result in an error depending on the precision assigned to the DECIMAL column.

Converting Between Number and CHAR

You can convert a CHAR (or NCHAR) column to a numeric column. However, if the CHAR or NCHAR column contains any characters that are not valid in a number column (for example, the letter *l* instead of the number *1*), your database server returns an error.

You can also convert a numeric column to a character column. However, if the character column is not large enough to receive the number, 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 (whether you are in a transaction or not).

Converting Between INTEGER and DATE or DATETIME

You can convert an integer column (SMALLINT, INTEGER, or INT8) to a DATE or DATETIME value. The database server interprets the integer as a value in the internal format of the DATE or DATETIME column. You can also convert a DATE or DATETIME column to an integer column. The database server stores the internal format of the DATE or DATETIME column as an integer.

For a DATE column, the internal format is a Julian date. For a DATETIME column, the internal format stores the date and time in a condensed integer format.

Converting Between DATE and DATETIME

You can convert DATE columns to DATETIME columns. However, if the DATETIME column contains more fields than the DATE column, 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 08/15/1999 becomes 1999-08-15.
- If you convert DATETIME YEAR TO DAY to the DATE format, the value 1999-08-15 becomes 08/15/1999.

- 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, 08/15/1999 becomes 1999-08-15 00:00:00.
- If you convert DATETIME YEAR TO SECOND to DATE, the database server converts existing DATETIME to DATE values but drops fields more precise than DAY. For example, 1999-08-15 12:15:37 becomes 08/15/1999.

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 create and use implicit and explicit casts, see the CREATE CAST statement in the *Informix Guide to SQL: Syntax* and the *Informix Guide to Database Design and Implementation*.

Implicit Casts

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.

Implicit casts allow you to convert a user-defined data type to a built-in type or vice versa.

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 the double colon (::) cast operator.

Explicit casts do not allow you to convert a user-defined data type to a built-in data type or vice versa.

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.
- 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 form that the database server understands (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 (LVARCHAR, SENDRECV, IMPEX, or IMPEXBIN), then explicitly cast this type to the second opaque type.

For information about casting and the IMPEX, IMPEXBIN, LVARCHAR, and SENDRECV data types, see *Extending Informix Dynamic Server 2000*.

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. However, to create an implicit cast, 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 define 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 *Informix Guide to Database Design and Implementation*.

What Extended Data Types Can Be Cast?

The following table shows the data type combinations that you can cast. 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 automatically provides a cast that is implicitly invoked or that you must explicitly invoke.

Target Type --->	Opaque Type	Distinct Type	Named Row Type	Unnamed Row Type	Collection Type	Built-in Type
Opaque Type	explicit or implicit	explicit	explicit[3]	Not Allowed	Not Allowed	explicit or implicit[3]
Distinct Type	explicit	explicit	explicit	Not Allowed	Not Allowed	explicit or implicit
Named Row Type	explicit[3]	explicit	explicit3	explicit1	Not Allowed	Not Allowed
Unnamed Row Type	Not Allowed	Not Allowed	explicit[1]	implicit1	Not Allowed	Not Allowed
Collection Type	Not Allowed	Not Allowed	Not Allowed	Not Allowed	explicit[2]	Not Allowed
Built-in Type	explicit or implicit[3]	explicit or implicit	Not Allowed	Not Allowed	Not Allowed	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.

Operator Precedence

An operator is a symbol or keyword that is used for operations on data types. Some operators only support built in data types; other operators support both built-in and extended data types.

The following table shows the precedence of the operators that Informix database servers support, in descending order of precedence. Operators with the same precedence are listed in the same row.

Operator Precedence
UNITS
+ (unary) -(unary)
::
* /
+ -
| |
ANY ALL SOME
NOT
< <= = > >= != <> IN BETWEEN LIKE MATCHES
AND
OR

Chapter

Environment Variables

3

In This Chapter 3-5

Types of Environment Variables 3-6

Where to Set Environment Variables in UNIX. 3-7

Where to Set Environment Variables in Windows NT 3-7

Using Environment Variables in UNIX 3-8
- Setting Environment Variables in an Environment-Configuration File 3-8
- Setting Environment Variables at Login Time 3-9
- Syntax for Setting Environment Variables. 3-10
- Unsetting Environment Variables 3-10
- Modifying an Environment-Variable Setting 3-11
- Viewing Your Environment-Variable Settings 3-12
- Checking Environment Variables with the chkenv Utility 3-12
- Rules of Precedence 3-13

Using Environment Variables in Windows NT 3-14
- Setting Environment Variables for Native Windows Applications 3-14
- Setting Environment Variables for Command-Prompt Utilities 3-15
  - Using the System Applet to Work with Environment Variables 3-15

Using the Command Prompt to Work with Environment Variables 3-16
Using dbservername.cmd to Initialize a Command-Prompt Environment 3-18
Rules of Precedence 3-18

List of Environment Variables 3-19

Environment Variables. 3-24
AC_CONFIG. 3-24
ARC_CONFIG 3-25
ARC_DEFAULT 3-25
ARC_KEYPAD 3-26
COCKPITSERVICE 3-27
CPFIRST 3-28
DBACCNOIGN. 3-29
DBANSIWARN 3-30
DBBLOBBUF. 3-31
DBCENTURY 3-32
Behavior of DBCENTURY with Expressions that Contain Date Values 3-34
DBDATE 3-36
Standard Formats for DBDATE 3-36
Behavior of DBDATE with Expressions that Contain Date Values 3-38
DBDELIMITER 3-39
DBEDIT 3-40
DBFLTMASK 3-41
DBLANG 3-42
DBMONEY 3-44
DBONPLOAD 3-45
DBPATH 3-46
DBPRINT 3-48
DBREMOTECMD 3-49
DBSPACETEMP 3-50
DBTEMP 3-52
DBTIME 3-53
DBUPSPACE. 3-56
DELIMIDENT 3-57

ENVIGNORE 3-58
FET_BUF_SIZE 3-59
IFMX_SMLTBL_BROADCAST_SIZE 3-60
IFX_DIRECTIVES 3-61
IFX_LONGID 3-62
IFX_NETBUF_PVTPOOL_SIZE 3-63
IFX_NETBUF_SIZE 3-63
IFX_UPDDESC 3-64
INFORMIXC 3-64
INFORMIXCONCSMCFG 3-65
INFORMIXCONRETRY 3-66
INFORMIXCONTIME 3-66
INFORMIXCPPMAP 3-68
INFORMIXDIR 3-68
INFORMIXKEYTAB. 3-69
INFORMIXOPCACHE 3-70
INFORMIXSERVER 3-70
INFORMIXSHMBASE 3-72
INFORMIXSQLHOSTS. 3-73
INFORMIXSTACKSIZE 3-74
INFORMIXTERM 3-75
INF_ROLE_SEP 3-76
ISM_COMPRESSION 3-77
ISM_DEBUG_FILE 3-77
ISM_DEBUG_LEVEL 3-78
ISM_ENCRYPTION 3-79
ISM_MAXLOGSIZE. 3-79
ISM_MAXLOGVERS 3-80
LD_LIBRARY_PATH 3-80
LIBPATH 3-81
NODEFDAC 3-81
ONCONFIG 3-82
OPTCOMPIND 3-83
OPTMSG 3-84
OPTOFC. 3-85
OPT_GOAL 3-86

PATH 3-87
PDQPRIORITY 3-88
PLCONFIG 3-90
PLOAD_LO_PATH 3-91
PLOAD_SHMBASE 3-91
PSORT_DBTEMP 3-92
PSORT_NPROCS 3-93
SHLIB_PATH 3-94
STMT_CACHE 3-95
TERM 3-96
TERMCAP 3-96
TERMINFO 3-97
THREADLIB 3-98
XFER_CONFIG 3-98

Index of Environment Variables 3-99

In This Chapter

Various *environment variables* affect the functionality of your 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. For example, you must either set or accept the default setting for certain UNIX environment variables.

This chapter describes how to use the environment variables that apply to one or more Informix products and shows how to set them. For specific information, see the following pages:

- The environment variables that this chapter discusses are listed alphabetically beginning on page 3-19.
- The environment variables are described beginning on page 3-23.
- A topical index of environment variables is included at the end of this chapter beginning on page 3-100.

Types of Environment Variables

The following types of environment variables are discussed in this chapter:

- Informix-specific environment variables

  Set Informix environment variables when you want to work with Informix products. Each Informix product manual specifies the environment variables that you must set to use that product.

- Operating-system-specific environment variables

  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 have to set the **TERMCAP** or **TERMINFO** environment variable to use some products effectively.

GLS

The GLS environment variables that let you work in a nondefault locale are described in the *Informix Guide to GLS Functionality*. These GLS variables are included in the list of environment variables in Figure 3-1 on page 3-19 and in the topic index in Figure 3-2 on page 3-100 but are not discussed in this manual. ♦

Tip: *Additional environment variables that are specific to your client application or SQL API might be discussed in the manual for that product.*

UNIX

Where to Set Environment Variables in UNIX

You can set environment variables in 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. For more information, see "Using Environment Variables in UNIX" on page 3-8.
- In an environment-configuration file

  An environment-configuration file is a common or private file where you can define all the environment variables that Informix products use. Use of an environment-configuration file reduces the number of environment variables that you must set at the command line or in a shell file.
- In a login file

  When you set an environment variable in your **.login**, **.cshrc**, or **.profile** file, it is assigned automatically every time you log into the system.

E/C

In 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 *Informix ESQL/C Programmer's Manual* and your C documentation. ♦

WIN NT

Where to Set Environment Variables in Windows NT

You might be able to set environment variables in several places in a Windows environment, depending on which Informix application you use.

For native Windows 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 "Setting Environment Variables for Command-Prompt Utilities" on page 3-15.

To use client applications such as ESQL/C or the Relational Object Manager in a Windows environment, use the Setnet32 utility to set environment variables. For information about the Setnet32 utility, see the *Informix Client Products Installation Guide* for your operating system.

E/C

In 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 NT system supports them. For more information on **ifx_putenv()** and **ifx_getenv()**, see the *Informix ESQL/C Programmer's Manual*. ♦

UNIX

Using Environment Variables in UNIX

The following sections discuss setting, unsetting, modifying, and viewing environment variables. If you already use an Informix product, some or all of the appropriate environment variables might be set.

Setting Environment Variables in an Environment-Configuration File

The common (shared) environment-configuration file that is provided with Informix products resides in **$INFORMIXDIR/etc/informix.rc**. The permission 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 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 #) and variable lines and their values (separated by blanks and 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 on page 3-59, to override one or more entries in an environment-configuration file. Use the Informix **chkenv** utility, described on page 3-12, 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 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 the commands that set your environment variables to the following login file:

For the C shell	**.login** or **.cshrc**
For the 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 supports a shortcut, as the fourth item indicates. The environment variables are case sensitive.

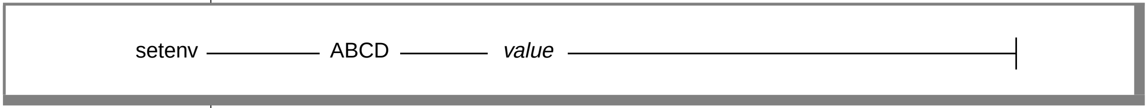
Shell	Command
C	`setenv ABCD value`

Bourne	`ABCD=value`
	`export ABCD`

Korn	`ABCD=value`
	`export ABCD`

Korn	`export ABCD=value`

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.



For more information on how to read syntax diagrams, see “Command-Line Conventions” in the Introduction.

Unsetting Environment Variables

To unset an environment variable, enter the following command.

Shell	Command
C	`unsetenv ABCD`
Bourne or Korn	`unset ABCD`

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 in UNIX environments. When you use an Informix product, you must add to the **PATH** the name of the directory where the executable files for the Informix products are stored.

In the following example, the **INFORMIXDIR** is **/usr/informix**. (That is, during installation, the 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 contradict the 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 curly braces with the existing **INFORMIXDIR** and **PATH**, as the following command shows:

```
setenv PATH ${INFORMIXDIR}/bin:${PATH}
```

For more information about setting and modifying environment variables, refer to the manuals for your operating system.

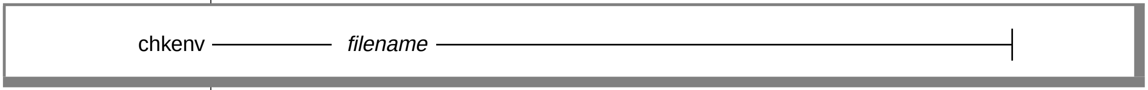
Viewing Your Environment-Variable Settings

After you have installed one or more 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 Informix products use.



Element	Purpose	Key Considerations
filename	Specifies the name of the environment-configuration file that you want to debug.	None

The shared environment-configuration file is stored in **$INFORMIXDIR/etc/informix.rc**. A private environment-configuration file is stored in the user's home directory as **.informix**.

If you do not provide the filename for **chkenv**, the utility checks both the shared and private environment configuration files. If you provide a file path, **chkenv** checks only that 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.

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 3-59.

Rules of Precedence

When an 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 (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 user's home directory (**~/.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.

For precedence information about GLS environment variables, see the *Informix Guide to GLS Functionality*.

***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.*

WIN NT

Using Environment Variables in Windows NT

The following sections discuss setting, viewing, unsetting, and modifying environment variables for native Windows applications and command-line utilities.

Setting Environment Variables for Native Windows Applications

Native Windows Informix 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**.

Manipulating 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 in place of *dbservername*.
3. With the Environment registry key selected in the left pane, you should see a list of environment variables and their defined 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.

For more information on using the Registry Editor, see your operating-system documentation.

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.*

Setting Environment Variables 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-prompt 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.

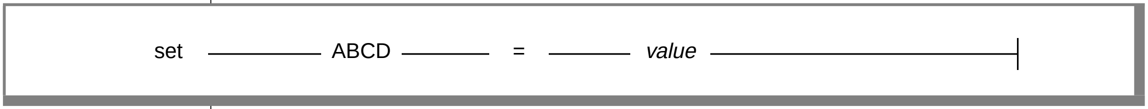
   Click the Environment tab near the top of the window. Two list boxes display System Environment Variables and User Environment Variables. System Environment Variables apply to an entire system, and User Environment Variables apply only to the sessions of the individual user.
2. To change the value of 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.*

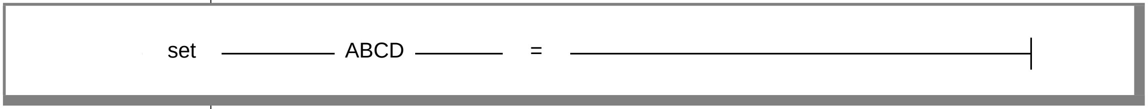
Using the Command Prompt to Work with Environment Variables

The following diagram shows the syntax for setting an environment variable at a command prompt in Windows NT.



For more information on how to read syntax diagrams, see "Command-Line Conventions" in the introduction.

To view your current settings after one or more Informix products are installed, enter the following command at a command prompt.



Sometimes you must add information to an environment variable that is already set. For example, the **PATH** environment variable is always set in Windows NT environments. When you use an Informix product, you must add the name of the directory where the executable files for the Informix products are stored to the **PATH**.

In the following example, **INFORMIXDIR** is **d:\informix**, (that is, during installation, 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 path that was set in **INFORMIXDIR** and to avoid resetting **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 NT 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 prompt and set the variable, **INFORMIXDIR**, and then open another command prompt 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 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 NT services access only environment variables that are set in the registry, the preceding rules of precedence do not apply for 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 your client products, the clients will adopt the original settings.*

List of Environment Variables

Figure 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 last column.

GLS

Although the GLS environment variables that let you work in a nondefault locale are listed in Figure 3-1, they are described in the *Informix Guide to GLS Functionality*. ♦

Figure 3-1
Alphabetical List Of Environment Variables

Environment Variable	XPS	IDS	Restrictions	Page
AC_CONFIG	✔	✔	None	3-24
ARC_CONFIG		✔	UNIX only	3-25
ARC_DEFAULT		✔	UNIX only	3-25
ARC_KEYPAD		✔	UNIX only	3-25
CC8BITLEVEL			ESQL/C only	GLS guide
CLIENT_LOCALE	✔	✔	None	GLS guide
COCKPITSERVICE	✔	✔	DB/Cockpit only	3-27

(1 of 5)

Environment Variable	XPS	IDS	Restrictions	Page
CPFIRST	✔	✔	None	3-28
DBACCNOIGN	✔	✔	DB-Access only	3-29
DBANSIWARN	✔	✔	None	3-31
DBBLOBBUF	✔	✔	None	3-31
DBCENTURY			SQL APIs only	3-33
DBDATE	✔	✔	None	3-37; GLS guide
DBDELIMITER	✔	✔	None	3-40
DBEDIT	✔	✔	None	3-41
DBFLTMASK	✔	✔	DB-Access only	3-42
DBLANG	✔	✔	None	3-43; GLS guide
DBMONEY	✔	✔	None	3-45; GLS guide
DBONPLOAD		✔	HPL only	3-46
DBPATH	✔	✔	None	3-46
DBPRINT	✔	✔	UNIX only	3-49
DBREMOTECMD	✔	✔	UNIX only	3-50
DBSPACETEMP	✔	✔	None	3-51
DBTEMP			Gateways only	3-53
DBTIME			SQL APIs only	3-54; GLS guide
DBUPSPACE	✔	✔	None	3-57

(2 of 5)

Environment Variable	XPS	IDS	Restrictions	Page
DB_LOCALE	✔	✔	None	GLS guide
DELIMIDENT	✔	✔	None	3-58
ENVIGNORE	✔	✔	UNIX only	3-59
ESQLMF	✔	✔	ESQL/C only	GLS guide
FET_BUF_SIZE	✔	✔	SQL APIs, DB-Access only	3-60
GLS8BITSYS	✔	✔	None	GLS guide
GL_DATE	✔	✔	None	GLS guide
GL_DATETIME	✔	✔	None	GLS guide
IFMX_SMLTBL_BROADCAST_SIZE	✔		None	3-61
IFX_DIRECTIVES	✔	✔	None	3-62
IFX_LONGID		✔	None	3-63
IFX_NETBUF_PVTPOOL_SIZE	✔	✔	UNIX only	3-64
IFX_NETBUF_SIZE	✔	✔	None	3-64
IFX_UPDDESC		✔	None	3-65
INFORMIXC			ESQL/C, UNIX only	3-65
INFORMIXCONCSMCFG		✔	None	3-66
INFORMIXCONRETRY	✔	✔	None	3-67
INFORMIXCONTIME	✔	✔	None	3-67
INFORMIXCPPMAP		✔	None	3-69

(3 of 5)

Environment Variable	XPS	IDS	Restrictions	Page
INFORMIXDIR	✔	✔	None	3-69
INFORMIXKEYTAB	✔	✔	UNIX only	3-70
INFORMIXOPCACHE		✔	Optical Subsystem only	3-71
INFORMIXSERVER	✔	✔	None	3-71
INFORMIXSHMBASE	✔	✔	UNIX only	3-73
INFORMIXSQLHOSTS	✔	✔	None	3-74
INFORMIXSTACKSIZE	✔	✔	None	3-75
INFORMIXTERM	✔	✔	DB-Access, UNIX only	3-76
INF_ROLE_SEP		✔	None	3-77
ISM_COMPRESSION	✔	✔	ISM, ON-Bar only	3-78
ISM_DEBUG_FILE	✔	✔	ISM only	3-78
ISM_DEBUG_LEVEL	✔	✔	ISM, ON-Bar only	3-79
ISM_ENCRYPTION	✔	✔	ISM, ON-Bar only	3-80
ISM_MAXLOGSIZE	✔	✔	ISM only	3-80
ISM_MAXLOGVERS	✔	✔	ISM only	3-81
LD_LIBRARY_PATH			SQL APIs, UNIX only	3-81
LIBPATH			SQL APIs, UNIX only	3-82
NODEFDAC	✔	✔	None	3-82
ONCONFIG	✔	✔	None	3-83
OPTCOMPIND	✔	✔	None	3-84

(4 of 5)

Environment Variable	XPS	IDS	Restrictions	Page
OPTMSG			ESQL/C only	3-85
OPTOFC			ESQL/C only	3-86
OPT_GOAL	✔	✔	UNIX only	3-87
PATH	✔	✔	None	3-88
PDQPRIORITY	✔	✔	None	3-89
PLCONFIG		✔	HPL only	3-91
PLOAD_LO_PATH		✔	HPL only	3-92
PLOAD_SHMBASE		✔	HPL only	3-92
PSORT_DBTEMP	✔	✔	None	3-93
PSORT_NPROCS	✔	✔	None	3-94
SERVER_LOCALE	✔	✔	None	GLS guide
SHLIB_PATH	✔	✔	UNIX only	3-95
STMT_CACHE		✔	None	3-96
TERM	✔	✔	UNIX only	3-97
TERMCAP	✔	✔	UNIX only	3-97
TERMINFO	✔	✔	UNIX only	3-98
THREADLIB			ESQL/C, UNIX only	3-99
XFER_CONFIG	✔		None	3-99

(5 of 5)

Tip: *You might encounter references to environment variables that are not listed in Figure 3-1. Most likely, these environment variables are not supported in Version 8.3, Version 9.2, or are used to maintain backward compatibility with certain earlier product versions. For information, refer to an earlier version of your Informix documentation.*

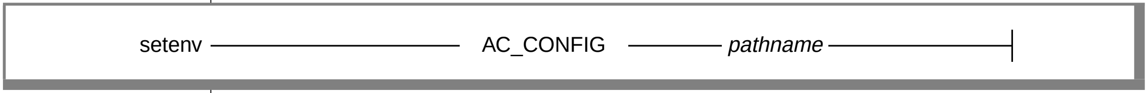
Environment Variables

The following sections discuss the environment variables that Informix products use.

Important: *The descriptions of the following environment variables include the syntax for setting the environment variable in the UNIX environment. For a general description of how to set these environment variables in Windows NT environments, see "Setting Environment Variables for Native Windows Applications" on page 3-14 and "Setting Environment Variables for Command-Prompt Utilities" on page 3-15.*

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. The **archecker** utility checks the validity and completeness of an ON-Bar storage-space backup. The **ac_config.std** file contains default **archecker** configuration parameters.



pathname	is the location of the **ac_config.std** configuration file in **$INFORMIXDIR/etc** or **%INFORMIXDIR%\etc**.

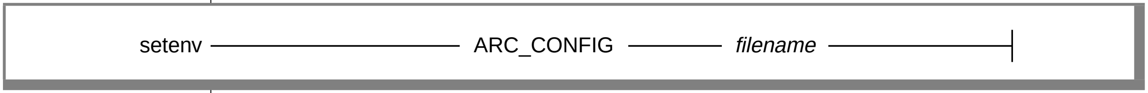
For information on **archecker**, see your *Backup and Restore Guide*.

IDS

UNIX

ARC_CONFIG

If you use the ON-Archive archive and tape-management system for your database server, you can set the **ARC_CONFIG** environment variable to specify the name of a nondefault configuration file.



filename	is the name of the configuration file in **$INFORMIXDIR/etc**.

The default configuration file for ON-Archive is **config.arc**, located in the **$INFORMIXDIR/etc** directory. When you want to create and use a different configuration file for ON-Archive, set **ARC_CONFIG** to the name of the file. Dynamic Server looks for the specified file in the directory **$INFORMIXDIR/etc**.

The **ARC_CONFIG** environment variable lets you change configuration parameters while preserving the default **config.arc** file. It lets you create multiple configuration files and select the one you want.

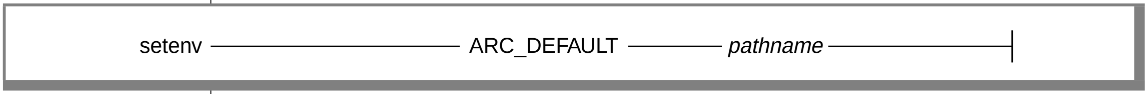
For more information on archiving, see your *Archive and Backup Guide*.

IDS

UNIX

ARC_DEFAULT

If you use the ON-Archive archive and tape-management system for your database server, you can set the **ARC_DEFAULT** environment variable to indicate where a personal default qualifier file is located.



pathname	is the full pathname of the personal default qualifier file.

For example, to set the **ARC_DEFAULT** environment variable to specify the file **/usr/jane/arcdefault.janeroe**, enter the following command:

```
setenv ARC_DEFAULT /usr/jane/arcdefault.janeroe
```

For more information on archiving, see your *Archive and Backup Guide*.

IDS

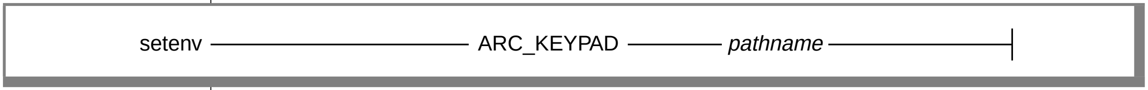
UNIX

ARC_KEYPAD

If you use the ON-Archive archive and tape-management system for your database server, you can set your **ARC_KEYPAD** environment variable to point to a **tctermcap** file that is different from the default **tctermcap** file. The default is the **$INFORMIXDIR/etc/tctermcap** file, and it contains instructions on how to modify the **tctermcap** file.

The **tctermcap** file serves the following purposes for the ON-Archive menu interface:

- It defines the terminal control attributes that allow ON-Archive to manipulate the screen and cursor.
- It defines the mappings between commands and key presses.
- It defines the characters used in drawing menus and borders for an API.



pathname	is the pathname for a **tctermcap** file.

For example, to set the **ARC_KEYPAD** environment variable to specify the file **/usr/jane/tctermcap.janeroe**, enter the following command:

```
setenv ARC_KEYPAD /usr/jane/tctermcap.janeroe
```

For more information on archiving, see your *Archive and Backup Guide*.

COCKPITSERVICE

Set the **COCKPITSERVICE** environment variable to specify a nondefault TCP service for DB/Cockpit.

```
setenv ———————— COCKPITSERVICE ———— servicename ————┤
```

servicename	is the name of the TCP service for client/server communication.

The **onprobe** server and **oncockpit** client components of DB/Cockpit communicate with each other through a TCP *service*. Before you can launch DB/Cockpit, **onprobe**, or **oncockpit**, you should define the TCP service and assign it a unique service number. For example:

```
setenv COCKPITSERVICE cockpit2
```

If you do not specify the service name either with the ***-service*** command-line option or in the **COCKPITSERVICE** environment variable, the default is **cockpit**.

CPFIRST

Set the **CPFIRST** environment variable to determine the nondefault compilation order for all ESQL/C source files in your programming environment.

When compiling an ESQL/C program, the default order is to run the ESQL/C preprocessor on the program source file and pass the resulting file to the C language preprocessor and compiler. However, you can 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 determine the 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 the **CPFIRST** environment variable to TRUE (uppercase only) to run the C preprocessor on all ESQL/C source files. The C preprocessor will run 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.

```
setenv ———————————— CPFIRST ———— TRUE ————————|
```

DBACCNOIGN

The **DBACCNOIGN** environment variable affects the behavior of the DB-Access utility if an error occurs under one of the following circumstances:

IDS

- You run DB-Access in nonmenu mode.
- 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 ————————┤
```

Nonmenu Mode Example

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
```

In this example, one statement has a misspelled table name. The **receipt** table does not exist.

If your environment does not have **DBACCNOIGN** set, DB-Access inserts two records into the **receipts** table and updates the **customer** table. The decrease in the **customer** balance exceeds the sum of the inserted receipts.

If **DBACCNOIGN** is set to 1, messages display to 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.

IDS

Load Statement Example

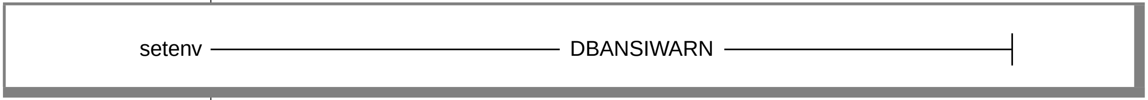
You can set **DBACCNOIGN** to protect data integrity during a LOAD statement, even if DB-Access runs the LOAD statement in menu mode.

Assume you execute the LOAD statement from the DB-Access SQL menu page. Forty-nine rows of data load correctly, but the fiftieth 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 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.



If you set the **DBANSIWARN** environment variable for DB-Access, it is functionally equivalent to including the **-ansi** flag when you invoke the utility from the command line. If you set **DBANSIWARN** before you run DB-Access, warnings are displayed on the screen within the SQL menu.

Set the **DBANSIWARN** environment variable before you compile an Informix ESQL/C program to check for Informix extensions to ANSI standard syntax. When Informix extensions to ANSI standard syntax are encountered in your program at compile time, warning messages are written to the screen.

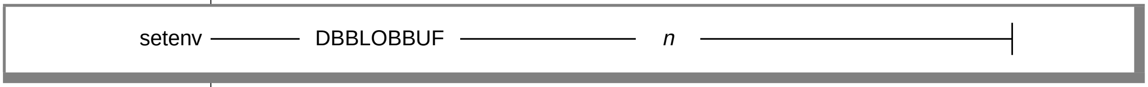
At run time, the **DBANSIWARN** environment variable causes the sixth character of the **sqlwarn** array in the SQL Communication Area (SQLCA) to be set to `W` when a statement is executed that includes any Informix extension to the ANSI standard for SQL syntax. (For more information on SQLCA, see the *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, unset the **DBANSIWARN** environment variable by entering the following command:

```
unsetenv DBANSIWARN
```

DBBLOBBUF

The **DBBLOBBUF** environment variable controls whether TEXT or BYTE data is stored temporarily in memory or in a file while being unloaded with the UNLOAD statement.



n represents the maximum size of TEXT or BYTE data in kilobytes.

If TEXT or BYTE (simple large object) data is smaller than the default of 10 kilobytes or the setting of the **DBBLOBBUF** environment variable, it is temporarily stored in memory. If the TEXT or BYTE data is larger than the default or the setting of the environment variable, it is written to a temporary file. This environment variable applies to the UNLOAD command only.

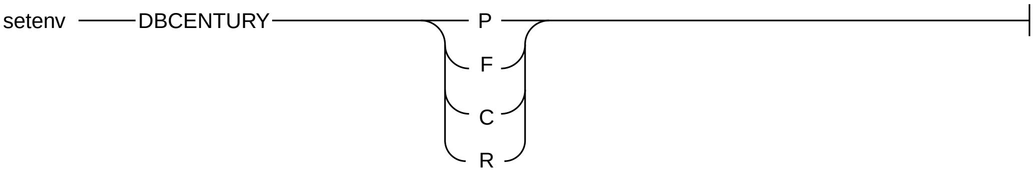
For instance, to set a buffer size of 15 kilobytes, set the **DBBLOBBUF** environment variable as the following example shows:

```
setenv DBBLOBBUF 15
```

In the example, any TEXT or BYTE data that is smaller than 15 kilobytes is stored temporarily in memory. TEXT or BYTE data larger than 15 kilobytes is stored temporarily in a file.

DBCENTURY

The **DBCENTURY** environment variable lets you choose the appropriate expansion for DATE and DATETIME values that have only a one- or two-digit year, such as 4/15/3 or 04/15/99.



By default, if only the decade is provided for a literal DATE or DATETIME value in a table column, the current century is used to expand the year. For example, if today's date is 09/30/1999, the date 12/31/99 expands to 12/31/1999, and 12/31/00 expands to 12/31/1900. **DBCENTURY** algorithms let you determine the century value of a year (the desired expansion of a two-digit year).

Algorithm	**Explanation**
P = Past	The previous and current centuries are used to expand the year value. These two dates are compared against the current date, and the date that is prior to the current date is chosen. If both dates are prior to the current date, the date that is closest to the current date is chosen.
F = Future	The previous and the next centuries are used to expand the year value. These two dates are compared against the current date, and the date that is after the current date is chosen. If both the expansions are after the current date, the date that is closest to the current date is chosen.
C = Closest	The previous, current, and next centuries are used to expand the year value, and the date that is closest to the current date is used.
R = Present	The two high-order digits of the current year are used to expand the year value.

When the **DBCENTURY** environment variable is not set or is set to R, the two high-order digits of the current year in its four-digit form are used to expand the year value. To override the default, specify all four digits.

Tip: *Setting **DBCENTURY** does not affect Informix products when the locale specifies a non-Gregorian calendar such as Hebrew or Islamic. A default century is used for alternate calendar systems when the century is not specified.*

Examples of How the DBCENTURY Environment Variable Expands Date Values

The following examples illustrate how the **DBCENTURY** environment variable expands DATE and DATETIME year formats.

Behavior of DBCENTURY = P

```
Example data type: DATE
Current date: 4/6/1999
User enters: 1/1/1
DBCENTURY = P, Past century algorithm
Previous century expansion : 1/1/1801
Present century expansion: 1/1/1901
Analysis: Both results are prior to the current date, but 1/1/1901 is closer to
the current date. 1/1/1901 is chosen.
```

Behavior of DBCENTURY = F

```
Example data type: DATETIME year to month
Current date: 5/7/2005
User enters: 1-1
DBCENTURY = F, Future century algorithm
Present century expansion: 2001-1
Next century expansion: 2101-1
Analysis: Only date 2101-1 is after the current date, so it is chosen as the
expansion of the year value.
```

Behavior of DBCENTURY = C

```
Example data type: DATE
Current date: 4/6/1999
User enters: 1/1/1
DBCENTURY = C, Closest century algorithm
Previous century expansion : 1/1/1801
Present century expansion: 1/1/1901
Next century expansion: 1/1/2001
Analysis: Because the next century expansion is the closest to the current date
1/1/2001 is chosen.
```

Behavior of DBCENTURY = R or DBCENTURY Not Set

```
Example data type: DATETIME year to month
Current date: 4/6/1999
User enters: 1-1
DBCENTURY = R, Present century algorithm
Present century expansion: 1901-1

Example data type: DATE
Current date: 4/6/2003
User enters: 0/1/1
DBCENTURY = not set
Present century expansion: 2000/1
Analysis: In both examples, the Present algorithm is used.
```

Important: *The interpretation of the variables P, F, and C will always vary with the current date. In the first example, 1/1/1 will be expanded as 1/1/2001 if the current date is 4/6/2001 and DBCENTURY = P. In the second example, 1/1 will be expanded as 2001/1 if the current date is 5/7/1995 and DBCENTURY = F.*

Behavior of DBCENTURY with Expressions that Contain Date Values

This section describes how the database server uses the **DBCENTURY** environment variable to interpret dates in fragmentation expressions, triggers, check constraints, and UDRs.

When an expression (check constraint, fragmentation expression, trigger, or UDR) contains a date value in which the year has 1 or 2 digits, the database server uses the setting of **DBCENTURY** at the time the database object (table, trigger, or UDR) is created. The setting of **DBCENTURY** at creation time is used to expand date values during execution of UDRs and triggers and during evaluation of date values within check constraints and fragment expressions.

For example, suppose a user creates a table and defines the following check constraint on a column named **birthdate**:

```
birthdate < '01/25/50'
```

The preceding expression is interpreted according to the value of **DBCENTURY** when the constraint is defined. If the table that contains the **birthdate** column is created on 06/29/98 and `DBCENTURY=C`, the check constraint expression is consistently interpreted as `birthdate < '01/25/1950'` regardless of the value of **DBCENTURY** when inserts or updates are performed on the **birthdate** column. In other words, even if different values of **DBCENTURY** are set when users perform inserts or updates on the **birthdate** column, the check constraint expression is interpreted according to the **DBCENTURY** setting at the time the check constraint is defined.

The value of **DBCENTURY** and the current date are not the only factors that determine how the database server interprets a date expression. The **DBDATE**, **DBTIME**, **GL_DATE**, and **GL_DATETIME** environment variables also influence how dates are interpreted. For information about **GL_DATE** and **GL_DATETIME**, see the *Informix Guide to GLS Functionality*.

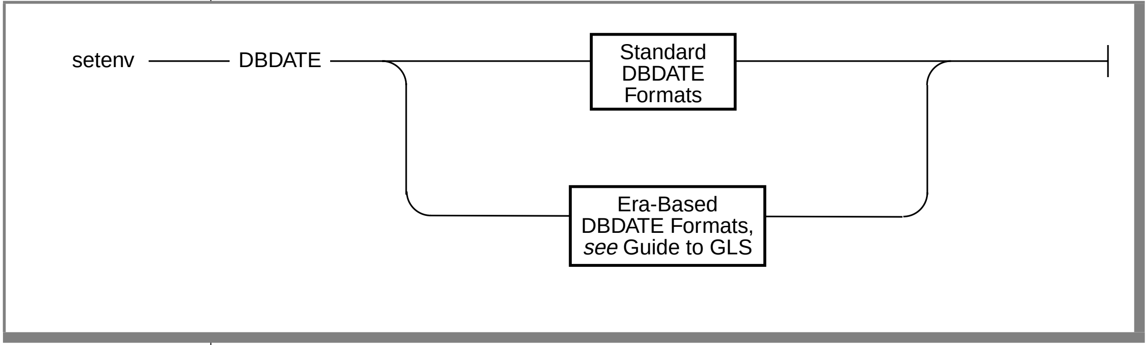
Important: *The behavior of* ***DBCENTURY*** *for Dynamic Server* and *Extended Parallel Server is not backwards compatible.*

DBDATE

The **DBDATE** environment variable specifies the end-user formats of DATE values. End-user formats affect the following situations:

- When you input DATE values, Informix products use the **DBDATE** environment variable to interpret the input.

  For example, if you specify a literal DATE value in an INSERT statement, Informix database servers expect this literal value to be compatible with the format that **DBDATE** specifies. Similarly, the database server interprets the date that you specify as input to the DATE() function in the format that the **DBDATE** environment variable specifies.
- When you display DATE values, Informix products use the **DBDATE** environment variable to format the output.



Standard Formats for DBDATE

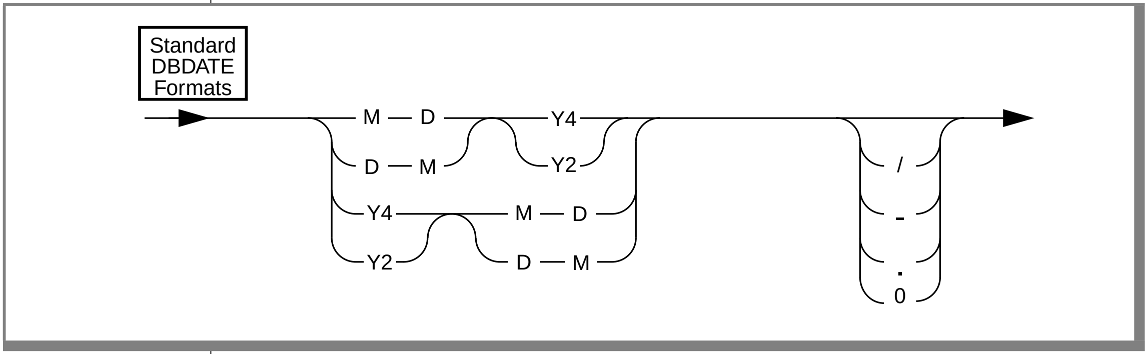
This section describes standard **DBDATE** formats.

GLS

For a description of era-based formats, see the *Informix Guide to GLS Functionality*. ♦

With standard formats, you can specify the following attributes:

- The order of the month, day, and year in a date
- Whether the year should be printed with two digits (Y2) or four digits (Y4)
- The separator between the month, day, and year



- . /	are characters that can be used as separators in a date format.
0	indicates that no separator is displayed.
D, M	are characters that represent the day and the month.
Y2, Y4	are characters that represent the year and the number of digits in the year.

For the U.S. ASCII English locale, the default setting for **DBDATE** is `MDY4/`, where `M` represents the month, `D` represents the day, `Y4` represents a four-digit year, and slash (/) is a separator (for example, `01/08/1999`).

Other acceptable characters for the separator are a hyphen (-), a period (.), or a zero (0). To indicate no separator, use the zero.

The slash (/) appears if you attempt to use a character other than a hyphen, period, or zero as a separator, or if you do not include a separator character in the **DBDATE** definition.

The following table shows some variations for setting the **DBDATE** environment variable.

Variation	January 8, 1999, appears as:
MDY4/	01/08/1999
DMY2-	08-01-99
MDY4	01/08/1999
Y2DM.	99.08.01
MDY20	010899
Y4MD*	1999/01/08

The formats `Y4MD*` (the asterisk is an unacceptable separator) and `MDY4` (no separator is defined) both display the default (slash) as a separator.

Important: *If you use the Y2 format, the setting of the* ***DBCENTURY*** *environment variable affects how the DATE values are expanded.*

Also, certain routines that 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 on page 3-54 and the* ***"Informix ESQL/C Programmer's Manual****."*

GLS

The setting of the **DBDATE** variable takes precedence over that of the **GL_DATE** environment variable, as well as over the default DATE formats that **CLIENT_LOCALE** specifies. For information about the **GL_DATE** and **CLIENT_LOCALE** environment variables, see the *Informix Guide to GLS Functionality*. ♦

Behavior of DBDATE with Expressions that Contain Date Values

This section describes how the database server uses the **DBDATE** environment variable to interpret date values in UDRs, triggers, fragmentation expressions, and check constraints.

When an expression (UDR, trigger, check constraint, or fragmentation expression) contains a date value, the database server uses the setting of **DBDATE** at the time the database object (UDR, trigger, or table) is created. The value of **DBDATE** at creation time determines the format for date values during execution of UDRs and triggers and during evaluation of date values within check constraints and fragment expressions.

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 set 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/`.

***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 by 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 the characters in the following list:

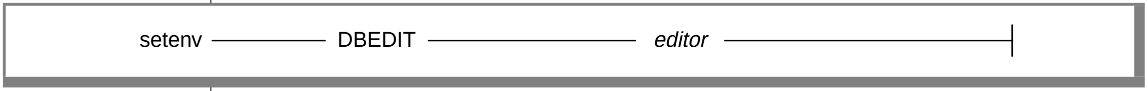
- Hexadecimal numbers (0 through 9, a through f, A through F)
- Newline or CTRL-J
- The backslash symbol (\)

The vertical bar (`|=ASCII 124`) is the default. To change the field delimiter to a plus (+), set the **DBDELIMITER** environment variable, as the following example shows:

```
setenv DBDELIMITER '+'
```

DBEDIT

The **DBEDIT** environment variable lets you name the text editor that you want to use to work with SQL statements and command files in DB-Access. If **DBEDIT** is set, the specified text editor is called directly. If **DBEDIT** is not set, you are prompted to specify a text editor as the default for the rest of the session.



editor is the name of the text editor you want to use.

For most 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 Informix product.

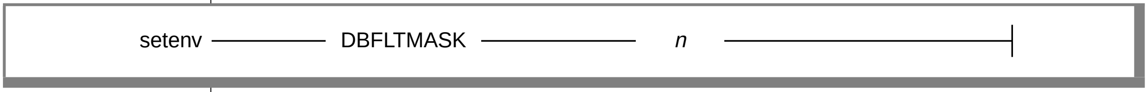
To specify the EMACS text editor, set the **DBEDIT** environment variable by entering the following command:

```
setenv DBEDIT emacs
```

DBFLTMASK

The DB-Access utility displays the floating-point values of data types FLOAT, SMALLFLOAT, and DECIMAL 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 that display to the right of the decimal point for floating-point values, you can set the **DBFLTMASK** environment variable to the number of digits desired.

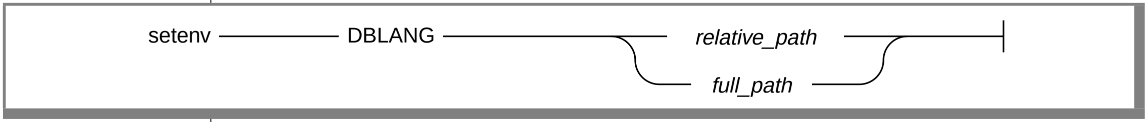


n	is the number of decimal digits that you want the Informix client application to display in the floating-point values. n must be smaller than 16, the default number of digits displayed.

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. However, if you set **DBFLTMASK** to a value greater than can fit into the 14-character buffer, 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 Informix product uses.



relative_path	is the subdirectory of **$INFORMIXDIR**.
full_path	is the full pathname of the directory that contains the compiled message files.

By default, Informix products put compiled messages in a locale-specific subdirectory of the **$INFORMIXDIR/msg** directory. These compiled message files have the suffix **.iem**. If you want to use a message directory other than **$INFORMIXDIR/msg**, where, for example, you can store message files that you create, perform the following steps:

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 directory is a subdirectory of **$INFORMIXDIR** or **$INFORMIXDIR/msg**, 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`.

Informix products that use the default U.S. ASCII English search for message files in the following order:

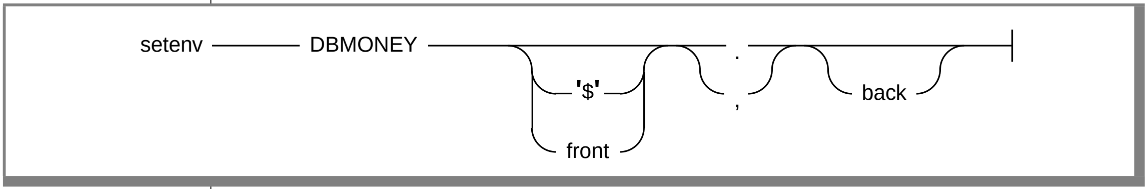
1. In **$DBLANG**, if **DBLANG** is set to a full pathname
2. In **$INFORMIXDIR/msg/$DBLANG**, if **DBLANG** is set to a relative pathname
3. In **$INFORMIXDIR/$DBLANG**, if **DBLANG** is set to a relative pathname
4. In **$INFORMIXDIR/msg/en_us/0333**
5. In **$INFORMIXDIR/msg/en_us.8859-1**
6. In **$INFORMIXDIR/msg**
7. In **$INFORMIXDIR/msg/english**

GLS

For more information on access paths for messages, see the description of **DBLANG** in the *Informix Guide to GLS Functionality*. ♦

DBMONEY

The **DBMONEY** environment variable specifies the display format of monetary values with FLOAT, DECIMAL, or MONEY data types.



$	is the default symbol that precedes the MONEY value.
,	is an optional symbol (comma) that separates the integral from the fractional part of the MONEY value.
.	is the default symbol that separates the integral from the fractional part of the MONEY value.
back	is the optional symbol that follows the MONEY value. The *back* symbol can be up to seven characters and can contain any character except an integer, a comma, or a period. If *back* contains a dollar sign ($), you must enclose the whole string in single quotes (').
front	is the optional symbol that precedes the MONEY value. The *front* symbol can be up to seven characters and can contain any character except an integer, a comma, or a period. If *front* contains a dollar sign ($), you must enclose the whole string in single quotes (').

If you use any character except an alphabetic character for *front* or *back*, you must enclose the character in quotes.

When you display MONEY values, Informix products use the **DBMONEY** environment variable to format the output.

Tip: *The setting of **DBMONEY** does not affect the internal format of the MONEY column in the database.*

If you do not set **DBMONEY**, then MONEY values for the default locale, U.S. ASCII 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, `10050` is formatted as `$100.50`.

Suppose you want to represent MONEY values in DM *(*Deutsche Mark*)*, which uses the currency symbol `DM` and a comma. Enter the following command to set the **DBMONEY** environment variable:

```
setenv DBMONEY DM,
```

Here, `DM` is the 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 amount `10050` is displayed as `DM100,50`.

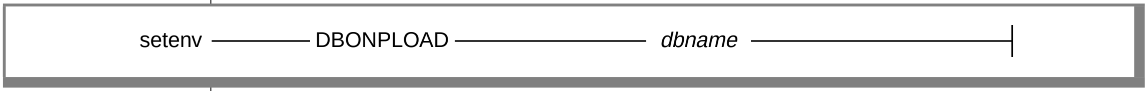
GLS

For more information about how the **DBMONEY** environment variable handles MONEY formats for nondefault locales, see the *Informix Guide to GLS Functionality*. ♦

IDS

DBONPLOAD

The **DBONPLOAD** environment variable specifies the name of the database that the **onpload** utility of the High-Performance Loader (HPL) uses. If the **DBONPLOAD** environment variable is set, the specified name is the name of the database. If the **DBONPLOAD** environment variable is not set, the default name of the database is **onpload**.



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 *Guide to the High-Performance Loader*.

DBPATH

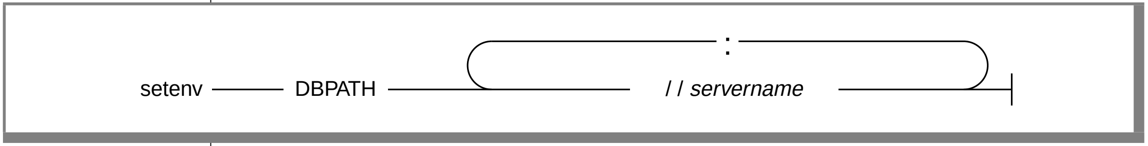
Use **DBPATH** to identify the database servers that contain databases. The **DBPATH** environment variable also specifies a list of directories (in addition to the current directory) in which DB-Access looks for command scripts (**.sql** files).

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 3-11.



servername	is the name of an Informix database server on which databases are stored. You cannot reference database files with a servername.

DBPATH can contain up to 16 entries. Each entry (*full_pathname*, *servername*, or *servername* and *full_pathname*) 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 and / or database server specified in the statement. If no database server is specified, the default database server as set in 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 / or directories in **DBPATH** are searched for in the indicated database. The entries to **DBPATH** are considered in order.

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 have the database server 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 Joachim's and Sonja's 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** environment variable specifies (see page 3-71). 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** to explicitly contain the database server and/or 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 setting allows you to locate only databases and not 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 *Administrator's Guide* and the *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:

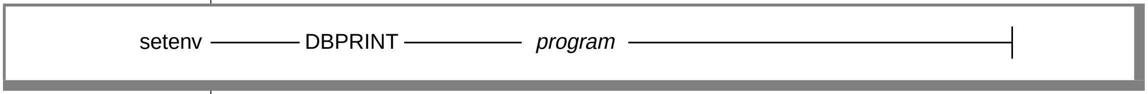
```
setenv DBPATH //marketing
```

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.

UNIX

DBPRINT

The **DBPRINT** environment variable specifies the printing program that you want to use.



program	names any command, shell script, or UNIX utility that handles standard ASCII input.

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 the **myprint** print program:

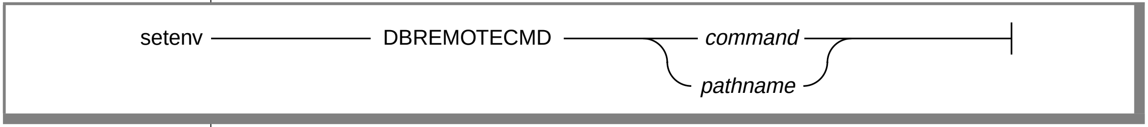
```
setenv DBPRINT myprint
```

UNIX

DBREMOTECMD

You can set the **DBREMOTECMD** environment variable to override the default remote shell used when you perform remote tape operations with the database server.

You can set the **DBREMOTECMD** environment variable with either a simple command or the full pathname. If you use the full pathname, the database server searches your **PATH** for the specified command.



command	is the command to override the default remote shell.
pathname	is the pathname to override the default remote shell.

Informix highly recommends the use of the full pathname syntax on the interactive UNIX platform to avoid problems with similarly named programs in other directories and possible confusion with the *restricted shell* (**/usr/bin/rsh**).

Enter the following command to set the **DBREMOTECMD** environment variable for a simple command name:

```
setenv DBREMOTECMD rcmd
```

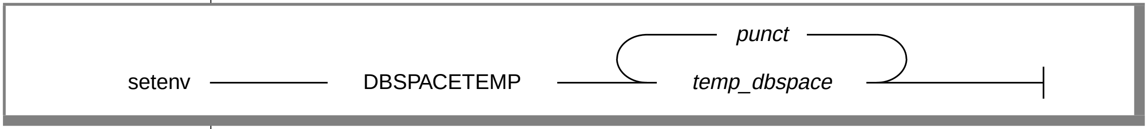
Enter the following command to set the **DBREMOTECMD** environment variable to specify the full pathname:

```
setenv DBREMOTECMD /usr/bin/remsh
```

For more information on **DBREMOTECMD**, see the discussion in your *Archive and Backup Guide* about how to use remote tape devices with your database server for archives, restores, and logical-log backups.

DBSPACETEMP

You can set your **DBSPACETEMP** environment variable to specify the dbspaces in which temporary tables are to be built. You can specify multiple dbspaces to spread temporary space across any number of disks.



punct	can be either colons or commas.
temp_dbspace	is a valid existing temporary dbspace.

The **DBSPACETEMP** environment variable overrides the default dbspaces that the DBSPACETEMP configuration parameter specifies in the configuration file for your database server.

Important: *The dbspaces that you list in* ***DBSPACETEMP*** *must be composed of chunks that are allocated as raw UNIX devices.*

For example, you might set the **DBSPACETEMP** environment variable with the following command:

```
setenv DBSPACETEMP sorttmp1:sorttmp2:sorttmp3
```

Separate the dbspace 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 dbspace specified by the environment variable if the dbspace 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 the **DBSPACETEMP** environment variable 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 dbspace for the table either in the IN *dbspace* clause or in the FRAGMENT BY clause, the database server uses the settings in the **DBSPACETEMP** environment variable 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 the **DBSPACETEMP** environment variable to determine where to create the table. If the **DBSPACETEMP** environment variable is not set, the database server uses the ONCONFIG parameter DBSPACETEMP. If this parameter is not set, the database server creates the explicit temporary table in the same dbspace where the database resides.

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 these implicit temporary tables, the database server uses disk space for writing the temporary data, in the following order:

UNIX

1. The operating-system directory or directories that the environment variable **PSORT_DBTEMP** specifies, if it is set. ♦
2. The dbspace or dbspaces that the environment variable **DBSPACETEMP** specifies, if it is set.
3. The dbspace or dbspaces that the ONCONFIG parameter DBSPACETEMP specifies.
4. The operating-system file space in **/tmp** (UNIX) or **%temp%** (Windows NT).

If the **DBSPACETEMP** environment variable is set to a nonexistent dbspace, 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.

Important: *If the **DBSPACETEMP** environment variable is set to an invalid value, the database server might fill **/tmp** to the limit and eventually bring down the system or kill the file system.*

DBTEMP

Set the **DBTEMP** environment variable to specify the full pathname of the directory into which you want Informix Enterprise Gateway products to place their temporary files and temporary tables.

```
setenv ────────── DBTEMP ────────── pathname ────────────────┤
```

pathname	is the full pathname of the directory for temporary files and temporary tables.

Set the **DBTEMP** environment variable to specify the pathname **usr/magda/mytemp** by entering the following command:

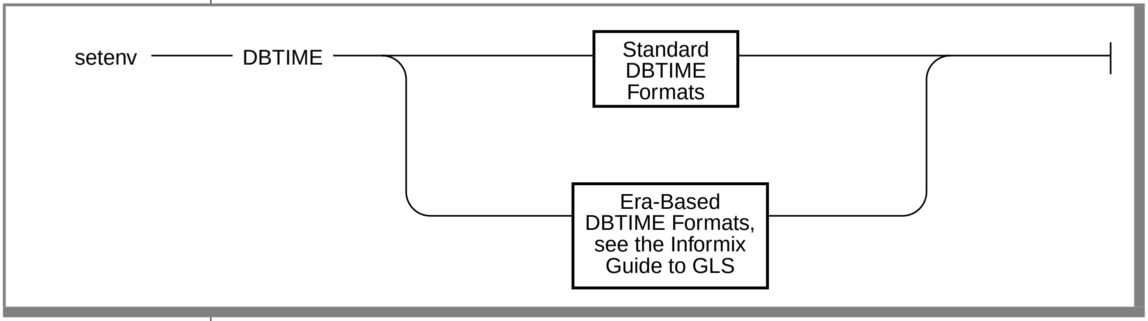
```
setenv DBTEMP usr/magda/mytemp
```

Important: ***DBTEMP*** *must not point to an NFS directory.*

If you do not set **DBTEMP**, temporary files are created in **/tmp**. If **DBTEMP** is not set, temporary tables are created in the directory of the database (that is, the **.dbs** directory). For more information, see your *INFORMIX-Enterprise Gateway User Manual*.

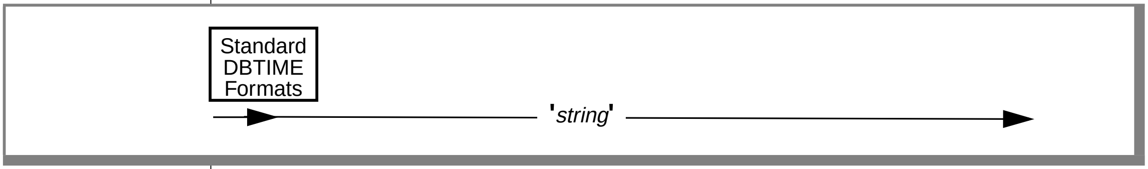
DBTIME

The **DBTIME** environment variable specifies the end-user formats of DATETIME values for a set of SQL API library functions.



You can set the **DBTIME** environment variable to manipulate DATETIME formats so that the formats conform more closely to various international or local TIME conventions. **DBTIME** takes effect only when you call certain Informix ESQL/C DATETIME routines; otherwise, use the **DBDATE** environment variable. (For details, see the *Informix ESQL/C Programmer's Manual*.)

You can set **DBTIME** to specify the exact format of an input/output (I/O) DATETIME string field with the formatting directives described in the following list. Otherwise, the behavior of the DATETIME formatting routine is undefined.



string the formatting directives that you can use, as the following list describes:

%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 the value of the fraction with precision that the integer n specifies. The default value of n is 2; the range of *n* 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 local standards).
%S	is replaced by the second as a decimal number [00,59].
%y	is replaced by the year as a four-digit decimal number. If the user enters a two-digit value, the format of this value is affected by the setting of the **DBCENTURY** environment variable. If **DBCENTURY** is not set, then the current century is used for the century digits.
%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 how to convert a DATETIME YEAR TO SECOND to the following ASCII string format:

```
Mar 21, 1999 at 16 h 30 m 28 s
```

Set **DBTIME** as the following list shows:

```
setenv DBTIME '%b %d, %Y at %H h %M m %S s'
```

The default **DBTIME** produces the conventional ANSI SQL string format that the following line shows:

```
1999-03-21 16:30:28
```

Set the default **DBTIME** as the following example shows:

```
setenv DBTIME '%Y-%m-%d %H:%M:%S'
```

An optional field width and precision specification can immediately follow the percent (%) character; it is interpreted as the following list describes:

[- | 0]*w*	where *w* is a decimal digit string specifying the minimum field width. By default, the value is right justified with spaces on the left. If `-` is specified, it is left justified with spaces on the right. If `0` is specified, it is right justified and padded with zeros on the left.
.p	where p is a decimal digit string specifying the number of digits to appear for `d`, `H`, `I`, `m`, `M`, `S`, `y`, and `Y` conversions, and the maximum number of characters to be used for `b` and `B` conversions. A precision specification is significant only when converting a DATETIME value to an ASCII string and not vice versa.

When you use field width and precision specifications, the following limitations apply:

- If a conversion specification supplies fewer digits than a precision specifies, it is padded with leading zeros.
- If a conversion specification supplies more characters than a precision specifies, excess characters are truncated on the right.
- If no field width or precision is specified for `d`, `H`, `I`, `m`, `M`, `S`, or `y` conversions, a default of `0.2` is used. A default of `0.4` is used for `Y` conversions.

The `F` conversion does not follow the field width and precision format conversions that are described earlier.

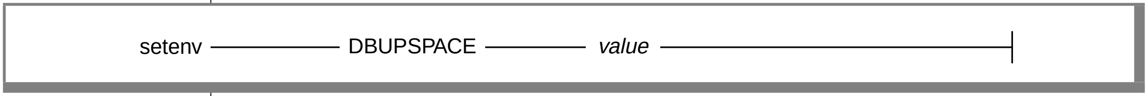
For related information, see the discussion of **DBDATE** on page 3-37.

GLS

For more information about how the **DBTIME** environment variable handles time formats for nondefault locales, see the *Informix Guide to GLS Functionality*. ♦

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.



value	represents a disk space amount in kilobytes.

For example, to set **DBUPSPACE** to 2,500 kilobytes, enter the following command:

```
setenv DBUPSPACE 2500
```

Once 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 try 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 for the one is greater than specified in **DBUPSPACE**.

DELIMIDENT

The **DELIMIDENT** environment variable specifies that strings set off by double quotes are delimited identifiers.

```
setenv ———— DELIMIDENT ———— value ————————————————┤
```

You can use delimited identifiers to specify identifiers that are identical to reserved keywords, such as TABLE or USAGE. You can also use them to specify database identifiers that contain nonalpha characters, but you cannot use them to specify storage identifiers that contain nonalpha characters. Database identifiers are names for database objects such as tables and columns, and storage identifiers are names for storage objects such as dbspaces and partition simple large objects.

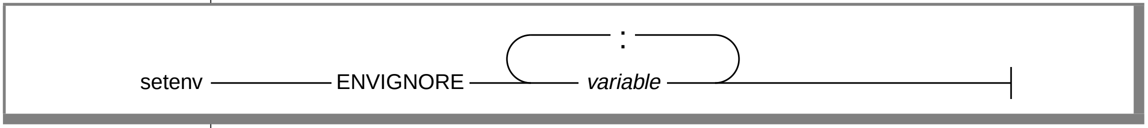
Delimited identifiers are case sensitive.

To use delimited identifiers, applications in ESQL/C must set the **DELIMIDENT** environment variable at compile time and execute time.

UNIX

ENVIGNORE

Use the **ENVIGNORE** environment variable to deactivate specified environment variable entries in the common (shared) and private environment-configuration files, **informix.rc** and **.informix** respectively.



variable is the list of environment variables that you want to deactivate.

For example, to ignore the **DBPATH** and **DBMONEY** entries in the environment-configuration files, enter the following command:

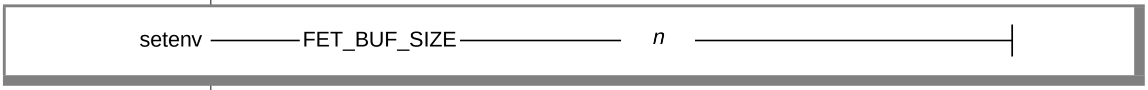
```
setenv ENVIGNORE DBPATH:DBMONEY
```

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 an Environment-Configuration File" on page 3-8.

ENVIGNORE cannot be set in an environment-configuration file.

FET_BUF_SIZE

The **FET_BUF_SIZE** environment variable lets you override the default setting for the size of the fetch buffer for all data except simple large objects. When set, **FET_BUF_SIZE** is effective for the entire environment.



n	represents the size of the buffer in bytes.

When set to a valid value, the environment variable overrides the previously set value. The default setting for the fetch buffer is dependent on row size.

If the buffer size is set to less than the default size or is out of the range of the small integer value, no error is raised. The new buffer size is ignored.

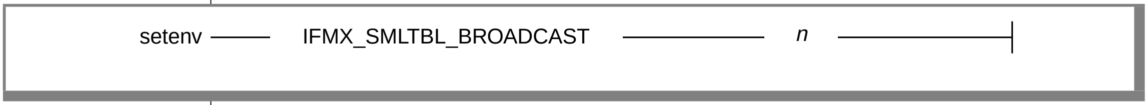
For example, to set a buffer size to `5,000` bytes, set the **FET_BUF_SIZE** environment variable by entering the following command:

```
setenv FET_BUF_SIZE 5000
```

XPS

IFMX_SMLTBL_BROADCAST_SIZE

The **IFMX_SMLTBL_BROADCAST_SIZE** environment variable setting determines the threshold size of tables that are used in Small Table Broadcast when the table size exceeds 128 kilobytes. The **IFMX_SMLTBL_BROADCAST_SIZE** environment variable is set on the database server.



n represents the size of the table in kilobytes.

Important: *Query performance can suffer if the **IFMX_SMLTBL_BROADCAST** environment variable is set beyond a certain table size. The recommended upper limit on table size depends on your computer and the configuration of your database server.*

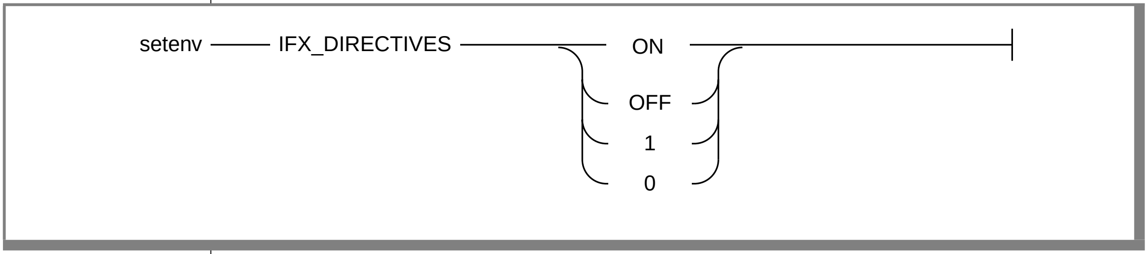
For more information about the **IFMX_SMLTBL_BROADCAST** environment variable, see your documentation notes or release notes.

IDS

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 use either `ON` and `OFF` or `1` and `0` to set the environment variable.



ON	Optimizer directives accepted
OFF	Optimizer directives not accepted
1	Optimizer directives accepted
0	Optimizer directives not accepted

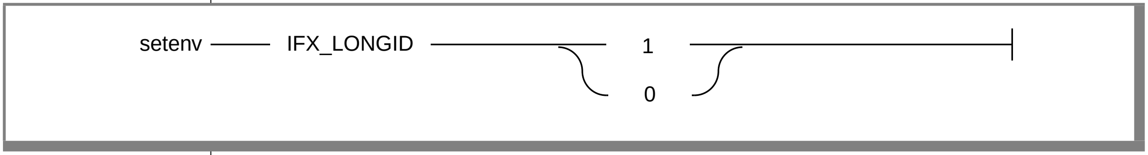
The setting of the **IFX_DIRECTIVES** environment variable overrides the value of the DIRECTIVES configuration parameter that is set for the database server. If the **IFX_DIRECTIVES** environment variable is not set, however, then all client sessions will inherit the database server configuration for directives that the ONCONFIG parameter DIRECTIVES determines. The default setting for the DIRECTIVES parameter is `ON`.

For more information about the DIRECTIVES parameter, see the *Administrator's Reference*. For more information on the performance impact of directives, see your *Performance Guide*.

IDS

IFX_LONGID

Use the **IFX_LONGID** environment variable setting and the version number of the client application to determine whether a particular client application is capable of handling long identifiers. Valid **IFX_LONGID** values are 1 and 0.



1	Client can handle long identifiers
0	Client cannot handle long identifiers

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 version is >= 9.0304, the client is considered *new* and thus 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.

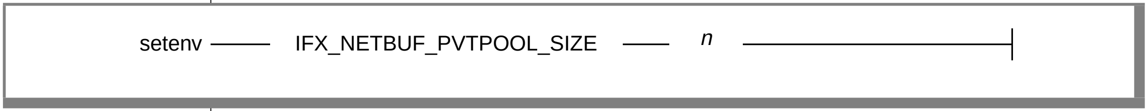
If you set the **IFX_LONGID** environment variable at the client, the setting affects only that client. If you bring up the database server with **IFX_LONGID** set, all client applications will adhere to that setting. If **IFX_LONGID** is set at both the client and the database server, the client setting 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.*

UNIX

IFX_NETBUF_PVTPOOL_SIZE

The **IFX_NETBUF_PVTPOOL_SIZE** environment variable specifies the maximum size of the free (unused) private network buffer pool for each database server session.

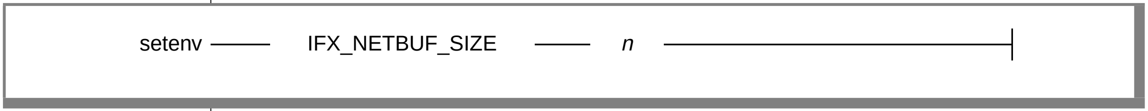


n	represents the number of units (buffers) in the pool.

The default size is 1 buffer. If **IFX_NETBUF_PVTPOOL_SIZE** is set to 0, 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.



n	represents the size (in bytes) for one network buffer.

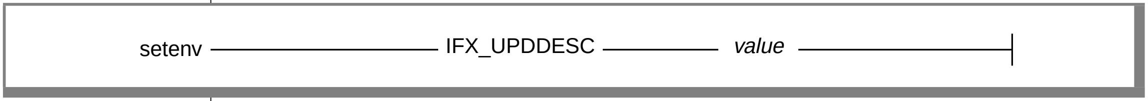
The default size is 4 kilobytes (4,096 bytes). The maximum size is 64 kilobytes (65,536 bytes) and the minimum size is 512 bytes. You can specify the value in hexadecimal or decimal form.

Tip: *You cannot set a different size for each session.*

IDS

IFX_UPDDESC

The **IFX_UPDDESC** environment variable controls the use of the describe-for-updates functionality. You must set **IFX_UPDDESC** at execution time before you do a DESCRIBE of an UPDATE statement.



You can set **IFX_UPDDESC** to any value. A null value (the environment variable is not set) indicates that the feature is not being used. Any nonnull value indicates that the feature is enabled.

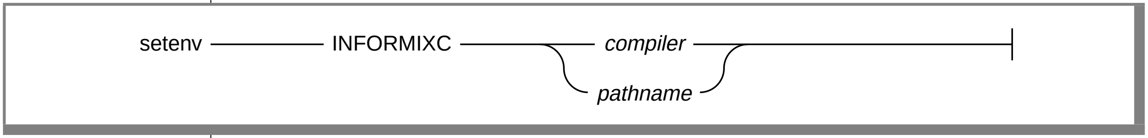
UNIX

INFORMIXC

The **INFORMIXC** environment variable specifies the name or pathname of the C compiler to be used to compile files that Informix ESQL/C generates.

If **INFORMIXC** is not set, the default compiler is cc.

***Tip:** On Windows NT, you pass either **-mcc** or **-bcc** options to the **esql** preprocessor to use either the Microsoft or Borland C compilers.*



compiler	is the name 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:

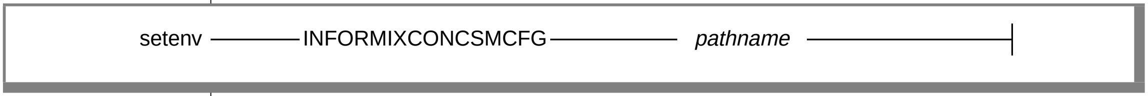
```
setenv INFORMIXC gcc
```

The setting is required only during the C compilation stage.

IDS

INFORMIXCONCSMCFG

The **INFORMIXCONCSMCFG** environment variable specifies the location of the **concsm.cfg** file that describes communications support modules.



pathname specifies the full pathname of the **concsm.cfg** file.

The following sample command specifies that the **concsm.cfg** file is in **/usr/myfiles**:

```
setenv INFORMIXCONCSMCFG /usr/myfiles
```

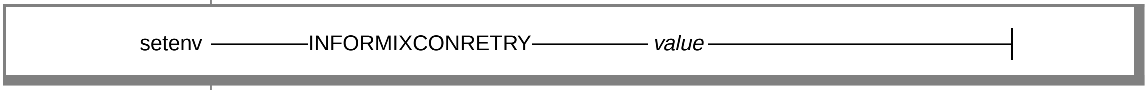
You can also specify a different name for the file. The following example specifies a filename of **csmconfig**:

```
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 *Administrator's Reference*.

INFORMIXCONRETRY

The **INFORMIXCONRETRY** environment variable specifies the maximum number of additional connection attempts that should be made to each server by the client during the time limit that the **INFORMIXCONTIME** environment variable specifies.



value represents the number of connection attempts to each server.

For example, enter the following command to set **INFORMIXCONRETRY** to three additional connection attempts (after the initial attempt):

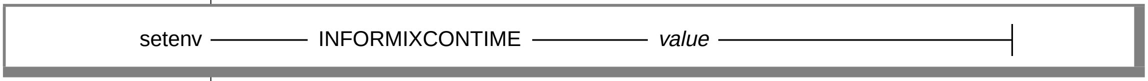
```
setenv INFORMIXCONRETRY 3
```

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 lets you specify that an SQL CONNECT statement should keep trying for at least the given number of seconds before returning an error.

You might encounter connection difficulties related to system or network load problems. For instance, if the database server is busy establishing new SQL client threads, some clients might fail because the database server cannot issue a network function call fast enough. The **INFORMIXCONTIME** and **INFORMIXCONRETRY** environment variables let you configure your client-side connection capability to retry the connection instead of returning an error.



value represents the minimum number of seconds spent in attempts to establish a connection to a database server.

For example, enter the following 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 execution of the CONNECT statement involves searching **DBPATH**, the following rules apply:

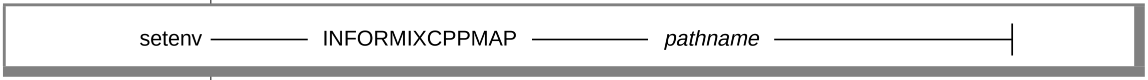
- All appropriate servers in the **DBPATH** setting are accessed at least once, even though the **INFORMIXCONTIME** value might be exceeded. Thus, the CONNECT statement might take longer than the **INFORMIXCONTIME** time limit to return an error that indicates connection failure or that the database was not found.
- The **INFORMIXCONRETRY** value specifies the number of additional connections that should be attempted for each server entry in **DBPATH**.
- The **INFORMIXCONTIME** value is divided among the number of 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.

The default value for **INFORMIXCONTIME** is 15 seconds. The setting for **INFORMIXCONTIME** takes precedence over the **INFORMIXCONRETRY** setting. Retry efforts could end after the **INFORMIXCONTIME** value is exceeded but before the **INFORMIXCONRETRY** value is reached.

IDS

INFORMIXCPPMAP

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.

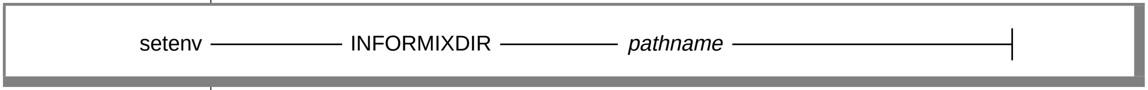


pathname	is the directory path where the C++ map file is stored.

The map file is a text file that can have any file name. The **INFORMIXCPPMAP** setting can include several map files separated by colons (:) on UNIX or semicolons (;) on Windows NT. On UNIX, the default map file is **$INFORMIXDIR/etc/c++map**. On Windows NT, 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 the **INFORMIXDIR** environment variable. Verify that the **INFORMIXDIR** environment variable 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 the **INFORMIXDIR** environment variable, see your *Installation Guide*.



pathname	is the directory path where the product files are installed.

To set the **INFORMIXDIR** environment variable to the desired installation directory, enter the following command:

```
setenv INFORMIXDIR /usr/informix
```

UNIX

INFORMIXKEYTAB

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 server and to acquire a credential for it. A client application uses the key if the user did not do **dce_login** with the current operating-system user name (which is the same as the DCE principle 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**:

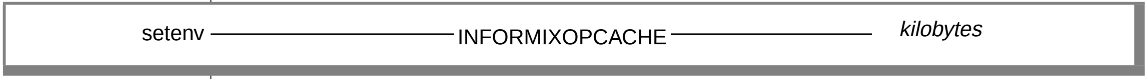
```
setenv INFORMIXKEYTAB /usr/myfiles/mykeytab
```

For more information about the DCE-GSS communications support module, see the *Administrator’s Guide*.

IDS

INFORMIXOPCACHE

The **INFORMIXOPCACHE** environment variable lets you specify the size of the memory cache for the staging-area blobspace of the client application.



kilobytes specifies the value you set for the optical memory cache.

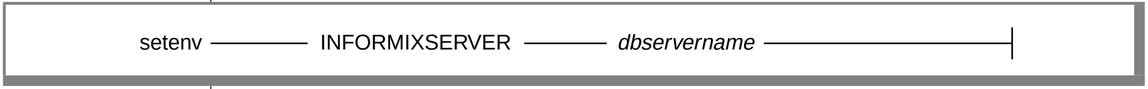
Set the **INFORMIXOPCACHE** environment variable by specifying the size of the memory cache in kilobytes. The specified size must be equal to or smaller than the size of the system-wide configuration parameter, OPCACHEMAX.

If you do not set the **INFORMIXOPCACHE** environment variable, 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 or the DB-Access utility.

The database server can be either local or remote. You must always set **INFORMIXSERVER** before you use an Informix product.



dbservername is the name of the default database server.

The value of **INFORMIXSERVER** must correspond to a valid *dbservername* entry in the **$INFORMIXDIR/etc/sqlhosts** file on the computer running the application. The *dbservername* must be specified using lowercase characters and cannot exceed 128 characters. For example, to specify the **coral** database server as the default for connection, enter the following command:

```
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.*

XPS

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
```

In the coserver name, dbservername is the value that you assigned to the DBSERVERNAME configuration parameter when you prepared 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. However, if **INFORMIXSERVER** is not set, Extended Parallel Server does not build the **sysmaster** tables. ♦

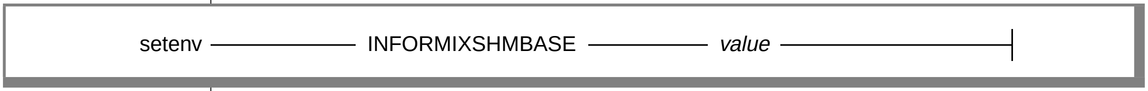
UNIX

INFORMIXSHMBASE

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**.*

Use **INFORMIXSHMBASE** to specify where shared-memory communication segments are attached to the client process so that client applications can avoid collisions with other memory segments that the application uses. If you do not set **INFORMIXSHMBASE**, the memory address of the communication segments defaults to an implementation-specific value such as 0x800000.



value	is 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, to set the memory address to the value `0x800000`, set the **INFORMIXSHMBASE** environment variable by entering the following command:

```
    setenv INFORMIXSHMBASE 8192
```

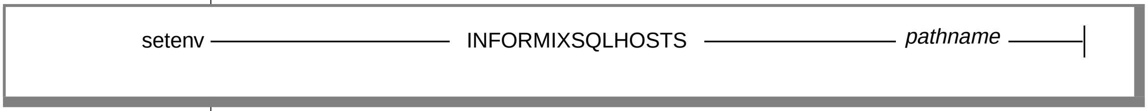
For more information, see your *Administrator's Guide* and the *Administrator's Reference*.

INFORMIXSQLHOSTS

The **INFORMIXSQLHOSTS** environment variable specifies the full pathname to the place where client-database server connectivity information is stored. On UNIX, by default, this environment variable points to the **$INFORMIXDIR/etc/sqlhosts** file. For example, to specify that the client or database server will look for connectivity information in the **mysqlhosts** file in the / **work** / **envt** directory, enter the following command:

```
setenv INFORMIXSQLHOSTS /work/envt/mysqlhosts
```

When the **INFORMIXSQLHOSTS** environment variable is set, the client or database server looks in the specified file for connectivity information. When the **INFORMIXSQLHOSTS** environment variable is not set, the client or database server looks in the **$INFORMIXDIR/etc/sqlhosts** file.



pathname	specifies the full pathname and filename of the file that contains connectivity information.

On Windows NT, by default, this environment variable points to the computer whose registry contains the SQLHOSTS subkey. To specify that the client or database server look for connectivity information on a computer named **arizona**, enter the following command:

```
set INFORMIXSQLHOSTS = \\arizona
```

For a description of the SqlHosts information, see your *Administrator's Guide*.

INFORMIXSTACKSIZE

INFORMIXSTACKSIZE specifies the stack size (in kilobytes) that the database server uses for a particular client session.

Use **INFORMIXSTACKSIZE** to override the value of the ONCONFIG parameter STACKSIZE for a particular application or user.

```
setenv ───────── INFORMIXSTACKSIZE ────── value ──────────────┤
```

value	is the stack size for SQL client threads in kilobytes.

For example, to decrease the **INFORMIXSTACKSIZE** to 20 kilobytes, enter the following command:

```
setenv INFORMIXSTACKSIZE 20
```

If **INFORMIXSTACKSIZE** is not set, the stack size is taken from the database server configuration parameter STACKSIZE, or it defaults to a platform-specific value. The default stack size value for the primary thread for an SQL client is 32 kilobytes for nonrecursive database activity.

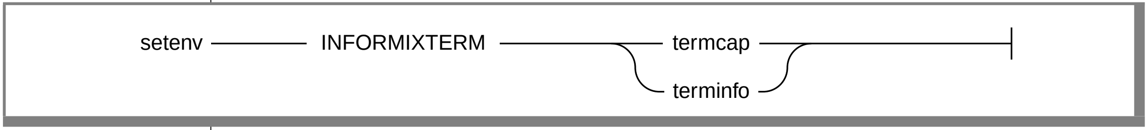
Warning: *For specific instructions on setting this value, see the "Administrator's Reference." If you incorrectly set the value of **INFORMIXSTACKSIZE**, it can cause the database server to fail.*

UNIX

INFORMIXTERM

The **INFORMIXTERM** environment variable specifies whether DB-Access should use the information in the **termcap** file or the **terminfo** directory.

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.



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 Informix 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 3-97.

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.

IDS

INF_ROLE_SEP

The **INF_ROLE_SEP** environment variable configures the security feature of role separation when the database server is installed. Role separation enforces separating administrative tasks that different people who are involved in running and auditing the database server perform.

Tip: *To enable role separation for database servers on Windows NT, choose the role-separation option during installation.*

If **INF_ROLE_SEP** is set, role separation is implemented and a separate group is specified to serve each of the following responsibilities: the database system security officer (DBSSO), the audit analysis officer (AAO), and the standard user. If **INF_ROLE_SEP** is not set, user **informix** (the default) can perform all administrative tasks.

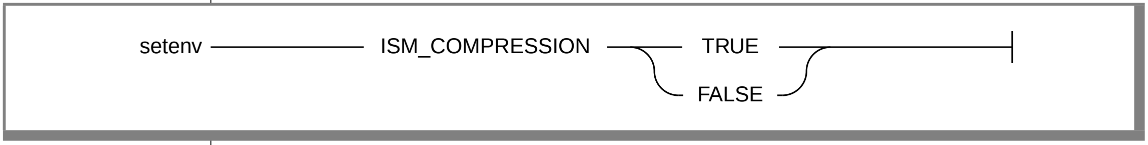
```
setenv ──────────────────── INF_ROLE_SEP ─────────────────    n    ───────┤
```

n	is any positive integer.

For more information about the security feature of role separation, see the *Trusted Facility Manual*. To learn how to configure role separation when you install your database server, see your *Installation Guide*.

ISM_COMPRESSION

Set the **ISM_COMPRESSION** environment variable in the ON-Bar environment to specify whether the Informix Storage Manager (ISM) should use data compression.

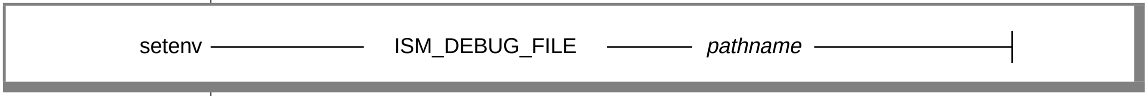


If **ISM_COMPRESSION** is set to TRUE in the environment of the ON-Bar process that makes a request, the ISM server uses a data-compression algorithm to store or retrieve the data specified in that request.

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 Informix Storage Manager server environment to specify where the XBSA messages should be written.



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 **$INFORMIXDIR/ism/applogs/xbsa.messages** on UNIX or **c:\nsr\applogs\xbsa.messages** on Windows NT.

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 ───────────────────┤
```

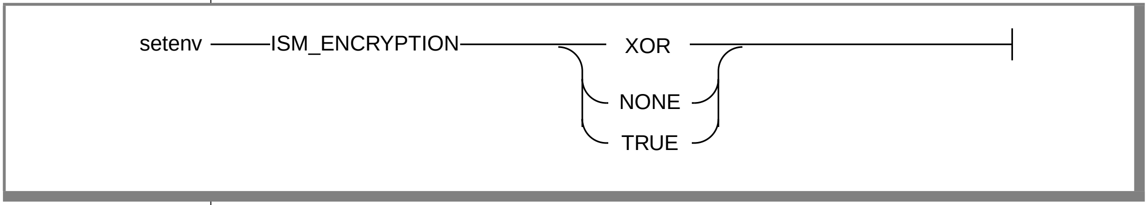
value	specifies the level of reporting detail.

You can specify a value between 0 and 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 the Informix Storage Manager (ISM) should use data encryption.



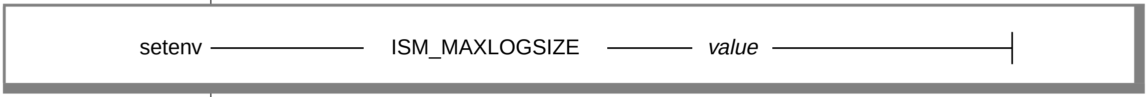
XOR	uses encryption.
NONE	does not use encryption.
TRUE	uses encryption.

If the **ISM_ENCRYPTION** environment variable 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.

If **ISM_ENCRYPTION** is set to NONE or is not set, the ISM server does not use encryption.

ISM_MAXLOGSIZE

Set the **ISM_MAXLOGSIZE** environment variable in the Informix Storage Manager (ISM) server environment to specify the size threshold of the ISM activity log.

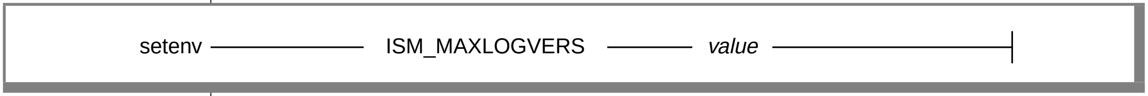


value	specifies the size threshold of the activity log.

If **ISM_MAXLOGSIZE** is not set, the default size limit is 1 megabyte. If **ISM_MAXLOGSIZE** is set to a null value, the threshold is 0 bytes.

ISM_MAXLOGVERS

Set the **ISM_MAXLOGSVERS** environment variable in the Informix Storage Manager (ISM) server environment to specify the maximum number of activity-log files to be preserved by the ISM server.



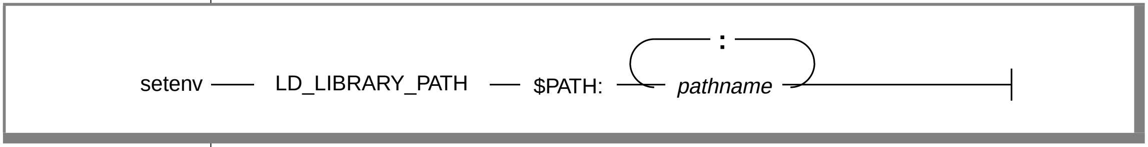
value	specifies the number of files to be preserved.

If **ISM_MAXLOGSVERS** is not set, the default number of files to be preserved is four. If **ISM_MAXLOGSVERS** is set to a null value, the ISM server preserves no activity log files.

UNIX

LD_LIBRARY_PATH

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.



pathname	specifies the search path for the library.

The following example sets the **LD_LIBRARY_PATH** environment variable to the desired directory:

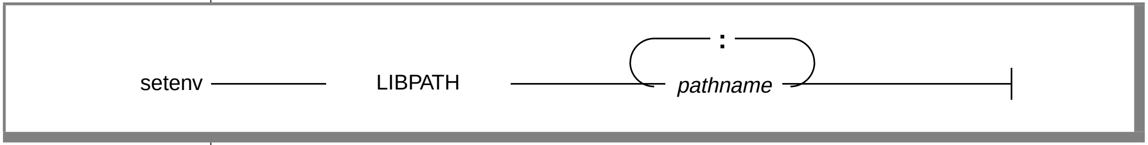
```
setenv LD_LIBRARY_PATH
${INFORMIXDIR}/lib:${INFORMIXDIR}/lib/esql:$LD_LIBRARY_PATH
```

For INTERSOLV DataDirect ODBC Driver on AIX, set **LIBPATH**. For INTERSOLV DataDirect ODBC Driver on HP-UX, set **SHLIB_PATH**.

UNIX

LIBPATH

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.

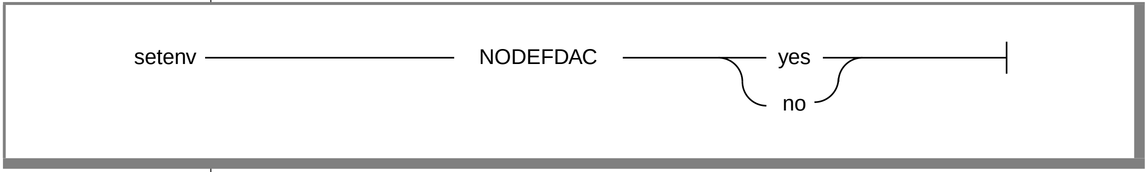


pathname	specifies the search path for the libraries.

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 in a database that is not ANSI compliant. If you do not set the **NODEFDAC** variable, it is, by default, set to `no`.



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 when the routine is created in owner mode.
no	allows default table privileges to be granted to PUBLIC. Also allows the Execute privilege on a new user-defined routine to be granted to PUBLIC when the user-defined routine is created in owner mode.

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 the **ONCONFIG** environment variable to the name of the 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.

```
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. Informix recommends 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.

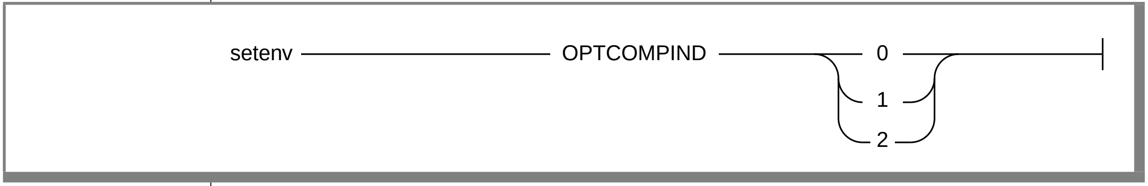
XPS

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. ♦

For more information on configuration parameters and the ONCONFIG file, see the *Administrator's Reference*.

OPTCOMPIND

You can set the **OPTCOMPIND** environment variable so that the optimizer can select the appropriate join method.



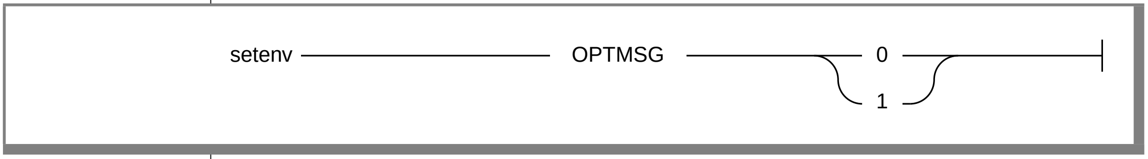
0	A nested-loop join is preferred, where possible, over a sort-merge join or a hash join.
1	When the transaction isolation mode is *not* Repeatable Read, the optimizer behaves as in setting 2; otherwise, the optimizer behaves as in setting 0.
2	Nested-loop joins are not necessarily preferred. The optimizer bases its decision purely on costs, regardless of transaction isolation mode.

When the **OPTCOMPIND** environment variable is not set, the database server uses the value specified for the ONCONFIG configuration parameter OPTCOMPIND. When neither the environment variable nor the configuration parameter is set, the default value is 2.

For more information on the ONCONFIG configuration parameter OPTCOMPIND, see the *Administrator's Reference*. For more information on the different join methods for your database server that the optimizer uses, see your *Performance Guide*.

OPTMSG

Set the **OPTMSG** environment variable at runtime before you start an Informix ESQL/C application to enable optimized message transfers (message chaining) for all SQL statements in an application. You also can 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 exits.



0 disables optimized message transfers.

1 enables optimized message transfers and implements the feature for any subsequent connection.

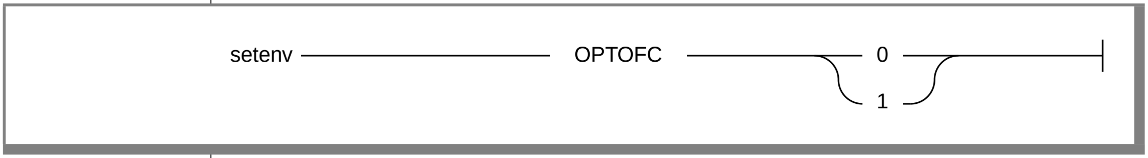
The default value is 0 (zero), which explicitly disables message chaining.

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 *Informix ESQL/C Programmer's Manual*.

OPTOFC

Set the **OPTOFC** environment variable to enable optimize-OPEN-FETCH-CLOSE functionality in an Informix ESQL/C application that uses DECLARE and OPEN statements to execute a cursor. The **OPTOFC** environment variable reduces the number of message requests between the application and the database server.



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).

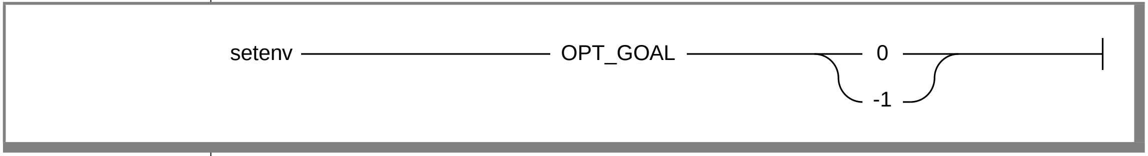
If you set **OPTOFC** from the shell, you must set it before you start the ESQL/C application.

For more information about enabling **OPTOFC** and related features, see the *Informix ESQL/C Programmer's Manual*.

UNIX

OPT_GOAL

Set the **OPT_GOAL** environment variable in the user environment, before you start an application, to specify the query performance goal for the optimizer.



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.

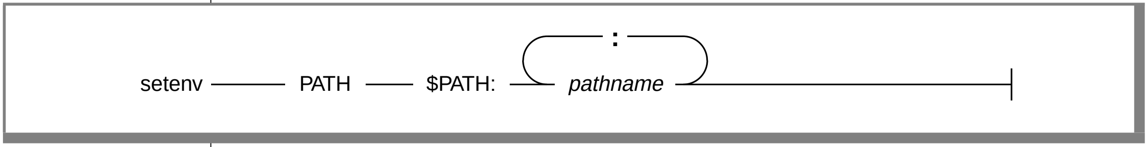
You can also specify the optimization goal for individual queries with optimizer directives or for a session with the SET OPTIMIZATION statement. Both methods take precedence over the **OPT_GOAL** environment variable setting. You can also set the OPT_GOAL configuration parameter for the Dynamic Server system; this method has the lowest level of precedence.

For more information about optimizing queries for your database server, see your *Performance Guide*. For information on the SET OPTIMIZATION statement, see the *Informix Guide to SQL: Syntax*.

PATH

The **PATH** environment variable tells the operating system where to search for executable programs. You must include the directory that contains your Informix product to your **PATH** environment variable before you can use the product. This directory should appear before **$INFORMIXDIR/bin**, which you must also include.

The UNIX **PATH** environment variable tells the shell which directories to search for executable programs. You must add the directory that contains your Informix product to your **PATH** environment variable before you can use the product.



pathname specifies the search path for the executables.

You can specify the correct search path in various ways. Be sure to include a colon between the directory names.

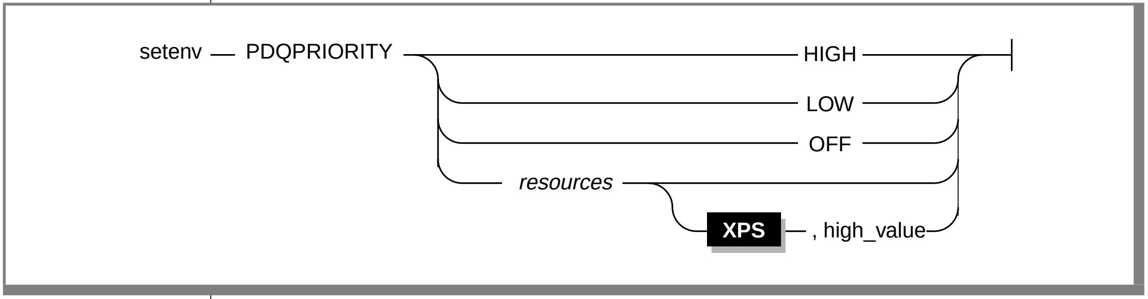
For additional information about how to modify your path, see "Modifying an Environment-Variable Setting" on page 3-11.

PDQPRIORITY

The **PDQPRIORITY** 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.

XPS

For Extended Parallel Server, the **PDQPRIORITY** environment variable determines only the allocation of memory resources. ♦



HIGH	When the database server allocates resources among all users, it gives as many resources as possible to the query.
LOW	Data is fetched from fragmented tables in parallel. For supported database servers other than Extended Parallel Server, when you specify LOW, the database server uses no forms of parallelism. For Extended Parallel Server, other parallel operations can occur when the setting is LOW.

OFF	PDQ processing is turned off (for supported database servers other than Extended Parallel Server).
resources	Integer value in the range 0 to 100. Value 0 is the same as OFF (for supported database servers other than Extended Parallel Server only). Value 1 is the same as LOW. For supported database servers other than Extended Parallel Server, the *resources* value specifies the query priority level and the amount of resources that the database server uses to process the query. For 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.
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 percentage.

IDS

When the **PDQPRIORITY** environment variable is:

- not set, the default value is OFF.
- set to HIGH, the database 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 so on. ♦

XPS

When the **PDQPRIORITY** environment variable is:

- not set, the default value is the value of the PDQPRIORITY configuration parameter.
- set to 0, the database 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 so on. PDQPRIORITY does not affect the degree of parallelism in Extended Parallel Server. ♦

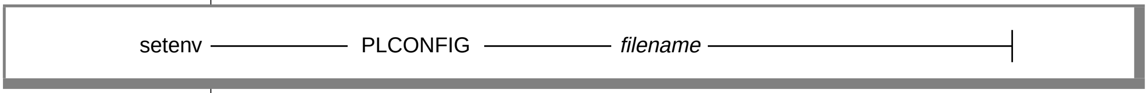
Usually, the more resources a database server uses, the better its performance for a given query. However, if the database server uses too many resources, contention among the resources can result and take resources away from other queries, which results in degraded performance. For more information on performance considerations for **PDQPRIORITY**, refer to your *Performance Guide*.

An application can override the setting of the environment variable when it issues the SQL statement SET PDQPRIORITY, which the *Informix Guide to SQL: Syntax* describes.

IDS

PLCONFIG

The **PLCONFIG** environment variable specifies the name of the configuration file that the High-Performance Loader (HPL) uses. This configuration file must reside in the **$INFORMIXDIR / etc** directory. If the **PLCONFIG** environment variable is not set, **$INFORMIXDIR / etc / plconfig** is the default configuration file.



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:

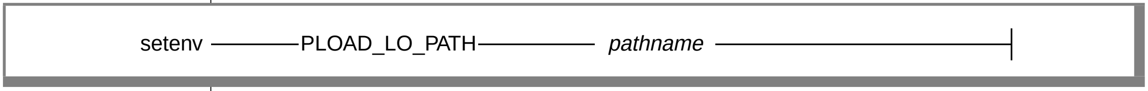
```
setenv PLCONFIG custom.cfg
```

For more information, see the *Guide to the High-Performance Loader*.

IDS

PLOAD_LO_PATH

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, CLOB, and BOOLEAN data types).



pathname	specifies the directory for the smart-large-object handles.

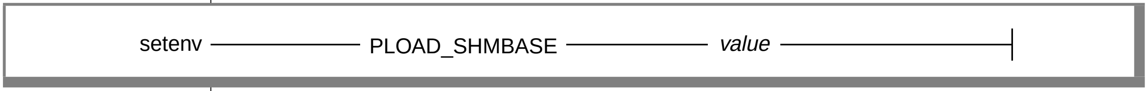
If **PLOAD_LO_PATH** is not set, the default directory is **/tmp**.

For more information, see the *Guide to the High-Performance Loader*.

IDS

PLOAD_SHMBASE

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.



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:

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**.

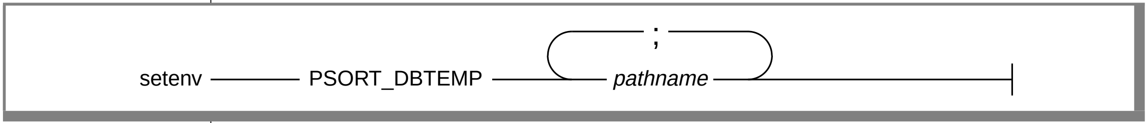
Tip: *Informix recommends that you let the HPL decide where to attach and that you set* ***PLOAD_SHMBASE*** *only if necessary to avoid shared-memory collisions between* ***onpload*** *and the database server.*

For more information, see the *Guide to the High-Performance Loader*.

PSORT_DBTEMP

The **PSORT_DBTEMP** environment variable specifies a directory or directories where the database server writes the temporary files it uses when it performs a sort.

The database server uses the directory that **PSORT_DBTEMP** specifies even if the environment variable **PSORT_NPROCS** is not set.



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 on page 3-51.

For additional information about the **PSORT_DBTEMP** environment variable, see your *Administrator's Guide* and your *Performance Guide*.

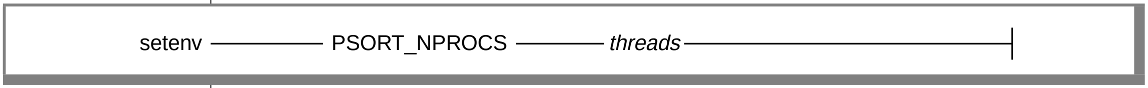
PSORT_NPROCS

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.

XPS

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.



threads	specifies the maximum number of threads to be used to sort a query. The maximum value of threads is 10.

The following command sets the **PSORT_NPROCS** environment variable to 4:

```
setenv PSORT_NPROCS 4
```

To disable parallel sorting, enter the following command:

```
unsetenv PSORT_NPROCS
```

Informix recommends 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 the **PDQPRIORITY** environment variable, check the available memory before you perform a large-scale sort (such as an index build) and make sure that you have enough memory.*

Default Values for Ordinary Sorts

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.

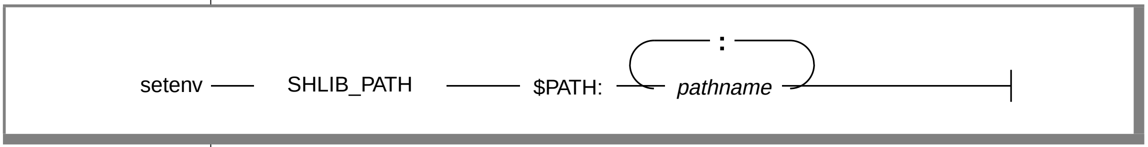
If the **PSORT_NPROCS** environment variable is not set, or if it is set to 0, you get two sort threads for each fragment of the index unless you have a single-CPU virtual processor. If you have a single-CPU virtual processor, you get one sort thread for each fragment of the index.

For additional information about the **PSORT_NPROCS** environment variable, see your *Administrator's Guide* and your *Performance Guide*.

UNIX

SHLIB_PATH

The **SHLIB_PATH** environment variable tells the shell on HP-UX 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.



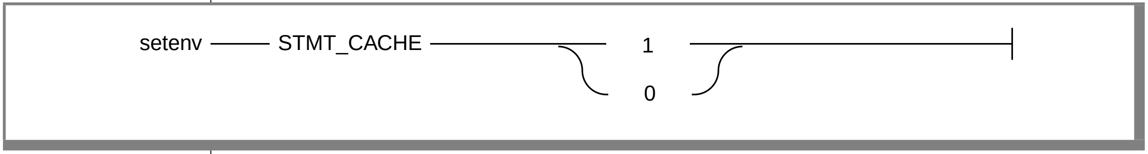
pathname	specifies the search path for the libraries.

On Solaris, set **LD_LIBRARY_PATH**. On AIX, set **LIBPATH**.

IDS

STMT_CACHE

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 speed query processing among different user sessions. Valid **STMT_CACHE** values are 1 and 0.



1	enables the SQL statement cache.
0	disables the SQL statement cache.

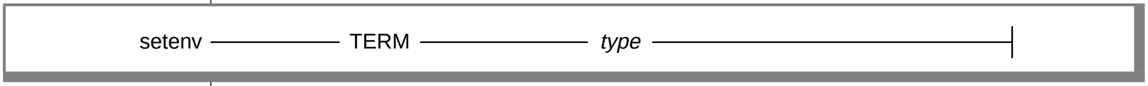
Set the **STMT_CACHE** environment variable for applications that do not use the SET STMT_CACHE statement to control the use of the SQL statement cache.

This environment variable has no effect if the SQL statement cache is disabled through the configuration setting. The SET STMT_CACHE statement in the application overrides the **STMT_CACHE** environment variable setting.

UNIX

TERM

The **TERM** environment variable is used for terminal handling. It lets DB-Access recognize and communicate with the terminal that you are using.



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.

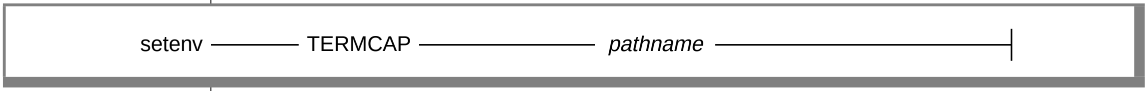
For example, to specify the vt100 terminal, set the **TERM** environment variable by entering the following command:

```
setenv TERM vt100
```

UNIX

TERMCAP

The **TERMCAP** environment variable is used for terminal handling. It tells DB-Access to communicate with the **termcap** file instead of the **terminfo** directory.



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 any of the following settings for **TERMCAP**. Use them in the following order:

1. The **termcap** file that you create
2. The **termcap** file that Informix 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**.

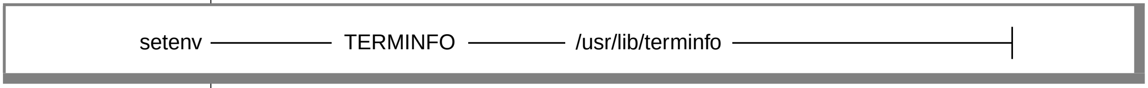
If you do not set the **TERMCAP** environment variable, the system file (that is, **/etc/termcap**) is used by default.

UNIX

TERMINFO

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.



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**.

UNIX

THREADLIB

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 the **THREADLIB** environment variable is not set or if the variable is set to an unsupported thread package.

XPS

XFER_CONFIG

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 on Windows NT.

Index of Environment Variables

Figure 3-2 provides an overview of the uses for the various Informix and UNIX environment variables that Version 8.3 and Version 9.2 support. It serves as an index to general topics and lists the related environment variables and the pages where the environment variables are introduced.

GLS

The term *GLS guide* in the Page column in Figure 3-2 indicates that the GLS environment variables are described in the *Informix Guide to GLS Functionality*. ♦

Figure 3-2
Uses for Environment Variables

Topic	Environment Variable	Page
ANSI compliance	**DBANSIWARN**	3-31
archecker utility	**AC_CONFIG**	3-24
Buffer: fetch size	**FET_BUF_SIZE**	3-60
Buffer: network size	**IFX_NETBUF_SIZE**	3-64
Buffer: network pool size	**IFX_NETBUF_PVTPOOL_SIZE**	3-64
BYTE or TEXT data buffer	**DBBLOBBUF**	3-31
C compiler	**INFORMIXC**	3-65
C compiler: processing of multibyte characters	**CC8BITLEVEL**	GLS guide
C++	**INFORMIXCPPMAP**	3-69
Cache: enabling	**STMT_CACHE**	3-96
Cache: size for Optical Subsystem	**INFORMIXOPCACHE**	3-71
Client locale	**CLIENT_LOCALE**	GLS guide
Client / server	**INFORMIXSERVER**	3-71
	INFORMIXSHMBASE	3-73

(1 of 12)

Topic	Environment Variable	Page
	INFORMIXSTACKSIZE	3-75
Client / server: GLS	**CLIENT_LOCALE**	GLS guide
	DB_LOCALE	GLS guide
	SERVER_LOCALE	GLS guide
Code-set conversion	**CLIENT_LOCALE**	GLS guide
	DB_LOCALE	GLS guide
Communication Support Module: DCE-GSS	**INFORMIXKEYTAB**	3-70
Communications Support Module: location of **concsm.cfg** file	**INFORMIXCONCSMCFG**	3-66
Compilation: ESQL / C	**THREADLIB**	3-99
Compiler	**CC8BITLEVEL**	GLS guide
	INFORMIXC	3-65
Configuration file: database server	**ONCONFIG**	3-83
Configuration file: ignore variables	**ENVIGNORE**	3-59
Configuration file: ON-Archive	**ARC_CONFIG**	3-25
	ARC_DEFAULT	3-25
Configuration file: **tctermcap**	**ARC_KEYPAD**	3-25
Configuration file: **xfer_config**	**XFER_CONFIG**	3-99
Configuration parameter: COSERVER	**INFORMIXSERVER**	3-71
Configuration parameter: DBSERVERNAME	**INFORMIXSERVER**	3-71
Configuration parameter: DBSPACETEMP	**DBSPACETEMP**	3-51
Configuration parameter: DIRECTIVES	**IFX_DIRECTIVES**	3-62

(2 of 12)

Topic	Environment Variable	Page
Configuration parameter: OPCACHEMAX	**INFORMIXOPCACHE**	3-71
Configuration parameter: OPTCOMPIND	**OPTCOMPIND**	3-84
Configuration parameter: OPT_GOAL	**OPT_GOAL**	3-87
Configuration parameter: PDQPRIORITY	**PDQPRIORITY**	3-89
Configuration parameter: STACKSIZE	**INFORMIXSTACKSIZE**	3-75
Connecting	**INFORMIXCONRETRY**	3-67
	INFORMIXCONTIME	3-67
	INFORMIXSERVER	3-71
	INFORMIXSQLHOSTS	3-74
Data distributions	**DBUPSPACE**	3-57
Database locale	**DB_LOCALE**	GLS guide
Database server	**INFORMIXSERVER**	3-71
	SERVER_LOCALE	GLS guide
Database server: archiving	**ARC_CONFIG**	3-25
	ARC_DEFAULT	3-25
	ARC_KEYPAD	3-26
	DBREMOTECMD	3-50
Database server: configuration file	**ONCONFIG**	3-83
Database server: parallel sorting	**PSORT_DBTEMP**	3-93
	PSORT_NPROCS	3-94
Database server: parallelism	**PDQPRIORITY**	3-89
Database server: role separation	**INF_ROLE_SEP**	3-77

(3 of 12)

Topic	Environment Variable	Page
Database server: shared memory	**INFORMIXSHMBASE**	3-73
Database server: stacksize	**INFORMIXSTACKSIZE**	3-75
Database server: tape management	**ARC_CONFIG**	3-25
	ARC_DEFAULT	3-25
	ARC_KEYPAD	3-26
	DBREMOTECMD	3-50
Database server: temporary tables, sort files	**DBSPACETEMP**	3-51
Date and time values	**DBCENTURY**	3-33
	DBDATE	3-37; GLS guide
	DBTIME	3-54; GLS guide
	GL_DATE	GLS guide
	GL_DATETIME	GLS guide
DB-Access utility	**DBACCNOIGN**	3-29
	DBANSIWARN	3-31
	DBDELIMITER	3-40
	DBEDIT	3-41
	DBFLTMASK	3-42
	DBPATH	3-46
	FET_BUF_SIZE	3-60
	INFORMIXSERVER	3-71
	INFORMIXTERM	3-76
	TERM	3-97

(4 of 12)

Topic	Environment Variable	Page
	TERMCAP	3-97
	TERMINFO	3-98
DB/Cockpit graphic tool	**COCKPITSERVICE**	3-27
dbexport utility	**DBDELIMITER**	3-40
Delimited identifiers	**DELIMIDENT**	3-58
Disk space	**DBUPSPACE**	3-57
Editor	**DBEDIT**	3-41
ESQL/C: ANSI compliance	**DBANSIWARN**	3-31
ESQL/C: C compiler	**INFORMIXC**	3-65
ESQL/C: DATETIME formatting	**DBTIME**	3-54; GLS guide
ESQL/C: delimited identifiers	**DELIMIDENT**	3-58
ESQL/C: multibyte filter	**ESQLMF**	GLS guide
ESQL/C: multibyte identifiers	**CLIENT_LOCALE**	GLS guide
ESQL/C: multithreaded applications	**THREADLIB**	3-99
ESQL/C: running C preprocessor before the ESQL/C preprocessor	**CPFIRST**	3-28
Executable programs	**PATH**	3-88
Fetch buffer size	**FET_BUF_SIZE**	3-60
Filenames: multibyte	**GLS8BITSYS**	GLS guide
Files: field delimiter	**DBDELIMITER**	3-40
Files: installation	**INFORMIXDIR**	3-69
Files: locale	**CLIENT_LOCALE**	GLS guide
	DB_LOCALE	GLS guide
	SERVER_LOCALE	GLS guide

(5 of 12)

Topic	Environment Variable	Page
Files: map for C++	**INFORMIXCPPMAP**	3-69
Files: message	**DBLANG**	3-43
Files: temporary	**DBSPACETEMP**	3-51
Files: temporary, for Gateways	**DBTEMP**	3-53
Files: temporary sorting	**PSORT_DBTEMP**	3-93
Files: **termcap**, **terminfo**	**INFORMIXTERM**	3-76
	TERM	3-97
	TERMCAP	3-97
	TERMINFO	3-98
Gateways	**DBTEMP**	3-53
High-Performance Loader	**DBONPLOAD**	3-46
	PLCONFIG	3-91
	PLOAD_LO_PATH	3-92
	PLOAD_SHMBASE	3-92
Identifiers: delimited	**DELIMIDENT**	3-58
Identifiers: long	**IFX_LONGID**	3-63
Identifiers: multibyte characters	**CLIENT_LOCALE**	GLS guide
	ESQLMF	GLS guide
Informix Storage Manager	**ISM_COMPRESSION**	3-78
	ISM_DEBUG_FILE	3-78
	ISM_DEBUG_LEVEL	3-79
	ISM_ENCRYPTION	3-80
	ISM_MAXLOGSIZE	3-80
	ISM_MAXLOGVERS	3-81

(6 of 12)

Topic	Environment Variable	Page
Installation	**INFORMIXDIR**	3-69
	PATH	3-88
Language environment	**DBLANG**	3-43; GLS guide
Libraries	**LD_LIBRARY_PATH**	3-81
	LIBPATH	3-82
	SHLIB_PATH	3-95
Locale	**CLIENT_LOCALE**	GLS guide
	DB_LOCALE	GLS guide
	SERVER_LOCALE	GLS guide
Long Identifiers	**IFX_LONGID**	3-63
Map file for C++	**INFORMIXCPPMAP**	3-69
Message chaining	**OPTMSG**	3-85
Message files	**DBLANG**	3-43; GLS guide
Money values	**DBMONEY**	3-45; GLS guide
Multibyte characters	**CLIENT_LOCALE**	GLS guide
	DB_LOCALE	GLS guide
	SERVER_LOCALE	GLS guide
Multibyte filter	**ESQLMF**	GLS guide
Multithreaded applications	**THREADLIB**	3-99
Network	**DBPATH**	3-46

(7 of 12)

Topic	Environment Variable	Page
Nondefault locale	**CLIENT_LOCALE**	GLS guide
	DB_LOCALE	GLS guide
	SERVER_LOCALE	GLS guide
ON-Archive utility	**ARC_DEFAULT**	3-25
	ARC_KEYPAD	3-26
	DBREMOTECMD	3-50
ON-Bar utility	**ISM_COMPRESSION**	3-78
	ISM_DEBUG_LEVEL	3-79
	ISM_ENCRYPTION	3-80
ONCONFIG parameters	See the “Configuration parameter” entries	
onprobe and **oncockpit** programs	**COCKPITSERVICE**	3-27
Optical Subsystem	**INFORMIXOPCACHE**	3-71
Optimization: directives, setting in the query	**IFX_DIRECTIVES**	3-62
Optimization: message transfers	**OPTMSG**	3-85
Optimization: selecting join method	**OPTCOMPIND**	3-84
Optimization: specifying query performance goal	**OPT_GOAL**	3-87
OPTOFC feature	**OPTOFC**	3-86
Parameters	See the “Configuration parameter” entries	
Pathname: for **archecker** configuration file	**AC_CONFIG**	3-24
Pathname: for C COMPILER	**INFORMIXC**	3-65
Pathname: for database files	**DBPATH**	3-46

(8 of 12)

Topic	Environment Variable	Page
Pathname: for executable programs	**PATH**	3-88
Pathname: for HPL smart-large-object handles	**PLOAD_LO_PATH**	3-92
Pathname: for installation	**INFORMIXDIR**	3-69
Pathname: for libraries	**LD_LIBRARY_PATH**	3-81
	LIBPATH	3-82
	SHLIB_PATH	3-95
Pathname: for message files	**DBLANG**	3-43; GLS guide
Pathname: for parallel sorting	**PSORT_DBTEMP**	3-93
Pathname: for remote shell	**DBREMOTECMD**	3-50
Pathname: for **xfer_config** file	**XFER_CONFIG**	3-99
Printing	**DBPRINT**	3-49
Privileges	**NODEFDAC**	3-77
Program: printing	**DBPRINT**	3-49
Query: optimization	**IFX_DIRECTIVES**	3-62
	OPTCOMPIND	3-84
	OPT_GOAL	3-87
Query: prioritization	**PDQPRIORITY**	3-89
Remote shell	**DBREMOTECMD**	3-50
Role separation	**INF_ROLE_SEP**	3-77
Routine: DATETIME formatting	**DBTIME**	3-54; GLS guide
Server	See the “Database server” entries	

(9 of 12)

Topic	Environment Variable	Page
Server locale	**SERVER_LOCALE**	GLS guide
Shared memory	**INFORMIXSHMBASE**	3-73
	PLOAD_SHMBASE	3-92
Shell: remote	**DBREMOTECMD**	3-50
Shell: search path	**PATH**	3-88
Sorting	**PSORT_DBTEMP**	3-93
	PSORT_NPROCS	3-94
	DBSPACETEMP	3-51
SQL statement: caching	**STMT_CACHE**	3-96
SQL statement: CONNECT	**INFORMIXCONTIME**	3-67
	INFORMIXSERVER	3-71
SQL statement: CREATE TEMP TABLE	**DBSPACETEMP**	3-51
SQL statement: DESCRIBE FOR UPDATE	**IFX_UPDDESC**	3-65
SQL statement: editing	**DBEDIT**	3-41
SQL statement: LOAD, UNLOAD	**DBDELIMITER**	3-40
SQL statement: SELECT INTO TEMP	**DBSPACETEMP**	3-51
SQL statement: SET PDQPRIORITY	**PDQPRIORITY**	3-89
SQL statement: SET STMT_CACHE	**STMT_CACHE**	3-96
SQL statement: UNLOAD	**DBBLOBBUF**	3-31
SQL statement: UPDATE STATISTICS	**DBUPSPACE**	3-57
Stacksize	**INFORMIXSTACKSIZE**	3-75
Tables: temporary	**DBSPACETEMP**	3-51
	PSORT_DBTEMP	3-93

(10 of 12)

Topic	Environment Variable	Page
Tables: temporary, for Gateways	**DBTEMP**	3-53
Temporary tables	**DBSPACETEMP**	3-51
	DBTEMP	3-53
	PSORT_DBTEMP	3-93
Terminal handling	**INFORMIXTERM**	3-76
	TERM	3-97
	TERMCAP	3-97
	TERMINFO	3-98
Utilities: **archecker**	**AC_CONFIG**	3-24
Utilities: DB-Access	**DBANSIWARN**	3-31
	DBDELIMITER	3-40
	DBEDIT	3-41
	DBFLTMASK	3-42
	DBPATH	3-46
	FET_BUF_SIZE	3-60
	INFORMIXSERVER	3-71
	INFORMIXTERM	3-76
	TERM	3-97
	TERMCAP	3-97
	TERMINFO	3-98
Utilities: **dbexport**	**DBDELIMITER**	3-40
Utilities: ON-Archive	**ARC_DEFAULT**	3-25
	ARC_KEYPAD	3-26
	DBREMOTECMD	3-50

(11 of 12)

Topic	Environment Variable	Page
Utilities: ON-Bar	**ISM_COMPRESSION**	3-78
	ISM_DEBUG_LEVEL	3-79
	ISM_ENCRYPTION	3-80
Values: date and time	**DBDATE**	3-37; GLS guide
	DBTIME	3-54; GLS guide
Values: money	**DBMONEY**	3-45; GLS guide
Variables: overriding	**ENVIGNORE**	3-59
Year 2000	**DBCENTURY**	3-33

(12 of 12)

The stores_demo Database

The **stores_demo** database contains a set of tables that describe an imaginary business. The examples in the *Informix Guide to SQL: Syntax*, the *Informix Guide to SQL: Tutorial*, and other Informix manuals are based on this demonstration database. The **stores_demo** database 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 or if they allow duplicate values.
- The second section (page A-10) shows a graphic map of the tables in the **stores_demo** database and indicates the relationships among columns.
- The third section (page A-12) describes the primary-foreign key relationships among columns in tables.
- The final section (page A-19) shows 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 *DB-Access User's Manual*. For information on how to design and implement a relational database, see the *Informix Guide to Database Design and Implementation*.

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-3)
- **orders** (page A-4)
- **items** (page A-5)
- **stock** (page A-6)
- **catalog** (page A-7)
- **cust_calls** (page A-8)
- **call_type** (page A-9)
- **manufact** (page A-9)
- **state** (page A-10)

The following sections describe each table. The unique identifier for each table (primary key) is shaded and indicated by a key symbol.

The customer Table

The **customer** table contains information about the retail stores that place orders from the distributor. Figure A-1 shows the columns of the **customer** table.

The **zipcode** column in Figure A-1 is indexed and allows duplicate values.

Figure 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. Figure A-2 shows the columns of the **orders** table.

Figure 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. Figure A-3 shows the columns of the **items** table.

Figure 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. Figure A-4 shows the columns of the **stock** table.

Figure 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: ▪ 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. Figure A-5 shows the columns of the **catalog** table.

Figure 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. Figure A-6 shows the columns of the **cust_calls** table.

Figure 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 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 call_type Table

The call codes associated with customer calls are stored in the **call_type** table. Figure A-7 shows the columns of the **call_type** table.

Figure 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. Figure A-8 shows the columns of the **manufact** table.

Figure 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. Figure A-9 shows the columns of the **state** table.

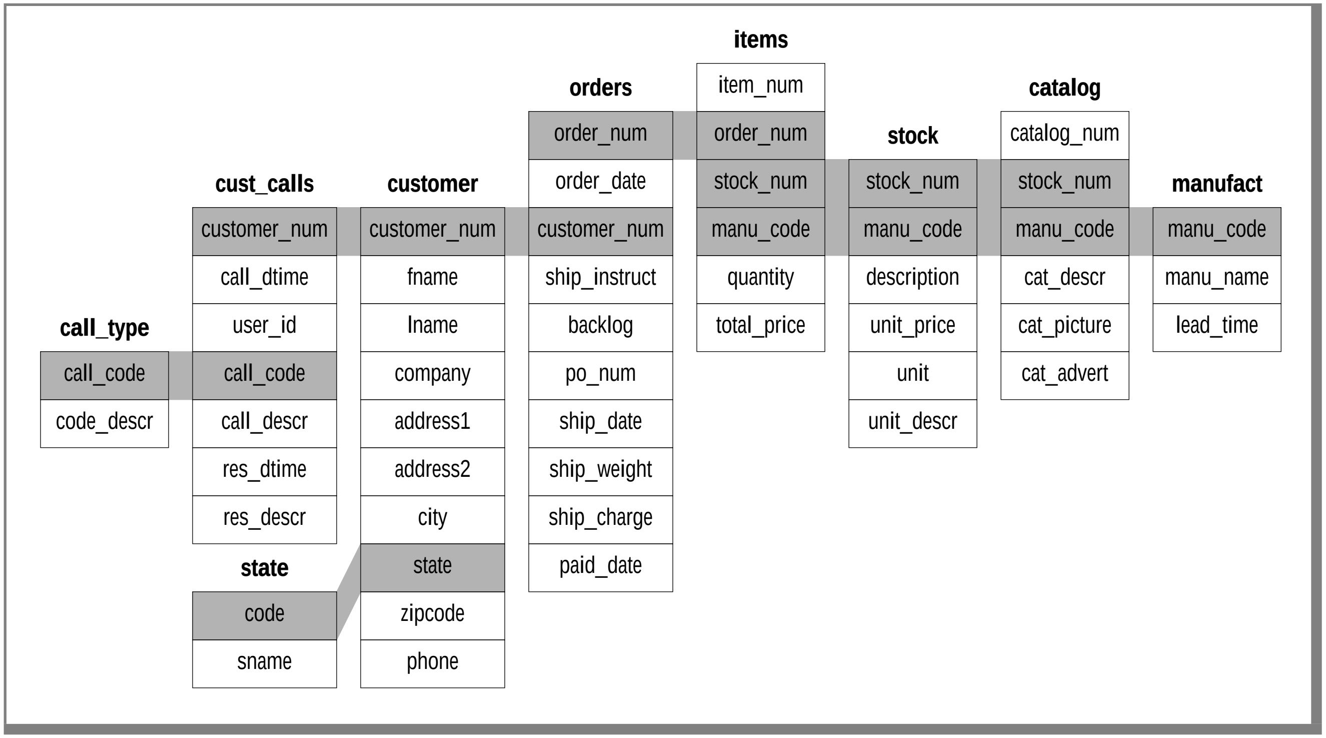
Figure A-9
The state Table

Column Name	Data Type	Description
code	CHAR(2)	state code
sname	CHAR(15)	state name

The stores_demo Database Map

Figure A-10 displays the joins in the **stores_demo** database. The grey shading that connects a column in one table to the same column in another table indicates the relationships, or *joins*, between tables.

Figure A-10
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-10 on page A-11 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-11 through Figure A-18 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-11 shows this relationship.

Figure A-11
Tables That the customer_num Column Joins

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

According to Figure A-11, 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' 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-12 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-12
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-13 shows this relationship.

Figure A-13
Tables That the stock_num and manu_code Columns Join

items Table (detail)

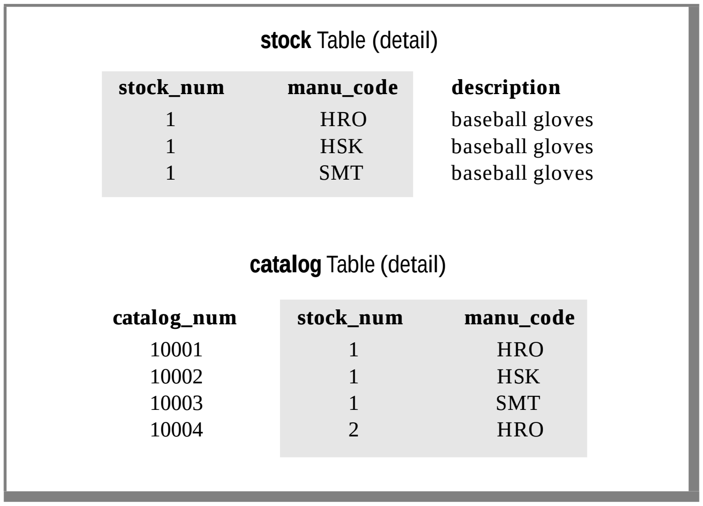
item_num	order_num	stock_num	manu_code
1	1001	1	HRO
1	1002	4	HSK
2	1002	3	HSK
1	1003	9	ANZ
2	1003	8	ANZ
3	1003	5	ANZ
1	1004	1	HRO

stock Table (detail)

stock_num	manu_code	description
1	HRO	baseball gloves
1	HSK	baseball gloves
1	SMT	baseball gloves

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-14 shows this relationship.



stock Table (detail)

stock_num	manu_code	description
1	HRO	baseball gloves
1	HSK	baseball gloves
1	SMT	baseball gloves

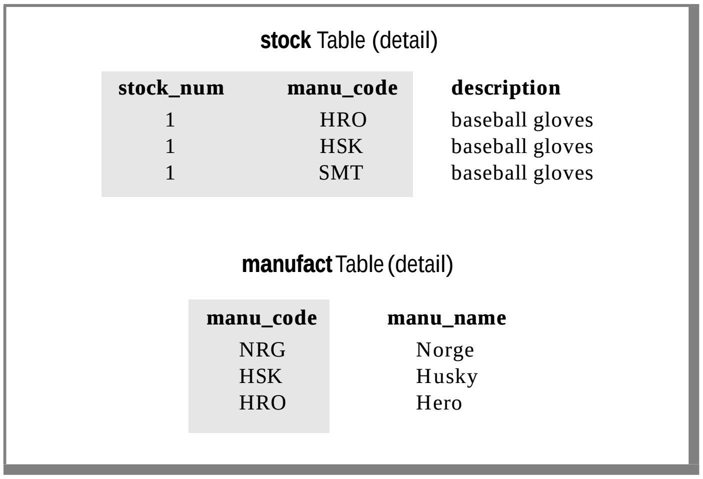
catalog Table (detail)

catalog_num	stock_num	manu_code
10001	1	HRO
10002	1	HSK
10003	1	SMT
10004	2	HRO

Figure A-14
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-15 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-15
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-16 shows this relationship.

Figure A-16
Tables That the customer_num Column Joins

customer Table (detail)

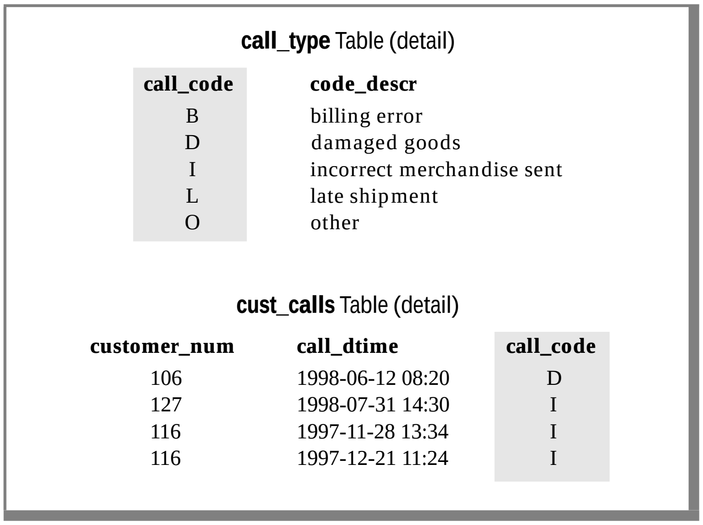
customer_num	**fname**	**lname**
101	Ludwig	Pauli
102	Carole	Sadler
103	Philip	Currie
104	Anthony	Higgins
105	Raymond	Vector
106	George	Watson

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

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** **t**able, the **call_code** column is a foreign key that references the **call_code** column in the **call_type** table. Figure A-17 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-17
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-18 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-18
Tables That the state/code Column Joins

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
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

(1 of 3)

customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone
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
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

(2 of 3)

customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone
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
128	Frank	Lessor	Phoenix University	Athletic Department	1817 N. Thomas Road	Phoenix	AZ	85008	602-533-1817

(3 of 3)

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

(1 of 3)

item_num	order_num	stock_num	manu_code	quantity	total_price
5	1007	7	HRO	1	600.00
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 of 3)

item_num	order_num	stock_num	manu_code	quantity	total_price
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
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

(3 of 3)

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	
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

(1 of 2)

order_num	order_date	customer_num	ship_instruct	backlog	po_num	ship_date	ship_weight	ship_charge	paid_date
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	DM354331	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

(2 of 2)

stock Table

stock_num	manu_code	description	unit_price	unit	unit_descr
1	HRO	baseball gloves	250.00	case	10 gloves / case
1	HSK	baseball gloves	800.00	case	10 gloves / case
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

(1 of 4)

stock_num	manu_code	description	unit_price	unit	unit_descr
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
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

(2 of 4)

stock_num	manu_code	description	unit_price	unit	unit_descr
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
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

(3 of 4)

stock_num	manu_code	description	unit_price	unit	unit_descr
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

(4 of 4)

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
10009	5	NRG	Graphite frame. Synthetic strings.	<BYTE value>	Wide Body Amplifies Your Natural Abilities by Providing More Power Through Aerodynamic Design

(1 of 10)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
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

(2 of 10)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
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
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

(3 of 10)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
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
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

(4 of 10)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
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
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-butted 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

(5 of 10)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
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 & gray or white & ivory. Specify size.	<BYTE value>	Karsten's Top Quality Shoe Combines Leather and Leather Mesh
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

(6 of 10)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
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
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

(7 of 10)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
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
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

(8 of 10)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
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
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

(9 of 10)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
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

(10 of 10)

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.
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.

(1 of 2)

customer_num	call_dtime	user_id	call_code	call_descr	res_dtime	res_descr
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.

(2 of 2)

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	D.C.	NV	Nevada
DE	Delaware	NY	New York
FL	Florida	OH	Ohio

(1 of 2)

code	sname	code	sname
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

(2 of 2)

The sales_demo and superstores_demo Databases

In addition to the **stores_demo** database that is described in detail in Appendix A, Informix products include the following demonstration databases:

XPS

- The **sales_demo** database illustrates a dimensional schema for data-warehousing applications. ♦

IDS

- The **superstores_demo** database illustrates an object-relational schema. ♦

This appendix discusses the contents of these two demonstration databases.

For information on how to create and populate the demonstration databases, including relevant SQL files, see the *DB-Access User's Manual*. For conceptual information about demonstration databases, see the *Informix Guide to Database Design and Implementation*.

XPS

The sales_demo Database

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.

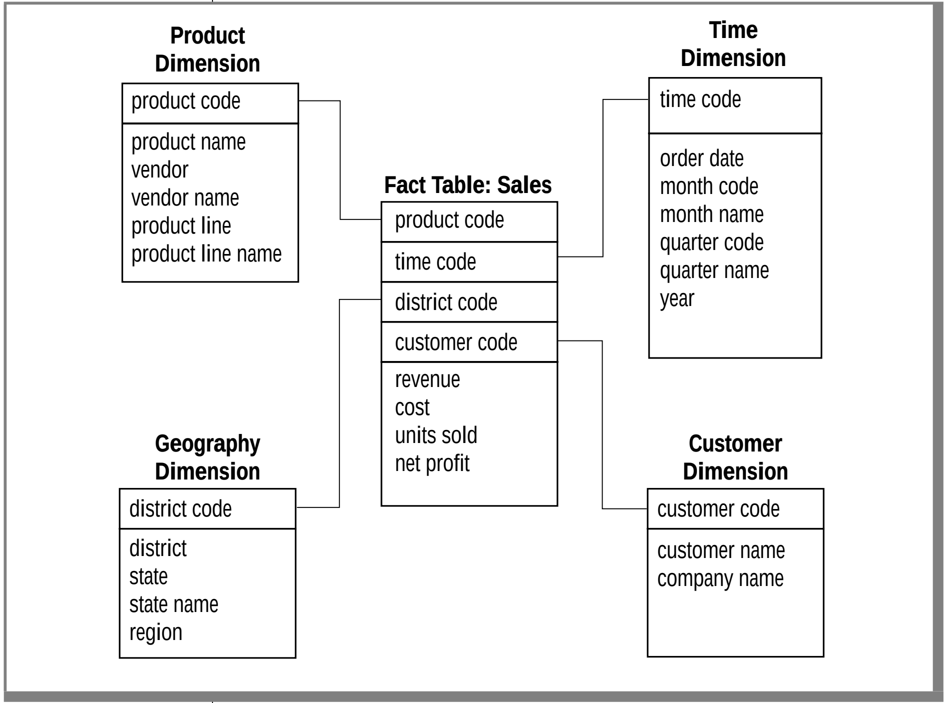


Figure B-1
The sales_demo Dimensional Data Model

For information on how to create and populate the **sales_demo** database, see the *DB-Access User's Manual*. For information on how to design and implement dimensional databases, see the *Informix Guide to Database Design and Implementation*. For information on the **stores_demo** database, see Appendix A.

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 ANSI compliant.

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. However, the primary and foreign key relationships that exist between the fact (**sales**) table and its dimension tables are not defined 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. Figure B-2 shows the columns of the **customer** table.

Figure B-2
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. Figure B-3 shows the columns of the **geography** table.

Figure B-3
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. Figure B-4 shows the columns of the **product** table.

Figure B-4
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. Figure B-5 shows the columns of the **sales** table.

Figure B-5
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. Figure B-6 shows the columns of the **time** table.

Figure B-6
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

IDS

The superstores_demo Database

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 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 *DB-Access User's Manual*. For information on how to work with object-relational databases, see the *Informix Guide to Database Design and Implementation*. 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 column names, column data types, and column descriptions for each table in the **superstores_demo** database. The unique identifier for each table (primary key) is shaded and indicated by a key symbol. Columns that represent extended data types are discussed in “User-Defined Routines and Extended Data Types” on page B-24. Primary-foreign key relationships between the tables are outlined in “Referential Relationships” on page B-27.

The call_type Table

The call codes associated with customer calls are stored in the **call_type** table. Figure B-7 shows the columns of the **call_type** table.

Figure B-7
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. Figure B-8 shows the columns of the **catalog** table.

Figure B-8
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. Figure B-9 shows the columns of the **cust_calls** table.

Figure B-9
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-25 on page B-27 shows.

For information about table hierarchies, see the *Informix Guide to Database Design and Implementation*.

The customer Table

The **customer** table contains information about the retail stores that place orders from the distributor. Figure B-10 shows the columns of the **customer** table.

Figure B-10
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. Figure B-11 shows the columns of the **retail_customer** table.

Figure B-11
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. Figure B-12 shows the columns of the **whlsale_customer** table.

Figure B-12
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. Figure B-13 shows the columns of the **items** table.

Figure B-13
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 on page B-27. For information about table hierarchies, see the *Informix Guide to Database Design and Implementation*.

The location Table

The **location** table contains general information about the locations (addresses) that the database tracks. Figure B-14 shows the columns of the **location** table.

Figure B-14
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. Figure B-15 shows the columns of the **location_non_us** table.

Figure B-15
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. Figure B-16 shows the columns of the **location_us** table.

Figure B-16
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. Figure B-17 shows the columns of the **manufact** table.

Figure B-17
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)	distributer 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. Figure B-18 shows the columns of the **orders** table.

Figure B-18
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 backlogged
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. Figure B-19 shows the columns of the **region** table.

Figure B-19
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. Figure B-20 shows the columns of the **sales_rep** table.

Figure B-20
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 for the 50 states of the United States as well as sales tax information. Figure B-21 shows the columns of the **state** table.

Figure B-21
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. Figure B-22 shows the columns of the **stock** table.

Figure B-22
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)	distributer 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 unit in bigger_unit

The stock_discount Table

The **stock_discount** table contains information about stock discounts. (There is no primary key). Figure B-23 shows the columns of the **stock_discount** table.

Figure B-23
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. Figure B-24 shows the columns of the **units** table.

Figure B-24
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 data type system that supports the following types of activities:

- Extension of existing data types by redefining some of the behavior for the data types that the database server provides.
- Definition of custom 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 *Extending Informix Dynamic Server 2000*.

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-25 shows how the hierarchical tables of the **superstores_demo** database are related.

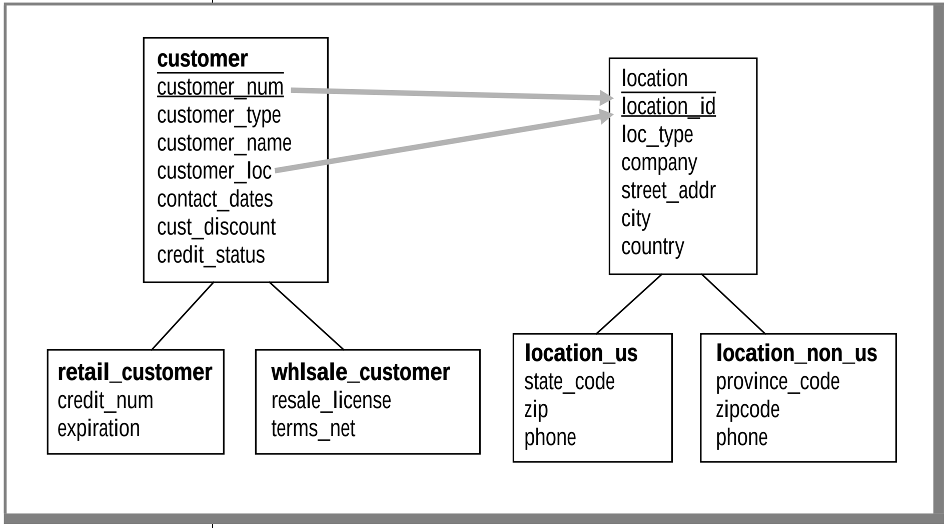


Figure B-25
Hierarchies of superstores_demo Tables

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 t**able, 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.

Glossary

Tip: *Other manuals in the Informix documentation sets, such as the "Informix Storage Manager Administrator's Guide," and the "Informix SNMP Subagent Guide" include a glossary of specialized terms. For additional product-specific information, refer to these glossaries.*

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 if the software that interprets them is 8-bit clean. *See also* non-ASCII character.
8-bit clean	An operating system or database server that can process character data that contains 8-bit characters. The operating system or database server reads the eighth bit as part of the code value. In other words, it does not ignore the eighth bit or make its own interpretation of the eighth bit.
16-bit code set	A code set (such as JIS X0208) in which approximately 65,000 distinct characters can be encoded.
access method	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 type 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, table, table fragment, or column. Informix maintains its own set of database, table, table fragment, and column access privileges, which are independent of the operating-system access privileges.

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 queried rows; for example, the total number, sum, average, and 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	A temporary alternative name for a table in a query; usually used in complex subqueries and required for self-joins. In a form-specification file or any SQL query, alias refers to a single-word alternative name used in place of a more complex table name (for example, *t1* as an alias for *owner.table_name*).
ALS	Legacy acronym for the *Asian Language Support* feature for working with Asian (multibyte) data. Supplanted by *Global Language Support* (*GLS*).
ANSI	Acronym for the American National Standards Institute. This group sets standards in many areas, including the computer industry and standards for SQL languages.
ANSI compliant	A database that conforms to certain ANSI standards. Informix databases can be created either as ANSI compliant or not ANSI compliant. An ANSI-compliant database enforces certain ANSI requirements, such as implicit transactions, required owner naming, and unbuffered logging (unbuffered logging only when using Dynamic Server), that are not enforced in databases that are not ANSI compliant.
API	*See* application programming interface (API).
application development tool	Software, such as INFORMIX-NewEra, that 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	Tools, such as forms and reports, used to write applications.
application program	An *executable file* or logically related set of files.

application programming interface (API)	A group of related software components, usually provided by a third party such as Informix, 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 user interfaces, and command-line programs. *See also* SQL API and DataBlade API.
arbitrary rule	A series of expressions that you define for *expression-based fragmentation*, using SQL relational and logical operators. Unlike the range rule, the arbitrary rule allows you to use any relational operator and any logical operator to define the expressions. Typically includes the use of the OR logical operator to group data.
archiving	Copying all the data and indexes of a database onto a new medium, usually a tape or a different physical device from the one that stores the database. Archived material is used for recovering from a failure and is usually performed by a database administrator. *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 type. Individual members of the array are referred to as *elements* and usually are distinguished by an integer argument that gives the position of the element in the array. Informix arrays can have up to three dimensions.
ASCII	Acronym for the American Standards Committee for Information Interchange. ASCII is a coding scheme that assigns numeric values to letters, numbers, punctuation marks, and certain other characters. ASCII describes an ordered set of printable and nonprintable characters used in computers and telecommunication. It contains 128 characters, each of which can be represented with 7 bits of information. *See also* single-byte character.
ASF	Acronym for Associated Services Facility. The code in the ASF portion of Informix products controls the connections between clients and servers. System developers use this term; users of Informix products see this term only in occasional error messages.
Asian Language Support (ALS)	A class of products that operate with multibyte code sets. ALS products support various multibyte code sets whose characters are composed of 8, 16, 24, and 32 bits. ALS servers and tools are available for Version 6.x and earlier family of products. These products might have been developed with the GLS Library or other software written specifically to handle Asian language processing. For more information, see the *Informix Migration Guide*.

attached index An index that is created without an explicit fragmentation strategy. You create an attached index by omitting both the distribution scheme (specified by the FRAGMENT BY clause) and the storage option (specified by the IN clause) of the CREATE INDEX or ALTER FRAGMENT ON INDEX statements. An attached index can be created on a fragmented table.

The location of the index data varies depending on the database server. In most cases, index pages reside in the same tblspaces as the data pages to which they refer.

For Dynamic Server, index pages for user indexes reside in separate tblspaces, but within the same dbspaces, as the data pages to which they refer. Only the **syscatalogs** indexes reside in the same tblspace as the corresponding data pages.

For Extended Parallel Server, both user and system-catalog index pages reside in separate tblspaces but in the same dbspaces as the corresponding data pages. *See also* detached index.

audit event (Not for Extended Parallel Server) Any database server activity or operation that could potentially access and alter data, which should 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 *Trusted Facility Manual*.)

audit file (Not for Extended Parallel Server) A file that contains records of audit events and resides in the specified audit directory. Audit files form an audit trail of information that can be extracted by the database secure auditing facility for analysis by the database administrator.

audit mask (Not for Extended Parallel Server) A structure that specifies which events should be audited by (or excluded from auditing by) the database secure auditing facility.

auxiliary statements The SQL statements that you use to obtain auxiliary information about tables and databases. These statements include INFO, OUTPUT, WHENEVER, and GET DIAGNOSTICS.

B+ tree A method of organizing an index into a tree structure for efficient record retrieval.

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	*See* table.
base type	*See* opaque data type.
before-image	The image of a row, page, or other item before any changes are made to it.
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 binary large object that is now known as 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. It can be stored and retrieved in pieces and has 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 *leaf nodes* and other branch nodes. The database server creates branch nodes 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 the operating system controls instead of 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 fundamental *data type* that the database server defines, for example, INTEGER, CHAR, or SERIAL.
byte	The smallest physical computer storage unit. A byte is not necessarily one character. In multibyte code sets, a character is 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	A feature that causes 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, eliminate identical information from any foreign-key column in a related table.
case-sensitivity	The condition of distinguishing between uppercase and lowercase letters. Be careful running Informix programs because certain commands and their options are case-sensitive; that is, they react differently to the same letters presented in uppercase and lowercase characters.
cast	A mechanism that converts one data type to another. *See also* built-in cast, 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 (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, and diacritical marks. A natural language is a language that people 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 condition that must be met before data can be assigned to a table column during an INSERT or UPDATE statement.
checkpoint	A point in time 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	(No blobspace chunks for Extended Parallel Server) 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.

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* locale and 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.
client/server connection statements	The SQL statements that you use to 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. It can be stored and retrieved in pieces and has database properties such as recovery and transaction rollback. *See also* BLOB.
close a cursor	To drop the association between a cursor and active set of rows that results from a query.
close a database	To deactivate the connection between a client and a database. Only one database can be active at a time.
close a file	To release 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 *A* character has a code point of 0x00.

code set	A computer representation of a character set that specifies how to map each character in a character set to a unique code point. For example, ASCII, ISO8859-1, Microsoft 1252, and EBCDIC are code sets that can represent the English language. A locale name specifies a code set. *See also* code point and character 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 order of characters within a code set based on their code points. 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	The sequence of values that specifies some logical order in which the character fields in a database are sorted and indexed. A collation sequence depends on either the order of the code set or the locale. Collating sequence is also known as *collation order.*
collation	The process of sorting characters based on a collation order.
collation order	*See* collating sequence and code-set order.
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 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 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 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 particular type of information that occurs in every row of the table. Also known as 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 in a column expression. *See also* subscript.
column substring	A substring in a column expression. *See also* substring.
command file	A system file that contains one or more statements or commands, such as SQL statements.
comment	The information in a program file that is not processed by the computer but documents the program. You use special characters such as a pound sign (#), curly braces ({ }), slash marks (/) and asterisks (*), or a double dash (--) to identify comments, depending on the programming environment.
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 a user can view only rows that are currently committed at the moment of the query request; that is, a 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* isolation and Read Committed.
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* join and 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	A symbolic notation composed of two pipe symbols (| |) used in expressions to indicate the joining of two strings.
concurrency	The ability of two or more processes to access the same database simultaneously.
configuration management (CM) coserver	A coserver that Informix designates 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 that is enabled by an option-field setting in the sqlhosts file whereby the database server attempts to establish a client connection with each coserver in a dbserver group until the connection is successful.
constant	Data whose value does not change during the execution of a program or command. In a program, 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 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 on-line Help in Informix products). A control character is sometimes referred to as a *control key*. *Compare with* printable character.
cooked files	*See* buffered disk I/O.
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 that you can use 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 name must belong to the triggering table.
corrupted database	A database whose tables or indexes contain incomplete or invalid data.
corrupted index	An index that does not correspond exactly to 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.
cursor	An identifier associated with a group of rows or 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 row information. In addition, you can declare a select cursor for update (initiating locking control for updated and deleted rows) or WITH HOLD (completing a transaction does not close the cursor). In ESQL/C, a cursor can be dynamic, meaning that it can be an identifier or a character/string variable.
cursor function	A *user-defined function* that returns one or more rows of data and therefore requires a *cursor* to execute. An *SPL function* 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 SQL statements that you use to grant and revoke permissions and to lock tables.
data definition statements	The SQL statements that you use to create, alter, drop, and rename data objects, including databases, tables, views, synonyms, triggers, and SPL routines.
data dictionary	The collection of tables that keeps track of the structure of the database. Information about the database is maintained in the data dictionary, which is also referred to as the system catalog. *See also* system catalog.

data distribution	A mapping of the data in a column into a set of the column values. The contents of the column are examined and divided into bins, each of which represents a percentage of the data. The organization of column values into bins is called the distribution for that column. Use the UPDATE STATISTICS statement 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 and transaction processing are used to control data integrity.
data integrity statements	The 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 you use to query tables, insert into tables, delete from tables, update tables, and load into and unload from tables.
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 *Informix Guide to GLS Functionality*. *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 useful to a particular organization or 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 a database server uses to interpret NCHAR and NVARCHAR data that is in a particular database. The **DB_LOCALE** environment variable can specify a nondefault database locale. *See also* 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, cursor, index, prepared statement, synonym, trigger, 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* that Informix provides to allow a C *user-defined routine* access to the database server. You can also use the DataBlade API to create client LIBMI applications (for backward compatibility with Illustra applications).
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 is responsible for managing one or more database servers.
DBA-privileged	A class of SPL routines that only a user with DBA database privileges creates.
DBMS	Acronym for *database management system*. It is 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 (such as chunks) units of storage. *See also* root dbspace.
DDL	Acronym for data definition language. *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 whose 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 operations such as 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	How a program acts or the values that are assumed if the user does not explicitly specify an action.
default locale	The locale that a product uses unless you specify a different (nondefault) locale. For Informix products, U.S. English is the default locale. *See also* locale.
default value	A value inserted in a *column* or *variable* when an explicit value is not specified. 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	An identifier surrounded by double quotes. The purpose of a delimited identifier is to allow usage of identifiers that are otherwise identical to SQL reserved keywords or that contain nonalphabetical characters. *See also* identifier.
delimiter	A boundary on an input field or the terminator for a column or row. Some files and prepared objects require semicolon (;), comma (,), space, or tab delimiters between statements.
deluxe mode	A method of loading or unloading data that uses regular inserts.
descriptor	A quoted string or embedded variable name that identifies an allocated system-descriptor area or an **sqlda** structure. It is used for the Informix SQL APIs. *See also* identifier.
detached index	The type of index you get when the distribution scheme (specified by the FRAGMENT BY clause) and the storage option (specified by the IN clause) of the CREATE INDEX or ALTER FRAGMENT ON INDEX statements differ from the distribution scheme of the underlying table. Index pages reside in separate dbspaces from the corresponding data pages. *See also* attached index.
device array	A list of I/O devices. *See also* component.
diagnostic area	A data structure 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 and diagnostics tables and associate them with a base table.
Dirty Read	An Informix isolation level set with the SET ISOLATION statement that does not account for 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 create 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 has a different name. 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.
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 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 or EXECUTE FUNCTION statement.

Dynamic Server instance	The set of processes, storage spaces, and shared memory that together comprise a complete database server.
dynamic SQL	The statements and structures that allow a program to form an SQL statement during execution, so that portions of the statement can be determined by user input.
dynamic statements	The SQL statements that are created at the time the program is executed rather than when the program is written. You use the PREPARE statement to create dynamic statements.
EBCDIC	Acronym for Extended Binary Coded Decimal Interchange Code. An 8-bit, 256-element character set.
element	A member of a *collection*. An element can be a single 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 a host language. 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 when the format is in literal strings or character variables. End-user formats are useful for data types that have a database format that is different from the format to which users are accustomed.
end-user routine	A *user-defined routine* 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 made 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) designated number. 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 following character, normally interpreted by the program, is to be printed as a literal character instead. The escape character is used with the interpreted character to "escape" or ignore the interpreted meaning.
escape key	The keyboard key, 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 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 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 select list	A SELECT statement in which the user explicitly specifies the columns that the query returns.
explicit transaction	A transaction that begins with a BEGIN statement and ends with a COMMIT statement. This type of transaction applies only to non-ANSI databases with logging. *See also* implicit ANSI 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	Anything from a simple numeric or alphabetic constant to a more complex series of column values, functions, quoted strings, operators, and keywords. A Boolean expression contains a logical operator (>, <, =, !=, IS NULL, and so on) and evaluates as TRUE, FALSE, or UNKNOWN. An arithmetic expression contains the operators (+, –, ×, / , and so on) and evaluates as 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 type*s, *opaque data type*s, and *distinct data type*s.
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 C and Java. 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. You can specify the name of an external space instead of the name of a dbspace in the IN clause of the CREATE TABLE and CREATE INDEX statements.
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. *See also* application process.
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 for selecting rows or records. For an SQL statement, the conditional expression in the WHERE clause is a filter that controls the set of rows that a query evaluates. This filter is sometimes referred to as a *predicate*. The High-Performance Loader (HPL) uses a filter *component* to screen data before loading it into a database.

filtering mode	The object mode of a database object that causes bad rows to be filtered out to the violations table during data manipulation operations. Only constraints and unique indexes can be in the filtering mode. When a constraint or unique index is in this mode, the database server enforces the constraint or the unique index requirement during INSERT, DELETE, and UPDATE operations but filters out rows that would violate the constraint or unique index requirement.
fixchar	A character data type, available in ESQL/C programs, in which the character string is fixed in length, padded with trailing blanks if necessary, 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 left 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, there must exist a matching entry in the unique or primary column, if all foreign-key columns contain non-null values.
format	A description of the organization of a data file. *See also* component.
formatting character	A percent sign (%) followed by a letter (c, n, o, or r). When used 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. *See also* table fragmentation.
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 data communications device that establishes communications between networks.
generalized-key (GK) index	A type of index for static tables with Extended Parallel Server that can speed certain queries by letting you store the result of an expression as a key in a B-tree or bitmap index. The three categories of GK index are *selective*, *virtual column*, and *join*.
gigabyte	Gigabyte is a unit of storage. A gigabyte equals 1024 megabytes or 1024^3 bytes.
Global Language Support (GLS)	An application environment that lets Informix APIs and database servers handle different languages, cultural conventions, and code sets. For information about the GLS feature, see the *Informix Guide to GLS Functionality*.
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 Informix GLS. An *API* of C routines that a C-language *external routine* can use to access Informix GLS *locales*. This API also includes functions that obtain culture-specific collation order, time and date formats, numeric formats, and functions that provide a uniform way of accessing character data, regardless of whether the locale 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	On-line 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 that governs a group of database servers in which at least one participant 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 that marks your place on the screen. A highlight often indicates the current option on a menu or the current character in an editing session. If a terminal cannot display highlighting, the current option often appears in angle brackets, and the current character is 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	A sequence of letters, digits, and underscores (_) that represents the unqualified name of a database or program object.
implicit ANSI transaction	A transaction that begins implicitly after a COMMIT statement and ends with the next COMMIT statement. This type of transaction applies only to ANSI databases. *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 the data conversion. *See also* explicit cast.
implicit connection	A connection made using a database statement (DATABASE, CREATE DATABASE, START DATABASE, DROP DATABASE). *See also* explicit connection.
implicit select list	A SELECT statement that uses the asterisk (*) symbol so that a query returns all columns of the table.
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) 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 or user name. The user ID does not need to 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. Any given user typically has the same Informix user ID on all networked computer systems involved in the database access.
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 to validate the user ID.
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	*See* simple 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	A placeholder within a prepared SQL statement that indicates a value is to be provided at the time the 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 magnetic medium (tape, cartridge, or floppy disk) or CD onto a computer and preparing it for use.
interactive	Refers to 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.

international-ization (I18n)	The process of making Informix products easily adaptable to any culture and language. Among other features, internationalized software provides support for culturally specific sorting and for adaptable date, time, and money formats. For more information, see the *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 other systems, 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. An indexed sequential access method allows you to find information in a specific order or to find specific pieces 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 characters from European languages, for example, é, ñ and ö.

isolation	The level of independence when multiple users attempt to access common data 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. Set the isolation level with the SET ISOLATION or SET TRANSACTION statement.
iterator function	A *cursor function* that is written in an external language such as C or Java.
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 index* that contains keys that are the result of a query that joins multiple tables.
jukebox	A cabinet that consists of one or more optical-disc drives, slots that store optical platters when they are not mounted, and a robotic arm that transfers platters between the slots and the drives. A jukebox is also known as an *autochanger*.
kernel	The part of the operating system that controls processes and the allocation of resources.
key	The pieces 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 or reverse alphabetical order by their last names (**last_name** serves as the key).
keyword	A word that has meaning to a program. For example, the word SELECT is a keyword in SQL.
kilobyte	A unit of storage that equals 1024 bytes.

Language Supplement	An Informix product that provides the locale files and error messages to support one or more languages. The International Language Supplement supports several European languages. Informix provides separate Language Supplements for several Asian languages.
large object	A data object that is logically stored in a table column 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	An index page that contains index items and horizontal pointers to other leaf nodes. The 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 particular kind of application. An *application programming interface* can include a library of routines that you can call from your *application program*. *See also* dynamic link library (DLL), shared library, and shared-object file.
light append	An unbuffered, unlogged insert operation.
link	To combine 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 and database server that uses a network connection even though the client 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 rules for a country, region, culture, or language. Informix products provide pre-defined 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 The order of characters as specified within a particular locale. Localized order can also specify a dictionary or phone-book order. For example, in dictionary order, uppercase characters and lowercase characters are treated the same; one does not take precedence over the other. *See also* collation 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 method 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 when an R-tree index is updated 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 does not need to be changed.

lock mode An option that describes whether a user who requests a lock on an already locked object is to not wait for the lock and instead receive an error, wait until the object is released to receive the lock, or wait a certain amount of time before receiving an error.

locking	The process of temporarily limiting access to an object (database, table, page, or row) to prevent conflicting interactions among concurrent processes. Locks can be in either exclusive mode, which restricts read and write access to only one user or share mode, which allows read-only access to other users. In addition, update locks exist that begin in share mode but are upgraded to exclusive mode when 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 the log was active. The logical log 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 1024^2 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 is 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 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. Applications that handle data in a multibyte code set 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 that are run within the same process. *See* thread.
named row type	A *row type* created with the CREATE ROW TYPE statement that has a defined name and *inheritance* properties and can be used to construct a *typed table*. A named row type is not equivalent to another named row type, even if its field definitions are the same.
national character	A character in a native language character set. Also known as *native character*.
native character	*See* national character.

Native Language Support (NLS)
A class of products that operate with single-byte code sets. An NLS product uses locales and code sets that the operating system supplies. NLS servers and tools are available for the Version 6.x and later family of products. For more information, see the *Informix Migration Guide*.

NLS
Legacy acronym for the *Native Language Support* feature for working with single-byte, non-English data. Supplanted by *Global Language Support* (*GLS*).

node
Within 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 that are arranged in a hierarchy. *See also* branch node, leaf node, and root node.

Within the context of a MPP system, a node is an individual computer. *See also* massively parallel processing system.

Within 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.

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 with a code point greater than 127. 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 the column cannot contain null values.

null value
A value that is unknown or not applicable. (A null 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 the enabled, disabled, or filtering mode. A trigger or duplicate index can be in the enabled or disabled mode. You use the SET statement to change the object 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.
ON-Monitor	(Not for Extended Parallel Server) An interface that presents a series of screens through which a database server administrator can monitor and modify a database server.
opaque data type	A fundamental *data type* that you define, which contains one or more values encapsulated with an internal length and input and output functions that convert text to and from an internal storage format. Opaque types 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 cursor, 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 symbol (such as =, >, <,+, -, and *) 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 platters	The removable optical discs that store 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 a subservient table in a query where the values for the outer part of the join are preserved even though no matching rows exist in the subservient table. Any dominant-table rows that do not have a matching row in the subservient table contain 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 Read Uncommitted. No other isolation level allows the 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 blobspace 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 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 non-null unique 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 from the columns of a single table. Projection is implemented through the select list in the SELECT clause of a SELECT statement and returns some of the columns and all the rows in a table. *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 rules that govern communication among computers. These rules govern format, timing, sequencing, and error control.
query	A request to the database to retrieve data that meets certain criteria, usually made with the SELECT statement. When used with the High-Performance Loader (HPL), selects records to unload from a relational database. *See also* component.
query optimization information statements	The SQL statements that are used to optimize 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 method 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	An ANSI level of isolation available through Dynamic Server and set with the SET TRANSACTION statement in which a user can view only rows that are currently committed at the moment of the query request. That is, a user cannot view rows that were changed as a part of a currently uncommitted transaction. It is the default level of isolation for databases that are not ANSI compliant. *See also* isolation and Committed Read.
Read Uncommitted	An ANSI level of isolation set with the SET TRANSACTION statement that does not account for locks and allows viewing of any existing rows, even rows that currently can be altered from inside an uncommitted transaction. Read Uncommitted is the lowest level of isolation (no isolation at all), and is thus the most efficient. *See also* isolation and 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 that is 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. *See also* home page.
remote	A connection that requires a network.
Repeatable Read	An Informix and ANSI level of isolation available with the Informix SET ISOLATION statement or the ANSI SET TRANSACTION statement, which ensures that all data read during a transaction is 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 i*nstance* 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. *See also* column.
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 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. Informix recommends 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.
sbspace	(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* in which a *variable* or *identifier* can be accessed. Three possible scopes exist: local (applies only in a single program block), modular (applies throughout a single module), and global (applies throughout the entire program). *See also* local variable and global variable.

scratch table	A nonlogging temporary table.
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	(Not for Extended Parallel Server) A facility of Informix database servers that lets a database server administrator keep track of unusual or potentially harmful user activity. Use the **onaudit** utility to enable auditing of events and create audit masks, and the **onshowaudit** utility to extract the audit event information for analysis.
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	Taking a horizontal subset of the rows of a single table that satisfies a particular 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.
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 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 particular 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 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 wants to interrupt or suspend the execution of a process. *See also* interrupt.
signature	*See* routine signature.
simple 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. Also known as an inner join. *See also* composite join.
simple large object	A *large object* that is stored in a *blobspace* 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 blobspace.
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. When an application handles data in these code sets, it 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 occurs in a non-ANSI logging database. *See also* explicit transaction and implicit ANSI 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.
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, and other information are registered in the system catalog tables of a database. The database server also stores the executable format in system catalog tables. An SPL routine can be an *SPL procedure* or an *SPL function*.
SPL variable	A *variable* that is created with the DEFINE statement and used 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 Structured Query Language (SQL) statements directly in an application. The embedded-language product 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 structure contains an **sqlvar_struct** structure for each column; each **sqlvar_struct** structure provides information such as the name, data type, and length of the column. The sqlda structure is an Informix-specific structure for handling dynamic columns. It is available only within an 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 set of characters (generally 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 surrounded by single quotes. A string is also a character data type, available in Informix ESQL/C programs, in which the character string is stripped of trailing blanks and is null-terminated.
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 offset into an array. Subscripts can be used to indicate the start or end position in a CHAR 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 and used in place of the original name for that table. A synonym does not replace the original table name; instead, it acts as an alias for the table.
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 *cast* that is built in to the database server. A built-in cast performs automatic conversions between 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-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 SQL SELECT statement.

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.
temp table	A logging temporary table that support indexes, constraints, and rollback.
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 on-line session ends.

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* in the same way that 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.
TLI	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 ANSI 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 mechanism that resides in the database. It specifies that when a particular action (insert, delete, or update) occurs on a particular table, the database server should automatically perform one or more additional actions.
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.
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 database data. Unbuffered I/O is supported by character-special files on UNIX and by both unbuffered files and the raw disk interface on Windows NT.
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.
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	*See* Informix user ID.
user ID password	*See* Informix user password.
user name	*See* Informix user ID.
user password	*See* Informix user password.
user thread	A database server thread that services requests from client applications. 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 picture of the contents in a database that allows a programmer to determine what information the user sees and manipulates. A view represents a virtual table based on 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 whose data you create to access data in an external file, external DBMS, or smart large object. 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(s).
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 that represents any sequence of zero or more characters or any single character. In SQL, for example, you can use the asterisk (*), question mark (?), brackets ([]), percent sign (%), and underscore (_) as wildcard characters. (The asterisk, question mark, and brackets are also wildcards in UNIX.)
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 the 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	A set of specifications that vendors and users can use to build portable software. Any vendor that carries the XPG brand on any particular 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	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.

Index

A

Access method
  sysams information about 1-17
  sysindices information about 1-52
  sysopclasses information about 1-56
  systabamdata information about 1-71
AC_CONFIG environment variable 3-24
ac_config.std file 3-24
ALTER TABLE statement
  and SERIAL 2-40
  and SERIAL8 2-41
  casting effects 2-64
  changing data types 2-6
  MODIFY NEXT SIZE clause 1-12
ANSI compliance
  -ansi flag 3-30
  DBANSIWARN environment variable 3-30
  level Intro-15
archecker utility 3-24
Archiving
  setting a different tctermcap file 3-26
  setting DBREMOTECMD to override default remote shell 3-49
  setting personal default qualifier file 3-25
ARC_CONFIG environment variable 3-25
ARC_DEFAULT environment variable 3-25
ARC_KEYPAD environment variable 3-26

B

BLOB data type
  attributes in syscolattribs 1-27
  casting not available 2-10
  coltype code for 1-32
  description of 2-9
  inserting data 2-11
Blobspace
  defined 2-50
  staging, memory cache for 3-70
Boldface type Intro-7
BOOLEAN data type
  coltype code for 1-32
  description of 2-11
Boolean expression
  with BYTE data type 2-12
  with TEXT data type 2-45
Bourne shell, .profile file 3-9
Buffer, setting size of fetch buffer 3-59
Built-in data type
  casting, built-in 2-64, 2-70
  kinds of 2-48
BYTE data type
  casting to BLOB data type 2-13
  description of 2-12
  increasing buffer size 3-31
  inserting data 2-12
  length (syscolumns) 1-34

location shown in sysblobs table 1-24
restrictions
in Boolean expression 2-12
with GROUP BY 2-12
with LIKE or MATCHES 2-12
with ORDER BY 2-12
selecting a BYTE column 2-12
setting buffer size 3-31

C

C compiler, setting INFORMIXC environment variable 3-64
C shell
.cshrc file 3-9
.login file 3-9
call_type table in stores_demo database, columns in A-9
call_type table in superstores_demo database, columns in B-10
Cast 2-63 to 2-70
built-in 2-64 to 2-67, 2-68
distinct data type 2-69
explicit 2-68
from BYTE to BLOB data type 2-13
from TEXT to CLOB data type 2-45
implicit 2-67, 2-68
precedence of 2-68
syscasts information 1-25
CHAR data type
casts for 2-65
collation 2-14
description of 2-13
multibyte values 2-14
nonprintable characters with 2-14
with numeric values 2-13
Character data types
collength (column length) information 1-33
list of 2-48
CHARACTER data type. *See* CHAR data type.
Character string
as DATETIME values 2-20, 2-56
as INTERVAL values 2-29
CHARACTER VARYING data type
description of 2-15
length 1-33
See also VARCHAR data type.
Check constraint
described in syschecks table 1-26
described in syscoldepend table 1-29
Checking contents of environment configuration file 3-12
chkenv utility
description of 3-12
error message for 3-13
Client/server
INFORMIXSQLHOSTS environment variable 3-73
shared memory communication segments 3-72, 3-73
specifying default database 3-70
specifying stacksize for client session 3-74
CLOB data type
attributes in syscolatribs 1-27
casting not available 2-15
code-set conversion of 2-16
collation 2-16
coltype code for 1-32
description of 2-15
inserting data 2-16
multibyte characters with 2-16
COCKPITSERVICE environment variable 3-27
Code set, ISO 8859-1 Intro-4
Code, sample, conventions for Intro-12
Collation
with CHAR data type 2-14
with CLOB data type 2-16
with TEXT data type 2-46
with VARCHAR data type 2-48
Collection data type
casting matrix 2-70
description of 2-60
LIST 2-30
MULTISET 2-33
SET 2-42
sysattrtypes information about elements 1-23
sysxtddesc contents for 1-82
sysxtdtypes information 1-84
Colon
as delimiter in DATETIME 2-19
as delimiter in INTERVAL 2-29
Color, setting INFORMIXTERM for 3-75
Column
changing data type 2-63
constraints, listed in sysconstraints table 1-36
defaults, described in sysdefaults table 1-37
described in syscolumns table 1-30
in stores_demo database A-3 to A-10
inserting data into BLOB 2-11
inserting into BYTE 2-12
inserting values into unnamed row type 2-39
referential constraints in sysreferences table 1-66
storing numeric values in VARCHAR 2-47
value, maximum/minimum 1-35
Column-level privilege, described in syscolauth table 1-28
Command-line conventions
elements of Intro-10
example diagram Intro-11
how to read Intro-11
Comment icons Intro-8
Compiling
environment variable for C compiler 3-64
ESQL/C programs, environment variable to change order of 3-28
multithreaded ESQL/C applications 3-98
Complex data type 2-59 to 2-62
collection types 2-60
row types 2-61
sysattrtypes information about members 1-23
Compliance
with industry standards Intro-15

Configuration file
  for database servers 3-82
  for ON-Archive utility 3-25
  for ON-Bar utility 3-24
  for tctermcap 3-26
config.arc file 3-25
CONNECT statement and INFORMIXSERVER environment variable 3-71
Connection
  authentication, DCE-GSS CSM environment variable 3-69
  INFORMIXCONRETRY environment variable 3-66
  INFORMIXCONTIME environment variable 3-66
Constraint
  check, described in syscoldepend table 1-29
  column, described in sysconstraints table 1-36
  not null, described in syscoldepend table 1-29
  referential, described in sysreferences table 1-66
Contact information Intro-16
Conventions, documentation Intro-6
Converting
  data types 2-63
  DATE and DATETIME 2-66
  INTEGER, DATE, and DATETIME 2-66
  number and char 2-65
  number to number 2-65
CPFIRST environment variable 3-28
CREATE SCHEMA statement, example 1-7
CREATE TABLE statement
  and typed tables 2-37
  assigning data types 2-6
CREATE VIEW statement, in sysviews table 1-10
customer table in sales_demo database, columns in B-4
customer table in stores_demo database, columns in A-3
customer table in superstores_demo database, columns in B-12, B-13, B-14
cust_calls table in stores_demo database, columns in A-8
cust_calls table in superstores_demo database, columns in B-11

D

Data distributions, specifying disk space to use 3-56
Data type
  approximate 1-89
  BLOB 2-9
  BOOLEAN 2-11
  BYTE 2-12
  casting 2-63 to 2-70
  categories of 2-5
  CHAR 2-13
  CHARACTER 2-14
  CHARACTER VARYING 2-15
  CLOB 2-15
  conversion 2-63
  DATE 2-16
  DATETIME 2-17
  DEC 2-21
  DECIMAL 2-21
  description of 2-9
  Distinct 2-23
  DOUBLE PRECISION 2-24
  exact numeric 1-89
  extended 2-58
  FLOAT 2-24
  floating-point 2-25
  for sequential integer 2-40
  for unique numeric code 2-40
  INT 2-25
  INT8 2-25
  INTEGER 2-26
  internal 2-9
  INTERVAL 2-26
  LIST 2-30
  LVARCHAR 2-31
  MONEY 2-32
  MULTISET 2-33
  Named row 2-36
  NCHAR 2-34
  NUMERIC 2-34
  NVARCHAR 2-34
  Opaque 2-35
  REAL 2-35
  Row, Named 2-36
  Row, Unnamed 2-37
  SERIAL 2-39
  SERIAL8 2-40
  SET 2-42
  simple large object 2-50
  SMALLFLOAT 2-43
  SMALLINT 2-44
  smart large object 2-50
  summary list 2-6
  TEXT 2-44
  Unnamed row 2-37
  VARCHAR 2-46
Database
  data types for 2-5
  map of stores_demo A-11
  objects, state of in sysobjectstate table 1-55
  sales_demo, description of B-2
  stores_demo, description of A-1
  superstores_demo, description of B-3, B-8
Database server
  attributes in Information Schema view 1-90
  pathname for 3-46
  specifying default for connection 3-70
DataBlade
  message information (syserrors) 1-41
  trace classes (systraceclasses) 1-75
  trace messages (systracemsgs) 1-76
Data-distribution information, in sysdistrib table 1-39
DATE data type
  casting to integer 2-66
  converting to DATETIME 2-66
  DATETIME, INTERVAL with 2-52, 2-55
  description of 2-16
  international date formats 2-17

source data 2-56
two-digit year values and DBCENTURY variable 2-17
Date value, setting DBDATE environment variable 3-36
DATETIME data type
character string values 2-20
converting to DATE 2-66
converting to integer 2-66
DATE INTERVAL with 2-52 to 2-57
description of 2-17
EXTEND function with 2-54
extending fields in 2-53
field qualifiers 2-18
formats with DBTIME 3-53
international date and time formats 2-20
length (syscolumns) 1-34
precision and size 2-18
source data 2-56
specifying display format 3-53
two-digit year values and DBDATE variable 2-20
using the DBTIME environment variable 3-53
year to fraction example 2-19
DAY keyword
as DATETIME field qualifier 2-18
as INTERVAL field qualifier 2-27
DB-Access utility Intro-5
DBACCNOIGN environment variable 3-29
DBANSIWARN environment variable 3-30
DBBLOBBUF environment variable 3-31
DBCENTURY environment variable 2-17
description of 3-32
effect on expressions with date values 3-34
effect on functionality of DBDATE 3-38
expanding date values, examples 3-33
DBDATE environment variable 2-17, 3-36
effect on expressions with date values 3-38
DBDELIMITER environment variable 3-39
DBEDIT environment variable 3-40
dbexport utility, specifying field delimiter with DBDELIMITER 3-39, 3-57
DBFLTMASK environment variable 3-41
DBLANG environment variable 3-42
dbload utility, specifying field delimiter with DBDELIMITER 3-40
DBMONEY environment variable 2-32, 3-44
DBONPLOAD environment variable 3-45
DBPATH environment variable 3-46
DBPRINT environment variable 3-48
DBREMOTECMD environment variable 3-49
Dbserver group, value in INFORMIXSERVER 3-71
dbservername.cmd batch file 3-18
DBSPACETEMP environment variable 3-50
DBTEMP environment variable 3-52
DBTIME environment variable 2-20, 3-53
DBUPSPACE environment variable 3-56
DCE-GSS communications support module (CSM), environment variable for 3-69
DEC data type. *See* DECIMAL data type.
DECIMAL data type
casts for 2-65
description of 2-21
disk storage 2-22
fixed point 2-21
floating point 2-21
length (syscolumns) 1-33
Decimal digits, display of 3-41
Decimal point
as delimiter in DATETIME 2-19
as delimiter in INTERVAL 2-29
Default
column, in sysdefaults table 1-37
configuration file 3-82
Default locale Intro-4
DELIMIDENT environment variable 3-57
Delimited identifier, setting DELIMIDENT environment variable 3-57
Delimiter
for DATETIME values 2-19
for INTERVAL values 2-29
Demonstration database
map of A-11
structure of tables A-2, B-9
tables in A-3 to A-10
Demonstration databases Intro-5
Dependencies, software Intro-4
Diagnostics, for base tables, described in sysviolations table 1-81
Directives for query optimization, environment variable for 3-61
Disk space
for temporary data 3-51
specifying for data distributions 3-56
Distinct data type
casts 2-69
description of 2-23
sysxtddesc contents for 1-82
sysxtdtypeauth contents for 1-83
sysxtdtypes contents for 1-84
Documentation notes Intro-14

Documentation, types of
documentation notes Intro-14
error message files Intro-13
machine notes Intro-14
on-line manuals Intro-13
printed manuals Intro-13
related reading Intro-15
release notes Intro-14
DOUBLE PRECISION data type. *See* FLOAT data type.

E

Editor, specifying with DBEDIT 3-40
ENVIGNORE environment variable
description 3-9, 3-58
relation to chkenv utility 3-13
Environment configuration file
debugging with chkenv 3-12
setting environment variables in UNIX 3-7, 3-8
Environment variable
AC_CONFIG 3-24
ARC_CONFIG 3-25
ARC_DEFAULT 3-25
ARC_KEYPAD 3-26
COCKPITSERVICE 3-27
command prompt utilities 3-16
CPFIRST 3-28
DBACCNOIGN 3-29
DBANSIWARN 3-30
DBBLOBBUF 3-31
DBCENTURY 2-17, 3-32
DBDATE 2-17, 3-36
DBDELIMITER 3-39
DBEDIT 3-40
DBFLTMASK 3-41
DBLANG 3-42
DBMONEY 2-32, 3-44
DBONPLOAD 3-45
DBPATH 3-46
DBPRINT 3-48
DBREMOTECMD 3-49
DBSPACETEMP 3-50
DBTEMP 3-52
DBTIME 2-20, 3-53
DBUPSPACE 3-56
definition of 3-5
DELIMIDENT 3-57
ENVIGNORE 3-58
FET_BUF_SIZE 3-59
for command prompt utilities for Windows
set in autoexec.bat 3-16
set in System applet 3-15
set on command-line 3-16
user environment variables 3-16
variable scope 3-15
GL_DATE 2-17
GL_DATETIME 2-20
how to set in Bourne shell 3-10
how to set in C shell 3-10
how to set in Korn shell 3-10
IFMX_SMLTBL_BROADCAST_SIZE 3-60
IFX_DIRECTIVES 3-61
IFX_LONGID 3-62
IFX_NETBUF_PVTPOOL_SIZE 3-63
IFX_NETBUF_SIZE 3-63
IFX_UPDDESC 3-64
INFORMIXC 3-64
INFORMIXCONCSMCFG 3-65
INFORMIXCONRETRY 3-66
INFORMIXCONTIME 3-66
INFORMIXCPPMAP 3-68
INFORMIXDIR 3-68
INFORMIXKEYTAB 3-69
INFORMIXOPCACHE 3-70
INFORMIXSERVER 3-70
INFORMIXSHMBASE 3-72, 3-73
INFORMIXSQLHOSTS 3-73
INFORMIXSTACKSIZE 3-74
INFORMIXTERM 3-75
INF_ROLE_SEP 3-76
ISM_COMPRESSION 3-77
ISM_DEBUG_FILE 3-77
ISM_DEBUG_LEVEL 3-78
ISM_ENCRYPTION 3-79
ISM_MAXLOGSIZE 3-79
ISM_MAXLOGVERS 3-80
LD_LIBRARY_PATH 3-80
LIBPATH 3-81
listed alphabetically 3-19
listed by topic 3-99
manipulating in Windows environments 3-14
modifying 3-11
NODEFDAC 3-81
ONCONFIG 3-82
OPTCOMPIND 3-83
OPTMSG 3-84
OPTOFC 3-85
OPT_GOAL 3-86
overriding a setting 3-9, 3-58
PATH 3-87
PDQPRIORITY 3-88
PLCONFIG 3-90
PLOAD_LO_PATH 3-91
PLOAD_SHMBASE 3-91
PSORT_DBTEMP 3-92
PSORT_NPROCS 3-93
rules of precedence in UNIX 3-13
rules of precedence in Windows NT 3-18
setting
at the command line 3-7
for command prompt utilities 3-15
for native Windows NT applications 3-14
in a login file 3-7
in a shell file 3-9
in an environment-configuration file 3-7
in autoexec.bat 3-16
in Windows environments 3-7
on the command-line 3-16
using the Registry Editor 3-14
using the System applet 3-15
SHLIB_PATH 3-94
STMT_CACHE 3-95
TERM 3-96
TERMCAP 3-96
TERMINFO 3-97
THREADLIB 3-98
types of 3-6
variable scope 3-15
view current setting 3-12
where to set 3-9
XFER_CONFIG 3-98
Environment variables Intro-7
en_us.8859-1 locale Intro-4

Error message files Intro-13
ESQL/C
  compiling multithreaded applications 3-98
  program compilation order 3-28
Executable programs, where to search 3-87
Explicit cast 2-68
EXTEND function, example 2-54
Extended data types 2-58
Extension checking, specifying with DBANSIWARN 3-30
Extent, changing size of system table 1-12

F

Feature icons Intro-8
Features of this product, new Intro-5
FET_BUF_SIZE environment variable 3-59
Field delimiter files, DBDELIMITER 3-39
Field qualifier for DATETIME 2-18
File
  shell 3-9
  temporary
    directory for (DBTEMP) 3-52
  temporary, for database server 3-50
  temporary, sorting 3-92
  termcap, terminfo 3-75, 3-96, 3-97
  with environment configuration information 3-12
  /etc/termcap 3-97
FILETOBLOB function 2-10
FILETOCLOB function 2-15
Find Error utility Intro-13
finderr utility Intro-13
Fixed point decimal 2-21
FLOAT data type
  casts for 2-65
  description of 2-24
Floating point decimal 2-21
Format
  specifying for DATE value with DBDATE 3-37
  specifying for DATETIME value with DBTIME 3-53
  specifying for MONEY value with DBMONEY 3-44
FRACTION keyword
  as DATETIME field qualifier 2-18
  as INTERVAL field qualifier 2-27
Fragmentation
  information in sysfragauth table 1-45
  information in sysfragments table 1-46
  PDQPRIORITY environment variable 3-89
  setting priority levels for PDQ 3-88
Function
  support for complex types 2-60
  used with BLOB columns 2-10
  used with CLOB columns 2-15
  used with LIST 2-31
  used with MULTISET 2-33
  used with SET 2-42

G

Generalized-key index
  sysnewdepend information 1-54
  sysrepository information 1-67
geography table in sales_demo database, columns in B-4
Global Language Support (GLS) Intro-4
GL_DATE environment variable 2-17
GL_DATETIME environment variable 2-20

H

High-Performance Loader, environment variable for 3-45, 3-90
HOUR keyword
  as DATETIME field qualifier 2-18
  as INTERVAL field qualifier 2-27
Hyphen
  as delimiter in DATETIME 2-19
  as delimiter in INTERVAL 2-29

I

Icons
  feature Intro-8
  Important Intro-8
  platform Intro-8
  product Intro-8
  Tip Intro-8
  Warning Intro-8
IFMX_SMLTBL_BROADCAST_SIZE environment variable 3-60
IFX_DIRECTIVES environment variable 3-61
IFX_LONGID environment variable 3-62
IFX_NETBUF_PVTPOOL_SIZE environment variable 3-63
IFX_NETBUF_SIZE environment variable 3-63
IFX_UPDDESC environment variable 3-64
Implicit cast 2-67
Important paragraphs, icon for Intro-8
Index
  default values for attached 3-94
  generalized-key, in sysnewdepend 1-54
  generalized-key, in sysrepository 1-67
  of environment variables 3-99
  sysindexes information 1-49
  sysindices information 1-51
  sysobjstate information 1-55
  threads for 3-94
Indexkey structure, fields in 1-52

Industry standards, compliance with Intro-15
Information Schema views
  accessing 1-87
  columns 1-88
  description of 1-86
  generating 1-86
  server_info 1-90
  sql_languages 1-90
  tables 1-88
Informix extension checking, specifying with DBANSIWARN 3-30
Informix Storage Manager 3-77, 3-79
INFORMIXC environment variable 3-64
INFORMIXCONCSMCFG environment variable 3-65
INFORMIXCONRETRY environment variable 3-66
INFORMIXCONTIME environment variable 3-66
INFORMIXCPPMAP environment variable 3-68
INFORMIXDIR environment variable 3-68
INFORMIXDIR/bin directory Intro-5
INFORMIXKEYTAB environment variable 3-69
INFORMIXOPCACHE environment variable 3-70
INFORMIXSERVER environment variable 3-70
INFORMIXSHMBASE environment variable 3-72, 3-73
INFORMIXSTACKSIZE environment variable 3-74
INFORMIXTERM environment variable 3-75
INF_ROLE_SEP environment variable 3-76
Input support function 2-31
Inserting values into SERIAL columns 2-40
Installation directory, specifying with INFORMIXDIR 3-68
Installation files, INFORMIXDIR environment variable 3-68
INT data type. *See* INTEGER data type.
INT8 data type
  casts for 2-65, 2-66
  description of 2-25
  using with SERIAL8 2-41
INTEGER data type
  casts for 2-65, 2-66
  description of 2-26
  length (syscolumns) 1-33
Intensity attributes, setting INFORMIXTERM for 3-75
INTERVAL data type
  DATE, DATETIME with 2-52, 2-57 to 2-58
  description of 2-26
  expressions with 2-52
  EXTEND function with 2-54
  field delimiters 2-29
  length (syscolumns) 1-34
ISM_COMPRESSION environment variable 3-77
ISM_DEBUG_FILE environment variable 3-77
ISM_DEBUG_LEVEL environment variable 3-78
ISM_ENCRYPTION environment variable 3-79
ISM_MAXLOGSIZE environment variable 3-79
ISM_MAXLOGVERS environment variable 3-80
ISO 8859-1 code set Intro-4
items table in stores_demo database, columns in A-5
items table in superstores_demo database, columns in B-15

K

keytab file, environment variable to specify path 3-69
Korn shell, .profile file 3-9

L

Language
  environment, setting with DBLANG 3-42
  sql_languages information schema view 1-90
  sysroutinelangs information 1-68
Large-object data type
  description of 2-49
  list of 2-48
LD_LIBRARY_PATH environment variable 3-80
LIBPATH environment variable 3-81
List
  of environment variables 3-19
  of environment variables, by topic 3-99
  of system catalog tables 1-14
LIST data type, description of 2-30
LOAD statement, specifying field delimiter with DBDELIMITER 3-40
Locale Intro-4
  default Intro-4
  en_us.8859-1 Intro-4
LOCOPY function 2-10, 2-15
Long identifiers
  and client version 3-62
  and IFX_LONGID environment variable 3-62
LOTOFILE function 2-10, 2-15
LVARCHAR data type, description of 2-31

M

Machine notes Intro-14
manufact table in superstores_demo database, columns in B-19
Memory cache, for staging blobspace 3-70
Message file
  specifying subdirectory with DBLANG 3-42
  XBSA 3-77

Message file for error
messages Intro-13
Messages
ANSI warning 3-30
error in syserrors 1-41
optimized transfers 3-84
reducing requests 3-85
warning in syserrors 1-41
MINUTE keyword
as DATETIME field qualifier 2-18
as INTERVAL field qualifier 2-27
MONEY data type
casts for 2-65
description of 2-32
display format specified with
DBMONEY 3-44
international money formats 2-32
length (syscolumns) 1-33
MONTH keyword
as DATETIME field qualifier 2-18
as INTERVAL field qualifier 2-27
Multibyte characters
with CHAR data type 2-14
with CLOB data type 2-16
with TEXT data type 2-46
with VARCHAR data type 2-48
MULTISET data type, description
of 2-33

N

Named row data type
casting permitted 2-70
defining 2-36
description of 2-36
equivalence 2-37
inheritance 2-37
typed tables 2-37
See also Row type.
NCHAR data type, description
of 2-34
Network environment variable,
DBPATH 3-46
New features of this
product Intro-5
NODEFDAC environment
variable 3-81
Nonprintable characters
with CHAR data type 2-14
with TEXT data type 2-46
with VARCHAR data type 2-47
Not null constraint, described in
syscoldepend table 1-29
Null value
allowed or not allowed 1-31
testing in BYTE expression 2-12
testing with TEXT data type 2-45
Numeric data types
casting between 2-65
casting to characters 2-65
list of 2-48
NUMERIC data type. *See*
DECIMAL data type.
NVARCHAR data type
description of 2-34
length (syscolumns) 1-33

O

Object mode of database objects,
described in sysobjstate
table 1-55
ONCONFIG environment
variable 3-82
On-line manuals Intro-13
Opaque data type
cast matrix 2-70
description of 2-35
smart large objects with 2-50
storage of 2-31
sysxtddesc contents for 1-82
sysxtdtypeauth contents for 1-83
sysxtdtypes contents for 1-84
Operator class
sysams information 1-18
sysopclasses information 1-56
Operator precedence 2-71
OPTCOMPIND environment
variable 3-83
Optical cluster, described in
sysopclstr table 1-57
OPTMSG environment
variable 3-84
OPTOFC environment
variable 3-85
OPT_GOAL environment
variable 3-86
orders table in superstores_demo
database, columns in B-16, B-17,
B-18, B-20
Output support function 2-31

P

Parallel database query. *See* PDQ.
Parallel distributed queries, setting
with PDQPRIORITY
environment variable 3-88
Parallel sorting, using
PSORT_NPROCS for 3-92
PATH environment variable 3-87
Pathname
for C compiler 3-64
for database server 3-46
for executable programs 3-87
for installation 3-68
for message files 3-42
for parallel sorting 3-92
for remote shell 3-49
specifying with DBPATH 3-46
specifying with PATH 3-87
PDQ
OPTCOMPIND environment
variable 3-83
PDQPRIORITY environment
variable 3-88
Platform icons Intro-8
PLCONFIG environment
variable 3-90
PLOAD_LO_PATH environment
variable 3-91
PLOAD_SHMBASE environment
variable 3-91
Precedence
operator 2-71
rules for command-line utility
environment variables 3-18
rules for environment variables
for Informix native Windows
applications 3-19
rules for environment variables in
UNIX 3-13

Prepared statement, version number in systables 1-75
Printed manuals Intro-13
Printing with DBPRINT 3-48
Privilege
  granting or preventing default table 3-81
  on a table fragment, described in sysfragauth table 1-45
  on a table, described in systabauth 1-72
  on database, described in the sysusers table 1-79
  related to routines, described in sysprocauth table 1-60
  usage, sysxtdtypeauth information 1-83
  user, described in sysusers table 1-79
Product icons Intro-8
product table in sales_demo database, columns in B-5
Program group
  Documentation notes Intro-15
  Release notes Intro-15
Protected routines 1-64
PSORT_DBTEMP environment variable 3-92
PSORT_NPROCS environment variable 3-93

Q

Qualifier field, for DATETIME 2-18
Query optimization information in sysprocplan table 1-65
Query optimizer, environment variable for directives 3-61
Quoted string, for DATE and DATETIME 2-56

R

REAL data type. *See* SMALLFLOAT data type.
regedt32.exe Registry Editor 3-14
region table in superstores_demo database, columns in B-20
Related reading Intro-15
Release notes Intro-14
Remote shell 3-49
rofferr utility Intro-13
Role
  granted to user, described in sysroleauth table 1-68
  separation, environment variable for 3-76
Routine
  characteristics, in sysprocedures table 1-61
  DATETIME formatting 3-53
  privileges on 1-59
  privileges, described in sysprocauth table 1-60
  protected 1-64
  query plan for, in sysprocplan 1-65
  SPL 2-61
  user-defined 2-35
  user-defined, and syserrors 1-41
  user-defined, and syslangauth 1-53
  user-defined, and sysroutinelangs 1-68
  user-defined, and systraceclasses 1-75
  user-defined, and systracemsgs 1-76
Row type 2-61
  casting permitted 2-70
  defined 2-61
  field information (sysattrtypes) 1-23
  Named, and equivalence 2-37
  Named, and inheritance 2-37
  Named, defining 2-36
  Named, description of 2-36
  sysattrtypes information 1-23
  sysxtddesc information 1-82
  sysxtdtypeauth information 1-83
  sysxtdtypes information 1-84
  Unnamed, defining 2-38
  Unnamed, description of 2-37
  Unnamed, inserting values into 2-39
Runtime program, setting DBANSIWARN 3-30

S

sales table in sales_demo database, columns in B-6
sales_demo database
  customer table columns B-4
  description of B-3
  geography table columns B-4
  introduced B-1
  product table columns B-5
  sales table columns B-6
  time table columns B-7
sales_rep table in superstores_demo database, columns in B-21
Sample-code conventions Intro-12
sbspace, defined 2-51
SECOND keyword
  as DATETIME field qualifier 2-18
  as INTERVAL field qualifier 2-27
Sequential integers
  with SERIAL data type 2-39
  with SERIAL8 data type 2-40
SERIAL data type
  description of 2-39
  inserting values 2-40
  length (syscolumns) 1-33
  resetting values 2-40
SERIAL8 data type
  assigning a starting value to 2-41
  description of 2-40
  inserting values 2-41
  length (syscolumns) 1-33
  resetting values 2-41
  using with INT8 2-41
SET data type, description of 2-42
Setnet32 utility 3-8
Setting environment variables
  in UNIX 3-8
  in Windows NT 3-14
Shared memory, setting with INFORMIXSHMBASE 3-72
Shell
  remote 3-49
  search path 3-87
  setting environment variables in a file 3-9
  specifying with DBREMOTECMD 3-49

SHLIB_PATH environment variable 3-94
Simple large objects
  description of 2-50
  length (syscolumns) 1-34
Single-precision floating-point number, storage of 2-24
SMALLFLOAT data type
  casts for 2-65
  description of 2-43
SMALLINT data type
  casts for 2-65, 2-66
  description of 2-44
  length (syscolumns) 1-33
Smart large objects
  attributes in syscolattribs 1-27
  description of 2-50
Software dependencies Intro-4
Sorting
  DBSPACETEMP environment variable 3-50
  PSORT_DBTEMP environment variable 3-92
  PSORT_NPROCS environment variable 3-93
  threads for 3-93
Space
  as delimiter in DATETIME 2-19
  as delimiter in INTERVAL 2-29
SQL code Intro-12
SQL Communications Area (SQLCA), effect of setting DBANSIWARN 3-30
Stacksize, setting INFORMIXSTACKSIZE 3-74
state table in stores_demo database, columns in A-10
state table in superstores_demo database, columns in B-21
Statement, SQL
  ANSI compliance and DBANSIWARN 3-30
  CONNECT and INFORMIXSERVER 3-71
  editing and DBEDIT 3-40, 3-45
  LOAD and DBDELIMITER 3-39, 3-57
  printing and DBPRINT 3-48
  UNLOAD and DBDELIMITER 3-39, 3-57
  UPDATE STATISTICS and DBUPSPACE 3-56
STMT_CACHE environment variable 3-95
stock table in stores_demo database, columns in A-6
stock table in superstores_demo database, columns in B-22
stock_discount table in superstores_demo database, columns in B-23
stores_demo database
  call_type table columns A-9
  catalog table columns A-7
  customer table columns A-3
  cust_calls table columns A-8
  data values A-19
  description of A-1
  items table columns A-5
  manufact table columns A-9
  map of A-11
  primary-foreign key relationships A-12 to A-19
  stock table columns A-6
  structure of tables A-2
Summary
  of environment variables 3-19
  of environment variables, by topic 3-99
  of system catalog tables 1-14
superstores_demo database
  call_type table columns B-10
  catalog table columns B-10
  customer table columns B-12, B-13, B-14
  cust_calls table columns B-11
  description of B-8
  introduced B-1
  items table columns B-15
  manufact table columns B-19
  orders table columns B-16, B-17, B-18, B-20
  primary-foreign key relationships B-27 to B-30
  sales_rep table columns B-21
  stock table columns B-22
  stock_discount table columns B-23
  structure of tables B-9
Synonym
  for each table view, described in syssynonyms table 1-69
  for each table, described in syssyntable 1-70
sysaggregates system catalog table 1-16
sysams system catalog table 1-17
sysattrtypes system catalog table 1-23
sysblobs system catalog table 1-24
syscasts system catalog table
  described 1-25
  for casting 2-63
syschecks system catalog table 1-26
syscolattribs system catalog table 1-27
syscolauth system catalog table 1-28
syscoldepend system catalog table 1-29
syscolumns system catalog table 1-30
sysconstraints system catalog table 1-36
sysdefaults system catalog table 1-37
sysdepend system catalog table 1-38
sysdistrib system catalog table 1-39
syserrors system catalog table 1-41
sysextcols system catalog table 1-42
sysextdfiles system catalog table 1-43
sysexternal system catalog table 1-44
sysfragauth system catalog table 1-45
sysfragments system catalog table 1-46
sysindexes system catalog table 1-49
sysindices system catalog table 1-51
sysinherits system catalog table 1-53

syslangauth system catalog
table 1-53
syslogmap system catalog
table 1-54
sysmaster database
initialization 3-71
versus system catalog tables 1-6
sysnewdepend system catalog
table 1-54
sysobjstate system catalog
table 1-55
sysopclasses system catalog
table 1-56
sysopclstr system catalog table 1-57
sysprocauth system catalog
table 1-59
sysprocbody system catalog
table 1-60
sysprocedures system catalog
table 1-61
sysprocplan system catalog
table 1-65
sysreferences system catalog
table 1-66
sysrepository system catalog
table 1-67
sysroleauth system catalog
table 1-68
sysroutinelangs system catalog
table 1-68
syssynonyms system catalog
table 1-69
syssyntable system catalog
table 1-70
systabamdata system catalog
table 1-71
systabauth system catalog
table 1-72
systables system catalog table 1-73
System catalog
accessing 1-12
altering contents 1-12
character variable length 1-33
collection data type elements 1-23
complex data type members 1-23
data type privileges 1-83
description of 1-5
example
syscolauth 1-10
syscolumns 1-9
sysindexes 1-11
systabauth 1-10
systables 1-8
sysviews 1-10
how used by database server 1-7
list of tables 1-14
messages 1-41
operator class table
(sysopclasses) 1-56
privilege information for user-
defined data type 1-83
programming languages
supported 1-53, 1-68
smart large object attributes
(syscolattribs) 1-27
sysaggregates 1-16
sysams 1-17
sysattrtypes 1-23
sysblobs 1-24
syscasts 1-25
syschecks 1-26
syscolattribs 1-27
syscolauth 1-28
syscoldepend 1-29
syscolumns 1-30
sysconstraints 1-36
sysdefaults 1-37
sysdepend 1-38
sysdistrib 1-39
syserrors 1-41
sysextcols 1-42
sysextdfiles 1-43
sysexternal 1-44
sysfragauth 1-45
sysfragments 1-46
sysindexes 1-49
sysindices 1-51
sysinherits 1-53
syslangauth 1-53
syslogmap 1-54
sysnewdepend 1-54
sysobjstate 1-55
sysopclasses 1-56
sysopclstr 1-57
sysprocauth 1-59
sysprocbody 1-60
sysprocedures 1-61
sysprocplan 1-65
sysreferences 1-66
sysrepository 1-67
sysroleauth 1-68
sysroutinelangs 1-68
syssynonyms 1-69
syssyntable 1-70
systabamdata 1-71
systabauth 1-72
systables 1-73
systraceclasses 1-75
systracemsgs 1-76
systrigbody 1-77
systriggers 1-78
sysusers 1-79
sysviews 1-80
sysviolations 1-81
sysxtddesc 1-82
sysxtdtypeauth 1-83
sysxtdtypes 1-84
table indexes 1-51
table inheritance 1-53
trace classes 1-75
trace messages 1-76
updating statistics 1-13
updating system catalog
tables 1-13
user-defined casts 1-25
user-defined data type
descriptions 1-82
using 1-6
versus sysmaster database 1-6
System requirements
database Intro-4
software Intro-4
systraceclasses system catalog
table 1-75
systracemsgs system catalog
table 1-76
systrigbody system catalog
table 1-77
systriggers system catalog
table 1-78
sysusers system catalog table 1-79
sysviews system catalog table 1-80
sysviolations systems catalog
table 1-81

sysxtddesc system catalog table 1-82
sysxtdtypeauth system catalog table 1-83
sysxtdtypes system catalog table
  described 1-84
  for Distinct types 2-23
  for named row types 2-36
  for opaque types 2-35

T

tabid, description of 1-9
Table
  changing the data type of a column 2-63
  dependencies, in sysdepend 1-38
  description in systables 1-73
  inheritance, sysinherits information 1-53
  separate from large object storage 2-49
  structure in stores_demo database A-2
  structure in superstores_demo database B-9
  synonyms in syssyntable 1-69
  system catalog tables 1-16 to 1-84
  typed, and named row type 2-37
  untyped, and unnamed row 2-39
Table-level privilege, shown in tabauth table 1-10
Tape management
  setting ARC_DEFAULT 3-25
  setting ARC_KEYPAD 3-26
  setting DBREMOTECMD 3-49
Temporary files
  in database server, setting DBSPACETEMP 3-50
  setting PSORT_DBTEMP 3-92
Temporary tables
  directory location (DBTEMP) 3-52
  specifying dbspace with DBSPACETEMP 3-50
TERM environment variable 3-96
TERMCAP environment variable 3-96
termcap file, and TERMCAP environment variable 3-91, 3-96, 3-98
Terminal handling
  and TERM environment variable 3-96
  and TERMCAP environment variable 3-96
  and TERMINFO environment variable 3-97
  setting INFORMIXTERM 3-75
terminfo directory 3-97
TERMINFO environment variable 3-97
TEXT data type
  casting to CLOB data type 2-45
  collation 2-46
  description of 2-44
  increasing buffer size 3-31
  inserting values 2-45
  length (syscolumns) 1-34
  location shown in sysblobs table 1-24
  nonprintable characters with 2-46
  restrictions
    with aggregate functions 2-45
    with GROUP BY 2-45
    with IN clause 2-45
    with LIKE or MATCHES 2-45
    with ORDER BY 2-45
  selecting a column 2-45
  setting buffer size 3-31
  use in Boolean expression 2-45
  with control characters 2-44
Text editor, specifying with DBEDIT 3-40
THREADLIB environment variable 3-98
Time data type
  length (syscolumns) 1-34
  list of 2-48
time table in sales_demo database, columns in B-7
Time value, setting DBTIME environment variable 3-53
Tip icons Intro-8
Trigger
  information in sysobjstate 1-55
  information in systrigbody 1-77
  information in systriggers 1-78

U

UDR. *See* User-defined routine.
Unnamed row data type
  See also Row type.
Unique numeric code
  with SERIAL data type 2-40
  with SERIAL8 data type 2-40
units table in superstores_demo database, columns in B-24
UNIX
  BSD, default print capability 3-48
  environment variables 3-6
  PATH environment variable 3-87
  specifying directories for intermediate writes 3-92
  System V
    default print capability 3-48
    terminfo library support 3-75
  TERM environment variable 3-96
  TERMCAP environment variable 3-96
  TERMINFO environment variable 3-97
UNIX operating system
  default locale for Intro-4
UNLOAD statement, specifying field delimiter with DBDELIMITER 3-40
Unnamed row data type
  defining 2-38
  description of 2-37
  inserting values into columns 2-39
UPDATE STATISTICS statement
  and DBUPSPACE environment variable 3-56
  effect on sysdistrib table 1-39
  sysindices (index statistics) 1-52
  update system catalog 1-13
User privileges, described in sysusers table 1-79
User-defined casts, using 2-67

User-defined data type
  casting 2-67
  casting into built-in type 2-63
  kinds of 2-58
  opaque 2-62
  privileges in sysxtdtypeauth 1-83
  sysxtddesc (extended type) information 1-82
  sysxtdtypes information about 1-84
User-defined name, for unnamed row 2-37
User-defined routine
  casts (syscasts) 1-25
  error messages (syserrors) 1-41
  for opaque type 2-35
  language authorization (syslangauth) 1-53
Users, types of Intro-3
Utility
  archecker 3-24
  chkenv 3-12
  command-line 3-14
  command-prompt 3-15
  DB-Access 3-30, 3-41, 3-70
  dbexport 3-39
  ON-Archive 3-25, 3-26
  ON-Bar 3-77, 3-78, 3-79
  onpload 3-45
  Setnet32 3-8
  setting environment variables for DB-Access 3-8

V

VARCHAR data type
  collation 2-48
  description of 2-46
  length in syscolumns 1-33
  multibyte characters with 2-48
  nonprintable characters with 2-47
  storing numeric values in column 2-47
  See also CHARACTER VARYING data type.
View
  dependencies, in sysdepend table 1-38
  described in sysviews table 1-80
  synonyms in syssynonyms table 1-69
Violations, for base tables, described in sysviolations table 1-81

W

Warning icons Intro-8
Warning, creating a new message 1-41
Windows environments
  manipulating environment variables 3-14
  setting environment variables 3-7
Windows NT
  default locale for Intro-4

X

XFER_CONFIG environment variable 3-98
X/Open
  and server_info view 1-91
  compliant databases 1-91
  Information Schema views 1-86
X/Open compliance level Intro-15

Y

YEAR keyword
  as DATETIME field qualifier 2-18
  as INTERVAL field qualifier 2-27
Year values, two and four digit 3-32

Symbols

( ), space, as delimiter
  in DATETIME 2-19
  in INTERVAL 2-29
-, hyphen, as delimiter
  in DATETIME 2-19
  in INTERVAL 2-29
:, colon, as delimiter
  in DATETIME 2-19
  in INTERVAL 2-29
., decimal point, as delimiter
  in DATETIME 2-19
  in INTERVAL 2-29