

INFORMIX[®]-Universal Server

Informix Guide to SQL: Reference

Version 9.1
March 1997
Part No. 000-3855

Published by INFORMIX® Press

Informix Software, Inc.
4100 Bohannon Drive
Menlo Park, CA 94025

Copyright © 1981-1997 by Informix Software, Inc. or their subsidiaries, provided that portions may be copyrighted by third parties, as set forth in documentation. All rights reserved.

The following are worldwide trademarks of Informix Software, Inc., or its subsidiaries, registered in the United States of America as indicated by “®,” and in numerous other countries worldwide:

INFORMIX®; INFORMIX®-OnLine Dynamic Server™; DataBlade®

The following are worldwide trademarks of the indicated owners or their subsidiaries, registered in the United States of America as indicated by “®,” and in numerous other countries worldwide:

Adobe Systems Incorporated: PostScript®
X/Open Company Ltd.: UNIX®; X/Open®
Micro Focus Ltd.: Micro Focus®; Micro Focus COBOL/2™
Ryan-McFarland (Liant) Corporation: Ryan-McFarland®

All other marks or symbols are registered trademarks or trademarks of their respective owners.

Documentation Team: Brian Deutscher, Evelyn Eldridge-Diaz, Smita Joshi, Geeta Karmarker

To the extent that this software allows the user to store, display, and otherwise manipulate various forms of data, including, without limitation, multimedia content such as photographs, movies, music and other binary large objects (blobs), use of any single blob may potentially infringe upon numerous different third-party intellectual and/or proprietary rights. It is the user's responsibility to avoid infringements of any such third-party rights.

RESTRICTED RIGHTS/SPECIAL LICENSE RIGHTS

Software and documentation acquired with US Government funds are provided with rights as follows: (1) if for civilian agency use, with Restricted Rights as defined in FAR 52.227-19; (2) if for Dept. of Defense use, with rights as restricted by vendor's standard license, unless superseded by negotiated vendor license as prescribed in DFAR 227.7202. Any whole or partial reproduction of software or documentation marked with this legend must reproduce the legend.

Table of Contents

Introduction

About This Manual	3
Organization of This Manual	4
Types of Users	5
Software Dependencies	5
Assumptions About Your Locale.	5
Demonstration Database	6
Major Features	6
Documentation Conventions	8
Typographical Conventions	8
Icon Conventions	9
Command-Line Conventions	10
Sample-Code Conventions.	13
Additional Documentation	14
On-Line Manuals	14
Printed Manuals	14
Error Message Files	15
Documentation Notes, Release Notes, Machine Notes	15
Compliance with Industry Standards	16
Informix Welcomes Your Comments	16

Chapter 1

System Catalog

Objects Tracked by the System Catalog Tables	1-3
Using the System Catalog	1-4
Accessing the System Catalog.	1-10
Updating System Catalog Data	1-10
Structure of the System Catalog	1-11
SYSAMS	1-12
SYSATTRTYPES	1-17
SYSBLOBS	1-18
SYSCASTS	1-19
SYSCHECKS	1-20

SYSCOLATTRIBS	1-20
SYSCOLAUTH	1-22
SYSCOLDEPEND	1-23
SYSCOLUMNS	1-23
SYSCONSTRAINTS	1-29
SYSDEFAULTS	1-30
SYSDEPEND	1-32
SYSDISTRIB	1-32
SYSERRORS	1-33
SYSFRAGAUTH	1-34
SYSFRAGMENTS	1-36
SYSINDICES	1-37
SYSINHERIS	1-39
SYSLANGAUTH	1-40
SYSLOGMAP	1-40
SYSOBJSTATE	1-40
SYSOPCLASSES	1-42
SYSOPCLSTR	1-43
SYSPROCAUTH	1-44
SYSPROCBODY	1-45
SYSPROCEDURES	1-46
SYSPROCPLAN	1-48
SYSREFERENCES	1-49
SYSROLEAUTH	1-50
SYSROUTINELANGS	1-51
SYSSYNONYMS	1-51
SYSSYNTABLE	1-52
SYSTABAUTH	1-53
SYSTABLES	1-54
SYSTRACECLASSES	1-56
SYSTRACEMSGS	1-57
SYSTRIGBODY	1-58
SYSTRIGGERS	1-59
SYSUSERS	1-60
SYSVIEWS	1-61
SYSVIOLATIONS	1-61
SYSXTDDESC	1-62
SYSXTDTYPEAUTH	1-63
SYSXTDTYPES	1-64
Information Schema	1-65
Generating the Information Schema Views	1-66
Accessing the Information Schema Views	1-67
Structure of the Information Schema Views.	1-67

Chapter 2

Data Types

Data Types Supported by Universal Server	2-5
Summary of Data Types	2-7
Built-In Data Types	2-9
Using DATE, DATETIME, and INTERVAL Data	2-10
Large-Object Data Types.	2-16
Extended Data Types	2-20
Complex Data Types	2-20
Opaque Data Types	2-24
Distinct Data Types	2-24
Data Type Casting	2-25
Using System-Defined Casts	2-25
Using Implicit Casts	2-29
Using Explicit Casts	2-29
What Extended Data Types Can Be Cast?	2-32
Supported Data Types	2-33
BLOB	2-33
BOOLEAN	2-34
BYTE	2-35
CHAR(n)	2-36
CHARACTER(n)	2-37
CHARACTER VARYING(m,r)	2-38
CLOB	2-38
DATE	2-39
DATETIME	2-40
DEC.	2-44
DECIMAL	2-44
Distinct Data Type	2-46
DOUBLE PRECISION	2-47
FLOAT(n).	2-47
INT	2-47
INT8	2-47
INTEGER	2-48
INTERVAL	2-48
LIST(e)	2-51
LVARCHAR	2-53
MONEY(p,s).	2-53
MULTISET(e)	2-54
Named Row Type	2-56
NCHAR(n)	2-57
NUMERIC(p,s)	2-58
NVARCHAR(m,r)	2-58
Opaque Data Type.	2-58

REAL	2-59
SERIAL(n)	2-59
SERIAL8(n)	2-61
SET(e)	2-62
SMALLFLOAT	2-64
SMALLINT	2-64
TEXT	2-65
Unnamed Row Type	2-67
VARCHAR(m,r)	2-69
Operator Precedence	2-71

Chapter 3 Environment Variables

Types of Environment Variables	3-3
Where to Set Environment Variables	3-4
Setting Environment Variables at the System Prompt	3-4
Setting Environment Variables in an Environment- Configuration File	3-4
Setting Environment Variables at Login Time	3-6
Manipulating Environment Variables	3-6
Setting Environment Variables	3-6
Viewing Your Current Settings	3-7
Unsetting Environment Variables	3-7
Modifying the Setting of an Environment Variable	3-8
Checking Environment Variables with the chkenv Utility	3-9
Rules of Precedence	3-10
List of Environment Variables	3-10
Environment Variables	3-14
ARC_DEFAULT	3-14
ARC_KEYPAD	3-14
DBANSIWARN	3-15
DBBLOBBUF	3-17
DBCENTURY	3-18
DBDATE	3-21
DBDELIMITER	3-24
DBEDIT	3-24
DBFLTMASK	3-25
DBLANG	3-26
DBMONEY	3-27
DBONPLOAD	3-29
DBPATH	3-29
DBPRINT	3-32
DBREMOTECMD	3-33
DBSPACETEMP	3-34

DBTEMP	3-35
DBTIME	3-36
DBUPSPACE.	3-39
DELIMIDENT	3-40
ENVIGNORE	3-41
FET_BUF_SIZE	3-41
INFORMIXC.	3-42
INFORMIXCONCSMCFG	3-43
INFORMIXCONRETRY	3-43
INFORMIXCONTIME	3-44
INFORMIXDIR	3-45
INFORMIXKEYTAB	3-46
INFORMIXOPCACHE	3-46
INFORMIXSERVER	3-47
INFORMIXSHMBASE	3-48
INFORMIXSQLHOSTS	3-49
INFORMIXSTACKSIZE	3-49
INFORMIXTERM	3-50
INF_ROLE_SEP.	3-51
IFX_AUTOFREE	3-51
IFX_DEFERRED_PREPARE	3-53
NODEFDAC.	3-54
ONCONFIG	3-54
OPTCOMPIND.	3-55
PATH	3-56
PDQPRIORITY	3-56
PLCONFIG	3-58
PSORT_DBTEMP	3-58
PSORT_NPROCS	3-59
TERM	3-61
TERMCAP	3-61
TERMINFO	3-62
THREADLIB.	3-63
Index of Environment Variables	3-63

Appendix A The stores7 Database

Glossary

Index

Introduction

About This Manual	3
Organization of This Manual	4
Types of Users	5
Software Dependencies	5
Assumptions About Your Locale	5
Demonstration Database	6
Major Features	6
Documentation Conventions	8
Typographical Conventions	8
Icon Conventions	9
Comment Icons	9
Feature Icons	10
Compliance Icons	10
Command-Line Conventions	10
How to Read a Command-Line Diagram	12
Sample-Code Conventions	13
Additional Documentation	14
On-Line Manuals	14
Printed Manuals	14
Error Message Files	15
Documentation Notes, Release Notes, Machine Notes	15
Compliance with Industry Standards	16
Informix Welcomes Your Comments	16

Read this introduction for an overview of the information provided in this manual and for an understanding of the documentation conventions used.

About This Manual

The *Informix Guide to SQL: Reference* manual describes the Informix system catalog tables, common environment variables that you might need to set, and the data types Universal Server supports. This manual also includes information on how to design and use ANSI-compliant databases, a description of the **stores7** demonstration database, and a glossary of database terminology.

This manual is part of a series of manuals that discuss the Informix implementation of SQL. This volume and the [Informix Guide to SQL: Syntax](#) are references that you can use on a daily basis after you read the [Informix Guide to SQL: Tutorial](#).

Important: *This manual does not cover the product called INFORMIX-SQL or any other Informix application development tool.*



Organization of This Manual

This manual includes the following chapters:

- This Introduction provides an overview of the manual and describes the documentation conventions used.
- [Chapter 1, “System Catalog,”](#) provides details of the Informix system catalog, which is a collection of system catalog tables that describe the structure of an INFORMIX-Universal Server database. The chapter explains how to access and update statistics in the system catalog, shows the system catalog structure, and lists the name and data type for each column in each table. This chapter also includes information about Information Schema Views.
- [Chapter 2, “Data Types,”](#) defines the column data types supported by Informix products, tells how to convert between different data types, and describes how to use specific values in arithmetic and relational expressions.
- [Chapter 3, “Environment Variables,”](#) describes the various environment variables that you can or should set to properly use your Informix products. These variables identify your terminal, specify the location of your software, and define other parameters of your product environment.
- [Appendix A](#) describes the structure and contents of the **stores7** demonstration database that is installed with the Informix database server products. It includes a map of the nine tables in the database, illustrates the columns on which they are joined, and displays the data in them.
- A [Glossary](#) of object-relational database terms follows the chapters.
- The Index is a combined index for the manuals in the SQL series. Each page reference in the index ends with a code that identifies the manual in which the page appears. The same index also appears in the [Informix Guide to SQL: Syntax](#) and the [Informix Guide to SQL: Tutorial](#).

Types of Users

This manual is written for SQL users, database administrators, and SQL developers who use Informix products and SQL on a regular basis.

Software Dependencies

This manual assumes that you are using the following Informix software:

- INFORMIX-Universal Server, Version 9.1

The database server must be installed either on your computer or on another computer to which your computer is connected over a network.

In this manual, all instances of Universal Server refer to INFORMIX-Universal Server.

- An Informix SQL application programming interface (API), such as INFORMIX-ESQL/C, Version 9.1, or the DB-Access database access utility, which is shipped as part of your database server.

The SQL API or DB-Access enables you to compose queries, send them to the database server, and view the results that the database server returns.

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 GLS (Global Language Support) locale.

This manual assumes that you use the default locale, **en_us.8859-1**. This locale supports U.S. English format conventions for dates, times, and currency. In addition, this locale supports the ISO 8859-1 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(s). For instructions on how to specify a nondefault locale, additional syntax, and other considerations related to GLS locales, see the [Guide to GLS Functionality](#).

Demonstration Database

The DB-Access utility, which is provided with your Informix database server products, includes a demonstration database called **stores7** that contains information about a fictitious wholesale sporting-goods distributor. Sample command files are also included.

Many examples in Informix manuals are based on the **stores7** demonstration database. The **stores7** database is described in detail and its contents are listed in [Appendix A](#).

The script that you use to install the demonstration database is called **dbaccessdemo7** and is located in the **\$INFORMIXDIR/bin** directory. For a complete explanation of how to create and populate the demonstration database on your database server, refer to the [DB-Access User Manual](#).

Major Features

The following SQL features are new with Universal Server, Version 9.1.

ALLOCATE COLLECTION	DROP TYPE
ALLOCATE ROW	EXECUTE FUNCTION
CREATE CAST	SET AUTOFREE
CREATE DISTINCT TYPE	SET DEFERRED_PREPARE
CREATE FUNCTION	Argument
CREATE FUNCTION FROM	Collection Derived Table
CREATE OPAQUE TYPE	External Routine Reference
CREATE OPCLASS	Function Name
CREATE ROUTINE FROM	Literal Collection
CREATE ROW TYPE	Literal Row
DEALLOCATE COLLECTION	Quoted Pathname
DEALLOCATE ROW	Return Clause
DROP CAST	Routine Modifier
DROP FUNCTION	Routine Parameter List
DROP OPCLASS	Specific Name
DROP ROUTINE	Statement Block
DROP ROW TYPE	

The following SQL features are enhanced for use with Universal Server, Version 9.1.

ALLOCATE DESCRIPTOR	FLUSH
ALTER FRAGMENT	FREE
ALTER INDEX	GET DESCRIPTOR
ALTER TABLE	GET DIAGNOSTICS
CREATE INDEX	GRANT
CREATE PROCEDURE	INFO
CREATE PROCEDURE FROM	INSERT
CREATE SCHEMA	OPEN
CREATE SYNONYM	PREPARE
CREATE TABLE	PUT
CREATE VIEW	REVOKE
DEALLOCATE DESCRIPTOR	SELECT
DECLARE	SET DESCRIPTOR
DELETE	SET EXPLAIN
DESCRIBE	UPDATE
DROP INDEX	UPDATE STATISTICS
DROP PROCEDURE	Condition
DROP TABLE	Data Type
EXECUTE	Expression
EXECUTE PROCEDURE	Procedure Name
FETCH	Quoted String

The Introduction to each Version 9.1 product manual contains a list of major features for that product. The Introduction to each manual in the Version 9.1 *Informix Guide to SQL* series contains a list of new SQL features.

Major features for Version 9.1 Informix products also appear in release notes.

Documentation Conventions

This section describes the conventions that this manual uses. These conventions make it easier to gather information from this and other Informix manuals.

The following conventions are covered:

- Typographical conventions
- Icon conventions
- Command-line conventions
- Sample-code conventions

Typographical Conventions

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

Convention	Meaning
KEYWORD	All keywords appear in uppercase letters in a serif font.
<i>italics</i>	Within text, new terms and emphasized words appear in italics. Within syntax diagrams, values that you are to specify appear in italics.
boldface	Identifiers (names of classes, objects, constants, events, functions, program variables, forms, labels, and reports), environment variables, database names, filenames, table names, column names, icons, menu items, command names, and other similar terms appear in boldface.
<code>monospace</code>	Information that the product displays and information that you enter appear in a monospace typeface.

(1 of 2)



Convention	Meaning
KEYSTROKE	Keys that you are to press appear in uppercase letters in a sans serif font.
◆	This symbol indicates the end of feature-, product-, platform-, or compliance-specific information.

(2 of 2)




***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 warnings, important notes, or tips. This information is always displayed in italics.

Icon	Description
	The <i>warning</i> icon identifies vital instructions, cautions, or critical information.
	The <i>important</i> icon identifies significant information about the feature or operation that is being described.
	The <i>tip</i> icon identifies additional details or shortcuts for the functionality that is being described.

Feature Icons


Feature icons identify paragraphs that contain feature-specific information.

Icon	Description
	Identifies information that relates to the Informix Global Language Support (GLS) feature.

These icons can apply to a row in a table, one or more paragraphs, or an entire section. A ♦ symbol indicates the end of the feature-specific information.

Compliance Icons

Compliance icons indicate paragraphs that provide guidelines for complying with a standard.

Icon	Description
	Identifies information that is specific to an ANSI-compliant database.

These icons can apply to a row in a table, one or more paragraphs, or an entire section. A ♦ symbol indicates the end of the compliance information.

Command-Line Conventions

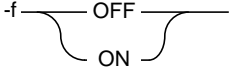
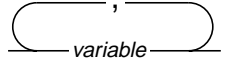
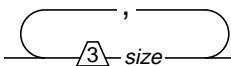
This section defines and illustrates the format of commands that are available in Informix products. These commands have their own conventions, which might include alternative forms of a command, required and optional parts of the command, and so forth.

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.
<i>variable</i>	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.
<div>Privileges p. 5-17</div> <div>Privileges</div>	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)

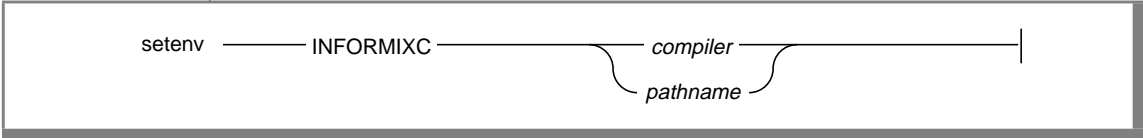
Element	Description
	A branch below the main path indicates an optional path. (Any term on the main path is required, unless a branch can circumvent it.)
	A loop indicates a path that you can repeat. Punctuation along the top of the loop indicates the separator symbol for list items.
	A gate ($\sqrt{3}$) 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. Here you can specify <i>size</i> 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. Then follow the diagram to the right, including the elements that you want. The elements in the diagram are case sensitive.

Figure 1 diagrams the following steps:

1. Type the word `setenv`.
2. Type the word `INFORMIXC`.
3. Supply either a compiler name or `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 stores7
...

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 use the Query-language option of DB-Access, you must delimit multiple statements with semicolons. If you use 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 how to use 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 files
- Documentation notes, release notes, and machine notes

On-Line Manuals

A 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 either the installation guide for your product or the installation insert that accompanies the documentation CD.

The documentation set that is provided on the CD describes Universal Server, its implementation of SQL, and its associated application-programming interfaces. For an overview of the manuals in the Universal Server documentation set, see [*Getting Started with INFORMIX-Universal Server*](#).

Printed Manuals

The Universal Server documentation set describes Universal Server, its implementation of SQL, and its associated application-programming interfaces. For an overview of the manuals in the Universal Server documentation set, see [*Getting Started with INFORMIX-Universal Server*](#).

To order printed manuals, call 1-800-331-1763 or send email to moreinfo@informix.com.

Please provide the following information:

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

Error Message Files

Informix software products provide ASCII files that contain all the Informix error messages and their corrective actions. To read the error messages in the ASCII file, Informix provides scripts that let you display error messages on the screen (**finderr**) or print formatted error messages (**rofferr**). For a detailed description of these scripts, see the Introduction to the [Informix Error Messages](#) manual.

Documentation Notes, Release Notes, Machine Notes

In addition to printed documentation, the following on-line files, located in the `$INFORMIXDIR/release/en_us/0333` directory, supplement the information in this manual.

On-Line File	Purpose
SQLRDOC_9.1	The documentation-notes file describes features that are not covered in this manual or that have been modified since publication.
SERVERS_9.1	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.
IUNIVERSAL_9.1	The machine-notes file describes any special actions that are required to configure and use Informix products on your computer. Machine notes are named for the product described.

Please examine these files because they contain vital information about application and performance issues.

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, on INFORMIX-Universal Server. In addition, many features of Universal Server comply with the SQL-92 Intermediate and Full Level and X/Open SQL CAE (common applications environment) standards.

Informix Welcomes Your Comments

Please tell us what you like or dislike about our manuals. To help us with future versions of our manuals, we want to know about 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

Write to us at the following address:

Informix Software, Inc.
SCT Technical Publications Department
4100 Bohannon Drive
Menlo Park, CA 94025

If you prefer to send email, our address is:

`doc@informix.com`

Or send a facsimile to the Informix Technical Publications Department at:

415-926-6571

We appreciate your feedback.

System Catalog

Objects Tracked by the System Catalog Tables	1-3
Using the System Catalog	1-4
Accessing the System Catalog	1-10
Updating System Catalog Data	1-10
Structure of the System Catalog	1-11
SYSAMS	1-12
SYSATTRTYPES	1-17
SYSBLOBS	1-18
SYSCASTS	1-19
SYSCHECKS	1-20
SYSCOLATTRIBS	1-20
SYSCOLAUTH	1-22
SYSCOLDEPEND	1-23
SYSCOLUMNS	1-23
Storing Column Data Type	1-26
Storing Column Length	1-27
Storing Maximum and Minimum Values	1-29
SYSCONSTRAINTS	1-29
SYSDEFAULTS	1-30
SYSDEPEND	1-32
SYSDISTRIB	1-32
SYSERRORS	1-33
SYSFRAGAUTH	1-34
SYSFRAGMENTS	1-36
SYSINDICES	1-37
SYSINHERIS	1-39
SYSLANGAUTH	1-40
SYSLOGMAP	1-40
SYSOBJSTATE	1-40
SYSOPCLASSES	1-42

SYSOPCLSTR	1-43
SYSROCAUTH	1-44
SYSROCBODY	1-45
SYSROCEDURES.	1-46
SYSROPLAN	1-48
SYSREFERENCES	1-49
SYSROLEAUTH	1-50
SYSROUTINELANGS	1-51
SYSROSYNONYMS	1-51
SYSROSYNTABLE	1-52
SYSROTABAUTH	1-53
SYSROTABLES	1-54
SYSROTRACECLASSES	1-56
SYSROTRACEMSGS	1-57
SYSROTRIGBODY	1-58
SYSROTRIGGERS	1-59
SYSROUSERS	1-60
SYSROVIEWS	1-61
SYSROVIOLATIONS	1-61
SYSROXTDESC	1-62
SYSROXTDTYPEAUTH	1-63
SYSROXTDTYPES	1-64
Information Schema	1-65
Generating the Information Schema Views	1-66
Accessing the Information Schema Views	1-67
Structure of the Information Schema Views	1-67
TABLES	1-67
COLUMNS	1-68
SQL_LANGUAGES	1-70
SERVER_INFO.	1-70

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 covers the following topics:

- How to access tables in the system catalog
- How to update statistics in the system catalog
- The structure, including the name and data type of each column, of the tables that make up the system catalog
- Information Schema views that are created from system catalog information

Objects Tracked by the System Catalog Tables

The system catalog tables track the following objects:

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

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

Universal Server generates system catalog tables automatically when you create a database, and you can query them as you would query any other table in the database. The data for a newly created database and the system catalog tables for that database reside in the same dbspace.

Using the System Catalog

The database server accesses the system catalog constantly. You can access the information in the system catalog tables as well. 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 CREATE SCHEMA block in Figure 1-1 adds the **customer** table, with its respective indexes and privileges, to the **stores7** database. This block also adds a **view**, **california**, that restricts the information accessible in the **customer** table to only the first and last names of the customer, the company name, and the phone number of all customers who reside in California.

```
CREATE SCHEMA AUTHORIZATION mary1
CREATE TABLE customer
  (customer_num SERIAL(101), fname CHAR(15), lname CHAR(15), company CHAR(20),
   address1 CHAR(20), address2 CHAR(20), city CHAR(15), state CHAR(2),
   zipcode CHAR(5), phone CHAR(18))
GRANT ALTER, ALL ON customer TO cath1 WITH GRANT OPTION AS mary1
GRANT SELECT ON CUSTOMER TO public
GRANT UPDATE (fname, lname, phone) ON customer TO nhowe
CREATE VIEW california AS
  SELECT fname, lname, company, phone FROM customer WHERE state = 'CA'
CREATE UNIQUE INDEX c_num_ix ON customer (customer_num)
CREATE INDEX state_ix ON customer (state);
```

Figure 1-1
*Sample CREATE
SCHEMA block*

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**
- **syscolauth**
- **sysindices**

The following two new rows of information are added to the **systables** system catalog table after the CREATE SCHEMA block shown in [Figure 1-1 on page 1-4](#) is run.

```
tabnamecustomer
ownermaryl
partnum16778361
tabid101
rowsize134
ncols10
nindexes2
nrows0
created04/26/1994
version 1
tabtypeT
locklevelP
npused0
fextsize16
nextsize16
flags0
site
dbname

tabnamecalifornia
ownermaryl
partnum0
tabid102
rowsize134
ncols4
nindexes0
nrows0
created04/26/1994
version 0
tabtypeV
locklevelB
npused0
fextsize0
nextsize0
flags0
site
dbname
```

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.

```
colnametabidcolnocoltypecollengthcolmincolmax
customer_num10112624
fname1012 0 15
lname1013 0 15
company10140 20
address11015020
address21016020
city1017 0 15
state1018 0 2
zipcode10190 5
phone10110 0 18

fname1021 0 15
lname1022 0 15
company10230 20
phone1024 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 shown in the following example are added to the **sysviews** system catalog table. These rows correspond to the CREATE VIEW portion of the CREATE SCHEMA block.

```
tabidseq viewtext
102 0 create view 'maryl'.california (customer_num, fname, lname, company
102 1 ,address1, address2, city, state, zipcode, phone) as 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 **sysstabauth** system catalog table. These rows correspond to the user privileges granted on **customer** and **california** tables, as the following example shows.

```
grantorgranteetabidtabauth
marylpublic101su-idx--
marylcath1101 SU-IDXAR
marylnhowe101 --*-----
maryl102 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—s (select), u (update), * (column-level privilege), i (insert), d (delete), x (index), a (alter), 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.

```
grantorgranteetabidcolnocolauth
marylnhowe101 2 -u-
marylnhowe101 3 -u-
marylnhowe101 10 -u-
```


The **colauth** column specifies the column-level privileges that are granted on the **customer** table. This column uses a 3-byte pattern—s (select), u (update), 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 **sysindices** system catalog table. 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	0
clustered		
part11	8	
part20	0	
part30	0	
part40	0	
part50	0	
part60	0	
part70	0	
part80	0	
part90	0	
part100	0	
part110	0	
part120	0	
part130	0	
part140	0	
part150	0	
part160	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 the 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
```



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 using the ALTER TABLE statement to modify the size of the next extent of system catalog tables. However, in certain cases it is valid to add entries to the system catalog tables, for instance, in the case of the **syserrors** system catalog table and the **sysracemsgs** system catalog table, where a developer of DataBlade modules can add message entries that appear in these catalog tables.

Updating System Catalog Data

The fact that the optimizer in Informix database servers determines the most efficient strategy for executing SQL queries 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 table to determine the best query strategy.

By using the UPDATE STATISTICS statement to update the system catalog table, 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 table. For example, if you delete rows in a table using 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 **sysindices** 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 tables. For more information on the UPDATE STATISTICS statement, see Chapter 1 of the [Informix Guide to SQL: Syntax](#).

Structure of the System Catalog

The following system catalog tables describe the structure a database on an Informix database server.

sysams	sysfragments	syssynonyms
sysattrtypes	sysindices	sys syntable
sysblobs	sysinherits	sys tabauth
syscasts	syslangauth	systables
syschecks	syslogmap	sys traceclasses
syscolattribs	sysobjstate	sys tracemsgs
syscolauth	sysopclasses	sys trigbody
syscoldepend	sysopclstr	sys triggers
syscolumns	sysprocauth	sys users
sysconstraints	sysprocbody	sys views
sysdefaults	sysprocedures	sys violations
sysdepend	sysprocplan	sys xtddesc
sysdistrib	sysreferences	sys xtdtypeauth
syserrors	sysroleauth	sys xtdtypes
sysfragauth	sysroutinelangs	

Do not confuse the system catalog tables of a database with the tables in the **sysmaster** database of Universal Server. The system catalog tables give information regarding a particular database on any Informix database server, while the **sysmaster** tables contain information about an entire Informix database server, which might manage many individual databases. The information in the **sysmaster** tables is primarily useful for Universal Server DBAs. For more information about the **sysmaster** tables, see the [INFORMIX-Universal Server Administrator's Guide](#).

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 (including the default locale), all character information in the system catalog tables is stored in CHAR columns.

This manual assumes that the locale has code-set collation and lists character columns with the CHAR data type. If your locale has localized collation, these character columns are NCHAR. For more information on collation orders, see Chapter 1 of the [Guide to GLS Functionality](#). For information about NCHAR and CHAR data types, see Chapter 3 of the [Guide to GLS Functionality](#) and [Chapter 2](#) of this guide. ♦

SYSAMS

The **sysams** system catalog table shows information needed to use built-in access methods as well as those created by the Create Access Method SQL statement described in the *Virtual-Table Interface Programmer's Manual*. The **sysams** table shows the following columns.

Column Name	Type	Explanation
am_name	CHAR(18)	Name of the access method
am_owner	CHAR(8)	Owner of the access method
am_id	INTEGER	Unique identifier for the access method. This value corresponds to the <i>am_id</i> in the systables system catalog table and to the <i>am_id</i> in the sysindices and sysopclasses system catalog tables.

(1 of 5)

Column Name	Type	Explanation
am_type	CHAR(1)	Type of access method: 'P' = Primary 'S' = Secondary
am_sptype	CHAR(3)	Type of space in which the access method can live: 'D' = dbspace 'X' = extspace 'S' = sbpace (smart-large-object space) 'A' = any space
am_defopclass	INTEGER	Default-operator class identifier. This is the <i>opclassid</i> 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 5)

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	SMALL-FLOAT	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 <i>am_scancost</i> function
am_create	INTEGER	The routine specified for the AM_CREATE purpose for this access method This is the <i>procid</i> for the routine in the sysprocedures system catalog table
am_drop	INTEGER	The routine specified for the AM_DROP purpose for this access method This is the <i>procid</i> listed for the routine in the sysprocedures system catalog table

(3 of 5)

Column Name	Type	Explanation
am_open	INTEGER	<p>The routine specified for the AM_OPEN purpose for this access method</p> <p>This is the <i>procid</i> listed for the routine in the sysprocedures system catalog table</p>
am_close	INTEGER	<p>The routine specified for the AM_CLOSE purpose for this access method</p> <p>This is the <i>procid</i> listed for the routine in the sysprocedures system catalog table</p>
am_insert	INTEGER	<p>The routine specified for the AM_INSERT purpose for this access method</p> <p>This is the <i>procid</i> listed for the routine in the sysprocedures system catalog table</p>
am_delete	INTEGER	<p>The routine specified for the AM_DELETE purpose for this access method</p> <p>This is the <i>procid</i> listed for the routine in the sysprocedures system catalog table</p>
am_update	INTEGER	<p>The routine specified for the AM_UPDATE purpose for this access method</p> <p>This is the <i>procid</i> listed for the routine in the sysprocedures system catalog table</p>
am_stats	INTEGER	<p>The routine specified for the AM_STATS purpose for this access method</p> <p>This is the <i>procid</i> listed for the routine in the sysprocedures system catalog table</p>
am_scancost	INTEGER	<p>The routine specified for the AM_SCANCOST purpose for this access method</p> <p>This is the <i>procid</i> listed for the routine in the sysprocedures system catalog table</p>
am_check	INTEGER	<p>The routine specified for the AM_CHECK purpose for this access method</p> <p>This is the <i>procid</i> listed for the routine in the sysprocedures system catalog table</p>

(4 of 5)

Column Name	Type	Explanation
am_beginscan	INTEGER	The routine specified for the AM_BEGINSCAN purpose for this access method This is the <i>procid</i> 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 This is the <i>procid</i> 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 This is the <i>procid</i> listed for the routine in the sysprocedures system catalog table
am_getnext	INTEGER	The routine specified for the AM_GETNEXT purpose for this access method This is the <i>procid</i> listed for the routine in the sysprocedures system catalog table
am_getbyid	INTEGER	The routine specified for the AM_GETBYID purpose for this access method This is the <i>procid</i> listed for the routine in the sysprocedures system catalog tables
am_build	Reserved for future use	

(5 of 5)

For information regarding access method functions, see the *Virtual-Table Interface Programmer's Manual*.

The composite index for the **am_name** and **am_owner** columns in this table allows only unique values.

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 shows the following columns.

Column Name	Type	Explanation
extended_id	INTEGER	Identifier for extended data types, same as in sysxdtypes
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	CHAR(18)	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. See the coltype column entries in the syscolumns system catalog table for a complete list of values associated with different data types.
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 sysxdtypes 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 **sysblobs** system catalog table specifies the storage location of a simple large object, namely TEXT and BYTE. The name, **sysblobs**, is used for historical reasons, even though the table describes simple large objects. It contains one row for each TEXT or BYTE column in a table. The **sysblobs** system catalog table shows the following columns.

Column Name	Type	Explanation
spacename	CHAR(18)	Blobspace, dbspace, or family name (for optical storage)
type	CHAR(1)	Media type: 'M' = Magnetic 'O' = Optical
tabid	INTEGER	Table identifier
colno	SMALLINT	Column number

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

For information about location and size of chunks of blobspaces, dbspaces, and sbspaces locations of TEXT, BYTE, BLOB, and CLOB columns, see the **syschunks** sysmaster table in [INFORMIX-Universal Server Administrator's Guide](#).

SYSCASTS

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

Column Name	Type	Explanation
owner	CHAR(8)	Owner of cast (user informix for system-defined 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	CHAR(18)	Function or procedure used to implement the cast (may 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	CHAR(8)	User name of the owner of the function or procedure named in the routine_name column
class	CHAR	Type of cast: ' I ' = Implicit cast ' S ' = System-defined cast ' X ' = Explicit 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 text and a binary-encoded form of the check constraint, it contains multiple rows for each check constraint. The **syschecks** system catalog table shows the following columns.

Column Name	Type	Explanation
constrid	INTEGER	Constraint identifier
type	CHAR(1)	Form in which the check constraint is stored: 'B' = Binary encoded 'T' = Text
seqno	SMALLINT	Line number of the check constraint
checktext	CHAR(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.

SYSCOLATTRIBS

The **syscolattrs** 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 shows the following columns.

Column Name	Type	Explanation
tabid	INTEGER	Table identifier
colno	SMALLINT	Column number
extentsize	INTEGER	Pages in smart-large-object extent, either expressed in kbytes or in multiples of sbspace page size
flags	INTEGER	This is 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 large-object column by a user.
		LO_NOKEEP_LASTACCESS_TIME No record is kept of the most recent access of this large-object column by a user.
	HI_INTEG	Data pages have headers and footers to detect incomplete writes and data corruption.

(1 of 2)

Column Name	Type	Explanation
		MODERATE_INTEG
		Data pages do not have headers and footers.
flags1	INTEGER	Reserved for future use
sbspace	CHAR(18)	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 shows the following columns.

Column Name	Type	Explanation
grantor	CHAR(8)	Grantor of privilege
grantee	CHAR(8)	Grantee (receiver) of privilege
tabid	INTEGER	Table identifier
colno	SMALLINT	Column number
colauth	CHAR(3)	3-byte pattern that specifies column privileges: 's' or 'S' = Select 'u' or 'U' = Update 'r' or 'R' = References

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

A 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 shows 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. The **syscolumns** system catalog table shows the following columns.

Column Name	Type	Explanation
colname	CHAR(18)	Column name
tabid	INTEGER	Table identifier
colno	SMALLINT	Column number sequentially assigned by the system (from left to right within each table)

(1 of 3)

Column Name	Type	Explanation
coltype	SMALLINT	Code (identifier) for column data type: <div><div>0 = CHAR14 = INTERVAL</div><div>1 = SMALLINT15 = NCHAR</div><div>2 = INTEGER16 = NVARCHAR</div><div>3 = FLOAT17 = INT8</div><div>4 = SMALLFLOAT18 = SERIAL8*</div><div>5 = DECIMAL19 = SET</div><div>6 = SERIAL*20 = MULTISSET</div><div>7 = DATE21 = LIST</div><div>8 = MONEY22 = ROW</div><div>9 = NULL23 = COLLECTION</div><div>10 = DATETIME24 = ROWREF</div><div>11 = BYTE40 = Variable-length opaque type</div><div>12 = TEXT41 = Fixed-length opaque type</div><div>13 = VARCHAR4118 = Named row type</div></div>
collength	SMALLINT	Column length (in bytes)
colmin	INTEGER	Second-smallest value
colmax	INTEGER	Second-largest value
minlen	INTEGER	Minimum column length (in bytes)

(2 of 3)

Column Name	Type	Explanation
maxlen	INTEGER	Maximum column length (in bytes)
extended_id	INTEGER	Type identifier, from the sysxdtypes 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.

(3 of 3)

The **coltype** ⁴¹¹⁸ for named row types is the decimal representation of the hexadecimal value 0 x 1016, which is the same as the hexadecimal **coltype** value for an unnamed row type (0 x 016), with the named-row type bit set.

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

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.

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

BLOB	BOOLEAN
CLOB	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 **sysxdtypes** 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:

$$(\text{precision} * 256) + \text{scale}$$

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:

$$\text{collength} = (\text{min_space} * 256) + \text{max_size}$$

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

$$\text{collength} + 65536 = (\text{min_space} * 256) + \text{max_size}$$

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 [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 shown in the following table.

Field Qualifier	Value
YEAR	0
MONTH	2
DAY	4
HOURL	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.

SYSCONSTRAINTS

The **sysconstraints** system catalog table lists the constraints placed on the columns in each database table. An entry is also placed in the **sysindices** 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 **sysindices** system catalog table. Because indexes can be shared, more than one constraint can be associated with an index.

The **sysconstraints** system catalog table shows the following columns.:

Column Name	Type	Explanation
constrid	SERIAL	System-assigned sequential identifier
constrname	CHAR(18)	Constraint name
owner	CHAR(8)	User name of owner
tabid	INTEGER	Table identifier

(1 of 2)

Column Name	Type	Explanation
constrtype	CHAR(1)	Constraint type: 'C' = Check constraint 'P' = Primary key 'R' = Referential 'U' = Unique 'N' = Not null
idxname	CHAR(18)	Index name

(2 of 2)

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 shows the following columns.

Column Name	Type	Explanation
tabid	INTEGER	Table identifier
colno	SMALLINT	Column identifier

(1 of 2)

Column Name	Type	Explanation
type	CHAR(1)	Default type: 'L' = Literal default 'U' = User 'C' = Current 'N' = Null 'T' = Today 'S' = Dbservername
default	CHAR(256)	If default type = L, the literal default value
class	CHAR(1)	Type of column: 'T' = table 't' = row type

(2 of 2)

If a literal is specified for the default value, it is stored in the **default** column as text. If the literal value is not of type CHAR, 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. The two parts are separated by a space.

If the data type of the column is not CHAR or VARCHAR, 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 shows the following columns.

Column Name	Type	Explanation
btabid	INTEGER	Table identifier of base table or view
btype	CHAR(1)	Base object type: 'T' = Table 'V' = View
dtabid	INTEGER	Table identifier of dependent table
dtype	CHAR(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 plans of SQL SELECT statements. Information is stored in the **sysdistrib** table when an UPDATE STATISTICS statement with mode MEDIUM or HIGH is run for a table.

The **sysdistrib** system catalog table shows 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

(1 of 2)

Column Name	Type	Explanation
constructed	DATE	Date when the data distribution was created
mode	CHAR(1)	Optimization level: ' L ' = Low ' M ' = Medium ' H ' = High
resolution	FLOAT	Specified in the UPDATE STATISTICS statement
confidence	FLOAT	Specified in the UPDATE STATISTICS statement
encdat	CHAR(256)	ASCII-encoded histogram in fixed-length character field; accessible only to user informix .

(2 of 2)

You can select any column from **sysdistrib** except **encdat**. User **informix** can select the **encdat** column.

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 shows the following columns.

Column Name	Type	Explanation
sqlstate	CHAR(5)	SQLSTATE value associated with the error. For more information about SQLSTATE values and their meanings, see the GET DIAGNOSTICS statement in the Informix Guide to SQL: Syntax .
locale	CHAR(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	VARCHAR(255)	Message text

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

SYSFRAGAUTH

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

Column Name	Type	Explanation
grantor	CHAR(8)	Grantor of privilege
grantee	CHAR(8)	Grantee (receiver) of privilege
tabid	INTEGER	Table identifier of the table that contains the fragment named in the fragment column
fragment	CHAR(18)	Name of dbspace where fragment is stored. Identifies the fragment on which privileges are granted.

(1 of 2)

Column Name	Type	Explanation
fragauth	CHAR(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: <div> 'u' or 'U' = Update 'i' or 'I' = Insert 'd' or 'D' = Delete </div>

(2 of 2)

If a code in the **fragauth** column is lowercase, the grantee cannot grant the privilege to other users. If a code in the **fragauth** column is uppercase, 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	dbbsp1	-ui---
dba	jane	101	dbbsp3	--i---
dba	mary	101	dbbsp4	--id--
dba	ted	101	dbbsp2	-UID--

SYSFRAGMENTS

The **sysfragments** table stores fragmentation information for tables and indexes. One row exists for each table or index fragment. The **sysfragments** table shows the following columns.

Column Name	Type	Explanation
fragtype	CHAR(1)	Fragment type: 'T' = Index 'T' = Table
tabid	INTEGER	Table identifier
indexname	CHAR(18)	Index identifier
colno	SMALLINT	Blob column identifier
partn	INTEGER	Physical location identifier
strategy	CHAR(1)	Distribution scheme type: 'R' = Round robin 'E' = Expression 'T' = Table-based
location	Reserved for future use; shows L for local	
servername	Reserved for future use	
evalpos	INTEGER	Position of fragment in the fragmentation list
exprtext	TEXT	Expression that was entered
exprbin	BYTE	Binary version of expression
exprarr	BYTE	Range partitioning data used to optimize expression in range-expression fragmentation strategy
flags	INTEGER	Internally used
dbspace	CHAR(18)	Dbspacename for fragment

(1 of 2)

Column Name	Type	Explanation
levels	SMALLINT	Number of B+ tree index levels
npused	INTEGER	For table fragmentation strategy this is the number of data pages; for index fragmentation strategy this is the number of leaf pages
nrows	INTEGER	For tables this is the number of rows in the fragment; for indexes this is the number of unique keys
clust	INTEGER	Degree of index clustering; smaller numbers correspond to greater clustering

(2 of 2)

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

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

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 shows the following columns.

Column Name	Type	Explanation
idxname	CHAR(18)	Index name
owner	CHAR(8)	Owner of index (user informix for system catalog tables and user name for database tables)
tabid	INTEGER	Table identifier
idxtype	CHAR(1)	Index type: 'U' = Unique 'D' = Duplicates

(1 of 2)

Column Name	Type	Explanation
clustered	CHAR(1)	Clustered or nonclustered index ('C' = Clustered)
levels	SMALLINT	Number of tree levels
leaves	INTEGER	Number of leaves
nunique	INTEGER	Number of unique keys in the first column
clust	INTEGER	Degree of clustering: smaller numbers correspond to greater clustering. The maximum value is the number of rows in the table, and the minimum value is the number of data pages in the table. This column is blank until the UPDATE STATISTICS statement is run on the table.
indexkeys	INDEXKEYARRAY	The indexkeys 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 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. This is the value of the <i>am_id</i> 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, **idxtab** and **idxname**, used with INFORMIX-OnLine Dynamic Server, are retained in Universal Server and used to index **sysindices**, using the same keys.

SYSINHERIS

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 shows 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	CHAR(1)	Inheritance class: 't' = named row type 'T' = table

SYSLANGAUTH

The **syslangauth** system catalog table contains the authorization information on computer languages that are used to write user routines. This table contains no values in this release.

Column Name	Type	Explanation
grantor		Reserved for future use
grantee		Reserved for future use
langid		Reserved for future use
langauth		Reserved for future use

SYSLOGMAP

This 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

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 created the object. Indexes that the database server created on the system catalog tables are not listed in the **sysobjstate** table because their object mode cannot be changed.

The **sysobjstate** system catalog table shows the following columns.

Column Name	Type	Explanation
objtype	CHAR(1)	The type of database object. This column has one of the following codes: 'C' = Constraint 'I' = Index 'T' = Trigger
owner	CHAR(8)	The owner of the database object
name	CHAR(18)	The name of the database object
tabid	INTEGER	Table identifier of the table on which the database object is defined
state	CHAR(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.

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 shows the following columns.

Column Name	Type	Explanation
opclassname	CHAR(18)	Name of the operator class
owner	CHAR(8)	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

There are two indexes on **sysopclasses**. There is a composite index on **opclassname** and **owner** columns and an index on **opclassid** column. Both indexes allow only unique values.

SYSOPCLSTR

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

Column Name	Type	Explanation
owner	CHAR(8)	Owner of the cluster
clstrname	CHAR(18)	Name of the cluster
clstrsize	INTEGER	Size of the cluster
tabid	INTEGER	Table identifier
blobcol1	SMALLINT	Blob column number 1
.	.	.
.	.	.
.	.	.
blobcol16	SMALLINT	Blob column number 16
clstrkey1	SMALLINT	Cluster key number 1
.	.	.
.	.	.
.	.	.
clstrkey16	SMALLINT	Cluster key number 16

A composite index for both the **clstrname** and **owner** columns allows only unique values. The **tabid** column allows duplicate values.

SYSPROCAUTH

The **sysprocauth** table describes the privileges granted on a routine. It contains one row for each set of privileges that are granted. The name, **sysprocauth**, is used for historical reasons, even though the table describes functions as well as procedures.

The **sysprocauth** system catalog table shows the following columns.

Column Name	Type	Explanation
grantor	CHAR(8)	Grantor of procedure
grantee	CHAR(8)	Grantee (receiver) of procedure
procid	INTEGER	Procedure identifier
procauth	CHAR(1)	Type of procedure permission granted: 'e' = Execute permission on procedure '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 routine in the database. Because the **sysprocbody** system catalog table stores the text of the routine, each routine can have multiple rows. The name, **sysprocbody**, is used for historical reasons, even though the table describes functions as well as procedures.

The **sysprocbody** system catalog table shows the following columns.:

Column Name	Type	Explanation
procid	INTEGER	Procedure identifier
datakey	CHAR(1)	Data-descriptor type: 'D' = User document text 'T' = Actual procedure source 'R' = Return-value type list 'S' = Procedure symbol table 'L' = Constant-procedure data string (that is, literal numbers or quoted strings) 'P' = Interpreter instruction code
seqno	INTEGER	Line number of the procedure
data	CHAR(256)	Actual text of the procedure

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: the encoded return values list, the encoded symbol table, constant data, compiled code for the procedure, or the text of the procedure 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. The name, **sysprocedures**, is used for historical reasons, even though the table describes functions as well as procedures.

The **sysprocedures** system catalog table shows the following columns.

Column Name	Type	Explanation
procname	CHAR(18)	Name of routine
owner	CHAR(8)	Name of owner
procid	SERIAL	Routine identifier
mode	CHAR(1)	Mode type: 'D' = DBA 'O' = Owner 'P' = Protected
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 the instruction code for the routine
numargs	INTEGER	Number of arguments in the procedure
isproc	CHAR(1)	Whether the routine is a procedure or a function 't' = procedure 'f' = function
specificname	VARCHAR(128)	The specific name for the routine
externalname	VARCHAR(255)	Location of the external routine. This item is language-specific in content and format.

(1 of 3)

Column Name	Type	Explanation
paramstyle	CHAR(1)	Parameter style 'I' = Informix
langid	INTEGER	Language identifier (from the sysroutine-langs system catalog table.)
paramtypes	RTNPARAMTYPES	Data types of the parameters. <i>rtnparamtypes</i> indicates a system-defined data type.
variant	BOOLEAN	VARIANT indicator. Whether the routine is VARIANT or not. 't' = is variant 'f' = is not variant
client	Reserved for future use	
handlesnulls	BOOLEAN	Null handling indicator: 't' = handles nulls 'f' = does not handle nulls
iterator	Reserved for future use	
percallcost	Reserved for future use	
commutator	Reserved for future use	
negator	Reserved for future use	
selfunc	Reserved for future use	

(2 of 3)

Column Name	Type	Explanation
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	CHAR(18)	CPU class in which the routine should be executed
stack	INTEGER	stack size in bytes required per invocation
(3 of 3)		

The following three indexes exist on the **sysprocedures** system catalog table. The index on **procname**, **isproc**, **numargs**, and **owner** columns allows duplicate values. The index on columns **specificname** and **owner** also allows duplicate values. The index on the **procid** column allows only unique values.

A database server can create special-purpose protected stored procedures for internal use. The **sysprocedures** table identifies these protected procedures with the letter P in the mode column. You cannot modify or drop protected stored procedures 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 stored routine. Because different parts of a routine plan can be created on different dates, the table can contain multiple rows for each routine. The name, **sysprocplan**, is used for historical reasons, even though the table describes functions as well as procedures.

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

Column Name	Type	Explanation
procid	INTEGER	Routine identifier
planid	INTEGER	Plan identifier
datakey	CHAR(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	CHAR(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** table shows 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	Reserved for future use; displays an R	

(1 of 2)

Column Name	Type	Explanation
delrule	CHAR(1)	Displays cascading delete or restrict rule: 'C ' = Cascading delete 'R' = Restrict (default)
matchtype	Reserved for future use; displays an N	
pendant	Reserved for future use; displays an N	

(2 of 2)

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

SYSROLEAUTH

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

Column Name	Type	Explanation
rolename	CHAR(usersize)	Name of the role
grantee	CHAR(usersize)	Grantee (receiver) of role
is_grantable	CHAR(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.

SYSROUTINELANGS

The **sysroutinelangs** system catalog table lists the languages supported for writing routines.

Column Name	Type	Explanation
langid		Reserved for future use
langname		Reserved for future use
langinitfunc		Reserved for future use

SYSSYNONYMS

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 shows the following columns.

Column Name	Type	Explanation
owner	CHAR(8)	User name of owner
synname	CHAR(18)	Synonym identifier
created	DATE	Date synonym created
tabid	INTEGER	Table identifier

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

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



SYSSYNTABLE

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

Column Name	Type	Explanation
tabid	INTEGER	Table identifier
servername	CHAR(18)	Server name
dbname	CHAR(18)	Database name
owner	CHAR(8)	User name of owner
tablename	CHAR(18)	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 **tablename** 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.

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 shows the following columns.

Column Name	Type	Explanation
grantor	CHAR(8)	Grantor of privilege
grantee	CHAR(8)	Grantee (receiver) of privilege
tabid	INTEGER	Table identifier
tabauth	CHAR(8)	8-byte pattern that specifies table privileges. Upper case letters indicate permission to grantee to grant that privilege to others: 's' or 'S' = Select 'u' or 'U' = Update '*' = Column-level authority 'i' or 'I' = Insert 'd' or 'D' = Delete 'x' or 'X' = Index 'a' or 'A' = Alter 'r' or 'R' = References

If the **tabauth** privilege code is uppercase (for example, S for select), a user who has this privilege also 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. This includes all database tables and the system catalog tables. The **systables** system catalog table shows the following columns.

Column Name	Type	Explanation
tablename	CHAR(18)	Name of table, view, or synonym
owner	CHAR(8)	Owner of table (user informix for system catalog tables and user name for database tables)
partnum	INTEGER	Tblspace identifier (similar to tabid)
tabid	SERIAL	System-assigned sequential ID number (system tables: 1 through 24, user tables: 100 through 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	CHAR(1)	Table type: 'T' = Table 'V' = View 'P' = Private synonym 'P' = Synonym (in an ANSI-compliant database) 'S' = Synonym

Column Name	Type	Explanation
locklevel	CHAR(1)	Lock mode for a table: 'B' = Page 'P' = Page 'R' = Row
npused	INTEGER	Number of data pages in use
feysize	INTEGER	Size of initial extent (in kilobytes)
nextsize	INTEGER	Size of all subsequent extents (in kilobytes)
flags	Reserved for future use	
site	Reserved for future use (used to store database collation and C type information)	
dbname	Reserved for future use	
type_xid	INTEGER	Data type of table, if the table is a typed table
am_id	INTEGER	Access method identifier. A primary access method can be specified when the table is created. A value of null or zero indicates that the built-in storage manager is being used. This is the <i>am_id</i> in the sysams system catalog table.

(2 of 2)

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 first **tabid** numbers up to 99 are reserved for system catalog tables, and the user-created tables receive **tabid** numbers beginning with 100.

The **tabid** column is indexed and must contain unique values. A composite index for the **tablename** and **owner** columns allows only unique values. The **version** column contains an encoded number that is put into 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 blob data used. The **tabid** column is indexed and must contain unique values. A composite index for the **tablename** and **owner** columns allows only unique values.

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. Enter the following SELECT statement to view these rows:

```
SELECT tablename, tabid FROM systables
```



SYSTRACECLASSES

The **systraceclasses** system catalog table contains the names and identifiers of trace classes. A trace class is a category of trace messages that may be used 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 shows the following columns.

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

SYSTRACEMSGS

The **systracemsgs** system catalog table stores internationalized trace messages that may be used in the debugging of 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 provided by the DataBlade API.

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 shows the following columns.

Column Name	Type	Explanation
name	CHAR(18)	The name of the message
msgid	SERIAL	The message template identifier
locale	CHAR(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.



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 text 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 shows the following columns.

Column Name	Type	Explanation
trigid	INTEGER	Trigger identifier
datakey	CHAR	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	INTEGER	Sequence number
data	CHAR(256)	English text or linearized code

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

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 shows the following columns.

Column Name	Type	Explanation
trigid	SERIAL	Trigger identifier
trigname	CHAR(18)	Trigger name
owner	CHAR(8)	Owner of trigger
tabid	INT	Identifier of the triggering table
event	CHAR(1)	Triggering event: 'I' = Insert trigger 'U' = Update trigger 'D' = Delete trigger
old	CHAR(18)	Name of value before update
new	CHAR(18)	Name of value after update
mode	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 shows the following columns.

Column Name	Type	Explanation
username	CHAR(8)	Name of the database user or role
usertype	CHAR(1)	Specifies database-level privileges: 'D' = DBA (all privileges) 'R' = Resource (create user-defined data types and permanent tables and indexes) 'G' = Role 'C' = Connect (work within existing tables)
priority	Reserved for future use	
password	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 into the database. The **sysviews** system catalog table shows the following columns.

Column Name	Type	Explanation
tabid	INTEGER	Table identifier
seqno	SMALLINT	Line number of the SELECT statement
viewtext	CHAR(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 violations and diagnostics tables for base tables. Every table in the database that has a violations and diagnostics table associated with it has a corresponding row in the **sysviolations** table. The **sysviolations** system catalog table shows 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 and diagnostics tables are defined.
viotid	INTEGER	Table identifier of the violations table
diatid	INTEGER	Table identifier of the diagnostics table

(1 of 2)

Column Name	Type	Explanation
maxrows	INTEGER	<p>Maximum number of rows that can be inserted into the diagnostics table during a single insert, update, or delete operation on a target table that has a filtering-mode object defined on it.</p> <p>Also signifies the maximum number of rows that can be inserted into 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).</p> <p>If no maximum has been specified for the diagnostics table, this column contains a null value.</p>

(2 of 2)

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

SYSXTDDESC

The **sysxtddesc** system catalog table provides a text description of each user-defined data type (collection, opaque, distinct, and row types) that is defined in the database. The **sysxtddesc** system catalog table shows 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	CHAR(255)	Textual description of the extended data type

SYSXDTYPEAUTH

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

The **sysxdttypeauth** system catalog table shows the following columns.

Column Name	Type	Explanation
grantor	CHAR(8)	Grantor of privilege
grantee	CHAR(8)	Grantee (receiver) of privilege
type	INTEGER	Identifier of the user-defined type
auth	CHAR(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 **sysxdttypeauth** privilege code is uppercase (for example, 'U' for usage), a user who has this privilege can also grant it to others. If the **sysxdttypeauth** 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.

SYSXTDTYPES

The **sysxdttype** 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 **sysxdtypes** system catalog table shows the following columns.

Column Name	Type	Explanation
extended_id	SERIAL	Unique identifier for extended data types
mode	CHAR(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	CHAR(8)	Owner of the data type
name	CHAR(18)	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, this is the value of the coltype column of 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 distinct type.
source	INTEGER	The sysxdtypes 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.

Column Name	Type	Explanation
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	CHAR(1)	If the data type is passed by value 'T' = type is passed by value 'F' = type is not passed by value
cannothash	CHAR(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.

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 table information without having to use the Informix system catalog tables 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 generating and accessing Information Schema views as well as information about their structure.

Generating the Information Schema Views

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

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

The views are populated by data in the Informix system catalog tables. If tables, views, or stored procedures 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 by using the DROP VIEW statement on each view. Re-create the views by running the script again.



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.

TABLES

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

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 have been granted to PUBLIC, you see one row describing that table.

COLUMNS

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

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 starting 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
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.

(1 of 2)

Column Name	Data Type	Explanation
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)

SQL_LANGUAGES

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 columns that the following table shows.

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.

SERVER_INFO

The **server_info** Information Schema view describes the database server to which the application is currently connected. It contains the columns that the following table shows.

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 information that the following table shows.

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.

Data Types

Data Types Supported by Universal Server	2-5
Summary of Data Types	2-7
Built-In Data Types	2-9
Using DATE, DATETIME, and INTERVAL Data	2-10
Manipulating DATETIME Values	2-11
Manipulating DATETIME with INTERVAL Values	2-12
Manipulating DATE with DATETIME and INTERVAL Values	2-13
Manipulating INTERVAL Values	2-15
Multiplying or Dividing INTERVAL Values	2-16
Large-Object Data Types	2-16
Smart Large Objects	2-17
Simple Large Objects	2-19
Extended Data Types	2-20
Complex Data Types	2-20
Nesting in Complex Data Types.	2-21
Support Function Inheritance with Complex Data Types	2-22
Collection Data Types	2-22
Row Data Types	2-24
Opaque Data Types	2-24
Distinct Data Types	2-24
Data Type Casting	2-25
Using System-Defined Casts	2-25
Converting Number to Number	2-26
Converting Between Number and CHAR	2-27
Converting Between an Integer and DATE or DATETIME.	2-28
Converting Between DATE and DATETIME	2-28
Using Implicit Casts	2-29
Using Explicit Casts	2-29
What Extended Data Types Can Be Cast?.	2-32

Supported Data Types	2-33
BLOB	2-33
BOOLEAN	2-34
BYTE	2-35
CHAR(n)	2-36
Collating CHAR Data	2-36
Multibyte Characters with CHAR	2-36
Treating CHAR Values as Numeric Values	2-37
Nonprintable Characters with CHAR	2-37
CHARACTER(n)	2-37
CHARACTER VARYING(m,r).	2-38
CLOB	2-38
Multibyte Characters with CLOB	2-39
DATE	2-39
DATETIME	2-40
DEC.	2-44
DECIMAL	2-44
DECIMAL Floating Point	2-44
DECIMAL Fixed Point	2-45
DECIMAL Storage	2-45
Distinct Data Type	2-46
DOUBLE PRECISION	2-47
FLOAT(n).	2-47
INT	2-47
INT8	2-47
INTEGER	2-48
INTERVAL	2-48
LIST(e)	2-51
LVARCHAR	2-53
MONEY(p,s).	2-53
MULTISET(e)	2-54
Named Row Type	2-56
Defining Named Row Types	2-56
Equivalence and Named Row Types	2-57
Named Row Types and Inheritance	2-57
Typed Tables	2-57
NCHAR(n)	2-57
NUMERIC(p,s)	2-58
NVARCHAR(m,r)	2-58
Opaque Data Type	2-58
REAL	2-59

SERIAL(n)	2-59
Assigning a Starting Value for SERIAL	2-60
Using SERIAL with INTEGER	2-60
SERIAL8(n).	2-61
Assigning a Starting Value for SERIAL8	2-61
Using SERIAL8 with INT8	2-62
SET(e)	2-62
SMALLFLOAT	2-64
SMALLINT.	2-64
TEXT	2-65
Nonprintable Characters with TEXT.	2-66
Multibyte Characters with TEXT	2-66
Collating TEXT Data	2-66
Unnamed Row Type	2-67
Defining Unnamed Row Types.	2-68
Inserting Values into Unnamed Row Type Columns	2-68
VARCHAR(m,r)	2-69
Multibyte Characters with VARCHAR	2-70
Collating VARCHAR	2-70
Nonprintable Characters with VARCHAR.	2-70
Storing Numeric Values in a VARCHAR Column	2-70
Operator Precedence	2-71

Every column in a table is assigned a *data type*. The data type precisely defines the type of values that you can store in that column. This chapter discusses those data types which can be used to create a column of a table in the database.

This chapter covers the following topics:

- Built-in data types provided by Universal Server
- User-defined data types supported by Universal Server
- Changing the data type of a column
- Casting one data type to another
- A summary of data types that Universal Server supports

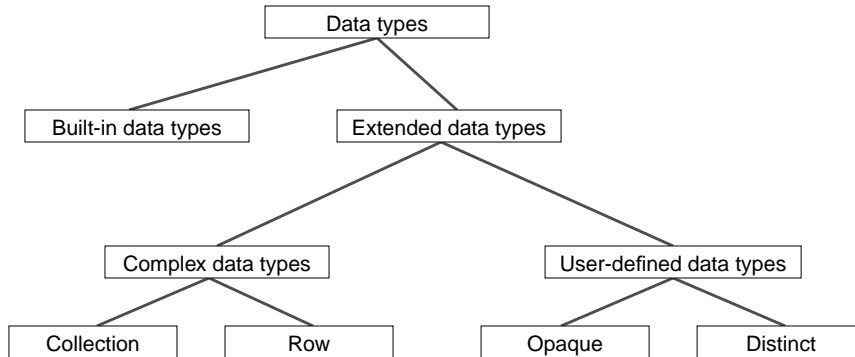
Data Types Supported by Universal Server

Universal Server provides the following support for data types:

- A number of built-in data types
- Ability to allow users to define their own extended data types

The different data types that the Universal Server supports for creating columns of database tables are shown in Figure 2-1.

Figure 2-1
Data Types Supported by Universal Server



Both built-in and extended data types have the following function (with specific exceptions that are mentioned when each type is discussed):

- They can be used to create columns of tables.
- They can be used as arguments and as return types of functions.
- They can be used to create distinct types.
- They can be cast to other types.
- They can be declared and accessed through SPL and ESQL/C.

You assign data types to columns with the CREATE TABLE statement and change them with the ALTER TABLE statement. When you change the data type of a column, all data is converted to the new data type, if possible. For more information on the ALTER TABLE and CREATE TABLE statements and data-type syntax conventions, refer to Chapter 1 of the [Informix Guide to SQL: Syntax](#). For a general introduction to data types, see the [Informix Guide to SQL: Tutorial](#).

Summary of Data Types

Informix products recognize the data types that Figure 2-2 lists. “[Supported Data Types](#)” on page 2-33 describes each of these data types in detail.

Figure 2-2
Data Types Recognized by Informix Products

Data Type	Explanation
BLOB	Stores binary data in random-access chunks
BOOLEAN	Stores Boolean values <code>true</code> and <code>false</code>
BYTE	Stores any kind of binary data
CHAR(<i>n</i>)	Stores single-byte or multibyte sequences of characters, including letters, numbers, and symbols; collation is code-set dependent
CHARACTER(<i>n</i>)	Is a synonym for CHAR
CHARACTER VARYING(<i>m,r</i>)	Stores single-byte and multibyte sequences of characters, including letters, numbers, and symbols of varying length (ANSI compliant); collation is code-set dependent
CLOB	Stores text data in random-access chunks
DATE	Stores calendar date
DATETIME	Stores calendar date combined with time of day
DEC	Is a synonym for DECIMAL
DECIMAL	Stores numbers with definable scale and precision
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
DOUBLE PRECISION	Behaves the same way as FLOAT
FLOAT(<i>n</i>)	Stores double-precision floating-point numbers corresponding to the double data type in C

(1 of 3)

Data Type	Explanation
INT	Is a synonym for INTEGER
INT8	Stores an 8-byte integer value. These whole numbers can be in the range $-(2^{63}-1)$ to $2^{63}-1$.
INTEGER	Stores whole numbers from $-2,147,483,647$ to $+2,147,483,647$
INTERVAL	Stores span of time
LIST(<i>e</i>)	Stores a collection (all elements of same element type, <i>e</i>) of values that have an implicit ordering (first, second, and so on); allows duplicate values
MONEY(<i>p,s</i>)	Stores currency amount
MULTISET(<i>e</i>)	Stores a collection (all elements of same element type, <i>e</i>) of values; allows duplicate values.
NCHAR(<i>n</i>)	Stores single-byte and multibyte sequences of characters, including letters, numbers, and symbols; collation is locale dependent
NUMERIC(<i>p,s</i>)	Is a synonym for DECIMAL
NVARCHAR(<i>m,r</i>)	Stores single-byte and multibyte sequences of characters, including letters, numbers, and symbols of varying length; collation is locale dependent
OPAQUE	Stores user-defined data type whose internal structure is inaccessible to the database server
REAL	Is a synonym for SMALLFLOAT
ROW, Unnamed	Stores an unnamed row type
ROW, Named	Stores a named row type
SERIAL	Stores sequential integers
SERIAL8	Stores large sequential integers; has same range as INT8
SET(<i>e</i>)	Stores a collection (all elements of same element type, <i>e</i>) of values; does not allow duplicate values

(2 of 3)

Data Type	Explanation
SMALLFLOAT	Stores single-precision floating-point numbers corresponding to the float data type in C
SMALLINT	Stores whole numbers from $-32,767$ to $+32,767$
TEXT	Stores any kind of text data
VARCHAR(<i>m,r</i>)	Stores multibyte strings of letters, numbers, and symbols of varying length to a maximum of 255 bytes; collation is code-set dependent

(3 of 3)

Built-In Data Types

Universal Server provides the following built-in data types:

Character data types:	CHAR, CHARACTER VARYING, LVARCHAR, NCHAR, NVARCHAR
Numeric data types:	DECIMAL, MONEY, SMALLINT, INTEGER, INT8, SERIAL, SERIAL8
Large-object data types:	Simple-large-object types: TEXT, BYTE Smart-large-object types: CLOB, BLOB
Time data type:	DATE, DATETIME, INTERVAL
Miscellaneous data types:	BOOLEAN

For a description of each of these data types, refer to the appropriate entry in [“Supported Data Types” on page 2-33](#).

The NCHAR and NVARCHAR data types are also character data types. ♦

For information on how to use time data types, see [“Using DATE, DATETIME, and INTERVAL Data” on page 2-10](#).

Using DATE, DATETIME, and INTERVAL Data

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 2-3 indicates the range of expressions that you can use with DATE, DATETIME, and INTERVAL data, along with the data type that results from each expression.

Figure 2-3
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
CURRENT	—	DATETIME	INTERVAL

(1 of 2)

Data Type of Operand 1	Operator	Data Type of Operand 2	Result
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 Chapter 1 of 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 (1994-9-30 12:30) YEAR TO MINUTE
- DATETIME (1994-8-1 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 1994, and the resulting INTERVAL is negative, indicating that the second date is later than the first.

```
DATETIME (1994-9-30) YEAR TO DAY  
- DATETIME (10-1) MONTH TO DAY
```

```
Result: INTERVAL (1) DAY TO DAY [assuming current year  
is 1994]
```

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 (1991-8-1) YEAR TO DAY  
+ INTERVAL (3-5) YEAR TO MONTH
```

```
Result: DATETIME (1995-01-01) 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 shown in the following example:

```
EXTEND (DATETIME (1994-8-1) YEAR TO DAY, YEAR TO MINUTE)
- INTERVAL (720) MINUTE(3) TO MINUTE

Result: DATETIME (1994-07-31 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 2-4 shows.

Figure 2-4
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 2-4 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 '10/30/1994' as a DATE string but not as a DATETIME string. Instead, you must use '1994-10-30' or '94-10-30' as the DATETIME string.

GLS

If you use a non-default locale, the DATE and DATETIME strings must match the formats that your locale defines. For more information, see the [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 on these environment variables, see the [Guide to GLS Functionality](#). ♦

You also can 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 shown in the following example:

```
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)
```

Note the use of numeric qualifiers to alert 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)
```

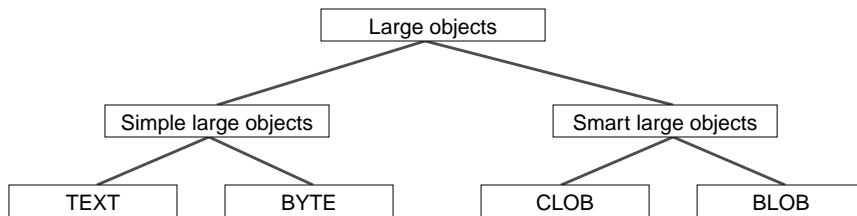
In this example, $15 * 2.5 = 37.5$ minutes, $30 * 2.5 = 75$ seconds, and $2 * 2.5 = 5$ fraction(4). The 0.5 minute is converted into 30 seconds and 60 seconds are converted into 1 minute, which produces the final result of 38 minutes, 45 seconds, and 0.0005 of a second. The results of any calculation include the same amount of precision as the original INTERVAL value.

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 2-5 shows the different large-object data types that Universal Server supports.

Figure 2-5
Large-Object Data Types Supported by Universal Server



For the relative advantages and disadvantages of using simple and smart large objects, see [Chapter 9, “Implementing Your Data Model,”](#) of the *Informix Guide to SQL: Tutorial*.

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 2-24](#).)

Universal 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 2-33 . |
| CLOB | Stores text data. For more detailed information about this data type, see the description on page 2-38 . |

Universal Server stores smart large objects in *sbspaces*. An *sbspace* is a logical storage area that contains one or more chunks that only store BLOB and CLOB data. For information on how to define *sbspaces*, see the *INFORMIX-Universal Server Administrator's 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 more information, see the [INFORMIX-Universal Server Administrator's 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 [INFORMIX-Universal Server DataBlade API Guide](#).

For information on using smart large objects, see [Chapter 9, "Implementing Your Data Model,"](#) of the [Informix Guide to SQL: Tutorial](#) and the [Data Type](#) and [Expression](#) segments in the [Informix Guide to SQL: Syntax](#).

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 determined by your disk capacity. Universal Server supports the following simple-large-object data types:

BYTE	Stores binary data. For more detailed information about this data type, see the description on page 2-35 .
TEXT	Stores text data. For more detailed information about this data type, see the description on page 2-65 .



Tip: In INFORMIX-OnLine Dynamic Server, the TEXT and BYTE data types are collectively called “binary large objects” or just “blobs.” Universal Server uses the term “simple large object” to refer to TEXT and BYTE to distinguish these data types from the CLOB and BLOB data types.

Unlike smart large objects (see [page 2-17](#)), 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.

Universal 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 the [INFORMIX-Universal Server Administrator's Guide](#).

Extended Data Types

Universal Server allows you to characterize data that cannot be represented well by the built-in data types. It allows you to create the following kinds of extended data types:

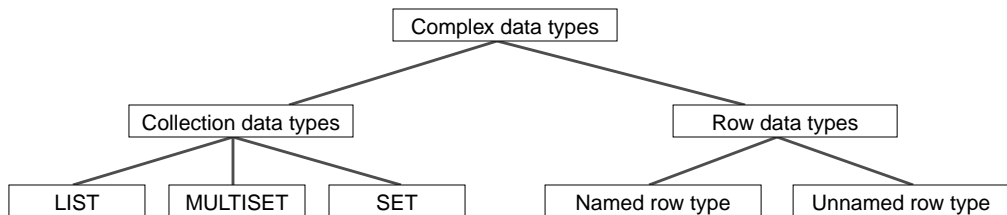
- Complex data types
- Opaque data types
- Distinct data types

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

Complex Data Types

A *complex data type* is a data type that you build from other data types (built-in and extended). Figure 2-6 shows the different complex types that Universal Server supports and the following table briefly describes the structure of these data types.

Figure 2-6
Complex Data Types Supported by Universal Server



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.

Nesting in Complex Data Types

Complex 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. For a discussion of nested complex types, see [Chapter 10](#), “[Understanding Complex Data Types](#),” in the *Informix Guide to SQL: Tutorial*.

Support Function Inheritance with Complex Data Types

All complex types inherit the following support functions:

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

This section summarizes the complex types. For more information on complex types, see [Chapter 10, “Understanding Complex Data Types”](#) in the [Informix Guide to SQL: Tutorial](#).

Collection Data Types

A collection data type is a complex type 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 TEXT, BYTE, SERIAL, or SERIAL8.



Important: *The value of an element cannot be the null value. You must specify the not null constraint for collection elements. No other constraints are valid for collections.*

Universal Server supports three kinds of collection types: SET, MULTISSET, and LIST. 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 Complex Data Type segment in 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 MULTISSET(INTEGER NOT NULL)
)
```

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

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

Leave the brackets empty to indicate an empty set:

```
INSERT INTO table1 VALUE (MULTISSET{})
```

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:

- Procedure variables in an SPL stored routine.
For more information, see [Chapter 14, “Creating and Using SPL Routines”](#) in 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 [“Casting Collection Data Types”](#) on page 13-13 of 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 there are 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 are: *named row types* and *unnamed row types*. 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 the two row types, see pages [2-56](#) and [2-67](#).

You can cast between named and unnamed row types. For a discussion of casting, see [Chapter 10, “Understanding Complex Data Types”](#) of the *Informix Guide to SQL: Tutorial*.

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 given on page [2-58](#).

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 [2-46](#).

Data Type Casting

There may be times when the data type that was assigned to a column with the CREATE TABLE statement is inappropriate. 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:

- You can 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 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 may be added.

- You can 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.

Both data type conversion and casting depend on casts defined in the **syscasts** system catalog table.

A cast is either system-defined, or else explicit or implicit.

Using System-Defined Casts

System-defined casts are owned by user **informix**. They govern conversions from one built-in data type to another. System-defined 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 system-defined casts when required. An infinite number of system-defined casts that may 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 system-defined 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 Number to Number

When you convert data from one numeric type to another, you occasionally find rounding errors. [Figure 2-7 on page 2-27](#) 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 2-7
Numeric Data Type Conversion Chart

FROM	TO					
	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 numeric column (for example, the letter *I* 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 an 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 08/15/1994 becomes 1994-08-15.
- If you convert DATETIME YEAR TO DAY to the DATE format, the value 1994-08-15 becomes 08/15/1994.
- 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, 08/15/1994 becomes 1994-08-15 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, 1994-08-15 12:15:37 becomes 08/15/1994.

Using Implicit Casts

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

The database server automatically invokes a single implicit cast when needed to evaluate and compare expressions or pass arguments. Operations that require more than one implicit cast fail.

Users can explicitly invoke an implicit cast using the CAST AS keywords or the double colon (::) cast operator.

For information on how to define implicit casts, see the CREATE CAST statement in [Informix Guide to SQL: Syntax](#).

Using Explicit Casts

Explicit casts are owned by the users who create them. They govern casts between user-defined data types and other data types.

Developers of user-defined data types must create certain 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. They do not allow you to convert a user-defined data type to a built-in type or vice versa.

Explicit casts, unlike implicit casts or system-defined 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.

For information on how to define explicit casts, see the CREATE CAST statement in the [Informix Guide to SQL: Syntax](#).

Determining Which Cast to Apply

The database server uses the following rules to determine which cast to apply in a particular situation:

- 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 two built-in types, the database server automatically invokes the appropriate system-defined 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 two types or
 - 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 system-defined 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.

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 that allow you to do this:

- 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.

All casts that have been registered for the source type can also be used 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 casting function for a distinct type and register the function as cast, see [Chapter 13, “Casting Data Types,”](#) of the *Informix Guide to SQL: Tutorial*.

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	NA	NA	explicit or implicit ³
Distinct Type	explicit	explicit	explicit	NA	NA	explicit or implicit
Named Row Type	explicit ³	explicit	explicit ¹	explicit ¹	NA	NA
Unnamed Row Type	NA	NA	explicit ¹	implicit ¹	NA	NA
Collection Type	NA	NA	NA	NA	explicit ²	NA
Built-in Type	explicit or implicit ³	explicit or implicit	NA	NA	NA	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.

NA = Not Allowed

Supported Data Types

BLOB

The BLOB is a smart-large-object data type that 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. The BLOB data type has no maximum size. A BLOB column can be up to 4 terabytes in length.

Use the CLOB data type (see [page 2-38](#)) for random access to text data. For general information about the CLOB and BLOB data types, see “[Smart Large Objects](#)” on [page 2-17](#).

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 Expression segment in 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 or vice versa. 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. For more information, see “[Smart Large Objects](#)” on [page 2-17](#).

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

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

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

BOOLEAN

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

BOOLEAN Values	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 simple-large-object 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 BYTE data type has no maximum size. A BYTE column has a theoretical limit of 2^{31} bytes and a practical limit determined by your disk capacity.

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 (**loc_t**) 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 see it all, use the regular syntax for selecting a column. You can also select any part of a BYTE column by using subscripts as shown in the following example:

```
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.

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.

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”.

A character column has a maximum length *n* bytes, where $1 \leq n \leq 32,767$. 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.

Collating CHAR Data

GLS

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 [Guide to GLS Functionality](#). ♦

Multibyte Characters with CHAR

GLS

Multibyte characters used in a database must be supported by the database locale. 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 [Guide to GLS Functionality](#). ♦

Treating CHAR Values as Numeric Values

If you plan to perform calculations on numbers stored in a column, you should assign a numeric data type to that column (see [page 2-9](#)). Although you can store numbers in CHAR columns, you might not be able to use them in some arithmetic operations. For example, if you are inserting 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 numbers with leading zeros 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 using 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 using DB-Access, your screen returns inconsistent results.

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 Informix VARCHAR data type supports the same functionality. See the description of the VARCHAR data type on [page 2-69](#).

CLOB

The CLOB smart-large-object 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 CLOB data type includes special operations for character strings that are inappropriate for BLOB values. The CLOB data type has no maximum size. A CLOB column can be up to 4 terabytes in length.



***Tip:** Use the BLOB data type (see [page 2-33](#)) for random access to binary data. For general information about the CLOB and BLOB data types, see “[Smart Large Objects](#)” on [page 2-17](#).*

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 Expression segment in 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 and vice versa. 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. For more information, see “[Smart Large Objects](#)” on [page 2-17](#).

Multibyte Characters with CLOB

Multibyte CLOB characters must be supported by the database locale. For more information, see the [Guide to GLS Functionality](#).

The CLOB data type is collated in code-set order. For more information on collation orders, see the [Guide to GLS Functionality](#).

For CLOB columns, the database server handles any required code-set conversions for the data. For more information, see the [Guide to GLS Functionality](#). ♦

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 on using CLOB data type, see Chapter 10 of the [Informix Guide to SQL: Tutorial](#).

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 default display format of a DATE column is shown in the following example:

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 to 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 year, Informix products fill in the century digits depending on how you set the **DBCENTURY** environment variable. For example, if you enter the year value as 95, whether that year value is stored as 1995 or 2095 depends on the setting of your **DBCENTURY** 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 3-18](#).

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. You can change the default DATE format by setting the **DBDATE** or **GL_DATE** environment variable. GLS functionality permits the display of international DATE formats. For more information, see the [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 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 listed in [Figure 2-8 on page 2-41](#).

Figure 2-8
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
HOURL	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). Other precisions are indicated explicitly by writing FRACTION(<i>n</i>), where <i>n</i> 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 into a column that is defined as YEAR TO MINUTE, as long as each value entered 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.

For a detailed description of the DATETIME syntax, see the DATETIME field-qualifier segment in the [Informix Guide to SQL: Syntax](#). If you are using 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. (See the discussion of literal DATETIME in Chapter 1 of the [Informix Guide to SQL: Syntax](#).) 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 listed in Figure 2-8.

Values for the field qualifiers are written as integers and separated by delimiters. Figure 2-9 lists the delimiters that are used with DATETIME values in the U.S. ASCII English locale.

Figure 2-9
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 2-10 shows a DATETIME YEAR TO FRACTION(3) value with delimiters.

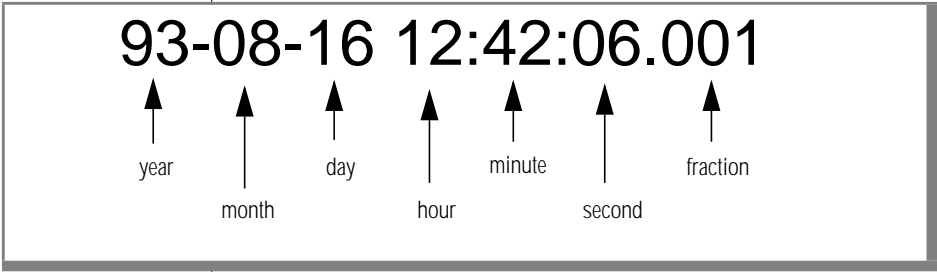


Figure 2-10
Example DATETIME Value with Delimiters

When you enter a value with fewer fields than the defined column, the value 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 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 also can 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, '1993-08-14 08:45', 'maryj', 'D',
    'Order late - placed 6/1/92')
```

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. For more information on entering DATETIME values as character strings, see Chapter 1 of the [Informix Guide to SQL: Syntax](#).

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, the century digits are filled in and interpreted depending on the value you assign to the **DBCENTURY** environment variable. For example, if you enter 95 as the year value, whether the year is interpreted as 1995 or 2095 depends on the setting of the **DBCENTURY** 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 setting and using the **DBCENTURY** environment variable, see [page 3-18](#).

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 a DATETIME value:

$$\text{total number of digits for all fields} / 2 + 1$$

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 [“Using Implicit Casts” on page 2-29](#). For information on how to use DATETIME as a constant expression, see Chapter 1 of the [Informix Guide to SQL: Syntax](#).

If you specify a locale other than U.S. ASCII English, the locale defines the culture-specific display formats for DATETIME values. For more information, see the [Guide to GLS Functionality](#). You can change the default display format by changing the setting of the `GL_DATETIME` environment variable. With an ESQL API, the `DBTIME` environment variable also affects DATETIME formatting. For more information, see [page 3-36](#). 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. ♦

DEC

The DEC data type is a synonym for DECIMAL.

DECIMAL

The DECIMAL data type can take the following two forms:

- `DECIMAL(p)` floating point
- `DECIMAL(p,s)` 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 value range between 10^{-130} and 10^{124} .

If you are using an ANSI-compliant database and specify DECIMAL(*p*), the value defaults to DECIMAL(*p*, 0). A DECIMAL data-type column typically stores numbers with fractional parts that must be stored and displayed exactly (for example, rates or percentages).

See the following discussion for more information about fixed-point decimal values.

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 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 $10^{p-s} - 10^{-s}$

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. This 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 illustrated in Figure 2-11. (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.

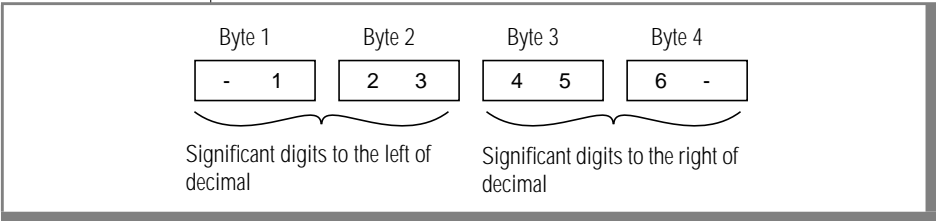


Figure 2-11
Schematic
Illustrating the
Storage of Digits in
a Decimal Value

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).

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

Distinct Data Type

A distinct type is a data type that is based on one of the following source types: a built-in type, or an existing opaque type or distinct type, or complex 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 server.

A distinct type and a source type cannot, however, 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 **sysxdtypes** system catalog table. The following SQL statements maintain the definitions of distinct types in the database:

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

For more information 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 2-31. For examples that show how to create a casting function for a distinct type and register the function as cast, see [Chapter 13](#) of the [Informix Guide to SQL: Tutorial](#).

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 you enter in this type of column and the number the database server displays can differ slightly. This 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 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.

INT

The INT data type is a synonym for INTEGER.

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 always is composed of one value, or a contiguous sequence of values, that represents a component of time. An INTERVAL data type is defined using the following example:

```
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 shown in Figure 2-12, and *n* optionally specifies the precision of the largest field (and smallest field if it is a FRACTION).

Figure 2-12
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
	HOURL	A number of hours
	MINUTE	A number of minutes
	SECOND	A number of seconds
	FRACTION	A decimal fraction of a second, with up to 5 digits of precision. The default precision is 3 digits (thousandth of a second). Other precisions are indicated explicitly by writing FRACTION(<i>n</i>), where <i>n</i> 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 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 segment in Chapter 1 of 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 displayed in [Figure 2-12 on page 2-49](#).

If you are using 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 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 you are entering. For example, to define an INTERVAL of DAY TO HOUR that can store up to 999 days, you could specify it as shown in the following example:

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

INTERVAL values use the same delimiters as DATETIME values. The delimiters are shown in [Figure 2-13](#).

Figure 2-13
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 also can 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 Chapter 1 of 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:

$$\text{total number of digits for all fields} / 2 + 1$$

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 how to use INTERVAL data in arithmetic and relational operations, see “[Using Implicit Casts](#)” on [page 2-29](#). For information on how to use INTERVAL as a constant expression, see the description of the INTERVAL Field Qualifier segment in Chapter 1 of the [Informix Guide to SQL: Syntax](#).

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 2-54](#) and the SET data type on [page 2-62](#).)

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 wish to insert a list-element value. For more information, see the INSERT statement in of 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:

- A built-in type, except SERIAL, SERIAL8, BYTE, and TEXT
- An opaque type
- A distinct type
- An unnamed row type
- Another collection 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 Data Type segment in 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
For more information, see the Condition segment in the [Informix Guide to SQL: Syntax](#).
- As an argument to the SQL CARDINALITY function to determine the number of elements in a LIST column
For more information, see the Expression segment in the [Informix Guide to SQL: Syntax](#).

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")
```

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 clients can manipulate. Each also has an output support function and cast, which convert it from its internal representation to LVARCHAR.

When used in other contexts, the LVARCHAR data type has the following restrictions:

- The in-row storage for an LVARCHAR value is limited to 4 kilobytes (4Kb).

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 always is 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 up 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.

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. For more information, see the [Guide to GLS Functionality](#). ♦

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 (.)

GLS

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, see the [Guide to GLS Functionality](#). ♦

You can change the format for MONEY values by changing the **DBMONEY** environment variable. See [page 3-27](#) for information on how to set the **DBMONEY** environment variable.

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 2-51](#).)

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:

- A built-in type, except SERIAL, SERIAL8, BYTE, and TEXT
- An opaque type
- An unnamed row type
- Another collection 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 Data Type segment in 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

For more information, see the Condition and Expression segments in the [Informix Guide to SQL: Syntax](#).

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 Type

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 (see the description on [page 2-67](#)). Use a named row type if you want to use type inheritance.

Defining Named Row Types

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

The fields of a row type can be any data type, except SERIAL and SERIAL8. 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:

- A built-in type, except the restriction against SERIAL, SERIAL8, TEXT, and BYTE mentioned above.
- A collection type
- A distinct type
- An opaque type
- A row 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 SQL syntax statements above, see [Informix Guide to SQL: Syntax](#).

For examples on how to create tables and populate tables with named row type columns, see Chapter 10, “[Understanding Complex Data Types](#)” in the [Informix Guide to SQL: Tutorial](#).

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 [Chapter 10, “Understanding Complex Data Types”](#) in the [Informix Guide to SQL: Tutorial](#).

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. See the CREATE TABLE statement in the [Informix Guide to SQL: Syntax](#) to learn how to create typed tables. Table inheritance and how it relates to type inheritance is also discussed in that section.

For examples on creating and populating typed tables, see [Chapter 12, “Accessing Complex Data Types,”](#) in the [Informix Guide to SQL: Tutorial](#).

NCHAR(*n*)

GLS

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 the 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 [Guide to GLS Functionality](#). ♦

NUMERIC(*p,s*)

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

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 [Guide to GLS Functionality](#). ♦

Opaque Data Type

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 **sysxdtypes** 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 statements, see the [Informix Guide to SQL: Syntax](#).

For information on how to create opaque types and an example of an opaque type, see the [Extending INFORMIX-Universal Server: Data Types](#) manual.

REAL

The REAL data type is a synonym for SMALLFLOAT.

SERIAL(*n*)

The SERIAL data type stores a sequential integer assigned automatically by the database server when a row is inserted. (For more information on inserting values into SERIAL columns, see the [Informix Guide to SQL: Syntax](#).) 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.

The following restrictions apply to SERIAL columns:

- You can define only one SERIAL column in a table.
However, a table can have one SERIAL and one SERIAL8 column.
- The SERIAL data type is not automatically a unique column.
You must apply a unique index to this column to prevent duplicate serial numbers.

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



Assigning a Starting Value for SERIAL

The default serial starting number is 1, but you can assign an initial value, *n*, when you create or alter the table. If you specify 0 as your starting number when inserting the first row in a table with a SERIAL column, the database server assigns the value 1 to that row in the serial column, and increments the value by one at each subsequent insert. The highest serial number you can assign is 2,147,483,647. If you assign a number greater than 2,147,483,647, you receive a syntax error.

Tip: *If you need to store assigned values larger than 2,147,483,647, use the SERIAL8 data type (see [page 2-61](#)).*

Once a nonzero number is assigned, it cannot be changed. You can, however, insert a value into 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 into 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 using the following formula:

$$\text{maximum existing value in SERIAL column} + 1$$

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.

Using SERIAL with INTEGER

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

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

SERIAL8(*n*)

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 Chapter 1 of 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:

$$\text{maximum existing value in SERIAL8 column} + 1$$

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.

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 the MULTiset in [“Supported Data Types” on page 2-33](#).) The elements in a SET have no ordinal position. That is, there is no concept of a first, second, or third element in a SET. (For a collection type with ordinal positions for elements, see the LIST data type on [page 2-51](#).)

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:

- A built-in type, except SERIAL, SERIAL8, BYTE, and TEXT
- An opaque type
- An unnamed row type
- Another collection 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 Data Type segment in 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

For more information, see the Condition segment in the [Informix Guide to SQL: Syntax](#).

- As an argument to the SQL CARDINALITY function to determine the number of elements in a SET column

For more information, see the Expression segment 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; 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 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 into 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.

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 very 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 simple-large-object data type stores any kind of text data. It can contain both single and multi-byte characters. For more information on multibyte characters of the TEXT data type, see [“Multibyte Characters with TEXT” on page 2-66](#).

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

TEXT columns typically store memos, manual chapters, business documents, program source files, and so on. In the default locale U.S. ASCII English, a TEXT data type 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 into TEXT columns in the following ways:

- With the **dbload** or **onload** utilities
- With the LOAD statement (DB-Access)
- From TEXT (**loc_t**) 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 also can select any part of a TEXT column by using subscripts, as shown in the following example:

```
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.

Nonprintable Characters with TEXT

Both printable and nonprintable characters can be inserted into text columns. Informix products do not check the data that is inserted into a column with the TEXT data type. For detailed information on entering and displaying nonprintable characters, refer to [“Nonprintable Characters with CHAR” on page 2-37](#).

Multibyte Characters with TEXT

Multibyte TEXT characters must be supported by the database locale. For more information, see the [Guide to GLS Functionality](#). ♦

Collating TEXT Data

TEXT data type is collated in code-set order. For more information on collation orders, see the [Guide to GLS Functionality](#). ♦

GLS

GLS

Unnamed Row Type

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:

- A built-in type, except SERIAL, SERIAL8
- A collection type
- A distinct type
- A domain name
- An opaque type
- A row type

For more information on defining unnamed row types, see the Data Type segment in the [Informix Guide to SQL: Syntax](#).

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

Defining 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)
```

- An unnamed row type can be created by an implicit or explicit cast from a named row type or from another unnamed row type.
- The rows of an untyped table are considered to be unnamed row types.

Inserting Values into Unnamed Row Type Columns

When you specify field values for an unnamed row type, list the field values after the constructor and between parentheses. For example, suppose you have an unnamed row-type column with the following type:

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](#).

VARCHAR(*m,r*)

The VARCHAR data type stores 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. For more information on multibyte VARCHAR sequences, see [“Multibyte Characters with VARCHAR” on page 2-70](#).

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, described on [page 2-38](#).

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 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.

GLS

GLS

Multibyte Characters with VARCHAR

Multibyte VARCHAR characters must be supported by the database locale. If you are storing multibyte characters, make sure to calculate the number of bytes needed. For more information, see the [Guide to GLS Functionality](#). ♦

Collating VARCHAR

The main difference between the NVARCHAR and the VARCHAR data types is the difference in collation sequencing. The 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 [Guide to GLS Functionality](#). ♦

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 2-37.

Storing Numeric Values in a VARCHAR Column

When you insert a numeric value into 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.



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 Universal Server supports. The table shows operators in decreasing order of precedence from the top. Operators with the same precedence are shown in the same row.

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

Environment Variables

Types of Environment Variables	3-3
Where to Set Environment Variables	3-4
Setting Environment Variables at the System Prompt.	3-4
Setting Environment Variables in an Environment-Configuration File	3-4
Setting Environment Variables at Login Time	3-6
Manipulating Environment Variables	3-6
Setting Environment Variables	3-6
Viewing Your Current Settings	3-7
Unsetting Environment Variables	3-7
Modifying the Setting of an Environment Variable	3-8
Checking Environment Variables with the chkenv Utility	3-9
Rules of Precedence	3-10
List of Environment Variables	3-10
Environment Variables	3-14
ARC_DEFAULT	3-14
ARC_KEYPAD	3-14
DBANSIWARN.	3-15
DBBLOBBUF	3-17
DBCENTURY	3-18
DBDATE	3-21
DBDELIMITER.	3-24
DBEDIT	3-24
DBFLTMASK	3-25
DBLANG.	3-26
DBMONEY	3-27
DBONPLOAD	3-29
DBPATH	3-29

DBPRINT	3-32
DBREMOTECD	3-33
DBSPACETEMP	3-34
DBTEMP	3-35
DBTIME	3-36
DBUPSPACE.	3-39
DELIMIDENT	3-40
ENVIGNORE	3-41
FET_BUF_SIZE	3-41
INFORMIXC.	3-42
INFORMIXCONCSMCFG	3-43
INFORMIXCONRETRY	3-43
INFORMIXCONTIME	3-44
INFORMIXDIR	3-45
INFORMIXKEYTAB	3-46
INFORMIXOPCACHE	3-46
INFORMIXSERVER	3-47
INFORMIXSHMBASE	3-48
INFORMIXSQLHOSTS	3-49
INFORMIXSTACKSIZE	3-49
INFORMIXTERM	3-50
INF_ROLE_SEP.	3-51
IFX_AUTOFREE	3-51
IFX_DEFERRED_PREPARE	3-53
NODEFDAC.	3-54
ONCONFIG	3-54
OPTCOMPIND	3-55
PATH	3-56
PDQPRIORITY	3-56
PLCONFIG	3-58
PSORT_DBTEMP	3-58
PSORT_NPROCS	3-59
Default Values for Ordinary Sorts	3-60
Default Values for Attached Indexes	3-60
TERM	3-61
TERMCAP	3-61
TERMINFO	3-62
THREADLIB.	3-63

Index of Environment Variables	3-63
--	------

Various *environment variables* affect how your Informix products function. You can set environment variables that identify your terminal, specify the location of your software, and define other parameters. The environment variables discussed in this chapter are grouped and listed alphabetically beginning on [page 3-10](#). In addition, an index of environment variables is included at the end of this chapter, on [page 3-63](#).

Some environment variables are required and others are optional. For example, you must 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.

Types of Environment Variables

The environment variables discussed in this chapter fall into the following categories:

- **Informix environment variables.** Set these standard environment variables when you want to work with Informix products. Each product manual specifies the environment variables that you must set to use that product.
- **UNIX environment variables.** Informix products rely on the correct setting of certain standard UNIX system environment variables. The **PATH** and **TERM** environment variables must always be set. You might also have to set the **TERMCAP** or **TERMINFO** environment variable to use some products effectively.

GLS

The GLS environment variables that allow you to work in a nondefault locale are described in Chapter 2 of the [Guide to GLS Functionality](#). However, these variables are included in the list of environment variables on [page 3-10](#) and in the index table in [Figure 3-2 on page 3-63](#). ♦

Where to Set Environment Variables

You can set environment variables in the following locations:

- At the system prompt on the command line
- In an environment-configuration file
- In a login file

Setting Environment Variables at the System Prompt

When you set an environment variable at the system prompt, you must reassign it the next time you log in to the system. For more information about how to do this, see [“Manipulating Environment Variables” on page 3-6](#).

Setting Environment Variables in an Environment-Configuration File

The environment-configuration file is a common or private file where you can define all the environment variables that are used by Informix products. Using an environment-configuration file reduces the number of environment variables that you must set at the command line or in a shell file.

The common (shared) environment-configuration file resides in the **\$INFORMIXDIR/etc/informix.rc** file. The permission for this shared file must be set to 644. A user can override the system or common environment variables by setting variables in a *private environment-configuration file*. The private environment-configuration file must have the following characteristics:

- The file is stored in the home directory of the user
- The file is named **.informix**
- Permissions are set, by the user, to readable

An environment-configuration file can contain comment lines [preceded by the pound (#) sign] and variable lines and their values (separated by blanks and tabs), as shown in the following example:

```
# This is an example of an environment-configuration file
#
DBDATE DMY4-
#
# These are ESQL/COBOL environment variable settings
#
INFORMIXCOB rmcobol
INFORMIXCOBTYP rm85
INFORMIXCOBDIR /usr/lib/rmcobol
```

You can use the **ENVIGNORE** environment variable to override one or more entries in this file. Use the Informix **chkenv** utility 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 entry or if the file is too large. The **chkenv** utility is described on [page 3-9](#).

The first time you set an environment variable in a shell file or configuration file, before you work with your Informix product you should *source* the file (if you are using a C shell) or use a period (.) to execute an environment-configuration file (if you are using a Bourne or Korn shell). This procedure tells the shell process to read your entry.

Setting Environment Variables at Login Time

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

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

Manipulating Environment Variables

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

Setting Environment Variables

Use standard UNIX commands to set environment variables. Depending on the type of shell you use, Figure 3-1 shows how you set the **ABCD** environment variable to *value*. The environment variables are case-sensitive.

Figure 3-1
Different Shell Settings

Shell	Command
C	setenv ABCD values
Bourne	ABCD=value export ABCD
Korn	ABCD=value export ABCD
Korn	export ABCD=value

Korn-shell syntax supports a shortcut, as shown in the last line of Figure 3-1.

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 shown in [Figure 3-1 on page 3-6](#).

```
setenv _____ ABCD _____ value _____|
```

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

Viewing Your Current Settings

After one or more Informix products are installed, enter the following command at the system prompt to view your current environment settings:

UNIX Version	Command
BSD UNIX	env
UNIX System V	printenv

Unsetting Environment Variables

To unset an environment variable, enter the following command:

Shell	Command
C	unsetenv ABCD
Bourne or Korn	unset ABCD

Modifying the Setting of an Environment Variable

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 shown in the following example:

```
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 shown in the following command:

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

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

Checking Environment Variables with the *chkenv* Utility

The **chkenv** utility checks the validity of shared or private environment-configuration files. Use it to provide debugging information when you define, in an environment-configuration file, all the environment variables that are used by your Informix products.

```
chkenv _____ filename _____ |
```

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

The common environment-configuration file is stored in **\$INFORMIXDIR/etc/informix.rc**. A private environment-configuration file is stored in the home directory of the user as **.informix**.

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 entry 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 settings.

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:

```
-33500 filename: Bad environment variable on line number.
```

If you want the product to ignore specified environment-variable settings 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 3-41](#).

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 has been defined in the environment (shell) by explicitly setting the value at the shell prompt.
2. The second-highest precedence goes to the value that has been defined in the private environment-configuration file in the home directory of the user (`~/.informix`).
3. The next-highest precedence goes to the value that has been defined in the common environment-configuration file (`$INFORMIXDIR/etc/informix.rc`).
4. The next highest precedence goes to the value that has been defined in your `.login` file.
5. The lowest precedence goes to the default value.

For precedence information about GLS environment variables, see the [Guide to GLS Functionality](#). ♦

List of Environment Variables

The following table 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

The GLS environment variables are discussed in the [Guide to GLS Functionality](#). ♦

Environment Variable	Restrictions	Page
ARC_DEFAULT	Universal Server only	3-14
ARC_KEYPAD	Universal Server only	3-14
CC8BITLEVEL	ESQL/C only	Guide to GLS Functionality
CLIENT_LOCALE		Guide to GLS Functionality
DBANSIWARN		3-15
DBBLOBBUF	Universal Server only	3-17
DBCENTURY	SQL APIs only	3-18
DBDATE		3-21; Guide to GLS Functionality
DBDELIMITER		3-24
DBEDIT		3-24
DBFLTMASK	DB-Access only	3-25
DBLANG		3-26; Guide to GLS Functionality
DBMONEY		3-27; Guide to GLS Functionality
DBONPLOAD	High-Performance Loader only	3-29
DBPATH		3-29
DBPRINT		3-32
DBREMOTECMD	Universal Server only	3-33
DBSPACETEMP	Universal Server only	3-34

(1 of 3)

List of Environment Variables

Environment Variable	Restrictions	Page
DBTIME	SQL APIs only	3-36; Guide to GLS Functionality
DBUPSPACE		3-39
DB_LOCALE		Guide to GLS Functionality
DELIMIDENT		3-40
ENVIGNORE		3-41
ESQLMF		Guide to GLS Functionality
FET_BUF_SIZE	SQL APIs and DB-Access only	3-41
GLS8BITSYS		Guide to GLS Functionality
GL_DATE		Guide to GLS Functionality
GL_DATETIME		Guide to GLS Functionality
INFORMIXC	ESQL/C only	3-42
INFORMIXCONCSMCFG	Universal Server only	3-43
INFORMIXCONRETRY		3-45
INFORMIXDIR		3-45
INFORMIXKEYTAB	Universal Server only	3-46
INFORMIXOPCACHE	OnLine/Optical only	3-46
INFORMIXSERVER		3-47
INFORMIXSHMBASE	Universal Server only	3-48
INFORMIXSQLHOSTS		3-49
INFORMIXSTACKSIZE	Universal Server only	3-49
INFORMIXTERM	DB-Access only	3-50
INF_ROLE_SEP	Universal Server only	3-51
NODEFDAC		3-51

(2 of 3)

Environment Variable	Restrictions	Page
ONCONFIG	Universal Server only	3-54
OPTCOMPIND	Universal Server only	3-55
PATH		3-56
PDQPRIORITY	Universal Server only	3-56
PLCONFIG	High-Performance Loader only	3-58
PSORT_DBTEMP	Universal Server only	3-58
PSORT_NPROCS	Universal Server only	3-59
SERVER_LOCALE		Guide to GLS Functionality
TERM		3-61
TERMCAP		3-61
TERMINFO		3-62
THREADLIB	ESQL/C only	3-63

(3 of 3)

Environment Variables

The following sections discuss the environment variables used by Informix products.

ARC_DEFAULT

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

```
setenv _____ ARC_DEFAULT _____ pathname _____
```

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 the [INFORMIX-Universal Server Archive and Backup Guide](#).

ARC_KEYPAD

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

The **ttermcap** 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 to draw menus and borders for an API.

```
setenv _____ ARC_KEYPAD _____ pathname _____ |
```

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

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

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

For more information on archiving, see the [*INFORMIX-Universal Server Archive and Backup Guide*](#).

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; setting it to any value or to no value, as the following diagram shows, is sufficient.

```
setenv _____ DBANSIWARN _____ |
```

Setting the **DBANSIWARN** environment variable for DB-Access is functionally equivalent to including the **-ansi** flag when invoking 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 SQL Communication Area (SQLCA) variable **sqlca.sqlwarn.sqlwarn5** to be set to **W** when a statement that is not ANSI-compliant is executed. (For more information on SQLCA, see the [INFORMIX-ESQL/C Programmer's Manual](#).)

Once 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 a simple large object, namely TEXT or BYTE, is stored temporarily in memory or in a file while being unloaded with the UNLOAD statement. The term, **DBBLOBBUF**, is used for historical reasons, even though it represents the size of simple large objects, and not of BLOBs, which are smart large objects.

setenv _____ DBBLOBBUF _____ *n* _____

n represents the maximum size of a simple large object in kilobytes.

If the simple large object is smaller than the default of 10 coagulates or the setting of the **DBBLOBBUF** environment variable, it is temporarily stored in memory. If the simple large object 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.

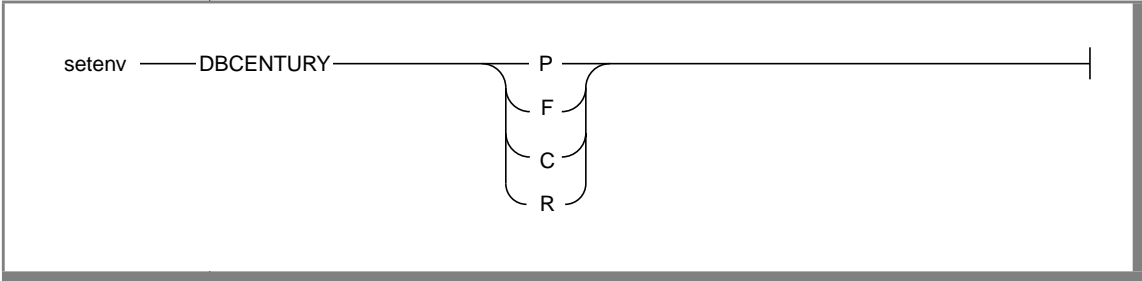
For instance, to set a buffer size of 15 kilobytes, set the **DBBLOBBUF** environment variable as the following example shows:

```
setenv DBBLOBBUF 15
```

In the example, any blobs smaller than 15 kilobytes are stored temporarily in memory. Blobs larger than 15 kilobytes are stored temporarily in a file.

DBCENTURY

The environment variable **DBCENTURY** allows you to choose the appropriate expansion for two-digit year DATE and DATETIME values.



Previously, if only the decade was provided for a literal DATE or DATETIME value in a table column, the present century was used to expand the year. For example, 12/31/96 would have been expanded to 12/31/1996. With this release, three new algorithms are added to complete the century value of a year: past (P), future (F), and closest (C).

Algorithm	Explanation
P = Past	The past and present 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 present 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 past, present, 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 present century is used to expand the year value.

When the **DBCENTURY** environment variable is not set, the current century is used as the system default.

You can override the default by specifying all four digits.

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: 4/6/1996
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: 1/1/1
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 and it is chosen as the expansion of the year value.

Behavior of DBCENTURY = C

Example data type: DATE
Current date: 4/6/1996
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.

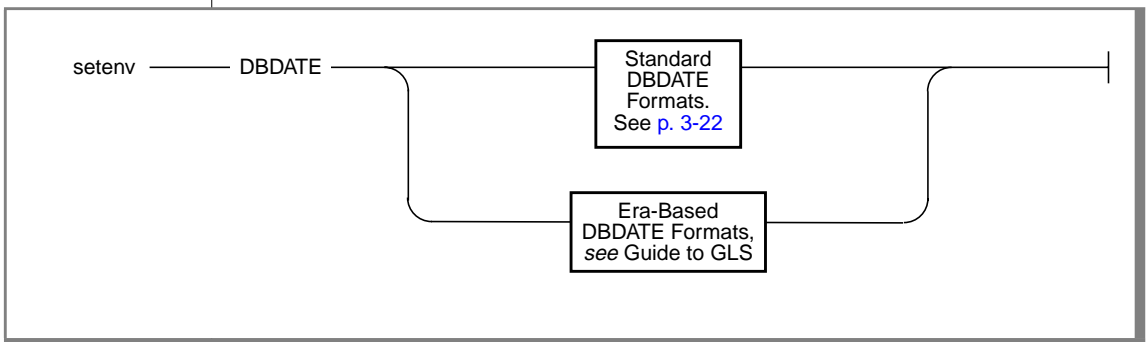
Behavior of DBCENTURY = R

Example data type: DATETIME year to month
Current date: 4/6/1996
User enters: 1/1/1
DBCENTURY = R, Present century algorithm
Present century expansion: 1901-1
Analysis: The present century expansion is used.

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 specified by **DBDATE**. Similarly, the database server interprets the date you are specifying as input to the **DATE()** function in the format specified by the **DBDATE** environment variable.
- When you display DATE values, Informix products use the **DBDATE** environment variable to format the output.

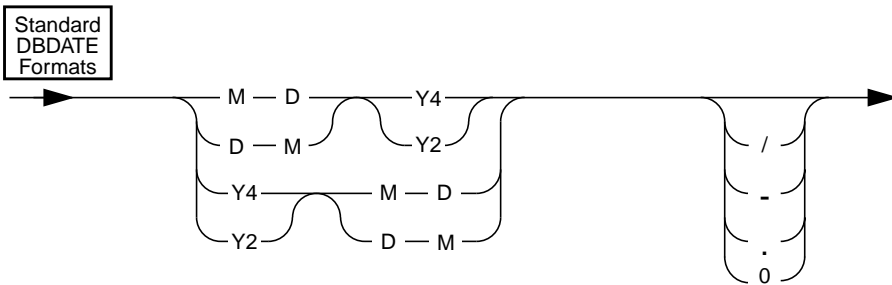


GLS

This section describes standard **DBDATE** formats. For a description of era-based formats, see the [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 representing 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, 10/08/1994).

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

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 a few variations of setting the **DBDATE** environment variable.

Variation	October 8, 1994 appears as:
MDY4/	10/08/1994
DMY2-	08-10-94
MDY4	10/08/1994
Y2DM.	94.08.10
MDY20	100894
Y4MD*	1994/10/08

Notice that 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, understand that the setting of the **DBCENTURY** environment variable affects how the DATE values are expanded.

Also, certain routines called by INFORMIX-ESQL/C 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 3-36](#) and the “[INFORMIX-ESQL/C Programmer’s Manual](#).” The setting of the **DBDATE** variable takes precedence over that of the **GL_DATE** environment variable, as well as over the default DATE formats as specified by **CLIENT_LOCALE**. For information about the **GL_DATE** and **CLIENT_LOCALE** environment variables, see the “[Guide to GLS Functionality](#)”. ♦



GLS

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 shown in the following example:

```
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 editor is called directly. If **DBEDIT** is not set, you are prompted to specify an editor as the default for the rest of the session.

```
setenv DBEDIT editor
```

editor is the name of the text editor you want to use.

For most systems, the default editor is **vi**. If you use another 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

By default, Informix client applications (including DB-Access utility or any ESQL program) display the floating-point values of data types **FLOAT**, **SMALLFLOAT**, and **DECIMAL** with 16 digits to the right of the decimal point. However, the actual number of decimal digits displayed depends on the size of the character buffer.

To override the default number of decimal digits in the display, you can set the **DBFLTMASK** environment variable to the number of digits desired.

```
setenv DBFLTMASK n
```

n is the number of decimal digits you want the Informix client application to display in the floating-point values.

DBLANG

The **DBLANG** environment variable specifies the subdirectory of **\$INFORMIXDIR** or the full pathname of the directory that contains the compiled message files used by an Informix product.



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 have created, perform the following steps.

To use a non-default message directory

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 only need to 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 specified by **\$DBLANG**. All the files in the message directory should have the owner and group **informix** and access permission **644**.

GLS

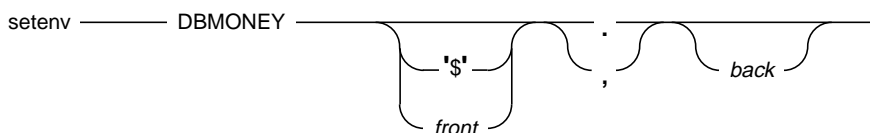
Informix products that use the default U.S. ASCII English locale search for message files in the following order:

1. In `$DBLANG`, if `DBLANG` is set to a full pathname
2. In `$INFORMIXDIR/msg/$DBLANG`, if `DBLANG` is set to a relative pathname
3. In `$INFORMIXDIR/$DBLANG`, if `DBLANG` is set to a relative pathname
4. In `$INFORMIXDIR/msg/en_us/0333`
5. In `$INFORMIXDIR/msg/en_us.8859-1`
6. In `$INFORMIXDIR/msg`
7. In `$INFORMIXDIR/msg/english`

For more information on access paths for messages, see the description of `DBLANG` in the [Guide to GLS Functionality](#). ♦

DBMONEY

The `DBMONEY` environment variable specifies the display format of monetary values using `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** represents 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 (\$) preceding the MONEY value, a period (.) separating 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. Set the **DBMONEY** environment variable by entering the following command:

```
setenv DBMONEY DM,
```

Here, DM is the currency symbol preceding 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.

For more information about how the **DBMONEY** environment variable handles MONEY formats for nondefault locales, see the [Guide to GLS Functionality](#). ♦

DBONPLOAD

The **DBONPLOAD** environment variable specifies the name of the database that the **onpload** utility of the High-Performance Loader 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**.

```
setenv _____ DBONPLOAD _____ dbname _____
```

dbname specifies the name of the database to be used by the **onpload** utility.

For example, to specify **load_db** as the name of the database, enter the following command:

```
setenv DBONPLOAD load_db
```

DBPATH

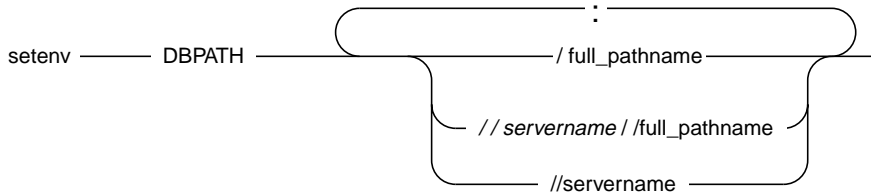
Use **DBPATH** to identify the database servers that contain databases (if you are using Universal Server). The **DBPATH** environment variable also specifies a list of directories (in addition to the current directory) in which DB-Access looks for command scripts (**.sql** files).

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 or the default directory

The **CREATE DATABASE** statement does not use **DBPATH**.

To add a new **DBPATH** entry to existing entries, see [“Modifying the Setting of an Environment Variable” on page 3-8](#).



full_pathname is a valid full pathname of a directory in which **.sql** files are stored.

servername is the name of a Universal 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 are setting **DBPATH**.

When you access a database using 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. (The default directory is the current working directory if the database server is on the local computer or your login directory if the database server is on a remote computer.) If a directory is specified but is not a full path, the directory is considered to be relative to the default directory.

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 the indicated database. The entries to **DBPATH** are considered in order.

Using DBPATH with DB-Access

If you are using DB-Access and you use the Choose option of 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. For Universal Server databases, 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 databases or **.sql** scripts on your local computer.

In the following example, the **DBPATH** setting causes DB-Access to search for the database files in your current directory and then in the Joachim and Sonja directories on the local computer:

```
setenv DBPATH /usr/joachim:/usr/sonja
```

As shown in the previous example, if the pathname specifies a directory name but not a database server name, the directory is sought on the computer running the default database server as specified by the **INFORMIXSERVER** environment variable (see [page 3-47](#)). For instance, with the previous example, if **INFORMIXSERVER** is set to **quality**, the **DBPATH** value is *interpreted* as shown in the following example, where the double slash precedes the database server name:

```
setenv DBPATH //quality/usr/joachim://quality/usr/sonja
```

Searching Networked Computers for Databases

If you are using 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 shown in the following example:

```
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 Universal Server or SE administrator must include each database server mentioned by **DBPATH** in the `$INFORMIXDIR/etc/sqlhosts` file. For information on communication-configuration files and dbservernames, see the [INFORMIX-Universal Server Administrator's Guide](#).

For example, if **INFORMIXSERVER** is set to **quality**, you can search for a Universal Server database first on the **quality** database server and then on the **marketing** database server by setting **DBPATH** shown in the following example:

```
setenv DBPATH //marketing
```

If you are using DB-Access in this example, the names of all the databases on the **quality** and **marketing** database servers are displayed with the Select option of the DATABASE menu.

DBPRINT

The **DBPRINT** environment variable specifies the printing program that you want to use.

```
setenv DBPRINT program
```

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**.

Set the **DBPRINT** environment variable to specify the **myprint** print program by entering the following command:

```
setenv DBPRINT myprint
```


DBREMOTECMD

You can set the **DBREMOTECMD** environment variable to override the default remote shell used when you perform remote tape operations with the INFORMIX-Universal Server. Set it using either a simple command or the full pathname. If you use the full pathname, the database server searches your **PATH** for the specified command.

```
setenv DBREMOTECMD _____|
                        |
                        |  command
                        |
                        |  pathname
```

command is the command to override the default remote shell.

pathname is the pathname to override the default remote shell.

Informix 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**).

Set the **DBREMOTECMD** environment variable for a simple command name by entering the following command:

```
setenv DBREMOTECMD rcmd
```

Set the **DBREMOTECMD** environment variable to specify the full pathname by entering the following command:

```
setenv DBREMOTECMD /usr/bin/remsh
```

For more information on **DBREMOTECMD**, see the discussion in the [INFORMIX-Universal Server Archive and Backup Guide](#) about using remote tape devices with Universal Server for archives, restores, and logical-log backups.

DBSPACETEMP

If you are using Universal Server, 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 specified by the **DBSPACETEMP** configuration parameter in the Universal Server configuration file.

For example, you might set the **DBSPACETEMP** environment variable by entering 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 the UNIX shell. Universal 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 are created by the user and implicit temporary tables that are created by Universal Server. You 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, Universal Server uses the settings in the **DBSPACETEMP** environment variable to determine where to create the table. If the **DBSPACETEMP** environment variable is not set, Universal Server uses the **ONCONFIG** parameter **DBSPACETEMP**. If this parameter is not set, Universal Server creates the temporary table in the same dbspace where the database resides.

If you create an explicit temporary table with the `SELECT INTO TEMP` statement, Universal Server uses the settings in the **DBSPACETEMP** environment variable to determine where to create the table. If the **DBSPACETEMP** environment variable is not set, Universal Server uses the `ONCONFIG` parameter **DBSPACETEMP**. If this parameter is not set, Universal Server creates the temporary table in the root dbspace.

Universal 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, Universal Server uses disk space for writing the temporary data, in the following order:

1. The operating system directory or directories specified by the environment variable **PSORT_DBTEMP**, if it is set
2. The dbspace or dbspaces specified by the environment variable **DBSPACETEMP**, if it is set
3. The dbspace or dbspaces specified by the `ONCONFIG` parameter **DBSPACETEMP**
4. The operating-system file space in **/tmp**

DBTEMP

Set the **DBTEMP** environment variable to specify the full pathname of the directory into which you want INFORMIX-SE or INFORMIX-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:

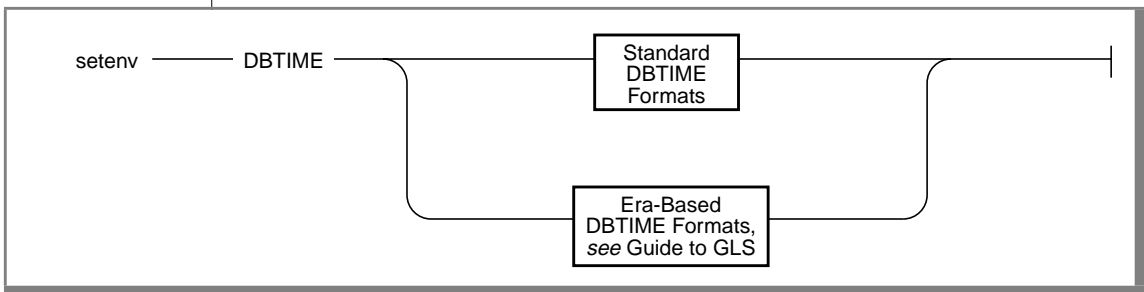
```
setenv DBTEMP usr/magda/mytemp
```

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).

Universal Server uses **DBSPACETEMP** to specify the location of temporary files.

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 more information, 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 by using the formatting directives described in the following list. Otherwise, the behavior of the DATETIME formatting routine is undefined.

Standard
DBTIME
Formats

'string'

string The formatting directives that you can use are described in the following list:

- %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 specified by the integer *n*. 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].

(1 of 2)

- %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).

(2 of 2)

For example, consider how to convert a DATETIME YEAR TO SECOND to the following ASCII string format:

```
Mar 21, 1994 at 16 h 30 m 28 s
```

You set **DBTIME** as shown in the following list:

```
setenv DBTIME '%b %d, %Y at %H h %M m %S s'
```

The default **DBTIME** produces the conventional ANSI SQL string format shown in the following line:

```
1994-03-21 16:30:28
```

The default **DBTIME** is set as shown in the following example:

```
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 described in the following list:

- [-|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 specified by a precision, it is padded with leading zeros.
- If a conversion specification supplies more characters than specified by a precision, 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.

See the discussion of `DBDATE` on [page 3-21](#) for related information.

DBUPSPACE

The `DBUPSPACE` environment variable lets you specify and constrain the amount of system disk space that the `UPDATE STATISTICS` statement can use when trying to simultaneously construct multiple-column distributions.

```
setenv _____ DBUPSPACE _____ value _____
```

value represents a disk-space amount in kilobytes.

For example, to set `DBUPSPACE` to 2,500 kilobytes, enter the following command:

```
setenv DBUPSPACE 2500
```

Then no more than 2,500 kilobytes of disk space can be used 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 that specified in **DBUPSPACE**.

DELIMIDENT

The **DELIMIDENT** environment variable specifies that strings set off by double quotes are delimited identifiers.

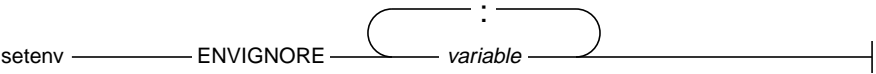
```
setenv _____ DELIMIDENT _____
```

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 nonalphabetical characters, but you cannot use them to specify storage identifiers that contain nonalphabetical characters. Note that database identifiers are names for database objects such as tables and columns, and storage identifiers are names for storage objects such as dbspaces and blobspaces.

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.

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:

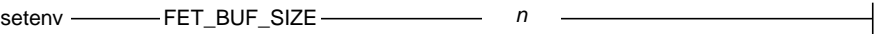
```
setenv ENVIGNORE DBPATH:DBMONEY
```

The common environment-configuration file is stored in **\$INFORMIXDIR/etc/informix.rc**. The private environment-configuration file is stored in the home directory of the user as **.informix**. For information on how to create or modify an environment-configuration file, see [“Setting Environment Variables in an Environment-Configuration File”](#) on page 3-4.

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 blobs. 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
```

INFORMIXC

The **INFORMIXC** environment variable specifies the name or pathname of the C compiler to be used to compile files generated by INFORMIX-ESQL/C. If **INFORMIXC** is not set, the default compiler is cc.

```
setenv ——— INFORMIXC ——— compiler ——— |
                                   pathname
```

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.

INFORMIXCONCSMCFG

The **INFORMIXCONCSMCFG** environment variable specifies the location of the **concsbm.cfg** file, which describes communications support modules

```
setenv INFORMIXCONCSMCFG pathname
```

pathname specifies the full pathname of the **concsbm.cfg** file.

For example, the following command specifies that the **concsbm.cfg** file is in **/usr/myfiles**.

```
setenv INFORMIXCONCSMCFG /usr/myfiles
```

You can also specify a different name for the file. The following command specifies a filename of **csmconfig**.

```
setenv INFORMIXCONCSMCFG /usr/myfiles/csmconfig
```

The default location of the **concsbm.cfg** file is **\$INFORMIXDIR/etc** file. For more information about communications support modules and the content of the **concsbm.cfg** file, refer to the [INFORMIX-Universal Server Administrator's Guide](#).

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 specified by the **INFORMIXCONTIME** environment variable.

```
setenv INFORMIXCONRETRY value
```

value represents the number of connection attempts to each server.

For example, set **INFORMIXCONRETRY** to three additional connection attempts (after the initial attempt) by entering the following command:

```
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 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.

```
setenv _____ INFORMIXCONTIME _____ value _____
```

value represents the minimum number of seconds spent in attempts to establish a connection to a server.

For example, set **INFORMIXCONTIME** to 60 seconds by entering the following command:

```
setenv INFORMIXCONTIME 60
```

If **INFORMIXCONTIME** is set to 60 and **INFORMIXCONRETRY** is set to 3, as shown in these examples, attempts to connect to the server (after the initial attempt at 0 seconds) will be 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 indicating 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 has been exceeded, but before the **INFORMIXCONRETRY** value has been reached.

INFORMIXDIR

The **INFORMIXDIR** environment variable specifies the directory that contains the subdirectories in which your product files are installed. You must always set **INFORMIXDIR**. If you have multiple versions of Universal 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 the [INFORMIX-Universal Server Installation Guide](#).

setenv `INFORMIXDIR` `pathname`

pathname is the directory path where the product files are installed.

Set the **INFORMIXDIR** environment variable to the desired installation directory by entering the following command:

```
setenv INFORMIXDIR /usr/informix
```

INFORMIXKEYTAB

The **INFORMIXKEYTAB** environment variable specifies the location of the keytab file that is used by the optional DCE-GSS communications support module.

setenv	_____	INFORMIXKEYTAB	_____	pathname
--------	-------	----------------	-------	----------

pathname specifies the full path and filename 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/keytab
```

For more information about the DCE-GSS communications support module, refer to the [INFORMIX-Universal Server Administrator's Guide](#).

INFORMIXOPCACHE

The **INFORMIXOPCACHE** environment variable lets you specify the size of the memory cache for the staging-area blobspace of the client application.

setenv	_____	INFORMIXOPCACHE	_____	kilobytes
--------	-------	-----------------	-------	-----------

kilobytes specifies the value you set for the optical memory cache.

You 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, INFORMIX-OnLine/Optical 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 a INFORMIX-Universal Server and can be either local or remote. You must always set **INFORMIXSERVER** before using an Informix product.

```
setenv INFORMIXSERVER dbservername
```

dbservername is the name of the default database server.

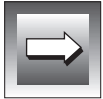
The value of **INFORMIXSERVER** must correspond to a valid *dbservername* entry in the **\$INFORMIXDIR/etc/sqlhosts** file on the computer running the application. The *dbservername* must be specified using lowercase characters and cannot exceed 18 characters for Universal Server. For example, specify the **coral** database server as the default for connection by entering 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: ***INFORMIXSERVER** must be set even if the application or DB-Access does not use implicit or explicit default connections.*





INFORMIXSHMBASE

The **INFORMIXSHMBASE** environment variable affects only client applications connected to Universal Server using the IPC shared-memory (**ipcsbm**) communication protocol.

Important: Resetting **INFORMIXSHMBASE** requires a thorough understanding of how the application uses memory. Normally you do not reset **INFORMIXSHMBASE**.

You 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 used by the application. If you do not set **INFORMIXSHMBASE**, the memory address of the communication segments defaults to an implementation-specific value such as 0x800000.

```
setenv _____ INFORMIXSHMBASE _____ value _____
```

value is used to calculate the memory address.

Universal 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 the [INFORMIX-Universal Server Administrator's Guide](#).

INFORMIXSQLHOSTS

The **INFORMIXSQLHOSTS** environment variable specifies the full pathname and filename of a file that contains connectivity information.

The file specified in the **INFORMIXSQLHOSTS** environment variable has the same format as the **\$INFORMIXDIR/etc/sqlhosts** file. For a description of the **\$INFORMIXDIR/etc/sqlhosts** file, see the [INFORMIX-Universal Server Administrator's Guide](#).

```
setenv _____ INFORMIXSQLHOSTS _____ pathname _____|
```

pathname specifies the full pathname and filename of the file that contains connectivity information.

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.

INFORMIXSTACKSIZE

INFORMIXSTACKSIZE specifies the stack size (in kilobytes) that Universal 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:

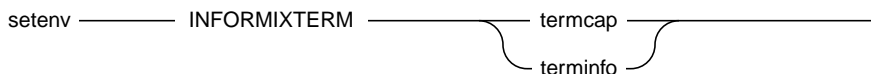
```
setenv INFORMIXSTACKSIZE 20
```

If **INFORMIXSTACKSIZE** is not set, the stack size is taken from the Universal 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 for setting this value, see the “[INFORMIX-Universal Server Administrator’s Guide](#).” If you incorrectly set the value of **INFORMIXSTACKSIZE**, it can cause Universal Server to crash.

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 border and graphics 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 supplied by Informix, 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 3-61](#).

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 Version 9.1 machine notes file for your product.

INF_ROLE_SEP

The `INF_ROLE_SEP` environment variable configures the security feature of role separation when Universal Server is installed. Role separation enforces separating administrative tasks that are performed by different people who are involved in running and auditing Universal Server.

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 _____|
```

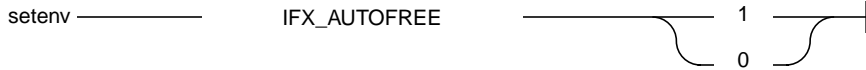
n is any positive integer.

See the [INFORMIX-Universal Server Trusted Facility Manual](#) to learn more about the security feature of role separation. See the [INFORMIX-Universal Server Installation Guide](#) to learn how to configure role separation when you install Universal Server.

IFX_AUTOFREE

If `IFX_AUTOFREE` is enabled, then memory allocated to every cursor in every thread will be automatically freed immediately upon the close of the cursor. In closing the cursor, users do not need to explicitly free server memory allocated to the cursor. This saves a network round-trip as no messages are sent to the server to free cursor memory.

When the cursor is automatically freed, its associated PREPARE statement is also freed and cannot be used to declare any new cursor thereafter.



- 1** IFX_AUTOFREE is enabled.
- 0** IFX_AUTOFREE is disabled.

If the environment variable is not set to any value, the feature is not enabled.

The SET AUTOFREE SQL syntax has the same functionality as the **IFX_AUTOFREE** environment variable. The **IFX_AUTOFREE** environment variable should be set before an application starts. The SET AUTOFREE syntax can be specified for a single cursor and overrides the setting of the environment variable for that cursor.



Important: The environment variable needs to be set before the application starts for the setting to take effect.

The SQL statement **SET DEFERRED_PREPARE** has the same functionality as the on (1) setting of this environment variable. The SQL statement overrides the setting of this environment variable.



Tip: If PREPARE/EXECUTE statements are used or if a DESCRIBE statement is executed before the first OPEN statement in the ESQL/C code when deferred prepare functionality is turned on, error is returned.

IFX_DEFERRED_PREPARE

The **IFX_DEFERRED_PREPARE** environment variable is set on the client side. It works primarily with dynamic ESQL/C cursors in applications which do a series of PREPARE/DECLARE/OPEN blocks of statements with no DESCRIBE statement following them. The setting on this environment variable optimizes the PREPARE statement.

setenv _____ IFX_DEFERRED_PREPARE _____ 1 _____
0 _____

If the **IFX_DEFERRED_PREPARE** variable is set to 1, the PREPARE statement in an ESQL/C code is not executed until the cursor has been declared on it and opened, thus optimizing the number of round-trip messages to the server. Setting of the variable to 1 enables every PREPARE statement in every thread in the application to be optimized.

Setting the value of the **IFX_DEFERRED_PREPARE** environment variable to 0 disables the optimization feature. If **IFX_DEFERRED_PREPARE** is not set, the optimization feature is not turned on.



Important: The environment variable needs to be set before the application starts for the setting to take effect.

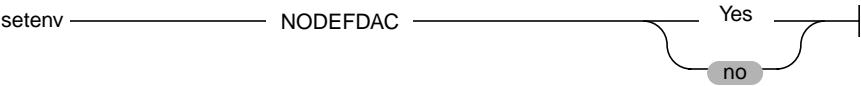
The SQL statement **SET DEFERRED_PREPARE** has the same functionality as the 'on' (1) setting of this environment variable. The SQL statement overrides the setting of this environment variable.



Tip: If PREPARE/EXECUTE statements are used or if a DESCRIBE statement is executed before the first OPEN statement in the ESQL/C code when deferred prepare functionality is turned on, an error is returned.

NODEFDAC

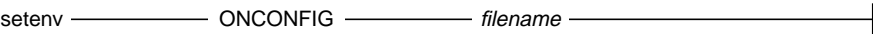
When the **NODEFDAC** environment variable is set to *yes*, it prevents default table privileges (Select, Insert, Update, and Delete) and routine privileges from being granted to PUBLIC when a new table or routine is created in a database that is not ANSI compliant. If you do not set the **NODEFDAC** variable, it is, by default, set to *no*.



- yes** prevents default table and routine 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 stored procedure from being granted to PUBLIC when the stored procedure is created in owner mode.
- no** allows default table privileges to be granted to PUBLIC. Also allows the Execute privilege on a new stored procedure to be granted to PUBLIC when the stored procedure is created in owner mode.

ONCONFIG

The **ONCONFIG** environment variable specifies a file that holds configuration parameters for Universal Server. This file is read as input during the initialization procedure.



filename is the name of a file in **\$INFORMIXDIR/etc** that contains Universal Server configuration parameters.

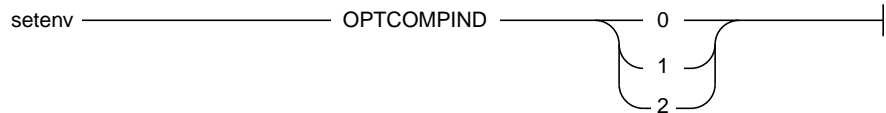
Prepare the ONCONFIG file by making a copy of the **onconfig.std** file and modifying the copy. Informix recommends that you name the ONCONFIG file so it can easily be related to a specific Universal Server database server. If you have multiple instances of Universal Server, each instance *must* have its own uniquely named ONCONFIG file.

If you do not set the **ONCONFIG** environment variable, the default filename is **onconfig**.

For more information, see the [INFORMIX-Universal Server Administrator's Guide](#).

OPTCOMPIND

You can set the **OPTCOMPIND** environment variable so that the optimizer can select the appropriate join method. The **OPTCOMPIND** environment variable applies only to Universal Server.



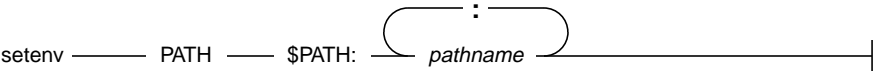
- 0 A nested-loop join is preferred, where possible, over a sort-merge join or a hash join.
- 1 When the transaction isolation mode is *not* Repeatable Read, the optimizer behaves as in setting 2; otherwise, the optimizer behaves as in setting 0.
- 2 Nested-loop joins are not necessarily preferred. The optimizer bases its decision purely on costs, regardless of transaction isolation mode.

When the **OPTCOMPIND** environment variable is not set, Universal 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 [INFORMIX-Universal Server Administrator's Guide](#). For more information on the different join methods used by the optimizer, see the [INFORMIX-Universal Server Performance Guide](#).

PATH

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.



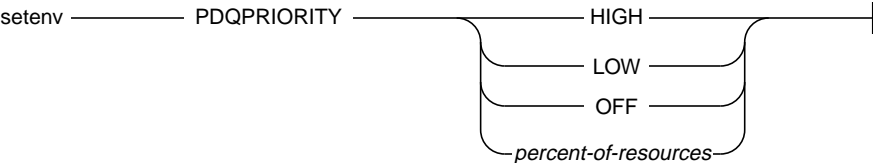
pathname specifies the search path for the executables.

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 the Setting of an Environment Variable” on page 3-8](#).

PDQPRIORITY

The **PDQPRIORITY** environment variable determines the degree of parallelism used by Universal Server. **PDQPRIORITY** affects how Universal Server allocates resources, including memory, processors, and disk reads.



HIGH	When Universal 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, but no other parallelism is used.
OFF	PDQ processing is turned off.
<i>percent-of-resources</i>	<p>An integer between 0 and 100 that indicates a query priority level. The higher the number, the more resources Universal Server uses.</p> <p>Two values have special meaning:</p> <p>0 is equivalent to the symbolic value of OFF.</p> <p>1 is equivalent to the symbolic value of LOW.</p>

When the **PDQPRIORITY** environment variable is not set, the default value is OFF.

When the environment variable is set to HIGH, the database server determines an appropriate value to use for **PDQPRIORITY** based on several criteria, including the number of available processors, the fragmentation of tables queried, the complexity of the query, and so on.

Usually the more resources Universal Server uses, the better its performance for a given query, but using too many resources can cause contention among the resources and also take away resources from other queries, resulting in degraded performance.

An application can override the setting of the environment variable when it issues the SQL statement SET PDQPRIORITY, which is described in the [Informix Guide to SQL: Syntax](#).

PLCONFIG

The **PLCONFIG** environment variable specifies the name of the configuration file that the High-Performance Loader uses. This configuration file must reside in the **\$INFORMIXDIR/etc** directory. If the **PLCONFIG** environment variable is not set, the default configuration file is the **\$INFORMIXDIR/etc/plconfig** file.

```
setenv _____ PLCONFIG _____ filename _____
```

filename specifies the simple filename of the configuration file to be used by the High-Performance Loader.

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
```

PSORT_DBTEMP

The **PSORT_DBTEMP** environment variable specifies a directory or directories where the Universal Server writes the temporary files it uses when performing a sort.

Universal Server uses the directory specified by **PSORT_DBTEMP** even if the environment variable **PSORT_NPROCS** is not set.

```
setenv _____ PSORT_DBTEMP _____ : _____  
pathname
```

pathname is the name of the UNIX directory used for intermediate writes during a sort.

Set the **PSORT_DBTEMP** environment variable to specify the directory (for example, **/usr/leif/tempsort**) by entering the following command:

```
setenv PSORT_DBTEMP /usr/leif/tempsort
```

For maximum performance, specify directories that reside in file systems on different disks.

You also might 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 3-34](#).

For additional information about the **PSORT_DBTEMP** environment variable, see the *INFORMIX-Universal Server Administrator's Guide* as well as the *INFORMIX-Universal Server Performance Guide*.

PSORT_NPROCS

The **PSORT_NPROCS** environment variable enables Universal Server to improve the performance of the parallel-process sorting package by allocating more threads for sorting. Before the sorting package performs a parallel sort, make sure that Universal Server has enough memory for the sort.

```
setenv PSORT_NPROCS threads
```

threads specifies the maximum number of threads to be used to sort a query. The maximum value of *threads* is 10.

Use the following command to set the **PSORT_NPROCS** environment variable to 4:

```
setenv PSORT_NPROCS 4
```

To maximize the effectiveness of the parallel sort, set **PSORT_NPROCS** to the number of available processors in the hardware.

You can disable parallel sorting by entering the following command:

```
unsetenv PSORT_NPROCS
```



Tip: If the **PDQPRIORITY** environment variable is not set, Universal Server allocates the minimum amount of memory to sorts. 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, Universal 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. Universal Server uses one thread for the sort.

If **PSORT_NPROCS** is set to 0, Universal 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 the [INFORMIX-Universal Server Administrator's Guide](#) as well as the [INFORMIX-Universal Server Performance Guide](#).

TERM

The UNIX **TERM** environment variable is used for terminal handling. It enables DB-Access to recognize and communicate with the terminal you are using.

```
setenv _____ TERM _____ type _____
```

type specifies the terminal type.

The terminal type specified in the **TERM** setting must correspond to an entry in the **termcap** file or **terminfo** directory. Before you can set the **TERM** environment variable, you must obtain the code for your terminal from the DBA.

For example, to specify the vt100 terminal, set the **TERM** environment variable by entering the following command:

```
setenv TERM vt100
```

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.

```
setenv _____ TERMCAP _____ pathname _____
```

pathname specifies the location of the **termcap** file.

The **termcap** file contains a list of various types of terminals and their characteristics. For example, you can provide DB-Access terminal-handling information, which is specified in the **/usr/informix/etc/termcap** file, by entering the following command:

```
setenv TERMCAP /usr/informix/etc/termcap
```

You can use any of the following settings for **TERMCAP**. They are used in the following order:

1. The **termcap** file that you create
2. The **termcap** file supplied by Informix (that is, **\$INFORMIXDIR/etc/termcap**)
3. The operating-system **termcap** file (that is, **/etc/termcap**)

If you set the **TERMCAP** environment variable, be sure that the **INFORMIXTERM** environment variable is set to the default, **termcap**.

If you do not set the **TERMCAP** environment variable, the system file (that is, **/etc/termcap**) is used by default.

TERMINFO

The **TERMINFO** environment variable is used for terminal handling. It is supported only on platforms that provide full support for the **terminfo** libraries provided by System V and Solaris UNIX systems.

```
setenv ———— TERMINFO ———— /usr/lib/terminfo ————|
```

TERMINFO tells DB-Access to communicate with the **terminfo** directory instead of the **termcap** file. The **terminfo** directory has subdirectories that contain files that pertain to terminals and their characteristics.

Set **TERMINFO** by entering the following command:

```
setenv TERMINFO /usr/lib/terminfo
```

If you set the **TERMINFO** environment variable, you must also set the **INFORMIXTERM** environment variable to **terminfo**.

THREADLIB

You 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.



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.

Index of Environment Variables

Figure 3-2 provides an overview of the uses for the various Informix and UNIX environment variables supported in Version 9.1. It serves as an index to general topics and lists the related environment variables and the pages where the environment variables are introduced.

Figure 3-2
Environment Variables Used with Informix Products

Topic	Environment Variables	Page
ANSI compliance	DBANSIWARN	3-15
Simple-large-object buffer	DBBLOBBUF	3-17
C compiler	INFORMIXC	3-42

(1 of 8)

Topic	Environment Variables	Page
C compiler: processing of multibyte characters	CC8BITLEVEL	Guide to GLS Functionality
Client locale	CLIENT_LOCALE	Guide to GLS Functionality
Client/server	INFORMIXSERVER	3-47
	INFORMIXSHMBASE	3-48
	INFORMIXSTACKSIZE	3-49
	CLIENT_LOCALE	Guide to GLS Functionality
	DB_LOCALE	Guide to GLS Functionality
	SERVER_LOCALE	Guide to GLS Functionality
Code-set conversion	CLIENT_LOCALE	Guide to GLS Functionality
	DB_LOCALE	Guide to GLS Functionality
Communications Support Services	INFORMIXCONCSMCFG	3-43
	INFORMIXKEYTAB	3-46
Compilation: ESQL/C	THREADLIB	3-63
Compiler	CC8BITLEVEL	Guide to GLS Functionality
	INFORMIXC	3-42
Configuration file: ignore variables	ENVIGNORE	3-41
Configuration file: ON-Archive	ARC_DEFAULT	3-14
Configuration file: Universal Server	ONCONFIG	3-54
Configuration file: tctermcap	ARC_KEYPAD	3-14

(2 of 8)

Topic	Environment Variables	Page
Connecting	INFORMIXCONRETRY	3-45
	INFORMIXSERVER	3-47
	INFORMIXSQLHOSTS	3-49
Data distributions	DBUPSPACE	3-39
Database locale	DB_LOCALE	Guide to GLS Functionality
Database server	INFORMIXSERVER	3-47
	SERVER_LOCALE	Guide to GLS Functionality
Date and time values	DBCENTURY	3-18
	DBDATE	3-21, Guide to GLS Functionality
	GL_DATE	Guide to GLS Functionality
	GL_DATETIME	Guide to GLS Functionality
	DBTIME	3-36
Delimited identifiers	DELIMIDENT	3-40
Disk space	DBUPSPACE	3-39
Editor	DBEDIT	3-24
ESQL/C: C compiler	INFORMIXC	3-42
ESQL/C: DATETIME formatting	DBTIME	3-36
ESQL/C: delimited identifiers	DELIMIDENT	3-40
ESQL/C: multibyte filter	ESQLMF	Guide to GLS Functionality

(3 of 8)

Topic	Environment Variables	Page
ESQL/C: multibyte identifiers	CLIENT_LOCALE	Guide to GLS Functionality
ESQL/COBOL: DATETIME formatting	DBTIME	3-36
ESQL/COBOL: delimited identifiers	DELIMIDENT	3-40
ESQL/COBOL: multibyte identifiers	CLIENT_LOCALE	Guide to GLS Functionality
Executable programs	PATH	3-56
Fetch buffer size	FET_BUF_SIZE	3-41
Filenames: multibyte	GLS8BITSYS	Guide to GLS Functionality
Files: field delimiter	DBDELIMITER	3-24
Files: installation	INFORMIXDIR	3-45
Files: locale	CLIENT_LOCALE	Guide to GLS Functionality
	DB_LOCALE	Guide to GLS Functionality
	SERVER_LOCALE	Guide to GLS Functionality
Files: message	DBLANG	3-26
Files: temporary (Universal Server)	DBSPACETEMP	3-34
Files: temporary sorting	PSORT_DBTEMP	3-58
Files: termcap , terminfo	INFORMIXTERM	3-50
	TERM	3-61
	TERMCAP	3-61
	TERMINFO	3-62
High-Performance Loader	DBONPLOAD	3-29

Topic	Environment Variables	Page
	PLCONFIG	3-58
Identifiers: delimited	DELIMIDENT	3-40
Identifiers: multibyte characters	CLIENT_LOCALE	Guide to GLS Functionality
	ESQLMF	Guide to GLS Functionality
INFORMIX-OnLine/Optical	INFORMIXOPCACHE	3-46
Installation	INFORMIXDIR	3-45
	PATH	3-56
Language environment	DBLANG	3-26
Locale	CLIENT_LOCALE	Guide to GLS Functionality
	DB_LOCALE	Guide to GLS Functionality
	SERVER_LOCALE	Guide to GLS Functionality
Message files	DBLANG	3-26
Money values	DBMONEY	3-27, Guide to GLS Functionality
Multibyte characters	CLIENT_LOCALE	Guide to GLS Functionality
	DB_LOCALE	Guide to GLS Functionality
	SERVER_LOCALE	Guide to GLS Functionality
Multibyte filter	ESQLMF	Guide to GLS Functionality
Multithreaded applications	THREADLIB	3-63

(5 of 8)

Topic	Environment Variables	Page
Nondefault locale	CLIENT_LOCALE	Guide to GLS Functionality
	DB_LOCALE	Guide to GLS Functionality
	SERVER_LOCALE	Guide to GLS Functionality
Universal Server: archiving	ARC_DEFAULT	3-14
	ARC_KEYPAD	3-14
	DBREMOTECMD	3-33
Universal Server: configuration parameters	ONCONFIG	3-54
Universal Server: parallel sorting	PSORT_DBTEMP	3-58
	PSORT_NPROCS	3-59
Universal Server: role separation	INF_ROLE_SEP	3-51
Universal Server: shared memory	INFORMIXSHMBASE	3-48
Universal Server: stacksize	INFORMIXSTACKSIZE	3-49
Universal Server: tape management	ARC_DEFAULT	3-14
	ARC_KEYPAD	3-14
	DBREMOTECMD	3-33
Universal Server: temporary tables, sort files	DBSPACETEMP	3-34
Pathname: for C compiler	INFORMIXC	3-42
Pathname: for database files	DBPATH	3-29
Pathname: for executable programs	PATH	3-56
Pathname: for installation	INFORMIXDIR	3-45
Pathname: for message files	DBLANG	3-26

(6 of 8)

Topic	Environment Variables	Page
Pathname: for parallel sorting	PSORT_DBTEMP	3-58
Pathname: for remote shell	DBREMOTECMD	3-33
Printing	DBPRINT	3-32
Privileges	NODEFDAC	3-51
Program: printing	DBPRINT	3-32
Remote shell	DBREMOTECMD	3-33
Role separation	INF_ROLE_SEP	3-51
Routine: DATETIME formatting	DBTIME	3-36
Server	See <i>Database server</i> .	
Server locale	SERVER_LOCALE	Guide to GLS Functionality
Shared memory	INFORMIXSHMBASE	3-48
Shell: remote	DBREMOTECMD	3-33
Shell: search path	PATH	3-56
Sorting	PSORT_DBTEMP	3-58
	PSORT_NPROCS	3-59
	DBSPACETEMP	3-34
SQL statement: CONNECT	INFORMIXSERVER	3-47
SQL statement: editing	DBEDIT	3-24
SQL statement: LOAD, UNLOAD	DBDELIMITER	3-24
SQL statement: UPDATE STATISTICS	DBUPSPACE	3-39
Stacksize	INFORMIXSTACKSIZE	3-49
Tables: temporary (Universal Server)	DBSPACETEMP	3-34
Temporary files	PSORT_DBTEMP	3-58

(7 of 8)

Topic	Environment Variables	Page
Temporary tables	DBSPACETEMP	3-34
Terminal handling	INFORMIXTERM	3-50
	TERM	3-61
	TERMCAP	3-61
	TERMINFO	3-62
Utilities: DB-Access	DBDELIMITER	3-24
	DBEDIT	3-24
	INFORMIXTERM	3-50
	DBFLTMASK	3-25
Utilities: dbexport	DBDELIMITER	3-24
Utilities: ON-Archive	ARC_DEFAULT	3-14
	ARC_KEYPAD	3-14
	DBREMOTECMD	3-33
Values: date and time	DBDATE	3-21 , Guide to GLS Functionality
	DBTIME	3-36
Values: money	DBMONEY	3-27
Variables: overriding	ENVIGNORE	3-41

The stores7 Database

The **stores7** database contains a set of tables that describe an imaginary business. The examples in the [Informix Guide to SQL: Syntax](#) and [Informix Guide to SQL: Tutorial](#) are based on this database. The **stores7** database is not ANSI-compliant. Information on creating the **stores7** database appears in “Demonstration Database” in the Introduction of this manual.

This appendix contains the following sections:

- The first section describes the structure of the tables in the **stores7** 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 shows a graphic map of the tables in the **stores7** database and indicates the relationships between columns.
- The third section describes the primary-foreign key relationships between columns in tables.
- The final section shows the data contained in each table of the **stores7** database.

Structure of the Tables

The **stores7** database contains information about a fictitious sporting-goods distributor that services stores in the western United States. This database includes the following tables:

- **customer**
- **orders**
- **items**
- **stock**
- **catalog**
- **cust_calls**
- **call_type**
- **manufact**
- **state**


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. The columns of the **customer** table are shown in [Figure A-1 on page A-3](#).

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. The columns of the **orders** table are shown in Figure A-2.


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. The columns of the items table are shown in Figure A-3.

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 > 0)
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 racer goggles from two manufacturers and running shoes from six manufacturers.

The stock table is a catalog of the items sold by the distributor. The columns of the **stock** table are shown in Figure A-4.


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. The columns of the **catalog** table are shown in Figure A-5.

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. The columns of the **cust_calls** table are shown in Figure A-6.


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

Figure A-6
The cust_calls Table

The call_type Table

The call codes associated with customer calls are stored in the **call_type** table. The columns of the **call_type** table are shown in Figure A-7.


Column Name	Data Type	Description
 call_code	CHAR(1)	call code
call_descr	CHAR (30)	description of call type

Figure A-7
The call_type Table

The manufact Table

Information about the nine manufacturers whose sporting goods are handled by the distributor is stored in the **manufact** table. The columns of the **manufact** table are shown in Figure A-8.


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

Figure A-8
The manufact Table

The state Table

The **state** table contains the names and postal abbreviations for the 50 states of the United States. The columns of the **state** table are shown in Figure A-9.


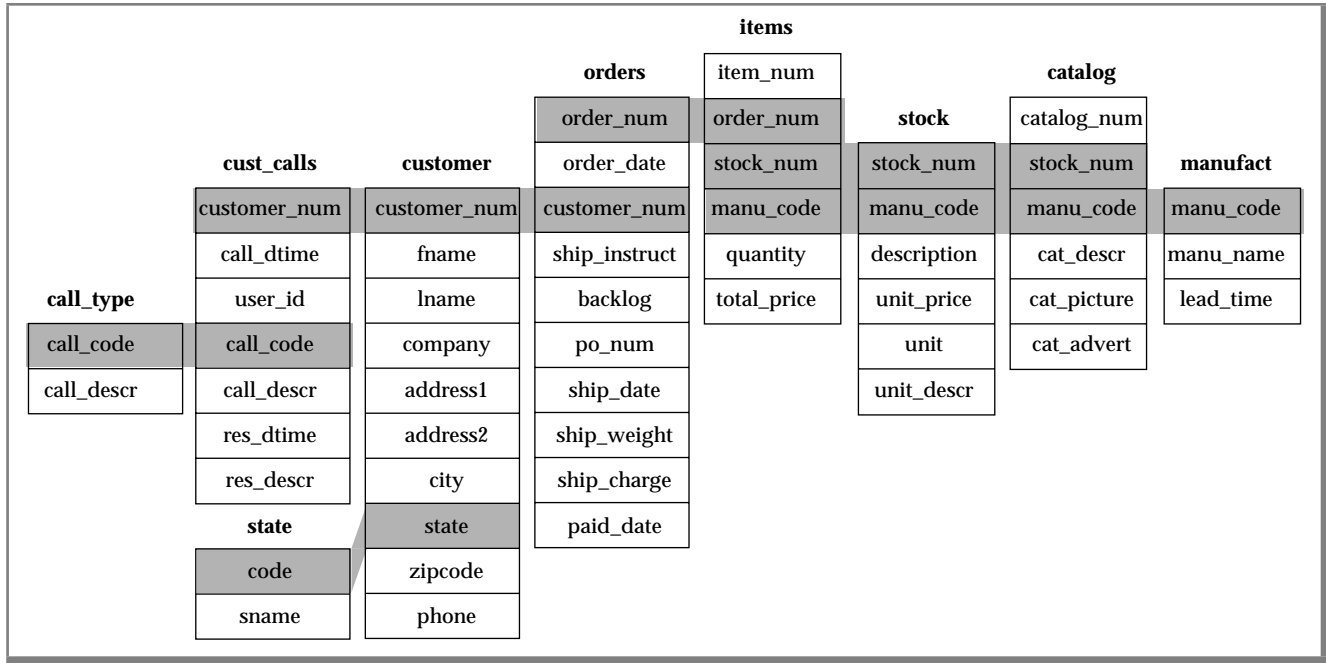
Name	Type	Description
 code	CHAR(2)	state code
sname	CHAR(15)	state name

Figure A-9
The state Table

The stores7 Database Map

Figure A-10 on page A-8 displays the joins in the **stores7** database. The lines connecting a column in one table to the same column in another table indicate the relationships, or *joins*, between tables.

Figure A-10
Joins in the stores7 Database



Primary-Foreign Key Relationships

The tables of the **stores7** database are linked by the primary-foreign key relationships shown in Figure A-10 on page A-8 and 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 identifying a customer, along with 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. This relationship is shown in Figure A-11.

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	05/20/1994	104
1002	05/21/1994	101
1003	05/22/1994	104
1004	05/22/1994	106

Figure A-11
Tables Joined by the
customer_num
Column

According to [Figure A-11 on page A-9](#), 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	05/20/1994	104	
1002	05/21/1994	101	
1003	05/22/1994	104	

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

Figure A-12
*Tables Joined by the
order_num Column*

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. This is illustrated in Figure A-13.

items Table (detail)

item_num	order_num	stock_num	manu_code
1	1001	1	HRO
1	1002	4	HSK
2	1002	3	HSK
1	1003	9	ANZ
2	1003	8	ANZ
3	1003	5	ANZ
1	1004	1	HRO

stock Table (detail)

stock_num	manu_code	description
1	HRO	baseball gloves
1	HSK	baseball gloves
1	SMT	baseball gloves

Figure A-13
Tables Joined by the
stock_num and
manu_code
Columns

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 Joined by the
stock_num and
manu_code
Columns

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. This relationship is illustrated in Figure A-15.

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 Joined by the
manu_code Column*

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. This relationship is illustrated in Figure A-16.

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	1994-06-12 08:20	maryj
127	1994-07-31 14:30	maryj
116	1993-11-28 13:34	mannyh
116	1993-12-21 11:24	mannyh

Figure A-16
*Tables Joined by the
customer_num
Column*

The call_type and cust_calls Table

The **call_type** and **cust_calls** tables are joined by the **call_code** column. The same call code can appear in more than one row of the **cust_calls** table because many customers can have the same *type* of problem. In the **cust_calls** table, the **call_code** column is a foreign key that references the **call_code** column in the **call_type** table. This relationship is illustrated in Figure A-17.

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	1994-06-12 08:20	D
127	1994-07-31 14:30	I
116	1993-11-28 13:34	I
116	1993-12-21 11:24	I

Figure A-17
Tables Joined by the
call_code Column

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 Joined by the
state/code Column

Data in the stores7 Database

The following tables display the data in the **stores7** 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 St		San Francisco	CA	94117	415-822-1289
103	Philip	Currie	Phil's Sports	654 Poplar	P. O. Box 3498	Palo Alto	CA	94303	415-328-4543
104	Anthony	Higgins	Play Ball!	East Shopping Cntr.	422 Bay Road	Redwood City	CA	94026	415-368-1100
105	Raymond	Vector	Los Altos Sports	1899 La Loma Drive		Los Altos	CA	94022	415-776-3249
106	George	Watson	Watson & Son	1143 Carver Place		Mountain View	CA	94063	415-389-8789
107	Charles	Ream	Athletic Supplies	41 Jordan Avenue		Palo Alto	CA	94304	415-356-9876
108	Donald	Quinn	Quinn's Sports	587 Alvarado		Redwood City	CA	94063	415-544-8729
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	415-743-3611
111	Frances	Keyes	Sports Center	3199 Sterling Court		Sunnyvale	CA	94085	408-277-7245
112	Margaret	Lawson	Runners & Others	234 Wyandotte Way		Los Altos	CA	94022	415-887-7235
113	Lana	Beatty	Sportstown	654 Oak Grove		Menlo Park	CA	94025	415-356-9982
114	Frank	Albertson	Sporting Place	947 Waverly Place		Redwood City	CA	94062	415-886-6677
115	Alfred	Grant	Gold Medal Sports	776 Gary Avenue		Menlo Park	CA	94025	415-356-1123
116	Jean	Parmelee	Olympic City	1104 Spinosa Drive		Mountain View	CA	94040	415-534-8822
117	Arnold	Sipes	Kids Korner	850 Lytton Court		Redwood City	CA	94063	415-245-4578
118	Dick	Baxter	Blue Ribbon Sports	5427 College		Oakland	CA	94609	415-655-0011

(1 of 2)

A-18
INFORMIX

customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone
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 Rd	Suite 1020	Jacksonville	FL	32256	904-823-4239
124	Chris	Putnum	Putnum's Putters	4715 S.E. Adams Blvd	Suite 909C	Bartlesville	OK	74006	918-355-2074
125	James	Henry	Total Fitness Sports	1450 Commonwealth Ave.		Brighton	MA	02135	617-232-4159
126	Eileen	Neelie	Neelie's Discount Sports	2539 South Utica St		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

(2 of 2)

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	05/20/1994	104	express	n	B77836	06/01/1994	20.40	10.00	07/22/1994
1002	05/21/1994	101	PO on box; deliver back door only	n	9270	05/26/1994	50.60	15.30	06/03/1994
1003	05/22/1994	104	express	n	B77890	05/23/1994	35.60	10.80	06/14/1994
1004	05/22/1994	106	ring bell twice	y	8006	05/30/1994	95.80	19.20	
1005	05/24/1994	116	call before delivery	n	2865	06/09/1994	80.80	16.20	06/21/1994
1006	05/30/1994	112	after 10AM	y	Q13557		70.80	14.20	
1007	05/31/1994	117		n	278693	06/05/1994	125.90	25.20	
1008	06/07/1994	110	closed Monday	y	LZ230	07/06/1994	45.60	13.80	07/21/1994
1009	06/14/1994	111	door next to grocery	n	4745	06/21/1994	20.40	10.00	08/21/1994
1010	06/17/1994	115	deliver 776 King St. if no answer	n	429Q	06/29/1994	40.60	12.30	08/22/1994
1011	06/18/1994	104	express	n	B77897	07/03/1994	10.40	5.00	08/29/1994
1012	06/18/1994	117		n	278701	06/29/1994	70.80	14.20	
1013	06/22/1994	104	express	n	B77930	07/10/1994	60.80	12.20	07/31/1994
1014	06/25/1994	106	ring bell, kick door loudly	n	8052	07/03/1994	40.60	12.30	07/10/1994
1015	06/27/1994	110	closed Mondays	n	MA003	07/16/1994	20.60	6.30	08/31/1994

(1 of 2)

INFORMIX

order_num	order_date	customer_num	ship_instruct	backlog	po_num	ship_date	ship_weight	ship_charge	paid_date
1016	06/29/1994	119	delivery entrance off Camp St.	n	PC6782	07/12/1994	35.00	11.80	
1017	07/09/1994	120	North side of clubhouse	n	DM3543 31	07/13/1994	60.00	18.00	
1018	07/10/1994	121	SW corner of Biltmore Mall	n	S22942	07/13/1994	70.50	20.00	08/06/1994
1019	07/11/1994	122	closed til noon Mondays	n	Z55709	07/16/1994	90.00	23.00	08/06/1994
1020	07/11/1994	123	express	n	W2286	07/16/1994	14.00	8.50	09/20/1994
1021	07/23/1994	124	ask for Elaine	n	C3288	07/25/1994	40.00	12.00	08/22/1994
1022	07/24/1994	126	express	n	W9925	07/30/1994	15.00	13.00	09/02/1994
1023	07/24/1994	127	no deliveries after 3 p.m.	n	KF2961	07/30/1994	60.00	18.00	08/22/1994

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	racers 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
10010	5	SMT	Aluminum frame. Synthetic strings.	<BYTE value>	Mid-Sized Racquet For the Improving Player

(1 of 11)

A-30 INFORMIX	catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
	10011	5	ANZ	Wood frame, cat-gut strings.	<BYTE value>	Antique Replica of Classic Wooden Racquet Built with Cat-Gut Strings
	10012	6	SMT	Soft yellow color for easy visibility in sunlight or artificial light.	<BYTE value>	High-Visibility Tennis, Day or Night
	10013	6	ANZ	Pro-core. Available in neon yellow, green, and pink.	<BYTE value>	Durable Construction Coupled with the Brightest Colors Available
	10014	7	HRO	Indoor. Classic NBA style. Brown leather.	<BYTE value>	Long-Life Basketballs for Indoor Gymnasiums
	10015	8	ANZ	Indoor. Finest leather. Professional quality.	<BYTE value>	Professional Volleyballs for Indoor Competitions
	10016	9	ANZ	Steel eyelets. Nylon cording. Double-stitched. Sanctioned by the National Athletic Congress.	<BYTE value>	Sanctioned Volleyball Netting for Indoor Professional and Collegiate Competition
	10017	101	PRC	Reinforced, hand-finished tubular. Polyurethane belted. Effective against punctures. Mixed tread for super wear and road grip.	<BYTE value>	Ultimate in Puncture Protection, Tires Designed for In-City Riding
	10018	101	SHM	Durable nylon casing with butyl tube for superior air retention. Center-ribbed tread with herringbone side. Coated sidewalls resist abrasion.	<BYTE value>	The Perfect Tire for Club Rides or Training
	10019	102	SHM	Thrust bearing and coated pivot washer/ spring sleeve for smooth action. Slotted levers with soft gum hoods. Two-tone paint treatment. Set includes calipers, levers, and cables.	<BYTE value>	Thrust-Bearing and Spring-Sleeve Brake Set Guarantees Smooth Action

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10020	102	PRC	Computer-aided design with low-profile pads. Cold-forged alloy calipers and beefy caliper bushing. Aero levers. Set includes calipers, levers, and cables.	<BYTE value>	Computer Design Delivers Rigid Yet Vibration-Free Brakes
10021	103	PRC	Compact leading-action design enhances shifting. Deep cage for super-small granny gears. Extra strong construction to resist off-road abuse.	<BYTE value>	Climb Any Mountain: ProCycle's Front Derailleur Adds Finesse to Your ATB
10022	104	PRC	Floating trapezoid geometry with extra thick parallelogram arms. 100-tooth capacity. Optimum alignment with any freewheel.	<BYTE value>	Computer-Aided Design Engineers 100-Tooth Capacity Into ProCycle's Rear Derailleur
10023	105	PRC	Front wheels laced with 15g spokes in a 3-cross pattern. Rear wheels laced with 14g spikes in a 3-cross pattern.	<BYTE value>	Durable Training Wheels That Hold True Under Toughest Conditions
10024	105	SHM	Polished alloy. Sealed-bearing, quick-release hubs. Double-buttet. 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 11)

A-32 INFORMIX	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

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10031	110	ANZ	No buckle so no plastic touches your chin. Meets both ANSI and Snell standards for impact protection. 7.5 oz. Lycra cover.	<BYTE value>	Minimum Chin Contact, Feather-Light, Maximum Protection Helmet
10032	110	SHM	Dense outer layer combines with softer inner layer to eliminate the mesh cover, no snagging on brush. Meets both ANSI and Snell standards for impact protection. 8.0 oz.	<BYTE value>	Mountain Bike Helmet: Smooth Cover Eliminates the Worry of Brush Snags But Delivers Maximum Protection
10033	110	HRO	Newest ultralight helmet uses plastic shell. Largest ventilation channels of any helmet on the market. 8.5 oz.	<BYTE value>	Lightweight Plastic with Vents Assures Cool Comfort Without Sacrificing Protection
10034	110	HSK	Aerodynamic (teardrop) helmet covered with anti-drag fabric. Credited with shaving 2 seconds/mile from winner's time in Tour de France time-trial. 7.5 oz.	<BYTE value>	Teardrop Design Used by Yellow Jerseys, You Can Time the Difference
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-buttressed 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

(5 of 11)

A-34 INFORMIX	catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
	10037	113	SHM	Ultra-lightweight. Racing frame geometry built for aerodynamic handlebars. Cantilever brakes. Index shifting. High-performance gearing. Quick-release hubs. Disk wheels. Bladed spokes.	<BYTE value>	Designed for the Serious Competitor, The Complete Racing Machine
	10038	114	PRC	Padded leather palm and stretch mesh merged with terry back; Available in tan, black, and cream. Sizes S, M, L, XL.	<BYTE value>	Riding Gloves For Comfort and Protection
	10039	201	NKL	Designed for comfort and stability. Available in white & blue or white & brown. Specify size.	<BYTE value>	Full-Comfort, Long-Wearing Golf Shoes for Men and Women
	10040	201	ANZ	Guaranteed waterproof. Full leather upper. Available in white, bone, brown, green, and blue. Specify size.	<BYTE value>	Waterproof Protection Ensures Maximum Comfort and Durability In All Climates
	10041	201	KAR	Leather and leather mesh for maximum ventilation. Waterproof lining to keep feet dry. Available in white & 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

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10044	203	NKL	Set of eight irons includes 3 through 9 irons and pitching wedge. Originally priced at \$489.00.	<BYTE value>	Set of Irons Available From Factory at Tremendous Savings: Discontinued Line
10045	204	KAR	Ideally balanced for optimum control. Nylon-covered shaft.	<BYTE value>	High-Quality Beginning Set of Irons Appropriate for High School Competitions
10046	205	NKL	Fluorescent yellow.	<BYTE value>	Long Drive Golf Balls: Fluorescent Yellow
10047	205	ANZ	White only.	<BYTE value>	Long Drive Golf Balls: White
10048	205	HRO	Combination fluorescent yellow and standard white.	<BYTE value>	HiFlier Golf Balls: Case Includes Fluorescent Yellow and Standard White
10049	301	NKL	Super shock-absorbing gel pads disperse vertical energy into a horizontal plane for extraordinary cushioned comfort. Great motion control. Men's only. Specify size.	<BYTE value>	Maximum Protection For High-Mileage Runners
10050	301	HRO	Engineered for serious training with exceptional stability. Fabulous shock absorption. Great durability. Specify men's/women's, size.	<BYTE value>	Pronators and Supinators Take Heart: A Serious Training Shoe For Runners Who Need Motion Control

(7 of 11)

A-36 INFORMIX	catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
	10051	301	SHM	For runners who log heavy miles and need a durable, supportive, stable platform. Mesh/synthetic upper gives excellent moisture dissipation. Stability system uses rear antipronation platform and forefoot control plate for extended protection during high-intensity training. Specify men's/women's size.	<BYTE value>	The Training Shoe Engineered for Marathoners and Ultra-Distance Runners
	10052	301	PRC	Supportive, stable racing flat. Plenty of forefoot cushioning with added motion control. Women's only. D widths available. Specify size.	<BYTE value>	A Woman's Racing Flat That Combines Extra Forefoot Protection With a Slender Heel
	10053	301	KAR	Anatomical last holds your foot firmly in place. Feather-weight cushioning delivers the responsiveness of a racing flat. Specify men's/women's size.	<BYTE value>	Durable Training Flat That Can Carry You Through Marathon Miles
	10054	301	ANZ	Cantilever sole provides shock absorption and energy rebound. Positive traction shoe with ample toe box. Ideal for runners who need a wide shoe. Available in men's and women's. Specify size.	<BYTE value>	Motion Control, Protection, and Extra Toebox Room
	10055	302	KAR	Reusable ice pack with velcro strap. For general use. Velcro strap allows easy application to arms or legs.	<BYTE value>	Finally, An Ice Pack for Achilles Injuries and Shin Splints that You Can Take to the Office

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
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
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. Light-weight model.	<BYTE value>	Consider a Touring Vacation For the Entire Family: A Light-weight, Touring Tandem for Parent and Child

(9 of 11)

A-38 INFORMIX	catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
	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
	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

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
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

(11 of 11)

cust_calls Table

customer_num	call_dtime	user_id	call_code	call_descr	res_dtime	res_descr
106	1994-06-12 8:20	maryj	D	Order was received, but two of the cans of ANZ tennis balls within the case were empty.	1994-06-12 8:25	Authorized credit for two cans to customer, issued apology. Called ANZ buyer to report the QA problem.
110	1994-07-07 10:24	richc	L	Order placed one month ago (6/7) not received.	1994-07-07 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	1994-07-01 15:00	richc	B	Bill does not reflect credit from previous order.	1994-07-02 8:21	Spoke with Jane Akant in Finance. She found the error and is sending new bill to customer.

(1 of 2)

customer_num	call_dtime	user_id	call_code	call_descr	res_dtime	res_descr
121	1994-07-10 14:05	maryj	O	Customer likes our merchandise. Requests that we stock more types of infant joggers. Will call back to place order.	1994-07-10 14:06	Sent note to marketing group of interest in infant joggers.
127	1994-07-31 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	1993-11-28 13:34	manny n	I	Received plain white swim caps (313 ANZ) instead of navy with team logo (313 SHM).	1993-11-28 16:47	Shipping found correct case in warehouse and express mailed it in time for swim meet.
116	1993-12-21 11:24	manny n	I	Second complaint from this customer! Received two cases right-handed outfielder gloves (1 HRO) instead of one case lefties.	1993-12-27 08:19	Memo to shipping (Ava Brown) to send case of left-handed gloves, pick up wrong case; memo to billing requesting 5% discount to placate customer due to second offense and lateness of resolution because of holiday.

manufact Table

manu_code	manu_name	lead_time
ANZ	Anza	5
HSK	Husky	5
HRO	Hero	4
NRG	Norge	7
SMT	Smith	3
SHM	Shimara	30
KAR	Karsten	21
NKL	Nikolus	8
PRC	ProCycle	9

state Table

code	sname	code	sname
AK	Alaska	MT	Montana
AL	Alabama	NE	Nebraska
AR	Arkansas	NC	North Carolina
AZ	Arizona	ND	North Dakota
CA	California	NH	New Hampshire
CT	Connecticut	NJ	New Jersey
CO	Colorado	NM	New Mexico
DC	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)

Glossary

access method	A set of server functions that Universal Server uses to access and manipulate an index or a table. See <i>primary access method</i> and <i>secondary access method</i> .
access privilege	An operation that a user has permission to perform in a specific database, table, table fragment, or column.
active set	The set of rows returned by a SELECT statement
aggregate function	A function that returns a single value by performing a mathematical operation on one or more rows, for example, sum, average, minimum, or maximum. Also called <i>math functions</i> or <i>set functions</i> .
API	Acronym for Application Programming Interface.
application process	The process that manages an ESQL/C program at runtime, executes the program logic, and initiates SQL requests. Memory that is allocated for program variables, program data, and the fetch buffer is part of the application process. See also <i>database server process</i> .
application programming interface (API)	A term used to refer to a <i>library</i> of related routines that provide access to a particular product or feature.
application program	An executable file or logically related set of files that perform one or more database management tasks.
argument	A value passed to a function, routine, stored procedure, or command.

array	An ordered set of items of the same data type. Individual members of the array are usually distinguished by an integer argument that gives the position of the element in the array.
attached index	An index that is created without a fragmentation strategy. An attached index can also be an index created on a fragmented table.
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 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 key data in ascending or descending order.
base type	See <i>opaque data type</i> .
blob	Legacy term for <i>simple large object</i> .
BLOB	Binary Large Object. A <i>smart large object</i> that stores any type of binary data, including images. A BLOB can be stored and retrieved in pieces and has database properties such as recovery and transaction rollback.
blobpage	The unit of disk allocation within a blob space. The size of a blob page is determined by the <i>DBA</i> and can vary from blob space to blob space.
blob space	A logical collection of chunks that is used to store TEXT and BYTE data.
BOOLEAN	A <i>built-in data type</i> that supports two one-byte values, one for a true condition and another for a false condition. External representation of these two values is 't' and 'f', respectively.
Boolean function	An <i>external function</i> or <i>SPL function</i> that returns only a Boolean value (TRUE or FALSE).
built-in data type	A fundamental <i>data type</i> defined by the database server, for example, INTEGER, CHAR, or SERIAL8.
BYTE	A <i>data type</i> for a <i>simple large object</i> that stores any type of binary data and can be as large as 2^{31} bytes.
cascading deletes	A feature that causes the deletion of rows from a child table that were associated by foreign key to a row that is deleted from the parent table.

cast	A mechanism that the database server uses to convert data from one data type to another.
casting function	A function that is used to implement a cast. The function performs the necessary operations for conversion between two data types. The function must be registered as a cast with the CREATE CAST statement before it can be used.
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.
chunk	The largest contiguous section disk space available. A group of chunks defines a <i>dbspace</i> or <i>blob space</i> .
client	An application program that requests services from a server program, typically a file server or a database server.
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	Character Large Object. A <i>smart large object</i> that stores large text items, such as PostScript or HTML files. A CLOB can be stored and retrieved in pieces and has database properties such as recovery and transaction rollback.
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 blobs stored in an optical cluster.
clustersize	The amount of space, specified in kilobytes, that is allocated to an optical cluster on an optical volume.
collating sequence	The sequence of values that specifies some logical order in which the character fields in a database are sorted and indexed. Collating sequence is also known as <i>collation order</i> .
collection	An instance of a <i>collection data type</i> ; a group of <i>elements</i> of the same <i>data type</i> stored in a SET, MULTiset, or LIST.

collection cursor	A database <i>cursor</i> that has an INFORMIX-ESQL/C <i>collection variable</i> associated with it and provides access to the individual <i>elements</i> of a column whose data type is a <i>collection data type</i> .
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 <i>collection variable</i> to access <i>elements</i> of the collection as rows of the collection derived table.
collection data type	A <i>complex data type</i> whose instances are groups of <i>elements</i> of the same <i>data type</i> , which can be any <i>opaque data type</i> , <i>distinct data type</i> , <i>built-in data type</i> , <i>collection data type</i> , or <i>row type</i> .
collection variable	An INFORMIX-ESQL/C <i>host variable</i> or <i>SPL variable</i> that holds an entire <i>collection</i> and provides access, through a <i>collection cursor</i> , to the individual <i>elements</i> of the collection.
column	A data element that contains a particular type of information that occurs in every row in the table. Also see <i>row</i> .
column expression	An expression that includes a column name and optionally uses column subscripts to define a column substring.
COMMIT WORK	A statement used to complete a transaction by accepting all changes to the database since the beginning of the transaction. See <i>transaction</i> , <i>ROLLBACK WORK</i> .
complex data type	A <i>data type</i> 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 <i>row type</i> , <i>collection type</i> , and <i>reference type</i> .
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 data type	See <i>row type</i> .
concurrency	The ability of two or more processes to access the same database simultaneously.
connection	An association 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 <i>explicit connection</i> and <i>implicit connection</i> .

constant	A nonvarying data element or value.
constraint	A restriction on what kinds of data can be inserted or updated in tables. See also <i>check constraint</i> , <i>primary-key constraint</i> , <i>referential constraint</i> , <i>not null constraint</i> , and <i>unique constraint</i> .
constructed data type	See <i>complex data type</i> .
constructor	See <i>type constructor</i> .
correlated subquery	A subquery (or inner SELECT) that depends on a value produced by the outer SELECT statement that contains it.
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.
cursor	An identifier associated with a group of rows or a collection data type and that points to one row or collection element at a time.
cursor function	A <i>user-defined function</i> that returns one or more rows of data and therefore requires a <i>cursor</i> to execute. An <i>SPL function</i> is a cursor function when its RETURN statement contains the WITH RESUME keywords. An <i>external function</i> is a cursor function when it is defined as an <i>iterator function</i> .
data distribution	A mapping of the data in a column into a set of the column values, in which 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.
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 replication	The ability to allow database objects to have more than one representation at more than one distinct site.
data restriction	Synonym for constraint. See <i>constraint</i> .
data type	A descriptor assigned to a variable or column that indicates the type of data that the variable or column can hold. See <i>built-in data type</i> , <i>user-defined data type</i> , <i>opaque data type</i> , <i>complex data type</i> , <i>distinct data type</i> .
database	A collection of information (contained in tables) that is useful to a particular organization or used for a specific purpose.
database administrator	The individual responsible for the contents and use of a database.
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 object	In the broad sense, any SQL entity recorded in a system catalog table, such as a table, index, or SPL procedure.
DataBlade API	An API of C routines that a C-language <i>external routine</i> must use instead of SQL statements to access data in INFORMIX-Universal Server.
DataBlade module	A collection of related <i>data types</i> , <i>routines</i> , operators, and access methods that extend an <i>object-relational database</i> to manage new kinds of data or add new features.
DBA	Acronym for <i>Database Administrator</i> .
DBMS	Acronym for <i>database management system</i> . It is all the components necessary to create and maintain a database, including the application development tools and the database server.
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.
declaration statement	A programming language statement that describes or defines objects, for example, defining a program variable. Compare to <i>procedural statement</i> .

default value	A value inserted into a column or variable when an explicit value is not specified. Default values can be assigned to columns using the ALTER TABLE and CREATE TABLE statements and to variables in stored procedures.
detached index	An index that is created with a fragmentation strategy or with the IN <i>dbspace</i> storage option.
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.
distinct data type	A <i>data type</i> 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 <i>opaque data type</i> , <i>built-in data type</i> , <i>named row type</i> , or <i>distinct data type</i>), but has a different name. To compare a <i>distinct data type</i> with its source type requires an <i>explicit cast</i> . A <i>distinct data type</i> inherits all routines that are defined on its source type.
duplicate index	An index that allows duplicate values in the indexed column.
dynamic routine-name specification	The execution of a routine whose name is determined at runtime through an SPL <i>variable</i> , instead of the explicit name of a routine, in the EXECUTE PROCEDURE or EXECUTE FUNCTION statement.
element	A member of a <i>collection</i> . An element can be a single value of any <i>built-in data type</i> , <i>opaque data type</i> , <i>distinct data type</i> , <i>named row type</i> , <i>unnamed row type</i> , or <i>collection data type</i> .
element type	The <i>data type</i> of the <i>elements</i> in a <i>collection</i> .
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. Also see <i>object mode</i> .
end-user function	See <i>end-user routine</i> .
end-user routine	A <i>user-defined routine</i> that end users can specify within an SQL statement to operate on a <i>data type</i> .

ESQL/C Smart Large-Object API	An <i>API</i> of C routines that an INFORMIX-ESQL/C client application can use to access <i>smart large objects</i> as operating-system files. The ESQL/C Smart Large-Object API is part of the INFORMIX-ESQL/C SQL <i>API</i> . Smart large objects can also be accessed by a set of functions in the <i>DataBlade API</i> .
exclusive access	When a user has sole access to a database or table and 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.
explicit cast	A <i>cast</i> that requires a user to specify the CAST AS keyword or cast operator (::) to convert data from one data type to another
explicit connection	A connection made to a database environment that uses the CONNECT statement.
explicit select list	A SELECT statement in which the user explicitly specifies the columns that the query returns.
extended data type	A term used to refer to data types that are not built-in; namely <i>complex data types</i> , <i>opaque data types</i> , and <i>distinct data types</i> .
external function	An <i>external routine</i> that can accept one or more arguments and returns a single value.
external procedure	An <i>external routine</i> that can accept one or more arguments, but does not return a value.
external routine	A <i>routine</i> written in a language external to the database (for example, C), whose body is stored outside the database, but whose name and parameters are registered in the system catalog tables.
external space	Storage space that is managed by a user-defined access method rather than the database server. You can specify the name of an external space, instead of the name of the name of a dbspace, in the IN clause of the CREATE TABLE and CREATE INDEX statements.
field	A component of a <i>named row type</i> or <i>unnamed row type</i> that contains a name and a <i>data type</i> and can be accessed in an SQL statement by using dot notation in the form <i>row type name.field name</i> .

filter	<p>A set of conditions for selecting rows.</p> <p>1) 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 <i>predicate</i>.</p> <p>2) The High-Performance Loader (HPL) uses a filter component to load a subset of rows.</p>
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.
function	A <i>routine</i> that can accept <i>arguments</i> and returns one or more values. See <i>user-defined function</i> .
function cursor	A cursor that is associated with an EXECUTE FUNCTION statement, which executes routines that return values. See <i>cursor function</i> .
function overloading	See <i>routine overloading</i> .
gateway	A data communications device that establishes communications between networks.
global variable	A <i>variable</i> whose value can be accessed from any ESQL/C or SPL routine in a program. Its <i>scope</i> is the entire program. Global variables should not be used in routines written with the <i>DataBlade API</i> .
GLS	See <i>Global Language Support</i> .
GLS API	An <i>API</i> of C routines that a C-language <i>external routine</i> can use to access Informix GLS <i>locales</i> . This API also includes functions that obtain culture-specific collation order, time and date formats, and numeric formats, and functions that provide a uniform way of accessing character data, regardless of whether the locale supports <i>single-byte characters</i> or <i>multibyte characters</i> .
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.
host variable	A C program variable that is referenced in an embedded statement. A host variable is identified by the dollar sign (\$) or colon (:) that precedes it.

identifier	A sequence of letters, digits, and underscores (_) that represents the name of a database, table, column, cursor, function, index, synonym, alias, view, prepared object, constraint, or procedure name.
implicit cast	A <i>cast</i> that the database server automatically performs to convert data from one data type to another. See also <i>explicit cast</i> .
implicit connection	A connection made using a database statement (DATABASE, CREATE DATABASE, START DATABASE, DROP DATABASE). See also <i>explicit connection</i> .
implicit select list	A SELECT statement that uses the asterisk (*) symbol so that a query returns all columns of the table.
index	A structure of pointers that point to rows of data in a table. An index optimizes the performance of database queries by ordering rows to make access faster.
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, if you choose, in a separate dbspace.
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.
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 <i>type inheritance</i> and <i>table inheritance</i> .
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 that is associated with an INSERT statement.
installation	The loading of software from some magnetic medium (tape, cartridge, or floppy disk) or CD onto a computer and preparing it for use.

interquery parallelization	The ability to process multiple queries simultaneously to avoid a performance delay when multiple independent queries access the same table. Also see <i>intraquery parallelization</i> .
intraquery parallelization	The ability to break a single query into subqueries and process the subqueries in parallel in order to reduce data retrieval time and improve performance. Also see <i>interquery parallelization</i> .
isolation	The level of independence among multiple users when they 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.
iterator function	A <i>cursor function</i> that is an external function, written in an external programming language such as C.
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 <i>table hierarchy</i> .
key	The pieces of information that are used to locate a row of data. A key defines the pieces of information you want to search for, as well as the order in which you want to process information in a table.
keyword	A word that has meaning to a program. For example, the word SELECT is a keyword in SQL.
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 <i>simple large objects</i> (TEXT, BYTE) or <i>smart large objects</i> (CLOB, BLOB).
library	A collection of precompiled functions or routines that are designed to perform tasks which are common to a particular kind of application.
link	The process of combining separately compiled program modules into an executable program.
literal	A character or numeric constant.
LIST	A <i>collection data type</i> created with the LIST constructor in which <i>elements</i> are ordered and duplicates are allowed.
local variable	A variable that has meaning only in the module in which it is defined. Its <i>scope</i> is the routine that defines it. See <i>variable</i> and <i>scope of reference</i> .

lock coupling	When an R-tree index is updated, lock coupling is used if a leaf's bounding box has changed. The change must be propagated to the parent node. The propagation is accomplished by moving upwards in the tree until a parent node that does not need to be changed is reached. Lock coupling is used during this upward movement. That is, a lock is held on the child node until a lock is obtained on the parent node.
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.
locking granularity	The size of a locked object. The size may be a database, table, page, or row.
LVARCHAR	A <i>built-in data type</i> that stores varying-length character data of up to 32 kilobytes.
math functions	See <i>aggregate functions</i> .
mirroring	The process of storing the same data on two chunks simultaneously, so that if one chunk fails, the data is still usable on the other chunk.
MULTISET	A <i>collection data type</i> created with the MULTISET constructor in which <i>elements</i> are not ordered and duplicates are allowed.
named row type	A <i>row type</i> created with the CREATE ROW TYPE statement that has a defined name and <i>inheritance</i> properties and can be used to construct a <i>typed table</i> . A named row type is not equivalent to another named row type, even if its field definitions are the same.
node	In indexing of relational databases, an ordered group of key values having a fixed number of elements. A B+ tree, for example, is a set of nodes that contain keys and pointers that are arranged in a hierarchy.
noncursor function	A <i>user-defined function</i> that returns only a single group of values and therefore does not require a <i>cursor</i> when it is executed.

noncursor procedure	A legacy SPL procedure that returns either a single group of values or no values and therefore does not require a <i>cursor</i> when it is executed. Those SPL routines that return only one row are now <i>noncursor functions</i> because procedures return no values.
non-variant function	An external function that always returns the same value when passed the same arguments.
not null constraint	A constraint on a column that specifies the column cannot contain null values.
null value	A value representing unknown or not applicable. (A null is not the same as a value of zero or blank.)
object	See <i>database object</i> .
object mode	The state of a database object as recorded in the sysobjstate system catalog table, for example, enabled, disabled, or filtering.
object-relational database	A database that adds object-oriented features to a <i>relational database</i> , including support for <i>user-defined types</i> , <i>user-defined routines</i> , <i>complex data types</i> , and <i>inheritance</i> .
opaque data type	A fundamental <i>data type</i> you define that 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 <i>user-defined routines</i> and <i>user-defined operators</i> that work on them. Synonym for <i>base type</i> , <i>user-defined base type</i> .
operator	In an SQL statement, a symbol (such as =, >, <, +, -, *, and so forth) that invokes an <i>operator function</i> .
operator binding	The implicit invocation of an <i>operator function</i> when an <i>operator</i> is used in an SQL statement with user-defined data types.
operator class	The set of functions (operators) that INFORMIX-Universal Server associates with a type of secondary access method for query optimization and building the index. Different kinds of operators can be associated with different secondary access methods. See <i>strategy functions</i> and <i>support functions</i> .
operator function	A <i>user-defined function</i> that has a corresponding <i>operator</i> symbol. An operator function processes one to three arguments and returns a value. For example, the plus function corresponds to the “+” operator symbol.
overloading	See <i>function overloading</i> .

pad	Usually a space or blank to fill empty places at the beginning or end of a line, string, or field.
page	The basic unit of disk and memory storage used by the database server and whose size is fixed for a particular operating system and platform.
parallelism	The ability to process a task in parallel by breaking the task into subtasks and processing the subtasks simultaneously in order to improve performance.
parameter	A variable that is given a constant value for a specified application. In a sub-routine, a parameter is the placeholder for the argument values that are passed to the subroutine at runtime.
permission	On some operating systems, the right to access files and directories.
pointer	A number that specifies the “address-in-memory” of the data or variable of interest.
polymorphism	See <i>routine overloading</i> and <i>type substitutability</i> .
predefined opaque data type	An opaque data type for which the database server provides the type definition. See BLOB, CLOB, BOOLEAN.
predicate lock	A lock held on index keys that qualify 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 to allow 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	A set of routines that perform table operations such as inserting, deleting, updating and scanning data in a table.
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 <i>unique key</i> .
primary-key constraint	Specifies that each entry in a column or set of columns contains a non-null unique value.
privilege	The right to use or change the contents of a database, table, table fragment, or column. See <i>access privileges</i> .

procedure overloading	See <i>routine overloading</i> .
procedural statement	A programming language statement that specifies actions; for example, looping and branching if a condition is met. Compare with <i>declaration statement</i> .
procedure	A <i>routine</i> that can accept <i>arguments</i> but does not return a value. See <i>external procedure</i> , <i>SPL procedure</i> , <i>user-defined procedure</i> , <i>stored procedure</i> .
procedure variable	See <i>variable</i> .
program block	A named collection of statements, such as a function, routine, or procedure, that performs a particular task; a unit of program code.
projection	Taking a subset from the columns of a single table. Projection is implemented through the select list of a SELECT statement and returns some of the columns and all the rows in a table. See <i>selection</i> .
R-tree index	A type of index that uses a tree structure based on overlapping bounding rectangles to speed access to spatial and multi-dimensional data types.
referential constraint	Defines the integrity relationship between columns within a table or between tables; also known as a <i>parent-child relationship</i> . Referencing columns also are known as <i>foreign keys</i> .
relational database	A database that logically represents all of its information in tables and supports at least the three relational operations known as selection, projection, and join.
ROLLBACK WORK	The statement 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 <i>transaction</i> , <i>COMMIT WORK</i> .
root supertype	The <i>named row type</i> at the top of a <i>type hierarchy</i> . A <i>root supertype</i> has no <i>supertype</i> above it.
routine	A collection of program statements that perform a particular task. Routines include <i>functions</i> , which return one or more values, and <i>procedures</i> , which do not return values. See <i>user-defined routine</i> .
routine overloading	The property that allows you to define more than one <i>routine</i> with the same name but different parameter lists. Routine overloading allows you to define <i>routines</i> that have the same name but different arguments and functionality.

routine resolution	The process that INFORMIX-Universal Server uses to determine which routine to execute, given the name of the routine, the type of routine (procedure or function), and a list of arguments. In an ANSI-compliant database, the owner is also part of the signature and is used to determine which routine to execute.
routine signature	The information that the database server uses to identify a <i>routine</i> . The <i>signature</i> of a routine includes the type of the routine (function or procedure), the routine name, the number of parameters, the data types of the parameters, and the order of the parameters. In an ANSI-compliant database, the name of the routine is specified as owner.name.
routine -specific name	See <i>specific name</i> .
row	A group of related items of information about a single entity across all columns in a database table. In an object-relational model, each row of a table stands for one <i>instance</i> of the subject of the table, which is one particular example of that entity.
ROW type	See <i>unnamed row type</i> .
row type	A <i>complex data type</i> that contains one or more related data <i>fields</i> , of any <i>data type</i> , that form a template for a record. The data in a <i>row type</i> can be stored in a row or column. See <i>named row type</i> and <i>unnamed row type</i> .
row variable	An INFORMIX-ESQL/C <i>host variable</i> or <i>SPL variable</i> that holds an entire <i>row type</i> and provides access to the individual <i>fields</i> of the row.
sbspace	A logical storage area that contains one or more chunks that only store BLOB and CLOB data.
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	The part of a routine or application program in which an identifier applies and can be accessed, such as local, global, and modular.
scroll cursor	A cursor that can fetch the next row or any prior row, so it can read rows multiple times.

secondary access method	A set of server functions to access and manipulate an index. Universal Server provides the B-tree and R-tree secondary access methods; DataBlade Modules provide other secondary access methods.
select cursor	A cursor that is associated with a SELECT statement.
selection	Taking the horizontal subset of rows of a single table that satisfies a particular condition. This kind of SELECT statement returns some of the rows and all of the columns in a table. Selection is implemented through the WHERE clause of a SELECT statement. See <i>projection</i> .
selectivity	The proportion of rows within the table that a query filter can pass.
sequential cursor	A cursor that can fetch only the next row in sequence so 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.
server name	The unique name of a database server, assigned by a database administrator and used by an application to select a database server.
server number	A unique number between 0 and 255, inclusive, assigned by a database administrator to a database server when the server is initialized.
session	The structure that is created for an application using the database server.
SET	A <i>collection data type</i> created with the SET type constructor, in which <i>elements</i> are not ordered and duplicate values can be inserted.
set functions	See <i>aggregate functions</i> .
shared library	A library that contains routines that are used by applications, are loaded into memory by the operating system as they are needed, and are shared with other applications.
shared lock	A lock that more than one thread can acquire on the same object, allowing for greater concurrency with multiple users.
shared memory	A portion of main memory that more than one process can use at a time so that processes can communicate and share common data, thus reducing disk I/O and improving performance.
signature	See <i>routine signature</i> .

simple blob	Synonym for <i>simple large object</i> .
simple large object	A <i>large object</i> that is stored in a <i>blob space</i> , is not recoverable, and does not obey transaction isolation modes. <i>Simple large objects</i> include <i>TEXT</i> and <i>BYTE</i> data types.
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).
singleton SELECT	A statement that returns a single row.
smart large object	A <i>large object</i> that is stored in an <i>sbspace</i> , 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 <i>CLOBs</i> and <i>BLOBs</i> .
SMI	Acronym for <i>System Monitoring Interface</i> .
source file	A file containing instructions (in ASCII text) that is used as the source for generating compiled programs.
specific name	A unique identifier that you define with the SPECIFIC keyword in a CREATE PROCEDURE or CREATE FUNCTION statement to serve as an alternate name for a routine. In an ANSI-compliant database, the specific name must be unique within the schema. In a database that is not ANSI-compliant, the specific name must be unique within the database. A specific name is distinct from the name of the actual routine. A routine that has been overloaded can be uniquely identified by its specific name. A specific name can be used with the DROP , GRANT , REVOKE , and UPDATE STATISTICS .
SPL	Acronym for <i>Stored Procedure Language</i> , an Informix extension to SQL that provides flow control features such as sequencing, branching, and looping, comparable to those provided in the SQL/PSM standard.
SPL function	An SPL routine that can accept arguments and returns one or more values.
SPL procedure	An SPL routine that can accept arguments and may update the database, but does not return a value.
SPL routine	A <i>routine</i> written in <i>Stored Procedure Language (SPL)</i> and <i>SQL</i> that is parsed, optimized, and stored in the system catalog tables in executable format. SPL routines include <i>SPL procedures</i> and <i>SPL functions</i> .
SPL variable	A variable defined and used in an SPL routine.

SQL API	An <i>API</i> that provides a library of functions to an Informix client application, such as an INFORMIX-ESQL/C application, so it can access a database server through <i>embedded SQL</i> and function calls. See <i>Client API</i> .
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 section of a program that usually begins and terminates with special symbols such as <i>begin</i> and <i>end</i> . A statement block is the smallest unit of 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. A statement identifier is used for dynamic SQL statement management by Informix SQL APIs.
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.
stored function	See <i>user-defined function</i> .
stored procedure	A term used in earlier versions of Informix products for SPL <i>procedures</i> and SPL <i>functions</i> . See <i>SPL routine</i> , <i>SPL procedure</i> , and <i>SPL function</i> .
Stored Procedure Language	See <i>SPL</i> , <i>SPL routine</i> .
strategy functions	The functions that the optimizer uses to decide what filters in a query can use a secondary access method (index).
subquery	A SELECT statement within a WHERE clause.
subtable	A <i>typed table</i> that inherits properties (column definitions, constraints, triggers) from a <i>supertable</i> above it in the <i>table hierarchy</i> and can add additional properties.
subtype	A <i>named row type</i> that inherits all representation (data <i>fields</i>) and behavior (<i>routines</i>) from a <i>supertype</i> above it in the <i>type hierarchy</i> and can add additional fields and routines. The number of <i>fields</i> in a <i>subtype</i> is always greater than or equal to the number of <i>fields</i> in its <i>supertype</i> .
supertable	A <i>typed table</i> whose properties (constraints, storage options, triggers) are inherited by a <i>subtable</i> beneath it in the <i>table hierarchy</i> . The scope of a query on a <i>supertable</i> is the <i>supertable</i> and its <i>subtables</i> .

supertype	A <i>named row type</i> whose representation (data <i>fields</i>) and behavior (<i>routines</i>) is inherited by a <i>subtype</i> below it in the <i>type hierarchy</i> .
support functions	<p>The functions that INFORMIX-Universal Server automatically invokes to process a data type.</p> <ol style="list-style-type: none"> 1) The database server uses a support function to perform operations (such as converting to and from the internal, external, and binary representations of the type) on opaque data types. 2) An index access method uses a support function in an operator class to perform operations (such as building or searching) on an index.
sysmaster database	A database that each database server has and that holds the catalog tables and system monitoring interface (SMI) tables.
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.
system-defined cast	A cast that is built in to the database server. A system-defined casts performs automatic conversions between different built-in data types.
System Monitoring Interface	A collection of tables in the sysmaster database that maintains dynamically updated information about the operation of the database server.
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. A table is sometimes referred to as a <i>base table</i> to distinguish it from the views, indexes, and other objects defined on the underlying table or associated with it.
table hierarchy	A relationship you can define among <i>typed tables</i> in which <i>subtables</i> inherit the behavior (constraints, triggers, storage options) from <i>supertables</i> . <i>Subtables</i> can add additional constraint definitions, storage options, and triggers.
table inheritance	The property that allows a <i>typed table</i> to inherit the behavior (constraints, storage options, triggers) from a <i>typed table</i> above it in the <i>table hierarchy</i> .
TEXT	A <i>data type</i> for a <i>simple large object</i> that stores text and can be as large as 2^{31} bytes.
transaction	A collection of one or more SQL statements that are treated as a single unit of work and are <i>committed</i> or <i>rolled back</i> together.

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 <i>logical log</i> .
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 and 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.
type constructor	An SQL keyword that indicates to the database server the type of complex data to create (for example, ROW, SET, MULTISSET, LIST).
type hierarchy	A relationship that you define among <i>named row types</i> in which <i>subtypes</i> inherit representation (data <i>fields</i>) and behavior (<i>routines</i>) from <i>supertypes</i> and can add additional <i>fields</i> and <i>routines</i> .
type inheritance	The property that allows a <i>named row type</i> or <i>typed table</i> to inherit representation (data <i>fields</i> , columns) and behavior (<i>routines</i> , operators, rules) from a named row type above it in the <i>type hierarchy</i> .
type substitutability	The ability to use an instance of a <i>subtype</i> when an instance of its <i>supertype</i> is expected.
typed collection variable	An ESQL/C <i>collection variable</i> or SPL <i>variable</i> that has a defined <i>collection data type</i> associated with it and can only hold a <i>collection</i> of its defined type. See <i>untyped collection variable</i> .
typed table	A table that is constructed from a <i>named row type</i> and whose rows contain instances of that <i>row type</i> . A typed table can be used as part of a <i>table hierarchy</i> . The columns of a <i>typed table</i> correspond to the <i>fields</i> of the <i>named row type</i> .
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 <i>primary key</i> .

unnamed row type	A <i>row type</i> 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 <i>fields</i> and if corresponding fields have the same <i>data type</i> , even if the fields have different names.
untyped collection variable	A generic ESQL/C <i>collection variable</i> or <i>SPL variable</i> that can hold a <i>collection</i> of any <i>collection data type</i> and takes on the data type of the last collection assigned to it. See <i>typed collection variable</i> .
user-defined base type	See <i>opaque data type</i> .
user-defined data type	A <i>data type</i> that you define for use in a Universal Server database. You can define <i>opaque data types</i> and <i>distinct data types</i> .
user-defined function	A <i>user-defined routine</i> that returns a value. A user-defined function can be a <i>support function</i> or an <i>end-user function</i> .
user-defined routine	A routine that is not included with the database server or <i>DataBlade module</i> , whose name and parameters are registered in the system catalog tables, and that can be invoked by an SQL statement or another routine. A user-defined routine can be written in an external language (external routine) or in SPL (SPL routine).
user-defined procedure	A <i>user-defined routine</i> that does not return a value. A user-defined procedure cannot be a <i>support routine</i> because all support routines return values.
user ID	See <i>Informix user ID</i> .
user ID password	See <i>Informix user password</i> .
user name	See <i>Informix user ID</i> .
user password	See <i>Informix user password</i> .
variable	The identifier for a location in memory that stores the value of a program object whose value can change during the execution of the program.
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 <code>START VIOLATIONS TABLE</code> statement to create a violations table and associate it with a base table.
virtual column	A derived column of information that is not stored in the database but that is created in an SQL statement by manipulating or combining database columns.
virtual processor	One of the multithreaded processes that make up the database server and are similar to the hardware processors in the computer.
virtual table	A table whose data that you create to access external data in a file or database that is not managed by Universal Server. The Virtual Table Interface allows Universal Server users to access the external data in a virtual table using SQL DML statements and join the external data with Universal Server table data.
white space	A series of one or more space characters. The locale file defines what characters are considered to be space characters.
wildcard	A special symbol, such as an asterisk or question mark, represents any sequence of zero or more characters or any single character in an SQL statement, operating system command, or program language statement.

Index

A

Access method
 definition of Glossary-1
 sysams information about 1-12
 Access privilege Glossary-1
 ALTER TABLE statement
 casting effects 2-26
 MODIFY NEXT SIZE clause 1-10
 ANSI compliance
 -ansi flag 3-15
 checking for
 (DBANSIWARN) 3-16
 DBANSIWARN environment
 variable 3-15
 icon Intro-10
 level Intro-16
 Application
 process Glossary-1
 program Glossary-1
 Application programming interface
 (API) Glossary-1
 Archiving
 personal default qualifier file
 (ARC_DEFAULT) 3-14
 remote shell for
 (DBREMOTECMD) 3-33
 tctermcap file for
 (ARC_KEYPAD) 3-14
 ARC_DEFAULT environment
 variable 3-14
 ARC_KEYPAD environment
 variable 3-14
 Argument Glossary-1

Array

 definition of Glossary-2
 Attached index Glossary-2
 Auxiliary statements Glossary-2

B

Binary Large Object (BLOB)
 buffer size variable
 (DBBLOBBUF) 3-17
 BLOB data type 2-33
 Glossary-2
 attributes in syscolattns 1-20
 casting not available 2-33
 coltype code for 1-26
 inserting data 2-34
 blobpage Glossary-2
 blobspace 2-19, Glossary-2
 blobspaces
 size and location 1-18
 BOOLEAN data type 2-34
 Glossary-2
 coltype code for 1-26
 description of 2-34
 Boolean expression
 TEXT data type limitation 2-65
 Boolean function Glossary-2
 Bourne shell
 setting environment variables 3-6
 .profile file 3-6
 B-tree index Glossary-2
 Buffer
 size for fetch
 (FET_BUF_SIZE) 3-41
 Built-in data type 2-9 to 2-19
 Glossary-2

- casting, system-defined 2-25, 2-32
- INT 2-47
- BYTE data type 2-35
 - Glossary-2
 - coltype code for 1-24
 - inserting data 2-35
 - length (syscolumns) 1-29
 - restrictions
 - in Boolean expression 2-35
 - with GROUP BY 2-35
 - with LIKE or MATCHES 2-35
 - with ORDER BY 2-35
 - selecting a BYTE column 2-36
 - sysblobs information 1-18

C

- C compiler (INFORMIXC) 3-42
- C shell
 - setting environment variables 3-6
 - .cshrc file 3-6
 - .login file 3-6
- call_type table in stores7 database, columns in A-6
- Cast 2-25 to 2-32
 - definition of Glossary-3
 - distinct data type 2-31
 - explicit 2-29, 2-30
 - implicit 2-29, 2-30
 - precedence of 2-30
 - syscasts information 1-19
 - system-defined 2-25 to 2-28, 2-30
- Casting function Glossary-3
- CHAR data type 2-36
 - casts for 2-27
 - collation order dependency 2-36
 - coltype code for 1-24
 - numeric values with 2-37
- CHARACTER data type 2-37
- Character data type
 - length (column length) information 1-27
 - list of 2-9
 - varying length 2-38
- Character string
 - as DATETIME values 2-14, 2-43
 - as INTERVAL values 2-51

- CHARACTER VARYING data
 - type 2-38
 - description of 2-38
 - length 1-27
- Check constraint
 - definition of Glossary-3
 - syschecks information 1-20
 - syscoldepend
 - (dependencies) 1-23
- Checkpoint Glossary-3
- chkenv utility 3-9
 - error message for 3-9
- Chunk Glossary-3
- Client Glossary-3
- Client/server
 - architecture Glossary-3
 - client stacksize
 - (INFORMIXSTACKSIZE) 3-4
 - 9
 - connection statements Glossary-3
 - default database
 - (INFORMIXSERVER) 3-47
 - shared memory communication segments
 - (INFORMIXSHMBASE) 3-49
- CLOB data type 2-38
 - attributes in syscolatrics 1-20
 - casting not available 2-38
 - code-set conversion of 2-39
 - collation 2-39
 - coltype code for 1-26
 - definition of Glossary-3
 - inserting data 2-39
- Cluster index Glossary-3
- Cluster key Glossary-3
- Cluster size Glossary-3
- Collation
 - with CLOB data type 2-39
- Collation order Glossary-3
- NCHAR data type 1-12
- Collection Glossary-3
- Collection cursor Glossary-4
- Collection data type 2-22
 - casting
 - summary matrix 2-32
 - coltype code for 1-24
 - definition of Glossary-4
 - LIST 2-51
 - MULTISET 2-54

- SET 2-62
- sysattrtypes information about
 - elements 1-17
 - sysxtddesc contents for 1-62
 - sysxtddtypes information 1-64
- Collection derived table
 - definition of Glossary-4
- Collection variable Glossary-4
- Colon (:), delimiter in
 - INTERVAL 2-50
- Color, setting INFORMIXTERM for 3-50
- Column
 - constraints
 - sysconstraints 1-29
 - syscoldepend 1-23
 - sysreferences 1-49
 - default value
 - sysdefaults 1-30
 - in stores7 database A-2 to A-7
 - privileges
 - syscolauth 1-22
 - syscolumns
 - description 1-23 to 1-31
- Column expression Glossary-4
- Column value
 - maximum/minimum 1-29
- columns Information Schema
 - view 1-68
- Command-line conventions
 - elements of Intro-11
 - example diagram Intro-12
 - how to read Intro-12
- Comment icons Intro-9
- COMMIT WORK statement
 - commit defined Glossary-4
- Commutator function Glossary-4
- Compiler
 - environment variable for C
 - (INFORMIXC) 3-42
- Complex data type 2-20 to 2-24
 - collection types 2-22
 - definition of Glossary-4
 - row types 2-24
 - sysattrtypes information about
 - members 1-17
- Compliance
 - icons Intro-10
 - with industry standards Intro-16

Composite index Glossary-4
 Concurrency
 description of Glossary-4
 Configuration file
 database server
 (ONCONFIG) 3-54
 ON-Archive utility
 (ARC_DEFAULT) 3-14
 tctermcap (TERMCAP) 3-14
 CONNECT DEFAULT statement
 INFORMIXSERVER
 environment variable
 effect 3-47
 Connecting to data
 INFORMIXCONRETRY
 environment variable 3-43
 INFORMIXCONTIME
 environment variable 3-44
 Connection Glossary-4
 INFORMIXCONRETRY
 environment variable 3-43
 INFORMIXCONTIME
 environment variable 3-44
 Constant Glossary-5
 Constraint
 definition of Glossary-5
 not null 1-23
 object mode (sysobjstate) 1-40
 syscoldepend
 (dependencies) 1-23
 sysconstraints information
 about 1-29
 Correlation name
 definition of Glossary-5
 CREATE SCHEMA statement
 example 1-4
 CREATE VIEW statement
 sysviews information 1-8
 Cursor
 definition of Glossary-5
 Cursor function Glossary-5
 customer table in stores7
 database A-2
 cust_calls table in stores7 database,
 columns in A-6

D

Data distribution Glossary-5
 Data distributions
 disk space for
 (DBUPSPACE) 3-39
 sysdistrib information 1-32
 Data integrity
 definition of Glossary-5
 statements Glossary-5
 Data manipulation
 statements Glossary-5
 Data replication Glossary-6
 Data type
 approximate 1-68
 casting 2-25 to 2-32
 definition of Glossary-6
 exact numeric 1-68
 extended 2-20, 2-71
 floating-point 2-47
 for sequential integer 2-59, 2-61
 for unique numeric code 2-59,
 2-61
 INT 2-47
 simple large object 2-19
 smart large object 2-17
 summary table 2-7
 Database
 data types 2-33
 defined Glossary-6
 map of
 stores7 A-8
 stores7 description of A-1
 tables in (systables) 1-54
 Database administrator
 (DBA) Glossary-6
 Database application Glossary-6
 Database environment Glossary-6
 Database management system
 (DBMS) Glossary-6
 Database object Glossary-6
 state of (sysobjstate) 1-40
 Database server
 attributes in Information Schema
 view 1-70
 default for connection
 (INFORMIXSERVER) 3-47
 Database, stores 7 A-1
 Datablade

 message information
 (syserrors) 1-33
 DataBlade API Glossary-6
 DataBlade module Glossary-6
 DATE data type 2-39
 casting to integer 2-28
 converting to DATETIME 2-28
 DATETIME, INTERVAL
 with 2-10, 2-13
 format for (DBDATE) 3-21
 international date formats 2-40
 source data 2-14
 syscolumns coltype code for 1-24
 two-digit year values and
 DBCENTURY variable 2-40
 year digits for
 (DBCENTURY) 3-18
 DATETIME data type 2-40
 character string values 2-43
 converting to DATE 2-28
 converting to integer 2-28
 DATE INTERVAL
 with 2-10 to 2-15
 EXTEND function with 2-13
 extending fields in 2-11
 field qualifiers 2-41
 format for (DBTIME) 3-36
 international formats 2-44
 length (syscolumns) 1-28
 source data 2-14
 syscolumns coltype code for 1-24
 two-digit year values and
 DBDATE variable 2-43
 year digits for
 (DBCENTURY) 3-18
 dbaccessdemo7 script Intro-6
 DBANSIWARN environment
 variable 3-15
 DBBLOBBUF environment
 variable 3-17
 DBCENTURY environment
 variable 3-18
 DBDATE with 3-23
 DBDATE environment
 variable 3-21
 DBDELIMITER environment
 variable 3-24
 DBEDIT environment variable 3-24

- DBFLTMASK environment
 - variable 3-25
- DBLANG environment
 - variable 3-26
- DBMONEY environment
 - variable 3-27
- DBONPLOAD environment
 - variable 3-29
- DBPATH environment
 - variable 3-29
- DBPRINT environment
 - variable 3-32
- DBREMOTECMD environment
 - variable 3-33
- dbspace
 - definition of Glossary-6
- dbspaces
 - size and location 1-18
- DBSPACETEMP environment
 - variable 3-34
- DBTEMP environment
 - variable 3-35
- DBTIME environment
 - variable 3-36
- DBUPSPACE environment
 - variable 3-39
- DECIMAL data type
 - casts for 2-27
 - description of 2-45
 - disk storage 2-45
 - fixed-point 2-45
 - floating-point 2-44
 - length (syscolumns) 1-27
 - syscolumns coltype code for 1-24
- Decimal digits, display of 3-25
- Decimal point (.)
 - as delimiter in INTERVAL 2-50
- Default locale Intro-5
- Default value Glossary-7
 - sysdefaults information 1-30
- DELIMIDENT environment
 - variable 3-40
- Delimiter
 - for DATETIME values 2-42
 - for INTERVAL values 2-50
- Demonstration database Intro-6
 - map of A-8
 - structure of tables A-2
 - tables in A-2 to A-7

- Detached index Glossary-7
- Diagnostics table
 - sysviolations information 1-61
- Disabled object mode
 - defined Glossary-7
- Disk space
 - distribution data size 3-39
- Distinct data type
 - casts 2-31
 - definition of 2-46, Glossary-7
 - sysxtddesc contents for 1-62
- Documentation conventions
 - command-line Intro-10
 - icon Intro-9
 - sample-code Intro-13
 - typographical Intro-8
- Documentation notes Intro-15
- Documentation, types of
 - documentation notes Intro-15
 - error message files Intro-15
 - machine notes Intro-15
 - on-line manuals Intro-14
 - printed manuals Intro-14
 - release notes Intro-15
- DOUBLE PRECISION data
 - type 2-47
- Duplicate index Glossary-7
- Dynamic routine-name
 - specification Glossary-7

E

- Editor, specifying with
 - DBEDIT 3-24
- Element Glossary-7
- Element type Glossary-7
- Enabled object mode
 - defined Glossary-7
- End-user routine Glossary-7
- ENVIGNORE environment
 - variable 3-41
 - chkenv utility with 3-9
- Environment configuration file 3-4
 - chkenv utility for 3-9
 - common (SINFORMIXDIR/etc/informix.rc) 3-9
 - debugging with chkenv 3-9
 - ignoring errors in (ENVIGNORE) 3-9
 - locations 3-9
 - private (informix/informix.rc) 3-9
- Environment variable 3-3 to 3-70
 - ARC_DEFAULT 3-14
 - ARC_KEYPAD 3-14
 - checking with chkenv utility 3-9
 - configuration file with 3-4
 - DBANSIWARN 3-15
 - DBBLOBBUF 3-17
 - DBCENTURY 3-18
 - DBDATE 3-21
 - DBDELIMITER 3-24
 - DBEDIT 3-24
 - DBFLTMASK 3-25
 - DBLANG 3-26
 - DBMONEY 3-27
 - DBONPLOAD 3-29
 - DBPATH 3-29
 - DBPRINT 3-32
 - DBREMOTECMD 3-33
 - DBSPACETEMP 3-34
 - DBTIME 3-36
 - DBUPSPACE 3-39
 - DELIMIDENT 3-40
 - ENVIGNORE 3-41
 - FET_BUF_SIZE 3-41
 - INFORMIXC 3-42
 - INFORMIXCONRETRY 3-43
 - INFORMIXCONTIME 3-44
 - INFORMIXDIR 3-45
 - INFORMIXOPCACHE 3-46
 - INFORMIXSERVER 3-47
 - INFORMIXSHMBASE 3-48
 - INFORMIXSQLHOSTS 3-49
 - INFORMIXSTACKSIZE 3-49
 - INFORMIXTERM 3-50
 - INF_ROLE_SEP 3-51
 - listed, by topic 3-63
 - modifying 3-8
 - NODEFDAC 3-54
 - ONCONFIG 3-54
 - OPTCOMPIND 3-55
 - overriding a setting (ENVIGNORE) 3-41
 - PATH 3-56
 - PDQPRIORITY 3-56

PLCONFIG 3-58
 PSORT_DBTEMP 3-58
 PSORT_NPROCS 3-59
 rules of precedence 3-10
 setting 3-4
 setting shell 3-6
 TERM 3-61
 TERMCAP 3-61
 TERMINFO 3-62
 THREADLIB 3-63
 types of 3-3
 view current setting 3-7
 en_us.8859-1 locale Intro-5
 Error message files Intro-15
 Error messages
 creating new messages 1-34
 ESQL/C Smart-Large-Object
 API Glossary-8
 Exclusive access Glossary-8
 Exclusive lock Glossary-8
 Executable file location
 PATH environment variable 3-56
 Explicit cast Glossary-8
 Explicit connection Glossary-8
 Explicit select list Glossary-8
 EXTEND function
 example 2-13
 Extended data type 2-20, 2-71,
 Glossary-8
 Extension, to SQL
 checking for with DBANSIWARN
 environment variable 3-15
 Extent
 changing size of system table 1-10
 External function
 definition of Glossary-8
 name in sysprocedures 1-46
 External procedure
 definition of Glossary-8
 External routine
 compiling multithreaded
 applications 3-63
 definition of Glossary-8
 languages supported in
 sysroutinelangs 1-51
 sysprocauth (Execute privileges)
 information 1-44
 sysprocedures information 1-46
 External space Glossary-8

F

Feature icons Intro-10
 Features, product Intro-6
 FET_BUF_SIZE environment
 variable 3-41
 Field Glossary-8
 Field delimiter
 (DBDELIMITER) 3-24
 File
 temporary
 dbspace (DBSPACETEMP) 3-34
 directory for sorting
 (PSORT_DBTEMP) 3-58
 directory for (DBTEMP) 3-36
 temporary, sorting 3-58
 termcap, terminfo 3-50, 3-61
 FILETOBLOB function 2-33
 FILETOCLOB function 2-38
 Filter Glossary-9
 Fixed point 2-45
 FLOAT data type 2-47
 casts for 2-27
 syscolumns coltype code for 1-24
 Floating point 2-44
 single-precision (FLOAT) 2-47
 Foreign key Glossary-9
 Format
 DATETIME, specifying
 (DBTIME) 3-36
 DATE, specifying
 (DBDATE) 3-22
 MONEY, specifying with
 DBMONEY 3-27
 Fragment
 privileges
 sysfragauth information 1-34
 Fragmentation
 PDQ priority, setting
 (PDQPRIORITY) 3-56
 sysfragments information 1-36
 Function
 definition of Glossary-9
 used with BLOB columns 2-33

G

Gateway Glossary-9
 Global Language Support
 (GLS) Intro-5
 locale
 GL_COLLATE and GL_TYPE in
 systables 1-56
 systables information 1-56
 Global variable Glossary-9
 GLS API Glossary-9

H

Hash rule Glossary-9
 High-Performance Loader,
 environment variable for 3-29,
 3-58
 Host variable Glossary-9

I

Icons
 comment Intro-9
 compliance Intro-10
 feature Intro-10
 Identifier Glossary-10
 IFX_DEFERRED_PREPARE
 environment variable 3-53
 Implicit cast 2-29, Glossary-10
 Implicit connection Glossary-10
 Implicit select list Glossary-10
 Index Glossary-10
 attached Glossary-2
 object mode (sysobjstate) 1-40
 on routines 1-48
 sysindices information 1-37
 threads for 3-60
 Index fragment Glossary-10
 IndexKey structure
 fields in 1-38, 1-39
 Industry standards, compliance
 with Intro-16
 Information Schema views
 accessing 1-67
 columns view 1-68
 description of 1-65 to 1-71
 generating 1-66

- server_info view 1-70
- sql_languages view 1-70
- tables view 1-67
- Informix extension checking,
 - specifying with DBANSIWARN 3-15
- Informix user ID Glossary-10
- Informix user
 - password Glossary-10
- INFORMIXC environment
 - variable 3-42
- INFORMIXCONCSMCFG
 - environment variable 3-43
- INFORMIXCONRETRY
 - environment variable 3-43
- INFORMIXCONTIME
 - environment variable 3-44
- INFORMIXDIR environment
 - variable 3-45
- INFORMIXDIR/bin
 - directory Intro-6
- INFORMIXKEYTAB environment
 - variable 3-46
- INFORMIXOPCACHE
 - environment variable 3-46
- INFORMIXSERVER environment
 - variable 3-47
- INFORMIXSHMBASE
 - environment variable 3-48
- INFORMIXSTACKSIZE
 - environment variable 3-49
- INFORMIXTERM environment
 - variable 3-50
- Informix, environment
 - configuration file 3-4
- informix.rc file 3-4
- INF_ROLE_SEP environment
 - variable 3-51
- Inheritance
 - definition of Glossary-10
- Input parameter Glossary-10
- Input support function 2-53
- Installation Glossary-10
- Installation directory, specifying
 - with INFORMIXDIR 3-46
- Installation files, INFORMIXDIR
 - environment variable 3-45
- INT data type 2-47
- INT8 data type 2-47

- casts for 2-27, 2-28
- syscolumns coltype code for 1-24
- INTEGER data type 2-48
 - casts for 2-27, 2-28
 - length (syscolumns) 1-27
 - syscolumns coltype code for 1-24
- Intensity attributes, setting
 - INFORMIXTERM for 3-50
- Interquery
 - parallelization Glossary-11
- INTERVAL data type 2-48
 - DATE, DATETIME with 2-10, 2-15 to 2-16
 - expressions with 2-10
 - EXTEND function with 2-13
 - field delimiters 2-50
 - length (syscolumns) 1-28
 - syscolumns coltype code for 1-24
- Intraquery
 - parallelization Glossary-11
- ISO 8859-1 code set Intro-5
- Isolation Glossary-11
- items table in stores7 database,
 - columns in A-4
- iterator function Glossary-11

J

- Jagged rows Glossary-11

K

- Key Glossary-11
- Keyword Glossary-11
- Korn shell
 - setting environment variables 3-6
 - .profile file 3-6

L

- Language
 - supported
 - sysroutinelangs
 - information 1-51
- Language environment
 - DBLANG 3-26
 - setting with DBLANG 3-26

- Large object 2-16
 - character 2-38
 - definition of Glossary-11
- Library Glossary-11
 - shared Glossary-17
- Link Glossary-11
- LIST Glossary-11
- LIST data type 2-51
 - syscolumns coltype code for 1-24
- Literal Glossary-11
- Local variable Glossary-11
- Locale Intro-5
- Lock Glossary-12
- Lock mode Glossary-12
- Locking Glossary-12
- Locking granularity Glossary-12
- LOCOPY function 2-33, 2-38
- LOTOFILE function 2-33, 2-38
- LVARCHAR Glossary-12

M

- Machine notes Intro-15
- Major features Intro-6
- Message files, specifying
 - subdirectory with DBLANG 3-26
- Messages
 - error
 - creating 1-34
 - (syserrors) 1-33
 - informational
 - creating 1-34
 - (syserrors) 1-33
 - warning
 - creating 1-34
 - (syserrors) 1-33
- Mirroring Glossary-12
- MONEY data type 2-53
 - casts for 2-27
 - display format specified with DBMONEY 3-27
 - international money formats 2-54
 - length(syscolumns) 1-27
 - syscolumns coltype code for 1-24
- MULTISET Glossary-12
- MULTISET data type 2-54
 - syscolumns coltype code for 1-24

Multithreaded external
applications
compiling 3-63

N

Named row data type Glossary-12
Named row type 2-56 to 2-57
casting permitted 2-32
NCHAR data type
collation order 2-57
multibyte characters 2-57
syscolumns coltype code for 1-24
Negator function Glossary-12
Network environment variable
DBPATH 3-29
Node Glossary-12
NODEFDAC environment variable
description of 3-54
Noncursor
function Glossary-12
noncursor procedure Glossary-13
Nonprintable characters
with CHAR data type 2-37
Non-variant function Glossary-13
Not null constraint 1-23,
Glossary-13
Null value Glossary-13
allowed or not allowed 1-25
syscolumns coltype code for 1-24
testing in BYTE expression 2-35
TEXT data type and 2-65
Numeric data type
casting between 2-26
casting to characters 2-27
list of 2-9
NVARCHAR data type 2-58
collation order 2-58
length (syscolumns) 1-28
multibyte characters 2-58
syscolumns coltype code for 1-24

O

Object mode Glossary-13
sysobjstate information 1-40
Object-relational
database Glossary-13

ON-Archive configuration file
(ARC_DEFAULT) 3-14
ONCONFIG environment
variable 3-54
On-line manuals Intro-14
Opaque data type 2-58
description of Glossary-13
smart large objects with 2-17
storage of 2-53
syscolumns coltype code for 1-24
sysxtddesc contents for 1-62
Opaque type
cast matrix 2-32
Operator Glossary-13
binding Glossary-13
function Glossary-13
Operator class
definition Glossary-13
sysopclasses information 1-42
OPTCOMPIND environment
variable, values 3-55
orders table in stores7 database,
columns in A-3
Output support function 2-53

P

Pad Glossary-14
Page Glossary-14
Parallel distributed queries
priority level
(PDQPRIORITY) 3-56
Parallel sorting threads
(PSORT_NPROCS) 3-58
Parallelism Glossary-14
Parameter Glossary-14
PATH environment variable 3-56
Pathname
C compiler (INFORMIXC) 3-42
database server (DBPATH) 3-29
executable programs
(PATH) 3-56
installation
(INFORMIXDIR) 3-45
message files (DBLANG) 3-26
parallel sorting
(PSORT_DBTEMP) 3-58

remote shell
(DBREMOTECMD) 3-33
PDQ
optimizing joins
(OPTCOMPIND) 3-55
prioritizing
(PDQPRIORITY) 3-56
Permission Glossary-14
PLCONFIG environment
variable 3-58
Pointer Glossary-14
Polymorphism Glossary-14
Precedence, rules for environment
variables 3-10
predefined data type Glossary-14
Predicate lock Glossary-14
PREPARE statement
deferred execution 3-53
Prepared statement Glossary-14
version number 1-56
Preprocessor Glossary-14
Primary access
method Glossary-14
Primary key
definition of Glossary-14
Primary key constraint
definition of Glossary-14
Print program 3-32
Printed manuals Intro-14
Printing with DBPRINT 3-32
Privilege Glossary-14
column-specific
syscolauth information 1-22
database-level
sysusers information 1-60
public
NODEFDAC environment
variable 3-54
table fragment
sysfragauth information
about 1-34
table-level
syscolauth sample entry 1-8
sysabauth information 1-8,
1-53
Usage
sysxtdtypeauth
information 1-63
Procedural statement Glossary-15

Procedure
 description of Glossary-15
 Program block Glossary-15
 Projection Glossary-15
 Protected routine 1-48
 PSORT_DBTEMP environment
 variable 3-58
 PSORT_NPROCS environment
 variable 3-59

Q

Query
 optimization information
 (sysprocplan) 1-48
 Quoted string
 DATE, DATETIME 2-14

R

Referential constraint
 definition of Glossary-15
 sysreferences information 1-49
 Relational database Glossary-15
 Release notes Intro-15
 Role
 separation (INF_ROLE_SEP) 3-51
 sysroleauth information 1-50
 ROLLBACK WORK statement
 roll back defined Glossary-15
 Root supertype Glossary-15
 Routine Glossary-15
 execution plan (sysprocplan) 1-48
 optimization information
 (sysprocplan) 1-48
 overloading Glossary-15
 protected 1-48
 resolution Glossary-16
 signature Glossary-16
 sysprocbody contains source 1-45
 sysprocedures information 1-46
 Row type 2-24
 casting permitted 2-32
 definition of Glossary-16
 field information
 (sysattrtypes) 1-17
 sysattrtypes information 1-17
 syscolumns coltype code for 1-24

sysxtddesc contents for 1-62
 sysxtddtypes information 1-64
 Row variable
 definition of Glossary-16
 R-tree index Glossary-15

S

Sample-code conventions Intro-13
 sbspace 2-17
 definition of Glossary-16
 sbspaces
 size and location 1-18
 Schema Glossary-16
 Scope Glossary-16
 Scroll cursor
 definition of Glossary-16
 Secondary access
 method Glossary-17
 definition of Glossary-17
 Select cursor
 definition of Glossary-17
 Selection Glossary-17
 Selectivity Glossary-17
 Sequential cursor
 definition of Glossary-17
 Sequential integers
 with SERIAL data type 2-59
 with SERIAL8 data type 2-61
 SERIAL data type 2-59
 inserting values 2-60
 length (syscolumns) 1-27
 resetting values 2-60
 syscolumns coltype code for 1-24
 SERIAL8 data type 2-61
 inserting values 2-62
 length (syscolumns) 1-27
 resetting values 2-62
 syscolumns coltype code for 1-24
 Serializable Glossary-17
 Server
 name Glossary-17
 number Glossary-17
 server_info Information Schema
 view 1-70
 Session Glossary-17
 SET data type 2-62, Glossary-17
 syscolumns coltype code for 1-24

Setting environment variables 3-6
 Shared library Glossary-17
 Shared memory Glossary-17
 client
 (INFORMIXSHMBASE) 3-48
 parameter file (ONCONFIG) 3-55
 Shell
 file for environment variables 3-6
 remote (DBREMOTECMD) 3-33
 remote, overriding
 (DBREMOTECMD) 3-33
 search path (PATH) 3-56
 Simple large object 2-19,
 Glossary-18
 cluster key Glossary-3
 length (syscolumns) 1-29
 location 1-18
 sysblobs information 1-18
 Simple predicate Glossary-18
 Singleton SELECT statement
 definition of Glossary-18
 SMALLFLOAT data type 2-64
 casts for 2-27
 syscolumns coltype code for 1-24
 SMALLINT data type 2-64
 casts for 2-27, 2-28
 length (syscolumns) 1-27
 syscolumns coltype code for 1-24
 Smart large object 2-17,
 Glossary-18
 attributes in syscolattns 1-20
 Software dependencies Intro-5
 Sorting
 parallel process performance
 (PSORT_NPROCS) 3-59
 temporary dbspace for
 (DBSPACETEMP) 3-34
 temporary directory for
 (PSORT_DBTEMP) 3-58
 threads for
 (PSORT_NPROCS) 3-59
 Source file Glossary-18
 Space ()
 as delimiter in DATETIME 2-42
 as delimiter in INTERVAL 2-50
 Specific name Glossary-18
 sysprocedures field 1-46
 SPL Glossary-18
 SPL function Glossary-18

- SPL procedure Glossary-18
- SPL routine Glossary-18
- SQL API Glossary-19
- SQL Communications Area (SQLCA)
 - DBANSIWARN with 3-16
- SQL statements
 - DBANSIWARN environment variable 3-16
 - editor for (DBEDIT) 3-29
 - print program for (DBPRINT) 3-32
- SQL3 Glossary-19
- sqlhosts file 3-49
- sql_languages Information Schema view 1-70
- Stacksize, setting
 - INFORMIXSTACKSIZE 3-49
- state table in stores7 database, columns in A-7
- Statement Glossary-19
 - block Glossary-19
 - identifier Glossary-19
- Status variable Glossary-19
- stock table in stores7 database, columns in A-5
- Stored procedure Glossary-19
- Stored Procedure
 - Language Glossary-19
- stores7 database Intro-6
 - call_type table columns A-6
 - catalog table columns A-5
 - customer table columns A-2
 - cust_calls table columns A-6
 - data values A-16
 - description of A-1
 - items table columns A-4
 - manufact table columns A-7
 - map of A-8
 - orders table columns A-3
 - primary-foreign key relationships A-9 to A-16
 - stock table columns A-5
 - structure of tables A-2
- Strategy functions Glossary-19
- Subquery
 - definition of Glossary-19
- Subtable Glossary-19
- Subtype Glossary-19

- Supertable Glossary-19
- Supertype Glossary-20
- Support functions Glossary-20
- Synonym
 - syssynonyms information 1-51
 - syssyntable information 1-52
- syscasts 2-25
- syscolattrs 1-20
- syscolauth
 - example 1-8
- syscolumns 1-23
 - example 1-7
- sysconstraints 1-29
- sysdefaults 1-30
- sysdepend 1-32
- sysdistrib 1-32
- syserrors 1-33
- sysfragauth 1-34
- sysfragments 1-36
- sysindexes
 - view 1-39
- sysindices 1-37
- sysinherits 1-39
- syslangauth 1-40
- syslogmap 1-40
- Sysmaster databases Glossary-20
- sysobjstate 1-40
- sysopclasses 1-42
- sysprocauth 1-44
- sysprocedures 1-46
- sysprocplan 1-48
- sysreferences 1-49
- sysroleauth 1-50
- sysroutinelangs 1-51
- syssynonyms 1-51
- syssyntable 1-52
- systabauth 1-8, 1-53
- systables 1-54
 - example 1-6
- System catalog 1-3 to 1-71
 - accessing 1-10
 - altering contents not recommended 1-10
 - character variable length 1-27
 - collection data type elements 1-17
 - column information constraints (sysconstraints) 1-29

- default values
 - (sysdefaults) 1-30
- complex data type members 1-17
- constraint information 1-20, 1-23
- data distribution
 - information 1-32
- data type privileges 1-63
- database-level privileges 1-60
- description of Glossary-20
- execute on routine privileges 1-44
- how used by database server 1-5
- list of tables 1-11
- messages 1-33
- object states 1-40
- operator class table (sysopclasses) 1-42
- optical cluster 1-43
- privilege information
 - user-defined data type 1-63
- privileges needed 1-10
- programming language 1-40
- programming languages
 - supported 1-51
- referential constraint
 - information 1-49
- referential constraints
 - (sysreferences) 1-49
- role names for privileges 1-50
- routine plans 1-48
- routines 1-46
- smart large object attributes (syscolattrs) 1-20
- structure 1-11
- sysams 1-12
- sysattrtypes 1-17
- sysblobs table 1-18
- syscasts 1-19
- syschecks 1-20
- syscolauth 1-22
- syscoldepend 1-23
- sysindices 1-9
- sysopclstr 1-43
- sysprocbody 1-45
- sysviews 1-7
- sysxtddesc 1-62
- sysxtdtypeauth 1-63
- sysxtdtype 1-64
- table fragment privileges 1-34
- table fragmentation 1-36

- table indexes 1-37
- table information
 - dependencies (sysdepend) 1-32
- table inheritance 1-39
- table synonyms 1-51, 1-52
- table-level privileges 1-53
- tables in database 1-54
- trace classes 1-56
- trace messages 1-57
- triggers 1-59
- UPDATE STATISTICS for
 - routines 1-48
- UPDATE STATISTICS
 - results 1-32
- updating statistics 1-11
- updating tables 1-10
- user-defined casts 1-19
- user-defined data type
 - descriptions 1-62
- view dependencies
 - (sysdepend) 1-32
- view synonyms 1-52
- views 1-61
- violations 1-61
- System catalog routines
 - systrigbody 1-58
- System indexes
 - idxname
 - used with sysindices 1-39
 - idxtab
 - used with sysindices 1-39
- System Monitoring Interface (SMI) Glossary-20
- System-defined cast
 - definition of Glossary-20
- sysraceclasses 1-56
- sysracemsgs 1-57
- sysracemsgs information 1-57
- systriggers 1-59
- sysusers 1-60
- sysviews 1-61
- sysviolations 1-61
- sysxdtypes 2-56, 2-58

T

- tabid 1-54, 1-55
- Table

- definition of Glossary-20
- dependencies 1-32
- hierarchy Glossary-20
- inheritance Glossary-20
- privileges
 - column-specific 1-22
 - syscolauth sample entry 1-8
 - systabauth information 1-53
 - systabauth sample entry 1-8
- structure in stores7 database A-2
- sysdepend (dependencies)
 - information 1-32
- systables information 1-54
- Table inheritance
 - sysinherits information 1-39
- Table synonym
 - syssyntable information 1-52
- tables Information Schema
 - view 1-67
- Tape management
 - default remote shell (DBREMOTECMD) 3-33
 - personal default qualifier for (ARC_DEFAULT) 3-14
 - tctermcap file
 - for(ARC_KEYPAD) 3-14
- Temporary table
 - dbspaces for (DBSPACETEMP) 3-34
 - directory location (DBTEMP) 3-36
- TERM environment variable 3-61
- TERMCAP environment
 - variable 3-61
- Terminal handling
 - DB-Access setup file (INFORMIXTERM) 3-50
 - termcap file location (TERMCAP) 3-61
 - terminal type (TERM) 3-61
 - terminfo file location (TERMINFO) 3-62
- TERMINFO environment
 - variable 3-62
- TEXT data type
 - collation 2-66
 - control characters with 2-65
 - definition of Glossary-20
 - inserting values 2-66

- Large object
 - character 2-65
 - length (syscolumns) 1-29
 - selecting a column 2-66
 - SQL restrictions 2-65
 - sysblobs information 1-18
 - syscolumns coltype code for 1-24
- Text editor, specifying with
 - DBEDIT 3-24
- THREADLIB environment
 - variable 3-63
- Time data type
 - length (syscolumns) 1-28
 - list of 2-9
- To 1-34
- Transaction Glossary-20
- Transaction logging Glossary-21
- Transaction mode Glossary-21
- Trigger
 - definition of Glossary-21
 - object mode (sysobjstate) 1-40
 - systrigbody (linearized code) 1-58
 - systriggers information 1-59
- Type
 - inheritance Glossary-21
 - substitutability Glossary-21
- Type constructor Glossary-21
- Type hierarchy Glossary-21
- Typed collection
 - variable Glossary-21
- Typed table Glossary-21
- Types 2-9

U

- Unique
 - constraint Glossary-21
 - index Glossary-21
- Unique numeric code
 - with SERIAL data type 2-59
 - with SERIAL8 data type 2-61
- UNIX
 - BSD, default print capability 3-32
 - directories for intermediate
 - writes 3-58
 - environment variables 3-3
 - PATH environment variable 3-56

- System V
 - default print capability 3-32
 - terminfo library support 3-50
- TERM environment variable 3-61
- TERMCAP environment
 - variable 3-61
- TERMINFO environment
 - variable 3-62
- Unnamed row type 2-67
 - definition of Glossary-22
- Untyped collection
 - variable Glossary-22
- UPDATE STATISTICS statement
 - disk space for
 - (DBUPSPACE) 3-39
 - sysdistrib results from 1-32
 - sysindices (index statistics) 1-39
 - update system catalog 1-11
- User-defined data type
 - definition of Glossary-22
 - kinds of 2-20
 - privileges
 - sysxtdtypeauth 1-63
 - sysxtddesc (extended type)
 - information 1-62
 - sysxtdtypes information
 - about 1-64
- User-defined procedure
 - definition of Glossary-22
- User-defined routine
 - description of Glossary-22
 - error messages (syserrors) 1-33
 - privileges
 - sysprocauth 1-44
- Utility program
 - chkenv 3-9

V

- VARCHAR data type 2-69
 - collation 2-70
 - coltype code for 1-24
 - length in syscolumns 1-27
- Variable
 - definition of Glossary-22
- View
 - dependencies 1-32
 - description of Glossary-22

- sysdepend (dependencies) 1-32
- syssynonyms (synonym) 1-51
- syssyntable (synonym) 1-52
- sysviews information 1-61
- Violations table Glossary-23
- Violations tables
 - sysviolations information 1-61
- Virtual column Glossary-23
- Virtual processor Glossary-23
- Virtual table Glossary-23

W

- Warning
 - creating a new warning
 - message 1-34
- White space Glossary-23
- Wildcard Glossary-23

X

- X/Open
 - Information Schema views 1-65
 - server_info view and 1-71
- X/Open compliance
 - level Intro-16
- X/Open-compliant databases 1-71

Symbols

- (), space, as delimiter
 - in DATETIME 2-42
 - in INTERVAL 2-50
- , hyphen, as delimiter
 - in DATETIME 2-42
 - in INTERVAL 2-50
- ., decimal point, as delimiter
 - in INTERVAL 2-50
- /etc/termcap 3-62
- :, colon, as delimiter
 - in DATETIME 2-42
 - in INTERVAL 2-50
- `, decimal point, as delimiter
 - in DATETIME 2-42

