Informix Guide to SQL

Reference

Version 7.1
December 1994
Part No. 000-7632

Published by INFORMIX® Press

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

Documentation Team: Sally Cox, Mary Kraemer, Geeta Karmarkar, Steve Klitzing, Catherine Lyman, Judith Sherwood, Eileen Wollam

Preface

The *Informix Guide to SQL: Reference* is intended to be used as a companion volume to the *Informix Guide to SQL: Tutorial* and the *Informix Guide to SQL: Syntax*. Like the other books in this series, this book is written for people who use SQL. It assumes that you already know how to use computers and who rely on them in their daily work.

Whereas the *Informix Guide to SQL: Tutorial* explains the philosophy and concepts behind relational databases, this volume and the *Informix Guide to SQL: Syntax* are references that you can use on a daily basis after you finish reading and experimenting with the *Informix Guide to SQL: Tutorial*. The *Informix Guide to SQL: Tutorial* contains material that you need to know when writing SQL-based applications with an Informix product.

You must have the following Informix software:

- An INFORMIX-OnLine Dynamic Server database server or an INFORMIX-SE database server.

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

- Either an Informix application development tool, such as INFORMIX-4GL; an SQL application programming interface (API), such as INFORMIX-ESQL / C; or the DB-Access database access utility, which is shipped as part of your database server.

  The application development tool, 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.

Summary of Chapters

The *Informix Guide to SQL: Reference* includes the following chapters:

- This Preface provides general information about the manual and lists additional reference materials that will help you understand Structured Query Language (SQL) concepts.
- The "Introduction" describes the Informix family of products and manuals, explains how to use this manual, introduces the demonstration database from which the product examples are drawn, and lists the new features for Version 7.1 of Informix database server products.
- Chapter 1, "Informix Databases," explains differences between the INFORMIX-OnLine Dynamic Server and INFORMIX-SE database servers, provides information about ANSI compliance in Informix databases, and provides a discussion of the Native Language Support (NLS).
- Chapter 2, "System Catalog," provides details of the Informix system catalog, which is a collection of system catalog tables that describe the structure of the **stores7** and other Informix databases. 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 include information about Information Schema Views.
- Chapter 3, "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 4, "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.
- Chapter 5, "SQL Utilities," contains the syntax and usage of the user utilities. These utilities allow you to import and export data to your databases, load files, and perform other tasks.

- Appendix A, "The stores7 Database," 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 common database terms follows the chapters, and a comprehensive index directs you to areas of particular interest.

Informix Welcomes Your Comments

A reader-response card is provided with this manual. Please use this card to tell us what you like or dislike about this manual. To help us with future versions of the manual, please tell us about any corrections or clarifications that you might find useful. Return this card to:

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

If you prefer to share your comments on-line, address your e-mail to:

doc@informix.com

Related Reading

If you want additional technical information on database management, consult the following books. The first book is an introductory text for readers who are new to database management, while the second book is a more complex technical work for SQL programmers and database administrators.

- *Database: A Primer* by C. J. Date (Addison-Wesley Publishing, 1983)
- *An Introduction to Database Systems* by C. J. Date (Addison-Wesley Publishing, 1994).

If you are interested in learning more about the SQL language, consider the following books:

- *A Guide to the SQL Standard* by C.J. Date with H. Darwen (Addison-Wesley Publishing, 1993)
- *Understanding the New SQL: A Complete Guide* by J. Melton and A. Simon (Morgan Kaufmann Publishers, 1993)
- *Using SQL* by J. Groff and P. Weinberg (Osborne McGraw-Hill, 1990)

This manual assumes that you are familiar with your computer operating system. If you have limited UNIX system experience, you might want to look at your operating system manual or a good introductory text before you read this manual. Some texts about UNIX systems are suggested in the following list:

- *Introducing the UNIX System* by H. McGilton and R. Morgan (McGraw-Hill Book Company, 1983)
- *Learning the UNIX Operating System*, by G. Todino, J. Strang, and J. Peek (O'Reilly & Associates, 1993)
- *A Practical Guide to the UNIX System*, by M. Sobell (Benjamin / Cummings Publishing, 1989)
- *UNIX for People* by P. Birns, P. Brown, and J. Muster (Prentice-Hall, 1985)
- *UNIX System V: A Practical Guide* by M. Sobell (Benjamin / Cummings Publishing, 1995)

Table of Contents

Chapter 1 Informix Databases

Chapter 2 System Catalog

Appendix A The stores7 Database

Introduction

Structured Query Language (SQL) is an English-like language that you can use when creating, managing, and using relational databases. The SQL provided with Informix products is an enhanced version of the industry-standard query language developed by International Business Machines Corporation (IBM).

In addition to SQL, Informix provides the Stored Procedure Language (SPL) with which you can write stored procedures. Stored procedures are programs that are stored as database objects.

Informix Products That Use SQL

Informix produces many application development tools and SQL application programming interfaces (API). Application development tools currently available include products such as INFORMIX-SQL, INFORMIX-4GL, and INFORMIX-NewEra. SQL APIs currently available include INFORMIX-ESQL/C and INFORMIX-ESQL/COBOL.

The INFORMIX-NewEra development environment can access Informix database servers directly through embedded SQL statements as well as through the call-level interface of the CCL/INFORMIX and CCL/ODBC function libraries.

Informix products work with a database server, either INFORMIX-OnLine Dynamic Server, INFORMIX-SE, or an INFORMIX-Gateway product. The DB-Access database access utility is shipped as a part of each database server.

If you are running client applications developed with Version 4.1 and 5.0 application development tools, you use INFORMIX-NET to connect the client to the network.

Products Covered in This Manual

All the information presented in this manual is valid for the following products. Differences in their use of SQL are indicated where appropriate.

- INFORMIX-ESQL / C, Version 7.1
- INFORMIX-ESQL / COBOL, Version 7.1
- INFORMIX-OnLine Dynamic Server, Version 7.1
- INFORMIX-SE, Version 7.1
- INFORMIX-Enterprise Gateway, Version 7.1

Other Useful Documentation

You can refer to the following related Informix documents that complement this manual:

- A companion volume to the Reference, the *Informix Guide to SQL: Tutorial*, provides a tutorial on SQL as it is implemented by Informix products. It describes the fundamental ideas and terminology that are used when planning, using, and implementing a relational database.
- An additional companion volume to the Reference, the *Informix Guide to SQL: Syntax*, provides a detailed description of all the SQL statements supported by Informix products. This guide also provides a detailed description of Stored Procedure Language (SPL) statements.
- The *SQL Quick Syntax Guide* contains syntax diagrams for all statements and segments described in this manual.
- You, or whoever installs your Informix products, should refer to the *UNIX Products Installation Guide* for your particular release to ensure that your Informix product is properly set up before you begin to work with it. A matrix depicting possible client / server configurations is included in the *UNIX Products Installation Guide*.

- Depending on the database server you are using, you or your system administrator need either the *INFORMIX-SE Administrator's Guide* or the *INFORMIX-OnLine Dynamic Server Administrator's Guide*.
- The *DB-Access User Manual* describes how to invoke the DB-Access utility to access, modify, and retrieve information from Informix database servers.
- When errors occur, you can look them up by number and learn their cause and solution in the *Informix Error Messages* manual. If you prefer, you can look up the error messages in the on-line message file described in the section "ASCII and PostScript Error Message Files" on page -10 later in this Introduction and in the Introduction to the *Informix Error Messages* manual.

How to Use This Manual

This manual assumes that you are using INFORMIX-OnLine Dynamic Server as your database server. Features and behavior specific to INFORMIX-SE are noted throughout the manual.

The following sections describe the conventions used in this manual for typographical format, syntax, and examples of code.

Typographical Conventions

Informix product manuals use a standard set of conventions to introduce new terms, illustrate screen displays, describe command syntax, and so on. The following typographical conventions are used throughout this manual:

italics	New terms, emphasized words, and variables are printed in italics.
boldface	Database names, table names, column names, file names, utilities, and other similar terms are printed in boldface.

`computer`	Information that OnLine displays and information that you enter is printed in a computer typeface.
KEYWORD	All keywords appear in uppercase letters.
✦	The diamond symbol appears at the beginning and the end of product-specific information.
	This symbol indicates a unique identifier (primary key) for each table.
	This symbol indicates a *warning*. Warnings provide critical information that, if ignored, could cause harm to your database.
	This symbol indicates *important* information that you should consider when working with the product.
	This symbol indicates a *tip*. It alerts you to useful information that, for instance, might indicate a shortcut or make it easier to navigate in the product or manual.

Additionally, when you are instructed to "enter" or "execute" text, immediately press RETURN after the entry. When you are instructed to "type" the text or "press" a key, no RETURN is required.

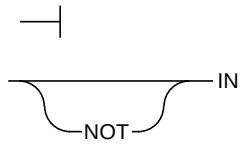
Command-Line Conventions

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

Each diagram displays the sequences of required and optional elements that are valid in a command. A diagram begins at the upper left with a command. It ends at the upper right 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.

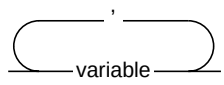
Along a command-line path, you might encounter the following elements:

command	This required element is usually the product name or other short word used to invoke the product or call 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 must use lowercase letters.
variable	A word in italics represents a value that you must supply. The nature of the value is explained immediately following the diagram unless the variable appears in a box. In that case, the page number of the detailed explanation follows the variable name.
-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 representing a filename. Type this extension exactly as shown, immediately after the name of the file and a period. The extension might be optional in certain products.
(.,;+*-/)	Punctuation and mathematical notations are literal symbols that you must enter exactly as shown.
' '	Single quotes are literal symbols that you must enter as shown.
Privileges p. XXX	A reference in a box represents a subdiagram on the same page (if no page number is supplied) or on a specified page. Imagine that the subdiagram is spliced into the main diagram at this point.
ALL	A shaded option is the default. Even if you do not explicitly type the option, it will be in effect unless you choose another option.
	Syntax enclosed in a pair of arrows indicates that this is a subdiagram.

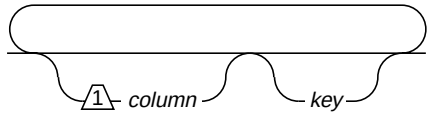


The vertical line is a terminator and indicates that the statement is complete.

A branch below the main line indicates an option



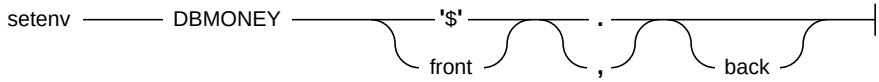
A loop indicates a path that can be repeated. Punctuation along the top of the loop indicates the separator symbol for list items.



A gate (1) in an option indicates that you can only use that option once, even if it is within a larger loop.

Figure 1 shows how to read the command-line diagram for setting the **DBMONEY** environment variable in a C shell:

Figure 1
Elements of a command-line diagram



To construct a correct command, start at the top left with the command `setenv`. Then follow the diagram to the right, including the elements that you want. The elements in the diagram are case-sensitive. This diagram conveys the following information:

1. You must type the command `setenv`.
2. You must supply the keyword **DBMONEY**.
3. You must choose either the $ or another *front* symbol.
4. You must choose either the period or comma as the symbol that separates the integral from the fractional part of the money value.
5. You can specify a *back* option or no *back* option.
6. After your *back* option choice, you come to the terminator. Your `setenv` command is complete. Press RETURN to execute the command.

As shown in the following example, you can set the **DBMONEY** environment variable for Canadian currency using the string '$CAN' for *front*, using the period as the separator, and omitting the *back* option:

```
setenv DBMONEY '$CAN'.
```

Example 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 delineated by semicolons. To use this SQL code for a specific product, you must apply the syntax rules for that product. For example, if you are using the Query-language option of DB-Access, you must delineate multiple statements with semicolons. If you are using an SQL API, you must use EXEC SQL and a semicolon (or other appropriate delimiters) at the start and end of each statement, respectively.

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
```

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

Also note that dots in the 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.

Useful On-Line Files

In addition to the Informix set of manuals, the following on-line files, located in the **$INFORMIXDIR/release** directory, might supplement the information in this manual:

Documentation Notes	describe features not covered in the manual or that have been modified since publication. The file containing the Documentation Notes for this product is called **SQLRDOC_7.1**.
Release Notes	describe feature differences from earlier versions of Informix products and how these differences might affect current products. The file containing the Release Notes for Version 7.1 of Informix database server products is called **SERVERS_7.1**.
Machine Notes	describe any special actions required to configure and use Informix products on your computer. Machine Notes are named for the product described, for example, the Machine Notes file for INFORMIX-OnLine Dynamic Server is **ONLINE_7.1**.

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

ASCII and PostScript Error Message Files

Informix software products provide ASCII files that contain all the Informix error messages and their corrective actions. To access 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**). See the Introduction to the *Informix Error Messages* manual for a detailed description of these scripts.

The optional Informix Messages and Corrections product provides PostScript files that contain the error messages and their corrective actions. If you have installed this product, you can print the PostScript files on a

PostScript printer. The PostScript error messages are distributed in a number of files of the format **errmsg1.ps**, **errmsg2.ps**, and so on. These files are located in the **$INFORMIXDIR/msg** directory.

The 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. The sample command files that make up a demonstration application are also included.

Most examples in this manual 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. The database name that you supply is the name given to the demonstration database. If you do not supply a database name, the name defaults to **stores7**. Use the following rules for naming your database:

- Names can be up to 18 characters long for INFORMIX-OnLine Dynamic Server databases and up to 10 characters long for INFORMIX-SE databases.
- The first character of a name must be a letter or an underscore (_).
- You can use letters, characters, and underscores (_) for the rest of the name.
- DB-Access makes no distinction between uppercase and lowercase letters.
- The database name must be unique.

When you run **dbaccessdemo7**, you are, as the creator of the database, the owner and Database Administrator (DBA) of that database.

If you install your Informix database server according to the installation instructions, the files that comprise the demonstration database are protected so you cannot make any changes to the original database.

You can run the **dbaccessdemo7** script again whenever you want to work with a fresh demonstration database. The script prompts you when the creation of the database is complete and asks if you would like to copy the sample command files to the current directory. Enter `N` if you have made changes to the sample files and do not want them replaced with the original versions. Enter `Y` if you want to copy over the sample command files.

Creating the Demonstration Database

Use the following steps to create and populate the demonstration database:

1. Set the **INFORMIXDIR** environment variable so that it contains the name of the directory in which your Informix products are installed. Set **INFORMIXSERVER** to the name of the default database server. The name of the default database server must exist in the **$INFORMIXDIR/etc/sqlhosts** file. (For a full description of environment variables, see Chapter 4, "Environment Variables.") For information about **sqlhosts**, see the *INFORMIX-OnLine Dynamic Server Administrator's Guide* or the *INFORMIX-SE Administrator's Guide*.
2. Create a new directory for the SQL command files. Create the directory by entering the following command:

   ```
   mkdir dirname
   ```
3. Make the new directory the current directory by entering the following command:

   ```
   cd dirname
   ```

4. Create the demonstration database and copy over the sample command files by entering the **dbaccessdemo7** command.

   To create the database without logging enter the follwoing command:

   ```
   dbaccessdemo7 dbname
   ```

   To create the demonstration database with logging enter the following command:

   ```
   dbaccessdemo7 -log dbname
   ```

   If you are using INFORMIX-OnLine Dynamic Server, by default the data for the database is put into the root dbspace. If you wish, you can specify a dbspace for the demonstration database.

   To create a demonstration database in a particular dbspace enter the following command:

   ```
   dbaccessdemo7 dbname -dbspace dbspacename
   ```

   You can specify all the options in one command as shown in the following command:

   ```
   dbaccessdemo7 -log dbname -dbspace dbspacename
   ```

   If you are using INFORMIX-SE, a subdirectory called ***dbname*.dbs** is created in your current directory and the database files associated with **stores7** are placed there. You will see both data (**.dat**) and index (**.idx**) files in the ***dbname*.dbs** directory. (If you specify a dbspace name, it will be ignored.)

   To use the database and the command files that have been copied to your directory, you must have UNIX read and execute permissions for each directory in the pathname of the directory from which you ran the **dbaccessdemo7** script. Check with your system administrator for more information about operating-system file and directory permissions. UNIX permissions are discussed in the *INFORMIX-OnLine Administrator's Guide* and the *INFORMIX-SE Administrator's Guide*

5. To give someone else the permissions to access the command files in your directory, use the UNIX `chmod` command.

6. To give someone else the access to the database that you have created, grant them the appropriate privileges using the GRANT statement. To remove privileges, use the REVOKE statement. The GRANT and REVOKE statements are described in Chapter 1 of the *Informix Guide to SQL: Syntax*.

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.L35-1992), which is identical to ISO 9075:1992 on INFORMIX-OnLine Dynamic Server. In addition, many features of OnLine comply with the SQL-92 Intermediate and Full Level and X/Open CAE (common applications environment) standards.

Informix SQL-based products are compliant with ANSI SQL-92 Entry Level (published as ANSI X3.135-1992) on INFORMIX-SE with the following exceptions:

- Effective checking of constraints
- Serializable transactions

New Features in Informix Version 7.1 Products That Use SQL

The Introduction to each Version 7.1 product manual contains a list of new features for that product. The Introduction to each manual in the Version 7.1 *Informix Guide to SQL* series contains a list of new SQL features.

A comprehensive list of all the new features for Version 7.1 Informix products is in the Release Notes file called **SERVERS_7.1**.

This section highlights the major new features implemented in Version 7.1 of Informix products that use SQL.

- New FRAGMENT BY clause in the CREATE TABLE statement

  The CREATE TABLE statement contains a new FRAGMENT BY clause that allows you to create fragmented tables. Fragmentation enables you to define groups of rows within a table based on a rule and to specify a separate dbspace for each group.

- New FRAGMENT BY EXPRESSION clause in the CREATE INDEX statement

  The CREATE INDEX statement contains a new FRAGMENT BY EXPRESSION clause that allows you to create fragmented indexes.

- New ALTER FRAGMENT statement

  You can use the ALTER FRAGMENT statement to dynamically alter an existing table or index-fragmentation strategy as well as to initially create fragments.

- New ADD ROWIDS and DROP ROWIDS clauses in the ALTER TABLE statement

  You can use the ADD ROWIDS clause to add the rowid column to fragmented tables. You can use the DROP ROWIDS clause to drop the rowid column from a fragmented table that has a rowid column.

- New FRAGMENTS keyword in INFO statement

  The FRAGMENTS keyword allows you to display the dbspace names where fragments are located for a specified table.

- New SET DATASKIP statement

  The SET DATASKIP statement allows you to control whether INFORMIX-OnLine Dynamic Server skips any dbspaces in the fragmentation list that are unavailable while a transaction is being processed.

- New **sysfragments** system catalog table

  The **sysfragments** table stores fragmentation information for tables and indexes.

- New SET PDQPRIORITY statement

  The SET PDQPRIORITY statement supports the parallel-data-query (PDQ) functionality. PDQ allows queries to be processed in parallel. The SET PDQPRIORITY statement allows an application to set the query priority level dynamically within the application. The query priority level refers to the degree of parallelism to be used for queries.

- New output in the SET EXPLAIN statement

  The output of the SET EXPLAIN statement has been enhanced to support the new fragmentation and PDQ features.

- Changes to the **dbexport** and **dbschema** utilities

  The **dbexport** and **dbschema** utilities have been enhanced to support the new fragmentation and PDQ features.

- New SET TRANSACTION statement

  You can now use the ANSI isolation levels for transactions. In addition, the SET TRANSACTION statement permits you to use read-only and read-write access modes.

- New TRIM function

  The TRIM function allows you to strip leading characters or blanks, trailing characters or blanks, or both from a character string.

- Support for Delimited Identifiers

  Delimited identifiers allow you to specify as identifiers character strings that are otherwise identical to SQL reserved keywords. They also allow you to specify nonalphanumeric characters in database object names. This feature is only activated when the **DELIMIDENT** environment variable is set.

- WITH REOPTIMIZATION clause for the OPEN CURSOR statement

  The WITH REOPTIMIZATION clause allows you to reoptimize a query plan if that query plan becomes deoptimized.

- INTO clause for the EXECUTE statement

  The INTO clause allows you to execute a prepared singleton select statement or execute a prepared EXECUTE PROCEDURE statement. In addition, you can specify output variables using the INTO clause.

- New Environment Variables

  The following environment variables, described in Chapter 4, "Environment Variables," are new in Version 7.1:

  - **DBBLOBBUF**
  - **DELIMIDENT**
  - **BUF_FET_SIZE**
  - **OPTCOMPIND**
  - **PDQPRIORITY**

- XPG4 SQL CAE compliance
  - X/Open information schemas

    The information schema consists of a set of read-only views from which you can retrieve information about any tables, views, and columns to which you have access.
  - Support for CHARACTER VARYING syntax

    The CHARACTER VARYING data type syntax, which is compliant with XPG4 and ANSI, performs like the Informix data type VARCHAR. The data type supports both a maximum and minimum length.
  - Specification of cursors as READ ONLY

    You can specify cursors as READ ONLY with the DECLARE statement.
  - Optional specification of RESTRICT/CASCADE on the DROP TABLE, DROP VIEW, and REVOKE statements

    You can now restrict drop and revoke actions with the RESTRICT keyword. Previously, all drop requests to tables and views and revoke requests cascaded to drop all related tables, views, and permissions. Now you can specify the RESTRICT keyword on these statements to disallow the drop or revoke.
- Hexadecimal load/unload format

  This feature allows the LOAD/UNLOAD statements, as well as the **dbexport**, **dbimport**, and **dbload** utilities, to format the nonprintable ASCII characters in hexadecimal format. It affects the formats of unload files if the CHAR/VARCHAR NCHAR/NVARCHAR data types contain nonprintable ASCII characters. It also allows you to load the nonprintable characters from the file into the database after they are prepared in hex format.

Chapter

1

Informix Databases

Before you create a database using an Informix product, you must make several decisions that affect which features will be available to applications that use the database. The following major decisions are:

- Which database server will house the database: INFORMIX-OnLine Dynamic Server or INFORMIX-SE?
- Does the database need to be ANSI-compliant?
- Will the database use characters from a language other than English in its tables?

This chapter describes the implications of each choice and summarizes how these choices affect your databases. For many of the features and behaviors described in this chapter, the specifics of implementation are discussed elsewhere in this Reference, the *Informix Guide to SQL: Syntax* or in the *Informix Guide to SQL: Tutorial*.

Choosing a Database Server

If you have access to both an INFORMIX-OnLine Dynamic Server and an INFORMIX-SE database server, you must choose one to serve your database. If you create a database with one database server, you can transfer it later to the other database server. Informix recommends, however, that you consider which database server is more appropriate for your needs before you create and populate the database.

Various factors might influence your choice of database server, for example, the size of your databases, speed, the need for distributed access, and ease of installation and maintenance. You should also be aware of differences in the SQL features that are available on INFORMIX-OnLine Dynamic Server and INFORMIX-SE. These differences are summarized in this section.

Your choice of database server decides the following issues for a database:

- Which data types are available
- Whether data-definition statements can be rolled back
- Whether buffered logging is supported
- Whether tables can be fragmented
- Which locking options are available
- Which isolation levels are available
- What information is stored in the system catalog tables
- Which SQL statements are available

This section summarizes how your choice of database server affects each of these issues. References are also provided to indicate where you can find more extended discussions of these issues.

Data Types

INFORMIX-OnLine Dynamic Server supports all the data types available to INFORMIX-SE as well as the following additional data types:

- BYTE
- CHARACTER VARYING
- TEXT
- VARCHAR
- NVARCHAR

INFORMIX-SE and INFORMIX-OnLine Dynamic Server also have different values for the maximum length of CHAR(*n*) and NCHAR(*n*) data types.

See "Database Data Types" on page 3-3 for more information on individual data types.

Rolling Back Statements in a Transaction

For both database servers, if you have not yet issued a COMMIT WORK statement for a transaction, the ROLLBACK WORK statement allows you to undo any changes that occurred since the beginning of that transaction. In INFORMIX-OnLine Dynamic Server, you can roll back any statement.

If you are using INFORMIX-SE, you cannot undo certain operations with ROLLBACK WORK. These operations comprise all the data definition statements as well as the GRANT and REVOKE statements. (For a list of the data definition statements, refer to Chapter 1 of the *Informix Guide to SQL: Syntax*.)

For a discussion of transactions, refer to Chapter 4 of the *Informix Guide to SQL: Tutorial*. See also the description of the ROLLBACK WORK statement in Chapter 1 of the *Informix Guide to SQL: Syntax*.

Transaction Logging

INFORMIX-OnLine Dynamic Server supports buffered logging and allows you to change logging from buffered to unbuffered or unbuffered to buffered with the SET LOG statement. You can also choose to log or not to log.

INFORMIX-SE does not support buffered logging.

For more information on the SET LOG statement, refer to Chapter 1 of the *Informix Guide to SQL: Syntax*. See the *INFORMIX-OnLine Dynamic Server Administrator's Guide* for more information about buffered and unbuffered logging.

Table and Index Fragmentation

INFORMIX-OnLine Dynamic Server supports the fragmentation of tables and indexes. The feature allows you to group rows within a table according to a distribution scheme. The rows are stored in separate dbspaces that you specify in a fragmentation strategy.

For more information about fragmentation and the SQL statements you use to created fragmented tables and indexes, see Chapter 1 of the *Informix Guide to SQL: Syntax*. See the *INFORMIX-OnLine Dynamic Server Performance Guide* for more information about how fragmentation can enhance performance.

Locking Issues

When you write applications for INFORMIX-OnLine Dynamic Server and INFORMIX-SE, you should be aware of how each handles the following locking issues:

- Which choices of lock scope are available
- Which choices of lock mode are available
- How shared locks are used
- Whether a process can wait for a lock

The following four sections briefly describe how the different database servers handle these locking issues.

Lock Scope

When you create a table, INFORMIX-OnLine Dynamic Server sets the lock scope to page level by default. If you want, you can set the lock scope to row level instead.

INFORMIX-SE supports only row-level locking.

Refer to Chapter 7 of the *Informix Guide to SQL: Tutorial* for more information on lock scope. See the descriptions of CREATE TABLE or ALTER TABLE in Chapter 1 of the *Informix Guide to SQL: Syntax* for how to specify a lock scope.

Lock Mode

INFORMIX-OnLine Dynamic Server supports three locking modes: shared, exclusive, and promotable.The OnLine database server implicitly places promotable locks on the rows, which are selected for an update operation. Just before the actual update, the OnLine promotes the lock to exclusive.

The INFORMIX-SE database server supports only shared and exclusive locking. This results in different locking strategies for table updates. The INFORMIX-SE database server does not have promotable locking, so it places exclusive locks on rows that are selected for update.

For more information on lock modes, refer to Chapter 7 of the *Informix Guide to SQL: Tutorial*. See also the description of the UPDATE statement in Chapter 1 of the *Informix Guide to SQL: Syntax*.

Use of Shared Locks

INFORMIX-OnLine Dynamic Server supports shared locks on tables. Multiple users may place shared locks on a table.

INFORMIX-SE also supports shared locks on tables. However, the INFORMIX-SE database server allows only one user at a time to have a shared lock on a table.

For more information, see the description of LOCK TABLE in Chapter 1 of the *Informix Guide to SQL: Syntax*.

Waiting for Locks

The SET LOCK MODE statement is always available with INFORMIX-OnLine Dynamic Server. This statement specifies whether a process that tries to access a locked row or table must wait until OnLine releases the lock.

Only those INFORMIX-SE database servers that use kernel locking have the SET LOCK MODE statement available.

For more information, refer to Chapter 7 of the *Informix Guide to SQL: Tutorial*. See also the description of SET LOCK MODE in Chapter 1 of the *Informix Guide to SQL: Syntax*.

Isolation Level

INFORMIX-OnLine Dynamic Server supports the SET TRANSACTION statement for databases that use transaction logging. Using this option, you can set the isolation level to Read Uncommitted, Read Committed, Repeatable Read, and Serializable. The default isolation level is Read Committed, unless the database is ANSI-compliant, in which case the default is Serializable.

INFORMIX-SE supports the SET TRANSACTION statement for access modes only. All databases running on INFORMIX-SE use the Read Uncommitted isolation level.

For more information, refer to the description of the SET TRANSACTION statement in Chapter 1 of the *Informix Guide to SQL: Syntax*. See also Chapter 7 of the *Informix Guide to SQL: Tutorial*.

System Catalog Tables

The following system catalog tables in an INFORMIX-SE database are defined or used differently than their corresponding system catalog tables in an INFORMIX-OnLine Dynamic Server database:

- **syscolumns**
- **sysindexes**
- **syssyntables**
- **systables**
- **sysreferences**

The following system catalog tables exist only on INFORMIX-OnLine Dynamic Server database servers:

- **sysfragments**
- **sysopclstr**
- **sysblobs**

For more information on the system catalog tables, see Chapter 2, "System Catalog."

SQL Statements Supported by Specific Database Servers

This section lists which SQL statements are supported by INFORMIX-OnLine Dynamic Server and which statements are supported by INFORMIX-SE. This information also appears in Chapter 1 of the *Informix Guide to SQL: Syntax*, where syntax specific to OnLine and SE is indicated by the appropriate icon.

SQL Statements Supported Only by INFORMIX-OnLine Dynamic Server

The following statements are supported only by OnLine:

- ALTER FRAGMENT
- SET CONSTRAINTS
- SET ISOLATION
- SET LOG

SQL Statements Containing Branches Specific to INFORMIX-OnLine Dynamic Server

The following statements contain at least one option or branch that is supported only by OnLine:

- ALTER TABLE
- CREATE DATABASE
- CREATE TABLE
- REVOKE
- SET LOCK MODE
- SET TRANSACTION

SQL Segments Containing Branches Specific to INFORMIX-OnLine Dynamic Server

Segments are elements of syntax that are extracted from the syntax of the SQL statements. Segments are described in Chapter 1 of the *Informix Guide to SQL: Syntax*. The following segments contain at least one option or branch that is supported only by OnLine:

- Condition
- Constraint
- Database Name
- Expression
- Index Name
- Procedure Name
- Synonym Name
- Table Name
- View Name

Important: *If one of the segments in the preceding list contains a branch that is specific to INFORMIX-OnLine Dynamic Server, the entire SQL statement is supported only by OnLine. For example, if a SELECT statement contains an expression with a branch that is supported only by OnLine, the entire SELECT statement is supported only by an OnLine database server.*

SQL Statements Supported Only by INFORMIX-SE

The following statements are supported only by the INFORMIX-SE database server:

- CHECK TABLE
- CREATE AUDIT
- DROP AUDIT
- RECOVER TABLE
- REPAIR TABLE
- ROLLFORWARD DATABASE
- START DATABASE

SQL Statements and Segments Containing Branches Specific to INFORMIX-SE

The following statements contain at least one option or branch that is supported only by INFORMIX-SE:

- CONNECT
- CREATE DATABASE
- CREATE TABLE

Segments are elements of syntax that are extracted from the syntax of the SQL statements. Segments are described in Chapter 1 of the *Informix Guide to SQL: Syntax*. The Database Name segment is the only segment that contains an option or branch that is supported only by INFORMIX-SE.

Using ANSI-Compliant Databases

An ANSI-compliant database is created using the MODE ANSI keywords. The differences between ANSI-compliant databases and those that are not ANSI-compliant are described on page 1-12.

You might want to create an ANSI-compliant database for the following reasons:

- Privileges and access to objects

  ANSI rules govern privileges and access to objects such as tables and synonyms. However, creating an ANSI-compliant database does not ensure that this database remains compliant with the ANSI/ISO SQL-92 standards. (If you take a non-ANSI action, such as CREATE INDEX, on an ANSI database, you will receive a warning, but it will not forbid the action.)

- Name isolation

  The ANSI table-naming scheme allows different users to create tables in a database without having to worry about name conflicts.

- Transaction isolation

- Data recovery

  ANSI-compliant databases enforce unbuffered logging and automatic transactions for INFORMIX-OnLine Dynamic Server database servers.

Designating a Database as ANSI-Compliant

You designate a database as ANSI-compliant by using the MODE ANSI keywords when you create it. Once you create a database using the MODE ANSI keywords, the database is considered ANSI-compliant. In an ANSI-compliant database, you cannot change the logging mode to buffered logging, and you cannot turn logging off.

Determining If an Existing Database Is ANSI-Compliant

The following list describes several methods to determine if a database is ANSI-compliant:

- If the default database server is INFORMIX-OnLine Dynamic Server, you can use the DB-Monitor utility to list all the databases that reside on that default database server. The Databases option of the Status Menu displays this list. In the Log Status column on the list, an ANSI-compliant database has the notation `U*`.
- If you are using an SQL API such as INFORMIX-ESQL / C, you can test the SQL Communications Area (SQLCA). Specifically, the third element in the SQLCAWARN structure contains a `W` immediately after you open an ANSI-compliant database using the DATABASE or CONNECT statement. For information on SQLCA, see Chapter 5 of the *Informix Guide to SQL: Tutorial* or your SQL API manual.
- If you are running on an INFORMIX-SE database server, you can query the **systables** system catalog table. If a database was created with the MODE ANSI keywords, **systables** lists a row with a **tabname** of ANSI and a **tabid** of 100. For example, if the following query on **systables** returns a row, the database is ANSI-compliant:

```
SELECT * FROM 'informix'.systables WHERE tabname = 'ANSI'
```

Differences between ANSI-Compliant Databases and Databases that are not ANSI-Compliant

Databases that you designate as ANSI-compliant (by using the MODE ANSI keywords when you create them) and databases that are not ANSI-compliant behave differently in the following areas:

- Transactions
- Transaction logging
- Owner naming
- Privileges on objects
- Default isolation level
- Character data types
- Decimal data type

- Escape characters
- Cursor behavior
- SQLCODE of the SQL Communications Area (SQLCA)
- Allowed SQL statements
- Synonym behavior

Transactions

All the SQL statements that you issue in an ANSI-compliant database are automatically contained in transactions. With a database that is not ANSI-compliant, you can choose whether to use transaction processing.

In a database that is not ANSI-compliant, a transaction is enclosed by a BEGIN WORK statement and a COMMIT WORK or a ROLLBACK WORK statement. However, in an ANSI-compliant database, the BEGIN WORK statement is redundant and unnecessary because all statements are automatically contained in a transaction. You need to indicate only the end of a transaction with a COMMIT WORK or ROLLBACK WORK statement.

See Chapter 4 of the *Informix Guide to SQL: Tutorial* for more information on transactions.

Transaction Logging

All ANSI-compliant databases on an INFORMIX-OnLine Dynamic Server database server run with unbuffered transaction logging. Databases that are not ANSI-compliant can run with either buffered logging or unbuffered logging. Unbuffered logging provides more comprehensive data recovery, but buffered logging provides better performance.

See the description of CREATE DATABASE in Chapter 1 of the *Informix Guide to SQL: Syntax* for more information.

Owner Naming

In an ANSI-compliant database, owner naming is enforced. When you supply an object name in an SQL statement, ANSI standards require that the name include the prefix *owner*., unless you are the owner of the object. The combination of *owner* and *name* must be unique in the database. If you are the owner of the object, the database server supplies your user name as the default.

Databases that are not ANSI-compliant do not enforce owner naming.

See "Table Name" in Chapter 1 of the *Informix Guide to SQL: Syntax* for more information.

Privileges on Objects

ANSI-compliant databases and databases that are not ANSI-compliant differ as to which users are granted table-level privileges by default when a table in a database is created. ANSI standards specify that the database server grants only the table owner (as well as the DBA if they are not the same user) any table-level privileges. In a database that is not ANSI-compliant, however, privileges are granted to PUBLIC. In addition, Informix provides two table-level privileges, Alter and Index, that are not included in the ANSI standards.

See Chapter 11 of the *Informix Guide to SQL: Tutorial* and the description of the GRANT statement in Chapter 1 of the *Informix Guide to SQL: Syntax* for more information on granting table-level privileges.

To run a stored procedure, you must have the Execute privilege for that procedure. When you create an owner-privileged procedure for an ANSI-compliant database, only the owner of the stored procedure has the Execute privilege. When you create an owner-privileged procedure in a database that is not ANSI-compliant, the database server grants the Execute privilege to public by default.

See Chapter 14 of the *Informix Guide to SQL: Tutorial* for more information on stored procedure privileges.

Default Isolation Level

The database isolation level specifies the degree to which your program is isolated from the concurrent actions of other programs. The default isolation level for all ANSI-compliant databases is Serializable. The default isolation level for databases that are not ANSI-compliant but do use logging is ANSI Read Committed on an OnLine database server and Informix Dirty Read on an INFORMIX-SE database server. The default isolation level in a database that is not ANSI-compliant and without logging is Read Uncommitted.

See Chapter 7 of the *Informix Guide to SQL: Tutorial* and the description of the SET TRANSACTION and SET ISOLATION statements in Chapter 1 of the *Informix Guide to SQL: Syntax* for more information.

Character Data Types

In an ANSI-compliant database, you get an error if any character field (CHAR, CHARACTER, VARCHAR, NCHAR, NVARCHAR, CHARACTER VARYING) is filled with a string that is longer than the specified width of the field.

Decimal Data Type

In an ANSI-compliant database, no scale is used for the DECIMAL data type. You can think of this as scale=0.

Escape Characters

In an ANSI-compliant database, escape characters can only escape the following characters: percent (%) and underscore(_). An escape character can also be used to escape itself. See the Condition segment in the *Informix Guide to SQL: Syntax* for additional information.

Cursor Behavior

If a database is not ANSI-compliant, you need to use the FOR UPDATE keywords when you declare an update cursor for a SELECT statement. The SELECT statement must also meet the following conditions:

- It selects from a single table.
- It does not include any aggregate functions.
- It does not include the DISTINCT, GROUP BY, INTO TEMP, ORDER BY, UNION, or UNIQUE clauses and keywords.

With an ANSI-compliant database, you do not have to explicitly use the FOR UPDATE keywords when you declare a cursor. In ANSI-compliant databases, *all* cursors that meet the restrictions described in the preceding list are potentially UPDATE cursors. You can specify that a cursor is read-only with the FOR READ ONLY keywords on the DECLARE statement.

See the description of the DECLARE statement in Chapter 1 of the *Informix Guide to SQL: Syntax* for more information.

The SQLCODE Field of the SQL Communications Area

If no rows satisfy the search criteria of a DELETE, an INSERT INTO *tablename* SELECT, a SELECT... INTO TEMP, or an UPDATE statement, the database server sets SQLCODE to 100 if the database is ANSI-compliant and to 0 if the database is not ANSI-compliant.

See the descriptions of SQLCODE in Chapter 5 of the *Informix Guide to SQL: Tutorial* for more information.

SQL Statements Allowed with ANSI-Compliant Databases

No restrictions exist on SQL statements allowed in applications used with an ANSI-compliant database. You can use the Informix extensions with an ANSI-compliant database just as you can with a database that is not ANSI-compliant.

Synonym Behavior

Synonyms are always private in an ANSI-compliant database. If you attempt to create a public synonym, or use the PRIVATE keyword to designate a private synonym in an ANSI-compliant database, you receive an error.

Using Native Language Support

If you plan to use non-English characters in your data or user-specified object names, you need to set your environment to accommodate Native Language Support (NLS). This allows you to work in a European language other than English and to conform to the customs of a specific non-U.S. locale. A database that allows local sorting and other operations on non-English character data is referred to as an *NLS database*. (A non-NLS database accepts non-English character data, if the locale is not set to C. However, sorting and other operations on the non-English character data are done according to ASCII conventions.)

If you do not need to work with non-English character data or identifiers, you can simply not activate NLS. Your Informix products thus perform as intended for a U.S. English environment, and you can skip this section. However, even if your data does not contain non-English characters, you can activate NLS to deviate from the standard ASCII collation order of your data.

Both the INFORMIX-OnLine Dynamic Server and INFORMIX-SE database servers allow database names in national character sets (the latter only if the operating system environment allows it). The associated utilities can display database names, user names, and so on, on the screen in the local language.

Informix NLS-ready products are available only on platforms that support NLS.

Once you activate the NLS functionality built into your Informix product by setting the appropriate environment variables, you can create or access database information in any language available on the system.

The implementation of NLS affects the following operations:

- Processing national characters and strings

  You can name all user-specifiable objects, such as tables, columns, views, statements, cursors, and variables using national character sets and can use a collation order that suits the local customs.

  You can use country-specific values instead of ASCII values to evaluate the WHERE and ORDER BY clauses of your SELECT statements or to sort indexes built on NCHAR and NVARCHAR character fields.

- Evaluation of regular expressions

  You can use country-specific values instead of ASCII values when comparing regular expressions involving NCHAR and NVARCHAR character fields.

- Representation of money and currency symbols

  You can specify monetary values using a format that is not ANSI-compliant for locales outside the U.S.

- Representation of dates and times

  You can specify date and time values in a format that is not ANSI-compliant for locales outside the U.S.

- Accessibility of formerly incompatible character code sets

  You can access and share data from databases that use nonstandard code sets and can use NLS collation features without modifying existing code.

The Informix implementation of NLS is based on the X/Open specification of NLS as outlined in the XPG3 documents. NLS terms are defined in the Glossary of this manual, and specific features are described where appropriate throughout this manual set.

Activating NLS

You set specific environment variables to activate the NLS functionality in Informix products and to create NLS databases. If you do not activate NLS, your Informix products function in English.

Setting Environment Variables for NLS

Set the following environment variables to activate or work with NLS:

DBNLS — Before starting a client application or accessing DB-Access, you must set **DBNLS** to 1 or 2, which enables NLS features.

LANG — Next, Informix recommends that you set **LANG** to specify the *locale* (language environment) for your database operations. Messages, collating sequences, character classifications, and local customs information are all associated with the **LANG** setting. The setting depends on your system and the locale you want to support.

Alternatively, you can set or modify individual characteristics with the environment variables described in the following list. You can set additional environment variables categories for a fuller implementation of NLS features. Set the variables that start with LC_ to modify or override the **LANG** setting.

The following NLS environment variables are listed here alphabetically and all are described in detail in the section "NLS Environment Variables" on page 4-11:

COLLCHAR — Set this *environment* variable if you want previously written client applications to take advantage of the NLS collation feature without changing the code.

DBAPICODE — Set this environment variable so that your client applications on different platforms can use a different code set than the one used by the database server and thus share data. **DBAPICODE** lets systems that use nonstandard or rare code sets access databases that store data using a standard code set.

LC_COLLATE	Set this *environment* VARIABLE to specify a collating or sort sequence for your locale. The setting affects the evaluation of regular expressions and character-string handling involving NCHAR and NVARCHAR data type fields.
LC_CTYPE	Set this *environment* VARIABLE to affect the behavior of regular expressions and character evaluation functions such as case conversion, spaces, and punctuation.
LC_MONETARY	Set this *environment* VARIABLE to specify the format and national currency symbol for money values.
LC_NUMERIC	Set this *environment* VARIABLE to specify the format and decimal separator for numeric values.
LC_TIME	Set this *environment* VARIABLE to define the format for national dates and times, including country-specific names and abbreviations for days of the week and months.

The following example illustrates various NLS environment variable and environment variable settings using UNIX commands in a C shell. Settings in the example activate NLS and specify the German language environment for database operations, the German Swiss dictionary collation order for that locale, and the ISO 8859/1 code set for money values in Swiss francs:

```
setenv DBNLS 1
setenv LANG DE
setenv LC_COLLATE DE_ch@dict
setenv LC_MONETARY CH.88591
```

These NLS environment variables and their interactions with Informix environment variables are described in detail in Chapter 4, "Environment Variables," of this manual. Example settings are shown in Chapter 4 but the settings are not standardized and might vary among systems.

Using Environment Variables to Determine the Locale

If **DBNLS** is set, activating NLS, the locale used by the database server is determined by the setting of various other NLS environment variables. This section indicates the order of precedence for selecting the locale in two cases.

1. For a Version 6.0 or later SQL API or a pre-Version 6.0 SQL API that goes through the relay module, the order of precedence for the locale is outlined in the following list, from highest to lowest:

   - **LC_CTYPE** and **LC_COLLATE** set by the user or the application
   - **LANG** set by the user or the application
   - Default locale (C)

   The database server uses the current locale to try to open the specified database. If the database name contains non-ASCII characters and the locale setting is different from that character set, the database name might be rejected because the statement contains illegal characters. Or, the wrong database might open because a character from locale X might be mapped to another character in locale Y. In other words, if database name contains characters that are not recognized by *all* European languages, you must be careful to set the local correctly before opening the database.

   For example, if the current locale is Z and the database name is teßt, the database cannot open unless the Z locale understands the character ß in the database name. If the Z locale does not understand the character ß, the database cannot open.

   This is true even if **DBNLS** is set to use open NLS (**DBNLS**=2). When **DBNLS** is set to 2, all it means is that once the database is opened, its locale becomes the active one. However, the SQL API locale must be able to find the database. If you want to use open NLS to advantage, all the database names should contain characters valid in the default locale C. (The database itself can be a non-C database and the data can contain non-C characters.) When accessing the database, set the locale to C; once the database opens, it switches to the locale specified in the database.

2. For a PC client talking to a UNIX server, the locale used by the server follows the order shown in the following list, from highest to lowest:
   - **LC_CTYPE** and **LC_COLLATE** set in the SQL API
   - **LC_CTYPE** and **LC_COLLATE** set when the server was brought up
   - **LANG** set when the server was brought up
   - Default locale (C)

As stated previously, the database server uses this locale to try to open the specified database. If the locale used is different from the one the database name is in, the server might not be able to open the database. Again, to use Open-NLS effectively, you should create the database with a name in ASCII characters and set **LC_CTYPE** and **LC_COLLATE** to C when trying to access the database. Or make sure that when the server is started, the locale is either set to C or not set (the default is C).

Determining NLS Functionality for a Session and a Database

If you are using DB-Access, you can use the Session option on the main menu to view the NLS capabilities and attributes for the current DB-Access session. This information includes the mode (NLS or non-NLS) and the **LC_COLLATE** and **LC_CTYPE** settings when in NLS mode.

You also can use the Nls option of the DB-Access DATABASE INFO menu to view the NLS collating sequence (**LC_COLLATE** setting) and character classification type (**LC_CTYPE** setting) stored in the active database. Because the **LANG** setting is not stored in the database, it does not appear. However, you probably can infer the NLS language setting from the collating sequence setting shown.

See Chapter 4 and Chapter 6 in the *DB-Access User Manual* for details of these menu options.

If you are using INFORMIX-ESQL / C or INFORMIX-ESQL / COBOL, you must rely on errors returned to the SQLCA data structure to learn whether you have compromised the NLS compatibility guidelines.

See your *INFORMIX-ESQL/C Programmer's Manual* or *INFORMIX-ESQL/COBOL Programmer's Manual* for details of the error-handling process. Specific error messages and their corrective actions are described in the *Informix Error Messages* manual.

NLS Data Types

With NLS activated, data in tables can contain characters from the local language. You can create columns that have the data type NCHAR (INFORMIX-OnLine Dynamic Server or INFORMIX-SE) or NVARCHAR (INFORMIX-OnLine Dynamic Server only) instead of CHAR or VARCHAR, respectively. Columns created with the NLS data types follow the rules of the language and collating sequence of the database.

Use the CREATE TABLE and ALTER TABLE statements to declare columns with the NCHAR or NVARCHAR data type. See Chapter 1 of the *Informix Guide to SQL: Syntax* for information on using these SQL statements. You can also use the DB-Access schema editor to declare NCHAR or NVARCHAR columns, as described in Chapter 5 of the *DB-Access User Manual*.

The specifications for the NCHAR and NVARCHAR data types are basically the same as for their non-NLS counterparts, except that NCHAR and NVARCHAR allow the use of non-English characters. See Chapter 3, "Data Types," for information on available character data types.

Once you activate NLS, the columns in the system catalog tables identified as CHAR are defined as NCHAR. (They remain CHAR if NLS has not been activated.) See Chapter 2, "System Catalog," for information on the system catalog tables.

Important: *If you do not need NLS functionality for your character data, Informix recommends using the CHAR column wherever possible. CHAR data require simple value-based comparisons, sorting, and indexing, and thus these columns are relatively inexpensive in terms of performance.*

NLS Versus Non-NLS Databases and Database Servers

When you create an NLS database, the database server saves the settings for **LC_COLLATE** and **LC_CTYPE** X/Open categories in the system catalog. The settings are used throughout the lifetime of the database for such operations as handling regular expressions, collating character strings, and ensuring proper use of code sets. (Operations on non-NLS databases follow the same rules as for Informix products before Version 6.0, namely, collation order is based on character code.)

A database created after **DBNLS** is set to 1 or 2 is an NLS database. An NLS database can be accessed by NLS or non-NLS programs and client applications through NLS database servers. An NLS database is limited to a single language or code set, as implied by the **LANG** setting that is stored in the database when the database is created.

If you want to use NLS with existing (non-NLS) databases, such as for building indexes using local cultural conventions, you must unload the data and rebuild the databases after activating NLS.

If your current database is not an NLS database, you cannot access an NLS database on another database server. However, you can drop an external NLS database even if your current database is not an NLS database, if the NLS environment variables are set correctly. When the DROP statement is executed, the current database is considered temporarily closed, thus allowing the NLS environment variable settings to take effect.

You can set **DBNLS** to 2 and set **COLLCHAR** to 1 to allow existing INFORMIX-4GL or pre-Version 6.0 INFORMIX-ESQL programs to access NLS databases and obtain proper sorting and collation of character data types from the database server. Setting **DBNLS** to 2 also allows applications with different settings for their locales to access an NLS database.

The Informix Version 6.0 and later database servers are NLS-ready; Informix database servers prior to Version 6.0 are considered non-NLS database servers. An NLS database server can access non-NLS databases as well as NLS databases. However, a non-NLS database server can access only non-NLS databases.

Informix software can support more than one language or locale in a single NLS environment although not in the same database.

Warning: *If you try to use a pre-Version 6.0 INFORMIX-OnLine or INFORMIX-SE with an NLS database, the results could be unpredictable and might even result in corrupted data.*

You might experience performance degradation on some INFORMIX-OnLine Dynamic Server database servers if you have activated NLS and multiple concurrent users have specified different locales. Significant overhead might be incurred during simultaneous collation due to the context switching that the OnLine database server must perform for the various locales. Informix, therefore, recommends that you do not undertake a major sorting operation if you know that another NLS locale is active.

Error Messages Regarding NLS Compatibility

Certain actions on a database or database server might result in an error, depending on the settings for the various NLS environment variables and X/Open categories. The corrective actions for these error messages are described in the *Informix Error Messages* book.

- If you attempt to create an NCHAR (or NVARCHAR) column or declare an NCHAR (or NVARCHAR) variable in a non-NLS database, you receive the following message:

```
-7202 Database was created without NLS functionality.
```

- If you try to access a database that has a different collating sequence than the one defined when the database was created, the database server rejects the connection and returns the following error message:

```
-7203 DBNLS is not set or LC_COLLATE must be 'category-value'
```

  The database server also returns the name of the correct **LC_COLLATE** setting in the SQLERRM field of the SQLCA data structure for Informix programming-language products.

- If you try to access a database that has a different character classification type than the one defined when the database was created, the database server rejects the connection and returns the following error message:

```
-7205 DBNLS is not set or LC_CTYPE must be set to 'category-value'
```

  The database server also returns the name of the correct **LC_CTYPE** setting in the SQLERRM field of the SQLCA data structure for Informix programming-language products.

- If you try to access an NLS database with a non-NLS-ready database server, you see the following message:

```
-7210 This server does not have NLS capability.
```

- If you try to access an external NLS database when the current database is not an NLS database, you receive the following message:

```
-7211 Cannot reference an external NLS database.
```

- Incompatibilities in the database server environment can also result in error messages. For instance, in a network environment, the client and server systems must have a compatible X/Open NLS implementation. Distributed operations can succeed only if all accessed databases are using the same code sets. Otherwise, error -7203 or -7205 is returned.

  Also, if your current database is not an NLS database, you cannot access an NLS database on another database server or within the same database server.

NLS Functionality in Version 6.0 and Later Database Server and SQL API Products

The Informix Version 6.0 and later products are NLS-ready. They allow you to use NLS databases and database servers when working with SQL, provided that you have activated NLS as described earlier.

In addition to the NLS functionality previously described for the INFORMIX-OnLine Dynamic Server and INFORMIX-SE database server products, the following Informix products have also been enhanced to support NLS:

- The DB-Access utility provided with INFORMIX-OnLine Dynamic Server and INFORMIX-SE, which allows all user-specifiable objects such as tables, columns, views, and statements to be named with national character sets and sorted according to local database collation rules
- The SQL APIs, which allow host and indicator variable names as well as names of all user-specifiable objects such as tables, columns, views, cursors, and statements in national character sets

If you also want to receive error and warning messages in a language other than English, you can install a language supplement for the specific locale along with your Informix Version 6.0 and later product. The **DBLANG** environment variable setting specifies the subdirectory under **$INFORMIXDIR/msg** where the message files for each supported native language reside.

In addition, when a language supplement is installed, the DB-Access menu names, menu options, and selection codes, as well as on-line Help, can appear in the local language. You also receive translated SQL command files for practice with NLS.

The installation scripts for the language supplements create the appropriate directories. Installation is described in on-line or printed installation notes that accompany the language supplements.

NLS Functionality in Pre-Version 6.0 Database Server and SQL API Products

Informix products prior to Version 6.0 are *not* NLS-ready. This also might apply to third-party applications.

In Informix software prior to Version 6.0, CHAR and VARCHAR data are sorted or collated according to ASCII conventions, not according to local conventions. This earlier software does not know how to process the NCHAR and NVARCHAR data types that were introduced in Version 6.0.

To use such products with Version 6.0 and later NLS databases and database servers, and obtain reasonable collation results, you should set the **DBNLS** environment variable to 2 and the **COLLCHAR** environment variable to 1. (You can, however, set **DBNLS** to 1 if **LANG** is set to the language of the database.)

With these settings, the database server is NLS-aware; it sorts or collates according to the local conventions, using the NCHAR and NVARCHAR data types. The application is not NLS-aware and has only the CHAR and VARCHAR data types; it sends queries to the NLS database server for collation according to local conventions and receives properly sorted results. (However, comparisons in the application and in the database server might be inconsistent.)

When **DBNLS** is set to 1, all databases created are NLS databases and are locked into a locale. The application must operate in the same locale as the database being accessed. Only Version 6.0 and later INFORMIX-ESQL/C, INFORMIX-ESQL/COBOL, and DB-Access can make use of the difference between CHAR/VARCHAR and NCHAR/NVARCHAR data types.

When **DBNLS** is set to 2, the locale of the application is not checked. When the database server accesses the database, it assumes the locale of the database.

You should be aware of the following behaviors when **DBNLS** is set to 2 and **COLLCHAR** is set to 1:

- The locale of identifiers used in an application must be consistent with the locale of the database server.
- If the application is not aware of the NLS data types, it should not directly access the column type information in the system catalog tables.
- INFORMIX-OnLine Dynamic Server does not allow concurrent access to (or distributed queries on) databases that have different locales. For example, if **database1** is the current database and has a French locale, and, in the same session, you attempt to open or query on **database2** that has a German locale, an error is returned. You can, however, open the German database if the French database is closed.

Chapter

2

System Catalog

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 constitute the system catalog
- Information Schema views that are created from system catalog information

Objects Tracked by the System Catalog Table

The system catalog tables track the following objects:

- Tables and constraints
- Views
- Triggers
- Authorized users and privileges associated with every table you create
- Stored procedures

The system catalog tables are generated automatically when you create a database, and you can query them as you would query any other table in the database. If you are using INFORMIX-OnLine Dynamic Server, the data for a newly created database and the 25 system catalog tables for that database reside in a common area of the disk called a dbspace. If you are using the INFORMIX-SE database server, the 22 system catalog tables for a newly created database reside in the **databasename.dbs** directory. All tables in the system catalog have the prefix **sys** (for example, the systables system catalog table).

Using the System Catalog

The database server accesses the system catalog constantly. Each time an SQL statement is processed, the database server accesses the system catalog to determine system privileges, add or verify table names or column names, and so on. For example, the following CREATE SCHEMA block adds the **customer** table, with its respective indexes and privileges, to the **stores7** database. This block also adds a view, **california**, that restricts the *view* into 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 maryl
CREATE TABLE customer
    (customer_num SERIAL(101), fname CHAR(15), lname CHAR(15), company CHAR(20),
    address1 CHAR(20), address2 CHAR(20), city CHAR(15), state CHAR(2),
    zipcode CHAR(5), phone CHAR(18))
GRANT ALTER, ALL ON customer TO cathl WITH GRANT OPTION AS maryl
GRANT SELECT ON CUSTOMER TO public
GRANT UPDATE (fname, lname, phone) ON customer TO nhowe
CREATE VIEW california AS
    SELECT fname, lname, company, phone FROM customer WHERE state = 'CA'
CREATE UNIQUE INDEX c_num_ix ON customer (customer_num)
CREATE INDEX state_ix ON customer (state);
```

To porcess 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**
- **sysindexes**

The following two new rows of information are added to the **systables** system catalog table after the CREATE SCHEMA block shown on page 2-4 is run.

```
tabname                 customer
owner                   maryl
partnum                 16778361
tabid                   101
rowsize                 134
ncols                   10
nindexes                2
nrows                   0
created                 04/26/1994
version                 1
tabtype                 T
locklevel               P
npused                  0
fextsize                16
nextsize                16
flags                   0
site
dbname

tabname                 california
owner                   maryl
partnum                 0
tabid                   102
rowsize                 134
ncols                   4
nindexes                0
nrows                   0
created                 04/26/1994
version                 0
tabtype                 V
locklevel               B
npused                  0
fextsize                0
nextsize                0
flags                   0
site
dbname
```

Each table recorded in the **systables** system catalog table is assigned a **tabid**, a system-assigned sequential ID number that uniquely identifies each table in the database. The system catalog tables receive **tabid** numbers 1 through 24, and the user-created tables receive **tabid** numbers beginning with 100.

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

```
colname      tabid  colno coltype  collength colmin colmax

customer_num 101    1     262      4
fname        101    2     0        15
lname        101    3     0        15
company      101    4     0        20
address1     101    5     0        20
address2     101    6     0        20
city         101    7     0        15
state        101    8     0        2
zipcode      101    9     0        5
phone        101    10    0        18

fname        102    1     0        15
lname        102    2     0        15
company      102    3     0        20
phone        102    4     0        18
```

In the **syscolumns** system catalog table, each column within a table is assigned a sequential column number, **colno**, that uniquely identifies the column within its table. In the **colno** column, the **frame** column of the **customer** table is assigned the value 1 and the frame column of the view **california** is assigned the value 2. Note that the **colmin** and **colmax** columns contain no entries. These two columns contain values only 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.

Tip: *For all system catalog tables, all columns identified as a CHAR data type are NCHAR data type if NLS functionality is activated. The data type remains as CHAR if NLS functionality is not activated.*

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:

```
tabid  seq  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 used to create 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 between the same columns used in a self-join.

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

```
grantor   grantee   tabid   tabauth

maryl     public    101     su-idx--
maryl     cathl     101     SU-IDXAR
maryl     nhowe     101     --*-----
          maryl     102     SU-ID---
```

The **tabauth** column of this table specifies the table-level privileges granted to users on the **customer** and **california** tables. This column uses an 8-byte pattern—**s** (select), **u** (update), ***** (column-level privilege), **i** (insert), **d** (delete), **x** (index), **a** (alter), **r** (references)—to identify the type of privilege granted. 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 is granted 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 granted on specific columns in the **customer** table, as shown in the following example:

```
grantor  grantee   tabid   colno  colauth

maryl    nhowe     101     2      -u-
maryl    nhowe     101     3      -u-
maryl    nhowe     101     10     -u-
```

The **colauth** column of this table specifies the column-level privileges granted on the **customer** table. This column uses a 3-byte pattern—**s** (select), **u** (update), **r** (references)—to identify the type of privilege granted. 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 100).

The CREATE SCHEMA block adds two rows to the **sysindexes** system catalog table. These rows correspond to the indexes created on the **customer** table, as shown in the following example:

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

In this table, the **idxtype** column identifies whether the index created is unique or a duplicate. For example, the index **c_num_ix** 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 the 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.*

Updating System Catalog Data

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

By using the UPDATE STATISTICS statement to update the system catalog, you can insure that the information provided to the optimizer is current. When you delete or modify a table, the database server does not automatically update the related statistical data in the system catalog. For example, if you delete rows in a table using the DELETE statement, the **nrows** column in the **systables** system catalog table, which holds the number of rows for that table, does not get updated. The UPDATE STATISTICS statement causes the database server to recalculate data in the **systables**, **sysdistrib**, **syscolumns**, and **sysindexes** system catalog tables. After you run UPDATE STATISTICS, the **systables** system catalog table holds the correct value in its **nrows** column. If you use the MEDIUM or HIGH mode with UPDATE STATISTICS, the **sysdistrib** system catalog table holds the updated data-distribution data after you run UPDATE STATISTICS.

Whenever you modify a table extensively, use the UPDATE STATISTICS statement to update data in the system catalog. For more information on the UPDATE STATISTICS statement, see Chapter 1 of the *Informix Guide to SQL: Syntax*.

Structure of the System Catalog

The following system catalog tables describe the structure of the Informix database:

- **sysblobs**
- **syschecks**
- **syscolauth**
- **syscoldepend**
- **syscolumns**
- **sysconstraints**
- **sysdefaults**
- **sysdepend**
- **sysdistrib**
- **sysfragments**
- **sysindexes**
- **sysopclstr**
- **sysprocauth**
- **sysprocbody**
- **sysprocedures**
- **sysprocplan**
- **sysreferences**
- **syssynonyms**
- **syssyntable**
- **systabauth**
- **systables**
- **systrigbody**
- **systriggers**
- **sysusers**
- **sysviews**

You should not confuse the system catalog tables of a database with the tables in the **sysmaster** database of INFORMIX-OnLine Dynamic Server database servers. The **sysmaster** tables also start with **sys**, but they contain information about an entire OnLine database server (which might manage many databases). The information in the **sysmaster** tables is primarily useful for OnLine database administrators. For more information about the **sysmaster** tables, see the *INFORMIX-OnLine Dynamic Server Administrator's Guide*.

SYSBLOBS

The **sysblobs** system catalog table specifies the storage location of a blob column. It contains one row for each blob column in a table. Available in INFORMIX-OnLine Dynamic Server only, the **sysblobs** system catalog table has the following columns:

Column Name	Type	Explanation
spacename	CHAR(18)	Blobspace, dbspace, or family name
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.

SYSCHECKS

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

Column Name	Type	Explanation
constrid	INTEGER	Constraint identifier
type	CHAR(1)	Form in which the check constraint is stored: B = Binary encoded T = ASCII 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 machine-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.

SYSCOLAUTH

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

Column Name	Type	Explanation
grantor	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 = Select u = Update r = References

If the **colauth** privilege code is uppercase (for example, **S** for select), a user who has this privilege can also grant it to others. If the **colauth** privilege code is lowercase (for example, **s** for select), the user who is granted 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 each check constraint. Because a check constraint can involve more than one column in a table, it can contain multiple rows for each check constraint. The **syscoldepend** system catalog table has the following columns:

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

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

SYSCOLUMNS

The **syscolumns** system catalog table describes each column in the database. One row exists for each column defined in a table or view. If you are using the OnLine database server, the **syscolumns** system catalog table has 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)
coltype	SMALLINT	Code for column data type: 0 = CHAR 1 = SMALLINT 2 = INTEGER 3 = FLOAT 4 = SMALLFLOAT 5 = DECIMAL 6 = SERIAL 7 = DATE 8 = MONEY 10 = DATETIME 11 = BYTE 12 = TEXT 13 = VARCHAR 14 = INTERVAL 15 = NCHAR 16 = NVARCHAR
collength	SMALLINT	Column length (in bytes)
colmin	INTEGER	Second minimum value
colmax	INTEGER	Second maximum value

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 value that the **collength** column holds depends on the data type of the column. If the data type of the column is BYTE or TEXT, **collength** holds the length of the descriptor. A **collength** value for a MONEY or DECIMAL column is determined using the following formula:

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

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

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

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

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

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

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 following values:

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

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

For information about using the HEX function to display the **collength** and **coltype** values, see the Column Expression discussion in the "Expression" segment section in Chapter 1 of the *Informix Guide to SQL: Syntax*.

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 value 2 is the **colmin** value and 4 is the **colmax** value. Storing the second-smallest and second-largest data values allows the database server to 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 UPDATE STATISTICS. 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, treated as an integer.

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

If you are using the INFORMIX-SE database server, the **syscolumns** system catalog table has 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 (ordinally from left to right within each table)
coltype	SMALLINT	Code for column data type: 0 = CHAR 1 = SMALLINT 2 = INTEGER 3 = FLOAT 4 = SMALLFLOAT 5 = DECIMAL 6 = SERIAL 7 = DATE 8 = MONEY 10 = DATETIME 14 = INTERVAL 15 = NCHAR
collength	SMALLINT	Column length (in bytes)

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

SYSCONSTRAINTS

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

Column Name	Type	Explanation
constrid	SERIAL	System-assigned sequential identifier
constrname	CHAR(18)	Constraint name
owner	CHAR(8)	User name of owner
tabid	INTEGER	Table identifier
constrtype	CHAR(1)	Constraint type:
		C = Check constraint
		P = Primary key
		R = Referential
		U = Unique
idxname	CHAR(18)	Index name

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

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

SYSDEFAULTS

The **sysdefaults** system catalog table lists the user-defined defaults placed on each column in the database. One row exists for each user-defined default value. If a default is not explicitly specified in the CREATE TABLE statement, no entry exists in this table. The **sysdefaults** system catalog table has the following columns:

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

If a literal is specified for the default value, it is stored in the **default** column as ASCII 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 both the **tabid** and **colno** columns allows only unique values.

SYSDEPEND

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

Column Name	Type	Explanation
btabid	INTEGER	Table identifier of base table or view
btype	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 use by the database server. Data distributions provide detailed table column information to the optimizer to improve the choice of execution paths for optimization 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 has the following columns:

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

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

SYSFRAGMENTS

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

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

Column Name	Type	Explanation
fragtype	CHAR(1)	Fragment type:
		I = 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	CHAR(1)	Reserved for future use. Shows L for local.
servername	CHAR(18)	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 **T** is used for attached indexes (where index fragmentation is the same as the table fragmentation).

SYSINDEXES

The **sysindexes** system catalog table describes the indexes in the database. It contains one row for each index defined in the database. The **sysindexes** system catalog table for the OnLine database server has 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
clustered	CHAR(1)	Clustered or nonclustered index (C = Clustered)
part1	SMALLINT	Column number (**colno**) of a single index or the 1st component of a composite index

(1 of 2)

Column Name	Type	Explanation
part2	SMALLINT	2nd component of a composite index
part3	SMALLINT	3rd component of a composite index
part4	SMALLINT	4th component of a composite index
part5	SMALLINT	5th component of a composite index
part6	SMALLINT	6th component of a composite index
part7	SMALLINT	7th component of a composite index
part8	SMALLINT	8th component of a composite index
part9	SMALLINT	9th component of a composite index
part10	SMALLINT	10th component of a composite index
part11	SMALLINT	11th component of a composite index
part12	SMALLINT	12th component of a composite index
part13	SMALLINT	13th component of a composite index
part14	SMALLINT	14th component of a composite index
part15	SMALLINT	15th component of a composite index
part16	SMALLINT	16th component of a composite index
levels	SMALLINT	Number of B+ tree levels
leaves	INTEGER	Number of leaves
nunique	INTEGER	Number of unique keys
clust	INTEGER	Degree of clustering: smaller numbers correspond to greater clustering

(2 of 2)

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

Each **part***nth* column component of a composite index (the **part1** through **part16** columns in this table) holds the column number (**colno**) of each part of the 16 possible parts of a composite index. If the component is ordered in descending order, the **colno** is entered as a negative value.

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

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

If you are using the INFORMIX-SE database server, the **sysindexes** system catalog table has the following columns:

Column Name	Type	Explanation
idxname	CHAR(18)	Index name
owner	CHAR(8)	Owner of index (user **informix** for system tables and user name for database tables)
tabid	INTEGER	Table identifier
idxtype	CHAR(1)	Index type: U = Unique D = Duplicates
clustered	CHAR(1)	Clustered or nonclustered index (C = Clustered)
part1	SMALLINT	Column number (**colno**) of a single index or the 1st component of a composite index
part2	SMALLINT	2nd component of a composite index
part3	SMALLINT	3rd component of a composite index
part4	SMALLINT	4th component of a composite index
part5	SMALLINT	5th component of a composite index
part6	SMALLINT	6th component of a composite index
part7	SMALLINT	7th component of a composite index
part8	SMALLINT	8th component of a composite index

Each **part***nth* column component of a composite index (the **part1** through **part8** columns in this table) holds the column number (**colno**) of each part of the eight possible parts of an index.

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

SYSOPCLSTR

The **sysopclstr** system catalog table defines each optical cluster in the database. Available for INFORMIX-OnLine Dynamic Server only, it contains one row for each optical cluster. The **sysopclstr** system catalog table has 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
blobcol2	SMALLINT	Blob column number 2
blobcol3	SMALLINT	Blob column number 3
blobcol4	SMALLINT	Blob column number 4
blobcol5	SMALLINT	Blob column number 5
blobco16	SMALLINT	Blob column number 6
blobcol7	SMALLINT	Blob column number 7
blobcol8	SMALLINT	Blob column number 8
blobcol9	SMALLINT	Blob column number 9
blobcol10	SMALLINT	Blob column number 10
blobcol11	SMALLINT	Blob column number 11

(1 of 2)

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

(2 of 2)

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

SYSPROCAUTH

The **sysprocauth** table describes the privileges granted on a procedure. It contains one row for each set of privileges granted. The **sysprocauth** system catalog table has 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 stored procedure in the database. Because the **sysprocbody** system catalog table stores the text of the procedure, each procedure can have multiple rows. The **sysprocbody** system catalog table has 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 stored, the **data** column contains the actual data, which can be one of the following types of data: 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 stored procedure in the database. It contains one row for each procedure. The **sysprocedures** system catalog table has the following columns:

Column Name	Type	Explanation
procname	CHAR(18)	Procedure name
owner	CHAR(8)	Owner name
procid	SERIAL	Procedure 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 procedure instruction code
numargs	INTEGER	Number of procedure arguments

A composite index for the **procname** and **owner** columns 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.

SYSPROCPLAN

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

Column Name	Type	Explanation
procid	INTEGER	Procedure identifier
planid	INTEGER	Plan identifier
datakey	CHAR(1)	Identifier procedure 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 placed on columns in the database. It contains a row for each referential constraint in the database. The **sysreferences** table has the following columns:

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

The **constrid** column is indexed and allows only unique values. The **primary** column is indexed and allows duplicate values. If you are using the **INFORMIX**-SE database server, the contents of the **delrule** column are reserved.

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 has 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 **synonym** columns allows only unique values. The **tabid** column is indexed and allows duplicate values.

Important: *Informix products Version 4.0 or later 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 has 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
tabname	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 **tabname** columns are used.

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

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

If you are using the INFORMIX-SE database server, only the **tabid** and **btabid** columns are used.

SYSTABAUTH

The systabauth system catalog table describes each set of privileges granted in a table. It contains one row for each set of table privileges granted in the database. The systabauth system catalog table has 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: s = Select u = Update * = Column-level authority i = Insert d = Delete x = Index a = Alter 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 defined in the database. This includes all database tables and the system catalog tables. If you are using INFORMIX-OnLine Dynamic Server, the **systables** system catalog table has the following columns:

Column Name	Type	Explanation
tabname	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: 1-24, user tables: 100-nnn)
rowsize	SMALLINT	Row size
ncols	SMALLINT	Number of columns
nindexes	SMALLINT	Number of indexes
nrows	INTEGER	Number of rows
created	DATE	Date created
version	INTEGER	Number that changes when table is altered
tabtype	CHAR(1)	Table type: T = Table V = View P = Private synonym P = Synonym (in AN ANSI-COMPLIANT DATABASE) S = Synonym

(1 of 2)

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
fextsize	INTEGER	Size of initial extent (in kilobytes)
nextsize	INTEGER	Size of all subsequent extents (in kilobytes)
flags	SMALLINT	Reserved for future use
site	CHAR(18)	Reserved for future use in OnLine (used to store NLS data in NLS databases)
dbname	CHAR(18)	Reserved for future use

(2 of 2)

The **tabid** column is indexed and must contain unique values. A composite index for both the **tabname** 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.

If you are using the INFORMIX-SE database server, the **systables** system catalog table has the following columns:

Column Name	Type	Explanation
tabname	CHAR(18)	Name of table
owner	CHAR(8)	Owner of table (user **informix** for system tables and user name for database tables)
dirpath	CHAR(64)	Directory path for the table file
tabid	SERIAL	System assigned sequential ID number (system tables: 1-21, user tables: 100-nnn)
rowsize	SMALLINT	Row size
ncols	SMALLINT	Number of columns
nindexes	SMALLINT	Number of indexes
nrows	INTEGER	Number of rows
created	DATE	Date created
version	INTEGER	Number that changes when table is altered
tabtype	CHAR(1)	Table type: T = Table V = View S = Synonym L = Log P = Synonym (in AN ANSI-COMPLIANT DATABASE)
audpath	CHAR(64)	Audit filename (full pathname)

The **dirpath** column contains the directory path for the table or log file. The **tabid** column is indexed and must contain unique values. A composite index for the **tabname** and **owner** columns allows only unique values.

When NLS functionality has been activated, two hidden rows are added to the **systables** system catalog table. NLSCOLL represents the LC_COLLATE setting and has a **tabid** of 90; NLSCTYPE represents the LC_CTYPE setting and has a **tabid** of 91. Enter the following SELECT statement to view these two hidden rows:

```
SELECT tabname, tabid FROM systables
```

SYSTRIGBODY

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

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

The **systrigbody** system catalog table has the following columns:

Column Name	Type	Explanation
trigid	INT	Trigger identifier
datakey	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	INT	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 has the following columns:

Column Name	Type	Explanation
trigid	SERIAL	Trigger identifier
trigname	CHAR(18)	Trigger name
owner	CHAR(8)	Owner of trigger
tabid	INT	ID of triggering table
event	CHAR	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	CHAR	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 granted in the database. It contains one row for each user who is granted privileges in the database. The **sysusers** system catalog table has the following columns:

Column Name	Type	Explanation
username	CHAR(8)	Name of the database user
usertype	CHAR(1)	Specifies database-level privileges: D = DBA (all privileges) R = Resource (create permanent tables and indexes) C = Connect (work within existing tables)
priority	SMALLINT	Reserved for future use
password	CHAR(8)	Reserved for future use

The **username** column is indexed and allows only unique values.

SYSVIEWS

The **sysviews** system catalog table describes each view defined in the database. Because the **sysviews** system catalog table stores the actual SELECT statement used to create the view, it can contain multiple rows for each view into the database. The **sysviews** system catalog table has the following columns:

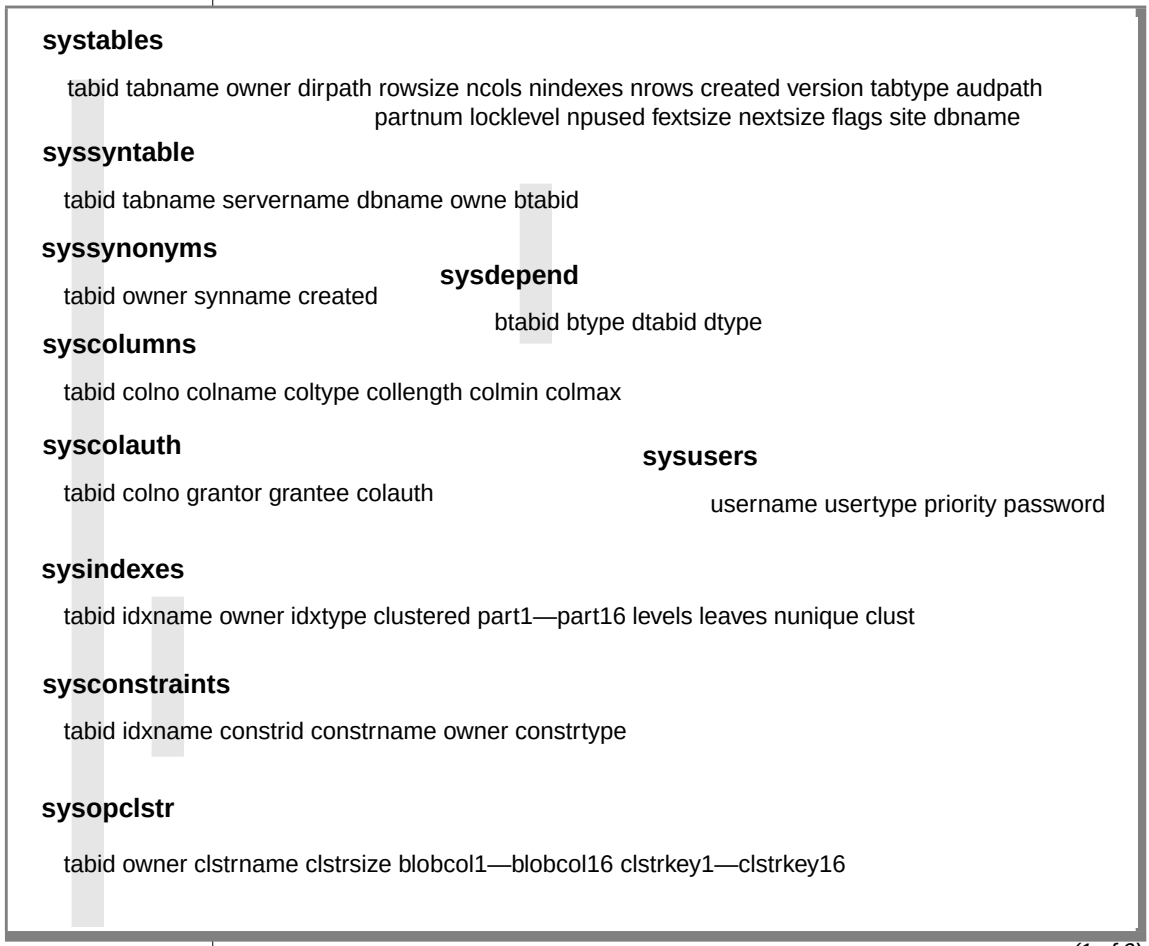
Column Name	Type	Explanation
tabid	INTEGER	Table identifier
seqno	SMALLINT	Line number of the SELECT statement
viewtext	CHAR(64)	Actual SELECT statement used to create the view

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

System Catalog Map

The following System Catalog Map displays the column names of the tables in an INFORMIX-OnLine Dynamic Server system catalog. The lines connecting a column in one table to a column in another table indicate columns that contain the same information.

Figure 2-1
System catalog map



(1 of 2)

systriggers

tabid trigid trigname owner event old new mode

systrigbody

trigid datakey seqno data

sysdistrib

tabid colno seqno constructed mode resolution confidence encdat

systabauth

tabid grantor grantee tabauth

sysviews

tabid seqno viewtext

sysblobs

tabid spacename type colno

sysdefaults

tabid colno type default

syscoldepend

tabid colno constrid

syschecks

constrid type seqno checktext

sysreferences

constrid primary ptabid updrule delrule matchtype pendant

sysprocplan

procid planid datakey seqno created datasize data

sysprocbody

procid datakey seqno data

sysprocauth

procid procauth grantee grantor

sysprocedures

procid procname owner mode retsize symsize datasize codesize numargs

(2 of 2)

Information Schema

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

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

Important: *Because the X/Open CAE standards Information Schema views differ from ANSI-compliant Information Schema views, it is not recommended that you 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, accessing, and the structure of the Information Schema Views.

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 may 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 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 following columns:

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

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

COLUMNS

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

Column Name	Data Type	Explanation
table_schema	VARCHAR(128)	Owner of table
table_name	VARCHAR(128)	Name of table or view
column_name	VARCHAR(128)	Name of the column of the table or view
ordinal_position	INTEGER	Ordinal position of the column. The ordinal_position of a column in a table is a sequential number starting at 1 for the first column.This column is an Informix extension to XPG4.
data_type	VARCHAR(254)	Contains the data type of the column, such as CHARACTER or DECIMAL.
char_max_length	INTEGER	Maximum length for character data types, null otherwise

(1 of 2)

Column Name	Data Type	Explanation
numeric_precision	INTEGER	Total number of digits allowed for exact numeric data types (DECIMAL, INTEGER, MONEY, and SMALLINT), and the number of digits of mantissa precision for approximate data types (FLOAT and SMALLFLOAT), and null for all other data types. The value is machine dependent for FLOAT and SMALLFLOAT.
numeric_prec_radix	INTEGER	Uses one of the following values: 2 approximate data types (FLOAT and SMALLFLOAT) 10 exact numeric data types (DECIMAL, INTEGER, MONEY, and SMALLINT) null for all other data types
numeric_scale	INTEGER	Number of significant digits to the right of the decimal point for DECIMAL and MONEY data types: 0 for INTEGER and SMALLINT data types null for all other data types
datetime_precision	INTEGER	Number of digits in the fractional part of the seconds for DATE and DATETIME columns, null otherwise. This column is an Informix extension to XPG4.
is_nullable	VARCHAR(3)	Indicates whether a column allows nulls; either YES or NO
remarks	VARCHAR(254)	Reserved

(2 of 2)

SQL_LANGUAGES

The **sql_languages** Information Schema view contains a row for each conformance to standards that the current database server supports. If the database server is INFORMIX-SE, the table shows no rows. The **sql_languages** Information Schema view contains the following columns:

Column Name	Data Type	Explanation
source	VARCHAR(254)	Organization that defines this SQL version
source_year	VARCHAR(254)	Year the source document was approved
conformance	VARCHAR(254)	Which conformance is supported
integrity	VARCHAR(254)	Indicates whether this is an integrity enhancement feature; either YES or NO
implementation	VARCHAR(254)	Identifies the vendor's SQL product
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 following columns:

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

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

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

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

Chapter

3

Data Types

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 covers the following topics:

- Data types supported by Informix products
- Data type conversions
- DATE, DATETIME, and INTERVAL values in arithmetic and relational expressions

Database Data Types

You assign data types with the CREATE TABLE statement and change them with the ALTER TABLE statement. When you change an existing data type, 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 detailed discussion of data types, see Chapter 9 of the *Informix Guide to SQL: Tutorial*.

Effects of NLS

The NCHAR data type is recognized only when NLS is enabled by setting the DBNLS environment variable. In addition, the LANG and the LC sublocale environment variable settings specify a language environment that might influence the default formats for date, monetary, and numeric values as well as collation order.

Summary of Data Types

Informix products recognize the data types listed in Figure 3-1. The remainder of this chapter describes each of these data types.

Figure 3-1
Data types recognized by informix products

Data Type	Explanation
BYTE	Stores any kind of binary data
CHAR	Stores any string of letters, numbers, and symbols
CHARACTER	Is a synonym for CHAR
CHARACTER VARYING	Stores character strings of varying length (ANSI-compliant)
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
DOUBLE PRECISION	Behaves the same way as FLOAT
FLOAT	Stores double-precision floating numbers corresponding to the double data type in C
INT	Is a synonym for INTEGER
INTEGER	Stores whole numbers from –2,147,483,647 to +2,147,483,647
INTERVAL	Stores span of time
MONEY	Stores currency amount
NCHAR	Stores any native string of letters, numbers, and symbols
NUMERIC	Is a synonym for DECIMAL
NVARCHAR	Stores native character strings of varying length
REAL	Is a synonym for SMALLFLOAT

(1 of 2)

Data Type	Explanation
SERIAL	Stores sequential integers
SMALLFLOAT	Stores single-precision floating 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	Stores character strings of varying length

(2 of 2)

BYTE

The BYTE data type stores any kind of binary data in an undifferentiated byte stream. Binary data typically consists of saved spreadsheets, program load modules, digitized voice patterns, and so on. The INFORMIX-SE database server does not support this data type.

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 host variables (INFORMIX-ESQL / C)
- By declaring a FILE (INFORMIX-ESQL / COBOL)

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.

Tip: *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 string of letters, numbers, and symbols. A character column has a maximum length *n*, where $1 \leq n \leq 32{,}767$. (If you are using the INFORMIX-SE database server, the maximum length is 32,511.) 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 value is shorter than *n*, the string is extended with spaces to make up the *n* bytes. If the value is longer than *n*, the string is truncated.

If you plan to perform calculations on numbers stored in a column, you should assign a number data type to that column. Although you can store numbers in CHAR columns, you might not be able to use them in some arithmetic operations. For example, if you 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 these numbers in CHAR columns.

CHAR values are compared to other CHAR values by taking the shorter value and padding it on the right with spaces until the values have equal length. Then, the two values are compared for the full length.

CHAR data types require 1 byte per character, or *n* bytes.

When NLS has been enabled, CHAR data columns are treated as NCHAR data columns. For additional information, see the description of the NCHAR data type on page 3-20.

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 a character string 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 INFORMIX-SE database server does not support this data type. The CHARACTER VARYING data type complies with ANSI standards; the Informix VARCHAR data type displays the same functionality.

You must specify the maximum size (*m*) of the CHARACTER VARYING column. The size of this parameter can range from 1 to 255 bytes. If you are placing an index on a CHARACTER VARYING 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 CHARACTER VARYING 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 CHARACTER VARYING economizes on space used in a table, it has no effect on the size of an index. In an index based on a CHARACTER VARYING column, each index key has length *m*, the maximum size of the column.

When you store a CHARACTER VARYING value in the database, only its defined characters are stored. The database server does not strip a CHARACTER VARYING object of any user-entered trailing blanks, nor does the database server pad the CHARACTER VARYING 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.

CHARACTER VARYING values are compared to other CHARACTER VARYING 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.

DATE

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

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 (0001-9999). 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, as in 91 or 92, Informix products assume that the year is in the twentieth century and assign the numbers 1 and 9 (19) as the first two digits of the year.

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.

DATE data types require 4 bytes per item.

You can change the default date format by changing the DBDATE environment variable. In addition, NLS lets you define the date and time format with LC_TIME. See Chapter 4, "Environment Variables," for more information.

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 3-2.

Figure 3-2
DATETIME field qualifiers

Qualifier Field	Valid Entries
YEAR	A year numbered from 1 to 9,999 (A.D.)
MONTH	A month numbered from 1 to 12
DAY	A day numbered from 1 to 31, as appropriate to the month
HOUR	An hour numbered from 0 (midnight) to 23
MINUTE	A minute numbered from 0 to 59
SECOND	A second numbered from 0 to 59
FRACTION	A decimal fraction of a second with up to 5 digits of precision. The default precision is 3 digits (a thousandth of a second). Other precisions are indicated explicitly by writing FRACTION(*n*), where *n* is the desired number of digits from 1 to 5.

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

If you 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 3-2.

Values for the field qualifiers are written as integers and separated by delimiters. Figure 3-3 lists the delimiters that are used with DATETIME values.

Figure 3-3
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 3-4 shows a DATETIME YEAR TO FRACTION(3) value with delimiters

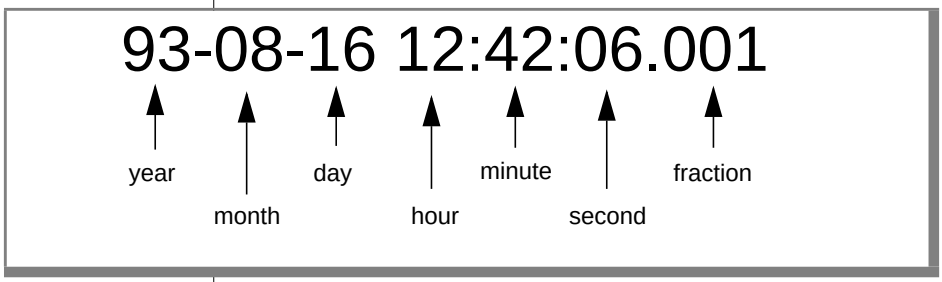


Figure 3-4
Example DATETIME value with delimiters

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 one 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. 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:

```
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 using DATETIME data in arithmetic and relational expressions, see "Range of Operations Using DATE, DATETIME, and INTERVAL" on page 3-30. For information on using DATETIME as a constant expression, see Chapter 1 of the *Informix Guide to SQL: Syntax*.

DEC

The DEC data type is a synonym for DECIMAL.

DECIMAL

The DECIMAL data type can take two forms: DECIMAL(*p*) floating point, and DECIMAL(*p,s*) fixed point.

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 a MODE ANSI database and specify DECIMAL (*p*), the value defaults to DECIMAL (*p*, 0). 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 $10^{p-s} - 10^{-s}$. A DECIMAL data type column typically stores numbers with fractional parts that must be stored and displayed exactly (for example, rates or percentages).

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 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 3-5. With the additional byte required for the exponent and sign, this data type requires a total of 5 bytes of storage.

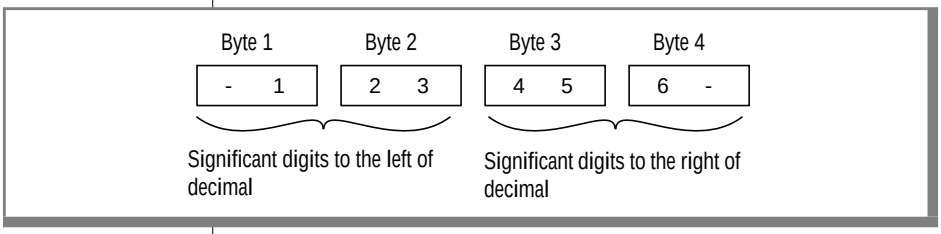


Figure 3-5
Schematic illustrating the storage of digits in a decimal value. (The exponent byte is not shown.)

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).

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

DOUBLE PRECISION

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

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 represent 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 3-6 on page 3-17, and *n* optionally specifies the precision of the largest field (and smallest field if it is a FRACTION).

Figure 3-6
Interval Classes

	Qualifier Field	Valid Entry
YEAR-MONTH INTERVAL Class	YEAR	A number of years
	MONTH	A number of months
DAY-TIME INTERVAL Class	DAY	A number of days
	HOUR	A number of hours
	MINUTE	A number of minutes
	SECOND	A number of seconds
	FRACTION	A decimal fraction of a second, with up to 5 digits of precision. The default precision is 3 digits (thousandth of a second). Other precisions are indicated explicitly by writing FRACTION(*n*), where *n* is the desired number of digits from 1 to 5.

As with a DATETIME column, you can define an INTERVAL column to include a subset of the fields 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 3-6 on page 3-17.

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 3-7.

Figure 3-7
INTERVAL delimiters

Delimiter	Placement in DATETIME Expression
hyphen	Between the YEAR and MONTH
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. Note that 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:

```
total number of digits for all fields/2 + 1
```

For example, a YEAR TO MONTH qualifier requires a total of six digits (four for year and two for month). This data value requires 4, or (6 / 2) + 1, bytes of storage.

For information on using INTERVAL data in arithmetic and relational operations, see "Range of Operations Using DATE, DATETIME, and INTERVAL" on page 3-30. For information on using INTERVAL as a constant expression, see the description of the INTERVAL Field Qualifier segment in Chapter 1 of the *Informix Guide to SQL: Syntax*.

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 data type MONEY(*p*) is defined as DECIMAL(*p*,2). If the precision and scale parameters are not specified, MONEY is interpreted as DECIMAL(16,2).

Values in MONEY columns are displayed with a currency symbol (by default, a dollar sign) and a decimal point. 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 9, or (16 + 3) / 2, bytes of storage.

You can change the display format for money values by changing the DBMONEY environment variable. In addition, NLS lets you define money value formats with the LC_MONETARY environment variable. See Chapter 4, "Environment Variables," for more information.

NCHAR(*n*)

The NCHAR data type is recognized only when NLS is enabled by setting the DBNLS environment variable. The NCHAR data type stores any string of letters, numbers, and symbols. A native-character column has a logical size of *n* characters, depending on the code set. For the supported European languages / code sets, the NCHAR data type requires 1 byte per character. The total size of an NCHAR column cannot exceed 32,767 bytes for the INFORMIX-OnLine Dynamic Server database server or 32,511 bytes for the INFORMIX-SE database server. If you do not specify *n*, NCHAR(1) is assumed.

Native-character columns typically store names, addresses, phone numbers, and so on. Because the length of this column is fixed, when a native-character value is retrieved or stored, exactly *n* bytes of data are transferred. If the value is shorter than *n*, the string is extended with spaces to make up the *n* bytes. If the value is longer than *n*, the string is truncated.

An NCHAR value can include tabs, spaces, and other nonprintable characters. For additional information about nonprintable characters, see "Nonprintable Characters with CHAR" on page 3-7.

If you plan to perform calculations on numbers stored in a column, you should assign a number data type to that column. Although you can store numbers in NCHAR 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 native character column is too small to hold the value. In this case, the insert fails. Numbers that have leading zeros (such as some zip codes) have the zeros stripped if they are stored as the number types INTEGER or SMALLINT. Instead, store these numbers in NCHAR columns.

Nonprintable Characters with NCHAR

An NCHAR 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.

NUMERIC(*p*,*s*)

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

NVARCHAR(*m*,*r*)

The NVARCHAR data type is recognized only when NLS is enabled by setting the DBNLS environment variable. The NVARCHAR data type stores a native-character string of varying length, where *m* is the maximum size of the column and *r* is the minimum amount of space reserved for that column, depending on the code set. The INFORMIX-SE database server does not support this data type.

An NVARCHAR value can include tabs, spaces, and other nonprintable characters. For additional information about nonprintable characters, see "Nonprintable Characters with NVARCHAR" on page 3-22.

You must specify the maximum size (*m*) of the NVARCHAR column. The size of this parameter cannot exceed 255 bytes, although the logical number of native characters can be smaller. If you are placing an index on an NVARCHAR column, the maximum size is 254 bytes. You can store shorter, but not longer, native-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 NVARCHAR column. If you do not specify a minimum space value, it defaults to 0. 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 using NVARCHAR economizes on space used in a table, it has no effect on the size of an index. In an index based on an NVARCHAR column, each index key has length *m*, the maximum size of the column.

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

Nonprintable Characters with NVARCHAR

An NVARCHAR 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, the characters can be stored or retrieved. When you select nonprintable characters, fetch them into host variables and display them using your own display mechanism.

NVARCHAR treats C null (binary 0) and string terminators as termination characters for nonprintable characters.

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

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 Chapter 1 of the *Informix Guide to SQL: Syntax*.) You can define only one SERIAL column in a table.

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.

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

Once a nonzero number is assigned, it cannot be changed. You can, however, insert a value 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:

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

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

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

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 data type stores any kind of text data. The INFORMIX-SE database server does not support this data type.

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. A data object of type TEXT can contain a combination of printable ASCII characters and the following control characters:

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

You can store, retrieve, update, or delete the contents of a TEXT column. However, you cannot use TEXT data items in arithmetic or string operations, or assign literals to TEXT items with the SET clause of the UPDATE statement. You also cannot use TEXT items in the following ways:

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

You can use TEXT objects in Boolean expressions only if you are testing for null values.

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

- With the **dbload** or **onload** utilities
- With the LOAD statement (DB-Access)
- From TEXT host variables (INFORMIX-ESQL / C)
- By declaring a FILE (INFORMIX-ESQL / COBOL)

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

A TEXT value can include tabs, spaces, and other nonprintable characters. However, you must use an Informix SQL API 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 TEXT character that you can store and retrieve in DB-Access is a tab. (You can insert only TEXT data with the LOAD statement through DB-Access.) If you try to display other nonprintable characters in DB-Access, your screen returns inconsistent results.

VARCHAR(*m*,*r*)

The VARCHAR data type stores a character string 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 INFORMIX-SE database server does not support this data type. 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 3-8.

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.

Nonprintable Characters with VARCHAR

A VARCHAR value can include tabs, spaces, and other nonprintable characters. However, you must use an SQL API to insert the nonprintable characters into host variables and to insert the host variables into your program. After passing nonprintable characters to the database server, the characters can be stored or retrieved. When you select nonprintable characters, fetch them into host variables and display them using your own display mechanism.

VARCHAR treats C null (binary 0) and string terminators as termination characters for nonprintable characters.

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

Data Type Conversions

You might want to change the data type of a column when you need to store larger values than the current data type can accommodate. For example, if you create a SMALLINT column and later find that you need to store integers larger than 32,768, you must change the data type of that column to store the larger value. You can use the ALTER TABLE statement to change the data type of that column.

If you change data types, the new data type must be able to store all the old values. For example, if you try to convert a column from the INTEGER data type to the SMALLINT data type and the following values exist in the INTEGER column, the database server does not change the data type because SMALLINT Columns cannot accommodate numbers greater than 32,768:

```
100    400    700    50000    700
```

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

Converting from Number to Number

When you convert columns from one number data type to another, you occasionally find rounding errors. Figure 3-8 on page 3-29 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 3-8
Numeric data type conversion chart

From:	To:				
	SMALLINT	**INTEGER**	**SMALLFLOAT**	**FLOAT**	**DECIMAL**
SMALLINT	OK	OK	OK	OK	O
INTEGER	X	OK	X	OK	O
SMALLFLOAT	X	X	OK	OK	O
FLOAT	X	X	F	OK	O
DECIMAL	X	X	F	F	O

Legend:
OK = No error
O = An error can occur depending on precision of the decimal
X = An error can occur depending on data
F = No error, but less significant digits might be lost

For example, if you convert a FLOAT column 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 number column and vice versa. However, if the CHAR or NCHAR column contains any characters that are not valid in a number column (for example, the letter *l* instead of the number *1*), your database server cannot complete the ALTER TABLE statement and leaves the column values as characters. You receive an error and the statement is rolled back (whether you are in a transaction or not).

In INFORMIX-SE, the original table is unchanged, but the system catalog tables might be in an inconsistent state.

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 DATE, 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.

Range of Operations Using DATE, DATETIME, and INTERVAL

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 point in time.

Figure 3-9 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 3-9
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
INTERVAL	+	CURRENT	DATETIME
CURRENT	+ or –	INTERVAL	DATETIME
DATETIME	+ or –	UNITS	DATETIME
INTERVAL	+ or –	UNITS	INTERVAL
INTERVAL	* or /	NUMBER	INTERVAL

No other combinations are allowed. You cannot add two DATETIME values because this operation does not produce either a point in 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 Chapter 1 of 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 increase explicitly 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 shown in Figure 3-10.

Figure 3-10
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 shown in Figure 3-10, 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 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

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.

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 (35-01) YEAR TO MONTH
```

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

Chapter

4

Environment Variables

Various *environment variables* affect the functionality of your Informix products. You can set environment variables that identify your terminal, specify the location of your software, and define other parameters. The environment variables discussed in this chapter are grouped and listed alphabetically beginning on page 4-11. In addition, an index of environment variables is included at the end of this chapter, on page 4-68.

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. The settings for some of these variables might take precedence over those for their NLS counterparts.

- NLS environment variables

  You must set some or all of these X/Open standard environment variables to benefit from NLS. These might cause your product to behave differently than when their standard Informix counterparts are set. The NLS environment is described in "Using Native Language Support" on page 1-17.

- UNIX environment variables

  This section describes the major standard UNIX system environment variables that are recognized by Informix products.

Where to Set Environment Variables

You can set environment variables in the following ways:

- 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 4-8.

Setting Environment Variables in a Environment Configuration File

This is a common or private file where you can define all the environment variables that are used by Informix products. Using a 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 **$INFORMIXDIR/etc/informix.rc**. 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 user's home directory
- The file is named **.informix**
- Permissions set to readable by the user

An environment configuration file can contain comment lines (preceded by #) and variable lines and their values (separated by blanks and tabs), as 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
INFORMIXCOBTYPE rm85
INFORMIXCOBDIR /usr/lib/rmcobol
```

Use the **ENVIGNORE** environment variable to later override one or more entries in this file. Use the following Informix **chkenv** utility to perform a sanity check on the contents of an environment-configuration file, and return an error message if the file contains a bad environment-variable entry or if the file is too large:

```
chkenv filename
```

The **chkenv** utility is described in Chapter 5, "SQL Utilities."

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 (C shell), or use a period (.) to execute an environment-configuration file (Bourne or Korn shell). This allows the process to read your entry.

Setting Environment Variables at Login Time

Add the commands that set your environment variables to the following login file:

For the C shell	**.login** or **.cshrc**
For the Bourne shell or Korn shell	**.profile**

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.

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

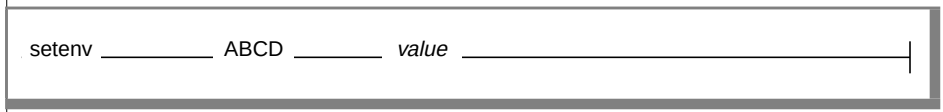
Use standard UNIX commands to set environment variables. Depending on the type of shell you use, Figure 4-1 shows how you set the **ABCD** environment variable to *value*. The environment variables are case-sensitive.

Figure 4-1
Different shell settings

C shell:	`setenv ABCD value`
Bourne shell	`ABCD=value` `export ABCD`
KORN SHELL:	`ABCD=value` `export ABCD`
Korn shell:	`export ABCD=value`

Note that **Korn**-shell syntax allows for a shortcut, as shown in the last line of Figure 4-1.

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 shell, use the syntax shown in Figure 4-1.



For more information on how to read syntax diagrams, see "Command-Line Conventions" on page 6 in the Introduction.

Viewing Your Current Settings

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

BSD UNIX: `env`

UNIX System V: `printenv`

Unsetting Environment Variables

To unset an environment variable, enter the following command:

C shell: `unsetenv ABCD`

Bourne shell or or **Korn shell:** `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 C shell **PATH** environment variable, use the following command:

```
setenv PATH /usr/informix/bin:$PATH
```

Rather than entering an explicit path name, you can use the value of the **INFORMIXDIR** environment variable (the **$INFORMIXDIR** variable), 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.

For additional information about the **INFORMIXDIR** environment variable, see the *INFORMIX*-NET*/PC Installation and Configuration Guide*.

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 as defined in the environment (shell), either by your login file or by explicitly setting the value at the shell prompt.
2. The second-highest precedence goes to the value as defined in the private environment-configuration file in the user's home directory (**~/.informix**).
3. The next-highest precedence goes to the value as defined in the common environment-configuration file (**$INFORMIXDIR/etc/informix.rc**).
4. The lowest precedence goes to the default value.

If NLS is activated, an exception exists to these rules. The setting for any of the LC_* environment variables (**LC_COLLATE**, **LC_CTYPE**, **LC_MONETARY**, **LC_NUMERIC**, **LC_TIME**) takes precedence over the setting for the **LANG** environment variable, no matter where they are set. For more information, see the discussion beginning on page 4-11.

NLS Environment Variables

This chapter alphabetically lists the NLS environment variables (**COLLCHAR**, **DBAPICODE**, **DBNLS**, **LANG**, **LC_COLLATE**, **LC_CTYPE**, **LC_MONETARY**, **LC_NUMERIC**, **LC_TIME**) that you can set to use your Informix products in an environment that supports a European language. The Informix implementation of these environment variables is based on the NLS standards as specified by X/Open in the XPG3 documents.

These environment variables can take effect only if your application is NLS-ready. For more information on NLS, see Chapter 1, "Informix Databases."

The descriptions of the NLS variables might include references to comparable Informix environment variables that you can set instead of or in addition to the NLS variables.

Rules of Precedence for NLS Environment Variables

The following list ranks the order of precedence for NLS environment variables, from highest to lowest:

1. Individual LC_* variables set in the shell
2. Individual LC_* variables set in **$HOME/.informix**
3. Individual LC_* variables set in **$INFORMIXDIR/etc/informix.rc**
4. The **LANG** variable set in the shell
5. The **LANG** variable set in **$HOME/.informix**
6. The **LANG** variable set in **$INFORMIXDIR/etc/informix.rc**
7. If no locale environment variable is set, the default locale, C, is used

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.

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. These environment variables are described in this chapter on the pages listed in the last column.

Figure 4-2

Environment Variable	Restrictions	Page
ARC_DEFAULT	ONLINE ONLY	4-15
ARC_KEYPAD	ONLINE ONLY	4-15
COLLCHAR		4-16
DBANSIWARN		4-18
DBAPICODE		4-19
DBBLOBBUF	ONLINE ONLY	4-20
DBDATE		4-21
DBDELIMITER		4-23
DBEDIT		4-23
DBLANG		4-24
DBMONEY		4-25
DBNLS		4-26
DBPATH		4-28
DBPRINT		4-31
DBREMOTECMD	ONLINE ONLY	4-32
DBSPACETEMP	ONLINE ONLY	4-33
DBTEMP	SE ONLY	4-34
DBTIME	SQL APIs only	4-34

(1 of 3)

Environment Variable	Restrictions	Page
DBUPSPACE		4-37
DELIMIDENT		4-38
ENVIGNORE		4-38
FET_BUF_SIZE	SQL APIs and DB-Access	4-39
INFORMIXC	ESQL/C ONLY	4-40
INFORMIXCOB	ESQL/COBOL ONLY	4-40
INFORMIXCOBDIR	ESQL/COBOL ONLY	4-41
INFORMIXCOBSTORE	ESQL/COBOL ONLY	4-42
INFORMIXCOBTYPE	ESQL/COBOL ONLY	4-43
INFORMIXCONRETRY		4-43
INFORMIXCONTIME		4-44
INFORMIXDIR		4-46
INFORMIXSERVER		4-46
INFORMIXSHMBASE	ONLINE ONLY	4-47
INFORMIXSTACKSIZE	ONLINE ONLY	4-48
INFORMIXTERM	DB-Access only	4-49
LANG		4-49
LC_COLLATE		4-51
LC_CTYPE		4-53
LC_MONETARY		4-55
LC_NUMERIC		4-56
LC_TIME		4-57
ONCONFIG	ONLINE ONLY	4-59
OPTCOMPIND	ONLINE ONLY	4-59

(2 of 3)

Environment Variable	Restrictions	Page
PATH		4-60
PDQPRIORITY	ONLINE ONLY	4-61
PSORT_DBTEMP	ONLINE ONLY	4-62
PSORT_NPROCS	ONLINE ONLY	4-63
SQLEXEC		4-64
SQLRM	(obsolete)	4-65
SQLRMDIR	(obsolete)	4-65
TERM		4-66
TERMCAP		4-66
TERMINFO		4-67

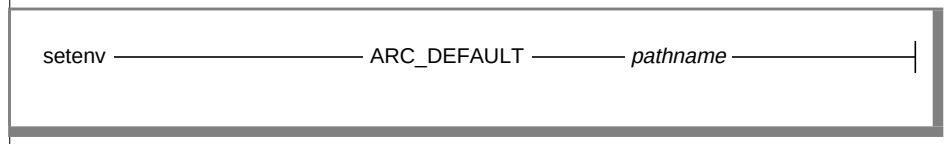
(3 of 3)

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 the INFORMIX-OnLine Dynamic Server, you can set the **ARC_DEFAULT** environment variable to indicate where a personal default qualifier file is located.



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

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

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

For more information on archiving, see the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.

ARC_KEYPAD

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

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

- It defines the terminal control attributes that allow ON-Archive to manipulate the screen and cursor.
- It defines the mappings between commands and key presses.

It defines the characters used in drawing menus and borders for an API.

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

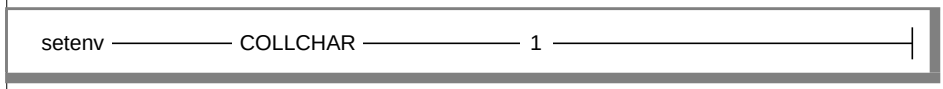
For example, to set the **ARC_KEYPAD** environment variable to specify the file **/usr/jane/tctermcap.janeroe** enter the following command:

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

For more information on archiving, see the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.

COLLCHAR

The optional **COLLCHAR** environment variable allows existing SQL API programs to take advantage of the NLS collation feature without modification. You must set **DBNLS** to `1` or `2` if you want to use **COLLCHAR**.



Set the **COLLCHAR** environment variable to the value `1` by entering the following command:

```
setenv COLLCHAR 1
```

Using pre-Version 6.0 SQL APIs

When **COLLCHAR** is set to 1, applications built using pre-Version 6.0 SQL APIs linked with the Version 6.0 or later libraries can create and sort CHAR data using the collation sequence set when an NLS database is created (the NCHAR data type), without changing any code.

During execution, the INFORMIX-OnLine Dynamic Server performs the following actions:

- It maps all NCHAR types in the database to CHAR before sending them to the SQL API application. The database server also maps all NVARCHAR types in the database to VARCHAR before sending them to the SQL API application.
- It maps all incoming CHAR references to NCHAR for internal execution. The database server also maps all incoming VARCHAR references to NVARCHAR for internal execution.

For example, when **COLLCHAR** is set, the following statement in an existing program

```
EXEC SQL CREATE TABLE mylar (..., col3 CHAR(20), ...)
```

is executed by the database server, as shown in the following example:

```
EXEC SQL CREATE TABLE mylar (..., col3 NCHAR(20), ...)
```

Conversely, after doing a DESCRIBE on a SELECT statement on the table **mylar**, which was created in the previous examples, the data type of **col3** appears as CHAR (even though it is stored in the database as NCHAR).

Using Version 6.0 or Later SQL APIs

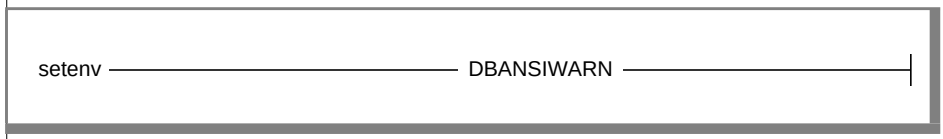
When you use Version 6.0 or later to create an NLS database or to create an application and access an NLS database using an INFORMIX-ESQL / C or INFORMIX-ESQL / COBOL application or DB-Access, the following restrictions apply:

- If **COLLCHAR** is not set, the application or NLS database can contain both CHAR and NCHAR / NVARCHAR data.
- If **COLLCHAR** is set to 1, you cannot create CHAR columns. However, the application can access an existing NLS database that contains CHAR columns, and data in those columns remains as CHAR. If the column is sorted, it is sorted as as an NCHAR column.

Important: *Direct access to the system catalog might result in an error for an application built with an SQL API prior to Version 6.0 or later because the data types are not mapped. For example, a SELECT on a system catalog table might show a column as NCHAR whereas it might otherwise appear as CHAR.*

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 shown in the following diagram, is sufficient:



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 or INFORMIX-ESQL / COBOL 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* or the *INFORMIX-ESQL/COBOL Programmer's Manual*.)

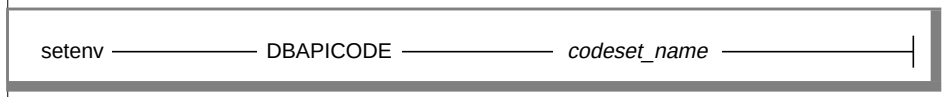
Once you set **DBANSIWARN**, Informix extension checking is automatic until you log out or unset **DBANSIWARN**. To turn off Informix extension checking, unset the **DBANSIWARN** environment variable by entering the following command:

```
unsetenv DBANSIWARN
```

DBAPICODE

The **DBAPICODE** environment variable lets systems that use nonstandard or rare code sets access databases that store data using a standard code set. Set **DBAPICODE** to let applications on different platforms use a different code set than the one stored in and used by the database server.

With **DBAPICODE** set, the applications communicate with the database server using the standard ASCII code set, but interact with the keyboard, terminal display, and program character strings using the code set specified in **DBAPICODE**.



codeset_name specifies the name of a code set mapping file, usually in the format *language_territory.codeset*

To use a specific **DBAPICODE** setting, there must be a mapping file for that code set in the message directory. For example, for an NLS-ready ESQL/C application to use the code set specified in the following command with an NLS database created with the code set **FR_fr.646**, there must be a mapping file called **mFR_fr.646** in the message directory **$INFORMIXDIR/msg**, or the directory must be pointed to by the **DBLANG** or **LANG** setting:

```
setenv DBAPICODE mFR_fr.646
```

The recommended setting for **DBAPICODE** follows a modified version of the form shown for the other NLS categories (that is, *language_territory.codeset*). However, this might not be possible because many terminals or platforms have unconventional code sets.

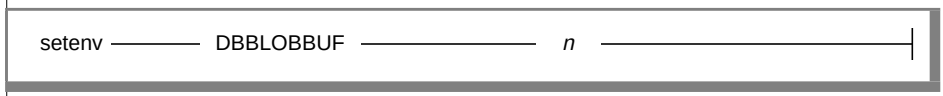
DBAPICODE specifies the character-mapping file between itself and the standard code set. In NLS-ready systems, the standard code set is defined in the **LANG** environment variable. In non-NLS systems, the standard code set is the default 8-bit character set.

Creating Mapping Files

The Informix **crtcmap** utility helps you create mapping files among non-standard and standard code sets for use with NLS. It prepends **m** to the name of the mapping file it creates. For example, it renames the output file **FR_fr.646** as **mFR_fr.646**. The **crtcmap** utility is described on page 5-5.

DBBLOBBUF

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



n represents the maximum size of a blob in kilobytes.

If the blob is smaller than the default of 10 kilogytes or the setting of the **DBBLOBBUF** environment variable, it is temporarily stored in memory. If the blob 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 shown in the following example:

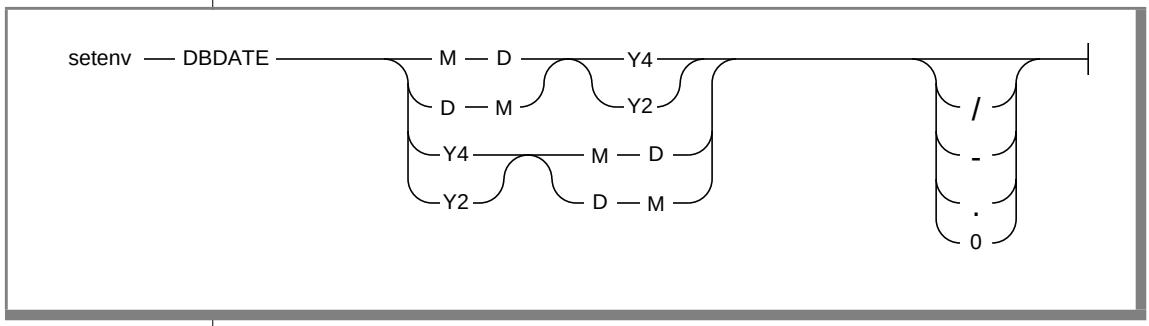
```
setenv DBBLOBBUF 15
```

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.

DBDATE

The **DBDATE** environment variable specifies the display formats of date values. 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 day and the month.
Y2, Y4	are characters that represent the year, and the number of digits in the year.

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:

setenv DBDATE	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

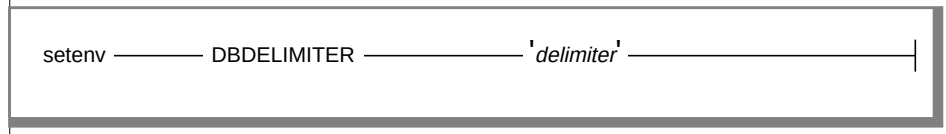
Notice that the formats `Y4MD*` (unacceptable separator defined) and `MDY4` (no separator defined) both display the default (slash) as a separator.

Important: *Certain routines called by INFORMIX-ESQL/COBOL or INFORMIX-ESQL/C can use the* ***DBTIME*** *variable, rather than* ***DBDATE****, to set DATETIME formats to international specifications. See the discussion of the DBTIME environment variable on page 4-34 and the INFORMIX-ESQL/C Programmer's Manual or the INFORMIX-ESQL/COBOL Programmer's Manual for more information.*

If you are using NLS, the contents of a declared DBDATE variable take precedence over the contents of the LC_TIME variable. However, the LC_TIME setting takes precedence over the default DBDATE format. (An exception is when the hardware does not support the standard date format or returns a default format that is too complicated. In those instances, **LC_TIME** is ignored, and the **DBDATE** format is used). See the discussion of the LC_TIME environment variable on page 4-57 for more information.

DBDELIMITER

The **DBDELIMITER** environment variable specifies the field delimiter used by the **dbexport** utility and with the LOAD and UNLOAD statements.



delimiter is the field delimiter for unloaded data files.

The delimiter can be any single character, except the characters in the following list:

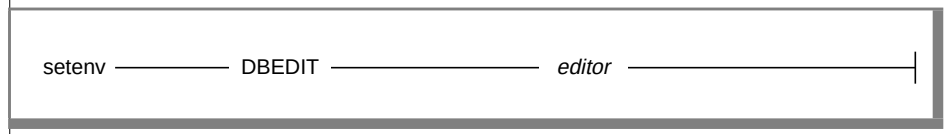
- Hexadecimal numbers (0 through 9, a through f, A through F)
- NEWLINE or CTRL-J
- The backslash symbol (\)

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 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.



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

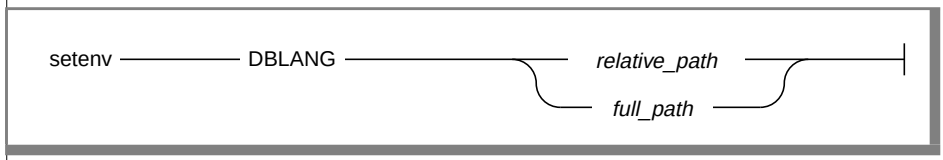
For most systems, the default editor is **vi**. If you use another editor, be sure that it creates flat ASCII files. Some word processors in *document mode* introduce printer control characters that can interfere with operation of your Informix product.

To specify the EMACS text editor, set the **DBEDIT** environment variable by entering the following command:

```
setenv DBEDIT emacs
```

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 your application.



relative_path	is the subdirectory of **$INFORMIXDIR**
full_path	the full pathname of the directory that contains the compiled message files.

The default subdirectory is **$INFORMIXDIR/msg** and the compiled message files have the suffix **.iem**. If you want to use a message directory other than **$INFORMIXDIR/msg**, perform the following steps:

1. Use the **mkdir** command to create the appropriate subdirectory in **$INFORMIXDIR**.
2. Set the owner and group of the subdirectory to **informix** and the access permission for this directory to 755.
3. Set the **DBLANG** environment variable to the new subdirectory, specifying only the subdirectory name and not the full pathname.
4. Copy the **.iem** files to the new message directory specified by **$INFORMIXDIR/$DBLANG**. All files in the message directory should have the owner and group **informix** and access permission 644.

To use NLS, you can set **DBLANG** to indicate the subdirectory under **$INFORMIXDIR/msg** in which the message files for each supported native language reside. For example, set **DBLANG** for your product to use the French message files by entering the following command:

```
setenv DBLANG french
```

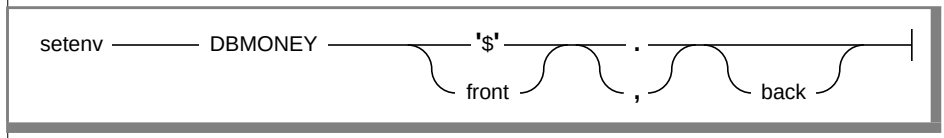
The database server accesses the message files in subdirectories of **$INFORMIXDIR/msg** in the following order:

1. In **$INFORMIXDIR/msg/$DBLANG**, if **DBLANG** is set (only if a *relative_path* was used with **DBLANG**).
2. In **$INFORMIXDIR/msg/$LANG**, if **LANG** is set and **DBLANG** is not set.
3. In **$INFORMIXDIR/msg**, if neither **DBLANG** nor **LANG** is set.

See the discussion of the **LANG** environment variable on page 4-49 for information on how to select a language environment for your product when NLS functionality is enabled.

DBMONEY

The **DBMONEY** environment variable specifies the display format for money values.



$ 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 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 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.

The default setting for **DBMONEY** is where a dollar sign ($) precedes the money value, a period (.) separates the integral from the fractional part of the money value, and no *back* symbol appears. 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`.

The contents of a declared **DBMONEY** environment variable take precedence over the contents of the **LC_MONETARY** variable that can be set for NLS. However, the **LC_MONETARY** setting takes precedence over the default **DBMONEY** format. See the discussion of **LC_MONETARY** on page 4-55.

DBNLS

Set the **DBNLS** environment variable to activate NLS functionality in your Informix products. If **DBNLS** is not set, the settings for the other NLS environment variables listed in this section do not take effect.

To enable NLS functionality in Version 6.0 or later products, set the **DBNLS** environment variable to the value 1 by entering the following command:

```
setenv DBNLS 1
```

Important: *A database created after **DBNLS** is set in an NLS database can be accessed only by applications running in a correctly set environment.*

If you want to use a pre-Version 6.0 Informix product with a Version 6.0 or later database server, set **DBNLS** to 2. This allows, for example, existing INFORMIX-4GL or pre-Version 6.0 INFORMIX-ESQL/C programs to access NLS databases and obtain proper sorting and collation of character data types from the database server. Setting **DBNLS** to 2 also allows Version 6.0 or later NLS applications with different locales to access an NLS database.

The following table describes what happens if you create or access an NLS database with Version 6.0 or later ESQL/C, ESQL/COBOL, or DB-Access:

	If DBNLS Is Set to 1	If DBNLS Is Set to 2
If you create an NLS database	The NLS database automatically inherits the locale of the application.	
If you access an NLS database	The application locale must match the desired locale for the NLS database. If not, an error is returned.	The locale of the application and the database can differ. The database server operates in the locale of the database.

If needed, you can deactivate the **DBNLS** environment variable with either of the following commands:

```
setenv DBNLS
unsetenv DBNLS
```

When using DB-Access, you can infer whether **DBNLS** is set through the Session option of the DB-Access main menu. If **DBNLS** is set, the screen shows NLS capabilities and attributes; if **DBNLS** is not set, no NLS capabilities are listed. See Chapter 6 in the *DB-Access User Manual* for details.

DBPATH

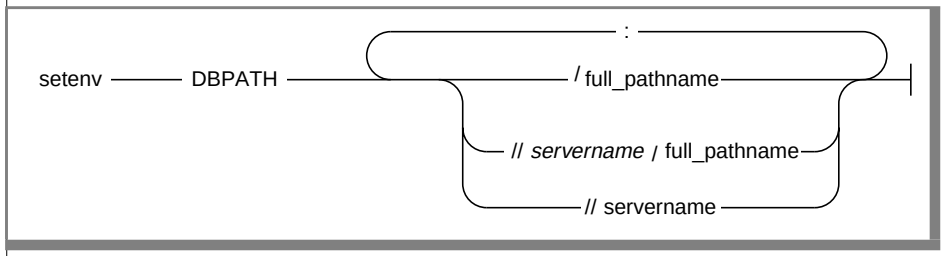
Use **DBPATH** to identify the database servers that contain databases (if you are using the INFORMIX-OnLine Dynamic Server), or the directories and / or database servers that contain databases (if you are using INFORMIX-SE). The **DBPATH** environment variable also specifies a list of directories (in 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, for INFORMIX-SE, 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 4-9.



full_pathname is a valid full pathname of a directory in which **.sql** files are stored or in which INFORMIX-SE databases are stored.

servername is the name of an INFORMIX-OnLine Dynamic Server or INFORMIX-SE database server on which databases are stored. You cannot reference database files with a servername.

DBPATH can contain up to 16 entries. Each entry (*pathname*, or *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. For INFORMIX-SE, if no directory is specified in the statement, the default directory is searched for the database. (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 INFORMIX-OnLine Dynamic Server databases, only the **.sql** files in the current working directory are displayed; for INFORMIX-SE databases, the **.sql** files in the directory containing the selected database 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. If you are using DB-Access with INFORMIX-SE, you can search for a database and **.sql** scripts; with the INFORMIX-OnLine Dynamic Server, you can look only for **.sql** scripts.

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

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

As 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 4-46.) 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 allows you to locate only databases and not locate command files.

The OnLine 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-OnLine Dynamic Server Administrator's Guide* or the *INFORMIX-SE Administrator's Guide*.

For example, if INFORMIXSERVER is set to **quality**, you can search for an INFORMIX-OnLine Dynamic 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
```

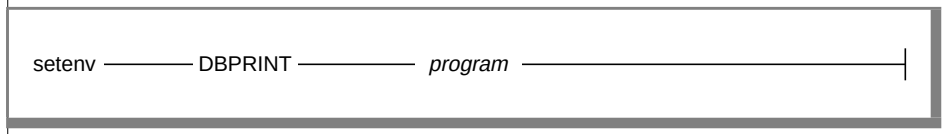
If you are using DB-Access in this example, the names of all the databases on the **quality** and **marketing** database servers are displayed with the Select option of the Database menu.

For INFORMIX-SE, you can set **DBPATH** to contain only the database server names (and no directory names) if you want to locate databases and not command scripts.

- If you specify a local SE database server, the current working directory is searched for databases.
- If you specify a remote SE database server, the search for databases is done in the **login** directory of the user on the computer where the database server is running.

DBPRINT

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



program names any command, shell script, or UNIX utility that handles standard ASCII input.

The default program is found in the following list:

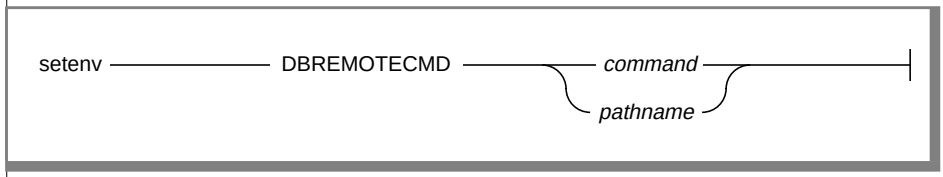
- For most BSD UNIX systems, the default program is **lpr**.
- For UNIX System V, the default program is usually **lp**.

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-OnLine Dynamic 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.



command	is the command to override the default remote shell.
pathname	is the pathname to override the default remote shell.

Informix highly recommends the use of the full pathname syntax on the interactive UNIX platform to avoid problems with similar-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
```

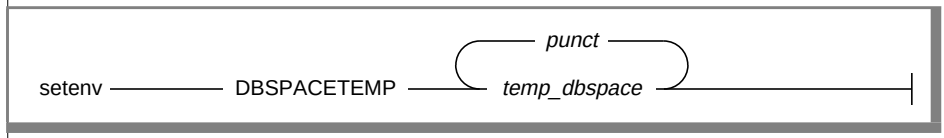
Set the **DBREMOTECMD** environment variable to specify the full pathname by entering the following command:

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

For more information on **DBREMOTECMD**, see the discussion in the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* about using remote tape devices with the INFORMIX-OnLine Dynamic Server for archives, restores, and logical log backups.

DBSPACETEMP

If you are using OnLine, you can set your **DBSPACETEMP** environment variable to specify dbspaces for building temporary tables and for holding the temporary files used for sorting. 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 **INFORMIX-**OnLine Dynamic Server configuration file. For example, you might set **DBSPACETEMP** 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.

If you do not set the **DBSPACETEMP** environment variable, the default is the root dbspace unless you create your temporary table with the CREATE TEMP TABLE statement, in which case the default is the dbspace that contains the database for the temporary table. If the dbspace named by the environment variable not exist, OnLine uses the root dbspace. OnLine does not create a new dbspace with the given name.

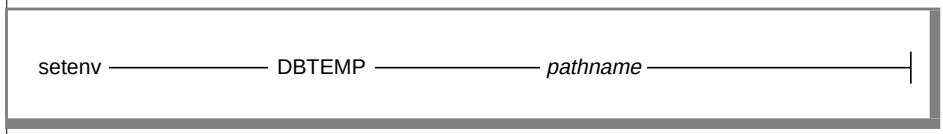
For sorting space, OnLine uses the following disk space for writing temporary information, in the following order:

1. The operating system directory or directories specified by the environment variable **PSORT_DBTEMP**, if set
2. The dbspace or dbspaces specified by the environment variable **DBSPACETEMP**, if 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 its temporary files and temporary tables.



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
```

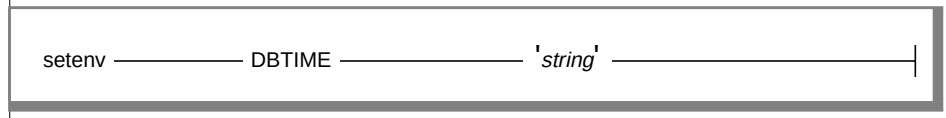
If you do not set **DBTEMP**, temporary files are created in **/tmp**. If **DBTEMP** is not set, temporary tables are created in the directory of the database (that is, the **.dbs** directory).

OnLine uses **DBSPACETEMP** to specify the location of temporary files.

DBTIME

You can set the **DBTIME** environment variable to manipulate DATETIME formats so the formats conform more closely to various international or local TIME conventions. **DBTIME** takes effect only when you call certain INFORMIX-ESQL / C or INFORMIX-ESQL / COBOL DATETIME routines; otherwise, use the **DBDATE** environment variable. (See the *INFORMIX-ESQL/C Programmer's Manual* or the *INFORMIX-ESQL/COBOL Programmer's Manual* for details.)

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.



string the formatting directives that you can use to create the formatting 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].
%F*n*	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].
%y	is replaced by the year as a 2-digit decimal number [00,99]. The format for this value is taken literally: “88” means “0088”, not “1988”.
%Y	is replaced by the year as a 4-digit decimal number.
%%	is replaced by % (to allow % in the format string).

For example, to convert a DATETIME *year* TO *second* to the ASCII string format shown in the following example:

```
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
```

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 zeroes 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:

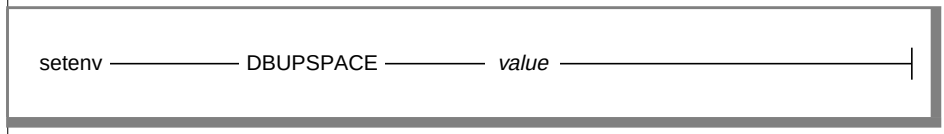
- If a conversion specification supplies fewer digits than specified by a precision, it is padded with leading zeroes.
- If a conversion specification supplies more characters than specified by a precision, excess characters are truncated on the right.
- If no field width or precision is specified for 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 described earlier.

See the discussion of **DBDATE** and **LC_TIME** on pages 4-21 and 4-57, respectively, 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.



value represents a disk space amount in kilobytes

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

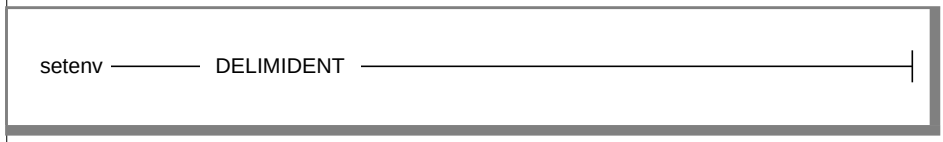
```
setenv DBUPSPACE 2500
```

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 specified in **DBUPSPACE**.

DELIMIDENT

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



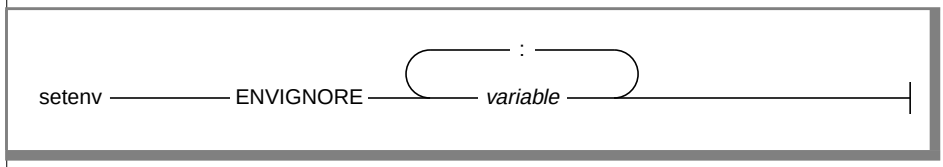
You can use delimited identifiers to specify identifiers that are identical to reserved keywords, such as TABLE or USAGE. You can also use them to specify database identifiers that contain nonalpha characters, but you cannot use them to specify storage identifiers that contain non-alpha characters. Note 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 and ESQL /COBOL 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:

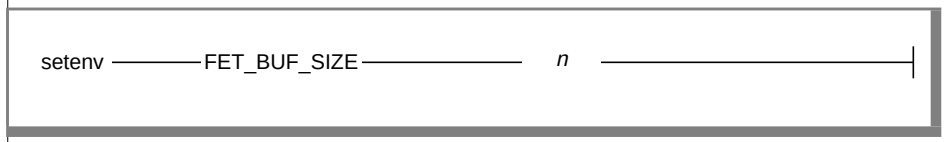
```
setenv ENVIGNORE DBPATH:DBMONEY
```

The common environment-configuration file is stored in **$INFORMIXDIR/etc/informix.rc**. The private environment-configuration file is stored in the user's home directory as **.informix**. See "Where to Set Environment Variables" on page 4-6 for information on creating or modifying an environment-configuration file.

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.

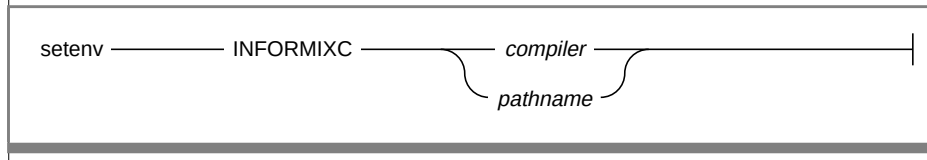
If the buffer size is set to less than the default size or is out of the range of the small integer value, no error is raised. The new buffer size is ignored.

For example, to set a buffer size to 5,000, 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.



compiler is the name of the C compiler.
pathname is the full pathname of the C compiler.

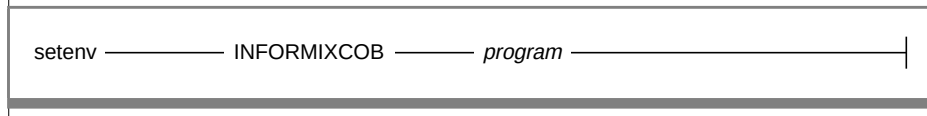
For example, to specify the GNU C compiler, enter the following command:

```
setenv INFORMIXC gcc
```

The setting is required only during the C compilation stage.

INFORMIXCOB

The **INFORMIXCOB** environment variable specifies the program name of the COBOL compiler that you use with INFORMIX-ESQL / COBOL. It identifies the command that calls up the compiler environment. You must set this environment variable before you compile your ESQL / COBOL program.



program is the program name of the COBOL compiler.

The program name depends on the manufacturer of the COBOL compiler, as shown in the following list:

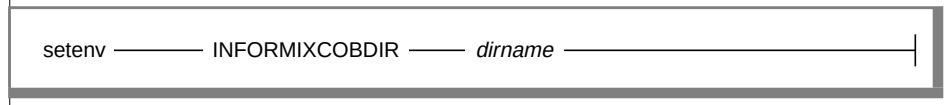
Compiler manufacturer	Enter the command:
Micro Focus	`setenv INFORMIXCOB cob`
Ryan-McFarland (Liant)	`setenv INFORMIXCOB rmcobol`

Your COBOL compiler might require you to set additional (UNIX) environment variables as listed in the compiler documentation. See the product manuals and the *INFORMIX-ESQL/COBOL Programmer's Manual* for compiler-specific information.

Refer to the *INFORMIX-ESQL/COBOL Programmer's Manual* for more information about **INFORMIXCOB**.

INFORMIXCOBDIR

The **INFORMIXCOBDIR** environment variable specifies the directory where the COBOL compiler resides. You must set this environment variable before you compile your INFORMIX-ESQL / COBOL program and create the run-time product.

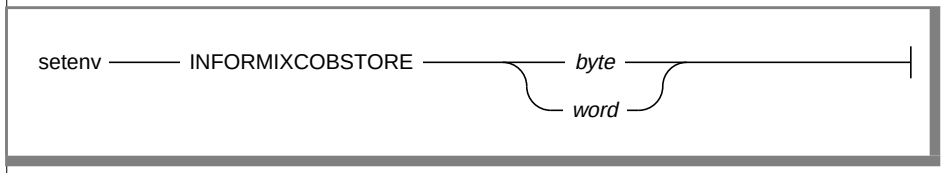


dirname is the full directory pathname of the directory where the COBOL compiler, run-time library, and objects reside.

Refer to the *INFORMIX-ESQL/COBOL Programmer's Manual* for more information about **INFORMIXCOBDIR**.

INFORMIXCOBSTORE

The **INFORMIXCOBSTORE** environment variable applies only to the Micro Focus (MF) COBOL/2 environment. It specifies the type of storage to use during compilation. You must set **INFORMIXCOBSTORE** before you compile your INFORMIX-ESQL/COBOL program and create the run-time product.



byte	specifies byte storage.
word	specifies word storage.

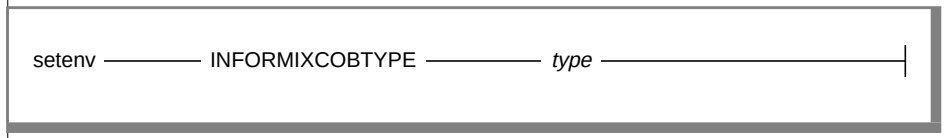
The number of bytes needed to store BINARY or COMPUTATIONAL data is based on the size (maximum number of digits) specified in the PICTURE clause. The MF COBOL/2 compiler also considers whether *byte* or *word* storage is specified when determining the number of bytes needed to store BINARY and COMPUTATIONAL data. (MF COBOL/2 uses only byte storage.)

If **INFORMIXCOBSTORE** is not set, the default storage mode is byte, which is more restrictive of available data type choices. If you are using byte storage, the only legal PIC sizes are 3, 4, 7, 8, and 9. If you are using word storage, PIC sizes can range from 1 through 9.

For a table showing the storage allocation for MF compilers, see the *INFORMIX-ESQL/COBOL Programmer's Manual*.

INFORMIXCOBTYPE

The **INFORMIXCOBTYPE** environment variable specifies a code that identifies the manufacturer of your COBOL compiler. You must set **INFORMIXCOBTYPE** before you compile an INFORMIX-ESQL / COBOL program and create the run-time product.



type indicates the manufacturer of the COBOL compiler that you use with ESQL / COBOL, as shown in the following list:

`mf2`	Micro Focus
`rm85`	Ryan-McFarland (Liant)

Refer to the *INFORMIX-ESQL/COBOL Programmer's Manual* for more information about **INFORMIXCOBTYPE**.

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.

Set the **INFORMIXCONRETRY** environment variable as shown in the following syntax:

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
```

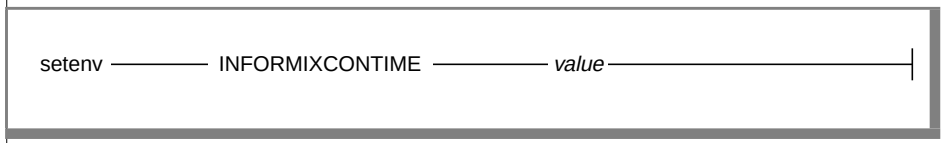
The default value for **INFORMIXCONRETRY** is one retry after the initial connection attempt. The **INFORMIXCONTIME** setting, described in the following section, takes precedence over the **INFORMIXCONRETRY** setting.

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.

Set the **INFORMIXCONTIME** environment variable as shown in the following syntax:



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**.

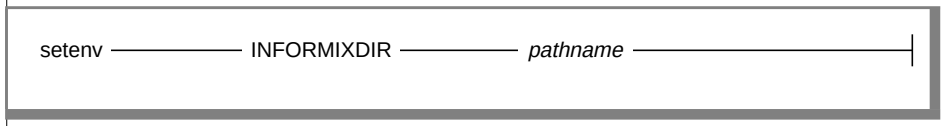
If execution of the CONNECT statement involves searching **DBPATH**, the following rules apply:

- 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 **INFORMIX-CONTIME** setting 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 the INFORMIX-OnLine Dynamic Server or INFORMIX-SE database server, set **INFORMIXDIR** to the appropriate directory name for the version that you want to access. For information about when to set the **INFORMIXDIR** environment variable, see the *UNIX Products Installation Guide*.



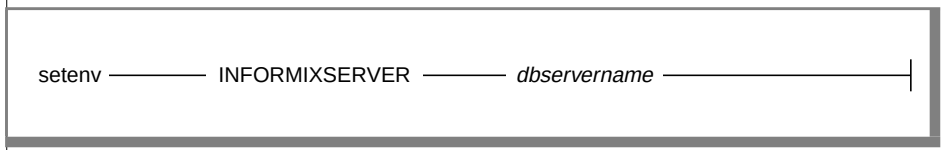
pathname is the directory path where the product files are installed.

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

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

INFORMIXSERVER

The **INFORMIXSERVER** environment variable specifies the default database server to which an explicit or implicit connection is made by an SQL API client or the DB-Access utility. The database server can be either INFORMIX-OnLine Dynamic Server or INFORMIX-SE and can be either local or remote. You must always set **INFORMIXSERVER** before using an Informix product.



dbservername is the name of the default database server.

The value of **INFORMIXSERVER** must correspond to a valid *dbservername* entry in the **$INFORMIXDIR/etc/sqlhosts** file on the computer running the application. The *dbservername* must be specified using lowercase characters

and cannot exceed 18 characters for the INFORMIX-OnLine Dynamic Server and cannot exceed 10 characters for the INFORMIX-SE database 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.

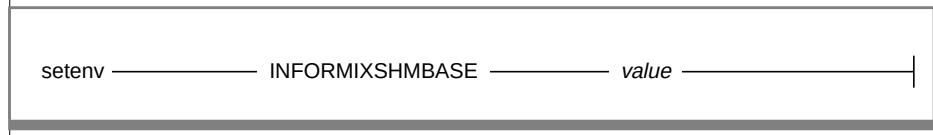
Important: *INFORMIXSERVER must be set even if the application or DB-Access does not use implicit or explicit default connections.*

INFORMIXSHMBASE

The **INFORMIXSHMBASE** environment variable affects only client applications connected to an INFORMIX-OnLine Dynamic Server using the IPC shared-memory (**ipcshm**) 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.



value is used to calculate the memory address.

The INFORMIX-OnLine Dynamic 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-OnLine Dynamic Server Administrator's Guide*.

INFORMIXSTACKSIZE

INFORMIXSTACKSIZE specifies the stack size (in kilobytes) that OnLine uses for a particular client session. Use **INFORMIXSTACKSIZE** to override the value of the ONCONFIG parameter STACKSIZE for a particular application or user.

value is the stack size for SQL client threads in kilobytes.

For example, decrease the **INFORMIXSTACKSIZE** to 20 kilobytes, by entering the following command:

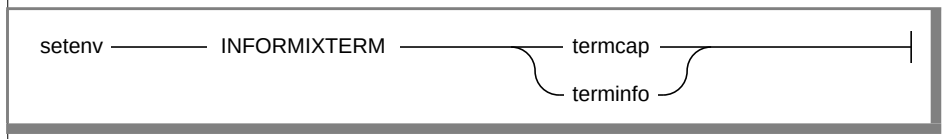
```
setenv INFORMIXSTACKSIZE 20
```

If **INFORMIXSTACKSIZE** is not set, the stack size is taken from the OnLine 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: *See the INFORMIX-OnLine Dynamic Server Administrator's Guide for specific instructions for setting this value. If you incorrectly set the value of **INFORMIXSTACKSIZE**, it can cause the INFORMIX-OnLine Dynamic 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** and **terminfo** files 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.

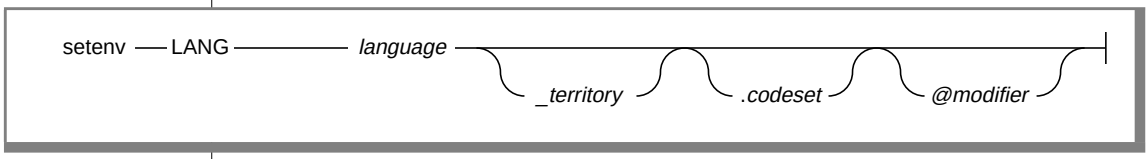
You can use the **termcap** file supplied by Informix, the system **termcap** file, or a **termcap** file that you create. You must set the TERMCAP environment variable if you do not use the default **termcap** file.

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 online Version 7.1 Machine Notes for your product.

LANG

Set the UNIX **LANG** environment variable to select the specific language environment (or *locale*) for your Informix products. The **LANG** setting implies certain default values for the language environment that you can modify or override by setting various LC_* environment variables. The setting for the LC_* environment variables always takes precedence over the **LANG** setting. For example, if a **LANG** setting for French is specified in the shell and **LC_COLLATE** is set for German in an environment-configuration file, the **LC_COLLATE** setting takes precedence. If **LANG** is not set, the default value

is C. (Most UNIX systems include, as part of the basic operating system, a locale whose name is C. This locale is used unless you explicitly change to some other locale.)



.codeset	modifies the default cultural convention settings implied in the *language* setting. For instance, the default International Standards Organization (ISO) code set might be 8859/1, but a user might prefer to use the ISO 646 code set or the ISO 6937:1983 code set instead.
@modifier	modifies the default cultural convention settings implied in the *language* setting. For example, you can set *@modifier* to specify the dictionary or telephone-book sorting order for a locale.
_territory	modifies the default cultural convention settings implied in the *language* setting. For example, you can set *_territory* to specify the Swiss version of the French, German, or Italian language.
language	is the name representing the language and cultural conventions for a specific locale, such as France, Germany, or Italy. The locale specified implies default settings for the *_territory* and *.codeset*.

Valid settings for the *.codeset*, *language*, *@modifier*, and *_territory* fields might vary among operating systems. They are not standardized and thus might not be supported by individual X/Open systems. Example settings in this section use a particular implementation for consistency but do not imply a specific naming convention.

The **LANG** information is transmitted with the request for database service when the database connection is established. The database server uses the information to set up the proper environment or to reject the service request if a potential exists for data inconsistency when a database connection is established with another database server.

The setting for the **LANG** environment variable identifies the active language and code set of the database application when the NLS functionality is enabled. This value is used throughout the life of the database to ensure the proper use of code sets.

Set the **LANG** environment variable to French by entering the following command:

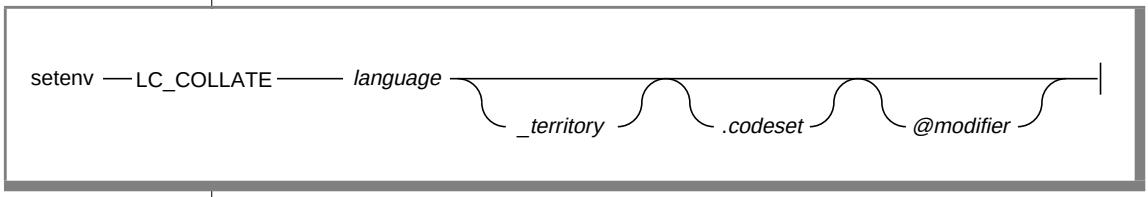
```
setenv LANG FR
```

You also can set the Informix environment variable **DBLANG** to indicate the subdirectory under **$INFORMIXDIR/msg** in which the message files for each supported native language reside. See the discussion of the **DBLANG** environment variable on page 4-24 for more information.

When using DB-Access, you can see the **LANG** setting and the mode (NLS or non-NLS) through the Session option on the DB-Access main menu. See Chapter 6 in the *DB-Access User Manual* for details.

LC_COLLATE

Set the **LC_COLLATE** environment variable to specify the collation sequence for regular expressions and character string comparison operations and to ensure the proper use of code sets.



.codeset	modifies the default cultural convention settings implied in the *language* setting. For instance, the default ISO code set might be 8859/1, whereas a user might prefer to use the ISO 646 code set or the ISO 6937:1983 code set instead.
@modifier	modifies the default cultural convention settings implied in the *language* setting. For example, you can set *@modifier* to specify the dictionary or telephone-book sorting order for a locale.

_territory	modifies the default cultural convention settings implied in the *language* setting. For example, you can set *_territory* to specify the Swiss version of the French, German, or Italian language.
language	is the name representing the language and cultural conventions for a specific locale, such as France, Germany, or Italy. The locale specified implies default settings for the *_territory* and *.codeset*.

Valid settings for the *.codeset*, *language*, *@modifier*, and *_territory* fields might vary among operating systems. They are not standardized and thus might not be supported by individual X/Open systems. Example settings in this section use a particular implementation for consistency but do not imply a specific naming convention.

The database server saves the **LC_COLLATE** setting when you create a database. The **LC_COLLATE** setting is used throughout the lifetime of the database. If **LC_COLLATE** is not set, the default is the value set for the **LANG** environment variable. If **LANG** is not set, the default value is C. If **LC_COLLATE** is set, this value has the following consequences:

- It affects the evaluation of *regular expressions* involving the NCHAR and NVARCHAR native-character fields. For example, the comparison of the regular expression [A-P] uses country-specific values instead of ASCII values. The range might thus include characters outside of the A-P range.
- It affects the use of the native-character fields NCHAR and NVARCHAR in *character-string* handling, in the following ways:
  - Indexes built on NCHAR and NVARCHAR are sorted using a country-specific comparison operator. For example, in Spanish, the double character *ll* might be treated as a single unique character.
  - Logical predicates containing the NCHAR and NVARCHAR character fields in the WHERE clause of a SELECT statement are evaluated using the comparison operator.
  - The character fields NCHAR and NVARCHAR in an ORDER BY clause are evaluated by applying the comparison operator.

For example, you can set the **LC_COLLATE** environment variable to specify the dictionary order as implemented in the Swiss version (CH) of the German language (DE) by entering the following command:

```
setenv LC_COLLATE DE_ch@dict
```

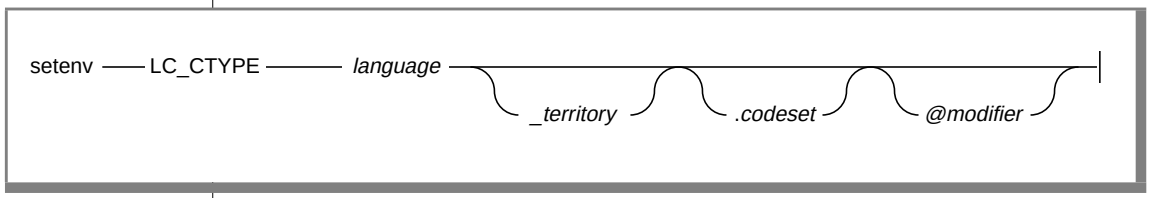
You can set the **LC_COLLATE** environment variable to specify the German telephone-book sorting order by entering the following command:

```
setenv LC_COLLATE DE@telephone
```

When using DB-Access, you can see the **LC_COLLATE** setting and the mode (NLS or non-NLS) for the application through the Session option of the DB-Access main menu. In addition, the Nls option of the DATABASE INFO menu shows the **LC_COLLATE** information stored in the active database. See Chapters 4 and 6 in the *DB-Access User Manual* for details.

LC_CTYPE

Set the **LC_CTYPE** environment variable to specify various character attributes such as the character classification and case conversion of regular expressions and character-evaluation functions, and to ensure the proper use of code sets. The **LC_CTYPE** setting determines character attributes such as spaces, punctuation, uppercase letters, and so on.



.codeset	modifies the default cultural convention settings implied in the *language* setting. For instance, the default ISO code set might be 8859/1, whereas a user might prefer to use the ISO 646 code set or the ISO 6937:1983 code set instead.
@modifier	modifies the default cultural convention settings implied in the *language* setting. For example, you can set *@modifier* to specify the dictionary or telephone-book sorting order for a locale.

_territory	modifies the default cultural convention settings implied in the *language* setting. For example, you can set *_territory* to specify the Swiss version of the French, German, or Italian language.
language	is the name representing the language and cultural conventions for a specific locale, such as France, Germany, or Italy. The locale specified implies default settings for the *_territory* and *.codeset*.

Valid settings for the *.codeset*, *language*, *@modifier*, and *_territory* fields might varyamong operating systems. They are not standardized and thus might not be supported by individual X/Open systems. Example settings in this section use a particular implementation for consistency but do not imply a specific naming convention.

The database server saves the **LC_CTYPE** setting when you create the database. The **LC_CTYPE** setting is used throughout the lifetime of the database.

For example, set the **LC_CTYPE** environment variable to specify the Swiss cultural conventions and the ISO 6937:1983 code set by entering the following command:

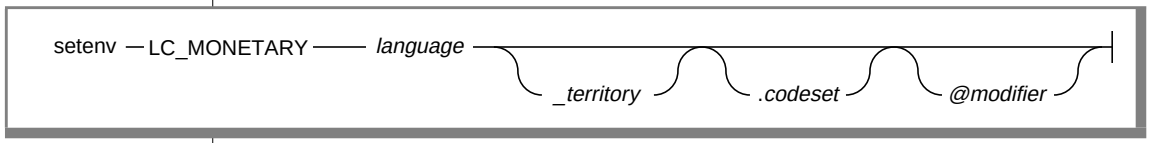
```
setenv LC_CTYPE FR_ch.6937
```

If **LC_CTYPE** is not set, the default is the value set for the **LANG** environment variable. If **LANG** is not set, the default value is C.

When using DB-Access, you can see the **LC_CTYPE** setting and the mode (NLS or non-NLS) through the Session option of the DB-Access main menu. In addition, the Nls option on the DATABASE INFO menu shows the **LC_CTYPE** information stored in the active database. See Chapters 4 and 6 in the *DB-Access User Manual* for details.

LC_MONETARY

The LC_MONETARY environment variable defines the rules and symbols used to format monetary values. Use it to specify the format for the national currency symbol, including its position within a numeric value, and the money format.



.codeset	modifies the default cultural convention settings implied in the *language* setting. For instance, the default ISO code set might be 8859/1, whereas a user might prefer to use the ISO 646 code set or the ISO 6937:1983 code set instead.
@modifier	modifies the default cultural convention settings implied in the *language* setting. For example, you can set *@modifier* to specify the dictionary or telephone-book sorting order for a locale.
_territory	modifies the default cultural convention settings implied in the *language* setting. For example, you can set *_territory* to specify the Swiss version of the French, German, or Italian language.
language	is the name representing the language and cultural conventions for a specific locale, such as France, Germany, or Italy. The locale specified implies default settings for the *_territory* and *.codeset*.

Valid settings for the ***.****codeset*, *language*, *@modifier*, and *_territory* fields might vary among operating systems. They are not standardized and thus might not be supported by individual X/Open systems. Example settings in this section use a particular implementation for consistency but do not imply a specific naming convention.

If **LC_MONETARY** is not set, the default is the value set for the **LANG** environment variable. If **LANG** is not set, the default value is C.

Set **LC_MONETARY** to the language whose currency symbol and format you want. For example, specify the French currency symbol and format inherent in the ISO 8859/1 code set, by entering the following command:

```
setenv LC_MONETARY FR_Fr.88591
```

Specify the British currency symbol and format, by entering the following command:

```
setenv LC_MONETARY EN_Uk.646
```

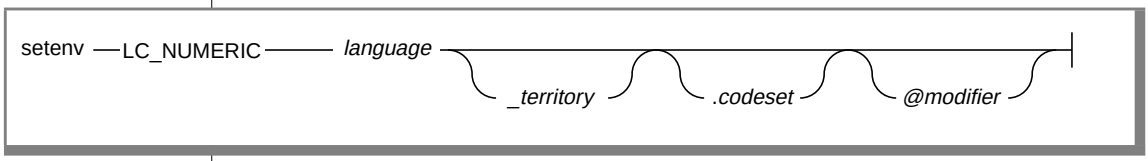
The currency symbol can be located in front of, after, or within the numeric value. For example, in Germany, Switzerland, and France, respectively, the monetary value 120.50 can be represented as shown in the following list:

- 120,50DM or DM120,50
- Fr120,50 or sFr120.50
- 120,50FF or 120,50F

The contents of a declared **DBMONEY** variable take precedence over the contents of the **LC_MONETARY** variable. However, **LC_MONETARY** takes precedence over the default **DBMONEY** format. See the discussion of **"DBMONEY" on page 4-25**.

LC_NUMERIC

Set the **LC_NUMERIC** environment variable to specify the default format and decimal separator for numeric values. The **LC_NUMERIC** setting takes effect only at the application level; it has no effect on the database server unless the value passed to the database server is in a quoted string. In that case, the conversion is based on the format set in **LC_NUMERIC**.



Valid settings for the *.codeset*, *language*, *@modifier*, and *_territory* fields might vary among operating systems. They are not standardized and thus might not be supported by individual X/Open systems. Example settings in this section use a particular implementation for consistency but do not imply a specific naming convention.

The application must convert the numeric data specified in the local format to the standard ANSI SQL format, where a period is the decimal separator.

For example, set the **LC_NUMERIC** environment variable to specify German usage by entering the following command:

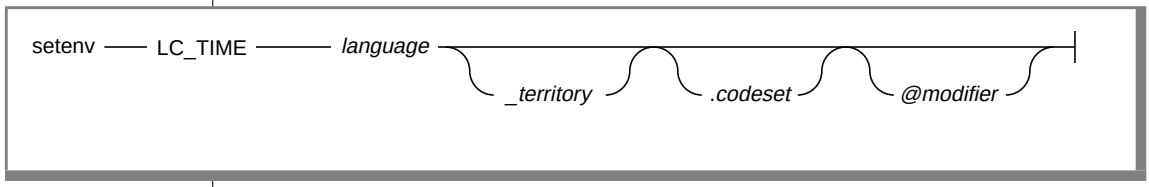
```
setenv LC_NUMERIC DE_De.88591
```

Here, `DE` means Deutsch, `De` represents Deutschland, and `88591` represents the German variant of the 8-bit character set ISO 88591:1983.

If **LC_NUMERIC** is not set, the default is the value set for the **LANG** environment variable. If **LANG** is not set, the default value is C.

LC_TIME

Set the **LC_TIME** environment variable to define the format for the national date and time. This category includes the country-specific names and abbreviations for the days of the week and for months.



.codeset	modifies the default cultural convention settings implied in the *language* setting. For instance, the default ISO code set might be 8859/1, whereas a user at a locale might prefer to use the ISO 646 code set or the ISO 6937:1983 code set instead.
@modifier	modifies the default cultural convention settings implied in the *language* setting. For example, you can set *@modifier* to specify the dictionary or telephone-book sorting order for a locale.

_territory	modifies the default cultural convention settings implied in the *language* setting. For example, you can set *_territory* to specify the Swiss version of the French, German, or Italian language.
language	is the name representing the language and cultural conventions for a specific locale, such as France, Germany, or Italy. The locale specified implies default settings for the *_territory* and *.codeset*.

Valid settings for the *.codeset*, *language*, *@modifier*, and *_territory* fields might vary among operating systems. They are not standardized and thus might not be supported by individual X/Open systems. Example settings in this section use a particular implementation for consistency but do not imply a specific naming convention.

The **LC_TIME** setting takes effect at the application level. It also has an effect on the database server: it affects the **DBDATE** environment variable setting for date values (but not DATETIME or INTERVAL formats) that the database server accepts. The application must convert the date and time specified in the local format to the standard ANSI SQL format. In addition, the database server accepts only the string representation of DATETIME and INTERVAL in the standard ANSI SQL format.

For example, you might set the **LC_TIME** environment variable to specify Italian usage by entering the following command:

```
setenv LC_TIME IT_It.88591
```

If **LC_TIME** is not set, the default is the value set for the **LANG** environment variable. If **LANG** is not set, the default value is C.

The contents of a declared **DBDATE** variable take precedence over the contents of the **LC_TIME** variable. However, **LC_TIME** takes precedence over the default **DBDATE** format, except in those instances when the computer does not support the standard date format, or returns a default format that is too complicated. If the **LC_TIME** format (month, day, year, and separator) is not compatible with or accepted by **DBDATE**, the default **DBDATE** format is used. See the discussion of **DBDATE** on page 4-21.

ONCONFIG

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

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

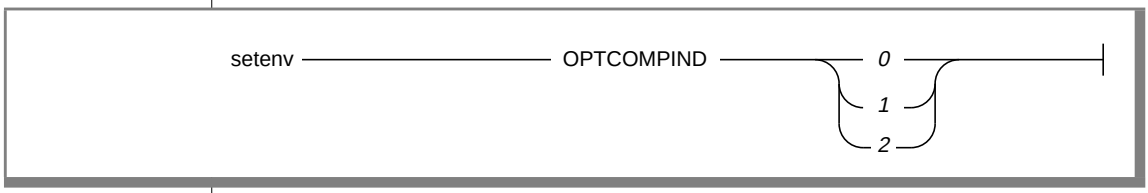
Prepare the **ONCONFIG** file by making a copy of the **onconfig.std** file and modifying the copy. Informix recommends that you name the **ONCONFIG** file so it can easily be related to a specific OnLine database server. If you have multiple instances of OnLine, each instance *must* have its own uniquely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

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

For more information, see the *INFORMIX-OnLine Dynamic Server Administrator's Guide*.

OPTCOMPIND

You can set the **OPTCOMPIND** environment variable to indicate the preferred join method, thus assisting the optimizer in selecting the appropriate join method. The **OPTCOMPIND** environment variable applies only to the INFORMIX-OnLine Dynamic Server.



0 The default. It indicates that 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).

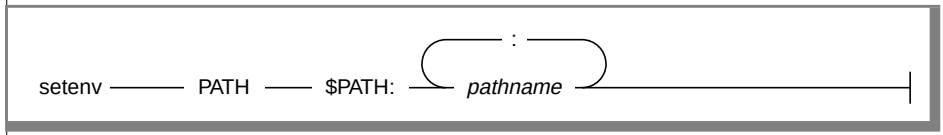
2 Nested-loop joins are not necessarily preferred. The optimizer bases it decision purely on costs regardless of transaction isolation mode.

When the **OPTCOMPIND** environment variable is not set, OnLine 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 0 (zero).

For more information on the ONCONFIG configuration parameter OPTCOMPIND, see the *INFORMIX-OnLine Dynamic Server Administrator's 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.



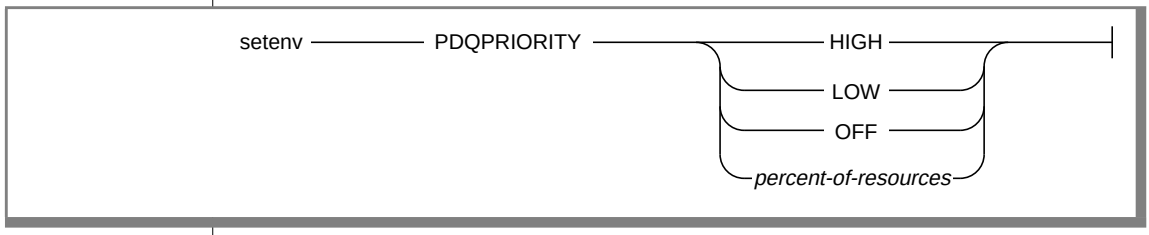
pathname specifies the search path for the executables.

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 4-9.

PDQPRIORITY

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



HIGH	When OnLine 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.
percent-of-resources	Represents an integer between 0 and 100. This number indicates a query priority level. The higher the number, the more resources OnLine uses. Three values have special meaning, described in the following list: 0 is equivalent to the symbolic value of OFF 1 is equivalent to the symbolic value of LOW 100 is equivalent to the symbolic value of HIGH

The **PDQPRIORITY** environment variable serves the same purpose as the ONCONFIG configuration parameter PDQPRIORITY, but it supersedes the configuration parameter. While the ONCONFIG configuration parameter PDQPRIORITY controls the parallelism of all applications, the **PDQPRIORITY** environment variable controls the parallelism of individual applications independently.

Note that although usually the more resources OnLine uses, the better the performance for a given query; using too many resources can cause contention among the resources and also take away resources from other queries, resulting in degraded performance.

Set the **PDQPRIORITY** environment variable to specify high priority by entering the following command:

```
setenv PDQPRIORITY HIGH
```

When the **PDQPRIORITY** environment variable is not set, OnLine uses the value specified for the ONCONFIG configuration parameter PDQPRIORITY. When neither the environment variable nor the configuration parameter is set, the default value is OFF.

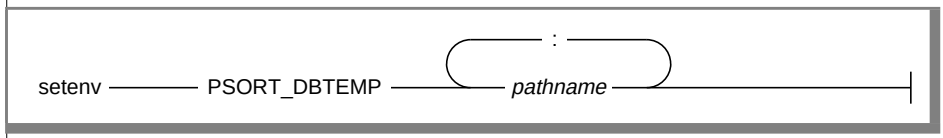
An application can override the setting of both the environment variable or the configuration parameter when it issues the SQL statement SET PDQPRIORITY, which is described in the *Informix Guide to SQL: Syntax*.

For more information on the ONCONFIG configuration parameter PDQPRIORITY, see the *INFORMIX-OnLine Dynamic Server Administrator's Guide*.

PSORT_DBTEMP

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

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



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

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

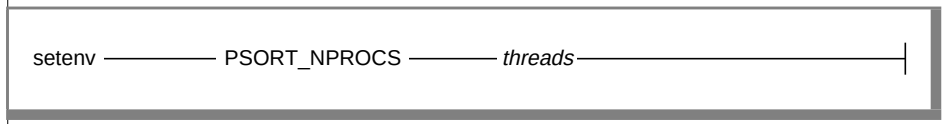
```
setenv PSORT_DBTEMP /usr/leif/tempsort
```

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 4-33.

PSORT_NPROCS

The **PSORT_NPROCS** environment variable enables the INFORMIX-OnLine Dynamic Server to improve the performance of the parallel-process sorting package by allocating more threads for sorting.



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. If **PSORT_NPROCS** is not set or set to zero, OnLine uses three threads.

You can disable parallel sorting by entering the following command:

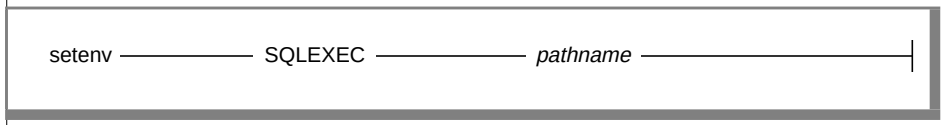
```
unsetenv PSORT_NPROCS
```

For additional information about the **PSORT_NPROCS** environment variable, see *INFORMIX-OnLine Administrator's Guide*.

SQLEXEC

Important: *This environment variable functions differently in Version 5.0 and Version 6.0 and later Informix products. For details of Version 5.0 functionality, refer to Chapter 4 of the December 1991 release of this manual.*

The **SQLEXEC** environment variable specifies the location of the Version 6.0 or later relay module executable that allows a Version 5.0 or earlier client to communicate with a local Version 6.0 or later INFORMIX-OnLine Dynamic Server or INFORMIX-SE database server. You must, therefore, set **SQLEXEC** only if you want to establish communication between a Version 5.0 or earlier client and a Version 6.0 or later database server.



pathname specifies the pathname for the relay module.

Set **SQLEXEC** to specify the full pathname of the relay module, which is in the **lib** subdirectory of your **$INFORMIXDIR** directory by entering the following command:

```
setenv SQLEXEC $INFORMIXDIR/lib/sqlrm
```

If you set the **SQLEXEC** environment variable on the C shell command line, you must include curly braces around the existing **INFORMIXDIR**, as shown in the following command:

```
setenv SQLEXEC ${INFORMIXDIR}/lib/sqlrm
```

For information on the relay module, see the *INFORMIX-OnLine Dynamic Server Administrator's Guide* or the *INFORMIX-SE Administrator's Guide*.

SQLRM

Important: *This environment variable functions differently in Version 5.0 and Version 6.0 or later Informix products. For details of Version 5.0 functionality, refer to Chapter 4 of the December 1991 release of this manual.*

In Version 6.0 and later, if the system administrator is configuring a client / server environment in which a Version 5.0 SQL API client accesses a local Version 6.0 or later database server, the **SQLRM** environment variable must be *unset* before **SQLEXEC** can be used to spawn a Version 6.0 or later relay module.

You can unset **SQLRM** by entering the following command:

```
unsetenv SQLRM
```

For information on the relay module, see the *INFORMIX-OnLine Dynamic Server Administrator's Guide* or the *INFORMIX-SE Administrator's Guide*.

SQLRMDIR

Important: *This environment variable functions differently in Version 5.0 and Version 6.0 and later Informix products. For details of Version 5.0 functionality, refer to Chapter 4 of the December 1991 release of this manual.*

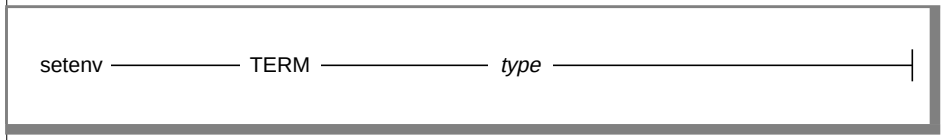
In Version 6.0 and later, if the database administrator is configuring a client / server environment in which a Version 5.0 SQL API client accesses a local Version 6.0 or later database server, the **SQLRM** environment variable must be *unset*.

Unset **SQLRMDIR** by entering the following command:

```
unsetenv SQLRMDIR
```

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.



type specifies the terminal type.

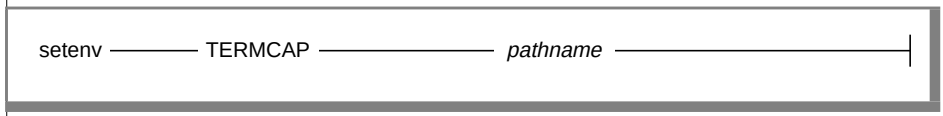
The terminal type specified in the **TERM** setting must correspond to an entry in the **termcap** file or **terminfo** directory. Before you can set the **TERM** environment variable, you must obtain the code for your terminal from the INFORMIX-OnLine Dynamic Server or INFORMIX-SE administrator.

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.



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

The **termcap** file contains a list of various types of terminals and their characteristics. Set TERMCAP by entering the following command:

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

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

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.

TERMINFO tells DB-Access to communicate with the **terminfo** directory instead of the **termcap** file. The **terminfo** directory has subdirectories that contain files pertaining 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**.

Index of Environment Variables

The table in Figure 4-3 provides an overview or cross-reference of the uses for the various Informix, NLS, and UNIX environment variables supported in Version 7.1. It serves as an index to general topics, the related environment variables, and the page where the environment variable is introduced.

Figure 4-3
Environment variables used with Informix products

Topic	Environment Variables	Page
ANSI compliance	DBANSIWARN	4-18
BLOB buffer	DBBLOBBUF	4-20
C COMPILER	INFORMIXC	4-40
Client / server	INFORMIXSERVER	4-46
	INFORMIXSHMBASE	4-47
	INFORMIXSTACKSIZE	4-48
	SQLEXEC	4-64
	SQLRM	4-65
	SQLRMDIR	4-65
COBOL compiler	INFORMIXCOB	4-40
	INFORMIXCOBDIR	4-41
	INFORMIXCOBSTORE	4-42
	INFORMIXCOBTYPE	4-43
Collation (NLS)	COLLCHAR	4-16
	LC_COLLATE	4-51

(1 of 7)

(2 of 7)

(3 of 7)

(4 of 7)

Topic	Environment Variables	Page
Pathname: for COBOL run times	INFORMIXCOBDIR	4-41
Pathname: for database files	DBPATH	4-28
Pathname: for executable programs	PATH	4-60
Pathname: for installation	INFORMIXDIR	4-46
Pathname: for message files	DBLANG	4-24
Pathname: for parallel sorting	PSORT_DBTEMP	4-62
Pathname: for relay module	SQLEXEC	4-64
Pathname: for remote shell	DBREMOTECMD	4-32
Pathname: for temporary files (SE)	DBTEMP	4-34
Printing	DBPRINT	4-31
Program: COBOL compiler	INFORMIXCOB	4-40
Program: printing	DBPRINT	4-31
Relay module	SQLEXEC	4-64
	SQLRM	4-65
	SQLRMDIR	4-65
Remote shell	DBREMOTECMD	4-32
Routine: DATETIME formatting	DBTIME	4-34
SE: temporary files	DBTEMP	4-34
Server	See *Database server*.	
Shared memory	INFORMIXSHMBASE	4-47
Shell: remote	DBREMOTECMD	4-32
Shell: search path	PATH	4-60
Sorting	PSORT_DBTEMP	4-62
	PSORT_NPROCS	4-63

(6 of 7)

Chapter

5

SQL Utilities

This chapter contains the syntax and usage of the following utilities which can be used with INFORMIX-OnLine Dynamic Server and INFORMIX-SE:

- The **chkenv** utility checks the validity of common or private environment-configuration files.
- The **crtcmap** utility creates mapping files between nonstandard and standard code sets for use with NLS.
- The **dbexport** utility unloads a database into ASCII files for later import into another database environment.
- The **dbimport** utility creates and populates a database from ASCII files.
- The **dbload** utility loads ASCII data into databases or tables created with Informix products.
- The **dbschema** utility creates a file containing the SQL statements necessary to replicate a specified table, view, or database. It also shows the distributions created by UPDATE STATISTICS.

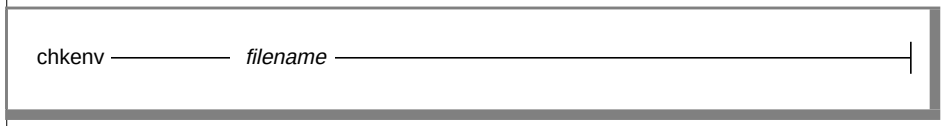
Before you use any of these utilities, you must set your **PATH**, **INFORMIXDIR**, and **INFORMIXSERVER** environment variables.

Tip: *If you are using INFORMIX-SE, Informix recommends that you not use the* ***.dbs*** *directory as your current directory when using a database-related utility. This keeps the* ***.dbs*** *directory free from file clutter and prevents multiple users from overwriting files that belong to other users.*

For information about system administration utilities, see the *INFORMIX-OnLine Dynamic Server Administrator's Guide* or the *INFORMIX-SE Administrator's Guide*.

The *chkenv* Utility

The **chkenv** utility checks the validity of common (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.



filename specifies the name of the environment-configuration file that you want to debug.

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

Issue the following command to perform a sanity check on the contents of the shared environment-configuration file:

```
chkenv informix.rc
```

The **chkenv** utility returns an error message if a bad environment variable entry is 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.

If the **chkenv** utility returns the following error,

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

Informix products ignore all lines in the environment-configuration file, starting at the point of the error.

You can also set the **ENVIGNORE** environment variable if you want the product to ignore specified environment variable settings in the file.

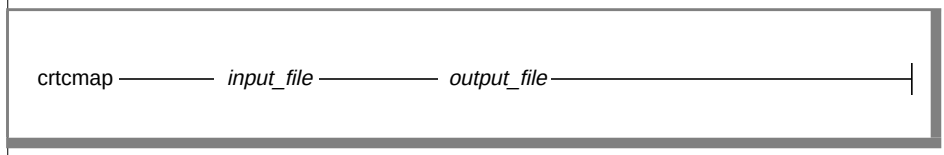
For a discussion of the use and format of environment-configuration files and the **ENVIGNORE** environment variable, see Chapter 4, "Environment Variables."

The *crtcmap* Utility

The **crtcmap** utility helps create mapping files between nonstandard and standard code sets for use with NLS. The mapping of character code sets to a standard form lets SQL APIs on different platforms, including distributed queries, access and share data by using a common character set representation of data in the database. You probably need this utility only if you use non-ISO-standard character code sets or NLS functionality.

When you set the **DBAPICODE** environment variable, you can specify the output files you create with the **crtcmap** utility. Your SQL APIs then communicate with the database server using the standard ASCII code set but interface with the keyboard, terminal display, and program character strings using the code set in the file specified in **DBAPICODE**, which was created with the **crtcmap** utility.

Creating Mapping Files for NLS



input_file specifies the name of the text file, in ASCII or the equivalent, where nonstandard characters are mapped to standard characters. Both the mapped and target character sets are represented in the file by hexadecimal values.

output_file specifies the name of the binary file where the output from the file mapping is stored. The **crtcmap** utility automatically prepends **m** (for "mapping") to the name of the Informix output file.

The **crtcmap** utility reads the text file specified in *input_file* and produces a binary *output_file* of mapped values for a code set. You must put the *output_file* in the correct **$INFORMIXDIR/msg** directory, as specified by the setting of the **LANG** or the **DBLANG** environment variable. See "Mapping-File Formats" on page 5-7 for information on creating an *input_file*.

The following sample command specifies the **fr_FR646.src** *input_file* and the **fr_FR646** *output_file*:

```
crtcmap fr_FR646.src fr_FR646
```

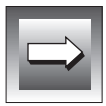
The **crtcmap** utility names the resulting binary output file **mfr_FR646**. The file name indicates the ISO 646 code set for the French language as used in France.

After you create and correctly place your output files in a message directory, you can set the **DBAPICODE** environment variable to specify the name of the mapping file you want to use to work with SQL or to access a database when NLS has been enabled in your Version 6.0 Informix product. To specify the mapping file created in the preceding command example, you set **DBAPICODE** as shown in the following example:

```
setenv DBAPICODE mfr_FR.646
```

Change the **DBAPICODE** setting as needed to refer to the different mapping file that you created with **crtcmap**.

The **DBAPICODE**, **LANG**, and **DBLANG** environment variables are discussed in Chapter 4, "Environment Variables."



Important: *Code sets, valid NLS environment variable settings, and NLS functionality might differ from system to system. You should use any existing standard naming conventions for your system whenever possible. Examples shown in this discussion are intended as guidelines but do not imply specific naming or mapping conventions. Therefore, examples of settings and files might not be supported by individual systems or hardware.*

Mapping-File Formats

The text file for the character-mapping table can consist of any number of lines. A line can be a comment or a one-to-one map of a nonstandard character code to the equivalent character in the target code set.

- A comment line starts with the # symbol and ends with a new-line character.
- A one-to-one map has the format (*hex_value*, *hex_value*) where *hex_value* is the hexadecimal representation of a character code.

  The *hex_value* begins with `0x` and is limited to a single byte. The first *hex_value* in a line represents the nonstandard character code; the second represents the desired code set that is eventually stored in the database.

The input text file should be written in ASCII or a code set that has the equivalent representation of ASCII for the significant characters and the new-line character. Significant characters include #, (,), x, a, b, c, d, e, f, A, B, C, D, E, F, the comma, and numeric characters.

The following two examples of input files cause errors due to inconsistency and ambiguity. In the first example, the character represented by `0x21` is mapped to both target characters `0x12` and `0x13`:

```
# Example input file 1
# (produces an error)
#
(0x20,0x30)
(0x21,0x12)
(0x21,0x13)
...
```

In the second example, the two characters represented by `0x21` and `0x22` are mapped to the same target character, `0x12`:

```
# Example input file 2
# (produces a warning)
#
(0x20,0x30)
(0x21,0x12)
(0x22,0x12)
...
```

The following sample input file shows an example of correct one-to-one mapping. Each character value is mapped to a different target character:

```
# Example input file 3
# (correctly indicates one-to-one mapping)
#
(0x20,0x30)
(0x21,0x12)
(0x33,0x18)
...
```

Error Messages and Warnings

When more than one character is mapped to the same target character, reverse mapping selects an arbitrary code for that set of characters. The results might still be inconsistent, and you are warned of duplication.

Incorrect mapping might result in duplication and ambiguity. After a database statement is executed and the results are returned, it warns you of potential unexpected behavior.

When an error message or warning appears, the **crtcmap** utility returns messages such as the following examples to standard output:

```
Error: Cannot create mapping file 'name'
Error: Syntax error on line number
Warning: Duplicate mapping entry on line number
```

Error messages are also returned to the message file.

The *dbexport* Utility

The **dbexport** utility unloads a database into ASCII files. The **dbexport** utility also creates an ASCII schema file which can be used by **dbimport** to re-create the database schema in another Informix environment. You can edit the schema file to modify the database created by **dbimport**.

The **dbexport** utility supports the following three destination options:

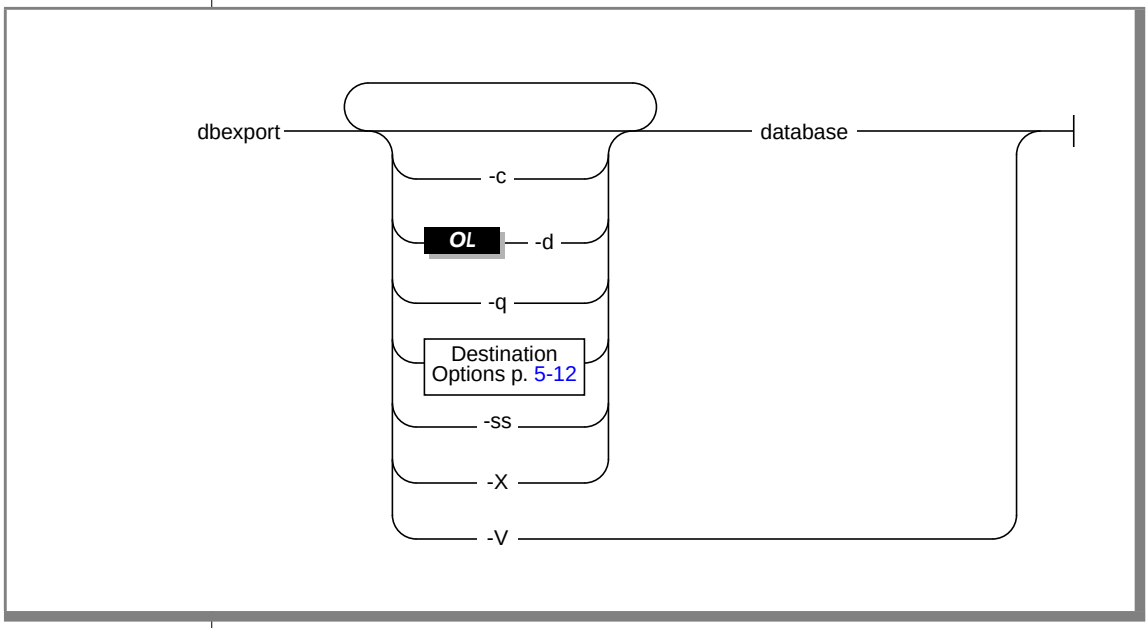
- Unloading a database and its schema file to disk
- Unloading a database and its schema file to tape
- Unloading the schema file to disk, where it can be examined and modified, and unloading the data files to tape

You must have DBA privilege or log in as user **informix** to export a database.

When the NLS environment variables are set correctly, as described in Chapter 4, "Environment Variables," **dbexport** can handle foreign characters in data and export the data from NLS databases. Refer to "Using dbexport with NLS" on page 5-14.

You can use delimited identifiers with the **dbexport** utility. The utility detects database objects that are keywords, mixed case, or have special characters and places double quotes around them.

Unloading a Database and Its Schema File



-c makes **dbexport** complete exporting unless a fatal error occurs.

-d makes **dbexport** export blob descriptors only, not blob data. (Refer to the *INFORMIX-OnLine/Optical User Manual*.)

database specifies the name of the database you want to export. If you want to use more than the simple name of the database, refer to the Database Name segment in Chapter 1 of the *Informix Guide to SQL: Syntax*.

-q suppresses the display (to the standard output) of error messages, warnings, and generated SQL data-definition statements.

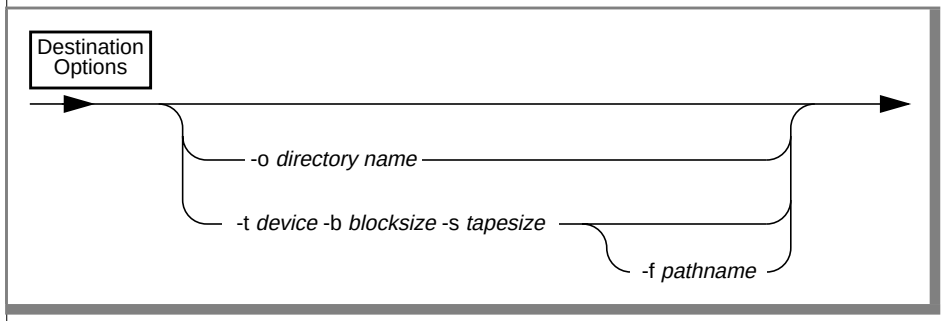
-ss	generates database server-specific information for all tables in the specified database. For INFORMIX-OnLine Dynamic Server databases, the **-ss** option specifies initial and next extent sizes, fragmentation information if the table is fragmented, the locking mode, the dbspace for a table, and the blobspace for any blobs. For INFORMIX-SE databases, the **-ss** option generates the pathname where each table was created if the table is in a path other than the database directory.
-X	recognizes HEX binary data in character fields.
-V	displays product version information.

In addition to the data files and the schema file, **dbexport** creates a file of messages in the current directory called **dbexport.out**. This file contains any error messages and warnings as well as a display of the SQL data definition statements it generates. The same material is also written to the standard output unless you specify the **-q** option.

During the export, the database is locked in exclusive mode. If an exclusive lock cannot be obtained, the program ends and displays a diagnostic message.

You can cancel **dbexport** at any time by pressing the Interrupt key. The **dbexport** utility asks for confirmation before terminating.

Destination Options



-b *blocksize*	specifies, in kilobytes, the block size of the tape device.
-f *pathname*	specifies the pathname on disk where you want the schema file stored, if you are storing the data files on tape. The **-f** pathname can be an complete pathname or simply a file name. If only a file name is given, the file is stored in the currentdirectory.
-o *directory name*	names the directory on disk where you want both the data files and the schema file stored. The directory specified as *directory name* must already exist.
-s *tapesize*	specifies, in kilobytes, the amount of data that can be stored on the tape. The tape size is limited to 2,097,151 kilobytes. The limit is required because of the way **dbexport** and **dbimport** track their positions into the tape.
-t *device*	names the pathname of the tape device where you want the ASCII data files and, possibly, the schema file stored. The **-t** option does *not* allow you to specify a remote tape device.

When you write to disk, **dbexport** creates a subdirectory, *database***.exp**, in the directory specified by the **-o** option. The **dbexport** utility creates a file, *tablename***.unl**, for each table in the database. The schema file is written to the file *database***.sql**. The **.unl** and **.sql** files are stored in the *database***.exp** directory.

If you do not specify a destination for the data and schema files, the subdirectory *database***.exp** is placed in the current working directory.

When you write the data files to tape, you can use the **-f** option to store the schema file to disk. You are not required to name the schema file *database***.sql**. You can give it any name.

The following command creates a **reports.exp** subdirectory in the current directory. It then unloads the **reports** database in the **turku** directory on the SE database server called **finland** and places the resulting files in the **reports.exp** directory.

```
dbexport //finland/turku/reports
```

The following command exports the OnLine database **stores7** to tape with a block size of 16 kilobytes and a tape capacity of 24,000 kilobytes. The schema file is written to **/tmp/stores7.imp**.

```
dbexport -t /dev/rmt0 -b 16 -s 24000 -f /tmp/stores7.imp  stores7
```

The following command exports the same **stores7** database to the directory named **/work/exports/stores7.exp**. The resulting schema file is **/work/exports/stores7.exp/stores7.sql**.

```
dbexport -o /work/exports stores7
```

The Contents of the Schema File

The schema file contains the SQL statements needed to re-create the exported database. You can edit the schema file to add or modify information.

Important: *Do not put comments into the schema file. Comments cause unpredictable results when read by the* ***dbimport*** *utility.*

If you use the **-ss** option, the schema file contains all the server-specific information used to create the database and tables, such as initial and next extent sizes, fragmentation information if the table is fragmented, lock mode, the dbspace where each table resides, and the blobspace where each blob column resides. The following information is not retained:

- Logging mode of the database (see "Database Logging Mode" on page 5-17)
- The starting value of a SERIAL column

The statements in the schema file that create tables, views, and indexes and grant privileges do so using the name of the user who originally created the database. In this way, the original owner retains the DBA privileges for the database and is the owner of all the tables, indexes, and views. In addition, whoever executes the **dbimport** command also has the DBA privileges for the database.

Using *dbexport* with NLS

You can use **dbexport** to unload ASCII data from a non-NLS database, set NLS environment variables, and then use **dbimport** to turn it into an NLS database. You also can use **dbexport** and then **dbimport** to change the locale of an NLS database. See the discussion of using **dbimport** with an NLS database on "Using dbimport with NLS" on page 5-22.

When you run **dbexport** on an NLS database, you must set the **DBNLS** and **LANG** environment variables to match the settings of the database. Use one of the following methods:

- You can check the current **LC_COLLATE** and **LC_CTYPE** settings, which represent the locale and take precedence over the **LANG** setting, by issuing the following statement on the INFORMIX-OnLine Dynamic Server:

```
SELECT tabname, site FROM systables WHERE tabid IN (90,91)
```

- You can also check the **LC_COLLATE** and **LC_CTYPE** settings by choosing the Nls option from the DATABASE INFO menu in DB-Access.
- You can use the SMI database to check the NLS status of all databases on your server with the following command:

```
SELECT name, is_nls FROM sysmaster:sysdatabases
```

For information about the SMI database refer to the *INFORMIX-OnLine Dynamic Server Administrator's Guide*.

The *dbimport* Utility

The **dbimport** utility creates a database and loads it with data from ASCII input files. The input files consist of a schema file that is used to re-create the database and data files that contain the database data. Normally, you generate the input files with the **dbexport** utility, but you can use any properly formatted input files.

The **dbimport** utility can use files from the following location options:

- All input files are located on disk.
- All input files are located on tape.
- The schema file is located on disk and the data files are located on tape.

Important: *Do not put comments into your input file. Comments cause unpredictable results when read by the* ***dbimport*** *utility.*

The **dbimport** utility supports the following options for the new database:

For INFORMIX-OnLine Dynamic Server:

- Create an ANSI-compliant database (includes unbuffered logging).
- Establish transaction logging for a database (unbuffered or buffered logging).
- Specify the dbspace where the database will reside.

For INFORMIX-SE:

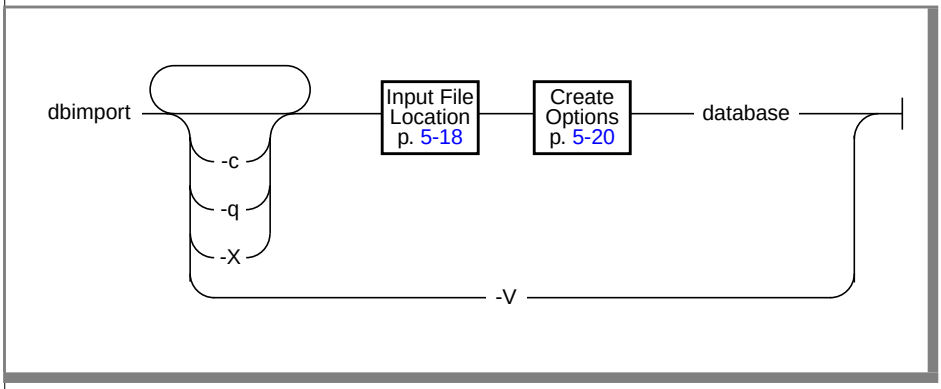
- Create an ANSI-compliant database (ANSI-compliant logging).
- Establish transaction logging for a database (unbuffered logging).

The user who runs **dbimport** is granted the DBA privilege on the newly created database. The **dbimport** process locks each table as it is being loaded and unlocks the table when the loading is completed.

When the NLS environment variables are set correctly, as described in Chapter 4, "Environment Variables," **dbimport** can import the data into NLS databases.

You can use delimited identifiers with the **dbimport** utility. The utility detects database objects that are keywords, mixed case, or have special characters and places double quotes around them.

Building a Database from ASCII Files Created by *dbexport*



-c	instructs **dbimport** to complete importing even though it encounters certain non-fatal errors.
database	specifies the name of the database you intend to create. If you want to use more than the simple name of the database, refer to the Database Name segment in Chapter 1 of the *Informix Guide to SQL: Syntax*.
-q	suppresses display (to the standard output) of error messages and warnings.
-X	recognizes HEX binary data in character fields.
-V	displays product version information.

If you specify **-c**, **dbimport** ignores the following errors:

- A data row that contains too many columns
- Inability to put a lock on a table
- Inability to release a lock

Even if you use the **-c** option, **dbimport** interrupts processing if one of the following fatal errors occurs:

- Unable to open the tape device specified
- Bad writes to the tape or disk
- Invalid command parameters
- Cannot open database or no system permission

The dbimport utility creates a file of messages called **dbimport.out** in the current directory. This file contains any error messages and warnings related to **dbimport** processing. The same material is also written to the standard output unless you specify the **-q** option.

You can cancel **dbimport** at any time by pressing the Interrupt key. The **dbimport** program asks for confirmation before terminating.

The user who runs **dbimport** is granted the DBA privilege on the newly created database.

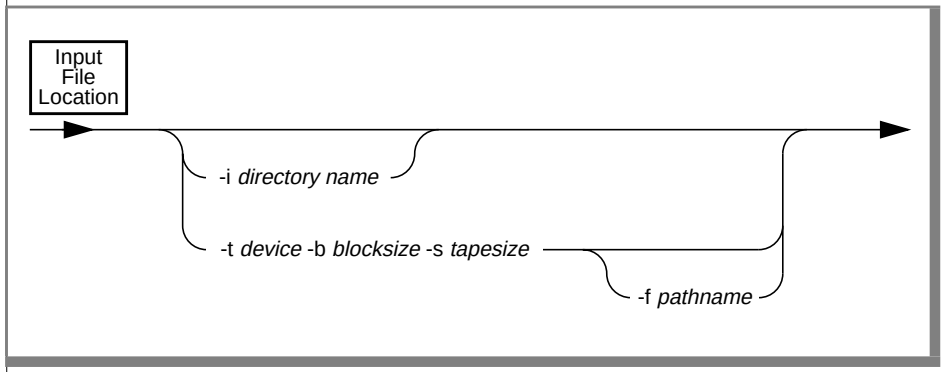
Database Logging Mode

The logging mode is not retained in the schema file. You can specify any of the following options when you import the database using **dbimport**:

- ANSI-compliant database with unbuffered logging
- Unbuffered logging
- Buffered logging

Refer to "Create Options" on page 5-20.

Input File Location Options



-b specifies, in kilobytes, the block size of the tape device. If you are importing from tape, you must use the same block size that you used to export the database.

-f *pathname* specifies the pathname on disk where **dbimport** can find the schema file to use as input to create the database when the data files are read from tape. If you use **-f**, you typically use the same command filename that you specified in the **dbexport** command. If you specify only a filename, **dbimport** looks for the file in the **.exp** subdirectory of your current directory. If you used the **-f** option when you exported the database, you usually use the **-f** option when you import the database (you can always change the name of the schema file).

-i *directory name* names the complete pathname of the directory on disk where **dbimport** can find the *database***.exp** directory that contains the input data files and schema file. This directory should be the same directory that you specified with the **dbexport -o** option. If you omit the **-i** option, the **dbimport** utility looks for the *database***.exp** directory under the current directory.

-s *tapesize* specifies, in kilobytes, the amount of data that can be stored on the tape. If you are importing from tape, you must use the same tape size that you used to export the database.

-t *device* specifies the pathname of the tape device where the tape containing the input files is mounted. The **-t** option does *not* allow you to specify a remote tape device.

If you do not specify an input file location, **dbimport** looks for data files in the directory *database***.exp** under the current directory and the schema file in *database***.exp/***database***.sql**.

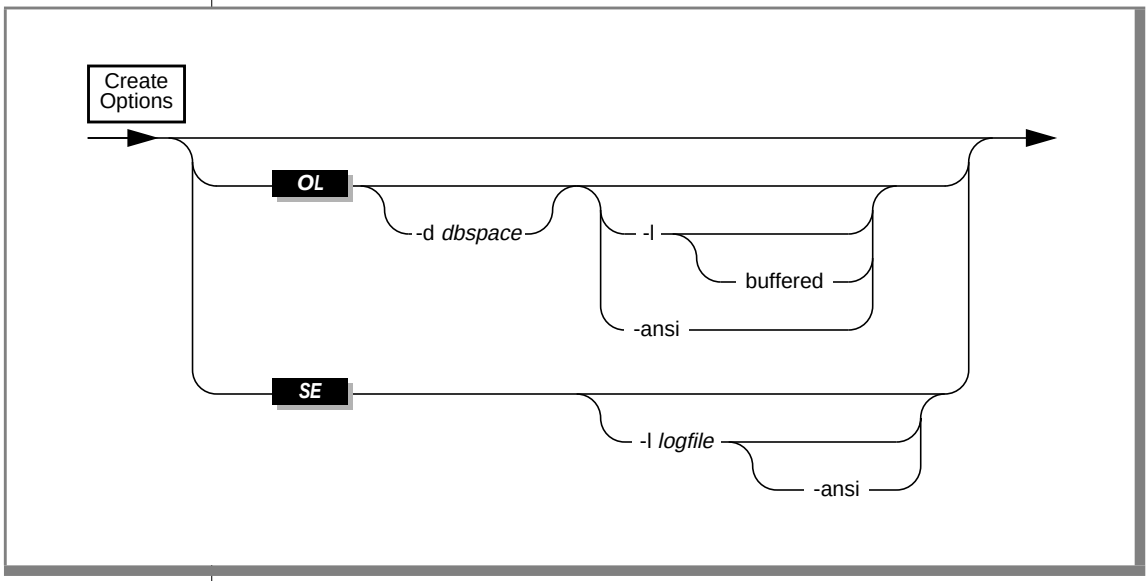
The following command imports the **stores7** database from a tape with a block size of 16 kilobytes and a capacity of 24,000 kilobytes. The schema file is read from **/tmp/stores7.imp**.

```
dbimport -c  -t /dev/rmt0 -b 16 -s 24000 -f /tmp/stores7.imp stores7
```

The following command imports the **stores7** database from the **stores7.exp** directory under the **/work/exports** directory. The schema file is assumed to be **/work/exports/stores7.exp/stores7.sql**.

```
dbimport -c -i /usr/informix/port stores7
```

Create Options



-ansi creates an ANSI-compliant database in which the ANSI rules for transaction logging are enabled. For more information about ANSI-compliant databases, see Chapter 1, "Informix Databases."

-d *dbspace* names the INFORMIX-OnLine Dynamic Server dbspace where the database is created. If no dbspace is specified, the database is created in the rootdbs.

-l establishes unbuffered transaction logging for the imported database. The **-l** option is equivalent to the WITH LOG IN clause of the CREATE DATABASE statement. For more information about the CREATE DATABASE statement, see Chapter 1 in the *Informix Guide to SQL: Syntax*.

-l buffered	establishes buffered transaction logging for the imported database.
-l *logfile*	establishes transaction logging for the imported database and specifies the name of the transaction-log file. The *logfile* filename must be an absolute pathname. The **-l** *logfile* option is equivalent to the WITH LOG IN clause of the CREATE DATABASE statement. For more information about the CREATE DATABASE statement, see Chapter 1 in the *Informix Guide to SQL: Syntax*.

For INFORMIX-OnLine Dynamic Server, if you do not specify a dbspace name, the **dbimport** utility creates the database in the root dbspace by default.

For INFORMIX-SE, the **dbimport** utility always creates the database in the current directory.

The following command imports the **stores7** database from the **/usr/informix/port/stores7.exp** directory to the current directory. The new database is ANSI-compliant and the transaction-log file is specified as **stores7.log** in **/usr/work**.

```
dbimport -c stores7 -i /usr/informix/port -l /usr/work/stores7.log -ansi
```

Changing the Database Name

The **dbimport** utility assumes that the new database has the same name as the database which you exported. If you export a database to tape, you cannot change its name when you import it with **dbimport**.

If you export a database to disk, you can change the database name. For the purpose of the following example, assume that **dbexport** unloaded the database **stores7** into the to the directory **/work/exports/stores7.exp**. Thus, the data files (the **.unl** files) are stored in **/work/exports/stores7.exp** and the schema file is **/work/exports/stores7.exp/stores7.sql**.

To change the database name to **newname**, perform the following steps:

1. Change the name of the **.exp** directory. That is, change **/work/exports/stores7.exp** to **/work/exports/newname.exp**.
2. Change the name of the schema file. That is, change **/work/exports/stores7.exp/stores7.sql** to **/work/exports/stores7.exp/newname.sql**. Do not change the names of the **.unl** files.
3. Import the database with the following command:

```
dbimport -i /work/exports newname
```

Using *dbimport* with NLS

You can use **dbimport** to create and populate an NLS database. Use the **dbexport** utility to unload ASCII data from a non-NLS database. Set the appropriate NLS environment variables and then use **dbimport** to change the non-NLS database into an NLS database. See the discussion of **dbexport** and NLS on "Using dbexport with NLS" on page 5-14.

Changing a non-NLS Database to an NLS Database

To change a non-NLS database to an NLS database, perform the following steps:

1. Run **dbexport** on the non-NLS database, as described in this chapter.
2. Set the **DBNLS** environment variable to 2.
3. Set the **LANG** environment variable to the desired locale for the NLS database.
4. Set the **COLLCHAR** environment variable to 1.
5. Run **dbimport** for the NLS database, as described in this chapter.

All table columns of the CHAR data type in the non-NLS database are converted to NCHAR, and VARCHAR columns are converted to NVARCHAR, in the NLS database. If you want the NCHAR (or NVARCHAR) data type in a CREATE PROCEDURE statement, however, you must set the **COLLCHAR** environment variable when the procedure is executed.

You can also use **dbimport** to change the locale of a database. For example, use **dbexport** to unload the schema as well as the data from a French database and set the appropriate NLS environment variables for a German database. You then use **dbimport** to create a German database and load the exported French data.

Changing the Locale of an NLA Database

To change the locale of an NLS database, perform the following steps:

1. Set the **DBNLS** environment variable to 2.

   Set the **LANG** environment variable to the current locale for the database.
2. Run **dbexport** on the original database, as described in this chapter.
3. Set **LANG** to the new locale.
4. Run **dbimport** for the new database, as described in this chapter.

If you run **dbexport** on an NLS database and then try to convert it to a non-NLS database with **dbimport**, you will probably encounter an error such as `illegal character`.

The *dbload* Utility

The **dbload** utility transfers data from one or more ASCII or NLS files into one or more existing tables. The **dbload** utility offers the following advantages over using the LOAD statement:

- You can use **dbload** to load data from input files that have been created with a variety of format arrangements. The **dbload** command file can accommodate data from entirely different database management systems.
- You can specify a starting point in the load by directing **dbload** to read but ignore *x* number of rows.

- You can specify a batch size so that after every *x* number of rows are inserted, the insert is committed.
- You can limit the number of bad rows read, beyond which **dbload** ends.

The cost of **dbload** flexibility is the time and effort spent creating the **dbload** command file, which is required for **dbload** operation. The ASCII input files are not specified as part of the **dbload** command line, and neither are the tables into which the data is inserted. This information is contained within the command file.

You can load blobs with the **dbload** utility as long as the blobs are in ASCII format.

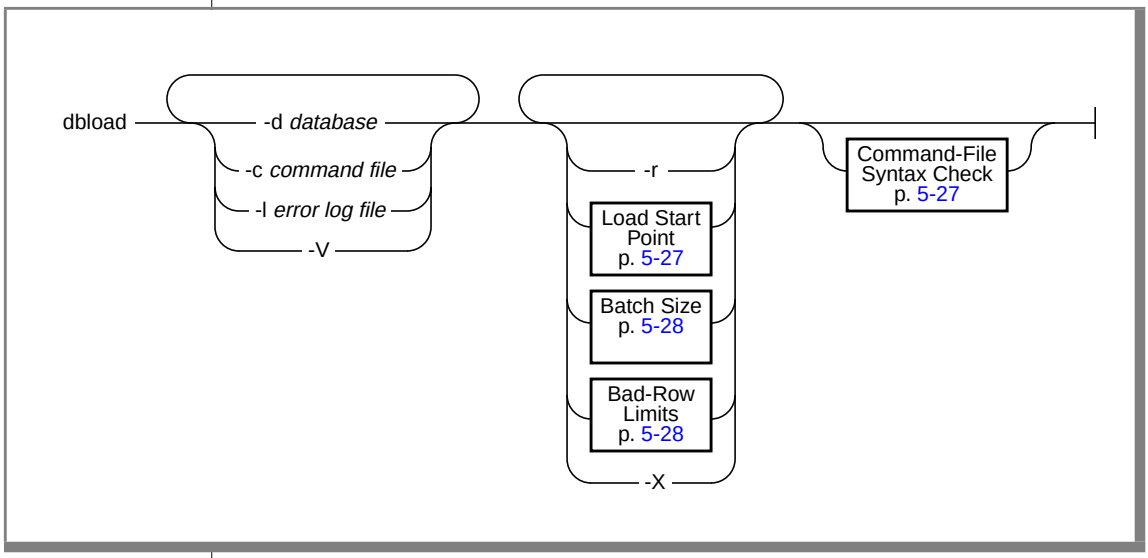
The presence of indexes greatly affects the speed with which the **dbload** utility loads data. For best performance, drop any indexes on the tables receiving the data before you run **dbload**. You can create new indexes after **dbload** has finished.

When the NLS environment variables are set correctly, as described in Chapter 4, "Environment Variables," **dbload** can load the data into NLS databases.

You can use delimited identifiers with the **dbload** utility. The utility detects database objects that are keywords, mixed case, or have special characters and places double quotes around them.

The **dbload** syntax and use information is described in the following sections. The **dbload** command file structure and instructions for creating a command file are found in "Creating a dbload Command File" on page 5-29.

Loading Data from a Command File into a Table



-c *command file* specifies the filename or pathname of a **dbload** command file.

-d *database* specifies the name of the database to receive the data. If you want to use more than the simple name of the database, refer to the Database Name segment in Chapter 1 of the *Informix Guide to SQL: Syntax*.

-l *error log file* specifies the filename or pathname of an error log file. The error log file specified by the **-l** flag stores any input file rows that **dbload** cannot insert into the database as well as diagnostic information. If you specify an existing file, its contents are overwritten. If you specify a file that does not exist, it is created.

-r	instructs **dbload** not to lock the tables during loading, enabling other users to update data in the table during the load. If you do not specify **-r**, the tables specified in the command file are locked during loading so that other users cannot update data in the table. Table locking reduces the number of locks needed during the load but reduces concurrency. If you are planning to load a large number of rows, use table locking and load during nonpeak hours.
-X	recognizes HEX binary data in character fields.
-V	displays product version information.

Tip: *If you specify part (but not all) of the required information,* ***dbload*** *prompts you for additional specifications. If you are missing all three options, you receive an error message.*

If you are on a network, you can specify a database on another database server by including the database server name and directory path with the database name.

The following command loads data into the **stores7** database in the **turku** directory on the **finland** database server (**finland** is an SE database server):

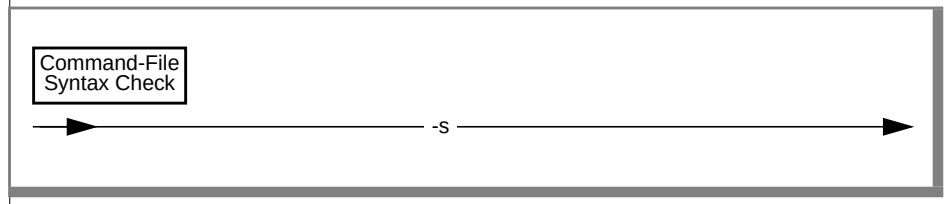
```
dbload -d //finland/turku/stores7 -c commands -l errlog
```

If your database supports transactions, **dbload** commits a transaction after every 100 rows are inserted. To modify this default value, specify a batch size. (See "Batch Size" on page 5-28.)

If your most recent **dbload** session ended prematurely, you can resume loading with the next record in the file by specifying the starting line number in the command-line syntax.

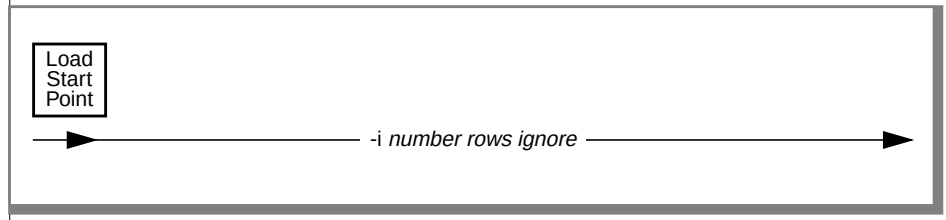
If you press the Interrupt key, **dbload** terminates and discards any new rows that were inserted but not yet committed to the database (if the database has transactions).

Command-File Syntax Check



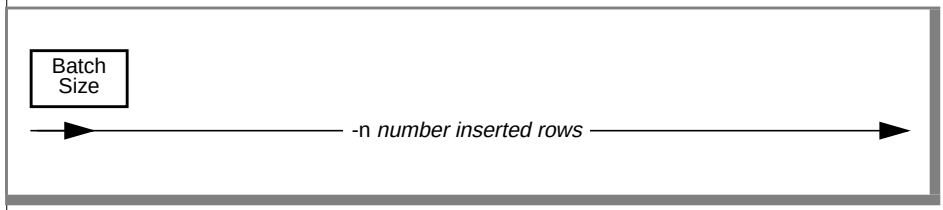
-s instructs **dbload** to check the syntax of the statements in the command file without inserting data. The **-s** option performs a syntax check on the FILE and INSERT statements in the specified **dbload** command file. The screen displays the command file with any errors marked where they are found.

Load Start Point



-i *number rows ignore* instructs **dbload** to ignore the specified number of new-line characters. The **-i** option instructs **dbload** to read and ignore the specified number of new-line characters in the input file before it begins processing. This option is useful if your most recent **dbload** session ended prematurely. For example, if **dbload** ends after inserting 240 lines of input, you can resume loading at line 241 by setting *number rows ignore* to 240. It is also useful if header information in the input file precedes the data records.

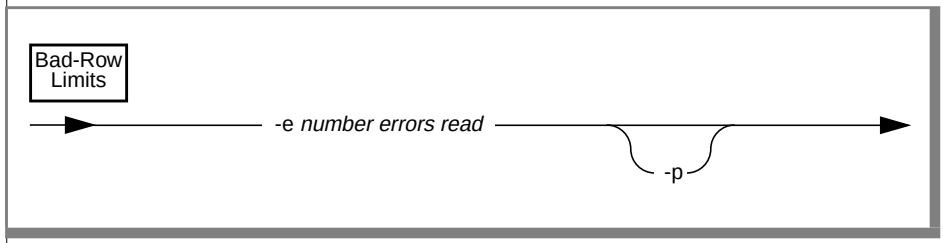
Batch Size



-n *number inserted rows*

instructs **dbload** to commit the transaction after the specified number of new rows are inserted. If your database supports transactions, **dbload** commits a transaction after every specified number of new rows is read and inserted. A message appears after each commit. (For information about transactions, see Chapter 4 of the *Informix Guide to SQL: Tutorial*.) If you do not specify the **-n** option, **dbload** commits a transaction after every 100 rows are inserted.

Bad-Row Limits



-e *number errors read*	specifies the number of bad rows that **dbload** reads before terminating. If you set **-e** *number errors read* to a positive integer, **dbload** terminates when it reads *number errors read* + 1 bad rows. If you set the value of *number errors read* to zero, **dbload** terminates when it reads the first bad row. If you do not specify a bad-row limit, the default is 10. This means that **dbload** terminates after it reads the eleventh bad row. In a database with transactions, unless you specify otherwise, **dbload** commits any new rows that were inserted since the last transaction.
-p	prompts for instructions if the number of bad rows exceeds the limit. If **dbload** exceeds the bad-row limit and the **-p** option is specified, **dbload** prompts you for instructions before it terminates. The prompt asks whether you want to roll back or to commit all rows inserted since the last transaction. If **dbload** exceeds the bad-row limit and the **-p** option is not specified, **dbload** commits all rows inserted since the last transaction.

Creating a *dbload* Command File

Before you use **dbload**, create an ASCII command file that names the input data files and the tables that receive the data. The command file maps fields from one or more input files into columns of one or more tables within your database.

The command file contains only FILE and INSERT statements. Each FILE statement names an input data file. The FILE statement also defines the data fields from the input file that are inserted into the table. Each INSERT statement names a table to receive the data. The INSERT statement also defines how **dbload** places the data described in the FILE statement into the table columns.

Within the command file, the FILE statement can appear in the following forms:

- Delimiter form
- Character-position form

The FILE statement has a size limit of 4,096 bytes.

Use the delimiter form of the FILE statement when every field in the input data row uses the same delimiter and every row ends with a new-line character. This format is typical of data rows with variable-length fields. You can also use the delimiter form of the FILE statement with fixed-length fields as long as the data rows meet the delimiter and new-line requirements. The delimiter form of the FILE and INSERT statements is easier to use than the character-position form.

Use the character-position form of the FILE statement when you cannot rely on delimiters and you need to identify the input data fields by character position within the input row. For example, use this form to indicate that the first input data field begins at character position 1 and continues until character position 20. You can also use this form if you must translate a character string into a null value. For example, if your input data file uses a sequence of blanks to indicate a null value, you must use this form if you want to instruct **dbload** to substitute null at every occurrence of the blank-character string.

You can use both forms of the FILE statement in a single command file. However, for clarity, the two forms are described separately in the following sections.

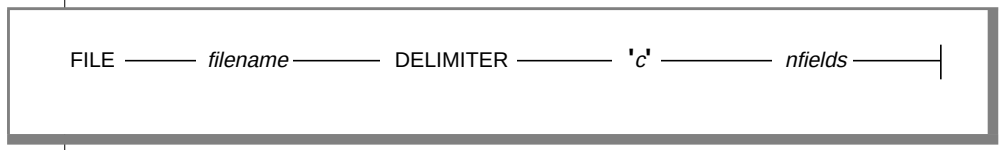
FILE and INSERT Statements: Delimiter Form

The following example of a **dbload** command file illustrates a simple delimiter form of the FILE and INSERT statements. The example is based on the **stores7** database. The three input data files, **stock.unl**, **customer.unl**, and **manufact.unl**, were created by an UNLOAD statement. (To see the **.unl** input data files, refer to the directory **$INFORMIXDIR/demo/***prod_name*.)

```
FILE stock.unl DELIMITER '|' 6;
INSERT INTO stock;
FILE customer.unl DELIMITER '|' 10;
INSERT INTO customer;
FILE manufact.unl DELIMITER '|' 3;
INSERT INTO manufact;
```

Syntax for the Delimiter Form

The following syntax shows an example of the delimiter FILE statement:



c	defines the field delimiter for the specific input file specified as *filename*.
filename	specifies the input file.
nfields	is an integer indicating the number of fields in each data row contained in *filename*.

If the delimiter specified by *c* appears as a literal character anywhere in the input file, it must be preceded with a backslash in the input file. For example, if the value of *c* is specified as `[`, you need to place a backslash before any literal `[` that appeared in the input file. Similarly, you must precede any backslash that appears in the input file with an additional backslash.

The **dbload** utility assigns the sequential names **f01**, **f02**, **f03**, and so on to fields in the input file. You cannot see these names, but if you refer to these fields to specify a value list in an associated INSERT statement, you must use the **f01**, **f02**, **f03** format. Refer to "How to Write a dbload Command File in Delimiter Form" on page 5-33 for details.

Two consecutive delimiters define a null field. As a precaution, you can place a delimiter immediately before the new-line character that marks the end of each data row. If you omit this delimiter, an error results whenever the last field of a data row is empty. If you are certain that none of the input data rows ends with an empty field, you can omit this step.

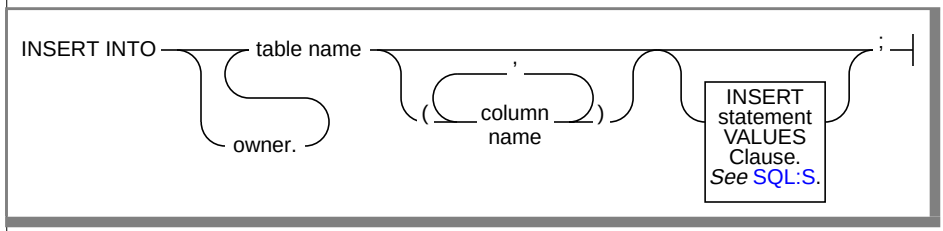
Inserted data types correspond to the explicit or default column list. If the data field width is different from its corresponding character column width, the data is made to fit. That is, inserted values are padded with blanks if the data is not wide enough for the column or truncated if the data is too wide for the column.

If the number of columns named is fewer than the number of columns in the table, **dbload** inserts the default value specified when the table was created for the unnamed columns. If no default value is specified, **dbload** attempts to insert a null value. If the attempt violates a NOT NULL restriction or a unique constraint, the insert fails and an error message is returned.

If the INSERT statement omits the column names, the default INSERT specification is every column in the named table. If the INSERT statement omits the VALUES clause, the default INSERT specification is every field of the previous FILE statement.

An error results if the number of column names listed (or implied by default) does not match the number of values listed (or implied by default).

The syntax of **dbload** INSERT statements resembles INSERT statements in SQL, except that in **dbload**, INSERT statements cannot incorporate SELECT statements. The following diagram shows the syntax of the **dbload** INSERT statement for delimiter form:



column name	specifies the column that receives the new data.
table name	specifies the table that receives the new data.
*owner***.**	is the user name of the owner of the table.

Users who execute **dbload** with this command file must have the Insert privilege on the named table.

How to Write a dbload Command File in Delimiter Form

The first FILE AND INSERT statement set in the delimiter example on page 5-30 is repeated in the following example:

```
FILE stock.unl DELIMITER '|' 6;
INSERT INTO stock;
```

The FILE statement describes the **stock.unl** data rows as composed of six fields, each separated by a vertical bar (|) as the delimiter. Compare the FILE statement with the data row shown in the following example, which appear in the input file **stock.unl**. (Because the last field is not followed by a delimiter, an error results if any data row ends with an empty field.)

```
1|SMT|baseball gloves|450.00|case|10 gloves/case
2|HRO|baseball|126.00|case|24/case
3|SHK|baseball bat|240.00|case|12/case
```

The example INSERT statement contains only the required elements. Because the column list is omitted, the INSERT statement implies that values are to be inserted into every field in the **stock** table. Because the VALUES clause is omitted, the INSERT statement implies that the input values for every field are defined in the most recent FILE statement. This INSERT statement is valid because the **stock** table contains six fields, which is the same number of values defined by the FILE statement. The first data row inserted into **stock** from this INSERT statement is shown in the following example:

	Column	Value
f01	stock_num	1
f02	manu_code	SMT
f03	description	baseball gloves
f04	unit_price	450.00
f05	unit	case
f06	unit_descr	10 gloves / case

The FILE and INSERT statement shown in the following example set illustrates a more complex INSERT statement syntax:

```
FILE stock.unl DELIMITER '|' 6;
INSERT INTO new_stock (col1, col2, col3, col5, col6)
    VALUES (f01, f03, f02, f05, 'autographed');
```

In this example, the VALUES clause uses the field names automatically assigned by **dbload**. You must reference the automatically assigned field names with the letter **f** followed by a number: **f01**, **f02**, **f10**, **f100**, **f999**, **f1000**, and so on. All other formats are incorrect.

Tip: *The first nine fields must include a zero: f01, f02, ..., f09.*

The user changed the column names, the order of the data, and the meaning of **col6** in the new **stock** table. Because the fourth column in **new_stock** (**col4**) is not named in the column list, the new data row contains a null in the **col4** position (assuming that the column permits nulls). If no default is specified for **col4**, the inserted value is null.

The first data row inserted into **new_stock** from this INSERT statement is shown in the following example:

Column	Value
col1	1
col2	baseball gloves
col3	SMT
col4	null
col5	case
col6	autographed

FILE and INSERT Statements: Character-Position Form

The examples in this section are based on an input data file, **cust_loc_data**, that contains the last four fields (**city**, **state**, **zipcode**, and **phone**) of the **customer** table. Fields in the input file are padded with blanks to create data rows in which the location of data fields and the number of characters are the same across all rows. The definitions for these fields are CHAR(15), CHAR(2), CHAR(5), and CHAR(12), respectively. For your reference, Figure 5-1 displays the character positions and five example data rows from the **cust_loc_data** file:

Figure 5-1
A sample data file

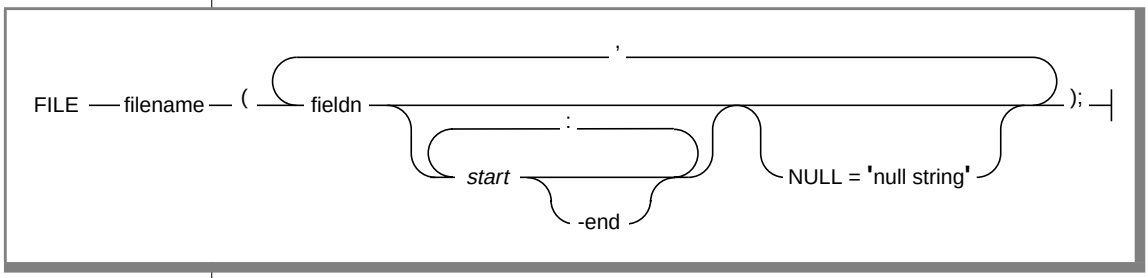
```
          1         2         3
12345678901234567890123456789012345

Sunnyvale      CA94086408-789-8075
Denver         CO80219303-936-7731
Blue Island    NY60406312-944-5691
Brighton       MA02135617-232-4159
Tempe          AZ85253xxx-xxx-xxxx
```

The following example of a **dbload** command file illustrates the character-position form of the FILE and INSERT statements. The example includes two new tables, **cust_address** and **cust_sort**, to receive the data. For the purpose of this example, **cust_address** contains four columns, the second of which is omitted from the column list. The **cust_sort** table contains two columns.

```
FILE cust_loc_data
    (city 1-15,
     state 16-17,
     area_cd 23-25 NULL = 'xxx',
     phone 23-34 NULL = 'xxx-xxx-xxxx',
     zip 18-22,
     state_area 16-17 : 23-25);
INSERT INTO cust_address (col1, col3, col4)
    VALUES (city, state, zip);
INSERT INTO cust_sort
    VALUES (area_cd, zip);
```

Syntax for the Character-Position Form

The character-position FILE statement is shown in the following diagram:



-end	is a hyphen followed by an integer that indicates the character position within a data row that ends a range of character positions.
fieldn	is a name that you assign to the data field that you are defining with the range of character positions.
filename	is the name of the input file.
null string	is a quoted string that specifies the data value for which **dbload** should substitute a null.
start	is an integer that indicates the character position within a data row that starts a range of character positions. If you use *start* without *end*, it represents a single character.

The same character position can be repeated in a data-field definition or in different fields.

The *null string* scope of reference is the data field for which you define it. You can define an explicit null string for each field that allows null entries.

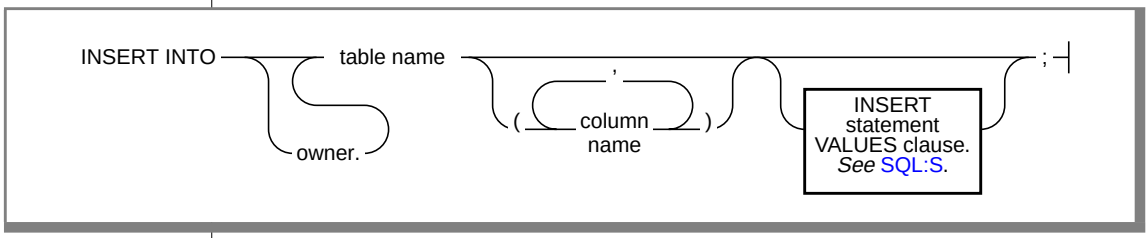
Inserted data types correspond to the explicit or default column list. If the data field width is different from its corresponding character column, inserted values are padded with blanks if the column is wider or are truncated if the field is wider.

If the number of columns named is fewer than the number of columns in the table, **dbload** inserts the default value specified for the unnamed columns. If no default value is specified, **dbload** attempts to insert a null value. If the attempt violates a NOT NULL restriction or a unique constraint, the insert fails and an error message is returned.

If the INSERT statement omits the column names, the default INSERT specification is every column in the named table. If the INSERT statement omits the VALUES clause, the default INSERT specification is every field of the previous FILE statement.

An error results if the number of column names listed (or implied by default) does not match the number of values listed (or implied by default).

The syntax of **dbload** INSERT statements resembles INSERT statements in SQL, except that in **dbload**, INSERT statements cannot incorporate SELECT statements. The following diagram shows the syntax of the **dbload** INSERT statement for character-position form:



column name	is the column that receives the new data.
table name	is the table that receives the new data.
*owner***.**	is the user name of the table owner.

The syntax for character-position form is identical to the syntax for delimiter form.

The user who executes **dbload** with this command file must have the Insert privilege on the named table. In INFORMIX-SE, the **dbload** utility recognizes valid SE table references, including owner designations. That is, the table name can be preceded by owner name but not by database server name or database name. Valid table-name syntax is defined in detail in the *Informix Guide to SQL: Syntax*.

How to Write a dbload Command File in Character-Position Form

The first FILE AND INSERT statement set in the character-position example on page 5-35 is repeated in the following example:

```
FILE cust_loc_data
    (city 1-15,
     state 16-17,
     area_cd 23-25 NULL = 'xxx',
     phone 23-34 NULL = 'xxx-xxx-xxxx',
     zip 18-22,
     state_area 16-17 : 23-25);
INSERT INTO cust_address (col1, col3, col4)
    VALUES (city, state, zip);
```

The FILE statement defines six data fields from the **cust_loc_data** table data rows. The statement names the fields and defines the length of each field using character positions. Compare the FILE statement in the preceding example with the data rows in the Figure 5-2.

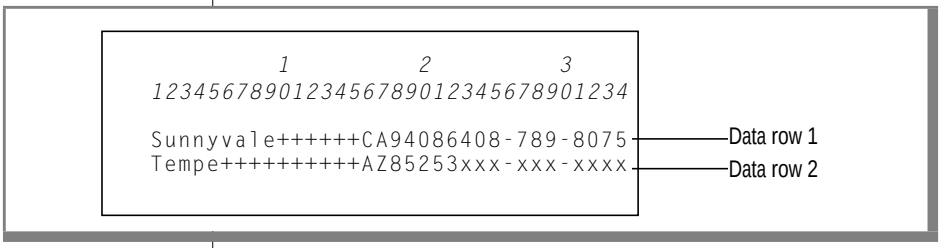


Figure 5-2
A sample data file

The FILE statement defines the following data fields, which are derived from the data rows in Figure 5-2.

Column	Values from Data Row 1	Values from Data Row 2
city	Sunnyvale++++++	Tempe++++++++++
state	CA	AZ
area_cd	408	null
phone	408-789-8075	null
zip	94086	85253
state_area	CA408	AZxxx

The null strings defined for the **phone** and **area_cd** fields generate the null values in those columns but do not affect the values stored in the **state_area** column.

THE INSERT statement uses the field names and values derived from the FILE statement as the value-list input. Consider the following INSERT statement:

```
INSERT INTO cust_address (col1, col3, col4)
    VALUES (city, state, zip);
```

Using the the FILE statement on page 5-38 and the data shown in Figure 5-2 on page 5-38, the INSERT statement puts the following information into the **cust_address** table:

Column	Values from Data Row 1	Values from Data Row 2
col1	Sunnyvale++++++	Tempe++++++++++
col2	null	null
col3	CA	AZ
col4	94086	85253

Because the second column in **cust_address** (**col2**) is not named, the new data row contains a null (assuming that the column permits nulls).

Consider the following INSERT statement:

```
INSERT INTO cust_sort
    VALUES (area_cd, zip);
```

This INSERT statement inserts the following data rows into the **cust_sort** table:

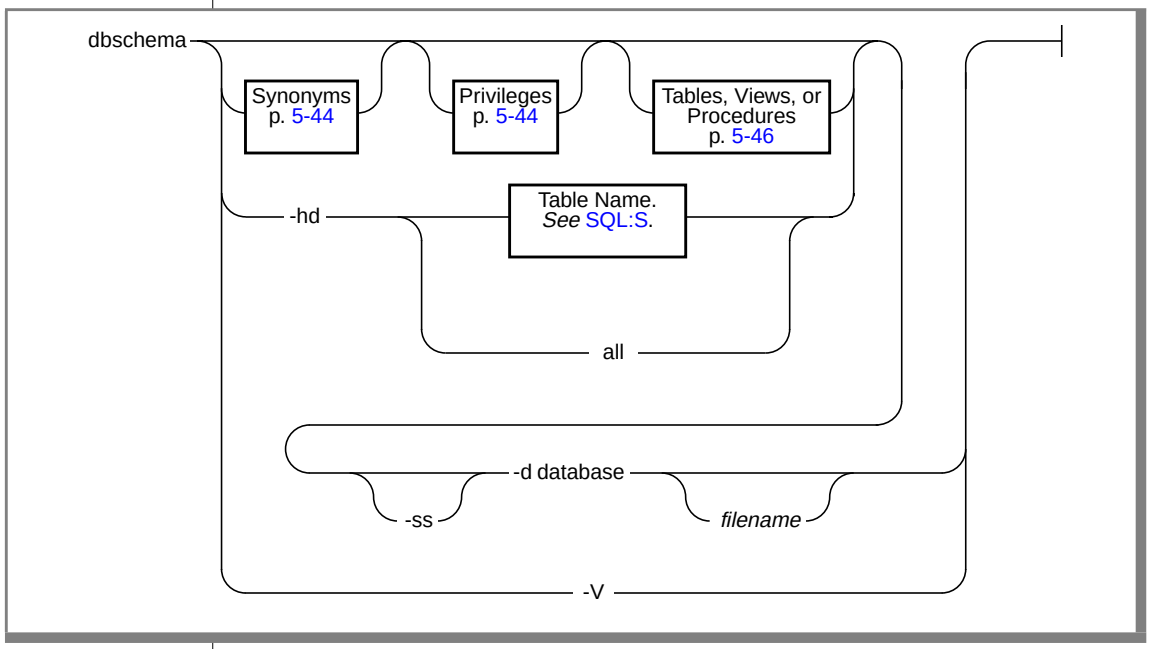
Column	Values from Data Row 1	Values from Data Row 2
col1	408	NULL
col2	94086	85253

Because no column list is provided, **dbload** reads the names of all the columns in **cust_sort** from the system catalog. (You cannot insert data into a temporary table because temporary tables are not entered into the system catalog.) Values to load into each column are specified by field names from the previous FILE statement. You do not need one FILE statement for each INSERT statement.

The *dbschema* Utility

You can use the **dbschema** utility for the following purposes:

- To display the SQL statements (the *schema*) required to replicate a database or a specific table, view, or procedure
- To display the schema for the Information Schema views
- To display the distribution information stored for one or more tables in the database



all directs **dbschema** to include all the tables in the database in the display of distributions.

-d *database* specifies the database to which the schema applies. The database can be on a remote database server. If you want to use more than the simple name of the database, refer to the Database Name segment in Chapter 1 of the *Informix Guide to SQL: Syntax*.

filename specifies the filename to contain the **dbschema** output. If you do not supply a *filename*, **dbschema** sends output to the screen. If you do supply a *filename*, **dbschema** creates a file to contain the **dbschema** output and gives it the name you specify.

-hd displays the distribution as data values. See "Displaying the Distribution Information for Tables" on page 5-47.

-ss	generates server-specific information for a table specified in previous options. This option is ignored if no table schema is generated. In INFORMIX-SE, the **-ss** option generates the pathname where the table was created if the table is not in the database directory. In INFORMIX-OnLine Dynamic Server, the **-ss** option always generates the lock mode, extent sizes, and the dbspace name if the dbspace name is different from the database dbspace. In addition, if tables are fragmented, the **-ss** option displays information about the fragmentation strategy.
-V	displays product version information.

You must be the DBA or have the Connect or Resource privilege to the database before you can run **dbschema** on it. If you are using INFORMIX-SE, the database must exist in your current directory or in a directory cited in your **DBPATH** environment variable.

When the NLS environment variables are set correctly, as described in Chapter 4, "Environment Variables," **dbschema** can handle foreign characters and NLS databases.

You can use delimited identifiers with the **dbschema** utility. The utility detects database objects that are keywords, mixed case, or have special characters and places double quotes around them.

Creating the Schema for a Database

You can create the schema for an entire database or for a portion of the database. The options for **dbschema** allow you to perform the following actions:

- Display CREATE SYNONYM statements by owner, for a specific table, or for the entire database.
- Display the CREATE TABLE, CREATE VIEW, or CREATE PROCEDURE statements for a specific table or for the entire database.
- Display all GRANT privilege statements that affect a specified user or that affect all users for a database or a specific table.

Using **dbschema** and specifying only the database name is equivalent to using **dbschema** with all its options (except for the **-hd** and **-ss** options). In addition, if Information Schema views have been created for the database, this schema is shown. For example, the following two statements are equivalent:

```
dbschema -d stores7
dbschema -s all -p all -t all -f all -d stores7
```

The SERIAL fields included in CREATE TABLE statements displayed by **dbschema** do not specify a starting value. New SERIAL fields created using the schema file have a starting value of one, regardless of their starting value in the original database. If this is not acceptable, you must modify the schema file.

Creating Schemas for Databases Across a Network

You can specify a database on any accessible OnLine database server using the syntax in the database name. You can specify a database on another SE database server by including the database server name and directory path with the database name.

The command shown in the following example displays the schema for the **stores7** database in the **turku** directory on the **finland** database server:

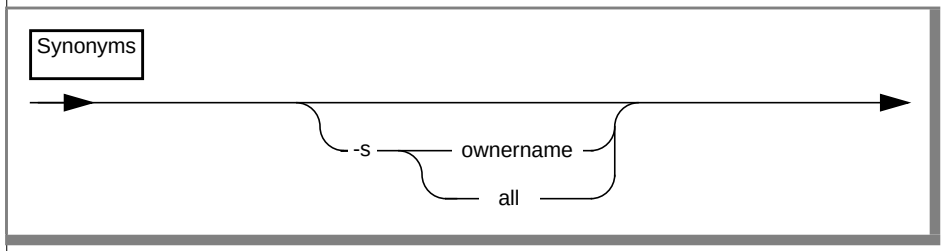
```
dbschema -d //finland/turku/stores7
```

Owner Naming with dbschema

The **dbschema** utility uses the *owner.object* convention when it generates any CREATE TABLE, CREATE INDEX, CREATE SYNONYM, CREATE VIEW, CREATE PROCEDURE, or GRANT statements, and when it reproduces any unique, referential, or check constraints. As a result, if you use the **dbschema** output to create a new object (table, index, view, procedure, constraint, or synonym), the new object is owned by the owner of the original object. If you want to change the owner of the new object, you must edit the **dbschema** output before you run it as an SQL script.

For more information about the CREATE TABLE, CREATE INDEX, CREATE SYNONYM, CREATE VIEW, CREATE PROCEDURE, and GRANT statements, see the *Informix Guide to SQL: Syntax*.

Obtaining the Synonym Schema



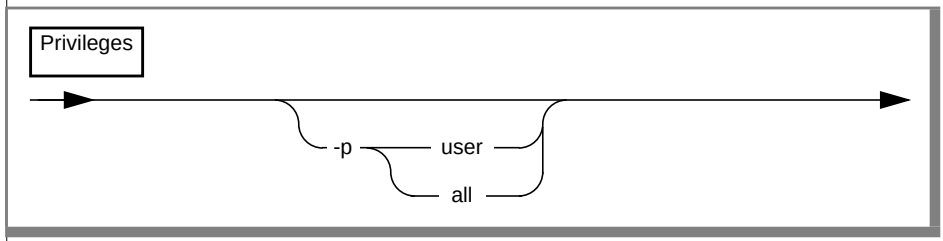
-s *ownername*	directs **dbschema** to display THE CREATE SYNONYM statements owned by *ownername*.
-s *all*	directs **dbschema** to display all CREATE SYNONYM statements for the database, table, or view specified.

Output from **dbschema** that is executed with the specified option `-s alice` might appear, as shown in the following example:

```
CREATE SYNONYM 'alice'.cust FOR 'alice'.customer
```

For more information about the CREATE SYNONYM statement, see the *Informix Guide to SQL: Syntax*.

Obtaining the Privilege Schema



-p *user*	directs **dbschema** to display the GRANT statements that grant privileges to *user*. You specify only one user.
-p *all*	directs **dbschema** to display the GRANT statements for all users for the database, table, or view specified.

You cannot specify a specific list of users with the **-p** option. You can specify either one user or all users.

In the **dbschema** output, the AS keyword indicates the grantor of a GRANT statement. The following example output indicates that **norma** issued the GRANT statement:

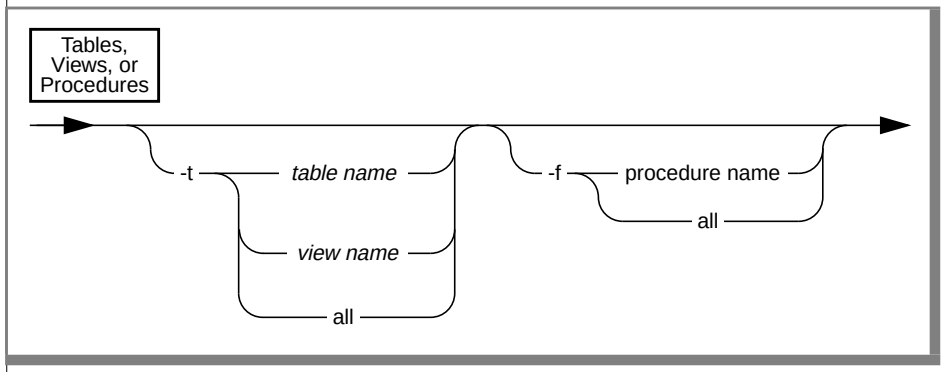
```
GRANT ALL ON 'tom'.customer TO 'claire' AS 'norma'
```

When the GRANT and AS keywords appear in the **dbschema** output, you might need to grant privileges before you run the **dbschema** output as an SQL script. Referring to the previous example output line, the following conditions must be true before you can run the statement as part of a script:

- **norma** must have the Connect privilege to the database.
- **norma** must have all privileges WITH GRANT OPTION for the table **tom.customer**.

For more information about the GRANT statement, see the *Informix Guide to SQL: Syntax*.

Specifying a Table, View, or Procedure



-f all	directs **dbschema** to limit the SQL statement output to those statements that are needed to replicate all procedures.
-f *procedure name*	directs **dbschema** to limit the SQL statement output to only those statements that are needed to replicate the specified procedure.
-t *table name*	directs **dbschema** to limit the SQL statement output to only those statements that are needed to replicate the specified table.
-t *view name*	directs **dbschema** to limit the SQL statement output to only those statements that are needed to replicate the specified view.
-t *all*	directs **dbschema** to include in the SQL statement output all statements that are needed to replicate all tables and views.

For more information about the CREATE PROCEDURE statement, see the *Informix Guide to SQL: Syntax*.

Using the -ss Option to Obtain Table Information

When you use the **-ss** option, you can retrieve information about fragmented tables, the lock mode, and extent sizes.

The following **dbschema** output shows the expressions specified for fragmented table:

```
DBSCHEMA Schema Utility       INFORMIX-SQL Version 7.1.UC1
Copyright (C) Informix Software, Inc., 1984-1994
{ TABLE "sallyc".t1 row size = 8 number of columns = 1 index size = 0 }
create table "sallyc".t1
(
c1 integer
) fragment by expression
(c1 < 100 ) in db1 ,
((c1 >= 100 ) AND (c1 < 200 ) ) in db2 ,
remainder in db4
extent size 16 next size 16 lock mode page;
revoke all on "sallyc".t1 from "public";
```

Displaying the Distribution Information for Tables

To display the distribution information stored for a table in a database, use the **-hd** option with the name of the table. If you specify the ALL keyword for the tablename, the distributions for all the tables in the database are displayed.

Distribution information is stored only if the UPDATE STATISTICS...MEDIUM or HIGH statement has been run for one or more columns of a table.

The output of **dbschema** for distributions is provided in the following parts:

- Distribution description
- Distribution information
- Overflow information

Each section provided by dbschema is explained in the following sections. As an example, the discussion uses the following distribution for the fictional table called **invoices**. This table contains 165 rows, including duplicates.

The output for this discussion can be generated with a call to **dbschema** that is similar to the following example:

```
dbschema -hd invoices -d pubs_stores7
```

Example Output

```
DBSCHEMA Schema Utility       INFORMIX-SQL Version 7.1.UC1
Copyright (C) Informix Software, Inc., 1984-1994
{

Distribution for cathl.invoices.invoice_num

Constructed on 03/10/1994

High Mode, 10.000000 Resolution


--- DISTRIBUTION ---

      (                          5)
  1: (  16,      7,            11)
  2: (  16,      6,            17)
  3: (  16,      8,            25)
  4: (  16,      8,            38)
  5: (  16,      7,            52)
  6: (  16,      8,            73)
  7: (  16,     12,            95)
  8: (  16,     12,           139)
  9: (  16,     11,           182)
 10: (  10,      5,           200)


--- OVERFLOW ---

  1: (   5,                   56)
  2: (   6,                   63)
}
```

Distribution Description

The first part of the **dbschema** output describes which data distributions have been created for the specified table. The name of the table is stated in the following example:

```
Distribution for cathl.invoices.invoice_num
```

The output is for the **invoices** table, which is owned by the user `cathl`. The particular column being described by this data distribution is **invoice_num**. If a table has distributions built on more than one column, **dbschema** lists the distributions for each column separately.

The date on which the distributions are constructed is listed. In this example, it is `03/10/1994`, which is the date when the UPDATE STATISTICS statement that generated the distributions was executed. You can use this date to tell how outdated your distributions are. Although the system records the date, it does not record the time.

The last line of the description portion of the output describes the mode (medium or high) in which the distributions were created, and the resolution. If you create the distributions with medium mode, the confidence of the sample is also listed. For example, if the UPDATE STATISTICS statement is executed with HIGH mode with a resolution of 10, the last line appears as shown in the following example:

```
High Mode, 10.000000 Resolution
```

The Distribution Information

The distribution information describes the bins created for the distribution, the range of values in the table and in each bin, and the number of distinct values in each bin. Consider the following example:

```
        (                          5)
  1: (  16,       7,              11)
  2: (  16,       6,              17)
  3: (  16,       8,              25)
  4: (  16,       8,              38)
  5: (  16,       7,              52)
  6: (  16,       8,              73)
  7: (  16,      12,              95)
  8: (  16,      12,             139)
  9: (  16,      11,             182)
 10: (  10,       5,             200)
```

The first value shown in the rightmost column is the smallest value in the table in this column. In this example, it is 5.

The column on the left shows the bin number, in this case 1 through 10. The first number in the parenthesis shows how many values are in the bin. For this table, 10 percent of the total number of rows (165) is, rounded down, 16. The first number is the same for all the bins except for the last. The last row might have a smaller value, indicating that it does not have as many row values. In this example, all the bins contain 16 rows except the last one, which contains 10.

The middle column within the parenthesis indicates how many distinct values are contained in this bin. Thus, if there are 11 distinct values for a 16-value bin, it implies that 1 or more of those values are duplicated at least once.

The right column within the parenthesis is the highest value in the bin. The highest value in the last bin is also the highest value in the table. For this example, the highest value in the last bin is 200.

The Overflow Information

The last portion of the **dbschema** output shows values that have many duplicates. The number of duplicates of indicated values must be greater than a critical amount that is determined as approximately 25 percent of the resolution times the number of rows. If left in the general distribution data, they would skew the distribution, so they are moved from the distribution to a separate list, as shown in the following example:

```
--- OVERFLOW ---

   1: (    5,                  56)
   2: (    6,                  63)
```

For this example, the critical amount is 0.25 * 0.10 * 165, or 4.125. Therefore, any value that is duplicated five or more times is listed in the overflow section. Two values in this distribution are duplicated five or more times in the table: The value 56 is duplicated five times and the value 63 is duplicated six times.

Appendix

The stores7 Database

A

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 the section "Creating the Demonstration Database" on page 12 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(18)	state (foreign key to **state** table
zipcode	CHAR(2)	zipcode
phone	CHAR(5)	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 > 1)
total_price	MONEY(8)	quantity ordered * unit price = total price of item

The *stock* Table

The distributor carries 41 types of sporting goods from various manufacturers. More than one manufacturer can supply an item. For example, the distributor offers 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 **catalog** table appears only if you are using an OnLine database server.

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
code_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.

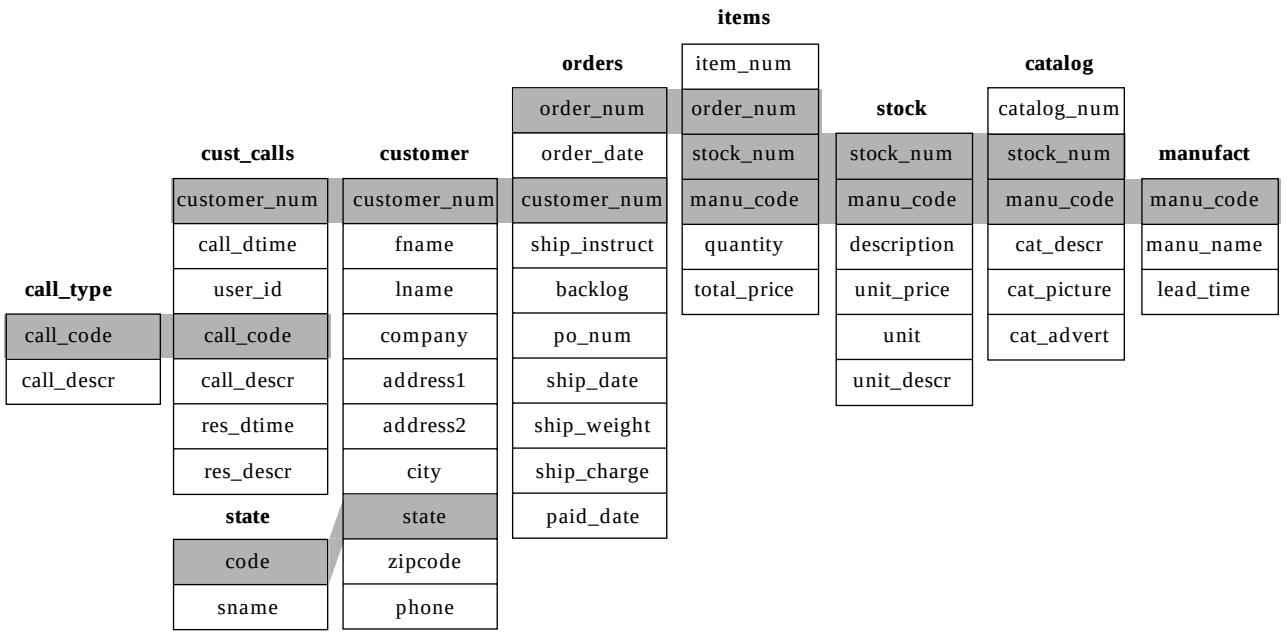
Name	Type	Description
code	CHAR(2)	state code
sname	CHAR(15)	state name

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

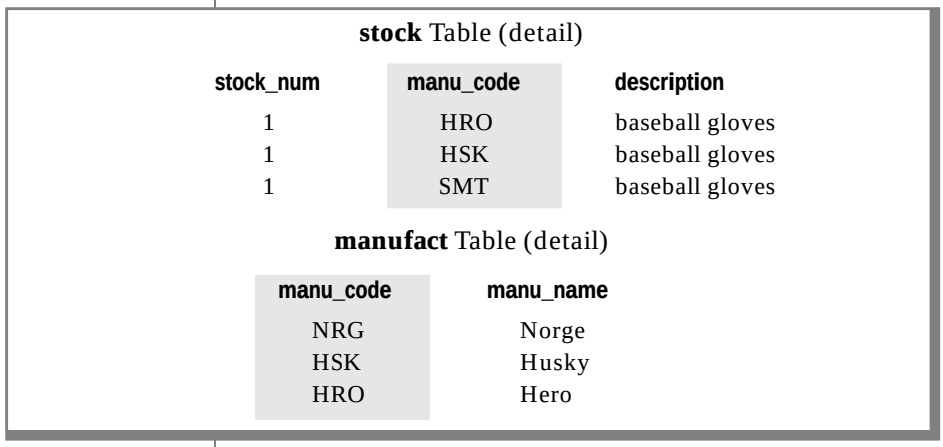
catalog Table (detail)

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 t**able, 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. This is shown in Figure A-18.

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

(1 of 3)

customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone
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
119	Bob	Shorter	The Triathletes Club	2405 Kings High-way		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 Common-wealth Ave.		Brighton	MA	02135	617-232-4159
126	Eileen	Neelie	Neelie's Discount Sports	2539 South Utica St		Denver	CO	80219	303-936-7731

customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone
127	Kim	Satifer	Big Blue Bike Shop	Blue Island Square	12222 Gregory Street	Blue Island	NY	60406	312-944-5691
128	Frank	Lessor	Phoenix University	Athletic Department	1817 N. Thomas Road	Phoenix	AZ	85008	602-533-1817

(3 of 3)

items *Table*

item_num	order_num	stock_num	manu_code	quantity	total_price
1	1001	1	HRO	1	250.00
1	1002	4	HSK	1	960.00
2	1002	3	HSK	1	240.00
1	1003	9	ANZ	1	20.00
2	1003	8	ANZ	1	840.00
3	1003	5	ANZ	5	99.00
1	1004	1	HRO	1	250.00
2	1004	2	HRO	1	126.00
3	1004	3	HSK	1	240.00
4	1004	1	HSK	1	800.00
1	1005	5	NRG	10	280.00
2	1005	5	ANZ	10	198.00
3	1005	6	SMT	1	36.00
4	1005	6	ANZ	1	48.00
1	1006	5	SMT	5	125.00
2	1006	5	NRG	5	140.00
3	1006	5	ANZ	5	99.00
4	1006	6	SMT	1	36.00
5	1006	6	ANZ	1	48.00
1	1007	1	HRO	1	250.00
2	1007	2	HRO	1	126.00
3	1007	3	HSK	1	240.00
4	1007	4	HRO	1	480.00

(1 of 3)

item_num	order_num	stock_num	manu_code	quantity	total_price
5	1007	7	HRO	1	600.00
1	1008	8	ANZ	1	840.00
2	1008	9	ANZ	5	100.00
1	1009	1	SMT	1	450.00
1	1010	6	SMT	1	36.00
2	1010	6	ANZ	1	48.00
1	1011	5	ANZ	5	99.00
1	1012	8	ANZ	1	840.00
2	1012	9	ANZ	10	200.00
1	1013	5	ANZ	1	19.80
2	1013	6	SMT	1	36.00
3	1013	6	ANZ	1	48.00
4	1013	9	ANZ	2	40.00
1	1014	4	HSK	1	960.00
2	1014	4	HRO	1	480.00
1	1015	1	SMT	1	450.00
1	1016	101	SHM	2	136.00
2	1016	109	PRC	3	90.00
3	1016	110	HSK	1	308.00
4	1016	114	PRC	1	120.00
1	1017	201	NKL	4	150.00
2	1017	202	KAR	1	230.00
3	1017	301	SHM	2	204.00
1	1018	307	PRC	2	500.00

(2 of 3)

item_num	order_num	stock_num	manu_code	quantity	total_price
2	1018	302	KAR	3	15.00
3	1018	110	PRC	1	236.00
4	1018	5	SMT	4	100.00
5	1018	304	HRO	1	280.00
1	1019	111	SHM	3	1499.97
1	1020	204	KAR	2	90.00
2	1020	301	KAR	4	348.00
1	1021	201	NKL	2	75.00
2	1021	201	ANZ	3	225.00
3	1021	202	KAR	3	690.00
4	1021	205	ANZ	2	624.00
1	1022	309	HRO	1	40.00
2	1022	303	PRC	2	96.00
3	1022	6	ANZ	2	96.00
1	1023	103	PRC	2	40.00
2	1023	104	PRC	2	116.00
3	1023	105	SHM	1	80.00
4	1023	110	SHM	1	228.00
5	1023	304	ANZ	1	170.00
6	1023	306	SHM	1	190.00

(3 of 3)

call_type *Table*

call_code	code_descr
B	billing error
D	damaged goods
I	incorrect merchandise sent
L	late shipment
O	other

orders *Table*

order_num	order_date	customer_num	ship_instruct	backlog	po_num	ship_date	ship_weight	ship_charge	paid_date
1001	05/20/1994	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)

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 club-house	n	DM354331	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	racer goggles	72.00	box	12/box
313	SHM	swim cap	72.00	box	12/box
313	ANZ	swim cap	60.00	box	12/box

(4 of 4)

catalog *Table*

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10001	1	HRO	Brown leather. Specify first base-man'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 Ex-pands the Sweet Spot
10006	3	SHM	Aluminum. Blue with black tape. 31", 20 oz or 22 oz; 32", 21 oz or 23 oz; 33", 22 oz or 24 oz.	<BYTE value>	Durable Aluminum for High School and Collegiate Athletes
10007	4	HSK	Norm Van Brocklin signature style.	<BYTE value>	Quality Pigskin with Norm Van Brocklin Signature
10008	4	HRO	NFL-Style pigskin.	<BYTE value>	Highest Quality Football for High School and Collegiate Competitions

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10009	5	NRG	Graphite frame. Synthetic strings.	<BYTE value>	Wide Body Amplifies Your Natural Abilities by Providing More Power Through Aerodynamic Design
10010	5	SMT	Aluminum frame. Synthetic strings.	<BYTE value>	Mid-Sized Racquet For the Improving Player
10011	5	ANZ	Wood frame, cat-gut strings.	<BYTE value>	Antique Replica of Classic Wooden Racquet Built with Cat-Gut Strings
10012	6	SMT	Soft yellow color for easy visibility in sunlight or artificial light.	<BYTE value>	High-Visibility Tennis, Day or Night
10013	6	ANZ	Pro-core. Available in neon yellow, green, and pink.	<BYTE value>	Durable Construction Coupled with the Brightest Colors Available
10014	7	HRO	Indoor. Classic NBA style. Brown leather.	<BYTE value>	Long-Life Basketballs for Indoor Gymnasiums
10015	8	ANZ	Indoor. Finest leather. Professional quality.	<BYTE value>	Professional Volleyballs for Indoor Competitions
10016	9	ANZ	Steel eyelets. Nylon cording. Double-stitched. Sanctioned by the National Athletic Congress.	<BYTE value>	Sanctioned Volleyball Netting for Indoor Professional and Collegiate Competition

(2 of 12)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
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
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

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10023	105	PRC	Front wheels laced with 15g spokes in a 3-cross pattern. Rear wheels laced with 14g spikes in a 3-cross pattern.	<BYTE value>	Durable Training Wheels That Hold True Under Toughest Conditions
10024	105	SHM	Polished alloy. Sealed-bearing, quick-release hubs. Double-butted. Front wheels are laced 15g / 2-cross. Rear wheels are laced 15g / 3-cross.	<BYTE value>	Extra Lightweight Wheels for Training or High-Performance Touring
10025	106	PRC	Hard anodized alloy with pearl finish. 6mm hex bolt hardware. Available in lengths of 90-140mm in 10mm increments.	<BYTE value>	ProCycle Stem with Pearl Finish
10026	107	PRC	Available in three styles: Men's racing; Men's touring; and Women's. Anatomical gel construction with lycra cover. Black or black / hot pink.	<BYTE value>	The Ultimate In Riding Comfort, Lightweight With Anatomical Support
10027	108	SHM	Double or triple crankset with choice of chainrings. For double crankset, chainrings from 38-54 teeth. For triple crankset, chainrings from 24-48 teeth.	<BYTE value>	Customize Your Mountain Bike With Extra-Durable Crankset
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

(4 of 12)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10029	109	SHM	Ingenious new design combines button on sole of shoe with slot on a pedal plate to give riders new options in riding efficiency. Choose full or partial locking. Four plates mean both top and bottom of pedals are slotted—no fishing around when you want to engage full power. Fast unlocking ensures safety when maneuverability is paramount.	<BYTE value>	Ingenious Pedal/Clip Design Delivers Maximum Power And Fast Unlocking
10030	110	PRC	Super-lightweight. Meets both ANSI and Snell standards for impact protection. 7.5 oz. Quick-release shadow buckle.	<BYTE value>	Feather-Light, Quick-Release, Maximum Protection Helmet
10031	110	ANZ	No buckle so no plastic touches your chin. Meets both ANSI and Snell standards for impact protection. 7.5 oz. Lycra cover.	<BYTE value>	Minimum Chin Contact, Feather-Light, Maximum Protection Helmet
10032	110	SHM	Dense outer layer combines with softer inner layer to eliminate the mesh cover, no snagging on brush. Meets both ANSI and Snell standards for impact protection. 8.0 oz.	<BYTE value>	Mountain Bike Helmet: Smooth Cover Eliminates the Worry of Brush Snags But Delivers Maximum Protection
10033	110	HRO	Newest ultralight helmet uses plastic shell. Largest ventilation channels of any helmet on the market. 8.5 oz.	<BYTE value>	Lightweight Plastic with Vents Assures Cool Comfort Without Sacrificing Protection

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10034	110	HSK	Aerodynamic (teardrop) helmet covered with anti-drag fabric. Credited with shaving 2 seconds / mile from winner's time in Tour de France time-trial. 7.5 oz.	<BYTE value>	Teardrop Design Used by Yellow Jerseys, You Can Time the Difference
10035	111	SHM	Light-action shifting 10 speed. Designed for the city commuter with shock-absorbing front fork and drilled eyelets for carry-all racks or bicycle trailers. Internal wiring for generator lights. 33 lbs.	<BYTE value>	Fully Equipped Bicycle Designed for the Serious Commuter Who Mixes Business With Pleasure
10036	112	SHM	Created for the beginner enthusiast. Ideal for club rides and light touring. Sophisticated triple-butted frame construction. Precise index shifting. 28 lbs.	<BYTE value>	We Selected the Ideal Combination of Touring Bike Equipment, Then Turned It Into This Package Deal: High-Performance on the Roads, Maximum Pleasure Everywhere
10037	113	SHM	Ultra-lightweight. Racing frame geometry built for aerodynamic handlebars. Cantilever brakes. Index shifting. High-performance gearing. Quick-release hubs. Disk wheels. Bladed spokes.	<BYTE value>	Designed for the Serious Competitor, The Complete Racing Machine
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

(6 of 12)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10039	201	NKL	Designed for comfort and stability. Available in white & blue or white & brown. Specify size.	<BYTE value>	Full-Comfort, Long-Wearing Golf Shoes for Men and Women
10040	201	ANZ	Guaranteed waterproof. Full leather upper. Available in white, bone, brown, green, and blue. Specify size.	<BYTE value>	Waterproof Protection Ensures Maximum Comfort and Durability In All Climates
10041	201	KAR	Leather and leather mesh for maximum ventilation. Waterproof lining to keep feet dry. Available in white & gray or white & ivory. Specify size.	<BYTE value>	Karsten's Top Quality Shoe Combines Leather and Leather Mesh
10042	202	NKL	Complete starter set utilizes gold shafts. Balanced for power.	<BYTE value>	Starter Set of Woods, Ideal for High School and Collegiate Classes
10043	202	KAR	Full set of woods designed for precision control and power performance.	<BYTE value>	High-Quality Woods Appropriate for High School Competitions or Serious Amateurs
10044	203	NKL	Set of eight irons includes 3 through 9 irons and pitching wedge. Originally priced at $489.00.	<BYTE value>	Set of Irons Available From Factory at Tremendous Savings: Discontinued Line
10045	204	KAR	Ideally balanced for optimum control. Nylon-covered shaft.	<BYTE value>	High-Quality Beginning Set of Irons Appropriate for High School Competitions
10046	205	NKL	Fluorescent yellow.	<BYTE value>	Long Drive Golf Balls: Fluorescent Yellow

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10047	205	ANZ	White only.	<BYTE value>	Long Drive Golf Balls: White
10048	205	HRO	Combination fluorescent yellow and standard white.	<BYTE value>	HiFlier Golf Balls: Case Includes Fluorescent Yellow and Standard White
10049	301	NKL	Super shock-absorbing gel pads disperse vertical energy into a horizontal plane for extraordinary cushioned comfort. Great motion control. Men's only. Specify size.	<BYTE value>	Maximum Protection For High-Mileage Runners
10050	301	HRO	Engineered for serious training with exceptional stability. Fabulous shock absorption. Great durability. Specify men's/women's, size.	<BYTE value>	Pronators and Supinators Take Heart: A Serious Training Shoe For Runners Who Need Motion Control
10051	301	SHM	For runners who log heavy miles and need a durable, supportive, stable platform. Mesh/synthetic upper gives excellent moisture dissipation. Stability system uses rear antipronation platform and forefoot control plate for extended protection during high-intensity training. Specify men's/women's size.	<BYTE value>	The Training Shoe Engineered for Marathoners and Ultra-Distance Runners
10052	301	PRC	Supportive, stable racing flat. Plenty of forefoot cushioning with added motion control. Women's only. D widths available. Specify size.	<BYTE value>	A Woman's Racing Flat That Combines Extra Forefoot Protection With a Slender Heel

(8 of 12)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10053	301	KAR	Anatomical last holds your foot firmly in place. Feather-weight cushioning delivers the responsiveness of a racing flat. Specify men's / women's size.	<BYTE value>	Durable Training Flat That Can Carry You Through Marathon Miles
10054	301	ANZ	Cantilever sole provides shock absorption and energy rebound. Positive traction shoe with ample toe box. Ideal for runners who need a wide shoe. Available in men's and women's. Specify size.	<BYTE value>	Motion Control, Protection, and Extra Toebox Room
10055	302	KAR	Reusable ice pack with velcro strap. For general use. Velcro strap allows easy application to arms or legs.	<BYTE value>	Finally, An Ice Pack for Achilles Injuries and Shin Splints that You Can Take to the Office
10056	303	PRC	Neon nylon. Perfect for running or aerobics. Indicate color: Fluorescent pink, yellow, green, and orange.	<BYTE value>	Knock Their Socks Off With YOUR Socks
10057	303	KAR	100% nylon blend for optimal wicking and comfort. We've taken out the cotton to eliminate the risk of blisters and reduce the opportunity for infection. Specify men's or women's.	<BYTE value>	100% Nylon Blend Socks - No Cotton

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10058	304	ANZ	Provides time, date, dual display of lap / cumulative splits, 4-lap memory, 10 hr count-down timer, event timer, alarm, hour chime, waterproof to 50m, velcro band.	<BYTE value>	Athletic Watch w / 4-Lap Memory
10059	304	HRO	Split timer, waterproof to 50m. Indicate color: Hot pink, mint green, space black.	<BYTE value>	Waterproof Triathlete Watch In Competition Colors
10060	305	HRO	Contains ace bandage, anti-bacterial cream, alcohol cleansing pads, adhesive bandages of assorted sizes, and instant-cold pack.	<BYTE value>	Comprehensive First-Aid Kit Essential for Team Practices, Team Traveling
10061	306	PRC	Converts a standard tandem bike into an adult / child bike. User-tested assembly instructions	<BYTE value>	Enjoy Bicycling With Your Child On a Tandem; Make Your Family Outing Safer
10062	306	SHM	Converts a standard tandem bike into an adult / child bike. Lightweight model.	<BYTE value>	Consider a Touring Vacation For the Entire Family: A Lightweight, Touring Tandem for Parent and Child
10063	307	PRC	Allows mom or dad to take the baby out too. Fits children up to 21 pounds. Navy blue with black trim.	<BYTE value>	Infant Jogger Keeps A Running Family Together
10064	308	PRC	Allows mom or dad to take both children! Rated for children up to 18 pounds.	<BYTE value>	As Your Family Grows, Infant Jogger Grows With You

(10 of 12)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
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
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

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
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

(12 of 12)

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.
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.

(1 of 2)

customer_num	call_dtime	user_id	call_code	call_descr	res_dtime	res_descr
116	1993-11-28 13:34	mannyn	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	mannyn	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.

(2 of 2)

manufact Table

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

state Table

code	sname	code	sname
AK	Alaska	MT	Montana
AL	Alabama	NE	Nebraska
AR	Arkansas	NC	North Carolina
AZ	Arizona	ND	North Dakota
CA	California	NH	New Hampshire
CT	Connecticut	NJ	New Jersey
CO	Colorado	NM	New Mexico
DC	D.C.	NV	Nevada
DE	Delaware	NY	New York
FL	Florida	OH	Ohio

(1 of 2)

code	sname	code	sname
GA	Georgia	OK	Oklahoma
HI	Hawaii	OR	Oregon
IA	Iowa	PA	Pennsylvania
ID	Idaho	PR	Puerto Rico
IL	Illinois	RI	Rhode Island
IN	Indiana	SC	South Carolina
KY	Kentucky	TN	Tennessee
LA	Louisiana	TX	Texas
MA	Massachusetts	UT	Utah
MD	Maryland	VA	Virginia
ME	Maine	VT	Vermont
MI	Michigan	WA	Washington
MN	Minnesota	WI	Wisconsin
MO	Missouri	WV	West Virginia
MS	Mississippi	WY	Wyoming

(2 of 2)

Glossary

access method A procedure that is used to retrieve rows from or insert rows into a storage location. In the SET EXPLAIN statement, access method refers to the type of table access in a query (for example, SEQUENTIAL SCAN as opposed to INDEX PATH).

access privileges The types of operations that a user has permission to perform in a specific database, table, or column. Informix maintains its own set of database and table access privileges, which are independent of the operating system access privileges for operating system files.

active set The collection of rows that satisfies a query associated with a cursor.

aggregate functions The functions that return a single value based on the values of columns in one or more rows of a table (for example, the total number, sum, average, and maximum or minimum of an expression in a query or report). Aggregate functions are sometimes referred to as *set functions* or *math functions*.

alias A temporary alternative name for a table in a query; usually used in complex subqueries and required for self-joins. In a form-specification file or any SQL query, alias refers to a single-word alternative name used in place of a more complex table name (for example, *t1* as an alias for *owner.table_name*).

ANSI Acronym for the American National Standards Institute. This group sets standards in many areas, including the computer industry and standards for SQL languages.

ANSI-compliant Refers to a database that conforms to certain ANSI performance standards. Informix databases can be created either as ANSI-compliant or not ANSI-compliant. An ANSI-compliant database enforces certain ANSI requirements, such as implicit transactions, required owner-naming, and unbuffered logging (unbuffered logging only when using INFORMIX-OnLine Dynamic Server), that are not enforced in databases that are not ANSI-compliant.

application development tool Software, such as INFORMIX-4GL, that you can use to create and maintain a database. The software allows a user to send instructions and data to and receive information from the database server.

application process The process that manages an ESQL, 4GL, or other program at run time. It executes the program logic and initiates SQL requests. Memory that is allocated for program variables, program data, and the fetch buffer is part of this process. See also *database server process*.

application program An executable file or logically related set of files that perform one or more database management tasks.

arbitrary rule A series of expressions that you define using SQL relational and logical operators. Unlike the range rule, the arbitrary rule allows you to use any relational operator and any logical operator to define the expressions. Typically includes the use of the OR logical operator to group data.

archiving Copying all the data and indexes of a database onto a new medium, usually a tape or a different physical device from the one that stores the database. Archiving is used for recovering from a failure and is usually performed by a database administrator. See *backup*.

argument A value passed to a function, routine, stored procedure, or command.

array A set of items of the same type. Individual members of the array are referred to as *elements* and are usually distinguished by an integer argument that gives the position of the element in the array. Informix arrays can have up to three dimensions.

attached index An index that is created without a fragmentation strategy. An attached index can also be an index created on a fragmented table.

audit event Any OnLine activity or operation that could potentially access and alter data, which should be recorded and monitored by the OnLine secure auditing facility. Examples of audit events include accessing tables, altering indexes, dropping chunks, granting database access, updating the current row, running OnLine utilities, and so on. (For a complete list of audit events, see Appendix A in the *INFORMIX-OnLine Dynamic Server Trusted Facility Manual*

audit file A file that contains records of audit events and resides in the specified audit directory. Audit files form an audit trail of information that can be extracted by the OnLine secure auditing facility for analysis by the OnLine administrator.

audit mask A structure that specifies which events should be audited by (or excluded from auditing by) the OnLine secure auditing facility.

audit trail A history of all changes to a table in SE database servers.

audit trail log A file that contains a history of all changes to a table. Starting from an archived database, an audit trail log can reconstruct all subsequent changes to the table in INFORMIX-SE database servers.

auxiliary statements The SQL statements that you use to obtain auxiliary information about tables and databases. These statements include INFO, OUTPUT, WHENEVER, and GET DIAGNOSTICS.

B+ tree A method of organizing an index into a binary tree for efficient record retrieval.

backup To make a duplicate of a computer file on another device or tape to preserve existing work, in case of a computer failure or other mishap. In INFORMIX-OnLine Dynamic Server, *backup* refers to duplicating logical log files and *archiving* refers to duplicating data.

before-image The image of a row, page, or other item before any changes are made to it.

blobpage The unit of disk allocation within a blobspace under INFORMIX-OnLine Dynamic Server. The OnLine administrator determines the size of a blobpage; the size can vary from blobspace to blobspace.

blobs Acronym for binary large objects. Blobs are data objects that effectively have no maximum size (theoretically as large as 2^{31} bytes).

blobspace A logical collection of chunks that is used to store TEXT and BYTE data in INFORMIX-OnLine Dynamic Server.

buffer A portion of computer memory where a program temporarily stores data. Data typically is read into or written out from buffers to disk.

buffered logging A type of logging that holds transactions in a memory buffer until the buffer is full, regardless of when the transaction is committed or rolled back. INFORMIX-OnLine Dynamic Server provides this option to speed up operations by reducing the number of disk writes.

byte A unit of storage that corresponds approximately to one character. It is the smallest addressable computer memory unit. A *kilobyte* is 1,024 bytes. A *megabyte* is 10^6 bytes. A byte is also known as an *octet*. (When BYTE appears in uppercase letters, it refers to the Informix data type.)

callback function A user-defined function called during execution of an SQL request.

Cartesian product The set that results when you pair each and every member of one set with each and every member of another set. A Cartesian product results from a multiple-table query when you do not specify the joining conditions among tables. See *join*.

cascading deletes Defines the delete action on referencing columns (foreign key) in a referential constraint. When defined on the referencing columns with the CREATE TABLE or ALTER TABLE statements, deletes from the referenced columns cascade to the referencing columns.

case-sensitivity The condition of distinguishing between uppercase and lowercase letters. Be careful running Informix programs because certain commands and their options are case-sensitive; that is, they react differently to the same letters presented in uppercase and lowercase characters.

character A logical unit of storage for the value code in a code set. It can be represented by one or more bytes and can be diacritic or alphabetic.

check constraint A condition that must be met before data can be assigned to a table column during an INSERT or UPDATE statement. Check constraints are defined using search conditions.

checkpoint A point in time during an INFORMIX-OnLine Dynamic Server operation when the pages on disk are synchronized with the pages in the shared-memory buffer pool. Checkpoints are marked by a special record written into the logical log.

child table The referencing table in a referential constraint. See *parent table*.

chunk A large continuous section of disk space for OnLine. A chunk can correspond to a UNIX disk partition. A specified set of chunks defines a dbspace or blobspace.

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.

client/server processing A function that allows an application to run on multiple computers across a network using the processing power of the computers involved.

close a cursor To drop the association between the active set of rows resulting from a query and a cursor.

close a database To relinquish the *current* status of a database. Only one current database can exist at a time.

close a file To release the association between a file and a program.

cluster an index To rearrange the physical data of a table according to a specific index.

cluster key The column in a table that logically relates a group of 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.

code set The representation of a national character set where each code value has a one-to-one mapping with a character graphic.

collating sequence The sequence of values or codes that specifies some logical order in which the character fields in a database are sorted and indexed. It may involve dictionary ordering or character ordering of data. Collating sequence is also known as *collation order.*

column A data element containing a particular type of information that occurs in every row of the table; also known as a *field.*

command file A system file that contains one or more statements or commands, such as SQL statements.

comment The information in a program file that is not processed by the computer but documents the program. You use special characters, such as a pound sign (#), curly braces ({ }), slash marks (/) and asterisks (*), or a double dash (--) to identify comments, depending on the programming environment.

COMMIT WORK To terminate a transaction by accepting all changes to the database since the beginning of the transaction.

COMMITTED READ An Informix level of isolation available through INFORMIX-OnLine Dynamic Server and set with the SET ISOLATION statement in which a user can view only rows that are currently committed at the moment of the query request; that is, a user cannot view rows that were changed as a part of a currently uncommitted transaction. It is the default level of isolation under OnLine for databases that are not ANSI-compliant. See *isolation*.

compile To translate a file that contains instructions (in a higher-level language) into a file containing the corresponding machine-level instructions.

compile-time errors The errors that occur at the time the program source code is converted to executable form. Compare with *run-time errors*.

composite index An index constructed on two or more columns of a table. The ordering imposed by the composite index varies least frequently on the first-named column and most frequently on the last-named column.

composite join A join between two tables based on the relationship among two or more columns in each table. See *join*.

concatenate To append a second string to the end of a first string.

concatenation operator A symbolic notation composed of two pipe symbols (| |) used in expressions to indicate the joining of two strings.

concurrency The ability of two or more processes to access the same database simultaneously.

configuration file A file read during INFORMIX-OnLine Dynamic Server disk or shared-memory initialization that contains the parameters that specify values for configurable behavior. OnLine and its archiving tool, ON-Archive, use configuration files.

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 *explicit connection*, *implicit connection*.

constant A nonvarying data element or value.

constraint A restriction on what kinds of data can be inserted or updated in tables. See also *check constraint*, *primary-key constraint*, *referential constraint*, and *unique constraint*.

control character A character whose occurrence in a particular context initiates, modifies, or stops a control function (an operation to control a device, for example, in moving a cursor or in reading data). In a program, you can define actions that use the CTRL key with another key to execute some programming action (for example, entering CTRL-W to obtain on-line Help in Informix products). A control character is sometimes referred to as a *control key*. Compare to *printable character*.

correlation name The prefix that you can use with a column name in a triggered action to refer to an old (before triggering statement) or a new (after triggering statement) column value. The associated column name must belong to the triggering table.

corrupted database A database whose tables or indexes contain incomplete or invalid data.

corrupted index An index that does not correspond exactly to its table.

current row The most recently retrieved row of the active set of a query.

cursor An identifier associated with a group of rows; conceptually, the pointer to the current row. You can use cursors for SELECT statements or EXECUTE PROCEDURE statements (associating the cursor with the rows returned by a query) or INSERT statements (associating the cursor with a buffer to insert multiple rows as a group). A select cursor is declared for sequential only (regular cursor) or nonsequential (scroll cursor) retrieval of row information. In addition, you can declare a select cursor for update (initiating locking control for updated and deleted rows) or with hold (completing a transaction does not close the cursor). In ESQL/C and ESQL/COBOL, a cursor can be dynamic, meaning that it can be an identifier or a character/string variable.

cursor manipulation statements The SQL statements that control cursors; specifically, the CLOSE, DECLARE, FETCH, FLUSH, OPEN, and PUT statements.

CURSOR STABILITY An Informix level of isolation available through OnLine and set with the SET ISOLATION statement in which the database server must secure a shared lock on a fetched row before the row can be viewed. The database server retains the lock until it receives a request to fetch a new row. See *isolation*.

data access statements The SQL statements that you use to grant and revoke permissions and to lock tables.

data definition statements The SQL statements that you use to create, alter, drop, and rename data objects, including databases, tables, views, synonyms, triggers, and procedures.

data dictionary The collection of tables that keeps track of the structure of the database. Information about the database is maintained in the data dictionary, which is also referred to as the system catalog. See *system catalog*.

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 When database objects have more than one representation at more than one distinct site. INFORMIX-OnLine Dynamic Server implements data replication at the level of database servers.

data restriction Synonym for constraint. See *constraint*.

data type A descriptor assigned to each column in a table or program variable, which indicates the type of data the column or program variable is intended to hold. Informix data types include SMALLINT, INTEGER, SERIAL, SMALLFLOAT, FLOAT, DECIMAL, MONEY, DATE, DATETIME, INTERVAL, CHAR, VARCHAR, TEXT, and BYTE. When NLS is enabled, Informix data types include NCHAR and NVARCHAR.

database A collection of information (contained in tables) that is useful to a particular organization or used for a specific purpose.

Database Administrator See *DBA*.

database application A program that applies database management techniques to implement specific data manipulation and reporting tasks.

database environment Used in the CONNECT statement, either the database server or the database server and database to which a user connects.

database management system See *DBMS*.

database server process The portion of the INFORMIX-SE database management system that actually manipulates the database files. It receives SQL statements from the application process, parses them, optimizes the approach to data retrieval, retrieves the database, and returns the data to the application. In the INFORMIX-OnLine Dynamic Server database management system, these activities are distributed among threads instead of using a single process. See *thread*.

DBA Acronym for *Database Administrator*. The DBA is the individual who is responsible for the contents and use of a database. This is different from the administrator of an INFORMIX-OnLine Dynamic Server database server, who is responsible for managing one or more OnLine database servers.

DBA-privileged Refers to a class of stored procedures created only by a user with DBA database privileges.

DBMS Acronym for *database management system*. It is all the components necessary to create and maintain a database, including the application development tools and the database server.

dbspace A logical collection of one or more INFORMIX-OnLine Dynamic Server 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.

DDL Acronym for data definition language. See *data definition statements*.

deadlock A situation in which two or more threads cannot proceed because each is waiting for data locked by the other (or another) thread. INFORMIX-OnLine Dynamic Server monitors and prevents potential deadlock situations by sending an error message to the application whose request for a lock may result in a deadlock.

debug file A file that receives output used for debugging purposes.

decision-support application An application that provides information, which is used for strategic planning, decision-making, and reporting. It typically executes in a batch environment in a sequential scan fashion and returns a large fraction of the rows scanned. Decision-support queries typically scan the entire database. See *OLTP application*.

decision-support query A query that is generated by a decision-support application. It often requires operations such as multiple joins, temporary tables, and extensive calculations, and can benefit significantly from PDQ. Compare with OLTP query.

declaration statement A programming language statement that describes or defines objects, for example, defining a program variable. Compare to *procedural statement*.

default How a program acts or the values that are assumed if the user does not explicitly specify an action.

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.

delete To remove any row or combination of rows with the DELETE statement.

delimited identifier An identifier surrounded by double quotes. The purpose of a delimited identifier is to allow usage of identifiers that are otherwise identical to SQL reserved keywords or that contain non-alphabetical characters. See also *identifier*.

delimiter A boundary on an input field or the terminator for a column or row. Some files and prepared objects require semicolon (;), comma (,), space, or tab delimiters between statements.

descriptor A quoted string or embedded variable name that identifies an allocated system-descriptor area or an **sqlda** structure. It is used for the Informix SQL APIs. See *identifier*.

detached index An index that is created with a fragmentation strategy or with the IN *dbspace* storage option.

diagnostic area A data structure that stores diagnostic information about an executed SQL statement.

DIRTY READ An Informix level of isolation set with the SET ISOLATION statement that does not account for locks and allows viewing of any existing rows, even rows that currently can be altered from inside an uncommitted transaction. DIRTY READ is the lowest level of isolation (no isolation at all). It is the level at which INFORMIX-SE normally operates and it is an optional level under INFORMIX-OnLine Dynamic Server. See *isolation*.

disk configuration The organization of data on a disk; also refers to the process of preparing a disk to store data.

disk I/O The process of transferring data between memory and disk.

display label A temporary name for a column or expression in a query.

distributed data The ability to access data in multiple databases. The databases can be on the same hardware or on a network. (INFORMIX-OnLine Dynamic Server can perform multiple-database queries.)

distribution scheme A method that OnLine uses to distribute rows or index entries to fragments and in some cases, to reduce the number of fragments that OnLine scans. In an expression-based distribution scheme, you decide where a fragment is to be stored in addition to providing a rule or algorithm that OnLine uses to place the rows into the fragments.

DML Acronym for data manipulation language. See *data manipulation statements*.

dominant table See *outer join*.

dynamic management statements The SQL statements that describe, execute, and prepare statements.

dynamic SQL The statements and structures that allow a program to form an SQL statement during execution, so that portions of the statement can be determined by user input.

dynamic statements The SQL statements that are created at the time the program is executed, rather than when the program is written. You use the PREPARE statement to create dynamic statements.

embedded SQL The SQL statements that are placed within a host language. Informix supports embedded SQL in C and COBOL.

environment variable A variable that is maintained by the operating system for each user and made available to all programs that the user runs.

error message A message that is displayed on the screen or written to a file, describing either the failure of an action or an illegal specification. Each error is identified by a (usually negative) designated number.

error trapping The code within a program that anticipates and reacts to run-time errors.

escape character A character that indicates that the following character, normally interpreted by the program, is to be printed as a literal character instead. The escape character is used with the interpreted character to "escape" or ignore the interpreted meaning.

escape key The keyboard key, usually marked ESC, that is used to terminate one mode and start another mode in most UNIX and DOS systems.

exception An error or warning returned by the database server or a state initiated by a stored procedure statement.

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.

executable file A binary file containing compiled code that can be run as a program; can also refer to a UNIX shell script or a DOS batch file.

execute To carry out a program, procedure, or a set of instructions.

explicit connection A connection made to a database environment that uses the CONNECT statement.

exponent The power to which a value is to be raised.

expression Anything from a simple numeric or alphabetic constant to a more complex series of column values, functions, quoted strings, operators, and keywords. A Boolean expression contains a logical operator (>, <, =, !=, IS NULL, and so on) and evaluates as TRUE, FALSE, or UNKNOWN. An arithmetic expression contains the operators (+, –, ×, /, and so on) and evaluates as a number.

expression-based distribution scheme User-defined distribution scheme created by formulating a series of fragment expressions to be used by OnLine to distribute rows to fragments.

extent A continuous segment of disk space that is allocated to a tblspace (a table) in INFORMIX-OnLine Dynamic Server. The user can specify both the initial extent size for a table and the size of all subsequent extents allocated to the table.

external table A database table that is not in the current database. It may or may not be in a database managed by the same database server.

family name A quoted string constant that specifies a family name in the optical family. See *optical family*.

fast recovery An automatic, fault-tolerant feature that INFORMIX-OnLine Dynamic Server implements any time the operating mode changes from off-line to quiescent mode. The aim of fast recovery is to return OnLine to a state of physical and logical consistency with a minimal loss of work in the event of a system failure.

fault tolerance See *high availability.*

fetch The action of moving a cursor to a new row and retrieving the row values into memory.

fetch buffer A buffer in the application process that the database server uses to send fetched row data (except blob data) to the application. See also *application process.*

field See *column*.

file A collection of related information stored together on a system, such as the words in a letter or report, a computer program, or a listing of data.

file server A network node that manages a set of disks and provides storage services to computers on the network.

fixchar A character data type, available in INFORMIX-ESQL / C programs, in which the character string is fixed in length, padded with blanks if necessary, and not null-terminated.

fixed-point number A number where the decimal point is fixed at a specific place regardless of the value of the number.

filename extension The part of a filename following the period. For example, DB-Access appends the extension **.sql** to command files.

flag A command-line option, usually indicated by a minus (-) sign in UNIX systems. For example, in DB-Access the **-e** flag echoes input to the screen.

floating-point number A number with fixed precision (total number of digits) and undefined scale (number of digits to the left of the decimal point). The decimal point *floats* as appropriate to represent an assigned value.

forced residency An option that forces UNIX to keep the *resident* portion of INFORMIX-OnLine Dynamic Server shared-memory segments resident in memory, preventing UNIX from swapping out these segments to disk. (This option is not available on all UNIX systems.)

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.

fragment See *index fragment* and *table fragment*.

fragment expression The individual expressions within a range, hash, or arbitrary rule.

fragmentation An OnLine feature that enables you to define groups of rows within a table based on a rule. OnLine stores these groups or fragments in separate dbspaces that you specify when you create a table or index fragmentation strategy.

fragmentation strategy Consists of a distribution scheme (round-robin or expression), a rule or algorithm, and a location for the fragments. The rule or algorithm is OnLine-defined for a round robin-based distribution scheme and user defined for an expression-based distribution scheme.

function See *program block*.

gateway A data communications device that establishes communications between networks.

global variable A variable whose value you can access from any module or function in a program or stored procedure. See *variable* and *scope of reference*.

hash rule A user-defined algorithm that maps each row in a table to a set of integers called hash values. OnLine uses these values to determine in which fragment it will store a given row. Hash rules are usually implemented using the MOD function.

hierarchy A treelike data structure in which some groups of data are subordinate to others such that only one group (called *root*) exists at the highest level, and each group except root is related to only one parent group on a higher level.

high availability The ability of a system to resist failure and data loss. High availability includes features such as fast recovery and mirroring. It is sometimes referred to as *fault tolerance*.

highlight An rectangular inverse-video area that marks your place on the screen. A highlight often indicates the current option on a menu or the current character in an editing session. If a terminal cannot display highlighting, the current option often appears in angle brackets, and the current character is underlined.

hold cursor A cursor that is created using the WITH HOLD keywords. A hold cursor remains open past the end of a transaction. It allows uninterrupted access to a set of rows across multiple transactions.

home page The page that contains the first byte of the data row. Even if a data row outgrows its original storage location, the home page does not change. The home page contains a forward pointer to the new location of the data row. See *remainder page*.

host variable A C or COBOL 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. Identifiers for column and tables names are limited to 18 characters.

implicit connection A connection made using a database statement (DATABASE, CREATE DATABASE, START DATABASE, DROP DATABASE).

incremental archiving A system of archiving that allows the option to archive only those parts of the data that have changed since the last archive was created.

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. The user ID does not need to refer to a fully functional user account on the computer system; only the user identity components of the user account information is significant to Informix database servers. Any given user typically has the same Informix user ID on all networked computer systems involved in the database access.

Informix user password A user ID password that must be valid on all computer systems (operating systems) involved in the client's database access. When the client specifies an explicit user ID, most computer systems require the Informix user password to validate the user ID.

initialize To assign a starting value.

inmigration The process by which INFORMIX-OnLine/Optical migrates a blob from the optical storage subsystem into the INFORMIX-OnLine Dynamic Server environment.

input The information that is received from an external source (for example, from the keyboard, a file, or another program) and processed by a program.

input parameter A placeholder within a prepared SQL statement that indicates a value is to be provided at the time the statement is executed.

insert cursor A cursor for insert operations. Allows bulk insert data to be buffered in memory and written to disk.

installation The loading of software from some magnetic medium (tape, cartridge, or floppy disk) onto a computer and preparing it for use.

interactive Refers to a program that accepts input from the user, processes the input, and then produces output on the screen, in a file, or on a printer.

interquery parallelization Occurs when OnLine processes multiple queries simultaneously. A common bottleneck develops when multiple, independent queries access the same table. Fragmentation combined with interquery parallelization can effectively eliminate this bottleneck. Compare this definition with intraquery parallelization.

interrupt A signal from a user or another process that can stop the current process temporarily or permanently. See *signal*.

interrupt key A key used to cancel or abort a program or to leave a current menu and return to the menu one level above. On many systems, the CONTROL-C keypress is used for the interrupt key. On other systems, the interrupt key is DEL or CONTROL-Break.

intraquery parallelization Occurs when OnLine breaks a single query using PDQ into subqueries and then processes the subqueries in parallel. Parallel processing of this type has important implications when each subquery retrieves data from a fragment of a table. Because each partial query operates on a smaller amount of data, the retrieval time is significantly reduced. Compare this definition with interquery parallelization.

IPX/SPX Acronym for Internetwork Packet Exchange / Sequenced Packet Exchange. It refers to the NetWare network protocol by Novell.

ISAM Acronym for Indexed Sequential Access Method. An indexed sequential access method allows you to find information in a specific order or to find specific pieces of information quickly through an index. See *access method*.

ISAM error Operating system or file access error.

ISO Acronym for the International Standards Organization. ISO sets worldwide standards for the computer industry, including standards for character input and manipulation, code sets, and SQL syntax.

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. INFORMIX-SE sets a level of *no isolation* (referred to as a DIRTY READ), which cannot be changed. INFORMIX-OnLine Dynamic Server allows the programmer to choose from Informix levels of isolation or ANSI levels of isolation. See the Informix levels of isolation: DIRTY READ, COMMITTED READ, CURSOR STABILITY, and REPEATABLE READ. See the ANSI levels of isolation: *READ* UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SERIALIZABLE.

join The process of combining information from two or more tables based on some common domain of information. Rows from one table are paired with rows from another table when information in the corresponding rows match on the joining criterion. For example, if a **customer_num** column exists in the **customer** and the **orders** tables, you can construct a query that pairs each **customer** row with all the associated **orders** rows based on the common **customer_num**. See *Cartesian product* and *outer join*.

jukebox A cabinet that consists of one or more optical-disc drives, slots that store optical platters when they are not mounted, and a robotic arm that transfers platters between the slots and the drives. A jukebox is also known as an *autochanger*.

kernel The part of the UNIX operating system that controls processes and the allocation of resources.

key The pieces of information that are used to locate a row of data. A key defines the pieces of information you want to search for, as well as the order in which you want to process information in a table. For example, you can index the **last_name** column in a **customer** table to find specific customers or to process the customers in alphabetical or reverse alphabetical order, according to their last names (**last_name** serves as the key).

keyword A word that has meaning to a program. For example, the word SELECT is a keyword in SQL.

language supplement The result of the product localization process. A language supplement for a specific Western European language can be installed with an Informix product to allow the user to see error and warning messages in a language other than U.S. English. If installed with DB-Access, the menu names and options and on-line Help for that product also appear in the specified language.

latch A mechanism of shared-memory resource management that is used by INFORMIX-OnLine Dynamic Server to coordinate processes (virtual processors) as they attempt to modify entries in shared memory.

level of isolation See *isolation*.

library A collection of precompiled functions or routines that are designed to perform tasks, which are common to a particular kind of application. Your product can include library functions or routines that you can call from your programs.

link The process of combining separately compiled program modules into an executable program.

literal A character or numeric constant.

local character A character in a native language character set; also known as a *national character* or a *native character.*

local loopback A connection between the client and database server that uses a network connection even though the client and the database servers are on the same computer.

local variable A variable that has meaning only in the module in which it is defined. See *variable* and *scope of reference*.

locale The environment that defines the behavior of the program at run time by specifying a language, code set, and local customs. It usually is based on the linguistic customs and rules of the region or territory. The locale can be expressed through an environment variable setting that dictates output formats for numbers, currency symbols, dates, and time or collation order for character strings and regular expressions.

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 (available only with INFORMIX-OnLine Dynamic Server). The lock mode also can refer to the standard unit of locking (either page or row) chosen by the programmer. The lock mode is set with the SET LOCK MODE statement.

locking The process of temporarily limiting access to an object (database, table, page, or row) to prevent conflicting interactions among concurrent processes. Locks can be in either exclusive mode, which restricts read and write access to only one user; or share mode, which allows read-only access to other users. In addition, update locks exist that begin in share mode but are upgraded to exclusive mode when a row is changed.

locking granularity The size of a locked object. The size may be a database, table, page, or row.

logical log An allocation of disk space managed by OnLine that contains records of all changes that were performed on a database during the period the log was active. The logical log is used to roll back transactions, recover from system failures, and restore databases from archives.

login The process of identifying oneself to a computer.

login password See *Informix user password*.

login user ID See *Informix user ID*.

macro A named set of instructions that the computer executes whenever the name is referenced.

Memory Grant Manager (MGM) An OnLine component that coordinates the use of memory and I/O bandwidth for decision-support queries. MGM uses the DS_MAX_QUERIES, DS_TOTAL_MEMORY, DS_MAX_SCANS, and PDQPRIORITY configuration parameters to determine what resources can or cannot be granted to a decision-support query.

mantissa The significant digits in a floating-point number.

math functions See *aggregate functions*.

menu A screen display that allows you to choose the commands that you want the computer to perform.

message log The UNIX file that INFORMIX-OnLine Dynamic Server keeps to record significant events, such as checkpoints, logical log file backups, recovery data, and errors.

mirroring Storing the same data on two chunks simultaneously. If one chunk fails, the data is still usable on the other chunk in the mirrored pair. This data storage option is available with INFORMIX-OnLine Dynamic Server and is handled by the OnLine administrator.

MODE ANSI The keywords specified on the CREATE DATABASE statement to make a database ANSI-compliant.

multithreading Refers to multiple threads that are run within the same process. See *thread*.

national character See *local character.*

native character See *local character.*

native language The spoken or written language that is used in a certain region of the world, such as English or French.

NLS Acronym for Native Language Support. NLS represents an application environment that supports single-byte, 8-bit character code sets; it supports alphabetic or Latin-based languages such as those found in Europe and Latin America. NLS is also known as *Native Language System* or *National Language Support*.

null value A value representing unknown or not applicable. (A null is not the same as a value of zero or blank.)

offset A term that is used in INFORMIX-OnLine Dynamic Server to specify the physical position of a chunk on a disk. The offset is the number of kilobytes indented into the named device (which is the specified disk partition) before starting the chunk.

OLTP application Characterized by quick, indexed access to a small number of data items. The applications are typically multiuser, and response times are measured in fractions of seconds.

OLTP queries The transactions handled by OLTP applications are usually simple and predefined. A typical OLTP system is an order-entry system where only a limited number of rows are accessed many times.

OnLine instance One OnLine database server.

ON-Monitor An interface that presents a series of screens through which an INFORMIX-OnLine Dynamic Server administrator can monitor and modify an OnLine database server.

open The process of making a resource available, such as preparing a file for access, activating a cursor, or initiating a window.

operating mode The INFORMIX-OnLine Dynamic Server state of operation. The operating modes are off-line, quiescent, on-line, read-only, shutdown, and recovery.

optical cluster An amount of space, on an optical disc, that is reserved for storing a group of logically related blobs.

optical family A group of optical-discs, theoretically unlimited in number.

optical platters The removable optical disc that stores data in an optical storage subsystem.

optical statements The SQL statements that you use to control optical clustering.

optical volume One side of a removable Write-Once-Read-Many (WORM) optical disc.

outer join An asymmetric joining of a dominant and a subordinate table in a query; the joining restrictions apply only to the subordinate or *outer* table. Rows in the dominant table are retrieved without considering the join, but rows from the outer table are included only if they also satisfy the join conditions. Any dominant-table rows that do not have a matching row from the outer table receive a row of nulls in place of an outer-table row.

outmigration The process by which INFORMIX-OnLine/Optical migrates a blob from the INFORMIX-OnLine Dynamic Server environment to an optical storage subsystem.

output The result that the computer produces in response to a query or a request for a report.

owner-privileged Refers to a class of stored procedure that can be created by any user who has Resource database privileges.

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 that is used by INFORMIX-OnLine Dynamic Server. The size is fixed for a particular operating system and hardware platform, and the OnLine administrator cannot change it.

parallelism Refers to ability of OnLine to process a task in parallel by breaking the task into subtasks and then processing them simultaneously.

parameter A variable that is given a constant value for a specified application. In a subroutine, a parameter is the placeholder for the argument values that are passed to the subroutine at run time.

parent-child relationship See *referential constraint*.

parent table The referenced table in a referential constraint. See *child table*.

partition See *table fragment*.

pattern An identifiable or repeatable series of characters or symbols.

PDQ Acronym for Parallel Database Query (or Queries). An OnLine feature that permits parallel rather than sequential execution of SQL queries by distributing the execution of a single query across several processors, which enhances performance.

PDQ priority Determines the amount of resources OnLine allocates to process a query in parallel. These resources include memory, threads (such as scan threads), and sort space. The level of parallelism is established by using the **PDQPRIORITY** environment variable or various OnLine configuration parameters (including PDQPRIORITY and MAX_PDQPRIORITY) or dynamically through the SET PDQPRIORITY statement.

permission On some operating systems, the right to access files and directories.

phantom row A row of a table that is initially modified or inserted during a transaction but is subsequently rolled back. Another user can see a phantom row if the isolation level is Informix DIRTY READ or ANSI READ UNCOMMITTED. No other isolation levels allow a changed but uncommitted row to be seen.

physical log An allocation of disk space in INFORMIX-OnLine Dynamic Server that contains the before-images of all pages changed since the last checkpoint.

pointer A number that specifies the "address-in-memory" of the data or variable of interest.

precision The total number of significant digits in a real number, both to the right and left of the decimal point. For example, the number 1437.2305 has a precision of 8. See *scale*.

prepared statement An SQL statement that is generated by the PREPARE statement from a character string or from a variable containing a character string. This feature allows you to form your request while the program is executing without having to modify and recompile the program.

preprocessor A program that takes high-level programs and produces code that a standard language compiler, such as C or Micro Focus COBOL/2, can compile.

primary key The information from a column or set of columns that uniquely identifies each row in a table. The primary key sometimes is called a *unique key*.

primary-key constraint Specifies that each entry in a column or set of columns contains a non-null unique value.

printable character A character that can be displayed on a terminal, screen, or printer. Printable characters include A-Z, a-z, 0-9, and punctuation marks. Compare with *control character*.

privilege The right to use or change the contents of a database or table. See *access privileges*.

procedural statement A programming language statement that specifies actions; for example, looping and branching if a condition is met. Compare with *declaration statement*.

procedure See *program block* and *stored procedure*.

procedure cursor A cursor that is associated with an EXECUTE PROCEDURE statement. It enables you to scan multiple rows of data, moving data row by row into a set of receiving variables.

process A discrete task, generally a program, that the operating system executes.

program block A named collection of statements, such as a function, routine, or procedure, that performs a particular task; a unit of program code.

program control The actions that a computer takes, as opposed to actions that a user takes.

projection Taking a vertical subset from the columns of a single table that retains the unique rows. Projection is implemented through the select list in the SELECT clause of a SELECT statement and returns some of the columns and all the rows in a table. See *selection* and *join*.

promotable lock A lock that can be changed from a shared lock to an exclusive lock. See *update lock*.

protocol A set of rules that govern communication among computers. These rules govern format, timing, sequencing, and error control.

query A request to the database to retrieve data that meets certain criteria, usually with the SELECT statement.

query optimization information statements The SQL statements that are used to optimize queries. These statements include SET EXPLAIN, SET OPTIMIZATION, and UPDATE STATISTICS.

range rule A user-defined algorithm that you use to define the boundaries of each fragment in a table using SQL relational and logical operators. Expressions in a range rule can use the following restricted set of operators: >, <, >=, <=, and the logical operator AND.

raw device A UNIX disk partition that is defined as a character-special device, which is not mounted. It can be used for INFORMIX-OnLine Dynamic Server disk space to increase performance with UNIX files.

READ COMMITTED An ANSI level of isolation available through INFORMIX-OnLine Dynamic Server and set with the SET TRANSACTION statement in which a user can view only rows that are currently committed at the moment of the query request; that is, a user cannot view rows that were changed as a part of a currently uncommitted transaction. It is the default level of isolation under OnLine for databases that are not ANSI-compliant. See *isolation*.

READ UNCOMMITTED An ANSI level of isolation set with the SET TRANSACTION statement that does not account for locks and allows viewing of any existing rows, even rows that currently can be altered from inside an uncommitted transaction. READ UNCOMMITTED is the lowest level of isolation (no isolation at all). See *isolation*.

real user ID See *Informix user ID*.

record See *row*.

recover a database To restore a database to a former condition after a system failure or other destructive event. The recovery restores the database as it existed immediately before the crash. This is sometimes referred to as *restore a database*.

referential constraint Defines the relationship between columns within a table or between tables; also known as a *parent-child relationship*. Referencing columns also are known as *foreign keys*.

relation See *table*.

relational database A database that uses a table structure to store data. Each table consists of a set of fixed-length rows divided into columns.

remainder page An additional page that is filled with data from a single row. INFORMIX-OnLine Dynamic Server uses remainder pages when the data for a row cannot fit in the initial page. Remainder pages are added and filled as needed. The original page entry contains pointers to the remainder pages. See *home page*.

remote A connection that requires a network.

REPEATABLE READ An Informix and ANSI level of isolation available through OnLine with the Informix SET ISOLATION statement or ANSI SET TRANSACTION statement, which ensures all data read during a transaction is not modified during the entire transaction. Transactions under ANSI REPEATABLE READ are also known as SERIALIZABLE. Informix REPEATABLE READ is the default level of isolation under OnLine for ANSI-compliant databases. See *isolation* and *SERIALIZABLE*.

reserved word A word in a statement or command that you cannot use in any other context of the language or program without receiving a warning or error message.

restore a database See *recover a database*.

roll back The process that reverses an action or series of actions on a database. The database is returned to the condition that existed before the actions were executed. See *transaction*.

roll forward The process that brings a database up to date. This process usually takes place when a database is recovered after a system crash or other failure. In INFORMIX-SE, an archive copy of the database is restored to the disk, and then the database is rolled forward to a point just before the failure. In INFORMIX-OnLine Dynamic Server, an archive copy of the database is restored to the disk, and then the logical log records are rolled forward to a point just before the failure.

root dbspace The initial dbspace for an INFORMIX-OnLine Dynamic Server system. In addition to any data, the root dbspace contains reserved pages and internal tables that track OnLine storage, recovery, and consistency information.

round-robin distribution scheme An OnLine-defined distribution scheme. Online distributes rows in such a way that the number of rows in each of the fragments remains approximately the same.

routine See *program block*.

row A group of related items of information about a single entity in a database table. In a table of customer information, for example, a row contains information about a single customer. A row is sometimes referred to as a *record* or *tuple*. (In a screen form, a row can refer to a line of the screen.)

rowid In nonfragmented tables, rowid refers to an integer that defines the physical location of a row. Rowids must be explicitly created to be used in fragmented tables and they do not define a physical location for a row. Rowids in fragmented tables are accessed by an index that is created when the rowid is created; this access method is slow. Informix recommends that users creating new applications move toward using primary keys as a method of row identification instead of using rowids.

rule How OnLine or the user determines into which fragment rows are placed. OnLine determines the rule for a round robin-based distribution scheme. The user determines the rule for an expression-based distribution scheme.

run-time environment The hardware and operating system services available at the time a program runs.

run-time errors Errors that occur during program execution. Compare with *compile-time errors*.

scale The number of digits to the right of the decimal place in DECIMAL notation. The number 1437.2350 has a scale of 4 (four digits to the right of the decimal point). See *precision*.

scan thread An OnLine thread that is assigned the task of reading rows from a table. When a query is executed in parallel, OnLine allocates multiple scan threads to perform the query in parallel.

schema The structure of a database or a table. The schema for a table lists the names of the columns, their data types, and (where applicable) the lengths, indexing, and other information about the structure of the table.

scope of reference The portion of a program in which an identifier applies and can be accessed. Three possible scope sizes exist: local (an identifier applies only within a single program block), modular (the identifier applies throughout a single module), and global (an identifier applies throughout the entire program).

scroll cursor A cursor created with the SCROLL keyword that allows you to fetch rows of the active set in any sequence.

secure auditing A facility of the INFORMIX-OnLine Dynamic Server that lets OnLine administrators keep track of unusual or potentially harmful user activity. Use the **onaudit** utility to enable auditing of events and create audit masks, and the **onshowaudit** utility to extract the audit event information for analysis.

select See *query*.

select cursor A cursor associated with the SELECT statement. It enables you to scan multiple rows of data, moving data row by row into a set of receiving variables.

selection Taking a horizontal subset of the rows of a single table that satisfies a particular condition. Selection is implemented through the WHERE clause of a SELECT statement and returns some of the rows and all of the columns in a table. See *projection* and *join*.

self-join A join between a table and itself. A self-join occurs when a table is used two or more times in a SELECT statement (with different aliases) and joined to itself.

semaphore A UNIX communication device that signals a process to awaken.

SERIALIZABLE An ANSI level of isolation available through INFORMIX-OnLine Dynamic Server and set with the SET TRANSACTION statement, ensuring all data read during a transaction is not modified during the entire transaction. ANSI SERIALIZABLE is the default level of isolation under OnLine for an ANSI-compliant database that is set with the SET TRANSACTION statement.

server name The unique name of a database server. The database server name is used by an application to select a database server. It is assigned by the administrator of the database server.

server number A unique number between 0 and 255, inclusive, that the INFORMIX-OnLine Dynamic Server administrator assigns to an OnLine database server at the time of initialization. If more than one OnLine database server is installed on the same computer, each database server must have a unique number.

session The structure that is created for an application using INFORMIX-OnLine Dynamic Server.

set functions See *aggregate functions*.

shared lock A lock that more than one thread can acquire on the same object. Shared locks allow for greater concurrency with multiple users; if two users have locks on a row, a third user cannot change the contents of that row until both users (not just the first) release the lock. Shared-locking strategies are used in all levels of isolation except Informix DIRTY READ and ANSI READ UNCOMMITTED.

shared memory Memory that is accessible to multiple processes. Shared memory allows multiple processes to access a common data space in memory. Common data does not have to be reread from disk for each process, reducing disk I/O and improving performance. Also used as an IPC mechanism.

shelf The location of an optical platter that is neither on an optical drive nor in a jukebox slot.

shuffling Shuffling refers to the process that occurs when OnLine moves rows or key values from one fragment to another. Shuffling occurs in a variety of circumstances including when you attach, detach, or drop a fragment.

signal A means of asynchronous communication between two processes. For example, signals are sent when a user or a program wants to interrupt or suspend the execution of a process. See *interrupt*.

singleton select A SELECT statement that returns a single row.

SMI Acronym for *system monitoring interface*. SMI is a collection of tables in the **sysmaster** database that maintains dynamically updated information about the operation of an INFORMIX-OnLine Dynamic Server database server.

source file A file containing instructions (in ASCII text) that is used as the source for generating compiled programs.

SPL Acronym for Stored Procedure Language. See *stored procedure*.

SQL Acronym for Structured Query Language. SQL is a database query language that was developed by IBM and standardized by ANSI. Informix relational database management products are based on an extended implementation of ANSI-standard SQL.

SQL API An SQL application programming interface that includes a library of callable functions, which are used to develop an application that accesses a relational database. Examples include the embedded-language products such as INFORMIX-ESQL / C and INFORMIX-ESQL / COBOL. See *host variable*.

SQLCA Acronym for SQL Communications Area. The SQLCA is a data structure that stores information about the most recently executed SQL statement. The result code returned by the database server to the SQLCA is used for error handling by Informix SQL APIs.

sqlda Acronym for SQL descriptor area. A dynamic SQL management structure that can be used with the DESCRIBE statement to store information about database columns or host variables used in dynamic SQL statements. The structure contains an **sqlvar_struct** structure for each column; each **sqlvar_struct** structure provides information such as the name, data type, and length of the column. The sqlda structure is an Informix-specific structure for handling dynamic columns. It is available only within an INFORMIX-ESQL/C program. See also *descriptor,* and *system-descriptor area*.

sqlexecd A daemon process that receives a connection request from a client and spawns an INFORMIX-SE database server process.

staging-area blobspace The blobspace where INFORMIX-OnLine Dynamic Server temporarily stores a blob that is being outmigrated to an optical storage subsystem.

statement A line, or set of lines, of program code that describes a single action (for example, a SELECT statement or an UPDATE statement).

statement block A section of a program that usually begins and terminates with special symbols such as *begin* and *end*. 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. It 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 procedure A function that is used along with SQL statements in an Informix program. Stored procedures are written using SQL and Stored Procedure Language (SPL) statements. The procedure is stored in the database in executable form.

string A set of characters (generally alphanumeric) that is manipulated as a single unit. A string might consist of a word (such as 'Smith'), a set of digits representing a number (such as '19543'), or any other collection of characters. Strings generally are surrounded by single quotes. A string is also a character data type, available in INFORMIX-ESQL/C programs, in which the character string is stripped of trailing blanks and is null-terminated.

subordinate table See *outer join*.

subquery A query that is embedded as part of another SQL statement. For example, an INSERT statement can contain a subquery in which a SELECT statement supplies the inserted values in place of a VALUES clause; an UPDATE statement can contain a subquery in which a SELECT statement supplies the updating values; or a SELECT statement can contain a subquery in which a second SELECT statement supplies the qualifying conditions of a WHERE clause for the first SELECT statement. (Parentheses always delimit a subquery, unless you are referring to a CREATE VIEW statement or unions.)

subscript Delimiters that indicate the start and end positions of a substring.

substring A portion of a character string.

synonym A name that is assigned to a table and used in place of the original name for that table. A synonym does not replace the original table name; instead, it acts as an alias for the table.

sysmaster database A database that every INFORMIX-OnLine Dynamic Server database server contains. It holds the ON-Archive catalog tables and system monitoring interface (SMI) tables.

system call A call to a function provided by the operating system.

system catalog A group of database tables that contain information about the database itself, such as the names of tables or columns in the database, the number of rows in a table, the information about indexes and database privileges, and so on. See *data dictionary.*

system-descriptor area A dynamic SQL management structure that is used with the ALLOCATE DESCRIPTOR, DEALLOCATE DESCRIPTOR, DESCRIBE, GET DESCRIPTOR, and SET DESCRIPTOR statements to store information about database columns or host variables used in dynamic SQL statements. The structure contains an item descriptor for each column; each item descriptor provides information such as the name, data type, length, scale, and precision of the column. The system-descriptor area is the X/Open standard for handling dynamic columns. See *descriptor* and *sqlda*.

system monitoring interface See *SMI*.

table A rectangular array of data in which each row describes a single entity and each column contains the values for each category of description. For example, a table can contain the names and addresses of customers. Each row corresponds to a different customer and the columns correspond to the name and address items. A table sometimes is referred to as a *relation*.

table fragment Consists of zero or more rows that are grouped together and stored in a dbspace that you specify when you create the fragment.

tblspace The logical collection of extents that are assigned to a table under INFORMIX-OnLine Dynamic Server.

TCP/IP The specific name of a particular standard transport layer protocol (TCP) and network layer protocol (IP). A popular network protocol used in DOS, UNIX, and other environments.

temporary An attribute of any file, index, or table that is used only during a single session. Temporary files or resources are typically removed or freed when program execution terminates or an on-line session ends.

thread A single sequential flow of control that represents a unit of work within a multithreaded process.

timeout The point at which a lock request is aborted because the requesting thread waited longer for the lock than the specified maximum time limit. A program developer can set a time limit in INFORMIX-OnLine Dynamic Server through the SET LOCK MODE statement.

TLI Acronym for Transport Level Interface. It is the interface designed for use by application programs that are independent of a network protocol.

trace To keep a running list of the values of program variables, arguments, expressions, and so on, in a program or stored procedure.

transaction A collection of one or more SQL statements that is treated as a single unit of work. If one statement in a transaction fails, the entire transaction can be *rolled back* (canceled). If the transaction is successful, the work is *committed* and all changes to the database from the transaction are accepted.

transaction logging The process of keeping records of transactions. See *logical log*.

trigger A mechanism that resides in the database. It specifies that when a particular action (insert, delete, or update) occurs on a particular table, the database server should automatically perform one or more additional actions.

tuple See *row*.

unique constraint Specifies that each entry in a column or set of columns has a unique value.

unique key See *primary key*.

UNIX real user ID See *Informix user ID*.

unlock To free an object (database, table, page, or row) that has been locked. For example, a locked table prevents others from adding, removing, updating, or (in the case of an exclusive lock) viewing rows in that table as long as it is locked. When the user or program unlocks the table, others are permitted access again.

update The process of changing the contents of one or more columns in one or more existing rows of a table.

update lock A promotable lock that is acquired during a SELECT...FOR UPDATE. An update lock behaves like a shared lock until the update actually occurs, and it then becomes an exclusive lock. It differs from a shared lock in that only one update lock can be acquired on an object at a time.

user ID See *Informix user ID*.

user ID password See *Informix user password*.

user name See *Informix user ID*.

user password See *Informix user password*.

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.

virtual column A derived column of information that is not stored in the database. For example, you can create virtual columns in a SELECT statement by arithmetically manipulating a single column, such as multiplying existing values by a constant, or by combining multiple columns, such as adding the values from two columns.

virtual processors The multithreaded processes that make up the INFORMIX-OnLine Dynamic Server database server. Virtual processors service a large number of clients by functioning in a way that is analogous to the way that hardware processors function in the computer.

warning A state or situation that is detected by the database server or compiler, possibly incorrect syntax or a problem. A warning does not necessarily affect the ability of the code to run.

wildcard A special symbol that represents any sequence of zero or more characters or any single character. In SQL, for example, you can use the asterisk (*), question mark (?), brackets ([ ]), percent sign (%), and underscore (_) as wildcard characters. (The asterisk, question mark, and brackets are also wildcards in UNIX.)

window A rectangular area on the screen in which you can take actions without leaving the context of the background program.

WORM Acronym for Write-Once-Read-Many optical media. When a bit of data is written to a WORM platter, a permanent mark is made on the platter.

X/Open An independent consortium that produces and develops specifications and standards for open-systems products and technology such as dynamic SQL and NLS.

X/Open Portability Guide A set of specifications which vendors and users can use to build portable software. Any vendor carrying the XPG brand on any particular software product is guaranteeing that the software correctly implements the X/Open Common Applications Environment (CAE) specifications. There are CAE specifications for SQL, XA, ISAM, RDA, and so on.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

Q

R

S

T

U