Guide to 7.1 Feature Enhancements

Version 7.1
May 1995
Part No. 000-7748

Published by INFORMIX® Press

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

Documentation Team: Signe Haugen, Geeta Karmarkar, Steve Klitzing, Susan Koehler, Mary Kraemer, Dawn Maneval, Tom Noronha, Eileen Wollam

Preface

The *Guide to Version 7.1 Feature Enhancements* supplements various books in the Informix Version 7.1UC1 manual set. It describes how to use the enhancements in the Informix Version 7.1UD1 product set, using both INFORMIX-OnLine Dynamic Server and INFORMIX-SE.

This guide assumes that you have database management experience and are familiar with relational database concepts. It also assumes that you have knowledge of Structured Query Language (SQL). The Informix implementation of SQL is described in detail in a set of Version 7.1 manuals called the *Informix Guide to SQL: Tutorial*, the *Informix Guide to SQL: Reference*, and the *Informix Guide to SQL: Syntax*.

You must have the following Informix software:

- Version 7.1UD1 INFORMIX-OnLine Dynamic Server or INFORMIX-SE

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

- An SQL application programming interface (API) such as INFORMIX-ESQL/C or INFORMIX-ESQL/COBOL, or the DB-Access utility that is shipped as part of your OnLine or SE database server.

  ESQL/C and ESQL/COBOL let you compose queries, send them to the database server, and view the results that the OnLine or SE database server returns. You also can use DB-Access to try out most of the new SQL statements and branches described in this guide.

Summary of Chapters

The *Guide to Version 7.1 Feature Enhancements* includes the following chapters:

- This Preface provides general information about the guide.
- The Introduction tells how this guide fits into the Informix family of products and books, explains how to use this guide, introduces the demonstration database from which the product examples are drawn, and lists the enhanced features for Version 7.1 of Informix database server products.
- Chapter 1, "OnLine Enhancements," describes how to effectively configure the OnLine using the enhanced set of configuration parameters. This chapter also describes five configuration parameters that OnLine now configures dynamically rather than statically.
- Chapter 2, "Connectivity Enhancements," describes changes to client / server connectivity that include enhancements to the **sqlhosts** file and a new stream-pipe connection type.
- Chapter 3, "SQL Enhancements," describes and illustrates changes to SQL statement syntax that support several new features including object modes and violation detection, fragment authorization, and parallel loads. Also described are changes to SQL that support compliance with the XPG4 standard and that ease the gathering of database statistics.
- Chapter 4, "ON-Archive Feature Enhancements," describes four new qualifiers that you can use with ON-Archive commands. It also explains how to execute a specific request with **onautovop**, automatic execution of **oncatlgr**, and improved user notification for events like logs-full, archive, backup, and restore. This chapter also contains the complete syntax for these enhancements.
- Chapter 5, "SQL API Enhancements," describes changes to the ESQL preprocessor and a new function that returns the names of databases managed by a given database server.
- Chapter 6, "DB-Access Enhancements," describes additions and changes to the DB-Access menus and screens to support the new USER clause of the CONNECT statement.

- "Error Messages" contains a list of the error messages and corrective actions that might appear when you work with Version 7.1UD1 products.
- The Index includes references throughout the *Guide to Version 7.1 Feature Enhancements.*

Informix Welcomes Your Comments

To help us improve future versions of this guide, please send us your comments, corrections, and suggestions. You can contact us in the following ways:

- Send a FAX to the Informix Technical Publications Department at (415) 926-6571.
- Send electronic mail to doc@informix.com.

We appreciate your feedback.

Related Reading

If you want additional technical information on database management, consult the following texts by C. J. Date:

- *An Introduction to Database Systems, Volume I* (Addison-Wesley Publishing, 1990)
- *An Introduction to Database Systems, Volume II* (Addison-Wesley Publishing, 1983)

This guide 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 guide.

Some suggested texts about UNIX systems follow:

- *A Practical Guide to the UNIX System*, Second Edition, by M. Sobell (Benjamin / Cummings Publishing, 1989)
- *A Practical Guide to UNIX System V* by M. Sobell (Benjamin / Cummings Publishing, 1985)

Table of Contents

Introduction

The enhancements contained in this release of Informix software enable you to perform the following tasks:

- Reserve logical log space for administrative tasks
- Control how OnLine reacts to administrative tasks
- Load and unload data in parallel
- Enable or disable triggers, indexes, and constraints
- Detect and analyze constraint violations
- Control access to data using roles
- Automatically audit transactions

For a complete listing of the new features in this release, see "New Features in Informix Version 7.1UD1 Products" on page 18 of this Introduction.

Informix Products Covered in this Guide

Informix produces a number of application development tools and application programming interfaces (APIs) that use Structured Query Language (SQL). Application development tools currently available include INFORMIX-NewEra and INFORMIX-4GL. SQL APIs include INFORMIX-ESQL/C and INFORMIX-ESQL/COBOL.

The information presented in this guide is valid for the following products and versions:

- INFORMIX-OnLine Dynamic Server, Version 7.1UD1, and its utilities, such as DB-Access
- INFORMIX-ESQL/C, Version 7.1UD1
- INFORMIX-ESQL/COBOL, Version 7.1UD1
- INFORMIX-SE, Version 7.1UD1

Other Useful Documentation

You might want to refer to a number of related Informix documents that the *Guide to 7.1 Feature Enhancements* supplements:

- Depending on the database server that you are using, you or your system administrator need either the *INFORMIX-OnLine Dynamic Server Administrator's Guide* and the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* or the *INFORMIX-SE Administrator's Guide*.
- The *DB-Access User Manual* describes the menus and screens for this database server utility, which lets you create databases and tables and issue SQL statements.
- The *Informix Error Messages* manual lists error messages and corrections for Version 7.1 and earlier. If you prefer, you can look up the error messages in the on-line message file described in that manual. Error messages new to this release are listed in the OnLine Error Messages section at the end of this supplement.
- The *Informix Guide to SQL: Tutorial* leads you through basic database design and implementation concepts. If you have never used SQL or an Informix SQL API before, you might want to read it.
- The *Informix Guide to SQL: Reference* provides full information on the structure and contents of the demonstration database that is provided with all Informix application development tools. It includes details of the Informix system catalog, describes Informix and common UNIX environment variables that should be set, and defines column data types supported by Informix products. It also contains a glossary of terms.
- The *Informix Guide to SQL: Syntax*, provides a detailed description of all of the SQL statements supported by Informix Version 7.1 and earlier products.
- You might also want to refer to the manual for your SQL API, such as the *INFORMIX-ESQL/C Programmer's Manual* or the *INFORMIX-ESQL/COBOL Programmer's Manual*.

- The *INFORMIX-OnLine Dynamic Server Migration Guide* discusses how to convert to or from Version 7.1 of OnLine.
- In addition to the *Guide to 7.1 Feature Enhancements*, you, or whoever installs your Informix products, should refer to the Version 7.1 *UNIX Products Installation Guide,* Rev. B, to ensure that your Informix product is properly set up before you begin to work with it. The *Installation Guide* contains a complete list of the Documentation Notes files for the 7.1 release. A matrix depicting possible client / server configurations is also included in the *Installation Guide*.

Throughout this guide, references to Informix manuals are to Version 7.1 unless otherwise noted.

How to Use This Guide

This guide assumes that you are using the INFORMIX-OnLine Dynamic Server. The following sections describe the conventions used in this guide for typographical and command-line formats, SQL statement 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 forth. The following typographical conventions are used throughout this guide:

italics	New terms, emphasized words, and variables are printed in italics.
boldface	Database names, table names, column names, filenames, utilities, and other similar terms are printed in boldface.
`computer`	Information that the product displays and information that you enter are 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 *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.

Syntax Conventions

Syntax diagrams describe the format of SQL statements or commands, including alternative forms of a statement, required and optional parts of the statement, and so forth. Syntax diagrams have their own conventions, which are defined in detail and illustrated in this section. SQL statements for Version 7.1 and earlier are listed in their entirety in Chapter 1 of the *Informix Guide to SQL: Syntax*. New SQL statements appear in Chapter 3, "SQL Enhancements," in this guide.

Each syntax diagram displays the sequences of required and optional elements that are valid in a statement or command. Briefly:

- All keywords are shown in uppercase letters for ease of identification, though you need not enter them that way.
- Words for which you must supply values are in italics.

Each diagram begins at the upper left with a keyword and 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 statement. Except for separators in loops, which the path approaches counterclockwise from the right, the path always approaches elements from the left and continues to the right.

Along a path, you might encounter the following elements:

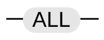
KEYWORD	You must spell a word in uppercase letters exactly as shown; however, you can use either uppercase or lowercase letters when you enter it.
(.,;+*-/)	Punctuation and mathematical notations are literal symbols that you must enter exactly as shown.
' '	Single quotes are literal symbols that you must enter as shown.
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.
ADD Clause p. 5-28	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.
Relational Operator *see* SQLS	A reference to SQLS in this guide represents an SQL statement or segment described in Chapter 1 of the *Informix Guide to SQL: Syntax*. Imagine that the statement or segment is spliced into the main diagram at this point.
ESQL	A code in an icon is a signal warning you that this path is valid only for some products or under certain conditions. The codes indicate the products or conditions that support the path. The following codes are used:

OL	This path is valid only for INFORMIX-OnLine Dynamic Server.
SE	This path is valid only for INFORMIX-SE.
D/B	This path is valid only for DB-Access.
ESQL	This path is valid for INFORMIX-ESQL/C and INFORMIX-ESQL/COBOL.
E/C	‘This path is valid only for INFORMIX-ESQL/C.
E/CO	This path is valid only for INFORMIX-ESQL/COBOL.
SPL	This path is valid only if you are using Informix Stored Procedure Language (SPL).

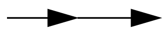
NLS This path is valid only if you have created your database as a Native Language Support (NLS) database.

OP This path is valid only for INFORMIX-OnLine / Optical.

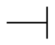
+ This path is an Informix extension to ANSI SQL-92 entry level standard SQL. If you initiate Informix extension checking and include this syntax branch, you receive a warning. If you have set the DBANSIWARN environment variable at compile time, or have used the **-ansi** compile flag, you receive warnings at compile time. If you have DBANSIWARN set at run time, or if you compiled with the **-ansi** flag, warning flags are set in the **sqlwarn** structure.



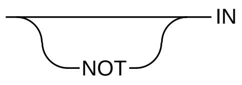
A shaded option is the default. Even if you do not explicitly type the option, it will be in effect unless you choose another option.



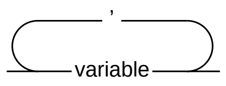
Syntax enclosed in a pair of arrows indicates that this is a subdiagram.



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



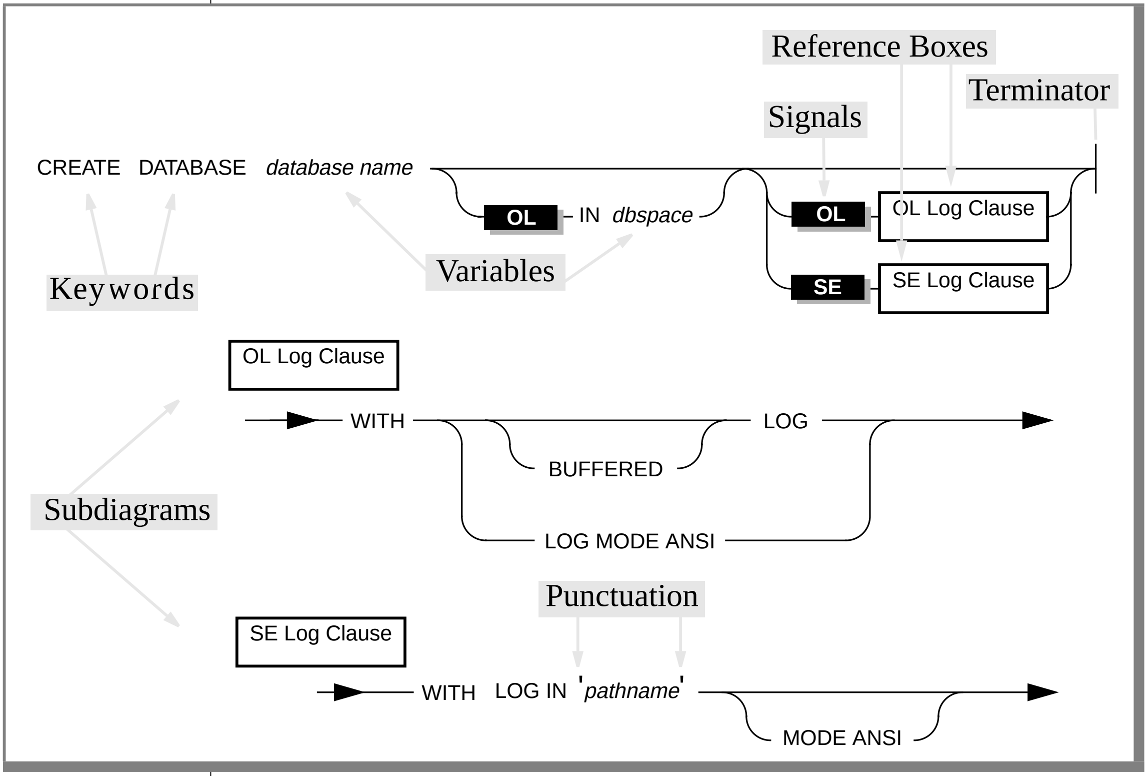
A branch below the main line indicates an option



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

Figure 1 shows the elements of a syntax diagram for the CREATE DATABASE statement. Many syntax diagram conventions are illustrated.

Figure 1
Elements of a syntax diagram



To construct a statement using Figure 1, start at the top left with the keywords CREATE DATABASE. Then follow the diagram to the right, proceeding through the options that you want. Figure 1 conveys the following information:

1. You must type the words CREATE DATABASE.
2. You must supply a *database name*.
3. You can stop, taking the direct route to the terminator, or you can take one or more of the optional paths.

4. If desired, you can designate a dbspace by typing the word IN and a dbspace name.
5. If desired, you can specify logging. Here, you are constrained by the database server with which you are working.
    - ❑ If you are using INFORMIX-OnLine Dynamic Server, go to the subdiagram named *OL Log Clause*. Follow the subdiagram by typing the keyword WITH, then choosing and typing either LOG, BUFFERED LOG, or LOG MODE ANSI. Then, follow the arrow back to the main diagram.
    - ❑ If you are using INFORMIX-SE, go to the subdiagram named *SE Log Clause*. Follow the subdiagram by typing the keywords WITH LOG IN, typing a quote, supplying a pathname, and closing the quotes. You can then choose the MODE ANSI option below the line or continue to follow the line across.
6. Once you are back at the main diagram, you come to the terminator. Your CREATE DATABASE statement is complete.

Icons in the Text

In the statement descriptions in this guide, icons that appear in the left margin indicate that the text located between the diamond symbols (♦) is valid only for a specific product or products, or under certain conditions. In addition to the icons described on page 7 of this Introduction, the following icons can appear in the left margin. These icons indicate material that is relevant under specific conditions:

ANSI	This icon indicates that the functionality described in the text that is located between the diamond symbols (♦) is valid only if your database is ANSI-compliant.
X/O	This icon indicates that the functionality described in the text that is located between the diamond symbols (♦) conforms to X/Open specifications for dynamic SQL. This functionality is available when you compile your SQL API with the **-xopen** flag.

Example Code Conventions

Examples of SQL code occur throughout this guide. 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 following example code:

```
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.

Command-Line Conventions

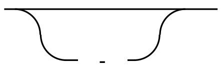
Command-line options are commands that you enter at the operating system prompt to perform certain Informix functions or go to specified menus in DB-Access. Valid command-line options for DB-Access are illustrated in a diagram in Chapter 1 of the *DB-Access User Manual*.

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

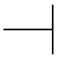
Each diagram displays the sequences of required and optional elements that are valid in a command. A diagram begins at the upper left with a command. It ends at the upper right with a vertical line. Between these points, you can trace any path that does not stop or back up. Each path describes a valid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

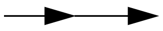
command	This required element is usually the product name or other short word used to invoke the product or call the compiler or preprocessor script for a compiled Informix product. It might appear alone or precede one or more options. You must spell a command exactly as shown and must use lowercase letters.
variable	A word in italics represents a value that you must supply, such as a database, file, or program name. The nature of the value is explained immediately following the diagram.
-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.
$\boxed{\text{Privileges p. 1-17}}$	A reference in a box represents a subdiagram on the same page or another 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.



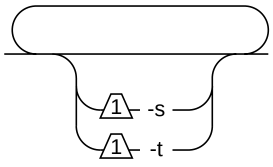
A branch below the main line indicates an optional path.



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



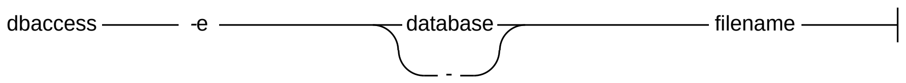
Commands enclosed in a pair of arrows indicate that this is a subdiagram.



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

Figure 2 shows the elements of a DB-Access command used to echo file input to the screen.

Figure 2
Elements of a command-line diagram



To construct a similar command, start at the top left with the command `dbaccess`. Then follow the diagram to the right, including the elements that you want. This diagram conveys the following information:

1. You must type the word `dbaccess`.
2. You can echo the SQL statements in a command file to the screen by typing the flag `-e` before the database name.
3. You must supply a ***`database`*** name or use a hyphen (-) to indicate that a database name is specified in the command file that you want to run.
4. You must specify the ***`filename`*** of a command file whose SQL statements you want to echo to the screen.

On some command-line diagrams, you can take the direct route to the terminator, or you can take an optional path indicated by a branch below the main line.

Once you are back at the main diagram, you come to the terminator. Your `dbaccess` command is complete. Press RETURN to execute the command.

Useful On-Line Files

The following on-line files, which are located in the **$INFORMIXDIR/release** directory, might supplement the information in this guide and its accompanying documents:

Documentation Notes	describe features that are not covered in the *Guide to 7.1 Feature Enhancements* or *UNIX Products Installation Guide*, or that have been modified since publication. The files that contain the Documentation Notes for these documents are **SUPPDOC_7.1** and **INSTALLDOC_7.1**. A complete list of the Documentation Notes files for the 7.1 release appears in the *UNIX Products Installation Guide*, Rev. B.
Release Notes	describe feature differences from earlier versions of Informix products and how these differences might affect current products. The file that contains the Release Notes for Version 7.1 of Informix database server products is called **SERVERS_7.1**.
Machine Notes	describe any special actions that are required to configure and use Informix products on your computer. Machine Notes are named for the product described. The Machine Notes file for INFORMIX-OnLine Dynamic Server is **ONLINE_7.1**. The Machine Notes file for INFORMIX-SE is **SE_7.1.** The Machine Notes file for INFORMIX-ESQL/C is **ESQLC_7.1**. The Machine Notes file for INFORMIX-ESQL/COBOL is **ESQLCOB_7.1**.

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

A number of Informix products also provide on-line Help files that walk you through each menu option. To invoke the Help feature, simply press CTRL-W wherever you are in your Informix product.

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.

Error messages for the 7.1 feature enhancements described in this guide are included at the end of this guide.

The Demonstration Database

The DB-Access utility, which is provided with your Version 7.1 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 of the examples in this guide are based on the **stores7** demonstration database. The **stores7** database is described in detail and its contents are listed in Appendix A of the *Informix Guide to SQL: Reference*.

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**. Follow these rules for naming your database:

- Names for databases can be up to 18 characters long for OnLine databases and up to 10 characters long for SE databases.
- The first character of a name must be a letter.
- 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 should 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 installed your Informix database server product according to the installation instructions, the files that make up 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 clean 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 of the *Informix Guide to SQL: Reference*.)
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 one of the following commands:

   To create the database without logging enter:

   ```
   dbaccessdemo7 dbname
   ```

   To create the demonstration database with logging enter:

   ```
   dbaccessdemo7 -log dbname
   ```

   If you are using OnLine, by default the data for the demonstration database is stored in 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 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 Dynamic Server 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 access to the database that you have created, grant them the appropriate privileges using the GRANT statement in DB-Access. 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 X3.135-1992), which is identical to ISO 9075:1992 on INFORMIX-OnLine Dynamic Server. In addition, many features of OnLine comply with Intermediate and Full Level SQL-92 and with X/Open SQL CAE (common applications environment) specifications.

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

In addition, Informix SQL-based products have been enhanced to comply with the X/Open SQL CAE specifications.

New Features in Informix Version 7.1UD1 Products

In addition to the following list, a comprehensive listing of all the new features for this release of Informix products is found in the Release Notes. The file that contains the Release Notes for Version 7.1UD1 Informix products is called **SERVERS_7.1**.

The following section highlights the major new features implemented in this release of Informix products.

Changes to INFORMIX-OnLine Dynamic Server

The following new OnLine features are supported in a stand-alone or network environment:

- Elimination of USERTHREADS, TRANSACTIONS, TBLSPACES, CHUNKS, and DBSPACES configuration parameters

  Beginning with this release, OnLine dynamically allocates the resources that these configuration parameters once controlled.
- Introduction of three new configuration parameters: LBU_PRESERVE, ONDBSPDOWN, and OPTCACHEMAX
- Ability to process certain types of inserts in parallel
- Extended command-line syntax for the **onstat** utility
- New messages in the OnLine message log
- New ON-Archive functionality that includes four new qualifiers, automatic execution of **oncatlgr** by **onautovop** and ON-Archive, and improved user notification of events such as logs full
- Enhancements to the OnLine secure features
- New default value for OPTCOMPIND
- Location of the auditing configuration parameters ADTMODE, ADTERR, ADTPATH, and ADTSIZE changed from the **$ONCONFIG** file to the auditing configuration file, **adtcfg**
- Additional columns in four SMI tables

See Chapter 1, "OnLine Enhancements," of this guide for more information about enhanced configuration parameters. See Chapter 4, "ON-Archive Feature Enhancements," for more information about archiving.

Changes to Connectivity

The following connectivity features are new or changed in this release:

- Stream-pipe connections for local OnLine connections
- Enhancements to the **sqlhosts** file
  - Wildcards and explicit addresses in the **hostname** and **servicename** fields
  - A fifth field (the **options** field) used to set additional options
- The **INFORMIXSQLHOSTS** environment variable
- **/INFORMIXTMP** directory for generated files
- Limitation of log-file size for SE

See Chapter 2, "Connectivity Enhancements," for more information about connectivity.

Changes to SQL

The following SQL features are new or changed in this release:

- You can specify the object mode of constraints, indexes, and triggers. You can also create special tables to detect integrity violations.
- You can create, drop, and enable roles. You can also grant and revoke privileges to roles.
- You can grant and revoke privileges on individual fragments of tables.
- You can change the user name under which database operations are performed in the current session.
- You can rename databases.
- You can suppress the construction of index information in the MEDIUM and HIGH modes of the UPDATE STATISTICS statement.

- You can use the new aggregate functions RANGE, STDEV, and VARIANCE.
- New environment variables have been added, and some existing environment variables have been modified.
- New system catalog tables have been added, and some existing tables have been modified.
- Some SQL utilities have been modified.
- The **sqlwarn** array within the SQL Communications Area (SQLCA) has been modified.
- The ANSI flagger used by Informix products has been modified to eliminate the flagging of certain SQL items as Informix extensions.
- Some SQL items have been modified to provide enhanced compliance with the *X/Open Portability Guide 4* (XPG4) specification for SQL.
- Some SQL statements have been modified to support new functionality in this release.
- The SET CONSTRAINTS statement has been dropped, and its functionality has been merged into the new SET statement.

See Chapter 3, "SQL Enhancements," for more information on SQL.

Changes to SQL APIs

- ESQL/C and ESQL/COBOL can parse and support new SQL syntax for fragmentation authorization, object modes, and violation detection.
- ESQL/COBOL supports fragmentation authorization, object modes, and violation-detection syntax on both MF2 and RM-85 systems.
- ESQL/C supports a new function, **sqgetdbs()**, for obtaining databases in the current database server.
- The **sqlwarn1** and **sqlwarn3** flags in the SQLCA structure can contain a `W` after a GRANT or REVOKE statement executes.

See Chapter 5, "SQL API Enhancements," of this guide for more information.

Changes to DB-Access

The DB-Access user interface bundled with OnLine supports the USER clause of the CONNECT statement.

DB-Access supports the USER clause of the CONNECT statement in two main ways:

- Through new screens and menus accessed through the CONNECTION and SQL menus
- Through the CONNECT...USER statement syntax entered through the Query-language menu option

DB-Access includes the following new screens and menus for fragmentation:

- A new USER NAME screen for entering the name to use in the USER clause of the CONNECT statement
- A new PASSWORD screen for entering a user password to use in the USING clause associated with the USER clause of the CONNECT statement

See Chapter 6, “DB-Access Enhancements,” for more information.

Other Changes

Version 7.1 of Informix products includes new and modified error messages and corrections. For more information, see the “Error Messages” section at the end of this guide.

Chapter

OnLine Enhancements

1

This chapter describes the configuration parameters and utility options that are new or modified in the current release of INFORMIX-OnLine Dynamic Server. This chapter covers the following topics:

- Conversion from static to dynamic allocation of resources that were previously controlled with configuration parameters
- How to control OnLine reaction to I/O errors using the new configuration parameter ONDBSPDOWN and the new **-O** option of **onmode**
- How to preserve space in your logical logs for administrative tasks using the new configuration parameter LBU_PRESERVE
- How to specify the size of the memory cache for the INFORMIX-OnLine/Optical subsystem using the configuration parameter OPCACHEMAX
- How to display information about the INFORMIX-OnLine/Optical memory cache and staging-area blobspace using **onstat -O**

Dynamic Allocation of Resources

In previous versions of OnLine, to change the value of certain configuration parameters, you had to bring OnLine to quiescent mode, change the value of the configuration parameter, and reinitialize shared memory. These configuration parameters are *static* configuration parameters. The following parameters are in this category:

- USERTHREADS
- TRANSACTIONS
- TBLSPACES
- CHUNKS
- DBSPACES

Beginning with the current version of OnLine, you can no longer configure these five parameters. Instead, whenever OnLine reaches the limit for a resource previously controlled by one of these parameters, it automatically or *dynamically* allocates an appropriate quantity of the required resource. OnLine performs the allocation and sizing of any underlying data structure in on-line mode, making it unnecessary for you to bring OnLine off-line for the changes to take effect.

If you wish, you can remove the five configuration parameters from your configuration file, but OnLine does not require you to take this action. If your configuration file still contains the old static configuration parameters, OnLine ignores both the parameters and any value you assign to them.

You do not need to take any action to initiate the dynamic allocation of resources. The current version of OnLine begins dynamic allocation automatically when you initialize the database server for the first time. You must, however, check the values of other parameters that depended on the five eliminated parameters, as explained in the next section.

Changes to the Default Values of Dependent Parameters

Informix changed the default values of the following parameters because of their dependence on the value of TRANSACTIONS, USERTHREADS, and CHUNKS:

- BUFFERS
- LRUS
- SHMVIRTSIZE
- NETTYPE
- LOCKS
- NUMAIOVPS

The new parameter defaults are discussed in the sections that follow. In each section, a description of the current default value is presented first, followed by a description the old default value.

BUFFERS

If you do not specify a value for BUFFERS in the configuration file, BUFFERS defaults to 1000 buffers. This value results in a 2-megabyte buffer pool when OnLine page size is 2 kilobytes, and a 4-megabyte buffer pool when OnLine page size is 4 kilobytes. The minimum number of buffers that you can specify is 100. This value results in a 200-kilobyte buffer pool when OnLine page size is 2 kilobytes, and a 400-kilobyte buffer pool when page size is 4 kilobytes.

The previous default value for BUFFERS was 4 * USERTHREADS. This default is no longer valid because Informix eliminated the USERTHREADS configuration parameter.

LRUS

If you do not specify LRUS in the configuration file, OnLine sets a default value based on the MULTIPROCESSOR configuration parameter. If you specify a value for MULTIPROCESSOR, OnLine uses the following formula to determine a default for LRUS:

```
LRUS = MAX(4, NUMCPUVPS)
```

If you do not specify MULTIPROCESSOR in the configuration file, OnLine uses a default value of 4.

The previous default value for LRUS was MIN(USERTHREADS/2, 8). This default is no longer valid because Informix eliminated the USERTHREADS configuration parameter.

SHMVIRTSIZE

If you do not specify a value for SHMVIRTSIZE in your configuration file, OnLine uses the value of SHMADD as a default. The default for SHMADD is 8 kilobytes (meaning OnLine adds 8-megabyte segments). If you do not specify SHMVIRTSIZE nor SHMADD, OnLine uses an initial virtual shared-memory segment of 8 megabytes.

The previous default value for SHMVIRTSIZE was 200 * USERTHREADS. This default is no longer valid because Informix eliminated the USERTHREADS configuration parameter.

NETTYPE

The third field of the NETTYPE configuration parameter, which specifies the maximum number of connections for that net type, defaults to 50. OnLine administrators who wish to limit or increase the default number of connections to the database server for a particular NETTYPE must specify the third field of the NETTYPE.

The previous default value for the third field of NETTYPE was the value of USERTHREADS. This default is no longer valid because Informix eliminated the USERTHREADS configuration parameter.

LOCKS

The range of values for LOCKS is 2000 to a maximum of 8,000,000.

In previous versions of OnLine, the range of values for LOCKS was the greater of 2000 and (100 * SQRT(TRANSACTIONS)) to a maximum of 8,000,000. This range of values is no longer valid because Informix eliminated the TRANSACTIONS configuration parameter.

NUMAIOVPS

OnLine calculates a default value of NUMAIOVPS as follows:

```
Default_NUMAIOVPS = 2 * (number_of_chunks)
```

For example, suppose that you allocate a total of five chunks, one for the root dbspace and four more for an additional dbspace. The next time that you bring up OnLine after you allocate the chunks, OnLine configures 10 AIO virtual processors.

In previous versions of OnLine, the default value of NUMAIOVPS depended on the value of the CHUNKS configuration parameter. This default is no longer valid because Informix eliminated the CHUNKS configuration parameter.

Changes in OnLine Architecture Associated with Dynamic Allocation of Resources

OnLine allocates the resources previously controlled by the five eliminated parameters in the following way:

- USERTHREADS and TRANSACTIONS

  OnLine allocates the underlying structures in blocks.

- CHUNKS and DBSPACES

  OnLine allocates space for the underlying structures for these items to allow for the current maximum of 2 kilobytes for CHUNKS and DBSPACES.

- TBLSPACES

  OnLine increases the number of tblspace structures as necessary. Informix eliminated the hash lists associated with TBLSPACES. OnLine organizes tblspace entries internally based on dbspace. OnLine maintains tblspace entries after it closes a tblspace to allow tblspace statistics to last for the life of the database server or the tblspace.

OnLine does not automatically free underlying structures that correspond to the new dynamically configured parameters. For user thread and transaction structures, OnLine does not even attempt to coalesce these tables or lists and free memory until you execute **onmode -F**.

Because OnLine allocates these structures dynamically, they now reside in the virtual instead of the resident portion of shared memory.

Error Messages Associated with Dynamic Allocation of Resources

Informix changed the messages that OnLine sends to the message log when OnLine cannot allocate a sufficient amount of user threads, transactions, or tblspaces. The following table presents each of the old messages and the corresponding new messages. Old messages are messages that OnLine sent previous to this release; new messages apply to versions beginning with this release.

OnLine delivers the new messages in the context of an assertion warning. Informix replaced some of the old messages with more than one new message to indicate more exactly the cause of and suggested action for each of the errors.

OnLine no longer issues some messages related to static tables because it no longer uses this data structure to manage user threads and transactions.

Parameter	Message Status	Message
USERTHREADS	Old	User thread table overflow - user id *n*
	New	Unable to allocate a user thread for user id *n* User thread limit of 32767 reached
	New	Unable to allocate a user thread for user id *n* Memory allocation failure Retry operation later or make more virtual memory available to OnLine
	Obsolete	Unable to allocate any recovery worker threads *recvry_typ* increase 'USERTHREADS'
	Obsolete	Unable to allocate the *n* requested recovery worker threads for *recvry_type*
	Obsolete	Allocating the maximum number of worker threads available *m*
	Old	WARNING! Physical Log size *current_log_size* is too small. Physical Log overflows may occur during peak activity. Physical Log size should be increased to *recommended_log_size* to guard against physical log overflows.
	New	WARNING! Physical Log size *current_log_size* is too small. Physical Log overflows may occur during peak activity. Recommended minimum Physical Log size is *recommended_log_size* times maximum concurrent user threads.
	Old	WARNING! Logical log layout may cause OnLine to get into a locked state. To guard against this problem, increase the size of the smallest logical log from *current_log_size* to *recommended_log_size*.
	New	WARNING! Logical log layout may cause OnLine to get into a locked state. Recommended smallest logical log size is *recommended_log_size* times maximum concurrent user threads.

(1 of 2)

Parameter	Message Status	Message
	New	WARNING! Buffer pool size may cause OnLine to get into a locked state. Recommended minimum buffer pool size is *recommended_buffer_size* times maximum concurrent user threads.
TRANSACTIONS	Old	Transaction table overflow-user id *n*, session id *m*
	New	Unable to allocate a transaction for user id *n*, session id *m* Transaction limit of 32767 reached
	New	Unable to allocate a transaction for user id *n*, session id *m* Memory allocation failure Make more virtual memory available to OnLine
	Obsolete	Transaction table overflow - user id *n*, session id *m* increase TRANSACTIONS
	Obsolete	Transaction table overflow due to parallel recovery
	Obsolete	Delaying until transaction slot available
	Obsolete	To improve recovery performance, increase TRANSACTIONS
TBLESPACES	Old	TBLSpace table overflow - user id *n*, sid *m*
	New	Unable to open TBLSpace 0x*dddddddd* for user id *n*, session id *m* Memory allocation failure Make more virtual memory available to OnLine

(2 of 2)

Memory is the only factor that limits the maximum number of concurrently open tblspaces. Previously, Informix arbitrarily set the limit at 32,000.

Controlling How OnLine Reacts to I/O Errors

Previous to this version of OnLine, a kicked-out cable or failed controller meant that OnLine immediately marked the dbspace affected by the cable or controller as down. With the current version of OnLine, you can prevent OnLine from marking a dbspace as down while you investigate the problem.

If you find that the problem is trivial, such as a loose cable, you can bring OnLine off-line and then on-line again without restoring the affected dbspace from archive. If you find that the problem was legitimate, such as a damaged disk, you can use **onmode -O** to mark the affected dbspace as down and continue processing.

What Are Disabling I/O Errors?

Before OnLine considers an I/O error to be disabling, the error must meet two criteria. First, the error must occur when OnLine attempts to perform an operation on a chunk that has at least one of the following characteristics:

- The chunk has no mirror.
- The primary or mirror companion of the chunk under question is off-line.

Second, the error must occur when OnLine attempts, but fails, to perform one of the following three operations:

- Seek, read, or write on a chunk
- Open a chunk
- Verify that chunk information on the first used page is valid

  OnLine does this as a sanity check immediately after opening a chunk.

What Causes Disabling I/O Errors?

Informix divides disabling I/O errors into two general categories, destructive and nondestructive. A disabling I/O error is destructive when the disk that contains a database becomes damaged in some way. This type of event threatens the integrity of data, and OnLine marks the chunk and dbspace as down. OnLine prohibits access to the damaged disk until you repair or replace the disk and perform a physical and logical restore.

A disabling I/O error is nondestructive when the error does not threaten the integrity of your data. Examples of nondestructive errors occur when someone accidentally kicks out a cable, you somehow erase the symbolic link that you set up to point to a chunk, or a disk controller becomes damaged.

Prior to this release, OnLine prohibited access to the chunk that generated the disabling error, even when the cause was nondestructive. As in the case where disk damage occurs, OnLine required that you correct the problem and perform a physical and logical restore before you could access data in the disabled dbspace.

New Alternatives for Dealing with Disabling I/O Errors

With the current release of OnLine, Informix supplies a new configuration parameter ONDBSPDOWN. You can set ONDBSPDOWN to one of the three modes explained in the following sections.

CONTINUE or 0

OnLine marks a noncritical dbspace as down and continues whenever a disabling I/O error associated with the dbspace occurs. This mode causes OnLine to treat disabling I/O errors as it did in earlier versions. OnLine handles critical dbspaces as in ABORT mode.

Use CONTINUE mode if the occurrence of nondestructive disabling I/O errors is infrequent. Note, however, that this mode requires you to restore from archive if a nondestructive I/O error occurs.

ABORT or 1

OnLine comes off-line without allowing a checkpoint to occur whenever a disabling I/O error occurs on any dbspace. Critical dbspaces operate in this mode only.

Consider using ABORT mode if you have established procedures for dealing with critical media failures. You can use the same procedures for disabling I/O errors that affect noncritical media.

WAIT or 2

When a disabling I/O error from a noncritical dbspace occurs, OnLine waits for the next checkpoint request and then blocks all updating threads. If you suspect that disabling I/O is nondestructive, WAIT mode allows you time to investigate the problem that caused the error.

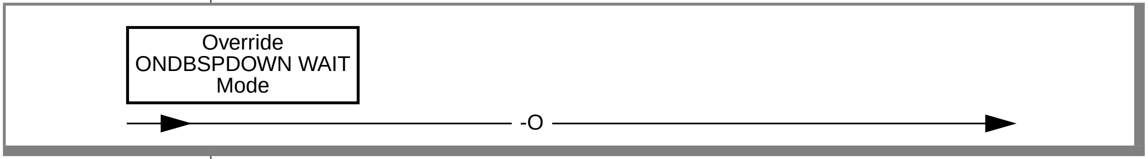
If you find that the cause of the disabling I/O error is nondestructive, take the following steps:

1. Correct the problem (plug in the cable, replace the failed controller, or remedy whatever your particular problem happens to be) causing the disabling I/O error.
2. Bring OnLine down using **onmode -k**.
3. Bring OnLine back up using **oninit**.

If you find that the cause of the disabling I/O error is destructive (data integrity is affected), or that you cannot correct the problem in a suitable time frame, you can override the WAIT mode with **onmode -O**. When you override the WAIT mode, OnLine marks the dbspace as down, completes the checkpoint, and releases the blocked threads. Processing can continue, but you must restore the disabled dbspace from archive before OnLine can read or write to it. See “Using onmode -O to Override ONDBSPDOWN WAIT Mode” on page 1-13 for **onmode -O** syntax and associated error messages.

Use WAIT mode if you experience disabling I/O errors that do not affect the integrity of your data and you have the resources to correct the error in a relatively short time span.

Using onmode -O to Override ONDBSPDOWN WAIT Mode



The **onmode -O** option overrides the WAIT mode of the ONDBSPDOWN configuration parameter. Use this option only in the following circumstances:

- ONDBSPDOWN is set to WAIT.
- A disabling I/O error occurs causing OnLine to block all updating threads.
- You cannot, or do not, wish to correct the problem that caused the disabling I/O error.
- You want OnLine to mark the disabled dbspace as down and continue processing.

When you execute this option, OnLine marks the dbspace responsible for the disabling I/O error as down, completes a checkpoint, and releases blocked threads. Then, **onmode** prompts you with the following message:

```
This will render any dbspaces which have incurred disabling I/O errors unusable
and require them to be restored from an archive.
Do you wish to continue?(y/n)
```

If **onmode** does not encounter any disabling I/O errors on noncritical dbspaces when you run the **-O** option, it notifies you with the following message:

```
There have been no disabling I/O errors on any non-critical dbspaces.
```

Monitoring OnLine for Disabling I/O Errors

OnLine notifies you about disabling I/O errors in two ways: the message log and event alarms.

Monitoring Disabling I/O Errors Using the Message Log

OnLine sends the following message to the message log when a disabling I/O error occurs:

```
Assert Failed: Chunk {chunk-number} is being taken OFFLINE.
Who: Description of user/session/thread running at the time
Result: State of the affected OnLine entity
Action: What action the OnLine administrator should take
See Also: DUMPDIR/af.uniqid containing more diagnostics
```

The result and corresponding suggested action depend on the current setting of ONDBSPDOWN, as described in the following table:

ONDBSPDOWN Setting	Result	Action
CONTINUE	dbspace/blobspace {space-name} is disabled	Restore dbspace/blobspace {space-name}
ABORT	OnLine must abort.	Reinitialize shared memory.
WAIT	OnLine blocks at next checkpoint	Shutdown using **onmode -k** or override **onmode -O**

For more information on how to interpret messages that OnLine sends to the message log, see Chapter 41, "OnLine Message Log Messages," of the *INFORMIX-OnLine Dynamic Server Administrator's Guide*.

Monitoring Disabling I/O Errors Using Event Alarms

When a dbspace incurs a disabling I/O error, OnLine passes the following values as parameters to your event-alarm executable:

Parameter	Value
Severity:	4 (Emergency)
Class:	5
Class message:	dbspace is disabled: ‘dbspace-name’
Specific message:	Chunk {chunk-number} is being taken OFFLINE.

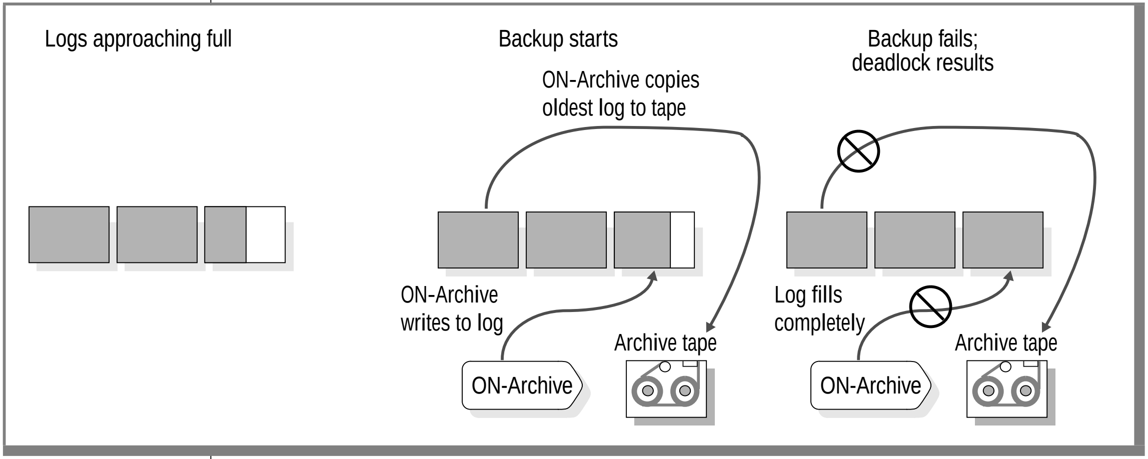
If you wish OnLine to notify you about disabling I/O errors using event alarms, you must write a script that OnLine executes when it detects a disabling I/O error. Chapter 33, “Monitoring OnLine,” of the *INFORMIX-OnLine Dynamic Server Administrator’s Guide* explains how to set up this executable and make OnLine aware of the location of the executable that you write.

Preserving Log Space for Administrative Tasks

During peak activity on high-volume, on-line transaction processing (OLTP) systems, OnLine administrators sometimes encounter a deadlock when OLTP activity fills the logical log faster than ON-Archive can back up the logs to tape and free them.

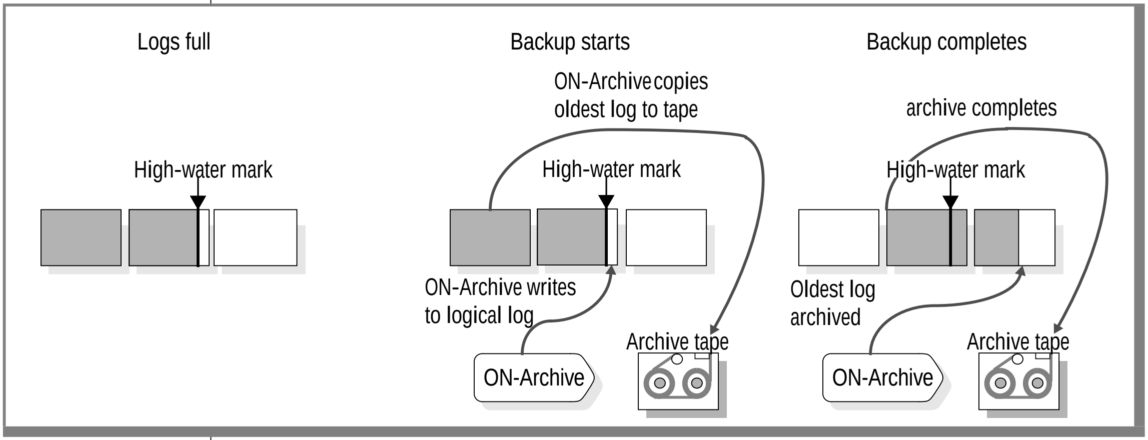
Prior to marking a backed-up log as free, OnLine updates the ON-Archive catalog tables to record the occurrence of the logical-log backup. This update itself generates logical-log activity, leading to a possible deadlock as the logs continue to fill. When a deadlock of this type occurs, the OnLine administrator must use the emergency log backup procedure even though a continuous logical-log backup is already in progress. Figure 1-1 illustrates how a deadlock of this type occurs.

Figure 1-1
ON-Archive with high-water mark off



The ON-Archive high-water mark feature provides a solution for logical-log deadlocks of this type. When you enable this feature, OnLine blocks OLTP activity when the next-to-last log fills, rather than the last log, as it usually does. In doing so, OnLine preserves the last logical-log file to record logging generated by administrative activities such as a backup of the logical log. Figure 1-2 on page 1-17 illustrates how the high-water mark feature prevents logical-log deadlocks.

Figure 1-2
ON-Archive with high-water mark on



For more information on the emergency backup procedure, see Chapter 6, "Backing Up the Logical Log," of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.

Enabling the Logs-Full High-Water Mark

To enable the logs-full high-water mark, set the LBU_PRESERVE configuration parameter to 1. When you set LBU_PRESERVE to 1, OnLine blocks DB-Access, ESQL/C, and all other clients from generating log records in the last logical-log file when the logs-full condition is reached. The default value of LBU_PRESERVE is 0, or off.

Whenever you change the value of LBU_PRESERVE, you must reinitialize shared memory for the change to take effect.

Monitoring the Logical Log for Fullness Using the Message Log

OnLine continues to send logs-full messages to the message log similar to the messages it sent in previous versions of OnLine. Whenever a logs full condition occurs, however, instead of sending the messages when the last log fills, OnLine sends the messages when the second-to-last log is full.

Monitoring the Logical Log for Fullness Using onstat

Whenever you set LBU_PRESERVE to 1 and OnLine is blocking to preserve log space for administrative tasks, the **onstat** utility displays the following message just after its banner line:

```
Blocked: LBU
```

For example, suppose that OnLine is running under the following conditions:

- You set LBU_PRESERVE to 1.
- All logs, but the last, are full.

In these circumstances, the first two lines of any of the **onstat** options appear, as shown in the follow example:

```
RSAM Version 7.10.U -- On-Line -- Up 00:12:53 -- 5152 Kbytes
Blocked: LBU
```

Cases Where You Must Still Use Emergency Log Backup

Although the logs-full high-water mark eliminates the need for emergency backup in the deadlock scenario described in the preceding sections, four known scenarios still require OnLine administrators to use emergency log backup. Each case is examined in detail in the following sections.

Building the System Monitoring Interface

A privileged client is responsible for building the system monitoring interface (SMI). This client can potentially invade the last log file. If you do not configure sufficient log space or a sufficient number of log files, the privileged client might not succeed in building SMI without a log backup. This situation can cause the logical log to fill.

Recovery

When you start OnLine after an uncontrolled shutdown, it needs log space to roll back any transactions that were uncommitted when the shutdown occurred. The threads that perform the recovery have privileges that allow them to use the last log file. Because of this privilege, it is possible that the logical log can become full, but only in the unlikely case that the number and size of transactions open when the shutdown occurred exceed the size of the logical log.

Small Logs, Many Users

If you configure your logical log files so that

```
Logical Log Size < 2 * page_size * number of users
```

and all users enter transactions of maximum complexity, it is possible that applications can invade the last log with OLTP activity. Only when you set the size of the log much smaller than two pages per user can a logs-full condition occur.

Administrative Activity When Logs Need Backing Up

Certain administrative clients have the privilege to invade the last logical-log file. The following list gives examples of such administrative clients:

- onspaces
- onparams
- oncheck
- ontape
- onmonitor
- ON-Archive
- oncatlgr
- onautovop
- ondatartr

Because these clients can invade the last logical log, certain circumstances might require you to perform an emergency log backup. For example, when the logical log approaches full, and you proceed to do large quantities of administrative work, you might need to perform an emergency log backup.

Auditing Configuration Parameters

Informix moved the following configuration parameters from **onconfig.std** to **adtcfg.std**:

- ADTERR
- ADTMODE
- ADTPATH
- ADTSIZE

For more information on **adtcfg.std** and the configuration parameters it contains, see the *INFORMIX-OnLine Dynamic Server Trusted Facility Manual*.

OPCACHEMAX Configuration Parameter

default value	128
units	kilobytes
range of values	Positive integers
takes effect	When OnLine needs more memory
ON-M*onitor*	Parameters, Initialize, StageBlob
refer to	See the *INFORMIX-OnLine/Optical User Manual*

The OPCACHEMAX configuration parameter specifies the size of the memory cache for the INFORMIX-OnLine / Optical subsystem. OnLine stores pieces of blobs in the memory cache before delivering them to the subsystem. Use this parameter only if you use optical storage with INFORMIX-OnLine / Optical.

OnLine Algorithm for Determining DS_TOTAL_MEMORY

Beginning with this release, OnLine derives a value for DS_TOTAL_MEMORY when you do not set DS_TOTAL_MEMORY, or you set it to an inappropriate value. Whenever OnLine changes the value that you assigned to DS_TOTAL_MEMORY, it notifies you by sending the following message to your console:

```
DS_TOTAL_MEMORY recalculated and changed from old_value Kb
        to new_value Kb
```

The algorithm that OnLine uses to derive the new value for DS_TOTAL_MEMORY is documented in the following sections. When you receive the preceding message, you can use the algorithm to investigate what values OnLine considers inappropriate and take corrective action based on your investigation.

Derive a Minimum for Decision-Support Memory

In the first part of the algorithm, OnLine establishes a minimum for decision-support memory. When you assign a value to the configuration parameter DS_MAX_QUERIES, OnLine sets the minimum amount of decision-support memory according to the following formula:

```
min_ds_total_memory = DS_MAX_QUERY * 128Kb
```

When you do not assign a value to DS_MAX_QUERIES, OnLine instead uses the following formula based on the value of NUMCPUVPS:

```
min_ds_total_memory = NUMCPUVPS * 2 * 128Kb
```

Derive a Working Value for Decision-Support Memory

In the second part of the algorithm, OnLine establishes a working value for the amount of decision-support memory. OnLine verifies this amount in the third and final part of the algorithm.

When DS_TOTAL_MEMORY Is Set

OnLine first checks if SHMTOTAL is set. When SHMTOTAL is set, OnLine uses the following formula to calculate DS_TOTAL_MEMORY:

```
IF DS_TOTAL_MEMORY <= SHMTOTAL - nondecision_support_memory THEN
   decision-support memory = DS_TOTAL_MEMORY
ELSE
    decision-support memory = SHMTOTAL - nondecision_support_memory
```

This algorithm effectively prevents you from setting DS_TOTAL_MEMORY to values that OnLine cannot possibly allocate to decision-support memory.

When SHMTOTAL is not set, OnLine sets decision-support memory equal to the value that you specified in DS_TOTAL_MEMORY.

When DS_TOTAL_MEMORY Is Not Set

When you do not set DS_TOTAL_MEMORY, OnLine proceeds as follows. First, OnLine checks if you set SHMTOTAL. When SHMTOTAL is set, OnLine uses the following formula to calculate the amount of decision-support memory:

```
decision-support memory = SHMTOTAL -
nondecision_support_memory
```

When OnLine finds that you did not set SHMTOTAL, it sets decision-support memory as shown in the following example:

```
decision-support memory = min_ds_total_memory
```

The variable **min_ds_total_memory** is described in "Derive a Minimum for Decision-Support Memory" on page 1-21.

Check Derived Value for Decision-Support Memory

The final part of the algorithm verifies that the amount of shared memory is greater than **min_ds_total_memory** and less than the maximum possible memory space for your computer. When OnLine finds that the derived value for decision-support memory is less than **min_ds_total_memory**, it sets decision-support memory equal to **min_ds_total_memory**.

When OnLine finds that the derived value for decision-support memory is greater than the maximum possible memory space for your computer, it sets decision-support memory equal to the maximum possible memory space.

Inform User When Derived Value Is Different from User Value

When, at any point during the processing of this algorithm, OnLine changes the value that you set for DS_TOTAL_MEMORY, it sends a message to your console in the following format:

```
DS_TOTAL_MEMORY recalculated and changed from old_value Kb
            to new_value Kb
```

The metavariable *old_value* represents the value that you assigned to DS_TOTAL_MEMORY in your configuration file. The metavariable *new_value* represents the value that OnLine derived.

New Default for OPTCOMPIND

Informix changed the default value of the configuration parameter OPTCOMPIND from `0` to `2`. A default of `2` forces the optimizer to make its decisions about optimization based purely on costs without taking into consideration translation isolation mode. With the new default, OnLine does not give preference to index scans (nested-loop joins) over table scans (other join methods). For more information about OPTCOMPIND, see Chapter 38, "OnLine Configuration Parameters," of the *INFORMIX-OnLine Dynamic Server Administrator's Guide*.

Enhancements to the onstat Utility

This section addresses three enhancements to the **onstat** OnLine utilities. The first enhancement is a new option that helps you monitor optical memory cache and staging-area blobspaces. The second enhancement is an additional line in the **onstat** banner line that displays whenever OnLine is blocking.

The third enhancement improves your ability to monitor the allocation of user threads and transactions. Because OnLine automatically allocates user threads and transactions as needed, you no longer need to set the configuration parameters USERTHREADS and TRANSACTIONS. You might want, however, to monitor the dynamic allocation of user-thread and transaction resources. As explained in the following sections, changes to the display of the **-u** and **-x** options allow you to monitor these resources.

onstat -O option

Use the **-O** option of the **onstat** utility to display information about the INFORMIX-OnLine/Optical memory cache and staging-area blobspace. You can interpret output from this option as follows. The totals shown in the display accumulate from session to session. OnLine resets the totals to `0` only when **you execute onstat -z**.

The first section of the display describes the following system-cache totals information:

size	is the size specified in the OPCACHEMAX configuration parameter.
alloc	is the number of 1-kilobyte pieces that OnLine allocated to the cache.
avail	describes how much of **alloc** (in kilobytes) is not used.
number	is the number of blobs that OnLine successfully put into the cache without overflowing.
kbytes	is the number of kilobytes of the blobs that OnLine put into the cache without overflowing.
number	is the number of blobs that OnLine wrote to the staging-area blobspace.
kbytes	is the number of kilobytes of the blobs that OnLine wrote to the staging-area blobspace.

Although the **size** output indicates the amount of memory that is specified in the configuration parameter OPCACHEMAX, OnLine does not allocate memory to OPCACHEMAX until necessary. Therefore, the **alloc** output reflects only the number of 1-kilobyte pieces of the largest blob that has been processed. When the values in the **alloc** and **avail** output are equal to each other, the cache is empty.

The second section of the display describes the following user-cache totals information:

SID	is the session id for the user.
use	is the userid of the client.
size	is the size specified in the **INFORMIXOPCACHE** environment variable, if set. If you do not set the **INFORMIXOPCACHE** environment variable, OnLine uses the size that you specify in the configuration parameter OPCACHEMAX.
number	is the number of blobs that OnLine put into cache without overflowing.

kbytes	is the number of kilobytes of the blobs that OnLine put into the cache without overflowing.
number	is the number of blobs that OnLine wrote to the staging-area blobspace.
kbytes	is the number of kilobytes of the blobs that OnLine wrote to the staging-area blobspace.

Improved Diagnostic Information

Whenever OnLine is blocked, **onstat** displays the following additional line after the banner line:

```
Blocked: reason
```

The metavariable *reason* can take one of the following values:

Reason	Description
CKPT	checkpoint
LONGTX	long transaction
ARCHIVE	ongoing archive
MEDIA_FAILURE	media-failure
HANG_SYSTEM	OnLine failure
DBS_DROP	dropping a dbspace
DDR	discrete data replication
LBU	logs full high-water mark

See "Monitoring the Logical Log for Fullness Using onstat" on page 1-18 for an example of what **onstat** displays when OnLine is blocked to preserve logical-log space for administrative tasks.

Changes to onstat -u Output

Informix enhanced the last line of **onstat -u** output to display the maximum number of concurrent user threads that were allocated since you initialized OnLine. For example, the last line of **onstat -u** output shown in Figure 1-3 is:

```
4active, 128 total, 17 maximum concurrent
```

The last part of the line, `17 maximum concurrent`, indicates that the maximum number of user threads that were running concurrently since you initialized OnLine was `17`.

```
RSAM Version 7.10.UD1 -- On-Line -- Up 00:50:22 -- 8896 Kbytes

Userthreads
address  flags    sessid   user     tty      wait     tout locks nreads   nwrites
80eb8c   ---P--D 0        informix -        0        0    0     33       19
80ef18   ---P--F 0        informix -        0        0    0     0        0
80f2a4   ---P--B 3        informix -        0        0    0     0        0
80f630   ---P--D 0        informix -        0        0    0     0        0
4active, 128 total, 17 maximum concurrent
```

Figure 1-3
onstat -u output

For more information on **onstat -u**, see Chapter 40, “OnLine Utilities,” of the *INFORMIX-OnLine Dynamic Server Administrator’s Guide*.

Change to onstat -x Output

Informix enhanced the last line of **onstat -x** output to display the maximum number of concurrent transactions since you initialized OnLine. For example, the last line of **onstat -u** output shown in Figure 1-4 is:

```
11active, 128 total, 6 maximum concurrent
```

The last part of the line, `6 maximum concurrent`, indicates that the maximum number of transactions that were running concurrently since you initialized OnLine was 6.

```
RSAM Version 7.10.UD1 -- On-Line -- Up 00:50:22 -- 8896 Kbytes

Transactions
address  flags userthread locks log begin isolation retrys coordinator
40a7e4   A---- 406464     0     0         COMMIT    0
40a938   A---- 4067c4     0     0         COMMIT    0
40aa8c   A---- 406b24     0     0         COMMIT    0
40abe0   A---- 40a124     0     0         COMMIT    0
11 active, 128 total, 6 maximum concurrent
```

Figure 1-4
onstat -x output

For more information on **onstat -x**, see Chapter 40, “OnLine Utilities,” in the *INFORMIX-OnLine Dynamic Server Administrator’s Guide*.

Parallel Inserts

Beginning with this release, OnLine performs the following types of inserts in parallel:

- Explicit inserts using SELECT...INTO TEMP
- Implicit inserts using INSERT INTO...SELECT

The following sections explain the details and restrictions that concern parallel inserts.

Explicit Inserts Using SELECT…INTO TEMP

OnLine can now insert rows in parallel into explicit temporary tables that you specify in SQL statements of the form SELECT....INTO TEMP. For example, OnLine can perform the inserts in parallel into the temporary table, **temp_table**, as shown in the following example:

```
SELECT * FROM table1 INTO TEMP temp_table
```

OnLine performs this type of parallel insert provided that you set the following configuration parameters as indicated:

- PDQPRIORITY > 0
- DBSPACETEMP is set to a list of two or more dbspaces.

The first item, PDQPRIORITY > 0, is a requirement that you must meet for any query that you wish OnLine to perform in parallel.

The second item, that DBSPACETEMP is set to a list of two or more dbspaces, is required because of the way that OnLine performs the insert. To perform the insert in parallel, OnLine first creates a fragmented temporary table. So that OnLine knows where to store the fragments of the temporary table, you must specify a list of two or more dbspaces in the DBSPACETEMP configuration parameter. In addition, you must set DBSPACETEMP to indicate storage space for the fragments *before* you execute the SELECT...INTO statement.

OnLine performs the parallel insert by writing in parallel to each of the fragments in a round-robin fashion. Performance improves as you increase the number of fragments.

Implicit Inserts Using INSERT INTO...SELECT

OnLine can also insert rows in parallel into implicit temporary tables that it creates when it processes SQL statements of the form INSERT INTO...SELECT. For example, OnLine processes the following INSERT statement in parallel:

```
INSERT INTO target_table SELECT * FROM source_table
```

OnLine processes this type of INSERT statement in parallel only when the source and target tables meet the following criteria:

- The target table has no enabled referential constraints or triggers.
- The target table is not a remote table.
- In a database with logging, the target table does not contain filtering constraints.
- The source table does not contain columns of data type TEXT or BYTE.

OnLine does not process multirow inserts that reference a stored procedure. For example, OnLine never processes the following statement in parallel:

```
INSERT INTO table1 EXECUTE PROCEDURE ins_proc
```

For more information on parallel processing, see Chapter 18, "What Is PDQ?" in the *INFORMIX-OnLine Dynamic Server Administrator's Guide*.

Enhancements to Existing SMI Tables

When you install and initialize OnLine for the first time, it checks for an existing **sysmaster** database. If OnLine finds an existing **sysmaster** database, it takes the following course of action:

- Converts the existing **sysmaster** database to the current SMI format
- Preserves all of the ON-Archive tables that it encounters
- Preserves the **sysaudit** table that contains all of the auditing masks

In addition, OnLine sends a message to the message log that indicates that it is either building or upgrading the **sysmaster** database.

The remainder of this section documents the columns that were added to four of the existing SMI tables.

syssessions

The **syssessions** table contains the following new column:

Column	Type	Description
pooladdr	int	Session pool address

syssesprof

The **syssesprof** table contains the following new columns:

Column	Type	Description
pagreads	int	Number of pages read from disk
pagwrites	int	Number of pages written to disk
total_sorts	int	Total number of sorts performed
dsksorts	int	Number of sorts that required disk I/O
max_sortdiskspace	int	Maximum disk space required by a sort

sysptprof

The **sysptprof** table contains the following new columns:

Column	Type	Description
pagreads	int	Number of pages read from disk
pagwrites	int	Number of pages written to disk

sysprofile

The **sysprofile** table contains the following new columns:

Column	Type	Description
totsorts	int	Total number of sorts performed
dsksorts	int	Number of sorts that required disk I/O
srtspmax	int	Maximum disk space required by a sort

Chapter

Connectivity Enhancements

2

This chapter discusses connectivity-related changes introduced in this release. The overall architecture of client/server connectivity is the same as that of earlier versions of INFORMIX-OnLine Dynamic Server and INFORMIX-SE.

For more information about connectivity, refer to the *INFORMIX-OnLine Dynamic Server Administrator's Guide* and the *INFORMIX-SE Administrator's Guide*.

Stream Pipes

If the client and server are on different computers, you must use network connections. If the client and server are on the same computer, you can connect to an OnLine database server using the following types of connections:

- Network (local loopback)
- Shared memory
- Stream pipe

Local-loopback and shared-memory connections are described in the *INFORMIX-OnLine Dynamic Server Administrator's Guide*.

Description of Stream Pipes

A *named stream pipe* is a UNIX interprocess communication (IPC) facility that allows processes on the same computer to communicate with each other. You can use stream-pipe connections any time the client and the database server are on the same computer. You can also use stream-pipe connections for distributed database operations when both database servers are on the same computer.

This documentation refers to a named stream pipe as simply a *stream pipe*. Named stream pipes are also sometimes referred to as *mounted streams*. Stream pipes are not available on all computers. Check the machine notes file (**ONLINE_7.1**) and your system documentation for information about stream pipes on your computer.

The sqlhosts Entries for Stream Pipes

For a stream-pipe connection, the entry in the **nettype** field of the **sqlhosts** file is `onipcstr`.

The value in the **servicename** field can be any short combination of letters that is unique among all the OnLine database servers on the same computer. For simplicity, Informix suggests that you use the *dbservername* in the **servicename** field for stream-pipe connections.

The following table shows a possible **sqlhosts** entry for a stream-pipe connection:

servername	nettype	hostname	servicename	options
server1	onipcstr	dallas	server1	

OnLine uses the *servicename* to create a file that stores information about the stream-pipe connection. OnLine stores this file as **/INFORMIXTMP/***servicename***.str**. For example, if the *servicename* is **server1**, the file that OnLine creates is **/INFORMIXTMP/server1.str**.
On some computers, OnLine also creates a second file called **/INFORMIXTMP/***servicename***.exp**. Do not edit or delete these files.

Advantages and Disadvantages of Stream Pipes

Stream-pipe connections have the following advantage:

- Unlike shared-memory connections, stream pipes do not pose the security risk of being overwritten or read by other programs that explicitly access the same portion of shared memory.

Stream-pipe connections have the following disadvantages:

- Stream-pipe connections might be slower than shared-memory connections on some computers.
- Stream pipes are not available on all computers.

Enhancements to the sqlhosts File

The **$INFORMIXDIR/etc/sqlhosts** file contains information that the client (or another database server) uses to find and connect to a database server. Each entry (each line) in the **sqlhosts** file describes a potential connection to a database server. The **sqlhosts** file includes the following fields:

- **servername**
- **nettype**
- **hostname**
- **servicename**
- **options**

In addition to the enhancements to the **sqlhosts** file itself, a new environment variable, **INFORMIXSQLHOSTS**, lets you specify an alternative location for the **sqlhosts** file. This environment variable is described in "INFORMIXSQLHOSTS" on page 3-152.

The following sections describe the **options** field and enhancements to the other fields. The *INFORMIX-OnLine Dynamic Server Administrator's Guide* and the *INFORMIX-SE Administrator's Guide* have full discussions of the **sqlhosts** file.

The servername Field

The **servername** field contains the name of a database server. The **servername** field has not changed in this version of the Informix database servers.

The nettype Field

The support of stream-pipe connections requires a new value for **nettype**, **onipcstr**. The valid **nettype** entries for OnLine are as follows:

nettype entry	Type of connection
onipcshm	shared-memory connection
onipcstr	stream-pipe connection
onsoctcp	TCP/IP connection using sockets
ontlitcp	TCP/IP connection using TLI
ontlispx	IPX/SPX connection using TLI

Tip: *You can use* ***ol*** *in the* ***nettype*** *entry instead of* ***on*** *(for example,* ***olipcshm****), but the preferred style uses* ***on****.*

The hostname Field

The **hostname** field specifies the name of the computer where the database server resides. The **hostname** field has the following enhancements:

- You can use a longer entry than previously.
- You can use internet IP addresses with TCP/IP connections.
- You can use wildcards with TCP/IP connections.

Length of the hostname Entry

In earlier releases of Informix database servers, the maximum length of an entry in the **hostname** field was 128 characters. This enhancement increases the maximum length of the **hostname** entry to 256 characters.

Using an IP Address with TCP/IP Connections

For TCP/IP connections (both TLI and sockets), you can now use the actual internet IP address as the host name instead of the host alias found in the **/etc/hosts** file. The IP address is always composed of four sets of one to three integers, separated by periods. Figure 2-1 shows a sample **/etc/hosts** file with IP addresses and host names. The host aliases are optional and can be omitted.

Figure 2-1
A sample /etc/hosts file

Internet IP address	Host name	Host alias(es)
157.11.192.127	smoke	
49.192.4.63	odyssey	
37.1.183.92	knight	sales

Using the IP address from Figure 2-1, the following two **sqlhosts** entries are equivalent:

servername	nettype	hostname	servicename	options
sales	ontlitcp	37.1.183.92	sales_ol	
sales	ontlitcp	knight	sales_ol	

Using an IP address speeds up connection time. However, because computers are usually known by their host name, using IP addresses in the **sqlhosts** file makes it less convenient to identify the computer with which an entry is associated.

You can find the IP address from the net address field of the **/etc/hosts** file or by using the UNIX **arp** or **ypmatch** command.

Wildcard Addressing with TCP/IP Connections

You can use wildcard addressing in the **hostname** field of the **sqlhosts** file when *both* of the following conditions are met:

- You are using TCP/IP connections.
- The computer where the database server resides has multiple network-interface cards (for example, three Ethernet cards).

If the preceding conditions are met, you can use an asterisk (*) as a *wildcard* in the **sqlhosts** file used by the database server. When you enter an asterisk in the **hostname** field, the database server can accept connections at any valid IP address on its host computer.

Each IP address is associated with a unique hostname. When a computer has multiple network-interface cards, as in Figure 2-2 on page 2-9, the **/etc/hosts** file must have an entry for each interface card. For example, the **/etc/hosts** file for the **texas** computer might include these entries:

	Internet IP address	Host name	Host alias(es)
Card 1	123.34.6.81	texas1	
Card 2	123.34.6.82	texas2	

You can use the wildcard (*) alone or as a prefix for a host name or IP address, as shown in Figure 2-3 on page 2-10. The wildcard in the **hostname** field is meaningful only to the database server. If a client application uses an **sqlhosts** file entry that contains a wildcard, the client application simply ignores the wildcard and searches for a host name after the wildcard.

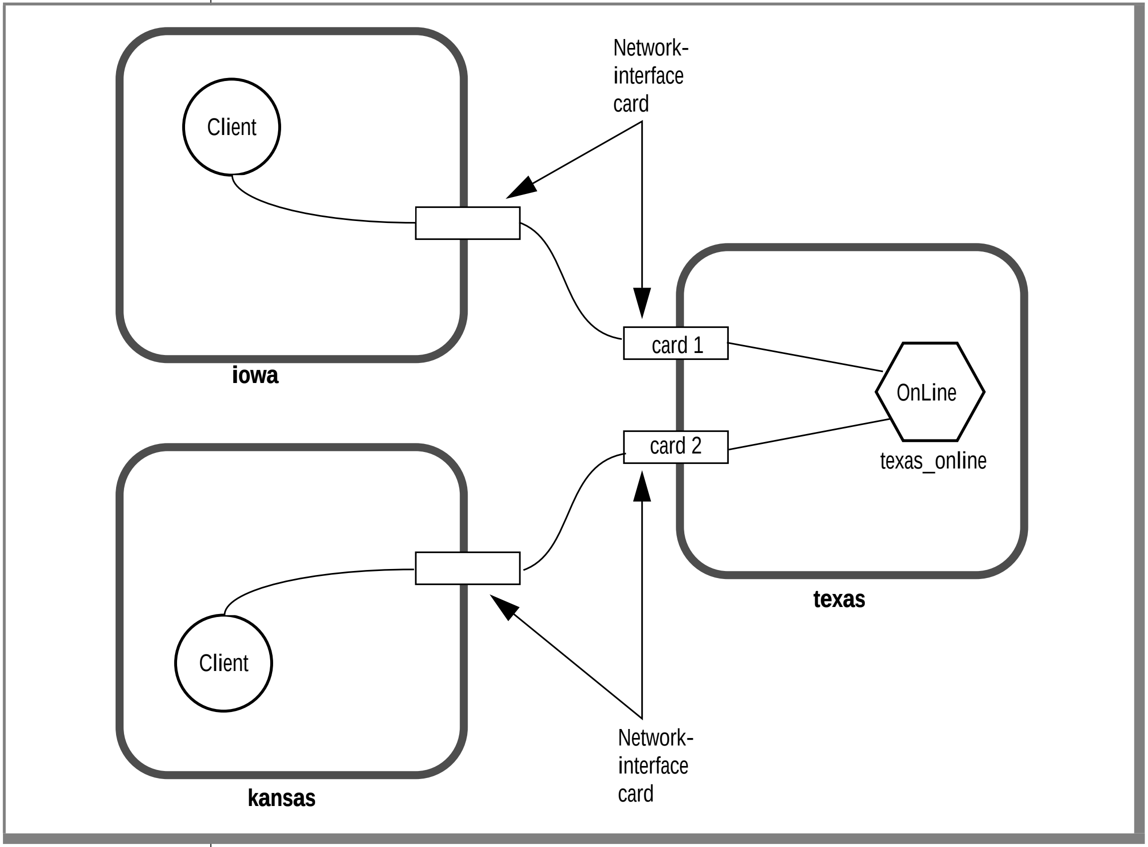
If the client application and database server share an **sqlhosts** file, you can specify both the asterisk and a host name or IP address in the **hostname** field (for example, *texas1 or *123.34.6.81). The client application ignores the asterisk and uses the host name (or IP address) to make the connection, and the database server uses the wildcard (*) to accept a connection from any IP address.

The wildcard format allows the database server to use any available connection. However, accepting a connection that uses the wildcard requires more CPU time than accepting a connection that uses an explicit host name or IP address. Do not use the wildcard format merely to save typing.

Using Wildcard Addressing

Figure 2-2 shows a database server on a computer (**texas**) that has two network-interface cards. The two client sites use different network cards to communicate with the database server.

Figure 2-2
Using multiple network-interface cards



The sqlhosts File for the Database Server

The **sqlhosts** file for the **texas_online** database server can include any *one* of the entries in Figure 2-3 on page 2-10.

Important: *You can include only one of these entries in your **sqlhosts** file.*

Figure 2-3
Possible entries in the sqlhosts file for the texas_online database server

servername	nettype	hostname	servicename	options
texas_online	ontlitcp	*texas1	pd1_on	
texas_online	ontlitcp	*123.34.6.81	pd1_on	
texas_online	ontlitcp	*texas2	pd1_on	
texas_online	ontlitcp	*123.34.6.82	pd1_on	
texas_online	ontlitcp	*	pd1_on	

If any of the preceding lines are in its **sqlhosts** file, the **texas_online** database server can accept client connections from either of the network cards. The database server finds the wildcard in the **hostname** field and ignores the explicit hostname.

Tip: For clarity and ease of maintenance, Informix recommends that you include a host name when you use the wildcard in the **hostname** field (that is, use `*host` instead of simply `*`).

The sqlhosts Files for the Client Applications

The **sqlhosts** file used by a client application must contain an explicit host name or IP address. The client application on **iowa** can use any *one* of the entries shown in Figure 2-4 in its **sqlhosts** file.

Important: *You can include only one of these entries in your **sqlhosts** file.*

Figure 2-4
Possible entries in the sqlhosts file for the client application on iowa

servername	nettype	hostname	servicename	options
texas_online	ontlitcp	*texas1	pd1_on	
texas_online	ontlitcp	*123.34.6.81	pd1_on	
texas_online	ontlitcp	texas1	pd1_on	
texas_online	ontlitcp	123.34.6.81	pd1_on	

The client application ignores the wildcard in the **hostname** field.

The client application on **kansas** can use any *one* of the entries shown in Figure 2-5 in its **sqlhosts** file.

Important: *You can include only one of these entries in your **sqlhosts** file.*

Figure 2-5
Possible entries in the sqlhosts file for the client application on kansas

servername	nettype	hostname	servicename	options
texas_online	ontlitcp	*texas2	pd1_on	
texas_online	ontlitcp	*123.34.6.82	pd1_on	
texas_online	ontlitcp	texas2	pd1_on	
texas_online	ontlitcp	123.34.6.82	pd1_on	

The servicename Field

The **servicename** field tells the network software how to find the database server on a specified host.

For the TCP/IP network protocol, you can use the actual TCP listen port number as the service name. The TCP port number is in the **port#** field of the **/etc/services** file. You can also use the UNIX **arp** or **ypmatch** command to find the port number. Figure 2-6 shows a sample **/etc/services** file.

Figure 2-6
A sample /etc/services file

servicename	port # /protocol	aliases
sales_ol	1536 / tcp	
olport	1425 / tcp	port5

Using the port number from Figure 2-6, the two **sqlhosts** entries in Figure 2-7 on page 2-12 are equivalent.

Figure 2-7
Comparison of two sqlhosts entries

servername	nettype	hostname	servicename	options
sales	ontlitcp	knight	sales_ol	
sales	ontlitcp	knight	1536	

Using the actual port number saves time when you make a connection. However, as with the IP address in the **hostname** field, using the actual port number might make administration of the **sqlhosts** file less convenient.

When the **nettype** field specifies a shared-memory connection (**onipcshm**) or a stream-pipe connection (**onipcstr**), OnLine uses the value in the **servicename** field to create a file that supports the connection. For both **onipcshm** and **onipcstr** connections, the **servicename** can be any short group of letters that is unique for all OnLine instances on the same computer. Informix recommends that you use the **dbservername** as the **servicename** for stream-pipe connections.

The options Field

The **options** field enhancement to the **sqlhosts** file provides additional flexibility in specifying connections. You can use the options described in this section with both OnLine and INFORMIX-SE.

The **options** field includes entries for the following features:

- The keep-alive option
- The security options
- The buffer-size option

When you change the options in the **sqlhosts** file, those changes affect the next connection a client application makes. You do not need to stop and restart the client application to allow the changes to take effect; however, a database server only reads its own **sqlhosts** entry during initialization. If you change the options for the database server, you must reinitialize the database server to allow the changes to take effect.

Important: *This behavior is different from the behavior of environment variables. If you change an environment variable, the change does not take effect until you reinitialize the database server.*

The **sqlhosts** file in Figure 2-8 shows examples of the keep-alive, security, and buffer-size options that are discussed in the following sections. On line 1, the `k=0` in the **options** field disables the keep-alive feature. The `r=0` disables **~/.netrc** file lookup for the client. On line 2, the `s=2` enables **/etc/hosts.equiv** lookup for the database server, and the `b=5120` sets the communications buffer size to 5120 kilobytes.

Figure 2-8
A sample sqlhosts file

	servername	**nettype**	**hostname**	**servicename**	**options**
Line 1	payroll	ontlitcp	dewar	py1	k=0,r=0
Line 2	personnel	ontlispx	skinner	prsnl_ol	s=2,b=5120

The keep-alive Option

The keep-alive option is a network option that TCP/IP and IPX/SPX use. It does not affect shared-memory, stream-pipe, or named-pipe connections.

The letter *k* identifies keep-alive entries in the **options** field, as follows:

```
k=0     disable the keep-alive feature
k=1     enable the keep-alive feature
```

When a connected client and server are not exchanging data, the keep-alive option enables the network service to periodically check the connection. If the receiving end of the connection does not respond within the time specified by the parameters of your operating system (you cannot modify these parameters), the connection is considered broken and all resources related to the connection are released.

When the keep-alive option is enabled, the operating system reserves resources for the connection until the specified time expires. If nonstandard disconnections are frequent (for example, PC users shut off the power without first exiting from the client application), the operating system might be wasting resources.

When the keep-alive option is disabled, the network service immediately detects the broken connection and frees up resources. However, this feature has a disadvantage. If the network is slow, the network service might treat a slow response as a broken connection.

When the keep-alive option is disabled, the network service immediately considers the connection broken when there are no responses from the receiving end.

If you do not include the keep-alive option in the **options** field, the keep-alive feature is enabled by default. You can set this option on the server side only, the client side only, or on both sides. For most cases, Informix recommends that you enable the keep-alive option.

The Security Option

The security option lets you disable the operating system security-file lookup. The letter *s* identifies server-side settings and the letter *r* identifies client-side settings. You can set both options in the **options** field. A client ignores *s* settings and a database server ignores *r* settings.

The following table shows the possible settings for r and s:

Setting	Result
r=0	Disables the **~/.netrc** lookup from the client side
r=1	Enables the **~/.netrc** lookup from the client side (default setting for the client side)
s=0	Disables both **/etc/hosts.equiv** and **~/.rhosts** lookup from the server side
s=1	Enables only the **/etc/hosts.equiv** lookup from the server side
s=2	Enables only the **~/.rhosts** lookup from the server side
s=3	Enables both **/etc/hosts.equiv** and **~/.rhosts** lookup on the server side (default setting for the server side)

The security options let you control the way a client (user) gains access to a database server by modifying the **sqlhosts** file. By default, an Informix database server searches the **/etc/hosts.equiv** and the **~/.rhosts** file of the user to determine whether a client host is trusted. With the security options, you can specifically enable or disable the use of either or both of the **hosts.equiv** and **.rhosts** files.

For example, if you want to prevent end users from specifying trusted hosts in their own **~/.rhosts** file, you can disable the **~/.rhosts** lookup by setting `s=1` in the **options** field of the **sqlhosts** file for the database server.

Important: *Do not disable the* ***/etc/hosts.equiv*** *lookup in database servers that are used in distributed database operations. That is, if you expect to use the database server in distributed processing, do not set* `s=0` *or* `s=2`.

The Buffer-Size Option

Use the buffer-size option (`b=`) to specify the space (in bytes) reserved for the communications buffer. The buffer-size option applies only to connections that use the TCP/IP network protocol. IPX/SPX, shared-memory, and stream-pipe connections ignore the buffer-size setting.

You can use this option when the default size is not efficient for a particular application. For example, for an application that uses many 32-kilobyte blobs, you could set the size of the communications buffer to 32-kilobytes (assuming that the network server on your operating system supports a 32-kilobyte buffer).

Adjusting the buffer size allows you to use system and network resources more efficiently; however, if the buffer size is set too high, the user receives a connection-reject error because no memory can be allocated. For example, if you set `b=64000` on a system that has 1000 users, the system might require 64 megabytes of memory for the communications buffers. This setting might exhaust the memory resources of the computer.

On many operating systems, the maximum buffer size supported for TCP/IP is 16 kilobytes. To determine the maximum allowable buffer size, refer to the documentation for your operating system or contact the technical-support services of the vendor of your operating system.

If your network includes several different types of computers, be particularly careful when you change the size of the communications buffer.

Tip: *Informix recommends that you set the client-side communications buffer and the server-side communications buffer to the same value.*

Syntax Rules for the options Field

Each item in the options field has the following format:

```
letter=value
```

You can combine several items in the **options** field. The items must be separated with commas and no whitespace is allowed. You can include the items in any order. The following examples show both legal and illegal syntax:

```
k=0,s=3,b=5120          valid entry
s=3,k=0,b=5120          equivalent to the preceding entry

k = 0,b = 5120          illegal: includes spaces
k=s=0                   illegal: cannot combine entries
```

The /INFORMIXTMP Directory

OnLine generates some network-related internal files and stores them in the **/INFORMIXTMP** directory. The following list includes examples of such files:

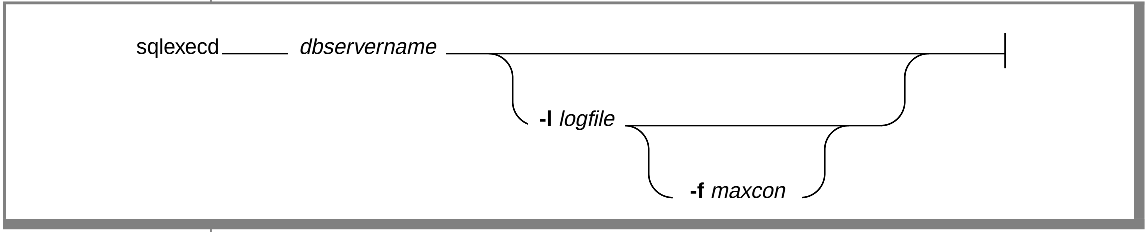
- **.inf**.*servicename*
- VP.*servername.nn***C**
- *servicename*.**str**
- *servicename*.**exp**

OnLine creates the **/INFORMIXTMP** directory at initialization time, if the directory does not already exist. You do not need to examine or edit the files in **/INFORMIXTMP**, but you do need to make sure that the UNIX system administrator knows that this directory is useful and should not be deleted.

Refer to the *INFORMIX-OnLine Dynamic Server Administrator's Guide* for more information about these files.

Size Option for the INFORMIX-SE sqlexecd Log File

This release of SE introduces a change in the behavior of the log file associated with the **sqlexecd** daemon. In earlier versions of SE, the only way to limit the length of the log file was to terminate the **sqlexecd** daemon and remove (or edit) the file. If you use the new **-f** option with the **sqlexecd** command, the log file no longer is allowed to grow indefinitely.



dbservername	is the name of the database server.
-l *logfile*	specifies the name of a file where all client-connection activity is recorded.
-f *maxcon*	specifies the maximum number of connection requests that can be recorded in the log file. The value of *maxcon* must be between 0 and 32767.

The -l Option

If the *logfile* specified with the **-l** option does not exist, the **sqlexecd** daemon creates a log file and gives it the name specified in the **-l** option. If the log file already exists, **sqlexecd** appends new information to the existing log file.

If you use a log file and do not use the **-f** option, you must make sure that the file system has sufficient space for saving new client information about connection activity.

You can specify the *logfile* as a simple filename, in which case the **sqlexecd** daemon places the log file in the current directory. You can also specify the *logfile* as a full pathname so that the log is stored in the specified directory.

The -f Option

The **-f** option lets you specify the maximum number (*maxcon*) of connection-request records that can be stored in the log file. When the number of records reaches the maximum, **sqlexecd** renames *logfile* to *logfile***.old** and creates a new *logfile*. When *logfile* again fills up, **sqlexecd** again renames *logfile* to *logfile***.old** and creates a new *logfile*.

If you specify an illegal value for *maxcon*, **sqlexecd** automatically uses 10,000 connection requests as the default value and the log file is not allowed to grow past 10,000 connections, or about 680 kilobytes. If you do not use the **-f** option, or if for some reason **sqlexecd** is not able to rename the log file when it becomes full, the log file is allowed to grow indefinitely.

The **sqlexecd** daemon does not keep creating more and more old files. If you want to save all of the connection records, you can periodically copy *logfile***.old** into another file that will not be overwritten.

The following example starts the **sqlexecd** daemon for a database server named **myserver**. The command starts the daemon with a log file named **mylog** in the current directory and 2000 as the maximum number of connection records.

```
sqlexecd myserver -l mylog -f 2000
```

Calculating the Maximum Size of the Log File

The number of connection records specified with the **-f** option is *not* the size of the log file. It is the number of connection requests that can be recorded in the log file. The length of each connection request is about 70 characters. You can calculate the approximate maximum size (in bytes) of the log file with the following calculation:

```
maximum_logfile_size = maxcon * 70
```

Tip: *This maximum size is not an exact value because connection-request records are not always exactly 70 bytes long. In certain cases, the connection-request record is much longer or shorter than 70 bytes. For example, some of the records contain the full path of the database you selected. If the pathname is long, the log record might be longer than 70 bytes.*

Chapter

SQL Enhancements

3

This chapter describes the new and changed SQL statements in this release. It also describes other new and changed SQL items such as SQL segments, system catalog tables, environment variables, utilities, and the SQL Communications Area (SQLCA).

For descriptions of new and changed SQL error messages, see the "Error Messages" section at the end of this guide.

How to Use This Chapter

This section covers the following topics:

- Scope of descriptions of SQL items
- Relationship of this chapter to the Version 7.1UC1 SQL manuals
- Organization of this chapter
- New and changed information by SQL item type

Scope of Descriptions

This chapter describes the new and changed SQL items in this release.

New items are described in their entirety. For example, the descriptions of new SQL statements include a statement of purpose, a syntax diagram, a syntax table, rules of usage, examples, and references to related items.

For changed items, usually only the additions and modifications are described. For example, if a changed statement has a changed syntax diagram for one clause, the description includes the changed syntax diagram for that clause and an explanation of the change but does not include syntax diagrams for unchanged clauses.

In a few cases, such as the REVOKE statement and the **dbschema** utility, a changed item is described in its entirety. In these cases the changes to the item are so extensive that a complete description is warranted.

Relationship of This Chapter to SQL Manuals

This chapter supplements the information in the Version 7.1UC1 SQL manuals:

- *SQL Quick Syntax Guide*, Version 7.1UD1
- *Informix Guide to SQL: Reference*, Version 7.1UC1
- *Informix Guide to SQL: Syntax*, Version 7.1UC1
- *Informix Guide to SQL: Tutorial*, Version 7.1UC1

Use this chapter with the SQL manuals to get current and complete information about SQL items. In particular, use the SQL manuals to get complete information for existing items that have changed. Because the descriptions of such items in this chapter cover only the changes to the item, you need to consult the appropriate SQL manual for information about the unchanged parts of the item.

See the next section for a table that lists the sections in this chapter and the corresponding SQL manual and chapter for each section.

Organization of This Chapter

This chapter is divided into several sections. Each section covers a particular type of SQL item and begins with a summary and list of the new and changed items within the section. New and changed SQL items appear in alphabetical order within the section.

The following table lists the sections of this chapter. For each section, the table also lists the title and chapter of the Version 7.1UC1 SQL manual that describes SQL items of this type.

The following abbreviations are used in this table:

- SQLR stands for the *Informix Guide to SQL: Reference*.
- SQLS stands for the *Informix Guide to SQL: Syntax*.
- SQLT stands for the *Informix Guide to SQL: Tutorial*.

These abbreviations also appear in the syntax diagrams in this chapter.

Section in This Chapter	Corresponding SQL Manual	Corresponding SQL Manual Chapter
"New and Changed SQL Statements"	SQLS	Chapter 1, "SQL Statements"
"Changed SQL Segments"	SQLS	Chapter 1, "SQL Statements"
"New and Changed System Catalog Tables"	SQLR	Chapter 2, "System Catalog"
"New and Changed Environment Variables"	SQLR	Chapter 4, "Environment Variables"
"Changed Utilities"	SQLR	Chapter 5, "SQL Utilities"
"Changes to the SQL Communications Area"	SQLT	Chapter 5, "Programming with SQL"

New and Changed SQL Statements

This section describes new and changed SQL statements.

The following SQL statements are new in this release:

- CREATE ROLE
- DROP ROLE
- GRANT FRAGMENT
- RENAME DATABASE
- REVOKE FRAGMENT
- SET
- SET ROLE
- SET SESSION AUTHORIZATION
- START VIOLATIONS TABLE
- STOP VIOLATIONS TABLE

The following SQL statements are changed in this release.

- ALTER TABLE
- CONNECT
- CREATE INDEX
- CREATE PROCEDURE
- CREATE TABLE
- CREATE TRIGGER
- DATABASE
- DROP TABLE
- GET DIAGNOSTICS
- GRANT
- REVOKE
- SELECT
- UPDATE STATISTICS

ALTER TABLE

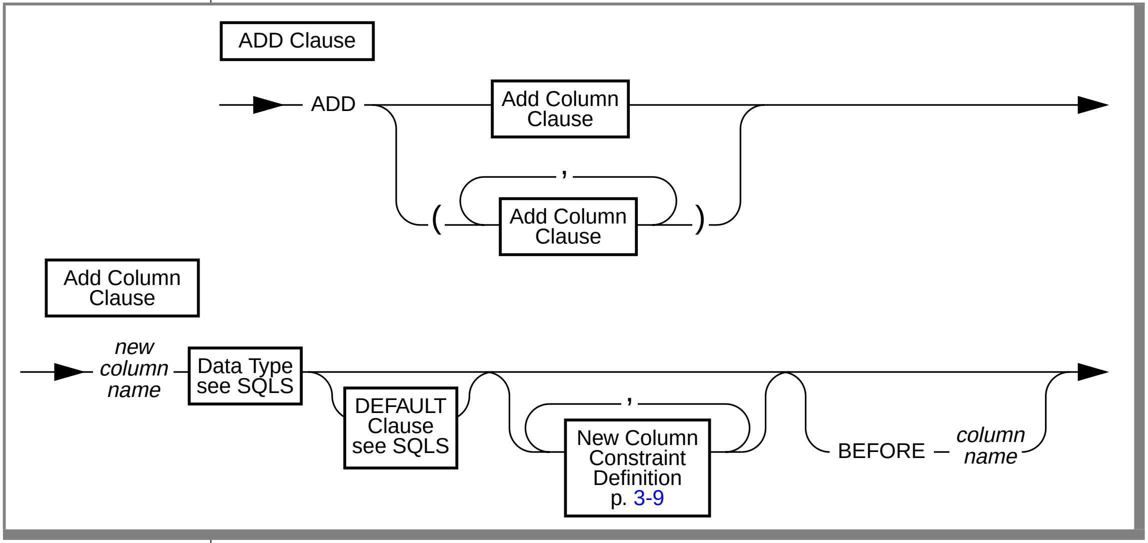
The ALTER TABLE statement has several changes in syntax and behavior in this release. The changed clauses and options are described on the following pages.

Usage

The following new general restrictions apply to the use of the ALTER TABLE statement:

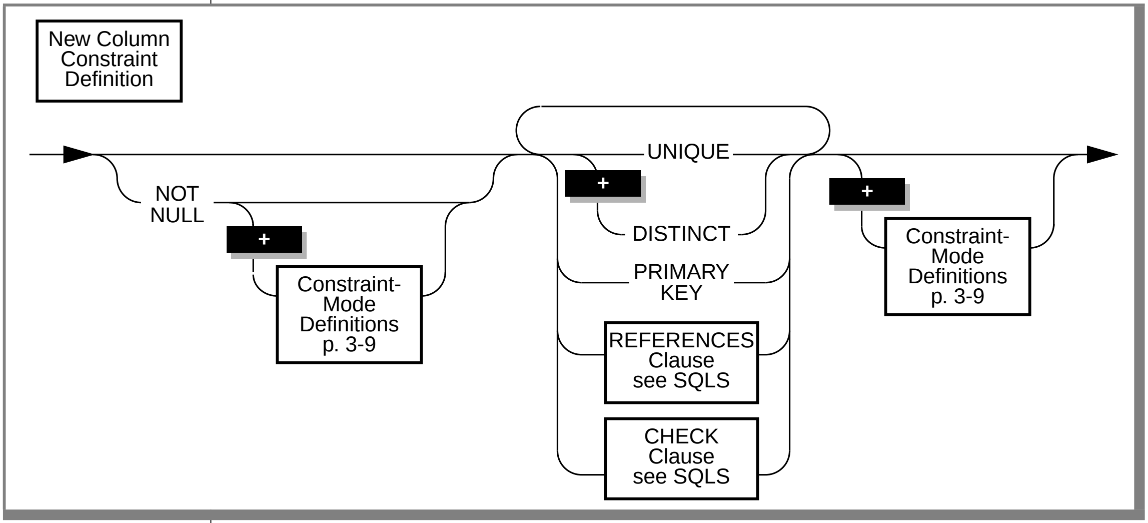
- You cannot add, drop, or modify a column if the table that contains the column has a violations and diagnostics table associated with it.
- You cannot alter a violations or diagnostics table.
- You cannot add a constraint to a violations or diagnostics table.
- Previously, you only needed the Alter table-level privilege to alter a table. Now, to use the ALTER TABLE statement, you must be granted both the Alter table-level privilege and the Resource database-level privilege on the database that contains the table.

ADD Clause



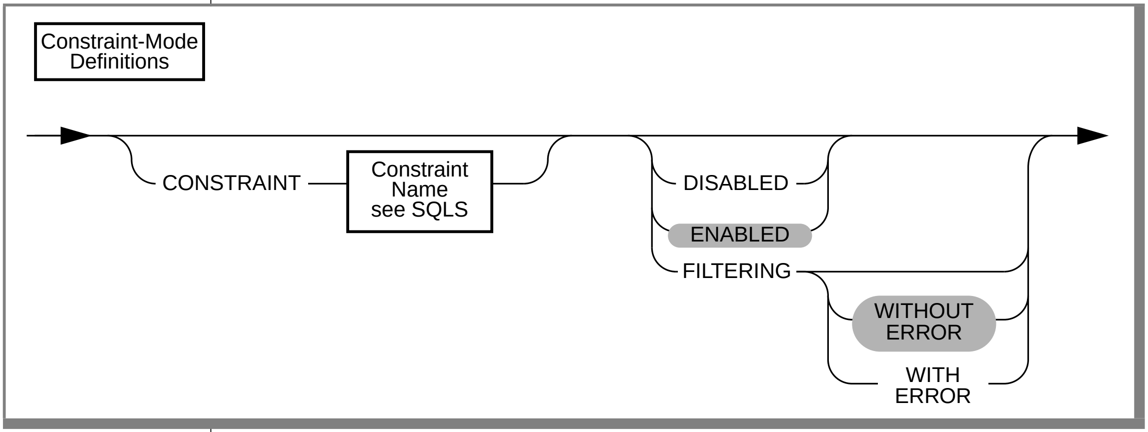
The NOT NULL keywords have been removed from the ADD clause in this release.

New Column Constraint Definition


New Column Constraint Definition
NOT NULL
+
Constraint-Mode Definitions p. 3-9
UNIQUE
DISTINCT
PRIMARY KEY
REFERENCES Clause see SQLS
CHECK Clause see SQLS
Constraint-Mode Definitions p. 3-9

You can now assign a constraint name to the NOT NULL keywords, and you can set the object mode of the not null constraint. In addition, you can now set the object mode of any type of constraint.

Constraint Mode Definitions



New Capabilities

The Constraint-Mode Definitions option expresses two new capabilities:

- You can assign a name to a not null constraint on a column, and you can set the not null constraint to one of the following object modes: disabled, enabled, or filtering.
- You can set any type of column-level constraint or table-level constraint to the disabled, enabled, or filtering object modes.

Description of Constraint Modes

You can set constraints to the following modes: disabled, enabled, or filtering. If you choose the filtering mode, you can specify the WITHOUT ERROR or WITH ERROR options. These modes and options are described in the following list:

disabled	A constraint created in disabled mode is not enforced during insert, delete, and update operations.
enabled	A constraint created in enabled mode is enforced during insert, delete, and update operations. If a target row causes a violation of the constraint, the statement fails.
filtering	A constraint created in filtering mode is enforced during insert, delete, and update operations. If a target row causes a violation of the constraint, the statement continues processing, but the bad row is written to the violations table associated with the target table. Diagnostic information about the constraint violation is written to the diagnostics table associated with the target table.

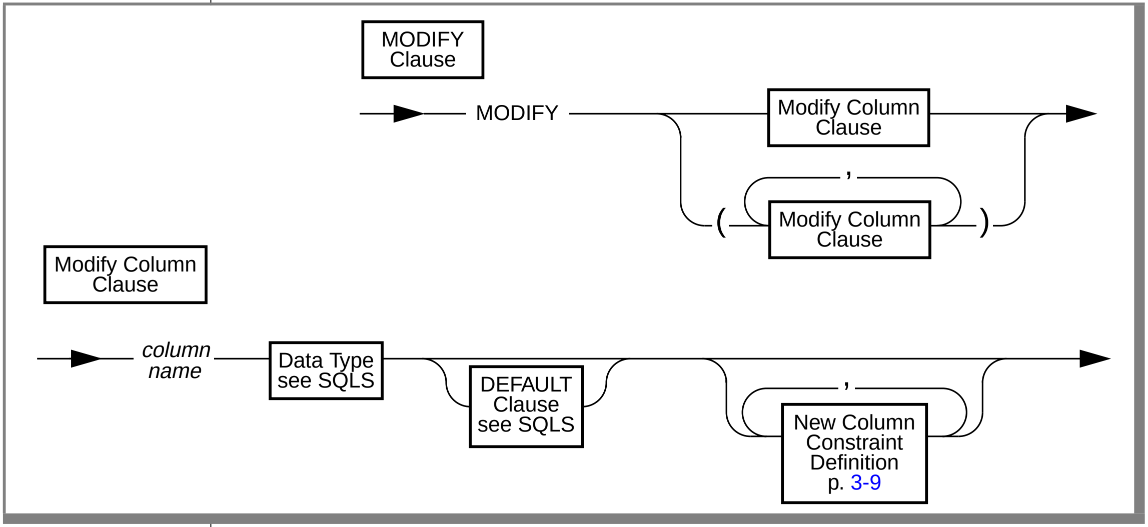
WITHOUT ERROR	When a filtering mode constraint is violated during an insert, delete, or update operation, no integrity-violation error is returned to the user.
WITH ERROR	When a filtering mode constraint is violated during an insert, delete, or update operation, an integrity-violation error is returned to the user.

Using Constraint Modes

You must observe the following rules when you use constraint modes:

- If you do not specify the object mode of a column-level or table-level constraint explicitly, the default mode is enabled.
- If you do not specify the with error or without error option for a filtering mode constraint, the default error option is without error.
- When you add a column-level or table-level constraint to a table and specify the disabled object mode for the constraint, your ALTER TABLE statement succeeds even if existing rows in the table violate the constraint.
- When you add a column-level or table-level constraint to a table and specify the enabled or filtering object mode for the constraint, your ALTER TABLE statement succeeds provided that no existing rows in the table violate the new constraint. However, if any existing rows in the table violate the constraint, your ALTER TABLE statement fails and returns an error.
- When you add a column-level or table-level constraint to a table in the enabled or filtering object mode, and existing rows in the table violate the constraint, erroneous rows in the base table are not filtered to the violations table. Thus, you cannot use a violations table to detect the erroneous rows in the base table.

MODIFY Clause



The NOT NULL keywords have been removed from the MODIFY clause in this release.

Adding a Constraint When Existing Rows Violate the Constraint

If you use the MODIFY clause to add a constraint in the enabled mode and receive an error message because existing rows would violate the constraint, you can take the following steps to add the constraint successfully:

1. Add the constraint in the disabled mode.

   Issue the ALTER TABLE statement again, but this time specify the DISABLED keyword in the MODIFY clause.

2. Start a violations and diagnostics table for the target table with the START VIOLATIONS TABLE statement.

3. Issue a SET statement to switch the object mode of the constraint to the enabled mode.

   When you issue this statement, existing rows in the target table that violate the constraint are duplicated in the violations table; however, you receive an integrity-violation error message, and the constraint remains disabled.

4. Issue a SELECT statement on the violations table to retrieve the nonconforming rows that are duplicated from the target table.

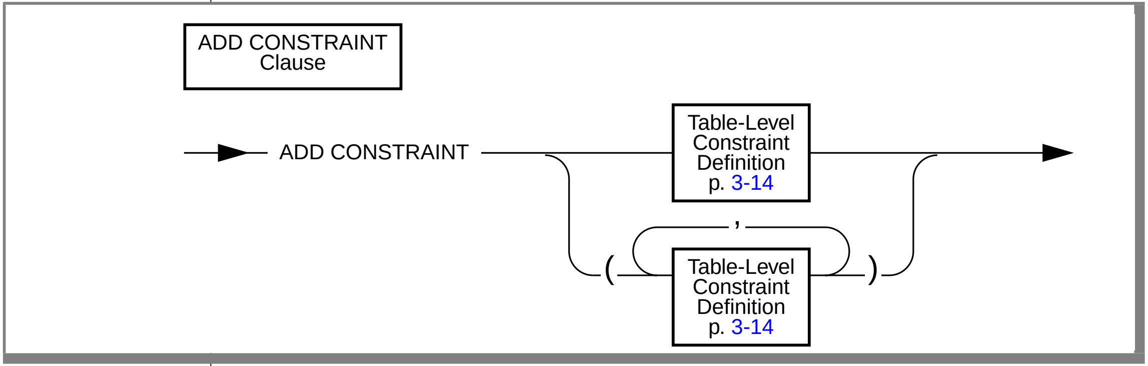
   You might need to join the violations and diagnostics tables to get all the necessary information.

5. Take corrective action on the rows in the target table that violate the constraint.

6. After you fix all the nonconforming rows in the target table, issue the SET statement again to switch the disabled constraint to the enabled mode.

   This time the constraint is enabled and no integrity-violation error message is returned because all rows in the target table now satisfy the new constraint.

ADD CONSTRAINT Clause


ADD CONSTRAINT Clause
ADD CONSTRAINT
Table-Level Constraint Definition p. 3-14
,
(
Table-Level Constraint Definition p. 3-14
)

Changes to the ADD CONSTRAINT Clause

You can now set the object mode of a table-level constraint with the ADD CONSTRAINT clause.

If you use the ADD CONSTRAINT clause to add a constraint in the enabled mode and receive an error message because existing rows would violate the constraint, you can follow a procedure to add the constraint successfully. See "Adding a Constraint When Existing Rows Violate the Constraint" on page 3-12.

Table-Level Constraint Definition

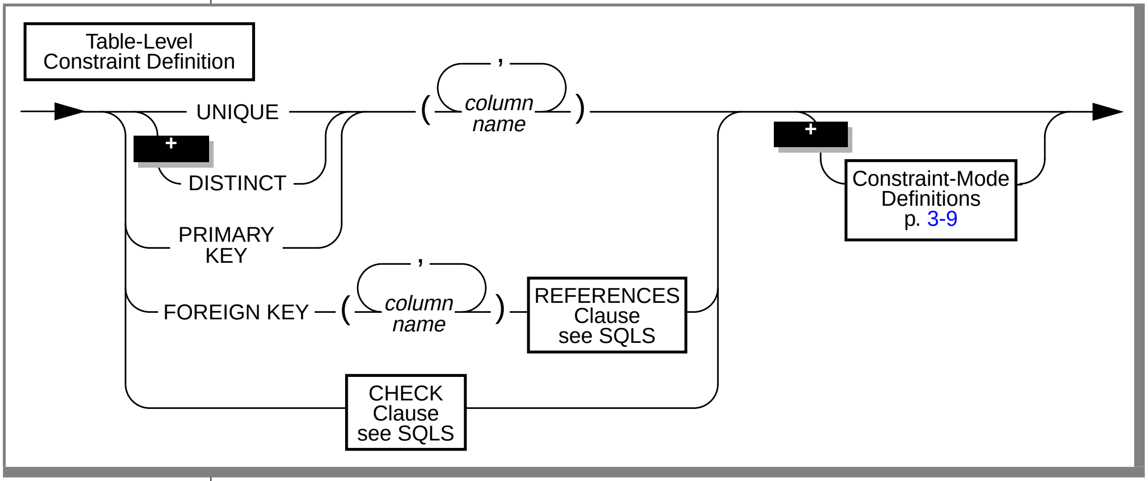


Table-Level Constraint Definition
UNIQUE
+
DISTINCT
PRIMARY KEY
FOREIGN KEY
,
column name
(
)
REFERENCES Clause see SQLS
CHECK Clause see SQLS
+
Constraint-Mode Definitions p. 3-9

Element	Purpose	Restrictions	Syntax
column name	The name of the column or columns on which the constraint is placed	If you are using OnLine, the maximum number of columns is 16, and the total length of all the columns cannot exceed 255 bytes. If you are using SE, the maximum number of columns is 8, and the total length of all the columns cannot exceed 120 bytes.	Identifier segment, *Informix Guide to SQL: Syntax*

The definition of a table-level constraint now includes the ability to set the object mode of the constraint.

DROP CONSTRAINT Clause

You can now use the DROP CONSTRAINT clause to drop a not null specification from a column definition because the database server treats any not null specification as a formal constraint.

If you do not assign a name to a not null constraint when you define the constraint in an ALTER TABLE or CREATE TABLE statement, the database server generates a name for the constraint, just as it generates a name for any other type of constraint that the user has not named explicitly.

To find out the system-generated or user-created name for any not null constraint, you can query the **sysconstraints** or **syscoldepend** system catalog table. For an example of such a query, see the DROP CONSTRAINT clause of the ALTER TABLE statement in the *Informix Guide to SQL: Syntax*.

References

See the SET, START VIOLATIONS TABLE, and STOP VIOLATIONS TABLE statements in this guide.

CONNECT

The CONNECT statement is changed in this release. The CONNECT statement is fully described in the *Informix Guide to SQL: Syntax*.

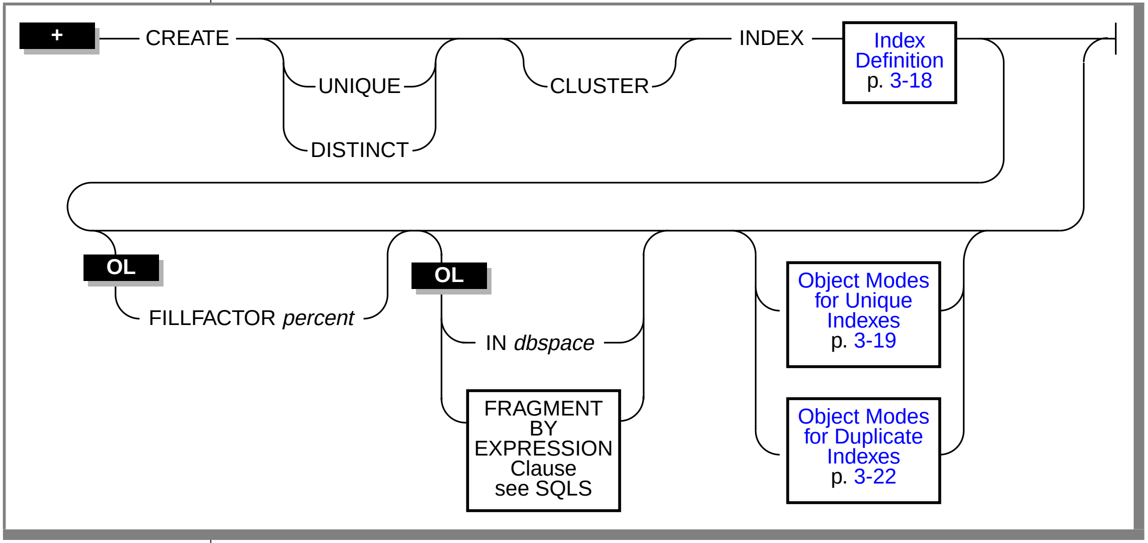
The user who executes the CONNECT statement cannot have the same user name as an existing role in the database.

The current user, or PUBLIC, must have the Connect database privilege for the database specified in the CONNECT statement.

CREATE INDEX

The CREATE INDEX statement has changed syntax and behavior in this release. The changes to this statement are described on the following pages.

Syntax

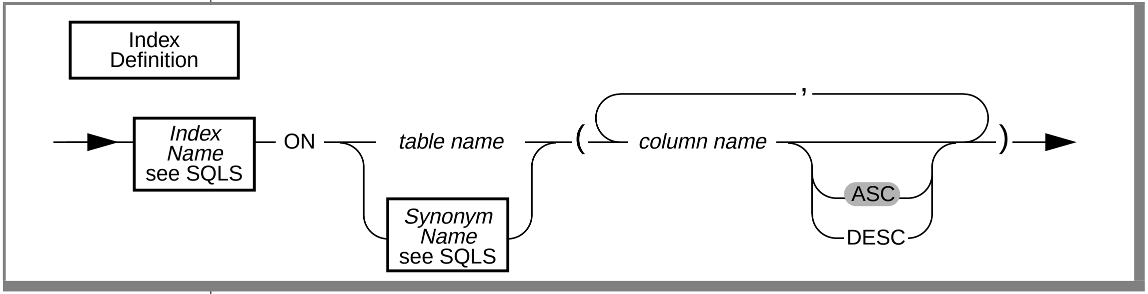

+
CREATE
UNIQUE
DISTINCT
CLUSTER
INDEX
Index Definition p. 3-18
OL
FILLFACTOR percent
OL
IN dbspace
FRAGMENT BY EXPRESSION Clause see SQLS
Object Modes for Unique Indexes p. 3-19
Object Modes for Duplicate Indexes p. 3-22

Element	Purpose	Restrictions	Syntax
dbspace	The name of the dbspace in which you want to place the index	The dbspace must exist at the time you execute the statement.	Identifier segment, *Informix Guide to SQL: Syntax*
percent	The percentage of each index page that is filled by index data when the index is created. The default value is 90.	Value must be in the range 1-100.	Literal Number segment, *Informix Guide to SQL: Syntax*

Usage

You can now set the object mode of a unique index or a duplicate index in the CREATE INDEX statement.

Index Definition

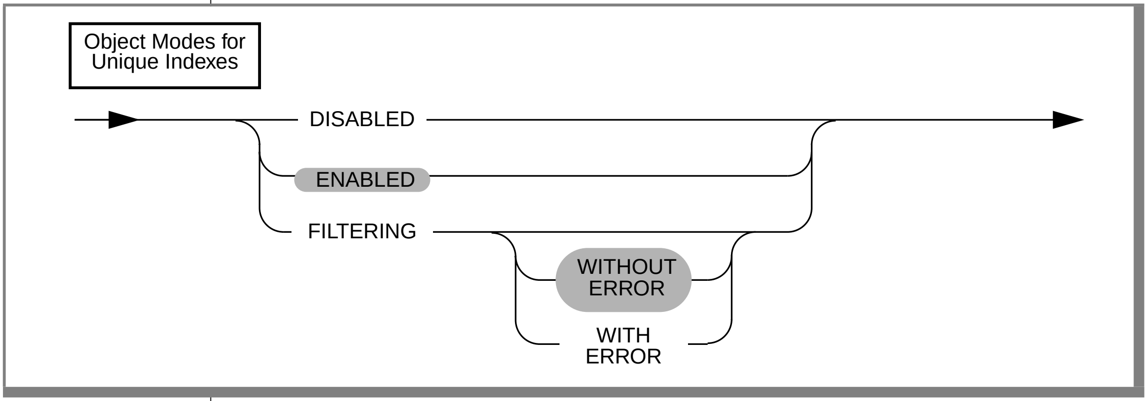


Element	Purpose	Restrictions	Syntax
column name	The name of the column or columns which you want to index	You must observe restrictions on the location of the columns, the maximum number of columns, the total width of the columns, existing constraints on the columns, and the number of indexes allowed on the same columns. See the CREATE INDEX statement in the *Informix Guide to SQL: Syntax* for an explanation of these restrictions.	Identifier segment, *Informix Guide to SQL: Syntax*
table name	The name of the table on which the index is created	The table must exist. The table can be a regular database table or a temporary table.	Identifier segment, *Informix Guide to SQL: Syntax*

In the Index Definition portion of the CREATE INDEX statement, you give a name to the index, specify the table on which the index is created, and specify the column or columns to be used for the index. In addition, the ASC and DESC keywords allow you to specify whether the index will be sorted in ascending or descending order.

See the *Informix Guide to SQL: Syntax* for a complete description of these aspects of the Index Definition.

Object Modes for Unique Indexes



Description of Object Modes for Unique Indexes

You can set unique indexes in the following modes: disabled, enabled, and filtering. If you choose the filtering mode, you can also specify the WITHOUT ERROR or WITH ERROR options. The following list explains these modes and options:

disabled	A unique index created in disabled mode is not updated after insert, delete, and update operations that modify the base table. Because the contents of the disabled index are not up to date, the optimizer does not use the index during the execution of queries.
enabled	A unique index created in enabled mode is updated after insert, delete, and update operations that modify the base table. Because the contents of the enabled index are up to date, the optimizer uses the index during the execution of queries. If an insert or update operation causes a duplicate key value to be added to a unique enabled index, the statement fails.

filtering	A unique index created in filtering mode is updated after insert, delete, and update operations that modify the base table. Because the contents of the filtering mode index are up to date, the optimizer uses the index during the execution of queries. If an insert or update operation causes a duplicate key value to be added to a unique index in filtering mode, the statement continues processing, but the bad row is written to the violations table associated with the base table. Diagnostic information about the unique- index violation is written to the diagnostics table associated with the base table.

WITHOUT ERROR	When a unique-index violation occurs during an insert or update operation, no integrity-violation error is returned to the user. You can specify this option only with the filtering object mode.
WITH ERROR	When a unique-index violation occurs during an insert or update operation, an integrity-violation error is returned to the user. You can specify this option only with the filtering object mode.

Specifying Object Modes for Unique Indexes

You must observe the following rules when you specify object modes for unique indexes in CREATE INDEX statements:

- You can set a unique index to the enabled, disabled, or filtering modes.
- If you do not specify the object mode of a unique index explicitly, the default mode is enabled.
- If you do not specify the with error or without error option for a filtering-mode unique index, the default error option is without error.
- When you add a new unique index to an existing base table and specify the disabled object mode for the index, your CREATE INDEX statement succeeds even if duplicate values in the indexed column would cause a unique-index violation.

- When you add a new unique index to an existing base table and specify the enabled or filtering object mode for the index, your CREATE INDEX statement succeeds provided that no duplicate values exist in the indexed column that would cause a unique-index violation. However, if any duplicate values exist in the indexed column, your CREATE INDEX statement fails and returns an error.
- When you add a new unique index to an existing base table in the enabled or filtering mode, and duplicate values exist in the indexed column, erroneous rows in the base table are not filtered to the violations table. Thus, you cannot use a violations table to detect the erroneous rows in the base table.

Adding a Unique Index When Duplicate Values Exist in the Column

If you attempt to add a unique index in the enabled mode and receive an error message because duplicate values are in the indexed column, you can take the following steps to add the index successfully:

1. Add the index in the disabled mode.

   Issue the CREATE INDEX statement again, but this time specify the DISABLED keyword.
2. Start a violations and diagnostics table for the target table with the START VIOLATIONS TABLE statement.
3. Issue a SET statement to switch the object mode of the index to the enabled mode.

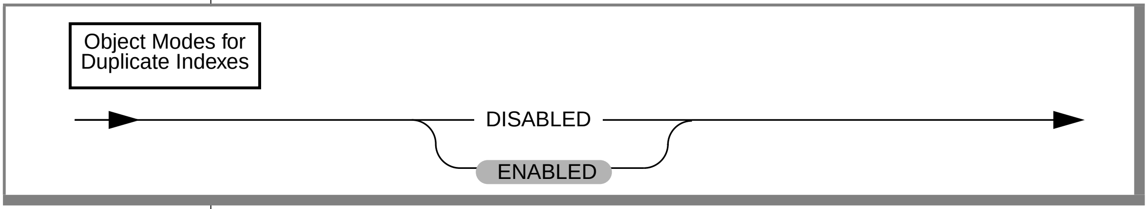
   When you issue this statement, existing rows in the target table that violate the unique-index requirement are duplicated in the violations table. However, you receive an integrity-violation error message, and the index remains disabled.
4. Issue a SELECT statement on the violations table to retrieve the nonconforming rows that are duplicated from the target table.

   You might need to join the violations and diagnostics tables to get all the necessary information.

5. Take corrective action on the rows in the target table that violate the unique-index requirement.
6. After you fix all the nonconforming rows in the target table, issue the SET statement again to switch the disabled index to the enabled mode.

   This time the index is enabled and no integrity violation error message is returned because all rows in the target table now satisfy the new unique-index requirement.

Object Modes for Duplicate Indexes



If you create a duplicate index, you can set the object mode of the index to the disabled or enabled mode. The following list explains these modes:

disabled	A duplicate index is created in disabled mode. The disabled index is not updated after insert, delete, and update operations that modify the base table. Because the contents of the disabled index are not up to date, the optimizer does not use the index during the execution of queries.
enabled	A duplicate index is created in enabled mode. The enabled index is updated after insert, delete, and update operations that modify the base table. Because the contents of the enabled index are up to date, the optimizer uses the index during the execution of queries. If an insert or update operation causes a duplicate key value to be added to a duplicate enabled index, the statement does not fail.

Specifying Object Modes for Duplicate Indexes

You must observe the following rules when you specify object modes for duplicate indexes in CREATE INDEX statements:

- You can set a duplicate index to the enabled or disabled mode, but you cannot set a duplicate index to the filtering mode.
- If you do not specify the object mode of a duplicate index explicitly, the default mode is enabled.

How the Database Server Treats Disabled Indexes

Whether a disabled index is a unique or duplicate index, the database server effectively ignores the index during data-manipulation operations.

When an index is disabled, the database server stops updating it and stops using it during queries, but the catalog information about the disabled index is retained. So you cannot create a new index on a column or set of columns if there is already a disabled index on that column or set of columns.

Similarly, you cannot create an active (not disabled) unique, foreign-key, or primary-key constraint on a column or set of columns if the indexes needed by the active constraint exist and are disabled.

CREATE PROCEDURE

The CREATE PROCEDURE statement is changed in this release. The CREATE PROCEDURE statement is fully described in the *Informix Guide to SQL: Syntax*.

Support for Roles

You can use roles with stored procedures. You can execute role-related statements (CREATE ROLE, DROP ROLE, and SET ROLE) within a stored procedure. You can also grant privileges to roles with the GRANT statement within a procedure. Privileges that a user has acquired through enabling a role are not relinquished when a procedure is executed.

For further information about roles, see the CREATE ROLE, DROP ROLE, GRANT, REVOKE, and SET ROLE statements in this guide.

Limiting Privileges Granted to PUBLIC

When set to `yes`, the new environment variable **NODEFDAC** prevents privileges for a new procedure that is created in owner mode in a database that is not ANSI-compliant from being granted to PUBLIC.

For information about preventing privileges from being granted to PUBLIC, see the **NODEFDAC** environment variable on page 3-153.

Restrictions on a Procedure Called in a Data Manipulation Statement

The following new statements cannot be present in a stored procedure that is called as part of an INSERT, UPDATE, DELETE, or SELECT statement:

- SET
- START VIOLATIONS TABLE
- STOP VIOLATIONS TABLE

These statements are allowed, however, in stored procedures that are *not* called as part of an INSERT, UPDATE, DELETE, or SELECT statement. In other words, these statements are allowed in stored procedures that are called from an EXECUTE PROCEDURE statement.

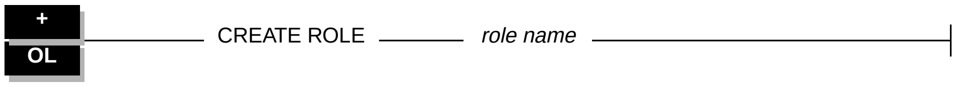
References

See the CREATE PROCEDURE statement in the *Informix Guide to SQL: Syntax* for a complete description of this statement.

CREATE ROLE

Use the CREATE ROLE statement to create a new role.

Syntax



Element	Purpose	Restrictions	Syntax
role name	Name assigned to a role created by the DBA	Maximum number of characters is eight. A role name cannot be a user name known to the database server or the operating system of the database server. A role name cannot be in the **username** column of the **sysusers** system catalog table or in the **grantor** or **grantee** columns of the **systabauth**, **syscolauth**, **sysprocauth**, **sysfragauth**, and **sysroleauth** system catalog tables.	Identifier, *Informix Guide to SQL: Syntax*

Usage

The database administrator (DBA) uses the CREATE ROLE statement to create a new role. A role can be considered as a classification, with both users and privileges granted to the role. The DBA can assign the privileges of a related work task, such as **engineer**, to a role and then grant users to that role, instead of granting every user the same set of privileges.

After a role is created, the DBA can use the GRANT statement to grant the role to users or to other roles. When a role is granted to a user, the user must use the SET ROLE statement to enable the role. Only then can the user use the privileges of the role.

The CREATE ROLE statement, used with the GRANT and SET ROLE statements, allows a DBA to create one set of privileges for a role and then grant the role to many users, instead of granting the same set of privileges to many users.

A role exists until it is dropped either by the DBA or by a user granted the role with the WITH GRANT OPTION. Use the DROP ROLE statement to drop a role.

To create the role **engineer**, enter the following statement:

```
CREATE ROLE engineer
```

References

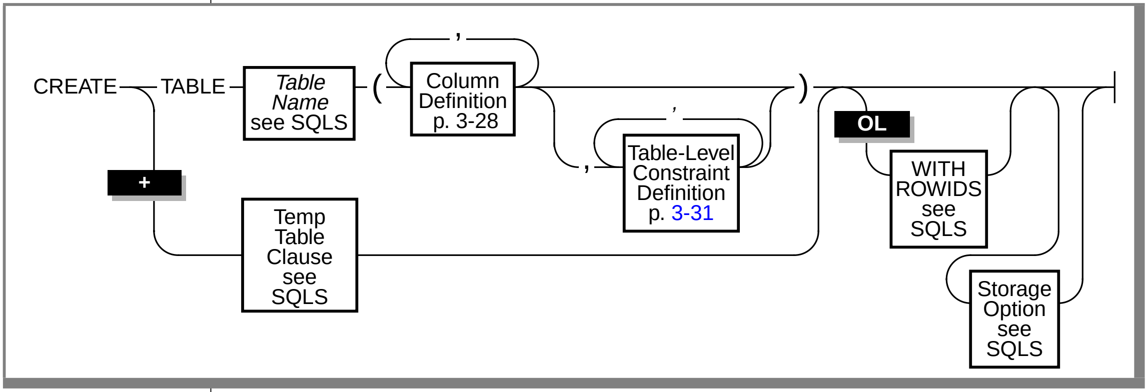
See the DROP ROLE, GRANT, REVOKE, and SET ROLE statements in this guide.

See the GRANT and REVOKE statements in the *Informix Guide to SQL: Syntax*.

CREATE TABLE

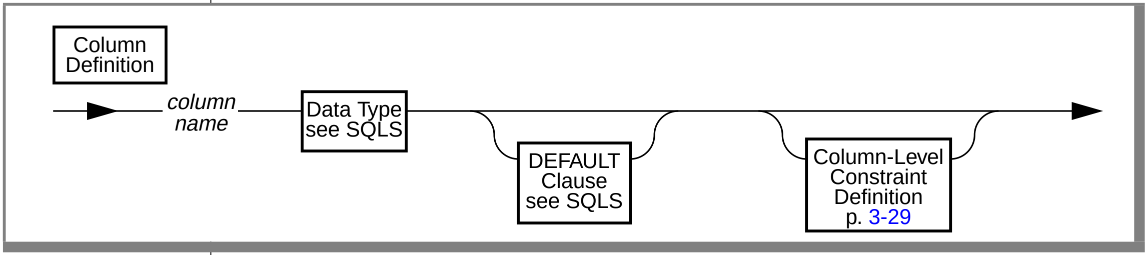
The CREATE TABLE statement has several changes in syntax and behavior in this release. The changed clauses and options are described on the following pages.

Syntax


CREATE
TABLE
Table Name see SQLS
(
,
Column Definition p. 3-28
,
Table-Level Constraint Definition p. 3-31
)
+
Temp Table Clause see SQLS
OL
WITH ROWIDS see SQLS
Storage Option see SQLS

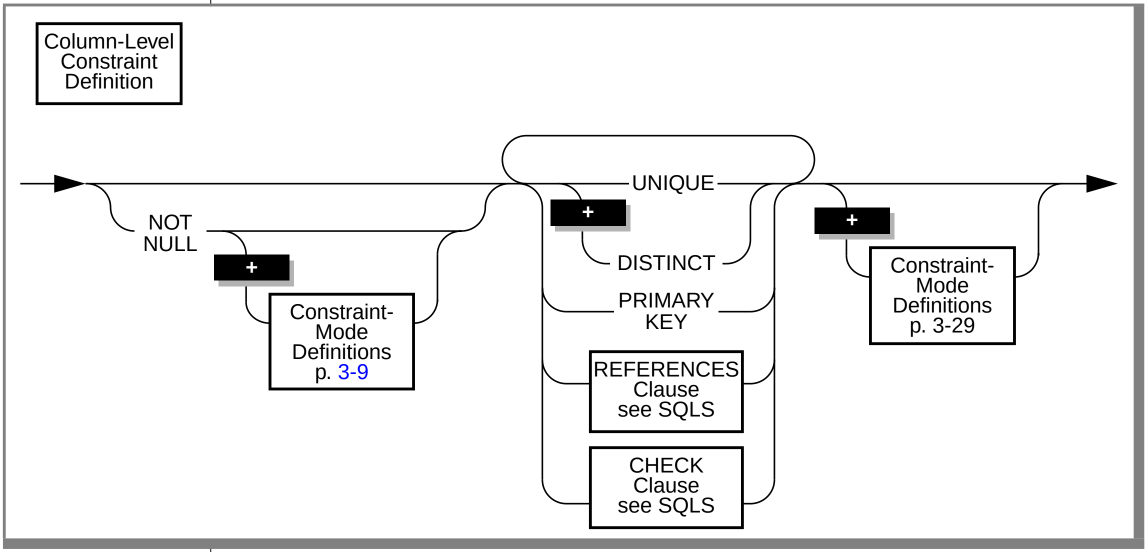
You can now set the object mode of a constraint in the CREATE TABLE statement.

Column-Definition Option



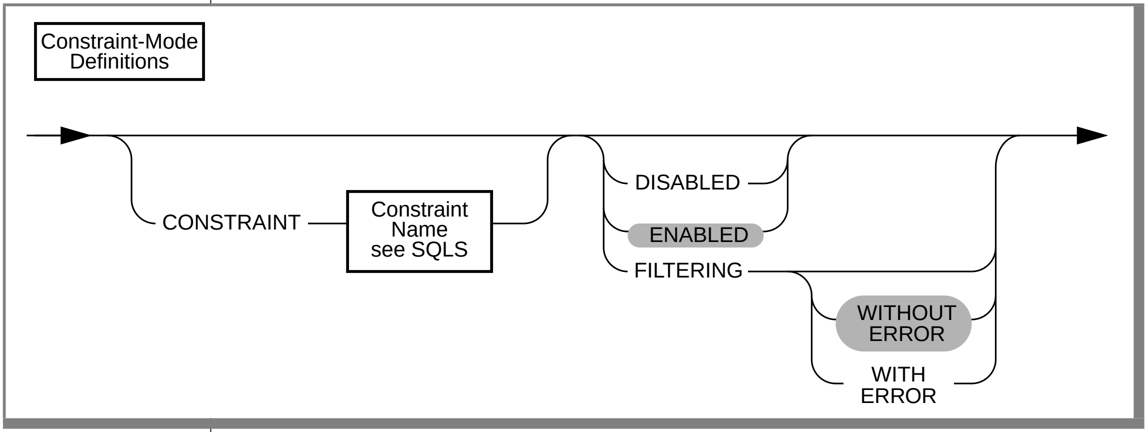
The NOT NULL keywords have been removed from the Column-Definition option in this release.

Column-Level Constraint-Definition Option


Column-Level Constraint Definition
NOT NULL
+
Constraint-Mode Definitions p. 3-9
UNIQUE
+
DISTINCT
PRIMARY KEY
REFERENCES Clause see SQLS
CHECK Clause see SQLS
+
Constraint-Mode Definitions p. 3-29

You can now assign a name to a not null constraint, and you can set the object mode of the not null constraint. In addition, you can set the object mode of any other type of constraint.

Constraint-Mode Definitions



New Capabilities

The Constraint-Mode Definitions option expresses two new capabilities:

- You can assign a name to a not null constraint on a column, and you can set the not null constraint to one of the following object modes: disabled, enabled, or filtering.
- You can set any type of column-level constraint or table-level constraint to the disabled, enabled, or filtering object modes.

Description of Constraint Modes

You can set constraints in the following modes: disabled, enabled, and filtering. If you choose filtering mode, you can specify the WITHOUT ERROR or WITH ERROR options. The following list explains these modes and options:

disabled	A constraint created in disabled mode is not enforced during insert, delete, and update operations.
enabled	A constraint created in enabled mode is enforced during insert, delete, and update operations. If a target row causes a violation of the constraint, the statement fails.
filtering	A constraint created in filtering mode is enforced during insert, delete, and update operations. If a target row causes a violation of the constraint, the statement continues processing, but the bad row is written to the violations table associated with the target table. Diagnostic information about the constraint violation is written to the diagnostics table associated with the target table.

WITHOUT ERROR	When a filtering mode constraint is violated during an insert, delete, or update operation, no integrity-violation error is returned to the user.
WITH ERROR	When a filtering mode constraint is violated during an insert, delete, or update operation, an integrity-violation error is returned to the user.

Using Constraint-Mode Definitions

You must observe the following rules concerning the use of constraint-mode definitions:

- If you do not specify the object mode of a column-level constraint or table-level constraint explicitly, the default mode is enabled.
- If you do not specify the with error or without error option for a filtering mode constraint, the default error option is without error.
- Constraints defined on temporary tables are always in the enabled mode. You cannot create a constraint on a temporary table in the disabled or filtering mode, nor can you use the SET statement to switch the object mode of a constraint on a temporary table to the disabled or filtering mode.
- You cannot assign a name to a not null constraint on a temporary table.
- You cannot create a constraint on a table that is serving as a violations or diagnostics table for another table.

Table-Level Constraint Definition Option

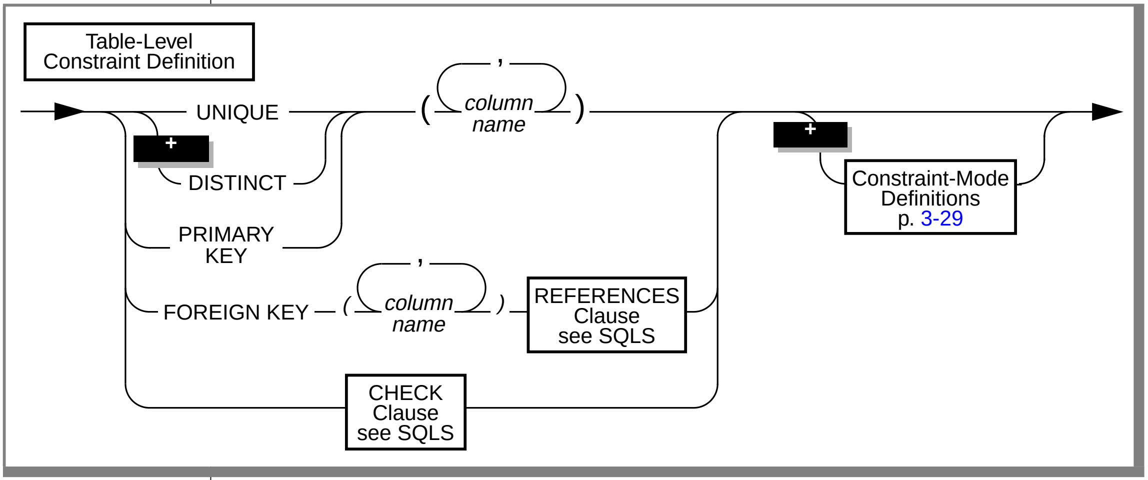


Table-Level Constraint Definition
UNIQUE
+
DISTINCT
PRIMARY KEY
FOREIGN KEY
(
,
column name
)
REFERENCES Clause see SQLS
CHECK Clause see SQLS
+
Constraint-Mode Definitions p. 3-29

You can now set the object mode of any type of table-level constraint.

Limiting Privileges Granted to PUBLIC

When set to `yes`, the new environment variable **NODEFDAC** prevents default privileges (Select, Insert, Update, Delete, and Index) on a new table in a database that is not ANSI-compliant from being granted to PUBLIC.

For information about preventing privileges from being granted to PUBLIC, see the **NODEFDAC** environment variable on page 3-153.

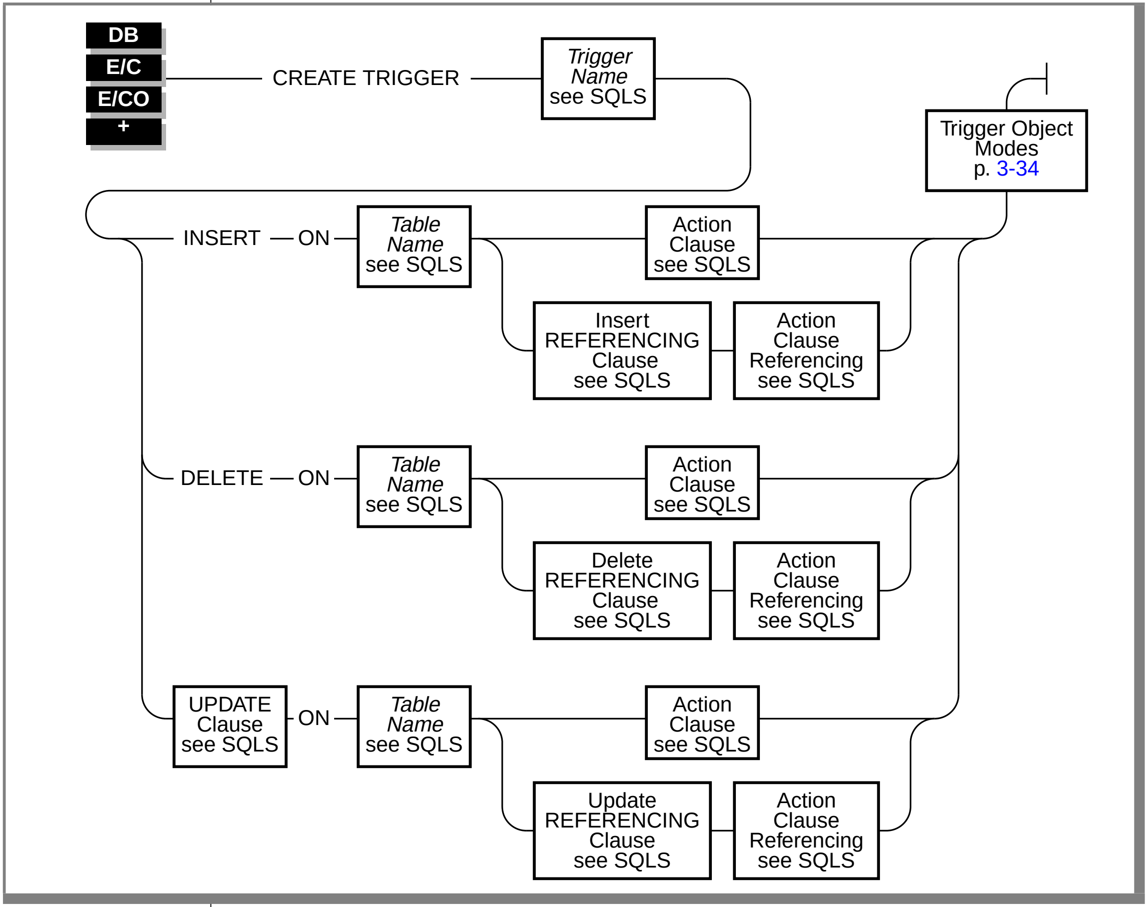
References

To understand the new functionality for specifying the object modes of database objects and detecting integrity violations in connection with these database objects, refer to the following new statements in this guide: SET, START VIOLATIONS TABLE, and STOP VIOLATIONS TABLE. See the CREATE ROLE, DROP ROLE, GRANT, REVOKE, and SET ROLE statements in this guide for further information about roles.

CREATE TRIGGER

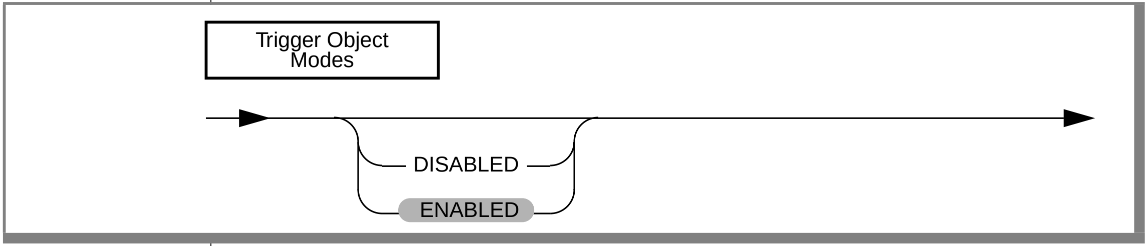
The CREATE TRIGGER statement has changes in syntax and behavior in this release. These changes are described in the following pages.

Syntax


DB
E/C
E/CO
+
CREATE TRIGGER
Trigger Name see SQLS
Trigger Object Modes p. 3-34
INSERT
ON
Table Name see SQLS
Action Clause see SQLS
Insert REFERENCING Clause see SQLS
Action Clause Referencing see SQLS
DELETE
ON
Table Name see SQLS
Action Clause see SQLS
Delete REFERENCING Clause see SQLS
Action Clause Referencing see SQLS
UPDATE Clause see SQLS
ON
Table Name see SQLS
Action Clause see SQLS
Update REFERENCING Clause see SQLS
Action Clause Referencing see SQLS

The new Trigger Object Modes option allows you to specify the object mode for the trigger that you are creating.

Trigger Object Modes



The Trigger Object Modes option allows you to create a trigger in either the enabled or disabled object mode.

Description of Object Modes

You can create triggers in the following object modes:

disabled	When a trigger is created in disabled mode, the database server does not execute the triggered action when the trigger event (an insert, delete, or update operation) takes place. In effect, the database server ignores the trigger even though its catalog information is maintained.
enabled	When a trigger is created in enabled mode, the database server executes the triggered action when the trigger event (an insert, delete, or update operation) takes place.

Specifying Object Modes for Triggers

You must observe the following rules when you specify the object mode for a trigger in the CREATE TRIGGER statement:

- If you do not specify the disabled or enabled object modes explicitly, the default object mode is enabled.
- In contrast to unique indexes and constraints of all types, you cannot set triggers to the filtering object mode because a trigger does not impose any type of data-integrity requirement on the tables in the database.

- You can use the SET statement to switch the mode of a disabled trigger to the enabled mode. Once the trigger has been re-enabled, the database server executes the triggered action whenever the trigger event takes place. However, the re-enabled trigger does not perform retroactively. The database server does not attempt to execute the trigger for rows that were inserted, deleted, or updated after the trigger was disabled and before it was enabled; therefore, you should be cautious about disabling a trigger. If disabling a trigger will eventually destroy the semantic integrity of the database, do not disable the trigger in the first place.
- You cannot create a trigger on a violations table or a diagnostics table.

Rules for Stored Procedures

The following new statements cannot be present in a stored procedure that is used as a triggered action:

- SET
- START VIOLATIONS TABLE
- STOP VIOLATIONS TABLE

For example, if an EXECUTE PROCEDURE statement is the triggered action, the procedure called by the EXECUTE PROCEDURE statement cannot contain any of these new statements. In the following CREATE TRIGGER statement, the triggered action is an EXECUTE PROCEDURE statement that calls the procedure named **upd_items_p1**:

```
CREATE TRIGGER upqty
UPDATE OF quantity ON items
BEFORE (EXECUTE PROCEDURE upd_items_p1)
```

The **upd_items_p1** procedure cannot contain the SET, START VIOLATIONS TABLE, or STOP VIOLATIONS TABLE statements in its statement block.

Support for Roles

You can use roles with triggers. Role-related statements (CREATE ROLE, DROP ROLE, and SET ROLE) can be triggered inside a trigger. Privileges that a user has acquired through enabling a role are not relinquished when a trigger is executed.

References

For further information about roles, see the CREATE ROLE, DROP ROLE, GRANT, REVOKE, and SET ROLE statements in this guide.

DATABASE

The DATABASE statement is changed in this release. The DATABASE statement is fully described in the *Informix Guide to SQL: Syntax*.

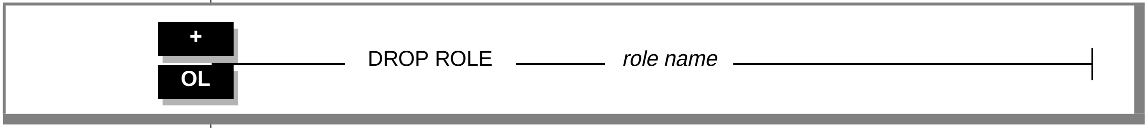
The user who executes the DATABASE statement cannot have the same user name as an existing role in the database.

The current user, or PUBLIC, must have the Connect database privilege for the database that is specified in the DATABASE statement.

DROP ROLE

Use the DROP ROLE statement to remove a previously created role.

Syntax



Element	Purpose	Restrictions	Syntax
role name	Name of the role being dropped	The role name must have been created with the CREATE ROLE statement	Identifier, *Informix Guide to SQL: Syntax*

Usage

The DROP ROLE statement is used to remove an existing role. Either the DBA or a user granted the role with the WITH GRANT OPTION can issue the DROP ROLE statement.

After a role is dropped, the privileges associated with that role, such as table-level privileges or fragment-level privileges, are dropped and a user cannot grant or enable a role. If a user is using the privileges of a role when the role is dropped, the user automatically loses those privileges.

A role exists until either the DBA or a user granted the role with the WITH GRANT OPTION uses the DROP ROLE statement to drop the role.

The following statement drops the role **engineer**:

```
DROP ROLE engineer
```

References

See the CREATE ROLE, GRANT, REVOKE, and SET ROLE statements in this guide.

See the GRANT and REVOKE statements in *Informix Guide to SQL: Syntax*.

DROP TABLE

The behavior of the DROP TABLE statement is changed in this release.

Effect of CASCADE and RESTRICT Keywords

The behavior of the CASCADE and RESTRICT keywords has changed.

If you drop a base table in the cascade mode, any associated violations and diagnostics tables are also dropped. For example, if the table named **customer** has a violations table named **customer_vio** and a diagnostics table named **customer_dia**, the following statement causes the **customer**, **customer_vio**, and **customer_dia** tables to be dropped:

```
DROP TABLE customer CASCADE
```

If you drop a base table in the restricted mode, and a violations and diagnostics table exist for the base table, your DROP TABLE statement fails. For example, if the table named **items** has a violations table named **items_vio** and a diagnostics table named **items_dia**, the following statement fails:

```
DROP TABLE items RESTRICT
```

By default, the DROP TABLE statement executes in cascade mode. So a DROP TABLE statement without the CASCADE or RESTRICT keywords causes the violations and diagnostics tables (if any) to be dropped with the specified base table. For example, if the table named **orders** has a violations table named **orders_vio** and a diagnostics table named **orders_dia**, the following statement causes all three tables to be dropped:

```
DROP TABLE orders
```

Restriction on Dropping Violations and Diagnostics Tables

You cannot drop a table that is serving as a violations or diagnostics table for a base table. Before you can drop a violations or diagnostics table, you must issue a STOP VIOLATIONS TABLE statement on the base table. Because the STOP VIOLATIONS TABLE statement drops the association between the base table and its violations and diagnostics tables, the former violations and diagnostics tables no longer function as violations and diagnostics tables. You can now use the DROP TABLE statement to drop the former violations and diagnostics tables.

For example, if the table named **customer** has a violations table named **customer_vio** and a diagnostics table named **customer_dia**, the following DROP TABLE statements on the violations and diagnostics tables fails:

```
DROP TABLE customer_vio;
DROP TABLE customer_dia;
```

However, if you first issue a STOP VIOLATIONS TABLE statement on the **customer** table, you are able to drop its former violations and diagnostics tables successfully. The following example shows the correct sequence of steps for dropping the violations and diagnostics tables for the **customer** table:

```
STOP VIOLATIONS TABLE FOR customer;
DROP TABLE customer_vio;
DROP TABLE customer_dia;
```

GET DIAGNOSTICS

The EXCEPTION clause of the GET DIAGNOSTICS statement retrieves the values of SQLSTATE codes from the SQLSTATE variable. A new SQLSTATE code has been added in this release, and GET DIAGNOSTICS statements with the EXCEPTION clause can retrieve this code. The new SQLSTATE code is as follows:

```
01007 Privilege not granted
```

This new warning code is returned to the SQLSTATE variable when a GRANT statement with the ALL keyword executes successfully but does not grant all seven table-level privileges to the grantee.

For example, assume that the user **ted** has the Select and Insert privileges on the **customer** table, together with the authority to grant those privileges to other users. User **ted** wants to grant all seven table-level privileges to a user named **tania**. User **ted** issues the following GRANT statement:

```
GRANT ALL ON customer TO tania
```

This statement executes successfully but returns SQLSTATE code 01007 to the SQLSTATE variable because only the Select and Insert privileges were granted to user **tania**. This statement did not grant the other five table-level privileges implied by the ALL keyword (the Delete, Update, References, Index, and Alter privileges) to user **tania**.

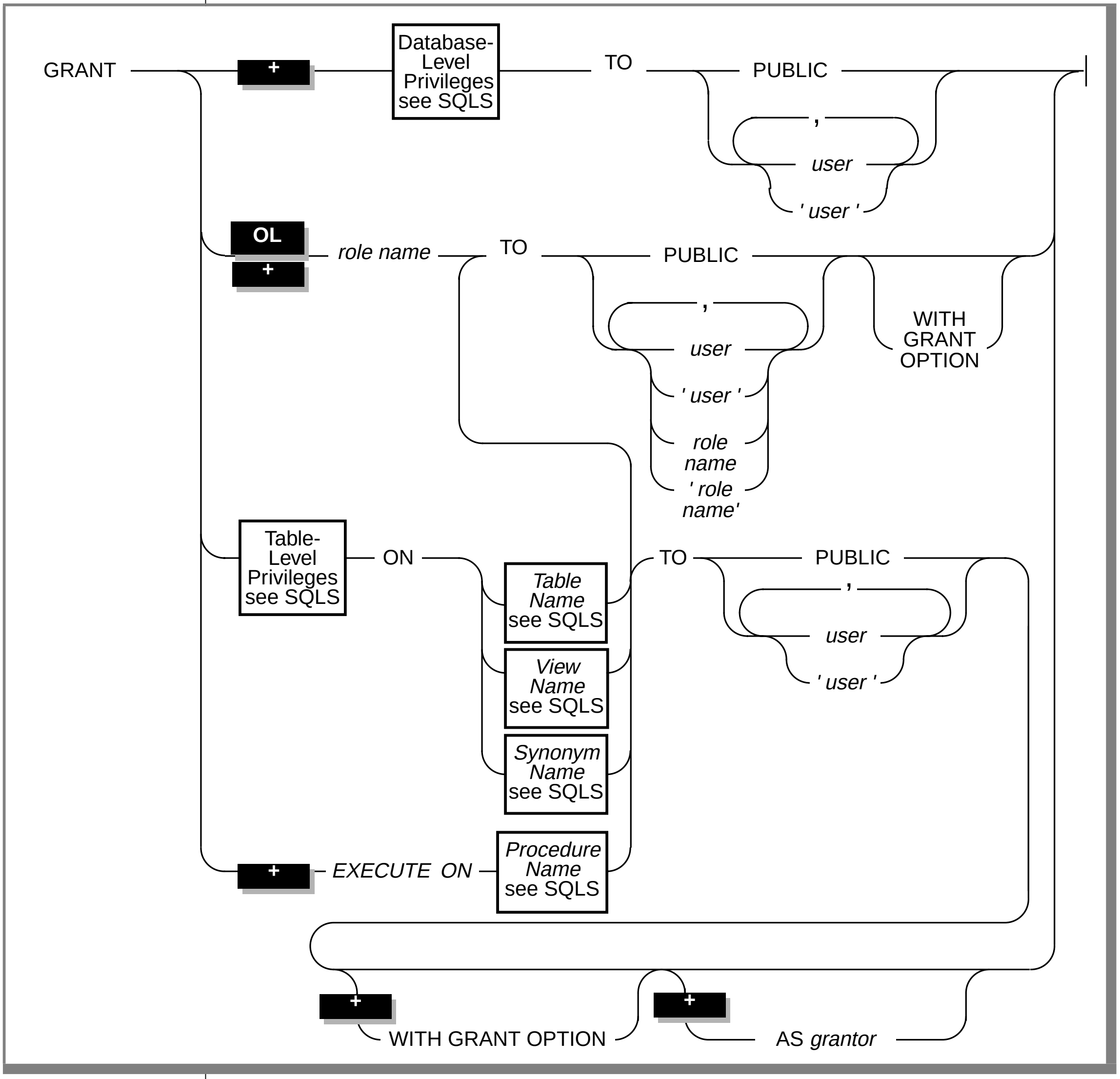
GRANT

The GRANT statement contains a number of changes in this release.

Use the GRANT statement to grant privileges for a table, database, procedure, or a role. The GRANT statement is fully described in the *Informix Guide to SQL: Syntax*.

If you are granting the Alter table-level privilege with the intent of allowing a user to make changes to a table, you must also grant the Resource database privilege for the database in which the table resides.

Syntax


GRANT
+
Database-Level Privileges see SQLS
TO
PUBLIC
,
user
' user '
OL
+
role name
TO
PUBLIC
,
user
' user '
role name
' role name'
WITH GRANT OPTION
Table-Level Privileges see SQLS
ON
Table Name see SQLS
View Name see SQLS
Synonym Name see SQLS
TO
PUBLIC
,
user
' user '
+
EXECUTE ON
Procedure Name see SQLS
+
WITH GRANT OPTION
+
AS grantor

Element	Purpose	Restrictions	Syntax
grantor	The name of the person who is to be listed as the source of the specified privilege in the **systabauth** or **sysroleauth** system catalog table. The person who issues the GRANT statement is the default grantor of the privilege.	If you specify someone else as the grantor of the specified privilege to *user*, you cannot later revoke that privilege from *user*.	Identifier segment, *Informix Guide to SQL: Syntax*
role name	The name of the role granted.	The role must have been created with the CREATE ROLE statement.	CREATE ROLE, p. 3-26 Identifier segment, *Informix Guide to SQL: Syntax*
user	The name of the user who receives the specified privilege.	If you use quotes around *user*, the name of the user is stored exactly as you typed it. In an ANSI-compliant database, the name of the user is stored as uppercase letters if you do not use quotes around *user*. If you grant a privilege to PUBLIC, you do not need to grant the privilege to individual users because PUBLIC extends the privilege to all authorized users. Also see “Restricting Privileges at the Table Level” in the GRANT statement in the *Informix Guide to SQL: Syntax*.	Identifier segment, *Informix Guide to SQL: Syntax*

Granting Privileges to Roles

The GRANT statement is extended to grant a role to another role or user. You can only grant roles that have been created with the CREATE ROLE statement. After a role is granted, you must use the SET ROLE statement to enable the role. Users who have been granted a role with the WITH GRANT OPTION can grant that role to other users or roles. Roles granted to users remain granted until cancelled by a REVOKE statement.

Table-level privileges and the Execute privilege to stored procedures can be granted to roles. Database-level privileges cannot be granted to roles.

The DBA or a user granted the role with the WITH GRANT OPTION can grant a role to a user. A role cannot be granted to itself, either directly or indirectly. The following statement causes an error:

```
GRANT engineer TO engineer
```

When you grant a role to a role, both roles must have been created with the CREATE ROLE statement and both roles must have the same set of privileges. The following example generates an error because the **engineer** role only has the Select privilege, but the **acct** role has the Insert privilege:

```
GRANT engineer TO acct
GRANT acct TO mfg
GRANT engineer TO mfg
```

The following example grants the user **maryf** the role **engineer**:

```
GRANT engineer TO maryf
```

The following example grants the role **acct** to the role **engineer**:

```
GRANT acct TO engineer
```

The following example grants the table-level privilege Insert on **table1** to the role **engineer**:

```
GRANT INSERT ON table1 TO engineer
```

The following example grants the user **maryf** the role **engineer** with the WITH GRANT OPTION. This privilege allows **maryf** to grant the role to other users or roles.

```
GRANT engineer TO maryf WITH GRANT OPTION
```

ALL Keyword in Table-Level Privileges

The behavior of the ALL keyword has changed. Formerly a GRANT statement with the ALL keyword failed if any of the seven table-level privileges did not exist for the grantor. Now, if any or all of the seven table-level privileges do not exist for the grantor, the GRANT statement with the ALL keyword succeeds, but the following SQLSTATE warning is returned:

```
01007 - Privilege not granted.
```

For example, assume that the user **ted** has the Select and Insert privileges on the **customer** table with the authority to grant those privileges to other users. User **ted** wants to grant all seven table-level privileges to user **tania**. So user **ted** issues the following GRANT statement:

```
GRANT ALL ON customer TO tania
```

This statement executes successfully but returns SQLSTATE code 01007. Why is the SQLSTATE warning returned if the statement is successful?

- The statement succeeds in granting the Select and Insert privileges to user **tania** because user **ted** has those privileges and the right to grant those privileges to other users.
- SQLSTATE code 01007 is returned because the other five privileges implied by the ALL keyword (the Delete, Update, References, Index, and Alter privileges) were not grantable by user **ted** and, therefore, were not granted to user **tania**.

ALTER Keyword in Table-Level Privileges

The meaning of the ALTER keyword has changed in this release. The Alter privilege corresponding to this keyword now includes the right to change the object mode of database objects. If a user has the Alter privilege, the user can set the object mode of unique indexes and constraints to the enabled, disabled, or filtering mode. In addition, a user with this privilege can set the object mode of nonunique indexes and triggers to the enabled or disabled mode.

We can state the complete behavior of the Alter privilege as follows:

ALTER	The Alter privilege provides the ability to add or delete columns, modify column data types, or add or delete constraints. *This privilege also provides the ability to set the object mode of unique indexes and constraints to the enabled, disabled, or filtering mode. In addition, this privilege provides the ability to set the object mode of nonunique indexes and triggers to the enabled or disabled modes.*

The second and third sentences in this description (highlighted by italics) describe the new behavior of the ALTER keyword in this release. The first sentence in this description describes behavior that is unchanged in this release.

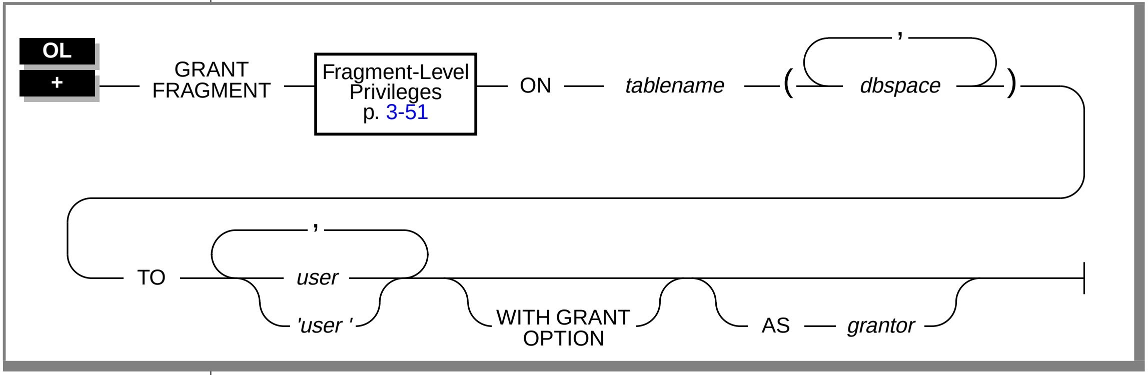
References

See the CREATE ROLE, DROP ROLE, REVOKE, and SET ROLE statements in this guide and the GRANT and REVOKE statements in the *Informix Guide to SQL: Syntax*. See the discussion of privileges and security in the *Informix Guide to SQL: Tutorial*.

GRANT FRAGMENT

The GRANT FRAGMENT statement enables you to grant Insert, Update, and Delete privileges on individual fragments of a fragmented table.

Syntax



Element	Purpose	Restrictions	Syntax
dbspace	The name of the dbspace where the fragment is stored. Use this parameter to specify the fragment or fragments on which privileges are to be granted. There is no default value.	You must specify at least one dbspace. The specified dbspaces must exist.	Identifier segment,*Informix Guide to SQL: Syntax*
grantor	The name of the user who is to be listed as the grantor of the specified privileges in the **grantor** column of the **sysfragauth** system catalog table. The user who issues the GRANT FRAGMENT statement is the default grantor of the privileges.	The user specified in *grantor* must be a valid user.	Identifier segment, *Informix Guide to SQL: Syntax*
tablename	The name of the table that contains the fragment or fragments on which privileges are to be granted. There is no default value.	The specified table must exist and must be fragmented by expression.	Table Name segment, *Informix Guide to SQL: Syntax*
user	The name of the user or users to whom the specified privileges are to be granted. There is no default value.	If you use quotes around *user*, the name of the user is stored exactly as you typed it. In an ANSI-compliant database, the name of the user is stored as uppercase letters if you do not use quotes around *user*.	Identifier segment, *Informix Guide to SQL: Syntax*

Usage

Use the GRANT FRAGMENT statement to grant the Insert, Update, or Delete privilege on one or more fragments of a fragmented table to one or more users.

The GRANT FRAGMENT statement is valid only for tables fragmented according to an expression-based distribution scheme. For an explanation of expression-based distribution schemes, see the ALTER FRAGMENT statement in the *Informix Guide to SQL: Syntax*.

Relationship Between the GRANT and GRANT FRAGMENT Statements

The GRANT FRAGMENT statement is similar to the GRANT statement. Both statements grant privileges to users. The difference between the two statements is that you use GRANT to grant privileges on a table while you use GRANT FRAGMENT to grant privileges on one or more fragments of a table.

Granting Privileges on One Fragment or a List of Fragments

You can grant fragment-level privileges on one fragment of a table or on a list of fragments.

Granting Privileges on One Fragment

The following statement grants the Insert, Update, and Delete privileges on the fragment of the **customer** table in **dbsp1** to the user **larry**:

```
GRANT FRAGMENT ALL ON customer (dbsp1) TO larry
```

Granting Privileges on More Than One Fragment

The following statement grants the Insert, Update, and Delete privileges on the fragments of the **customer** table in **dbsp1** and **dbsp2** to the user **millie**:

```
GRANT FRAGMENT ALL ON customer (dbsp1, dbsp2) TO millie
```

Granting Privileges on All Fragments of a Table

If you want to grant privileges on all fragments of a table to the same user or users, you can use the GRANT statement instead of the GRANT FRAGMENT statement. However, you can also use the GRANT FRAGMENT statement for this purpose.

Assume that the **customer** table is fragmented by expression into three fragments, and these fragments reside in the dbspaces named **dbsp1**, **dbsp2**, and **dbsp3**. You can use either of the following statements to grant the Insert privilege on all fragments of the table to the user **helen**:

```
GRANT FRAGMENT INSERT ON customer (dbsp1, dbsp2, dbsp3)
TO helen;

GRANT INSERT ON customer TO helen;
```

Granting Privileges to One User or a List of Users

You can grant fragment-level privileges to a single user or to a list of users.

Granting Privileges to One User

The following statement grants the Insert, Update, and Delete privileges on the fragment of the **customer** table in **dbsp3** to the user **oswald**:

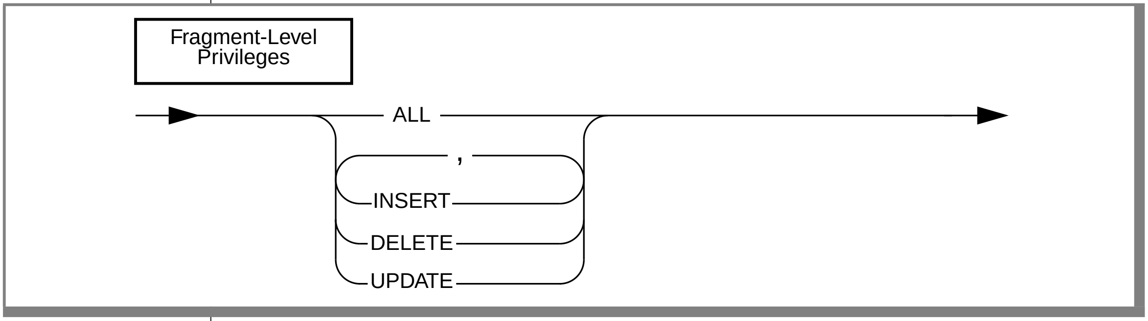
```
GRANT FRAGMENT ALL ON customer (dbsp3) TO oswald
```

Granting Privileges to a List of Users

The following statement grants the Insert, Update, and Delete privileges on the fragment of the **customer** table in **dbsp3** to the users **jerome** and **hilda**:

```
GRANT FRAGMENT ALL ON customer (dbsp3) TO jerome, hilda
```

Fragment-Level Privileges



The following list defines each of the fragment-level privileges:

ALL	Grants Insert, Update, and Delete privileges on a table fragment.
INSERT	Grants Insert privilege on a table fragment. This privilege gives the user the ability to insert rows in the fragment.
DELETE	Grants Delete privilege on a table fragment. This privilege gives the user the ability to delete rows in the fragment.
UPDATE	Grants Update privilege on a table fragment. This privilege gives the user the ability to update rows in the fragment and to name any column of the table in an UPDATE statement.

Definition of Fragment-Level Authority

When a fragmented table is created in an ANSI-compliant database, the table owner implicitly receives all table-level privileges on the new table, but no other users receive privileges.

When a fragmented table is created in a database that is not ANSI-compliant, the table owner implicitly receives all table-level privileges on the new table, and other users (that is, PUBLIC) receive the following default set of privileges on the table: Select, Update, Insert, Delete, and Index. The privileges granted to PUBLIC are explicitly recorded in the **systabauth** system catalog table.

A user who has table privileges on a fragmented table has the privileges implicitly on all fragments of the table. These privileges are not recorded in the **sysfragauth** system catalog table.

Whether or not the database is ANSI-compliant, you can use the GRANT FRAGMENT statement to grant explicit Insert, Update, and Delete privileges on one or more fragments of a table that is fragmented by expression. The privileges granted by the GRANT FRAGMENT statement are explicitly recorded in the **sysfragauth** system catalog table.

The Insert, Update, and Delete privileges that are conferred on table fragments by the GRANT FRAGMENT statement are collectively known as fragment-level privileges or fragment-level authority.

Role of Fragment-Level Authority in Command Validation

Fragment-level authority lets users execute INSERT, DELETE, and UPDATE statements on table fragments even if they lack Insert, Update, and Delete privileges on the table as a whole. Users who lack privileges at the table level can insert, delete, and update rows in authorized fragments because of the algorithm by which OnLine validates commands. This algorithm consists of the following checks:

1. When a user executes an INSERT, DELETE, or UPDATE statement, the database server first checks whether the user has the table authority necessary for the operation attempted. If the table authority exists, the command continues processing.
2. If the table authority does not exist, the database server checks whether the table is fragmented by expression. If the table is not fragmented by expression, the database server returns an error to the user. This error indicates that the user does not have the privilege to execute the command.
3. If the table is fragmented by expression, the database server checks whether the user has the fragment authority necessary for the operation attempted. If the fragment authority exists, the command continues processing. If the fragment authority does not exist, the database server returns an error to the user. This error indicates that the user does not have the privilege to execute the command.

Duration of Fragment-Level Authority

The duration of fragment-level authority is tied to the duration of the fragmentation strategy for the table as a whole.

If you drop a fragmentation strategy by means of a DROP TABLE statement or by means of the INIT, DROP, or DETACH clauses of an ALTER FRAGMENT statement, you also drop any authorities that exist for the affected fragments. Similarly, if you drop a dbspace, you also drop any authorities that exist for the fragment that resides in that dbspace.

Tables that are created as a result of a DETACH or INIT clause of an ALTER FRAGMENT statement do not keep the authorities that the former fragment or fragments had when they were part of the fragmented table. Instead such tables assume the default table authorities.

If a table with fragment authorities defined on it is changed to a table with a round-robin strategy or some other expression strategy, the fragment authorities are also dropped, and the table assumes the default table authorities.

Granting One Privilege, a List of Privileges, or All Privileges

When you specify fragment-level privileges in a GRANT FRAGMENT statement, you can specify one privilege, a list of privileges, or all privileges.

Granting One Privilege

The following statement grants the Update privilege on the fragment of the **customer** table in **dbsp1** to the user **ed**:

```
GRANT FRAGMENT UPDATE ON customer (dbsp1) TO ed
```

Granting a List of Privileges

The following statement grants the Update and Insert privileges on the fragment of the **customer** table in **dbsp1** to the user **susan**:

```
GRANT FRAGMENT UPDATE, INSERT ON customer (dbsp1) TO susan
```

Granting All Privileges

The following statement grants the Insert, Update, and Delete privileges on the fragment of the **customer** table in **dbsp1** to the user **harry**:

```
GRANT FRAGMENT ALL ON customer (dbsp1) TO harry
```

WITH GRANT OPTION Clause

By including the WITH GRANT OPTION clause in the GRANT FRAGMENT statement, you convey the specified fragment-level privileges to a user and the right to grant those same privileges to other users.

The following statement grants the Update privilege on the fragment of the **customer** table in **dbsp3** to the user **george**, and gives this user the right to grant the Update privilege on the same fragment to other users:

```
GRANT FRAGMENT UPDATE ON customer (dbsp3) TO george
    WITH GRANT OPTION
```

AS grantor Clause

The AS *grantor* clause is optional in a GRANT FRAGMENT statement. Use this clause to specify the grantor of the privilege.

Including the AS grantor Clause

When you include the AS *grantor* clause in the GRANT FRAGMENT statement, you specify that the user named in the *grantor* parameter is listed as the grantor of the privilege in the **grantor** column of the **sysfragauth** system catalog table.

In the following example, the user **brenda** grants the Delete privilege on the fragment of the **customer** table in **dbsp3** to the user **martha**. In her GRANT FRAGMENT statement, the user **brenda** uses the AS *grantor* clause to specify that the user **jack** is listed as the grantor of the privilege in the **sysfragauth** system catalog table:

```
GRANT FRAGMENT DELETE ON customer (dbsp3) TO martha AS jack
```

Omitting the AS grantor Clause

When a GRANT FRAGMENT statement does not include the AS *grantor* clause, the user who issues the statement is the default grantor of the privileges specified in the statement.

In the following example, the user **brenda** grants the Update privilege on the fragment of the **customer** table in **dbsp3** to the user **fred**. Because this statement does not specify the AS *grantor* clause, the user **brenda** is listed by default as the grantor of the privilege in the **sysfragauth** system catalog table:

```
GRANT FRAGMENT UPDATE ON customer (dbsp3) TO fred
```

Consequences of the AS grantor Clause

If you omit the AS *grantor* clause, or if you specify your own user name in the *grantor* parameter, you can later revoke the privilege that you granted to the specified user. However, if you specify someone other than yourself as the grantor of the specified privilege to the specified user, you cannot later revoke that privilege from that user.

References

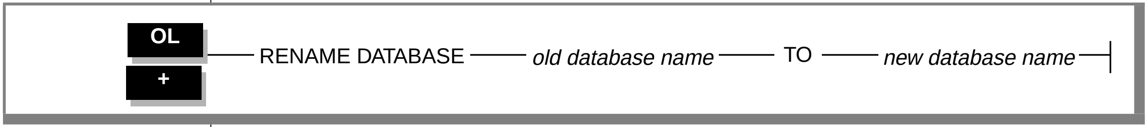
See the GRANT and REVOKE FRAGMENT statements in this guide.

See the GRANT statement in the *Informix Guide to SQL: Syntax*.

RENAME DATABASE

Use the RENAME DATABASE statement to change the name of a database.

Syntax



Element	Purpose	Restrictions	Syntax
new database name	The new name that you want to assign to the database	Name must be unique. You cannot rename the current database. The database to be renamed must not be opened by any users when the RENAME DATABASE command is issued.	Database Name segment, Identifier segment, *Informix Guide to SQL: Syntax*
old database name	The name of the database you want to rename	The database name must exist.	Database Name segment, Identifier segment, *Informix Guide to SQL: Syntax*

Usage

You can rename a database if *either* of the following statements is true:

- You created the database.
- You have the DBA privilege on the database.

You can only rename local databases. You can rename a local database from inside a stored procedure.

References

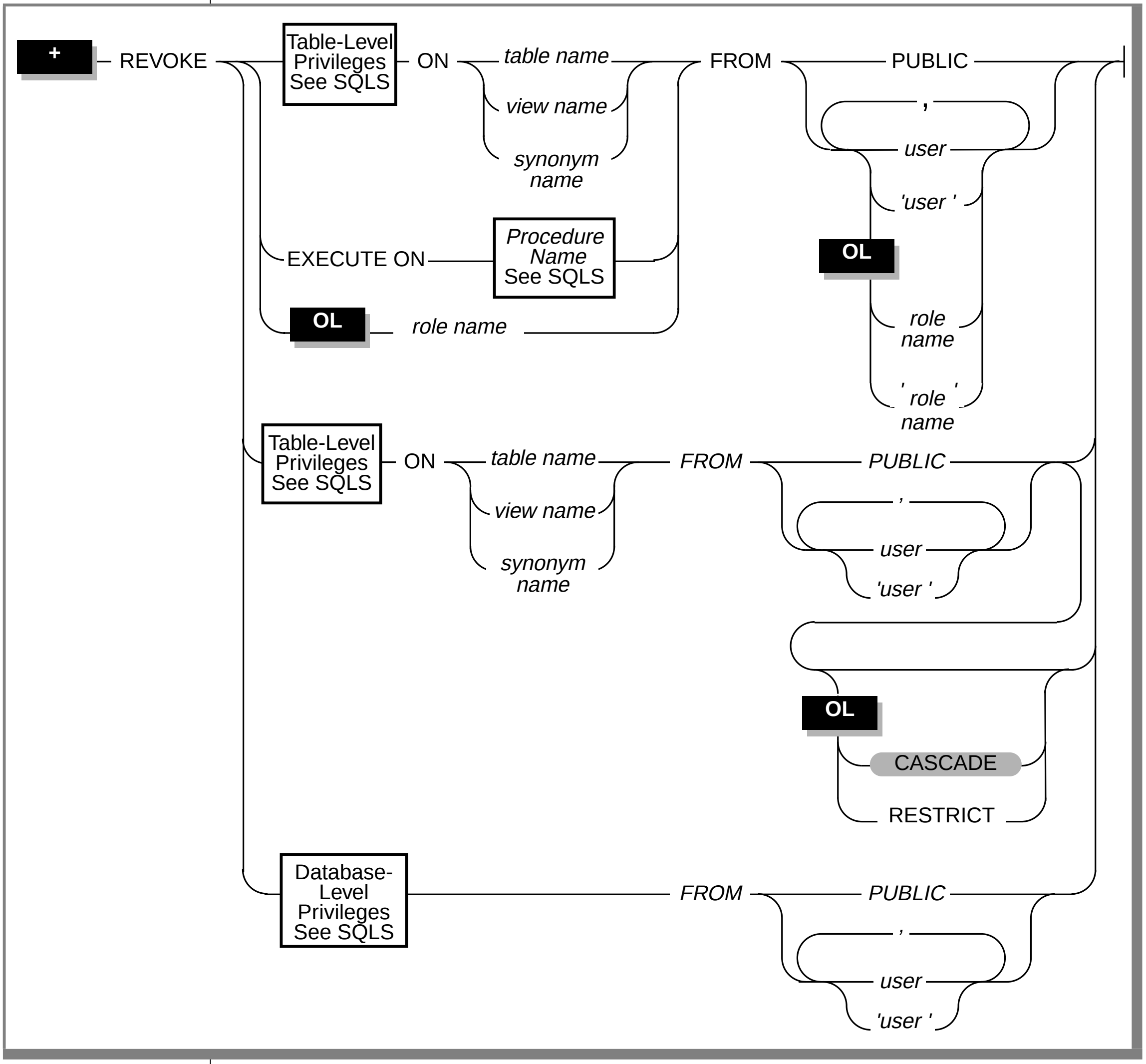
See the CREATE DATABASE statement in the *Informix Guide to SQL: Syntax*.

REVOKE

The REVOKE statement is changed in this release. The entire REVOKE statement is presented here.

Use the REVOKE statement to remove another user's privileges for a table, database, procedure, or role. You can also use the REVOKE statement to revoke a role from another user.

Syntax


+
REVOKE
Table-Level Privileges See SQLS
ON
table name
view name
synonym name
FROM
PUBLIC
,
user
'user '
OL
role name
' role ' name
EXECUTE ON
Procedure Name See SQLS
OL
role name
Table-Level Privileges See SQLS
ON
table name
view name
synonym name
FROM
PUBLIC
,
user
'user '
OL
CASCADE
RESTRICT
Database-Level Privileges See SQLS
FROM
PUBLIC
,
user
'user '

Element	Purpose	Restriction	Syntax
role name	Names the role whose privilege is being revoked or names the role being revoked from a user or role.	The role must have been created with the CREATE ROLE statement and granted with the GRANT statement.	CREATE ROLE, p. 3-26 Identifier segment, *Informix Guide to SQL: Syntax*
synonym name	The synonym name being revoked	Must be an existing synonym name.	Synonym Name segment, *Informix Guide to SQL: Syntax*
table name	The table name being revoked	Must be an existing table name.	Table Name segment, *Informix Guide to SQL: Syntax*
user	Names the user or role whose privileges are revoked	Must be a valid user or role.	Identifier segment, *Informix Guide to SQL: Syntax*
view name	The view name being revoked	Must be an existing view name.	View Name segment, *Informix Guide to SQL: Syntax*

Usage

You can use the REVOKE statement with the GRANT statement to finely control the ability of users to modify the database as well as access and modify data in the tables.

If you use the PUBLIC keyword after the FROM keyword, the REVOKE statement revokes privileges from all users.

You can revoke all or some of the privileges that you granted to other users. No one can revoke privileges granted by another user.

If you revoke the EXECUTE privilege on a stored procedure from a user, that user can no longer run that procedure using either the EXECUTE PROCEDURE or CALL statements.

If you use quotes, *user* appears exactly as typed.

Users cannot revoke privileges from themselves.

Using the REVOKE Statement with Roles

You can use the REVOKE statement to remove privileges from a role and remove a role from a user or another role. Once a role is revoked from a user, the user cannot enable that role. You can revoke all or some of the roles granted to a user or role. If a role is revoked from a user who granted the role to other users, the role is also revoked from the subsequent users.

You can use the REVOKE statement to revoke table-level privileges from a role; however, you cannot use the RESTRICT clause when you do so.

Only the DBA or a user granted a role with the WITH GRANT OPTION can revoke privileges for a role.

If you revoke the Execute privilege on a stored procedure from a role, that role can no longer run that procedure.

Users cannot revoke roles from themselves. When you revoke a role, you cannot revoke the WITH GRANT OPTION separately. If the role was granted with the WITH GRANT OPTION, both the role and grant option are revoked.

The following example revokes the **engineer** role from the user **maryf**:

```
REVOKE engineer FROM maryf
```

Revoking Privileges Granted from WITH GRANT OPTION

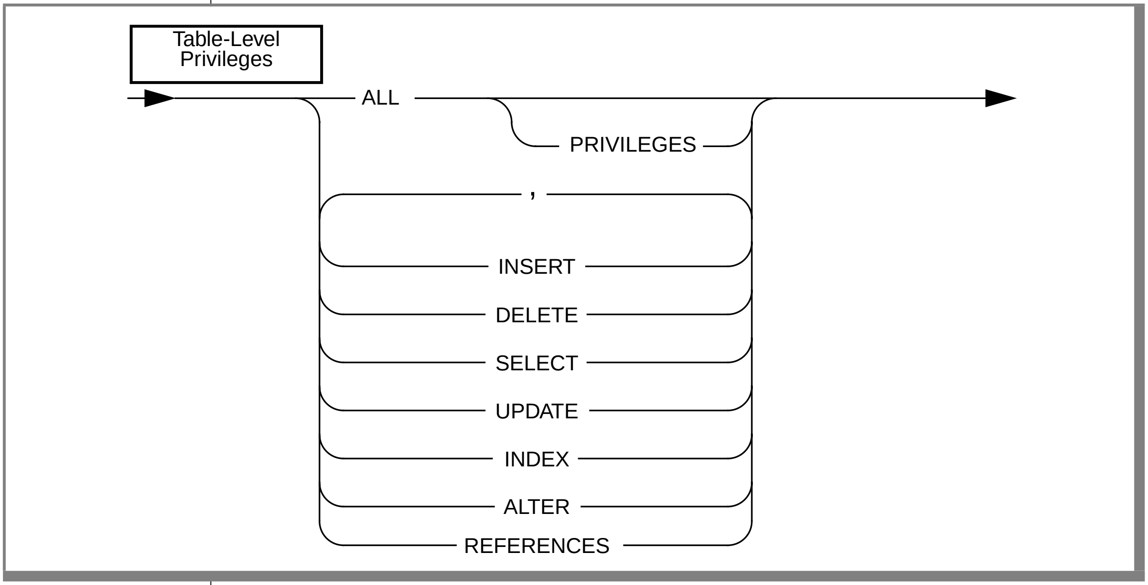
If you revoke from *user* the privileges that you granted using the WITH GRANT OPTION keywords, you sever the chain of privileges granted by that *user*. In this case, when you revoke privileges from *user*, you automatically revoke the privileges of all users who received privileges from *user* or from the chain that *user* created. You can also specify this default condition with the CASCADE keyword.

REVOKE and ROLLBACK WORK

SE

You cannot use a ROLLBACK WORK statement to undo a REVOKE statement that successfully executes. If you roll back a transaction that contains a REVOKE statement, the privilege is not granted again to the user and you do not receive an error message. ♦

Table-Level Privileges



To revoke a table-level privilege from a user, you must revoke all occurrences of the privilege. For example, if two users grant the same privilege to a user, both of them must revoke the privilege. If one grantor revokes the privilege, the user retains the privilege received from the other grantor. (The database server keeps a record of each table-level grant in the **syscolauth** and **systabauth** system catalog tables.)

If a table owner grants a privilege to PUBLIC, the owner cannot revoke the same privilege from any particular user. For example, if the table owner grants the Select privilege to PUBLIC and then attempts to revoke the Select privilege from **mary**, the REVOKE statement generates an error. The Select privilege was granted to PUBLIC, not to **mary**, and therefore the privilege cannot be revoked from **mary**. (ISAM error number 111, `No record found`, refers to the lack of a record in either the **syscolauth** or **systabauth** system catalog table, which would represent the grant that the table owner now wants to revoke.)

You can revoke table-level privileges individually or in combination. List the keywords that correspond to the privileges that you are revoking from *user*. The keywords are described in the following list. Unlike the GRANT statement, the REVOKE statement does not allow you to qualify the Select, Update, or References privilege with a column name. Thus you cannot revoke access on specific columns.

INSERT	provides the ability to insert rows.
DELETE	provides the ability to delete rows.
SELECT	provides the ability to display data obtained from a SELECT statement.
UPDATE	provides the ability to change column values.
INDEX	provides the ability to create permanent indexes. You must have the Resource privilege to take advantage of the Index privilege. (Any user with the Connect privilege can create indexes on temporary tables.)
ALTER	provides the ability to modify column data types or to add or delete columns.
REFERENCES	provides the ability to reference columns in referential constraints. You must have the Resource privilege to take advantage of the References privilege. (However, you can add a referential constraint during an ALTER TABLE statement. This method does not require that you have the Resource privilege on the database.) Revoke the References privilege to disallow cascading deletes.
ALL	provides all the preceding privileges. The PRIVILEGES keyword is optional.

The following example revokes the Index and Alter privileges from all users for the **customer** table; these privileges are then granted specifically to user **mary**.

```
REVOKE INDEX, ALTER ON customer FROM PUBLIC;
GRANT INDEX, ALTER ON customer TO mary;
```

Because you cannot revoke access on specific columns, when you revoke the Select, Update, or References privilege from a user, you revoke the privilege for all columns in the table. You must use a GRANT statement to specifically regrant any column-specific privilege that should be available to the user, as shown in the following example:

```
REVOKE ALL ON customer FROM PUBLIC;
GRANT ALL ON customer TO john, cathy;
GRANT SELECT (fname, lname, company, city)
    ON customer TO PUBLIC;
```

ALL Keyword in Table-Level Privileges

The behavior of the ALL keyword has changed. Formerly a REVOKE statement with the ALL keyword returned the following SQLSTATE code if any or all of the seven table-level privileges did not exist for the revokee:

```
00000 - Success
```

Now, if any or all of the seven table-level privileges do not exist for the revokee, the REVOKE statement with the ALL keyword returns the following SQLSTATE code:

```
01006 - Privilege not revoked
```

For example, assume that the user **hal** has the Select and Insert privileges on the **customer** table. User **jocelyn** wants to revoke all seven table-level privileges from user **hal**. So user **jocelyn** issues the following REVOKE statement:

```
REVOKE ALL ON customer FROM hal
```

This statement executes successfully but returns SQLSTATE code 01006.Why is the SQLSTATE warning returned if the statement is successful?

- The statement succeeds in revoking the Select and Insert privileges from user **hal** because user **hal** had those privileges.
- SQLSTATE code 01006 is returned because the other five privileges implied by the ALL keyword (the Delete, Update, References, Index, and Alter privileges) did not exist for user **hal**; therefore, these privileges were not revoked.

ALTER Keyword in Table-Level Privileges

The individual table-level privileges that you can grant with the GRANT statement are exactly the same as the table-level privileges that you can revoke with the REVOKE statement. The table-level privileges in both statements are the Insert, Delete, Select, Update, References, Index, and Alter privileges. When the behavior of one of these privileges changes in the GRANT statement, the behavior of the same privilege changes correspondingly in the REVOKE statement.

In this release the meaning of the Alter privilege in the GRANT statement has changed, so the meaning of the Alter privilege in the REVOKE statement has changed accordingly.

In the GRANT statement, the ALTER keyword that corresponds to the Alter privilege now confers the right to change the object mode of database objects. If a user has the Alter privilege, the user can set the object mode of unique indexes and constraints to the enabled, disabled, or filtering mode. In addition, a user with this privilege can set the object mode of non-unique indexes and triggers to the enabled or disabled modes.

By the same token, the ALTER keyword in the REVOKE statement now takes away the right to change the object mode of database objects. If the Alter privilege is revoked from a user, that user can no longer set the object mode of unique indexes and constraints to the enabled, disabled, or filtering mode. In addition, that user can no longer set the object mode of non-unique indexes and triggers to the enabled or disabled modes.

We can state the complete behavior of the ALTER keyword in the REVOKE statement as follows:

ALTER	This keyword causes the Alter privilege to be revoked. The Alter privilege provides the ability to add or delete columns, modify column data types, or add or delete constraints. *This privilege also provides the ability to set the object mode of unique indexes and constraints to the enabled, disabled, or filtering mode. In addition, this privilege provides the ability to set the object mode of non-unique indexes and triggers to the enabled or disabled modes.*

The italicized text in the preceding description highlights the new behavior of the ALTER keyword in this release.

RESTRICT Clause

The behavior of the RESTRICT clause has changed. Formerly a REVOKE statement with the RESTRICT keyword failed if the revokee had the right to grant this privilege to other users, whether or not the revokee had actually granted the privilege to any other user. Now a REVOKE statement with the RESTRICT keyword succeeds if the revokee has the right to grant this privilege to other users but has not actually granted this privilege to any other user.

We can illustrate the new behavior of the RESTRICT keyword by means of the following examples.

Assume that the user **clara** has granted the Select privilege on the **customer** table to the user **ted**, and she has also granted user **ted** the right to grant the Select privilege to other users. User **ted** has used this authority to grant the Select privilege on the **customer** table to the user named **tania**. Now user **clara** attempts to revoke the Select privilege from user **ted** with the following REVOKE statement:

```
REVOKE SELECT ON customer FROM ted RESTRICT
```

This statement fails because user **ted** has granted the Select privilege to user **tania**.

What if the revokee has the right to grant the privilege to other users but has not actually granted this privilege to any other user? For example, assume that the user **clara** has granted the Select privilege on the **customer** table to the user **roger**, and she has also granted user **roger** the right to grant the Select privilege to other users. However, user **roger** has not used this authority to granted the Select privilege to any other user. Now user **clara** attempts to revoke the Select privilege from user **roger** with the following REVOKE statement:

```
REVOKE SELECT ON customer FROM roger RESTRICT
```

This statement succeeds because user **roger** has not granted the Select privilege to any other user.

We can summarize the effects of the RESTRICT clause in the current release by means of a simple list of conditions. A REVOKE statement that includes the RESTRICT clause fails if any of the following conditions are true:

- *The user from whom the privilege is to be revoked has granted this privilege to another user or users.*
- A view depends on a Select privilege that is being revoked.
- A foreign-key constraint depends on a References privilege that is being revoked.

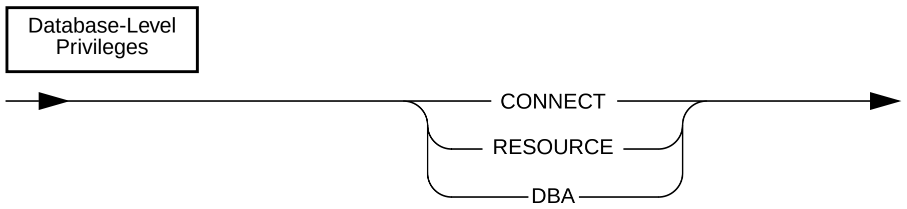
The first condition in this list (in italic type) is the only condition that is new in this release. The other two conditions in the list remain unchanged.

Revoking Table-Level Privileges from a User

ANSI

In an ANSI-compliant database, if you do not use quotes around *user*, the name of the user is stored in uppercase letters. ♦

Database-Level Privileges



Only a user with the DBA privilege can grant or revoke database-level privileges.

Three levels of database privileges control access. These privilege levels are, from lowest to highest, Connect, Resource, and DBA. To revoke a database privilege, specify one of the keywords CONNECT, RESOURCE, or DBA in the REVOKE statement.

Because of the hierarchical organization of the privileges (as outlined in the privilege definitions described later in this section), if you revoke either the Resource or the Connect privilege from a user with the DBA privilege, the statement has no effect. If you revoke the DBA privilege from a user who has the DBA privilege, the user retains the Connect privilege on the database. To deny database access to a user with the DBA or Resource privilege, you must first revoke the DBA or the Resource privilege and then revoke the Connect privilege in a separate REVOKE statement.

Similarly, if you revoke the Connect privilege from a user with the Resource privilege, the statement has no effect. If you revoke the Resource privilege from a user, the user retains the Connect privilege on the database.

The database privileges are associated with the following keywords:

CONNECT gives you the ability to query and modify data. You can modify the database schema if you own the object that you want to modify. Any user with the Connect privilege can perform the following functions:

- Execute SELECT, INSERT, UPDATE, and DELETE statements, provided that the user has the necessary table-level privileges
- Create views, provided that the user has the Select privilege on the underlying tables
- Create synonyms
- Create temporary tables and create indexes on the temporary tables
- Alter or drop a table or an index, provided that the user owns the table or index (or has the Alter, Index, or References privilege on the table)
- Grant privileges on a table, provided that the user owns the table (or has been given privileges on the table with the WITH GRANT OPTION keyword)

RESOURCE gives you the ability to extend the structure of the database. In addition to the capabilities of the Connect privilege, the holder of the Resource privilege can perform the following functions:

- Create new tables
- Create new indexes
- Create new procedures

DBA	allows the holder of DBA privilege to perform the following functions in addition to the capabilities of the Resource privilege:

- Grant any privilege, including the DBA privilege, to another user
- Use the NEXT SIZE keyword to alter extent sizes in the system catalog tables
- Drop any object, regardless of who owns it
- Create tables, views, and indexes as well as specify another user as owner of the objects
- Execute the DROP DATABASE statement
- SE Execute the START DATABASE, and ROLLFORWARD DATABASE statements✥
- Insert, delete, or update rows of any system catalog table except **systables**

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

References

See the CREATE ROLE, DROP ROLE, GRANT, and SET ROLE statements in this guide.

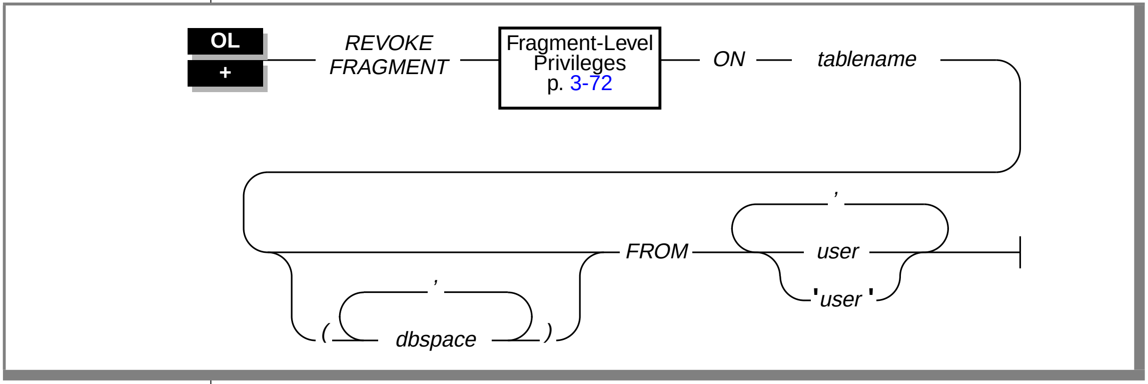
See the GRANT and REVOKE statements in the *Informix Guide to SQL: Syntax*.

See the discussion of privileges and security in the *Informix Guide to SQL: Tutorial*.

REVOKE FRAGMENT

The REVOKE FRAGMENT statement enables you to revoke privileges that have been granted on individual fragments of a fragmented table. You can use this statement to revoke the Insert, Update, and Delete fragment-level privileges from users.

Syntax



Element	Purpose	Restrictions	Syntax
dbspace	The name of the dbspace where the fragment is stored. Use this parameter to specify the fragment or fragments on which privileges are to be revoked. If you do not specify a fragment, the REVOKE statement applies to all fragments in the specified table that have the specified privileges.	The specified dbspace or dbspaces must exist.	Identifier segment,*Informix Guide to SQL: Syntax*
tablename	The name of the table that contains the fragment or fragments on which privileges are to be revoked. There is no default value.	The specified table must exist and must be fragmented by expression.	Table Name segment, *Informix Guide to SQL: Syntax*
user	The name of the user or users from whom the specified privileges are to be revoked. There is no default value.	The user must be a valid user.	Identifier segment, *Informix Guide to SQL: Syntax*

Usage

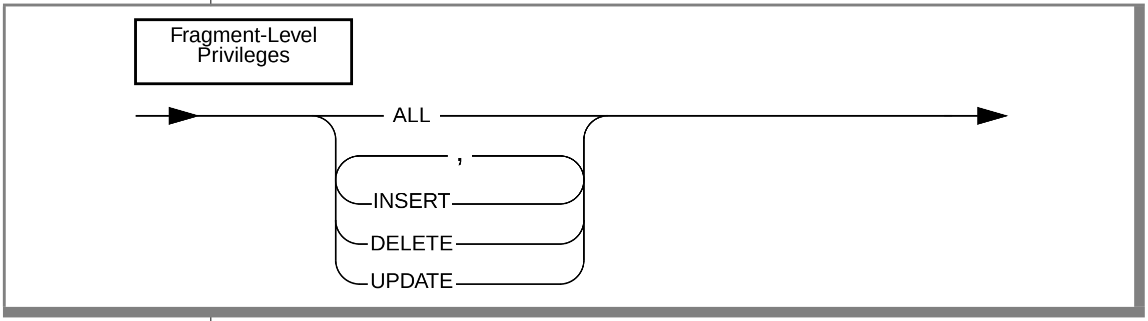
Use the REVOKE FRAGMENT statement to revoke the Insert, Update, or Delete privilege on one or more fragments of a fragmented table from one or more users.

The REVOKE FRAGMENT statement is only valid for tables that are fragmented according to an expression-based distribution scheme. See the ALTER FRAGMENT statement in the *Informix Guide to SQL: Syntax* for an explanation of expression-based distribution schemes.

You can specify one fragment or a list of fragments in the REVOKE FRAGMENT statement. You specify a fragment by naming the dbspace in which the fragment resides.

You do not have to specify a particular fragment or a list of fragments in the REVOKE FRAGMENT statement. If you do not specify any fragments in the statement, the specified users lose the specified privileges on all fragments for which the users currently have those privileges.

Fragment-Level Privileges



You can revoke fragment-level privileges individually or in combination. List the keywords that correspond to the privileges that you are revoking from *user*. The keywords are described in the following list:

ALL	Revokes all privileges currently granted on a table fragment.
INSERT	Revokes Insert privilege on a table fragment. This privilege gives the user the ability to insert rows in the fragment.
DELETE	Revokes Delete privilege on a table fragment. This privilege gives the user the ability to delete rows in the fragment.
UPDATE	Revokes Update privilege on a table fragment. This privilege gives the user the ability to update rows in the fragment and to name any column of the table in an UPDATE statement.

If you specify the ALL keyword in a REVOKE FRAGMENT statement, the specified users lose all fragment-level privileges that they currently have on the specified fragments.

For example, assume that a user currently has the Update privilege on one fragment of a table. If you use the ALL keyword to revoke all current privileges on this fragment from this user, the user loses the Update privilege that he or she had on this fragment.

Examples of the REVOKE FRAGMENT Statement

The examples that follow are based on the **customer** table. All the examples assume that the **customer** table is fragmented by expression into three fragments that reside in the dbspaces named **dbsp1**, **dbsp2**, and **dbsp3**.

Revoking One Privilege

The following statement revokes the Update privilege on the fragment of the **customer** table in **dbsp1** from the user **ed**:

```
REVOKE FRAGMENT UPDATE ON customer (dbsp1) FROM ed
```

Revoking More Than One Privilege

The following statement revokes the Update and Insert privileges on the fragment of the **customer** table in **dbsp1** from the user **susan**:

```
REVOKE FRAGMENT UPDATE, INSERT ON customer (dbsp1) FROM susan
```

Revoking All Privileges

The following statement revokes all privileges currently granted to the user **harry** on the fragment of the **customer** table in **dbsp1**.:

```
REVOKE FRAGMENT ALL ON customer (dbsp1) FROM harry
```

Revoking Privileges on More Than One Fragment

The following statement revokes all privileges currently granted to the user **millie** on the fragments of the **customer** table in **dbsp1** and **dbsp2**:

```
REVOKE FRAGMENT ALL ON customer (dbsp1, dbsp2) FROM millie
```

Revoking Privileges from More Than One User

The following statement revokes all privileges currently granted to the users **jerome** and **hilda** on the fragment of the **customer** table in **dbsp3**:

```
REVOKE FRAGMENT ALL ON customer (dbsp3) FROM jerome, hilda
```

Revoking Privileges Without Specifying Fragments

The following statement revokes all current privileges from the user **mel** on all fragments for which this user currently has privileges:

```
REVOKE FRAGMENT ALL ON customer FROM mel
```

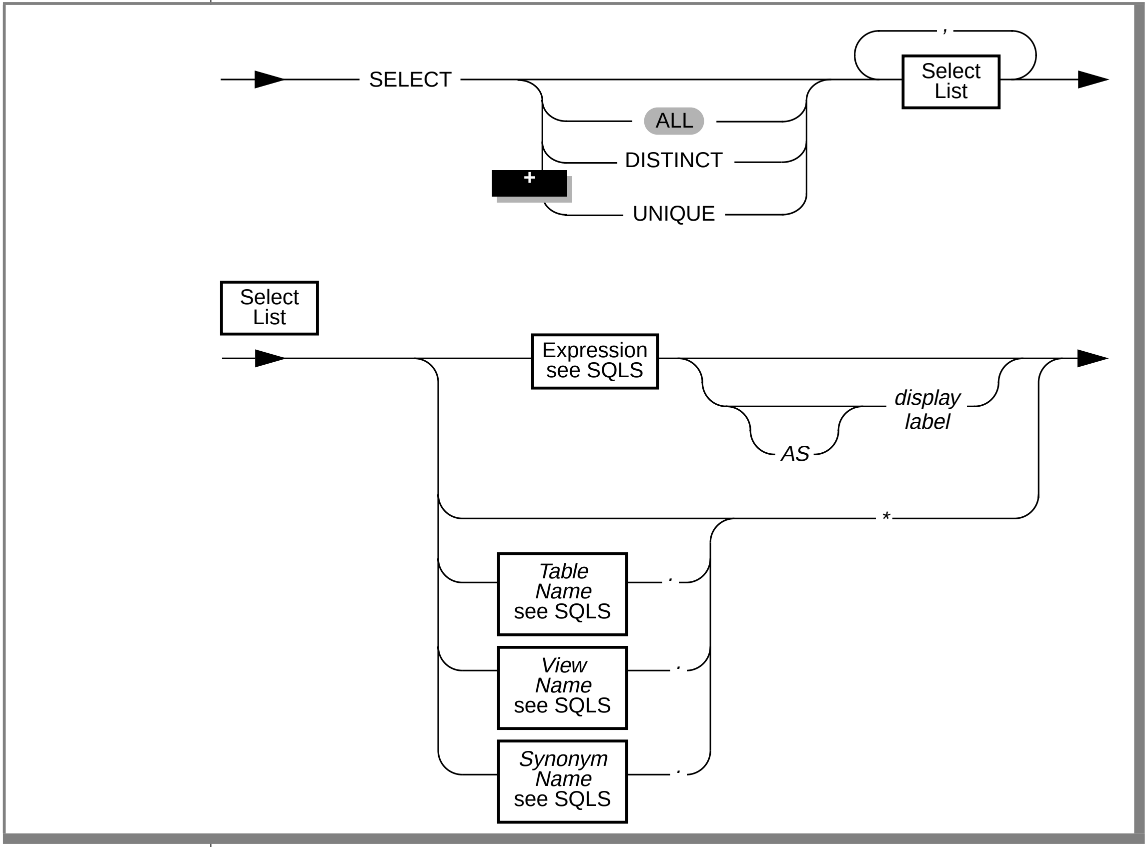
References

See the REVOKE and GRANT FRAGMENT statements in this guide.

SELECT

The SELECT statement is changed in this release. The AS keyword in the SELECT clause is now treated as ANSI-compliant by the ANSI flagger.

SELECT Clause



As a result of modifications to the ANSI flagger used by Informix products, the flagger now treats the optional AS keyword in the SELECT clause as ANSI-compliant.

The following SELECT statement uses the AS keyword with a display label. When you enter this statement in DB-Access or use it in an ESQL/C or ESQL/COBOL program, you no longer receive an ANSI noncompliance warning:

```
    SELECT call_dtime AS time_of_call FROM cust_calls
```

For a complete explanation of the AS keyword and display labels, see the SELECT statement in the *Informix Guide to SQL: Syntax*.

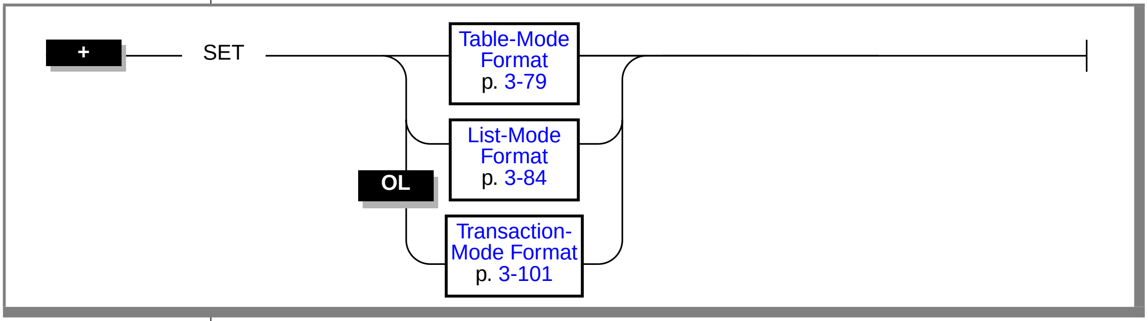
References

See the SELECT statement in the *Informix Guide to SQL: Syntax* for a complete description of SELECT.

SET

The SET statement allows you to change the object mode of the following database objects: constraints, indexes, and triggers. You can also use the SET statement to specify the transaction mode of constraints.

Syntax



Usage

The SET statement has the following purposes:

- To change the object mode of constraints, indexes, and triggers.

  When you change the object mode of constraints, indexes, or triggers, the change is permanent. The setting produced by the SET statement remains in effect until you change the object mode of the object again.

- To set the transaction mode of constraints by specifying whether constraints are checked at the statement level or at the transaction level.

  When you set the transaction mode of constraints, the effect of the SET statement is limited to the transaction in which it is executed. The setting produced by the SET statement is only effective during the transaction. For further information on setting the transaction mode for constraints, see "Transaction-Mode Format" on page 3-101.

Terminology for Object Modes

The SET statement operates on database objects by changing the object mode of those objects. The terms *database objects* and *objects* have a restricted meaning in the context of the SET statement. Both terms refer to the constraints, indexes, and triggers in a database.

Similarly, the term *object modes* has a restricted meaning in the context of the SET statement. The term refers to the three states that a database object can have: enabled, disabled, and filtering. The **sysobjstate** system catalog table lists all of the objects in the database and the current object mode of each object.

Do not confuse the terms *objects* and *object modes* as used in the SET statement with the term *objects* in INFORMIX-NewEra. In the context of INFORMIX-NewEra, *objects* refers to objects within an application.

Methods for Changing Object Modes

The SET statement provides the following formats for changing object modes: table mode and list mode. For an explanation of the table mode format, see "Table-Mode Format" on page 3-79. For an explanation of the list mode format, see "List-Mode Format" on page 3-84.

Privileges Required for Changing Object Modes

To change the object mode of a constraint, index, or trigger, you must have the necessary privileges. Specifically, you must meet one of the following requirements:

- You must have the DBA privilege on the database.
- You must be the owner of the table on which the object is defined and must have the Resource privilege on the database.
- You must have the Alter privilege on the table on which the object is defined and the Resource privilege on the database.

Table-Mode Format

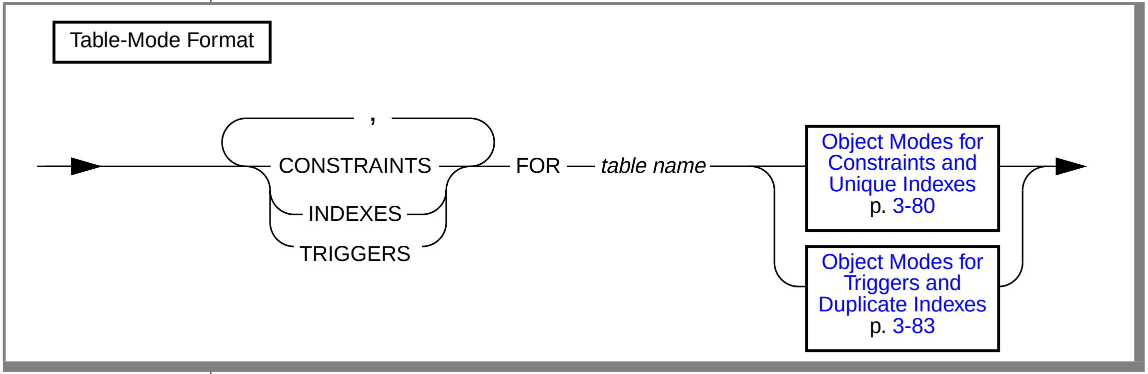


Table-Mode Format
,
CONSTRAINTS
INDEXES
TRIGGERS
FOR
table name
Object Modes for Constraints and Unique Indexes p. 3-80
Object Modes for Triggers and Duplicate Indexes p. 3-83

Element	Purpose	Restrictions	Syntax
table name	The name of the table whose objects will have their object mode changed. There is no default value.	The table must be a local table. You cannot set the object modes of objects defined on a temporary table to the disabled or filtering modes. For information on the privileges required to change the object mode of the objects defined on a table, see "Privileges Required for Changing Object Modes" on page 3-78.	Identifier segment, *Informix Guide to SQL: Syntax*

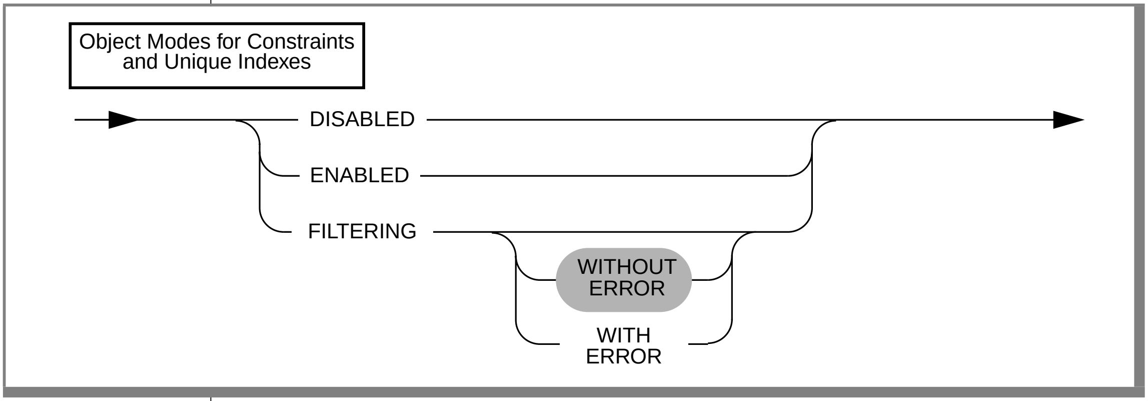
Use the table-mode format to change the object mode of all objects of a given type that have been defined on a particular table. For example, to change the object mode of all constraints defined on the **cust_subset** table to the disabled mode, enter the following statement:

```
SET CONSTRAINTS FOR cust_subset DISABLED
```

By using the table-mode format, you can change the object modes of more than one object type with a single SET statement. For example, to change the object mode of all constraints, indexes, and triggers defined on the **cust_subset** table to the enabled mode, enter the following statement:

```
SET CONSTRAINTS, INDEXES, TRIGGERS FOR cust_subset
    ENABLED
```

Object Modes for Constraints and Unique Indexes



You can specify the disabled, enabled, or filtering object modes for a constraint or a unique index. You must specify one of these object modes in your SET statement. There is no default object mode in the SET statement.

You can also specify the object mode for a constraint when you create the constraint with the ALTER TABLE or CREATE TABLE statements. If you do not specify the object mode for a constraint in one of these statements or in a SET statement, the constraint is in the enabled object mode by default.

You can also specify the object mode for a unique index when you create the index with the CREATE INDEX statement. If you do not specify the object mode for a unique index in the CREATE INDEX statement or in a SET statement, the unique index is in the enabled object mode by default.

For definitions of the disabled, enabled, and filtering object modes see "Using Object Modes with Data Manipulation Statements" on page 3-86. For an explanation of the benefits of these object modes, see "Benefits of Object Modes" on page 3-98.

Error Options for Filtering Mode

When you change the object mode of a constraint or unique index to the filtering mode, you can specify the following error options: WITHOUT ERROR or WITH ERROR.

The WITHOUT ERROR Option

The WITHOUT ERROR option signifies that when the database server executes an INSERT, DELETE, or UPDATE statement and one or more of the target rows causes a constraint violation or unique-index violation, no integrity-violation error message is returned to the user. The WITHOUT ERROR option is the default error option.

The WITH ERROR Option

The WITH ERROR option signifies that when the database server executes an INSERT, DELETE, or UPDATE statement and one or more of the target rows causes a constraint violation or unique-index violation, an integrity-violation error message is returned to the user.

Scope of Error Options

The WITH ERROR and WITHOUT ERROR options only apply when the database server executes an INSERT, DELETE, or UPDATE statement and one or more of the target rows causes a constraint violation or unique index violation. These error options control whether the database server displays an integrity-violation error message after it executes these statements.

These error options do not apply when you attempt to change the object mode of a disabled constraint or disabled unique index to the enabled or filtering mode and the SET statement fails because one or more rows in the target table violates the constraint or the unique-index requirement. In these cases, if a violations table has been started for the table that contains the inconsistent data, the database server returns an integrity-violation error message regardless of the error option specified in the SET statement.

Violations and Diagnostics Tables for Filtering Mode

When you specify the filtering mode for constraints or unique indexes in a SET statement, violations and diagnostics tables are not automatically started for the target table. When you set objects to the filtering mode, be sure to start the violations and diagnostics tables for the target table on which the filtering mode objects are defined. The violations table captures rows that fail to meet integrity requirements. The diagnostics table captures information about each row that fails to meet integrity requirements.

When to Start the Violations and Diagnostics Tables

You are not required to start the violations and diagnostics tables before you set objects to the filtering mode. If you have not started a violations and diagnostics table when you set an object to the filtering mode, the database server executes your SET statement and does not return an error. Similarly, if you issue an INSERT, DELETE, or UPDATE statement on the target table, and you have not started a violations and diagnostics table for the target table, the database server executes the statement and does not return an error as long as all of the integrity requirements on the table are satisfied.

If you have not started a violations and diagnostics table for the target table with filtering mode objects, the database server does not return an error until an INSERT, DELETE, or UPDATE statement fails to satisfy an integrity requirement on the table. If an INSERT, DELETE, or UPDATE statement fails to satisfy the constraint or unique-index requirement for a particular row, the database server cannot filter the bad row to the violations table because no violations table is associated with the target table. The user receives an error message indicating that no violations table has been started for the target table.

To prevent such errors, start the violations and diagnostics tables for the target table at one of the following points:

- You can start the violations and diagnostics tables before you set any objects defined on the table to the filtering mode.
- You can start the violations and diagnostics tables after you set objects to the filtering mode but before any users issue INSERT, DELETE, or UPDATE statements that could violate any integrity requirements on the target table.

How to Start the Violations and Diagnostics Tables

To create the violations and diagnostics tables and associate them with the target table, you use the START VIOLATIONS TABLE statement. In this statement, you specify the name of the target table for which the violations and diagnostics tables are to be started. You can also assign names to the violations and diagnostics tables in this statement.

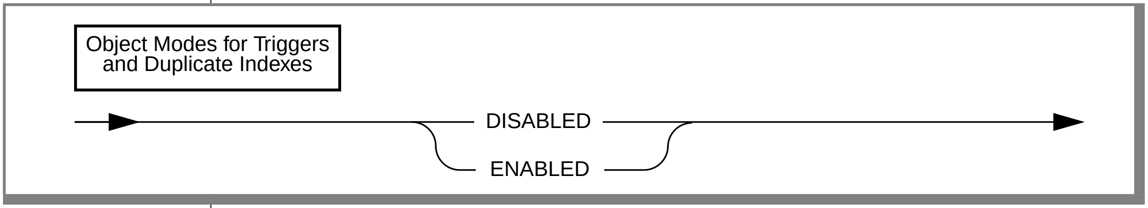
For further information on the START VIOLATIONS TABLE statement and the structure of the violations and diagnostics tables themselves, see "START VIOLATIONS TABLE" on page 3-109.

How to Stop the Violations and Diagnostics Tables

After you turn off filtering mode for the objects defined on a target table and you no longer need the violations and diagnostics tables, you can use the STOP VIOLATIONS TABLE statement to drop the association between the target table and the violations and diagnostics tables. In this statement, you specify the name of the target table whose association with the violations and diagnostics tables is to be dropped.

For further information on the STOP VIOLATIONS TABLE statement, see "STOP VIOLATIONS TABLE" on page 3-128.

Object Modes for Triggers and Duplicate Indexes



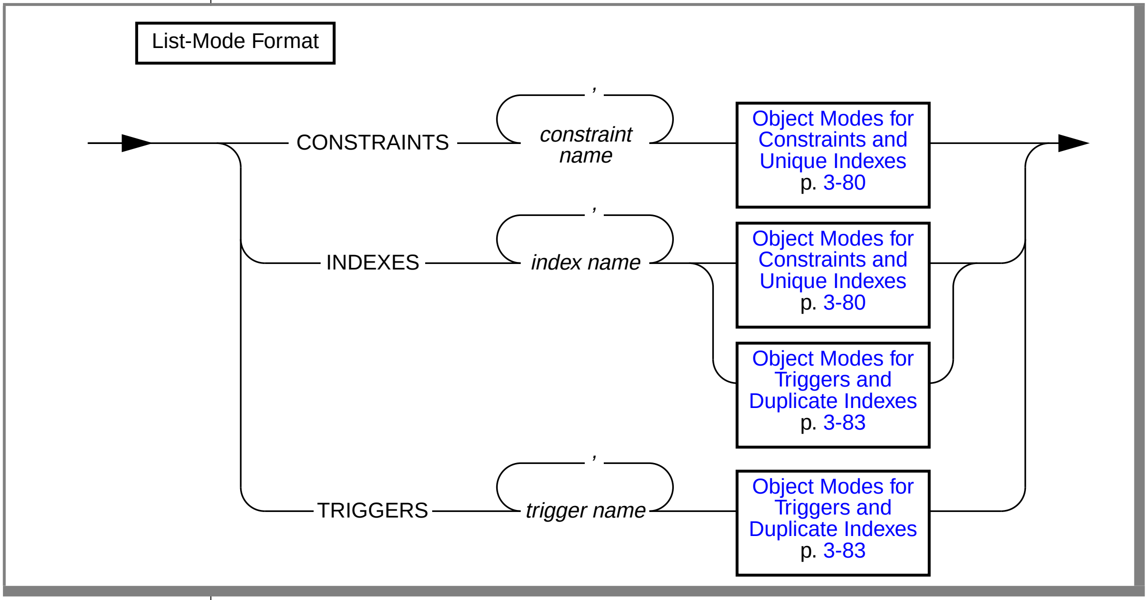
You can specify the disabled or enabled object modes for triggers or duplicate indexes. You must specify one of these object modes in your SET statement. There is no default object mode in the SET statement.

You can also specify the object mode for a trigger when you create the trigger with the CREATE TRIGGER statement. If you do not specify the object mode for a trigger in the CREATE TRIGGER statement or in a SET statement, the trigger is in the enabled object mode by default.

You can also specify the object mode for a duplicate index when you create the index with the CREATE INDEX statement. If you do not specify the object mode for a duplicate index in the CREATE INDEX statement or in a SET statement, the duplicate index is in the enabled object mode by default.

For definitions of the disabled and enabled object modes, see "Using Object Modes with Data Manipulation Statements" on page 3-86. For an explanation of the benefits of these two object modes, see "Benefits of Object Modes" on page 3-98.

List-Mode Format


List-Mode Format
CONSTRAINTS
,
constraint name
Object Modes for Constraints and Unique Indexes p. 3-80
INDEXES
,
index name
Object Modes for Constraints and Unique Indexes p. 3-80
Object Modes for Triggers and Duplicate Indexes p. 3-83
TRIGGERS
,
trigger name
Object Modes for Triggers and Duplicate Indexes p. 3-83

Element	Purpose	Restrictions	Syntax
constraint name	The name of the constraint whose object mode is to be set, or a list of constraint names. There is no default value.	Each constraint in the list must be a local constraint. All constraints in the list must be defined on the same table.	Identifier segment,*Informix Guide to SQL: Syntax*
index name	The name of the index whose object mode is to be set, or a list of index names. There is no default value.	Each index in the list must be a local index. All indexes in the list must be defined on the same table.	Identifier segment, *Informix Guide to SQL: Syntax*
trigger name	The name of the trigger whose object mode is to be set, or a list of trigger names. There is no default value.	Each trigger in the list must be a local trigger. All triggers in the list must be defined on the same table.	Identifier segment, *Informix Guide to SQL: Syntax*

Use the list-mode format to change the object mode for a particular constraint, index, or trigger. For example, to change the object mode of the unique index **unq_ssn** on the **cust_subset** table to filtering mode, enter the following statement:

```
SET INDEXES unq_ssn FILTERING
```

You can also use the list-mode format to change the object mode for a list of constraints, indexes, or triggers that are defined on the same table. Assume that four triggers are defined on the **cust_subset** table: **insert_trig**, **update_trig**, **delete_trig**, and **execute_trig**. Also assume that all four triggers are in the enabled mode. If you want to change the object mode of all the triggers except **execute_trig** to the disabled mode, enter the following statement:

```
SET TRIGGERS insert_trig, update_trig, delete_trig DISABLED
```

Using Object Modes with Data Manipulation Statements

You can use object modes to control the effects of INSERT, DELETE, and UPDATE statements. Your choice of object modes affects the tables whose data you are manipulating, the behavior of the objects defined on those tables, and the behavior of the data manipulation statements themselves.

What do we mean by the terms *enabled*, *disabled*, and *filtering*? Definitions of these object modes follow. These definitions explain how each object mode affects tables and data manipulation statements. The definitions focus on the object modes of constraints as an illustration, but the same principles apply to indexes and triggers as well.

Definition of Enabled Mode

Constraints, indexes, and triggers are in the enabled mode by default. When an object is in the enabled mode, the database server recognizes the existence of the object and takes the object into consideration while executing data manipulation statements. For example, when a constraint is enabled, any INSERT, UPDATE, or DELETE statement that violates the constraint fails, and the target row remains unchanged. In addition, the user receives an error message.

Definition of Disabled Mode

When an object is in the disabled mode, the database server acts as if the object did not exist and does not take it into consideration during the execution of data manipulation statements. For example, when a constraint is disabled, any INSERT, UPDATE, or DELETE statement that violates the constraint succeeds, and the target row is changed. The user does not receive an error message.

Definition of Filtering Mode

When an object is in the filtering mode, the object behaves the same as in the enabled mode in that the database server recognizes the existence of the object during INSERT, UPDATE, and DELETE statements. For example, when a constraint is in the filtering mode, and an INSERT, DELETE, or UPDATE statement is executed, any target rows that violate the constraint remain unchanged.

However, the database server handles data manipulation statements differently for objects in enabled and filtering mode, as described in the following paragraphs:

- If a constraint or unique index is in the enabled mode, the database server carries out the INSERT, UPDATE, or DELETE statement only if all the target rows affected by the statement satisfy the constraint or the unique index requirement. The database server updates all the target rows in the table.
- If a constraint or unique index is in the filtering mode, the database server carries out the INSERT, UPDATE, or DELETE statement even if one or more of the target rows fail to satisfy the constraint or the unique index requirement. The database server updates the good rows in the table—the target rows that satisfy the constraint or unique index requirement. The database server does not update the bad rows in the table—that is, the target rows that fail to satisfy the constraint or unique index requirement. Instead the database server sends each bad row to a special table called the violations table. The database server places information about the nature of the violation for each bad row in another special table called the diagnostics table.

Example of Object Modes with Data Manipulation Statements

We can illustrate the differences between the enabled, disabled, and filtering modes by using an INSERT statement as an example. Consider an INSERT statement in which a user tries to add a row that does not satisfy an integrity constraint on a table. For example, assume that a user **joe** has created a table named **cust_subset**, and this table consists of the following columns: **ssn** (customer's social security number), **fname** (customer's first name), **lname** (customer's last name), and **city** (city in which the customer lives). The **ssn** column has the INT data type. The other three columns have the CHAR data type.

Assume that user **joe** has defined the **lname** column as not null but has not assigned a name to the not null constraint, so the database server has implicitly assigned the name **n104_7** to this constraint. Finally, assume that user **joe** has created a unique index named **unq_ssn** on the **ssn** column.

Now a user **linda** who has the Insert privilege on the **cust_subset** table enters the following INSERT statement on this table:

```
INSERT INTO cust_subset (ssn, fname, city)
    VALUES (973824499, "jane", "los altos")
```

User **linda** has entered values for all the columns of the new row except for the **lname** column, even though the **lname** column has been defined as a not null column. The database server behaves in the following ways, depending on the object mode of the constraint:

- If the constraint is disabled, the row is inserted in the target table, and no error is returned to the user.
- If the constraint is enabled, the row is not inserted in the target table. A constraint violation error is returned to the user, and the effects of the statement are rolled back (if the database is an OnLine database with logging).
- If the constraint is filtering, the row is not inserted in the target table. Instead the row is inserted in the violations table. Information about the integrity violation caused by the row is placed in the diagnostics table. The effects of the INSERT statement are not rolled back. You receive an error message if you specified the with error option for the filtering-mode constraint. By analyzing the contents of the violations and the diagnostics tables, you can identify the reason for the failure and either take corrective action or roll back the operation.

We can better grasp the distinctions among disabled, enabled, and filtering modes by viewing the actual results of the INSERT statement shown in the preceding example.

Results of the Insert Operation When the Constraint Is Disabled

If the not null constraint on the **cust_subset** table is disabled, the INSERT statement that user **linda** issues successfully inserts the new row in this table. The new row of the **cust_subset** table has the following column values.

ssn	fname	lname	city
973824499	jane	NULL	los altos

Results of the Insert Operation When the Constraint Is Enabled

If the not null constraint on the **cust_subset** table is enabled, the INSERT statement fails to insert the new row in this table. Instead user **linda** receives the following error message when she enters the INSERT statement:

```
    -292 An implied insert column lname does not accept NULLs.
```

Results of the Insert Operation When the Constraint Is in Filtering Mode

If the not null constraint on the **cust_subset** table is set to the filtering mode, the INSERT statement that user **linda** issues fails to insert the new row in this table. Instead the new row is inserted into the violations table, and a diagnostic row that describes the integrity violation is added to the diagnostics table.

Assume that user **joe** has started a violations and diagnostics table for the **cust_subset** table. The violations table is named **cust_subset_vio**, and the diagnostics table is named **cust_subset_dia**. The new row added to the **cust_subset_vio** violations table when user **linda** issues the INSERT statement on the **cust_subset** target table has the following column values.

ssn	fname	lname	city	informix_tupleid	informix_optype	informix_recowner
973824499	jane	NULL	los altos	1	I	linda

This new row in the **cust_subset_vio** violations table has the following characteristics:

- The first four columns of the violations table exactly match the columns of the target table. These four columns have the same names and the same data types as the corresponding columns of the target table, and they have the column values that were supplied by the INSERT statement that user **linda** entered.
- The value 1 in the **informix_tupleid** column is a unique serial identifier assigned to the nonconforming row.

- The value I in the **informix_optype** column is a code that identifies the type of operation that has caused this nonconforming row to be created. Specifically, the I stands for an insert operation.
- The value linda in the **informix_recowner** column identifies the user who issued the statement that caused this nonconforming row to be created.

The INSERT statement that user **linda** issued on the **cust_subset** target table also causes a diagnostic row to be added to the **cust_subset_dia** diagnostics table. The new diagnostic row added to the diagnostics table has the following column values.

informix_tupleid	objtype	objowner	objname
1	C	joe	n104_7

This new diagnostic row in the **cust_subset_dia** diagnostics table has the following characteristics:

- This row of the diagnostics table is linked to the corresponding row of the violations table by means of the **informix_tupleid** column that appears in both tables. The value 1 appears in this column in both tables.
- The value C in the **objtype** column identifies the type of integrity violation that the corresponding row in the violations table caused. Specifically, the value C stands for a constraint violation.
- The value joe in the **objowner** column identifies the owner of the constraint for which an integrity violation was detected.
- The value n104_7 in the **objname** column gives the name of the constraint for which an integrity violation was detected.

By joining the violations and diagnostics tables, user **joe** (who owns the **cust_subset** target table and its associated special tables) or the DBA can find out that the row in the violations table whose **informix_tupleid** value is 1 was created after an INSERT statement and that this row is violating a constraint. The table owner or DBA can query the **sysconstraints** system catalog table to determine that this constraint is a not null constraint. Now that the reason for the failure of the INSERT statement is known, user **joe** or the DBA can take corrective action.

Multiple Diagnostic Rows for One Violations Row

In the preceding example, only one row in the diagnostics table corresponds to the new row in the violations table. However, more than one diagnostic row can be added to the diagnostics table when a single new row is added to the violations table. For example, if the **ssn** value (973824499) that user **linda** entered in the INSERT statement had been the same as an existing value in the **ssn** column of the **cust_subset** target table, only one new row would appear in the violations table but the following two diagnostic rows would be present in the **cust_subset_dia** diagnostics table.

informix_tupleid	objtype	objowner	objname
1	C	joe	n104_7
1	I	joe	unq_ssn

Both rows in the diagnostics table correspond to the same row of the violations table because both of these rows have the value `1` in the **informix_tupleid** column. However, the first diagnostic row identifies the constraint violation caused by the INSERT statement that user **linda** issued, while the second diagnostic row identifies the unique-index violation caused by the same INSERT statement. In this second diagnostic row, the value `I` in the **objtype** column stands for a unique-index violation, and the value `unq_ssn` in the **objname** column gives the name of the index for which the integrity violation was detected.

For information on when and how to start violations and diagnostics tables for a target table, see "Violations and Diagnostics Tables for Filtering Mode" on page 3-82. For further information on the structure of the violations and diagnostics tables, see the START VIOLATIONS TABLE statement in this guide.

Using Object Modes to Achieve Data Integrity

In addition to using object modes with data manipulation statements, you can also use object modes when you add a new constraint or new unique index to a target table. By selecting the correct object mode, you can add the constraint or index to the target table easily even if existing rows in the target table violate the new integrity specification.

You can add a new constraint or index easily by taking the following steps. If you follow this procedure, you do not have to examine the entire target table to identify rows that fail to satisfy the constraint or unique-index requirement.

- Add the constraint or index in the enabled mode. If all existing rows in the table satisfy the constraint or unique-index requirement, your ALTER TABLE or CREATE INDEX statement executes successfully, and you do not need to take any further steps. However, if any existing rows in the table fail to satisfy the constraint or unique-index requirement, your ALTER TABLE or CREATE INDEX statement returns an error message, and you need to take the following steps.
- Add the constraint or index in the disabled mode. Issue the ALTER TABLE statement again and specify the DISABLED keyword in the ADD CONSTRAINT or MODIFY clause, or issue the CREATE INDEX statement again and specify the DISABLED keyword.
- Start a violations and diagnostics table for the target table with the START VIOLATIONS TABLE statement.
- Issue a SET statement to switch the object mode of the constraint or index to the enabled mode. When you issue this statement, the statement fails and existing rows in the target table that violate the constraint or the unique-index requirement are duplicated in the violations table. The constraint or index remains disabled, and you receive an integrity-violation error message.
- Issue a SELECT statement on the violations table to retrieve the nonconforming rows that were duplicated from the target table. You might need to join the violations and diagnostics tables to get all the necessary information.

- Take corrective action on the rows in the target table that violate the constraint.
- After you fix all the nonconforming rows in the target table, issue the SET statement again to switch the disabled constraint or index to the enabled mode. This time the constraint or index is enabled and no integrity-violation error message is returned because all rows in the target table now satisfy the new constraint or unique-index requirement.

Example of Using Object Modes to Achieve Data Integrity

The following example shows how to use object modes to add a constraint and unique index to a target table easily. Assume that a user **joe** has created a table named **cust_subset**, and this table consists of the following columns: **ssn** (customer's social security number), **fname** (customer's first name), **lname** (customer's last name), and **city** (city in which the customer lives).

Also assume that no constraints or unique indexes are defined on the **cust_subset** table and that the **fname** column is the primary key. In addition, assume that no violations and diagnostics tables currently exist for this target table. Finally, assume that this table currently contains four rows with the following column values.

ssn	fname	lname	city
111763227	mark	jackson	sunnyvale
222781244	rhonda	NULL	palo alto
111763227	steve	NULL	san mateo
333992276	tammy	jones	san jose

Adding the Objects in the Enabled Mode

User **joe**, the owner of the **cust_subset** table, enters the following statements to add a unique index on the **ssn** column and a not null constraint on the **lname** column:

```
CREATE UNIQUE INDEX unq_ssn ON cust_subset (ssn) ENABLED;
ALTER TABLE cust_subset MODIFY (lname CHAR(15)
    NOT NULL CONSTRAINT lname_notblank ENABLED);
```

Both of these statements fail because existing rows in the **cust_subset** table violate the integrity specifications. The row whose **fname** value is `rhonda` violates the not null constraint on the **lname** column. The row whose **fname** value is `steve` violates both the not null constraint on the **lname** column and the unique-index requirement on the **ssn** column.

Adding the Objects in the Disabled Mode

To recover from the preceding errors, user **joe** reenters the CREATE INDEX and ALTER TABLE statements and specifies the disabled mode in both statements, as follows:

```
CREATE UNIQUE INDEX unq_ssn ON cust_subset (ssn) DISABLED;
ALTER TABLE cust_subset MODIFY (lname CHAR(15)
    NOT NULL CONSTRAINT lname_notblank DISABLED);
```

Both of these statements execute successfully because the database server does not enforce unique-index requirements or constraint specifications when these objects are disabled.

Starting a Violations and Diagnostics Table

Now that the new constraint and index are added for the **cust_subset** table, user **joe** takes steps to find out which existing rows in the **cust_subset** table violate the constraint and the index.

First, user **joe** enters the following statement to start a violations and diagnostics table for the **cust_subset** table:

```
START VIOLATIONS TABLE FOR cust_subset
```

Because user **joe** has not assigned names to the violations and diagnostics tables in this statement, the tables are named **cust_subset_vio** and **cust_subset_dia** by default.

Using the SET Statement to Capture Violations

Now that violations and diagnostics tables exist for the target table, user **joe** issues the following SET statement to switch the mode of the new index and constraint from the disabled mode to the enabled mode:

```
SET CONSTRAINTS, INDEXES FOR cust_subset ENABLED
```

The result of this SET statement is that the existing rows in the **cust_subset** table that violate the constraint and the unique-index requirement are copied to the **cust_subset_vio** violations table, and diagnostic information about the nonconforming rows is added to the **cust_subset_dia** diagnostics table. The SET statement fails, and the constraint and index remain disabled.

The following table shows the contents of the **cust_subset_vio** violations table after user **joe** issues the SET statement:

ssn	fname	lname	city	informix_tupleid	informix_optype	informix_recowner
222781244	rhonda	NULL	palo alto	1	S	joe
111763227	steve	NULL	san mateo	2	S	joe

These two rows in the **cust_subset_vio** violations table have the following characteristics:

- The row in the **cust_subset** target table whose **fname** value is `rhonda` is duplicated to the **cust_subset_vio** violations table because this row violates the not null constraint on the **lname** column.
- The row in the **cust_subset** target table whose **fname** value is `steve` is duplicated to the **cust_subset_vio** violations table because this row violates the not null constraint on the **lname** column and the unique-index requirement on the **ssn** column.
- The value `1` in the **informix_tupleid** column for the first row and the value `2` in the **informix_tupleid** column for the second row are unique serial identifiers assigned to the nonconforming rows.

- The value S in the **informix_optype** column for each row is a code that identifies the type of operation that has caused this nonconforming row to be placed in the violations table. Specifically, the S stands for a SET statement.
- The value joe in the **informix_recowner** column for each row identifies the user who issued the statement that caused this nonconforming row to be placed in the violations table.

The following table shows contents of the **cust_subset_dia** diagnostics table after user **joe** issues the SET statement:

informix_tupleid	objtype	objowner	objname
1	C	joe	lname_notblank
2	C	joe	lname_notblank
2	I	joe	unq_ssn

These three rows in the **cust_subset_dia** diagnostics table have the following characteristics:

- Each row in the diagnostics table and the corresponding row in the violations table are joined by the **informix_tupleid** column that appears in both tables.
- The first row in the diagnostics table has an **informix_tupleid** value of 1. It is joined to the row in the violations table whose **informix_tupleid** value is 1. The value C in the **objtype** column for this diagnostic row identifies the type of integrity violation that was caused by the corresponding row in the violations table. Specifically, the value C stands for a constraint violation. The value lname_notblank in the **objname** column for this diagnostic row gives the name of the constraint for which an integrity violation was detected.

- The second row in the diagnostics table has an **informix_tupleid** value of 2. It is joined to the row in the violations table whose **informix_tupleid** value is 2. The value C in the **objtype** column for this second diagnostic row indicates that a constraint violation was caused by the corresponding row in the violations table. The value `lname_notblank` in the **objname** column for this diagnostic row gives the name of the constraint for which an integrity violation was detected.
- The third row in the diagnostics table has an **informix_tupleid** value of 2. It is also joined to the row in the violations table whose **informix_tupleid** value is 2. The value I in the **objtype** column for this third diagnostic row indicates that a unique-index violation was caused by the corresponding row in the violations table. The value `unq_ssn` in the **objname** column for this diagnostic row gives the name of the index for which an integrity violation was detected.
- The value `joe` in the **objowner** column of all three diagnostic rows identifies the owner of the object for which an integrity violation was detected. The name of user **joe** appears in all three rows because he created the constraint and index on the **cust_subset** table.

Identifying Nonconforming Rows and Obtaining Diagnostic Information

To determine the contents of the violations table, user **joe** enters a SELECT statement to retrieve all rows from the table. Then, to obtain complete diagnostic information about the nonconforming rows, user **joe** joins the violations and diagnostics tables by means of another SELECT statement. User **joe** can perform these operations either interactively or through a program.

Taking Corrective Action on the Nonconforming Rows

After the user **joe** identifies the nonconforming rows in the **cust_subset** table, he can correct them. For example, he can enter UPDATE statements on the **cust_subset** table either interactively or through a program.

Enabling the Disabled Objects

Once all the nonconforming rows in the **cust_subset** table are corrected, user **joe** issues the following SET statement to set the new constraint and index to the enabled mode:

```
    SET CONSTRAINTS, INDEXES FOR cust_subset ENABLED
```

This time the SET statement executes successfully. The new constraint and new unique index are enabled and no error message is returned to user **joe** because all rows in the **cust_subset** table now satisfy the new constraint specification and unique-index requirement.

Benefits of Object Modes

The preceding examples show how object modes work when users execute data manipulation statements on target tables or add new constraints and indexes to target tables. The preceding examples suggest some of the benefits of the different object modes. The following sections state these benefits explicitly.

Benefits of Disabled Mode

The benefits of the disabled mode are as follows:

- You can use the disabled mode to insert many rows quickly into a target table. Especially during load operations, updates of the existing indexes and enforcement of referential constraints make up a big part of the total cost of the operation. By disabling the indexes and referential constraints during the load operation, you improve the performance and efficiency of the load.
- To add a new constraint or new unique index to an existing table, you can add the object even if some rows in the table do not satisfy the new integrity specification. If the constraint or index is added to the table in disabled mode, your ALTER TABLE or CREATE INDEX statement does not fail no matter how many existing rows violate the new integrity requirement.

  If a violations table has been started, a SET statement that switches the disabled objects to the enabled or filtering mode fails, but causes the nonconforming rows in the target table to be duplicated in the violations table so that you can identify the rows and take corrective action. After you fix the nonconforming rows in the target table, you can reissue the SET statement to switch the disabled objects to the enabled or filtering mode.

Benefits of Enabled Mode

The enabled mode is the default object mode for all database objects. We can summarize the benefits of this mode for each type of database object as follows:

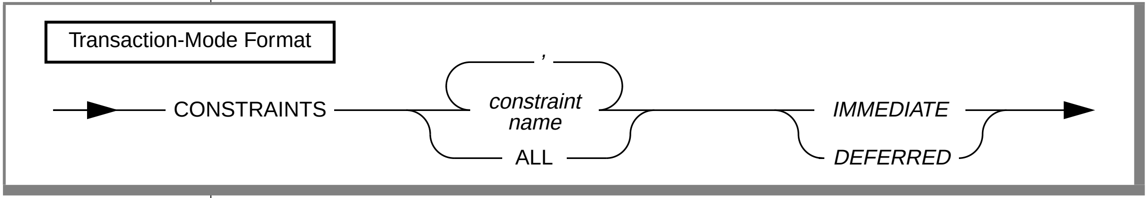
- The benefit of enabled mode for constraints is that the database server enforces the constraint and thus ensures the consistency of the data in the database.
- The benefit of enabled mode for indexes is that the database server updates the index after insert, delete, and update operations. Thus the index is up to date and is used by the optimizer during database queries.
- The benefit of enabled mode for triggers is that the trigger event always sets the triggered action in motion. Thus the purpose of the trigger is always realized during actual data-manipulation operations.

Benefits of Filtering Mode

The benefits of setting a constraint or unique index to the filtering mode are as follows:

- During load operations, inserts that violate a filtering mode constraint or unique index do not cause the load operation to fail. Instead, the database server filters the bad rows to the violations table and continues the load operation.
- When an INSERT, DELETE, or UPDATE statement that affects multiple rows causes a filtering mode constraint or unique index to be violated for a particular row or rows, the statement does not fail. Instead, the database server filters the bad row or rows to the violations table and continues to execute the statement.
- When any INSERT, DELETE, or UPDATE statement violates a filtering mode constraint or unique index, the user can identify the failed row or rows and take corrective action. The violations and diagnostics tables capture the necessary information, and users can take corrective action after they analyze this information.

Transaction-Mode Format



Element	Purpose	Restrictions	Syntax
constraint name	The name of the constraint whose transaction mode is to be changed, or a list of constraint names. There is no default value.	The specified constraint must exist in an OnLine database with logging. You cannot change the transaction mode of a constraint to deferred mode unless the constraint is currently in the enabled mode. All constraints in a list of constraints must exist in the same database.	Identifier segment, *Informix Guide to SQL: Syntax*

You can use the transaction-mode format of the SET statement to set the transaction mode of constraints.

You use the IMMEDIATE keyword to set the transaction mode of constraints to statement-level checking. You use the DEFERRED keyword to set the transaction mode to transaction-level checking.

You can set the transaction mode of constraints only in an OnLine database with logging.

Statement-Level Checking

When you set the transaction mode to immediate, statement-level checking is turned on, and all specified constraints are checked at the end of each INSERT, UPDATE, or DELETE statement. If a constraint violation occurs, the statement is not executed. Immediate is the default transaction mode of constraints.

Transaction-Level Checking

When you set the transaction mode of constraints to deferred, statement-level checking is turned off, and all specified constraints are not checked until the transaction is committed. If a constraint violation occurs while the transaction is being committed, the transaction is rolled back.

Tip: *If you defer checking a primary-key constraint, the checking of the not null constraint for that column or set of columns is also deferred.*

Duration of Transaction Modes

The duration of the transaction mode that the SET statement specifies is the transaction in which the SET statement is executed. You cannot execute this form of the SET statement outside a transaction. Once a COMMIT WORK or ROLLBACK WORK statement is successfully completed, the transaction mode of all constraints reverts to IMMEDIATE.

Switching Transaction Modes

To switch from transaction-level checking to statement-level checking, you can use the SET statement to set the transaction mode to immediate, or you can use a COMMIT WORK or ROLLBACK WORK statement in your transaction.

Specifying All Constraints or a List of Constraints

You can specify all constraints in the database in your SET statement, or you can specify a single constraint or list of constraints.

Specifying All Constraints

If you specify the ALL keyword, the SET statement sets the transaction mode for all constraints in the database. If any statement in the transaction requires that any constraint on any table in the database be checked, the database server performs the checks at the statement level or the transaction level, depending on the setting that you specify in the SET statement.

Specifying a List of Constraints

If you specify a single constraint name or a list of constraints, the SET statement sets the transaction mode for the specified constraints only. If any statement in the transaction requires checking of a constraint that you did not specify in the SET statement, that constraint is checked at the statement level regardless of the setting that you specified in the SET statement for other constraints.

When you specify a list of constraints, the constraints do not have to be defined on the same table, but they must exist in the same database.

Specifying Remote Constraints

You can set the transaction mode of local constraints or remote constraints. That is, the constraints specified in the transaction-mode form of the SET statement can be constraints defined on local tables or constraints defined on remote tables.

Examples of Setting the Transaction Mode for Constraints

The following example shows how to defer checking constraints within a transaction until the transaction is complete. The SET CONSTRAINTS statement in the example specifies that any constraints on any tables in the database are not checked until the COMMIT WORK statement is encountered.

```
BEGIN WORK
SET CONSTRAINTS ALL DEFERRED
.
.
.
COMMIT WORK
```

The following example specifies that a list of constraints is not checked until the transaction is complete:

```
BEGIN WORK
SET CONSTRAINTS update_const, insert_const DEFERRED
.
.
.
COMMIT WORK
```

References

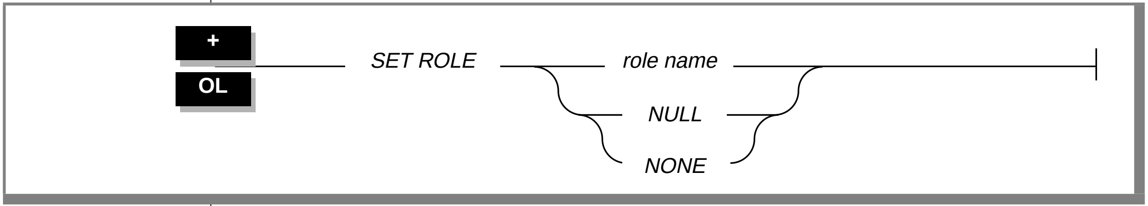
See the START VIOLATIONS TABLE and STOP VIOLATIONS TABLE statements in this guide.

For information on the system catalog tables associated with the SET statement, see the SYSOBJSTATE and SYSVIOLATIONS tables in this guide.

SET ROLE

Use the SET ROLE statement to enable the privileges of a role.

Syntax



Element	Purpose	Restrictions	Syntax
role name	Name of the role	The role must have been created with the CREATE ROLE statement	Identifier segment, *Informix Guide to SQL: Syntax*

Usage

Any user who is granted a role can enable the role using the SET ROLE statement. A user can only enable one role at a time. If a user executes the SET ROLE statement after a role is already set, the new role replaces the old role.

All users are, by default, assigned the role NULL or NONE (NULL and NONE are synonymous.) The roles NULL and NONE have no privileges. When you set the role to NULL or NONE, you disable the current role.

When a user sets a role, the user gains the privileges of the role, in addition to the privileges of PUBLIC and the user's own privileges. If a role is granted to another role, the user gains the privileges of both roles, in addition to those of PUBLIC and the user's own privileges. After a SET ROLE statement executes successfully, the role remains effective until the current database is closed or the user executes another SET ROLE statement. Additionally, the user, not the role, retains ownership of all the objects, such as tables, created during a session.

A user cannot execute the SET ROLE statement while in a transaction. If the SET ROLE statement is executed while a transaction is active, an error occurs.

If the SET ROLE statement is executed as a part of a trigger or stored procedure and the owner of the trigger or stored procedure was granted the role with the WITH GRANT OPTION, the role is enabled even if the user is not granted the role.

The following example sets the role **engineer**:

```
SET ROLE engineer
```

The following example sets a role and then relinquishes the role after performing a SELECT operation:

```
EXEC SQL SET ROLE engineer;
EXEC SQL SELECT fname, lname, project
        INTO :efname, :elname, :eproject
        WHERE project_num > 100 AND lname = 'Larkin';
printf ("%s is working on %s\n", :efname, :eproject);
EXEC SQL SET ROLE NULL;
```

References

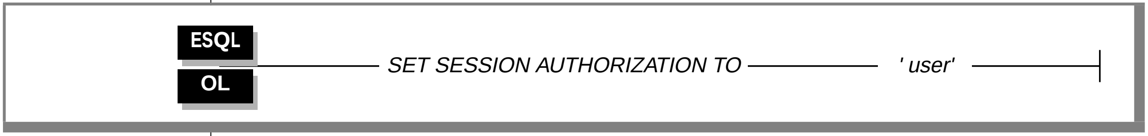
See the GRANT and REVOKE statements in this guide and in the *Informix Guide to SQL: Syntax*.

See the CREATE ROLE and DROP ROLE statements in this guide.

SET SESSION AUTHORIZATION

The SET SESSION AUTHORIZATION statement lets you change the user name under which database operations are performed in the current OnLine session. This statement is enabled by the DBA privilege, which you must obtain from the DBA before the start of your current session. The new identity remains in effect until you execute another SET SESSION AUTHORIZATION statement or until you close the current database.

Syntax



Element	Purpose	Restrictions	Syntax
' *user* '	The name of a user	Must be a valid user name	Identifier, *Informix Guide to SQL: Syntax*

Usage

The SET SESSION AUTHORIZATION statement allows a user with the DBA privilege to bypass the privileges that protect database objects. You can use this statement to gain access to a table and adopt the identity of a table owner to grant access privileges. You must obtain the DBA privilege before you start a session in which you use this statement. Otherwise, this statement returns an error.

When you use this statement, the user name to which the authorization is set must have the Connect privilege on the current database. Additionally, the DBA cannot set the authorization to PUBLIC or to any defined role in the current database.

The SET SESSION AUTHORIZATION statement applies only to OnLine, INFORMIX-ESQL/C and INFORMIX-ESQL/COBOL. When you cannot issue the statement directly, you can prepare and execute a dynamic SQL statement instead. For more information on dynamic SQL, refer to the *Informix Guide to SQL: Tutorial*.

Setting a session to another user causes a change in a user name in the current active database server. In other words, these users are, as far as this database server process is concerned, completely dispossessed of any privileges that they might have while accessing the database server through some administrative utility. Additionally, the new session user is not able to initiate an administrative operation (execute a utility, for example) by virtue of the acquired identity.

After the SET SESSION AUTHORIZATION statement successfully executes, the user must use the SET ROLE statement to assume a role granted to the current user. Any role enabled by a previous user is relinquished.

Using the SET SESSION AUTHORIZATION Statement to Obtain Privileges

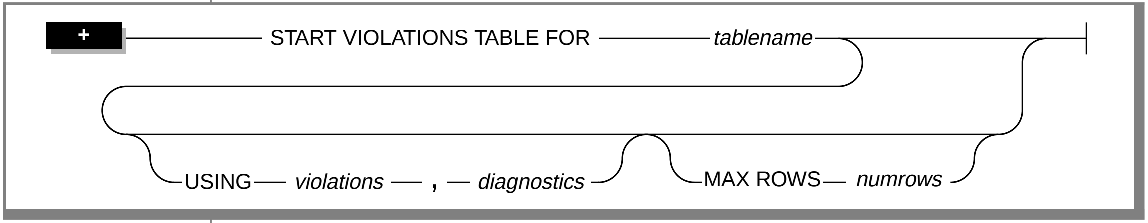
You can use the SET SESSION AUTHORIZATION statement either to obtain access to the data directly or to grant the database-level or table-level privileges needed for the database operation to proceed. The following example shows how to use the SET SESSION AUTHORIZATION statement to obtain table-level privileges:

```
SET SESSION AUTHORIZATION TO 'cathl';
GRANT ALL ON spec TO mary;
SET SESSION AUTHORIZATION TO 'mary';
UPDATE case
    SET col1 = SELECT state FROM zip
                WHERE zip_code = 94433;
```

START VIOLATIONS TABLE

The START VIOLATIONS TABLE statement creates a violations table and a diagnostics table for a specified target table. The database server associates the violations and diagnostics tables with the target table by recording the relationship among the three tables in the **sysviolations** system catalog table.

Syntax



Element	Purpose	Restrictions	Syntax
diagnostics	The name of the diagnostics table to be associated with the target table. The default name is the name of the target table followed by the characters **_dia**. For further information on the diagnostics table, see "Structure of the Diagnostics Table" on page 3-121.	Whether you specify the name of the diagnostics table explicitly or the database server generates the name implicitly, the name cannot match the name of any existing table in the database.	Identifier segment, *Informix Guide to SQL: Syntax*
numrows	The maximum number of rows that can be inserted into the diagnostics table when a single statement (for example, INSERT or SET) is executed on the target table. There is no default value for *numrows*. If you do not specify a value for *numrows*, there is no upper limit on the number of rows that can be inserted into the diagnostics table when a single statement is executed on the target table.	You must specify an integer value in the range 1 to the maximum value of the INTEGER data type.	Literal Number segment, *Informix Guide to SQL: Syntax*

(1 of 2)

Element	Purpose	Restrictions	Syntax
table name	The name of the target table for which a violations table and diagnostics table are to be created. There is no default value.	If you do not include the USING clause in the statement, the name of the target table must be less than 15 characters. The target table cannot have a violations and diagnostics table associated with it before you execute the statement. The target table cannot be a system catalog table. The target table must be a local table. For information on the privileges required to start a violations and diagnostics table for a target table, see "Privileges Required for Starting Violations Tables" on page 3-112.	Identifier segment, *Informix Guide to SQL: Syntax*
violations	The name of the violations table to be associated with the target table. The default name is the name of the target table followed by the characters **_vio**. For further information on the violations table, see "Structure of the Violations Table" on page 3-113.	Whether you specify the name of the violations table explicitly or the database server generates the name implicitly, the name cannot match the name of any existing table in the database.	Identifier segment, *Informix Guide to SQL: Syntax*

(2 of 2)

Usage

The START VIOLATIONS TABLE statement creates the special violations table that holds rows that fail to satisfy constraints and unique indexes during insert, update, and delete operations on target tables. This statement also creates the special diagnostics table that contains information about the integrity violations caused by each row in the violations table.

Relationship of START VIOLATIONS TABLE and SET Statements

The START VIOLATIONS TABLE statement is closely related to the SET statement. If you use the SET statement to set the constraints or unique indexes defined on a table to the filtering object mode, but you do not use the START VIOLATIONS TABLE statement to start the violations and diagnostics tables for this target table, any rows that violate a constraint or unique index requirement during an insert, update, or delete operation are not filtered out to a violations table. Instead you receive an error message indicating that you must start a violations table for the target table.

Similarly, if you use the SET statement to set a disabled constraint or disabled unique index to the enabled or filtering object mode, but you do not use the START VIOLATIONS TABLE statement to start the violations and diagnostics tables for the table on which the objects are defined, any existing rows in the table that do not satisfy the constraint or unique-index requirement are not filtered out to a violations table. If, in these cases, you want the ability to identify existing rows that do not satisfy the constraint or unique-index requirement, you must issue the START VIOLATIONS TABLE statement to start the violations and diagnostics tables before you issue the SET statement to set the objects to the enabled or filtering object mode.

Starting and Stopping the Violations and Diagnostics Tables

After you use a START VIOLATIONS TABLE statement to create an association between a target table and the violations and diagnostics tables, the only way to drop the association between the target table and the violations and diagnostics tables is to issue a STOP VIOLATIONS TABLE statement for the target table. For further information on the STOP VIOLATIONS TABLE statement, see “STOP VIOLATIONS TABLE” on page 3-128.

Examples of START VIOLATIONS TABLE Statements

The following examples show different ways to execute the START VIOLATIONS TABLE statement.

Starting Violations and Diagnostics Tables Without Specifying Their Names

The following statement starts violations and diagnostics tables for the target table named **cust_subset**. The violations table is named **cust_subset_vio** by default, and the diagnostics table is named **cust_subset_dia** by default.

```
START VIOLATIONS TABLE FOR cust_subset
```

Starting Violations and Diagnostics Tables and Specifying Their Names

The following statement starts a violations and diagnostics table for the target table named **items**. The USING clause assigns explicit names to the violations and diagnostics tables. The violations table is to be named **exceptions**, and the diagnostics table is to be named **reasons**.

```
START VIOLATIONS TABLE FOR items
USING exceptions, reasons
```

Specifying the Maximum Number of Rows in the Diagnostics Table

The following statement starts violations and diagnostics tables for the target table named **orders**. The MAX ROWS clause specifies the maximum number of rows that can be inserted into the diagnostics table when a single statement, such as an INSERT or SET statement, is executed on the target table.

```
START VIOLATIONS TABLE FOR orders MAX ROWS 50000
```

Privileges Required for Starting Violations Tables

To start a violations and diagnostics table for a target table, you must meet one of the following requirements:

- You must have the DBA privilege on the database.
- You must be the owner of the target table and have the Resource privilege on the database.
- You must have the Alter privilege on the target table and the Resource privilege on the database.

Structure of the Violations Table

When you issue a START VIOLATIONS TABLE statement for a target table, the violations table created by the statement has a predefined structure. This structure consists of the columns of the target table and three additional columns.

The following table shows the structure of the violations table.

Column Name	Type	Explanation
All columns of the target table, in the same order that they appear in the target table	These columns of the violations table match the data type of the corresponding columns in the target table, except that SERIAL columns in the target table are converted to INTEGER data types in the violations table.	The table definition of the target table is reproduced in the violations table so that rows that violate constraints or unique index requirements during insert, update, and delete operations can be filtered to the violations table. Users can examine these bad rows in the violations table, analyze the related rows that contain diagnostics information in the diagnostics table, and take corrective actions.
informix_tupleid	SERIAL	This column contains the unique serial identifier that is assigned to the nonconforming row.

informix_optype	CHAR(1)	This column indicates the type of operation that caused this bad row. This column can have the following values:
		I = Insert
		D = Delete
		O = Update (with this row containing the original values)
		N = Update (with this row containing the new values)
		S = SET statement

informix_recowner	CHAR(8)	This column identifies the user who issued the statement that created this bad row.

Relationship Between the Violations and Diagnostics Tables

Users can take advantage of the relationships among the target table, violations table, and diagnostics table to obtain complete diagnostic information about rows that have caused data-integrity violations during INSERT, DELETE, and UPDATE statements.

Each row of the violations table has at least one corresponding row in the diagnostics table. The row in the violations table contains a copy of the row in the target table for which a data-integrity violation was detected. The row in the diagnostics table contains information about the nature of the data-integrity violation caused by the bad row in the violations table. The row in the violations table has a unique serial identifier in the **informix_tupleid** column. The row in the diagnostics table has the same serial identifier in its **informix_tupleid** column.

A given row in the violations table can have more than one corresponding row in the diagnostics table. The multiple rows in the diagnostics table all have the same serial identifier in their **informix_tupleid** column so that they are all linked to the same row in the violations table. Multiple rows can exist in the diagnostics table for the same row in the violations table because a bad row in the violations table can cause more than one data-integrity violation.

For example, a bad row can violate a unique-index requirement for one column, a not null constraint for another column, and a check constraint for yet another column. In this case, the diagnostics table contains three rows for the single bad row in the violations table. Each of these diagnostic rows identifies a different data-integrity violation caused by the nonconforming row in the violations table.

By joining the violations and diagnostics tables, the DBA or target table owner can obtain complete diagnostic information about any or all bad rows in the violations table. You can use SELECT statements to perform these joins interactively, or you can write a program to perform them within transactions.

Initial Privileges on the Violations Table

When you issue the START VIOLATIONS TABLE statement to create the violations table, the database server uses the set of privileges granted on the target table as a basis for granting privileges on the violations table. However, the database server follows different rules when it grants each type of privilege.

The following table shows the initial set of privileges on the violations table. The **Privilege** column lists the privilege. The **Condition** column explains the conditions under which the database server grants the privilege to a user.

Privilege	Condition
Insert	The user has the Insert privilege on the violations table if the user has any of the following privileges on the target table: the Insert privilege, the Delete privilege, or the Update privilege on any column.
Delete	The user has the Delete privilege on the violations table if the user has any of the following privileges on the target table: the Insert privilege, the Delete privilege, or the Update privilege on any column.
Select	The user has the Select privilege on the **informix_tupleid**, **informix_optype**, and **informix_recowner** columns of the violations table if the user has the Select privilege on any column of the target table.
	The user has the Select privilege on any other column of the violations table if the user has the Select privilege on the same column in the target table.
Update	The user has the Update privilege on the **informix_tupleid**, **informix_optype**, and **informix_recowner** columns of the violations table if the user has the Update privilege on any column of the target table.
	The user has the Update privilege on any other column of the violations table if the user has the Update privilege on the same column in the target table.

(1 of 2)

Privilege	**Condition**
Index	The user has the Index privilege on the violations table if the user has the Index privilege on the target table.
Alter	The Alter privilege is not granted on the violations table. (Users cannot alter violations tables.)
References	The References privilege is not granted on the violations table. (Users cannot add referential constraints to violations tables.)

(2 of 2)

The following rules apply to ownership of the violations table and privileges on the violations table:

- When the violations table is created, the owner of the target table becomes the owner of the violations table.
- The owner of the violations table automatically receives all table-level privileges on the violations table, including the Alter and References privileges. However, the database server prevents the owner of the violations table from altering the violations table or adding a referential constraint to the violations table.
- You can use the GRANT and REVOKE statements to modify the initial set of privileges on the violations table.
- When you issue an INSERT, DELETE, or UPDATE statement on a target table that has a filtering mode unique index or constraint defined on it, you must have the Insert privilege on the violations and diagnostics tables.

  If you do not have the Insert privilege on the violations and diagnostics tables, the database server executes the INSERT, DELETE, or UPDATE statement on the target table provided that you have the necessary privileges on the target table. The database server does not return an error concerning the lack of insert permission on the violations and diagnostics tables unless an integrity violation is detected during the execution of the INSERT, DELETE, or UPDATE statement.

Similarly, when you issue a SET statement to set a disabled constraint or disabled unique index to the enabled or filtering mode, and a violations and diagnostics table exist for the target table, you must have the Insert privilege on the violations and diagnostics tables.

If you do not have the Insert privilege on the violations and diagnostics tables, the database server executes the SET statement provided that you have the necessary privileges on the target table. The database server does not return an error concerning the lack of insert permission on the violations and diagnostics tables unless an integrity violation is detected during the execution of the SET statement.

- The grantor of the initial set of privileges on the violations table is the same as the grantor of the privileges on the target table. For example, if the user **henry** has been granted the Insert privilege on the target table by both the user **jill** and the user **albert**, the Insert privilege on the violations table is granted to user **henry** both by user **jill** and by user **albert**.
- Once a violations table has been started for a target table, revoking a privilege on the target table from a user does not automatically revoke the same privilege on the violations table from that user. Instead you must explicitly revoke the privilege on the violations table from the user.
- If you have fragment-level privileges on the target table, you have the corresponding fragment-level privileges on the violations table.

Example of Privileges on the Violations Table

The following example illustrates how the initial set of privileges on a violations table is derived from the current set of privileges on the target table.

For example, assume that we have created a table named **cust_subset** and that this table consists of the following columns: **ssn** (customer's social security number), **fname** (customer's first name), **lname** (customer's last name), and **city** (city in which the customer lives).

The following set of privileges exists on the **cust_subset** table:

- User **alvin** is the owner of the table.
- User **barbara** has the Insert and Index privileges on the table. She also has the Select privilege on the **ssn** and **lname** columns.

- User **carrie** has the Update privilege on the **city** column. She also has the Select privilege on the **ssn** column.
- User **danny** has the Alter privilege on the table.

Now user **alvin** starts a violations table named **cust_subset_viols** and a diagnostics table named **cust_subset_diags** for the **cust_subset** table, as follows:

```
START VIOLATIONS TABLE FOR cust_subset
    USING cust_subset_viols, cust_subset_diags
```

The database server grants the following set of initial privileges on the **cust_subset_viols** violations table:

- User **alvin** is the owner of the violations table, so he has all table-level privileges on the table.
- User **barbara** has the Insert, Delete, and Index privileges on the violations table. She also has the Select privilege on the following columns of the violations table: the **ssn** column, the **lname** column, the **informix_tupleid** column, the **informix_optype** column, and the **informix_recowner** column.
- User **carrie** has the Insert and Delete privileges on the violations table. She has the Update privilege on the following columns of the violations table: the **city** column, the **informix_tupleid** column, the **informix_optype** column, and the **informix_recowner** column. She has the Select privilege on the following columns of the violations table: the **ssn** column, the **informix_tupleid** column, the **informix_optype** column, and the **informix_recowner** column.
- User **danny** has no privileges on the violations table.

Using the Violations Table

The following rules concern the structure and use of the violations table:

- Every pair of update rows in the violations table has the same value in the **informix_tupleid** column to indicate that both rows refer to the same row in the target table.
- If the target table has columns named **informix_tupleid**, **informix_optype**, or **informix_recowner**, the database server attempts to generate alternative names for these columns in the violations table by appending a digit to the end of the column name (for example, **informix_tupleid1**). If this attempt fails, the database server returns an error, and the violations table is not started for the target table.
- When a table functions as a violations table, it cannot have triggers or constraints defined on it.
- When a table functions as a violations table, users can create indexes on the table, even though the existence of an index affects performance. Unique indexes on the violations table cannot be set to the filtering object mode.
- If a target table has a violations and diagnostics table associated with it, dropping the target table in cascade mode (the default mode) causes the violations and diagnostics tables to be dropped also. If the target table is dropped in the restricted mode, the existence of the violations and diagnostics tables causes the DROP TABLE statement to fail.
- Once a violations table is started for a target table, you cannot use the ALTER TABLE statement to add, modify, or drop columns in the target table, violations table, or diagnostics table. Before you can alter any of these tables, you must issue a STOP TABLE VIOLATIONS statement for the target table.
- The database server does not clear out the contents of the violations table before or after it uses the violations table during an Insert, Update, Delete, or Set operation.

- If a target table has a filtering-mode constraint or unique index defined on it and a violations table associated with it, users cannot insert into the target table by selecting from the violations table. Before you insert rows into the target table by selecting from the violations table, you must take one of the following steps:
    - You can set the object mode of the constraint or unique index to the enabled or disabled object mode.
    - You can issue a STOP VIOLATIONS TABLE statement for the target table.

  If it is inconvenient to take either of these steps, but you still want to copy records from the violations table into the target table, a third option is to select from the violations table into a temporary table and then insert the contents of the temporary table into the target table.
- If the target table specified in the START VIOLATIONS TABLE statement is fragmented, the violations table has the same fragmentation strategy as the target table. Each fragment of the violations table is stored in the same dbspace as the corresponding fragment of the target table.
- If the target table specified in the START VIOLATIONS TABLE statement is not fragmented, the database server places the violations table in the same dbspace as the target table.
- If the target table has blob columns, blobs in the violations table are created in the same blob space as the blobs in the target table.

Example of a Violations Table

To start a violations and diagnostics table for the target table named **customer** in the **stores7** demonstration database, enter the following statement:

```
START VIOLATIONS TABLE FOR customer
```

Because your START VIOLATIONS statement does not include a USING clause, the violations table is named **customer_vio** by default. The **customer_vio** table includes the following columns:

```
customer_num
fname
lname
company
address1
address2
city
state
zipcode
phone
informix_tupleid
informix_optype
informix_recowner
```

The **customer_vio** table has the same table definition as the **customer** table except that the **customer_vio** table has three additional columns that contain information about the operation that caused the bad row.

Structure of the Diagnostics Table

When you issue a START VIOLATIONS TABLE statement for a target table, the diagnostics table created by the statement has a predefined structure. This structure is independent of the structure of the target table.

The following table shows the structure of the diagnostics table.

Column Name	Type	Explanation
informix_tupleid	INTEGER	This column in the diagnostics table implicitly refers to the values in the **informix_tupleid** column in the violations table. However, this relationship is not declared as a foreign-key to primary-key relationship.

objtype	CHAR(1)	This column identifies the type of the violation. This column can have the following values.
		C = Constraint violation
		I = Unique index violation

(1 of 2)

Column Name	Type	Explanation
objowner	CHAR(8)	This column identifies the owner of the constraint or index for which an integrity violation was detected.
objname	CHAR(18)	This column contains the name of the constraint or index for which an integrity violation was detected.

(2 of 2)

Initial Privileges on the Diagnostics Table

When the START VIOLATIONS TABLE statement creates the diagnostics table, the database server uses the set of privileges granted on the target table as a basis for granting privileges on the diagnostics table. However, the database server follows different rules when it grants each type of privilege.

The following table shows the initial set of privileges on the diagnostics table. The **Privilege** column lists the privilege. The **Condition** column explains the conditions under which the database server grants the privilege to a user.

Privilege	Condition
Insert	The user has the Insert privilege on the diagnostics table if the user has any of the following privileges on the target table: the Insert privilege, the Delete privilege, or the Update privilege on any column.
Delete	The user has the Delete privilege on the diagnostics table if the user has any of the following privileges on the target table: the Insert privilege, the Delete privilege, or the Update privilege on any column.
Select	The user has the Select privilege on the diagnostics table if the user has the Select privilege on any column in the target table.
Update	The user has the Update privilege on the diagnostics table if the user has the Update privilege on any column in the target table.

(1 of 2)

Privilege	Condition
Index	The user has the Index privilege on the diagnostics table if the user has the Index privilege on the target table.
Alter	The Alter privilege is not granted on the diagnostics table. (Users cannot alter diagnostics tables.)
References	The References privilege is not granted on the diagnostics table. (Users cannot add referential constraints to diagnostics tables.)

(2 of 2)

The following rules concern privileges on the diagnostics table:

- When the diagnostics table is created, the owner of the target table becomes the owner of the diagnostics table.
- The owner of the diagnostics table automatically receives all table-level privileges on the diagnostics table, including the Alter and References privileges. However, the database server prevents the owner of the diagnostics table from altering the diagnostics table or adding a referential constraint to the diagnostics table.
- You can use the GRANT and REVOKE statements to modify the initial set of privileges on the diagnostics table.
- When you issue an INSERT, DELETE, or UPDATE statement on a target table that has a filtering mode unique index or constraint defined on it, you must have the Insert privilege on the violations and diagnostics tables.

  If you do not have the Insert privilege on the violations and diagnostics tables, the database server executes the INSERT, DELETE, or UPDATE statement on the target table provided that you have the necessary privileges on the target table. The database server does not return an error concerning the lack of insert permission on the violations and diagnostics tables unless an integrity violation is detected during the execution of the INSERT, DELETE, or UPDATE statement.

  Similarly, when you issue a SET statement to set a disabled constraint or disabled unique index to the enabled or filtering mode, and a violations and diagnostics table exist for the target table, you must have the Insert privilege on the violations and diagnostics tables.

If you do not have the Insert privilege on the violations and diagnostics tables, the database server executes the SET statement provided that you have the necessary privileges on the target table. The database server does not return an error concerning the lack of insert permission on the violations and diagnostics tables unless an integrity violation is detected during the execution of the SET statement.

- The grantor of the initial set of privileges on the diagnostics table is the same as the grantor of the privileges on the target table. For example, if the user **jenny** has been granted the Insert privilege on the target table by both the user **wayne** and the user **laurie**, the Insert privilege on the diagnostics table is granted to user **jenny** both by user **wayne** and by user **laurie**.
- Once a diagnostics table has been started for a target table, revoking a privilege on the target table from a user does not automatically revoke the same privilege on the diagnostics table from that user. Instead you must explicitly revoke the privilege on the diagnostics table from the user.
- If you have fragment-level privileges on the target table, you have the corresponding table-level privileges on the diagnostics table.

Example of Privileges on the Diagnostics Table

The following example illustrates how the initial set of privileges on a diagnostics table is derived from the current set of privileges on the target table.

For example, assume that there is a table called **cust_subset** and that this table consists of the following columns: **ssn** (customer's social security number), **fname** (customer's first name), **lname** (customer's last name), and **city** (city in which the customer lives).

The following set of privileges exists on the **cust_subset** table:

- User **alvin** is the owner of the table.
- User **barbara** has the Insert and Index privileges on the table. She also has the Select privilege on the **ssn** and **lname** columns.

- User **carrie** has the Update privilege on the **city** column. She also has the Select privilege on the **ssn** column.
- User **danny** has the Alter privilege on the table.

Now user **alvin** starts a violations table named **cust_subset_viols** and a diagnostics table named **cust_subset_diags** for the **cust_subset** table, as follows:

```
START VIOLATIONS TABLE FOR cust_subset
    USING cust_subset_viols, cust_subset_diags
```

The database server grants the following set of initial privileges on the **cust_subset_diags** diagnostics table:

- User **alvin** is the owner of the diagnostics table, so he has all table-level privileges on the table.
- User **barbara** has the Insert, Delete, Select, and Index privileges on the diagnostics table.
- User **carrie** has the Insert, Delete, Select, and Update privileges on the diagnostics table.
- User **danny** has no privileges on the diagnostics table.

Using the Diagnostics Table

For information on the relationship between the diagnostics table and the violations table, see "Relationship Between the Violations and Diagnostics Tables" on page 3-114.

The following issues concern the structure and use of the diagnostics table:

- The MAX ROWS clause of the START VIOLATIONS TABLE statement sets a limit on the number of rows that can be inserted into the diagnostics table when you execute a single statement, such as an INSERT or SET statement, on the target table.
- The MAX ROWS clause limits the number of rows only for operations in which the table functions as a diagnostics table.
- When a table functions as a diagnostics table, it cannot have triggers or constraints defined on it.

- When a table functions as a diagnostics table, users can create indexes on the table, even though the existence of an index affects performance. You cannot set unique indexes on the diagnostics table to the filtering object mode.
- If a target table has a violations and diagnostics table associated with it, dropping the target table in the cascade mode (the default mode) causes the violations and diagnostics tables to be dropped also. If the target table is dropped in the restricted mode, the existence of the violations and diagnostics tables causes the DROP TABLE statement to fail.
- Once a violations table is started for a target table, you cannot use the ALTER TABLE statement to add, modify, or drop columns in the target table, violations table, or diagnostics table. Before you can alter any of these tables, you must issue a STOP TABLE VIOLATIONS statement for the target table.
- The database server does not clear out the contents of the diagnostics table before or after it uses the diagnostics table during an Insert, Update, Delete, or Set operation.
- If the target table specified in the START VIOLATIONS TABLE statement is fragmented, the diagnostics table is fragmented with a round-robin strategy over the same dbspaces in which the target table is fragmented.

Example of a Diagnostics Table

To start a violations and diagnostics table for the target table named **stock** in the **stores7** demonstration database, enter the following statement:

```
START VIOLATIONS TABLE FOR stock
```

Because your START VIOLATIONS TABLE statement does not include a USING clause, the diagnostics table is named **stock_dia** by default. The **stock_dia** table includes the following columns:

```
informix_tupleid
objtype
objowner
objname
```

This list of columns shows an important difference between the diagnostics table and violations table for a target table. Whereas the violations table has a matching column for every column in the target table, the columns of the diagnostics table do not match any columns in the target table. The diagnostics table created by any START VIOLATIONS TABLE statement always has the same columns with the same column names and data types.

References

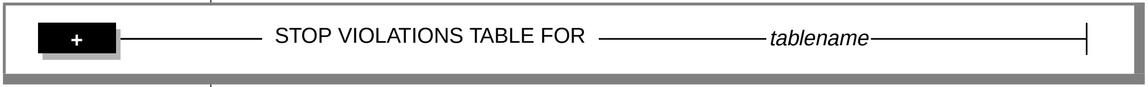
See the STOP VIOLATIONS TABLE and SET statements in this guide.

For information on the system catalog tables associated with the START VIOLATIONS TABLE statement, see the SYSOBJSTATE and SYSVIOLATIONS tables in this guide.

STOP VIOLATIONS TABLE

The STOP VIOLATIONS TABLE statement drops the association between a target table and the special violations and diagnostics tables.

Syntax



Element	Purpose	Restrictions	Syntax
table name	The name of the target table whose association with the violations and diagnostics table is to be dropped. There is no default value.	The target table must have a violations and diagnostics table associated with it before you can execute the statement. The target table must be a local table. For information on the privileges required to stop violations and diagnostics tables for a target table, see "Privileges Required for Stopping a Violations Table" on page 3-129.	Identifier segment, *Informix Guide to SQL: Syntax*

Usage

The STOP VIOLATIONS TABLE statement drops the association between the target table and the violations and diagnostics tables. After you issue this statement, the former violations and diagnostics tables continue to exist, but they no longer function as violations and diagnostics tables for the target table. They now have the status of regular database tables instead of violations and diagnostics tables for the target table. You must issue the DROP TABLE statement to explicitly drop these two tables.

When Insert, Delete, and Update operations cause data-integrity violations for rows of the target table, the nonconforming rows are no longer filtered to the former violations table, and diagnostics information about the data-integrity violations is not placed in the former diagnostics table.

Example of Stopping a Violations and Diagnostics Table

Assume that a target table named **cust_subset** has an associated violations table named **cust_subset_vio** and an associated diagnostics table named **cust_subset_dia**. To drop the association between the target table and the violations and diagnostics tables, enter the following statement:

```
STOP VIOLATIONS TABLE FOR cust_subset
```

Example of Dropping a Violations and Diagnostics Table

After you execute the STOP VIOLATIONS TABLE statement in the preceding example, the **cust_subset_vio** and **cust_subset_dia** tables continue to exist, but they are no longer associated with the **cust_subset** table. Instead they now have the status of regular database tables. To drop these two tables, enter the following statements:

```
DROP TABLE cust_subset_vio;
DROP TABLE cust_subset_dia;
```

Privileges Required for Stopping a Violations Table

To stop a violations and diagnostics table for a target table, you must meet one of the following requirements:

- You must have the DBA privilege on the database.
- You must be the owner of the target table and have the Resource privilege on the database.
- You must have the Alter privilege on the target table and the Resource privilege on the database.

References

See the SET and START VIOLATIONS TABLE statements in this guide.

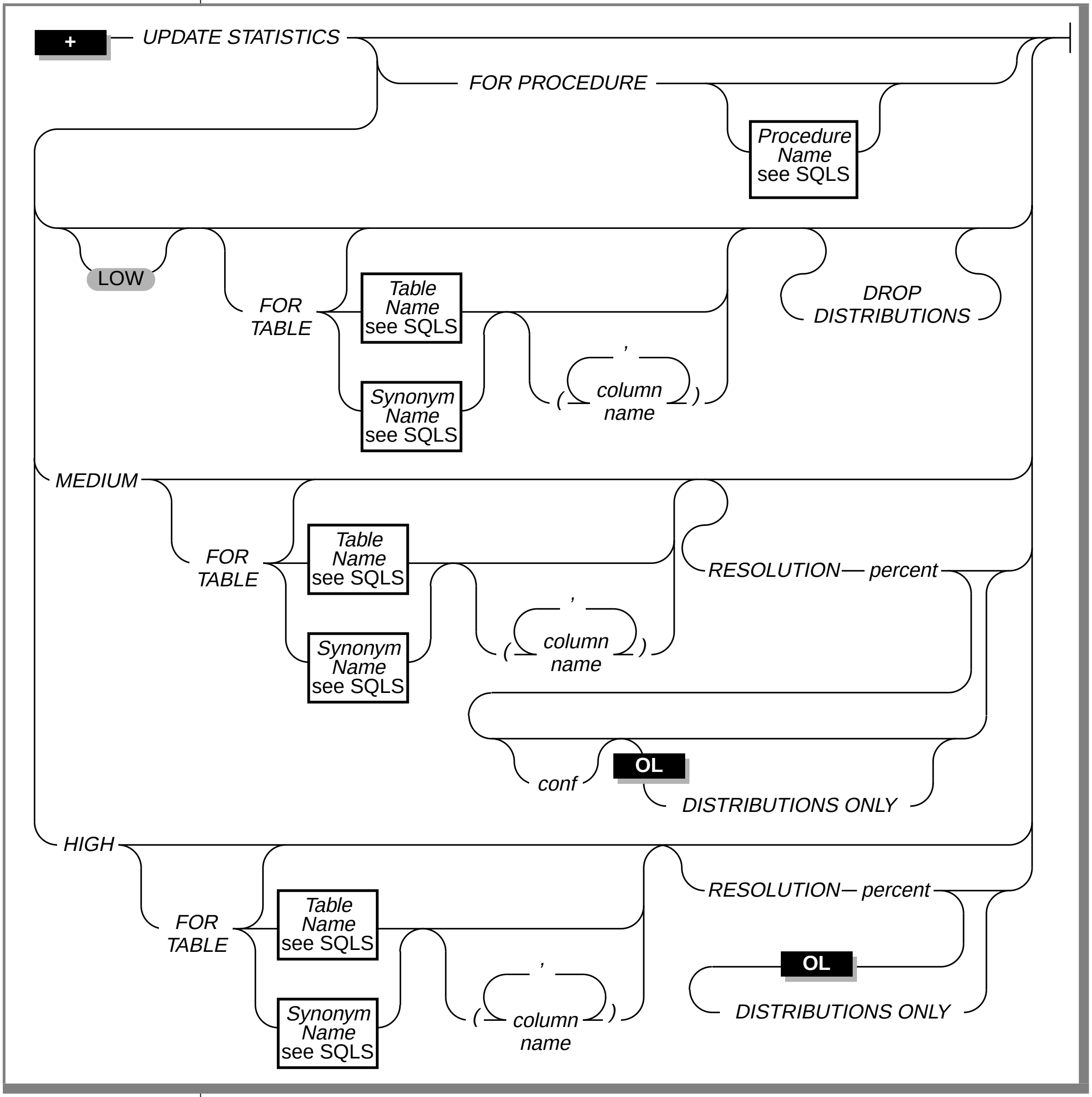
For information on the system catalog tables associated with the STOP VIOLATIONS TABLE statement, see the SYSOBJSTATE and SYSVIOLATIONS tables in this guide.

UPDATE STATISTICS

The UPDATE STATISTICS statement has the following changes in this release:

- The DISTRIBUTIONS ONLY option has been added to the MEDIUM and HIGH clauses.
- A new procedure is recommended for giving the optimizer the best possible information to use in determining the optimal execution path for queries.

Syntax


+
UPDATE STATISTICS
FOR PROCEDURE
Procedure Name see SQLS
LOW
FOR TABLE
Table Name see SQLS
Synonym Name see SQLS
,
(column name)
DROP DISTRIBUTIONS
MEDIUM
RESOLUTION— percent
conf
OL
DISTRIBUTIONS ONLY
HIGH

In this release, when you specify the DISTRIBUTIONS ONLY option with either the MEDIUM or HIGH clause, index information is not constructed for the specified tables or columns.

The DISTRIBUTIONS ONLY option is meaningful only if you are using OnLine.

Behavior in Previous Releases

In previous releases, the medium and high modes included the functionality of the low mode. Thus, when you ran the UPDATE STATISTICS statement in medium or high modes, the functionality of the low mode was performed automatically. This functionality consisted of constructing index information and table information for the specified objects.

Behavior in This Release

In this release, the functionality of the low mode is still performed when you run the UPDATE STATISTICS statement in medium or high modes. However, you can specify the DISTRIBUTIONS ONLY option with the MEDIUM or HIGH clauses to prevent the construction of index information for the specified objects.

Table information is still constructed for the specified objects when you specify the DISTRIBUTIONS ONLY option. The table information that is constructed includes the number of pages used, the number of rows, and fragment information.

The DISTRIBUTIONS ONLY option suppresses the construction of index information but not the construction of table information for the following reasons:

- The table information is required to construct accurate distributions.
- The construction of index information can take a considerable amount of time, but the construction of table information requires very little time and very few system resources.

Examples of UPDATE STATISTICS Statements

In the following example, the UPDATE STATISTICS statement gathers distributions information, index information, and table information for the **customer** table.

```
UPDATE STATISTICS MEDIUM FOR TABLE customer
```

However, in the following example, only distributions information and table information are gathered for the **customer** table. The DISTRIBUTIONS ONLY option prevents the construction of index information.

```
UPDATE STATISTICS MEDIUM FOR TABLE customer
    DISTRIBUTIONS ONLY
```

Procedure for Updating Statistics

Informix recommends the following procedure for giving the optimizer the best possible information while incurring the lowest performance penalty:

1. Run UPDATE STATISTICS in medium mode with the DISTRIBUTIONS ONLY option for each table. (If you are the database owner or DBA and you want to gather statistics for the entire database, you can do that with a single command instead.). The default parameters are sufficient unless the table is very large. In this case, use a resolution of 1.00 and a confidence level of 0.99.
2. Run UPDATE STATISTICS in high mode for all columns that head an index. For the fastest execution time of the UPDATE STATISTICS statement, you must execute one UPDATE STATISTICS statement in the high mode for each such column. In a future release, Informix will remove this limitation.
3. For each multicolumn index, run UPDATE STATISTICS in low mode for all its columns.

This procedure executes rapidly because it only constructs the index-information statistics once for each index.

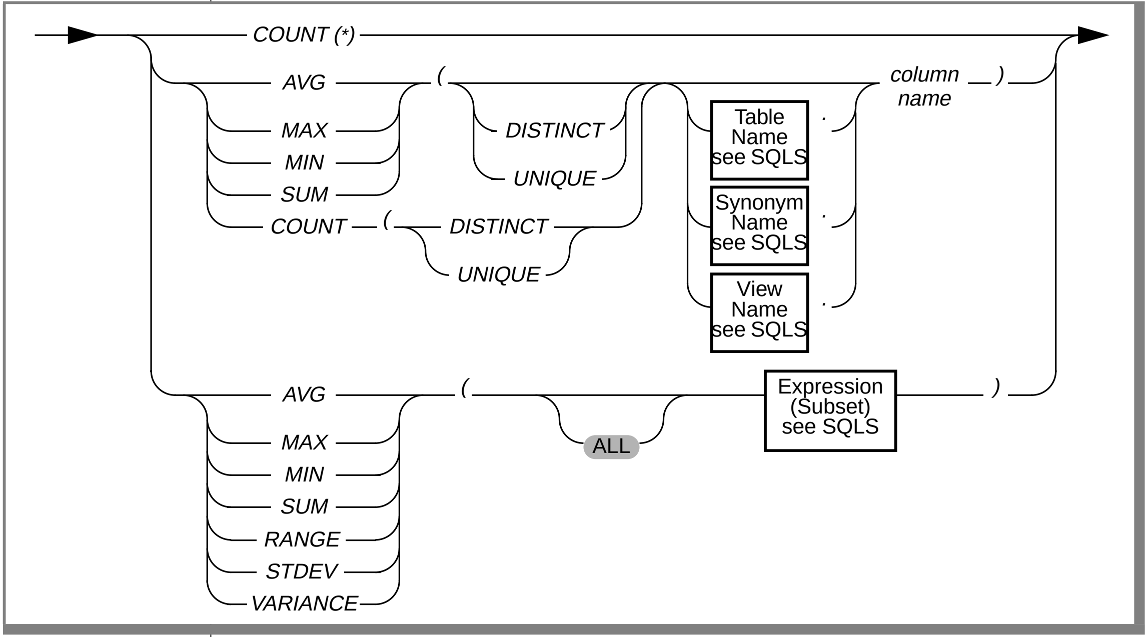
Changed SQL Segments

The following SQL segments are changed in this release:

- Aggregate Expression portion of the Expression segment
- Identifier segment

Aggregate Expression

An aggregate expression uses an aggregate function to summarize selected database data. The following diagram shows the aggregate functions, including three new functions: RANGE, STDEV, and VARIANCE.



Element	Purpose	Restrictions	Syntax
column name	The name of the column to which the specified aggregate function is applied. See "Summary of Aggregate Function Behavior" in the *Informix Guide to SQL: Syntax* for an example showing how each keyword that precedes *column name* performs a different calculation on the data values in *column name*.	If you specify an aggregate expression and one or more columns in the SELECT clause of a SELECT statement, you must put all the column names that are not used within the aggregate expression or a time expression in the GROUP BY clause. You cannot apply an aggregate function to a BYTE or TEXT column. See "Subset of Expressions Allowed in an Aggregate Expression" in the *Informix Guide to SQL: Syntax* for other general restrictions. For restrictions that depend on the keywords that precede *column name*, see the headings for individual keywords in the *Informix Guide to SQL: Syntax*.	Identifier, *Informix Guide to SQL: Syntax*

The *Informix Guide to SQL: Syntax* and the *Informix Guide to SQL: Tutorial* describe the general behavior and restrictions of aggregate expressions.

The following functions are added to the aggregate expression portion of the Expression segment:

- Range (RANGE)
- Standard Deviation (STDEV)
- Variance (VARIANCE)

Important: *All computations for the RANGE, STDEV, and VARIANCE functions are performed in 32-digit precision, which should be sufficient for many sets of input data. The computation, however, loses precision or returns wrong results when all of the input data values have 16 or more digits of precision.*

RANGE Function

The RANGE function computes the range for a sample of a population. It computes the difference between the maximum and the minimum values, as follows:

```
range(expr) = max(expr) - min(expr)
```

You can apply the RANGE function only to numeric columns. The following query finds the range of ages for a population:

```
SELECT RANGE(age) FROM u_pop
```

As with other aggregates, the RANGE function applies to the rows of a group when the query includes a GROUP BY clause, as shown in the following example:

```
SELECT RANGE(age) FROM u_pop
    GROUP BY birth
```

Nulls are ignored unless every value in the specified column is null. If every column value is null, the RANGE function returns a null for that column.

STDEV Function

The STDEV function computes the standard deviation for a sample of a population. It is the square root of the VARIANCE function.

You can apply the STDEV function only to numeric columns. The following query finds the standard deviation on a population:

```
SELECT STDEV(age) FROM u_pop WHERE u_pop.age > 0
```

As with the other aggregates, the STDEV function applies to the rows of a group when the query includes a GROUP BY clause, as shown in the following example:

```
SELECT STDEV(age) FROM u_pop
    GROUP BY birth
    WHERE STDEV(age) > 0
```

Nulls are ignored unless every value in the specified column is null. If every column value is null, the STDEV function returns a null for that column.

VARIANCE Function

The VARIANCE function returns the variance for a sample of values as an unbiased estimate of the variance of the population. It computes the following value:

```
(SUM(Xi**2) - SUM(Xi)**(2/N))/(N-1)
```

In this example, *Xi* is each value in the column and *N* is the total number of values in the column. You can apply the VARIANCE function only to numeric columns. The following query finds the variance on a population:

```
SELECT VARIANCE(age) FROM u_pop WHERE u_pop.age > 0
```

As with the other aggregates, the VARIANCE function applies to the rows of a group when the query includes a GROUP BY clause, as shown in the following example:

```
SELECT VARIANCE(age) FROM u_pop
    GROUP BY birth
    WHERE VARIANCE(age) > 0
```

Nulls are ignored unless every value in the specified column is null. If every column value is null, the VARIANCE function returns a null for that column.

Identifier Segment

The Identifier segment is changed in this release.

- Delimited identifiers are no longer flagged as noncompliant with the ANSI standard.
- When you use the AS keyword as a workaround for an ambiguous nondelimited identifier used as a column label, the AS keyword is no longer flagged as noncompliant with the ANSI standard.

Delimited Identifiers

When you specify the identifier segment within an SQL statement, the database server interprets the identifier as a delimited identifier if you set the **DELIMIDENT** environment variable and place double quotes around the identifier. Delimited identifiers allow you to specify names for database objects that are otherwise identical to SQL reserved words such as TABLE, WHERE, and DECLARE.

As a result of modifications to the ANSI flagger used by Informix products, delimited identifiers are no longer flagged as noncompliant with the ANSI SQL-92 Entry Level Standard.

The following SELECT statement uses the delimited identifier "table" to specify the name of the table from which data is retrieved. When you enter this statement in DB-Access or use it in an ESQL/C or ESQL/COBOL program, an ANSI noncompliance flag is longer be generated.

```
SELECT * FROM "table"
```

Workarounds That Use the Keyword AS

Although you can use almost any word as an SQL identifier, syntactic ambiguities can occur. Syntax errors are especially likely when you use a reserved word as a nondelimited identifier. One workaround is to use the AS keyword in the following situations:

- You can use the AS keyword before the column label in the SELECT clause of a SELECT statement. The column label is also known as the display label.
- You can use the AS keyword before the table alias in the FROM clause of a SELECT statement.

As a result of modifications to the ANSI flagger used by Informix products, the use of the AS keyword before a column label in SELECT statements is no longer flagged as noncompliant with the ANSI SQL-92 Entry Level Standard.

The following SELECT statement uses the AS keyword before a column label. When you enter this statement in DB-Access or use it in an ESQL/C or ESQL/COBOL program, an ANSI non-compliance flag is no longer generated.

```
SELECT call_dtime AS minute FROM cust_calls
```

The use of the AS keyword before a table alias in SELECT statements is still flagged to indicate ANSI noncompliance in this release.

The following SELECT statement uses the AS keyword before a table alias. When you enter this statement in DB-Access or use it in an ESQL/C or ESQL/COBOL program, an ANSI non-compliance flag is still generated in this release.

```
SELECT * FROM mytab AS order
```

New and Changed System Catalog Tables

This section describes new and changed system catalog tables.

The following system catalog tables are new in this release:

- **sysfragauth**
- **sysobjstate**
- **sysroleauth**
- **sysviolations**

The following system catalog tables are changed in this release:

- **syscoldepend**
- **sysconstraints**
- **sysusers**

SYSCOLDEPEND

The **syscoldepend** system catalog table is changed in this release. Formerly this system catalog table tracked only the table columns that are specified in check constraints. Now this system catalog table also tracks the table columns specified in not null constraints.

If a not null constraint is defined on one or more columns of a base table, one row is created in the **syscoldepend** table for each column involved in the constraint.

See the **syscoldepend** system catalog table in the *Informix Guide to SQL: Reference* for a description of the columns in this table.

SYSCONSTRAINTS

The **sysconstraints** system catalog table is changed in this release. Formerly the database server did not treat not null specifications as formal constraints, so not null specifications were not recorded in the **sysconstraints** table. Now the database server treats not null specifications as formal constraints, so not null specifications are recorded in the **sysconstraints** table.

How Not Null Constraints Are Named

Users can now assign names to not null specifications in CREATE TABLE and ALTER TABLE statements. For further information on naming constraints, see the CREATE TABLE and ALTER TABLE statements in this guide.

If the user does not assign a name to a not null specification, the database server creates a name for the not null specification. System-generated names for not null constraints follow the pattern for other internally generated constraint names. This pattern consists of the letter `n` followed by the table identifier, an underscore symbol, and a system-generated constraint identifier.

For example, if the table on which the constraint is defined has the table identifier **104**, the system-generated name for a not null constraint on this table might be **n104_7**.

New Code in the constrtype Column

A new code can now appear in the **constrtype** column of the **sysconstraints** table to identify a not null constraint. Not null constraints have the letter `N` in this column.

SYSFRAGAUTH

The **sysfragauth** system catalog table is new in this release. It stores information about the privileges granted on table fragments. The **sysfragauth** 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. This column identifies the table that contains the fragment named in the **fragment** column.
fragment	CHAR(18)	Name of dbspace where fragment is stored. This column identifies the fragment on which privileges are granted.

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

If a code in the **fragauth** column is lowercase, the grantee cannot grant the privilege to other users. If a code in the **fragauth** column is uppercase, the grantee can grant the privilege to other users.

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

The following example displays the fragment-level privileges for one base table, as they appear in the **sysfragauth** system catalog table. Note that grantee **ted** can grant the update, delete, and insert privileges to other users.

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

For information on the statements that grant and revoke fragment-level privileges on tables, see "GRANT FRAGMENT" on page 3-48 and "REVOKE FRAGMENT" on page 3-70.

SYSOBJSTATE

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

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

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

Column Name	Type	Explanation

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

owner	CHAR(8)	The owner of the database object
name	CHAR(18)	The name of the database object
tabid	INTEGER	The table identifier. This column identifies the table on which the database object is defined.

state	CHAR(1)	The current state (object mode) of the database object. This column has one of the following codes:
		D = Disabled
		E = Enabled
		F = Filtering, with no integrity-violation errors
		G = Filtering, with integrity-violation errors

A composite index for the **objtype**, **name**, **owner**, and **tabid** columns allows only unique values.

SYSROLEAUTH

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

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

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

SYSUSERS

The **sysusers** system catalog table is changed in this release. It 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 user or role
usertype	CHAR(1)	Specifies the privileges: D = DBA (all privileges) R = Resource (create permanent tables and indexes) C = Connect (work within existing tables) G = Role
priority	SMALLINT	Reserved for future use
password	CHAR(16)	Reserved for future use

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

SYSVIOLATIONS

The **sysviolations** system catalog table is new in this release. It stores information about the violations and diagnostics tables for base tables. Every table in the database that has a violations and diagnostics table associated with it has a corresponding row in the **sysviolations** table. The **sysviolations** database table has the following columns:

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

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

New and Changed Environment Variables

This section describes new and changed environment variables.

The following environment variables are new in this release:

- **INFORMIXOPCACHE**
- **INFORMIXSQLHOSTS**
- **NODEFDAC**

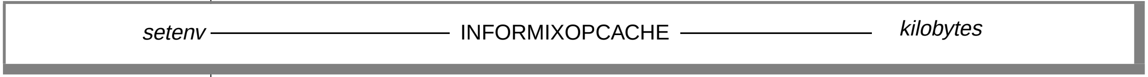
The following environment variables are changed in this release:

- **OPTCOMPIND**
- **PSORT_NPROCS**

INFORMIXOPCACHE

The **INFORMIXOPCACHE** environment variable allows you to specify the size of the memory cache for the Optical StageBlob area of the client application.

You set the **INFORMIXOPCACHE** environment variable by specifying the size of the memory cache in kilobytes. The size that you specify must be equal to or smaller than the size of the system-wide configuration parameter, OPCACHEMAX. If you do not set the **INFORMIXOPCACHE** environment variable, the default cache size is 128 kilobytes or the size specified in the configuration parameter OPCACHEMAX. The default for OPCACHEMAX is 128 kilobytes. If you set **INFORMIXOPCACHE** to a value of 0, INFORMIX-OnLine / Optical does not use the cache.



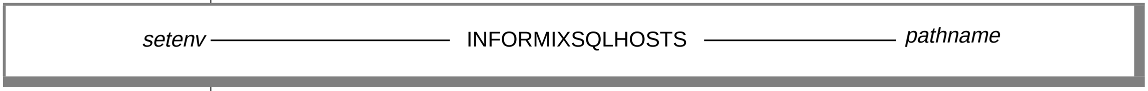
kilobytes	specifies the value you set for the optical memory cache. Must be equal to or smaller than the size of the system-wide configuration parameter, OPCACHEMAX.

INFORMIXSQLHOSTS

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

When the **INFORMIXSQLHOSTS** environment variable is set, the client or database server looks in the specified file for connectivity information. When the **INFORMIXSQLHOSTS** environment variable is not set, the client or database server looks in the **$INFORMIXDIR/etc/sqlhosts** file.

The file specified in the **INFORMIXSQLHOSTS** environment variable has the same format as the **$INFORMIXDIR/etc/sqlhosts** file. For a description of the **$INFORMIXDIR/etc/sqlhosts** file, see the *INFORMIX-OnLine Dynamic Server Administrator's Guide*.



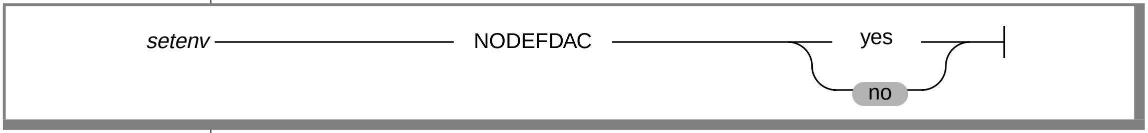
pathname	specifies the full pathname and filename of the file that contains connectivity information.

For example, to specify that the client or database server will look for connectivity information in the **mysqlhosts** file in the /**work**/**envt** directory, enter the following command:

```
setenv INFORMIXSQLHOSTS /work/envt/mysqlhosts
```

NODEFDAC

When set to `yes`, the **NODEFDAC** environment variable prevents default table privileges (Select, Insert, Update, and Delete) from being granted to PUBLIC when a new table is created in a database that is not ANSI-compliant.



yes	prevents default table privileges from being granted to PUBLIC on new tables in a database that is not ANSI-compliant. This setting also prevents the Execute privilege for a new stored procedure from being granted to PUBLIC when the stored procedure is created in owner mode.
no	allows default table privileges to be granted to PUBLIC. Also allows the Execute privilege on a new stored procedure to be granted to PUBLIC when the stored procedure is created in owner mode.

OPTCOMPIND

The **OPTCOMPIND** environment variable indicates the preferred join method. This environment variable has no default value. It is either set or not set. When the **OPTCOMPIND** environment variable is not set, OnLine uses an algorithm to determine the preferred join method. The algorithm that OnLine uses has changed in this release.

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 2*. Previously, the default value was 0.

PSORT_NPROCS

The **PSORT_NPROCS** environment variable has a new consideration concerning memory, changed information about default values for ordinary sorts, and new information about default values for attached indexes.

Checking Available Memory Before Sorts

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

Users sometimes perform large-scale sorts, such as index builds, without setting the **PDQPRIORITY** environment variable. If the **PDQPRIORITY** environment variable is not set, OnLine allocates the minimum amount of memory to sorts. This minimum memory is not enough to start even two sort threads. If you have not set the **PDQPRIORITY** environment variable, check the available memory before you perform a sort and make sure that you have enough memory for a large-scale sort.

Default Values for Ordinary Sorts

If the **PSORT_NPROCS** environment variable is set, OnLine uses the specified number of sort threads as an upper limit for ordinary sorts.

If **PSORT_NPROCS** is not set, parallel sorting does not take place. OnLine uses one thread for the sort.

If **PSORT_NPROCS** is set to zero (`0`), OnLine uses three threads for the sort.

Default Values for Attached Indexes

The default number of threads is different for attached indexes.

If the **PSORT_NPROCS** environment variable is set, you get the specified number of sort threads for each fragment of the index that is being built.

If the **PSORT_NPROCS** environment variable is not set, or if it is set to zero, you get two sort threads for each fragment of the index unless you have a single-CPU virtual processor. If you have a single-CPU virtual processor, you get one sort thread for each fragment of the index.

Changed Utilities

This section describes changed SQL utilities.

The following utilities are changed in this release:

- **dbexport**
- **dbload**
- **dbschema**

The dbexport Utility

The **dbexport** utility is changed in this release.

Support for Roles

The schema file created by the **dbexport** utility has been extended to contain the statement CREATE ROLE, the extension to the GRANT statement that permits roles to be granted to users and other roles, and the granting of privileges to roles.

Support for Object Modes and Violation Detection

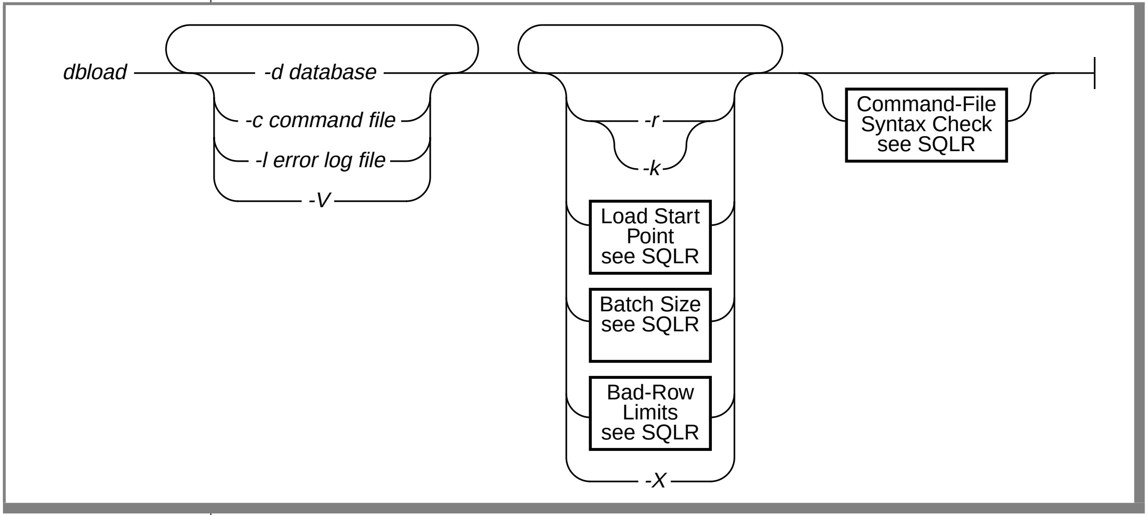
This release includes the following changes to the **dbexport** utility to support the new object modes and violation-detection functionality:

- In the output of **dbexport**, the names of not null constraints now appear after the not null specifications. This change is necessary because you can use the output of the utility as input to create another database. If the same names are not used for not null constraints in both databases, problems could result.
- The output of **dbexport** now shows the object mode of objects that are in the disabled state. These objects can be constraints, triggers, or indexes.
- The output of **dbexport** now shows the object mode of objects that are in the filtering state. These objects can be constraints or unique indexes.
- The output of **dbexport** now shows the violations and diagnostics tables associated with a base table (if violations and diagnostics tables have been started for the base table).

For more information about the new object modes and violation-detection functionality in this release, see the following statements in this guide: SET, START VIOLATIONS TABLE, and STOP VIOLATIONS TABLE.

The dbload Utility

The **dbload** utility is changed in this release. The **-k** option is new, and the **-r** option has a new restriction.



The **dbload** syntax includes a new **-k** option. In addition, the functionality of the existing **-r** option has been revised to accommodate the new **-k** option. Descriptions of these options follow. For explanations of the other **dbload** options, see the **dbload** utility in the *Informix Guide to SQL: Reference*.

-k	The **-k** option instructs **dbload** to lock the tables listed in the command file in exclusive mode during the load operation. If you do not specify the **-k** option, the tables specified in the command file are locked in shared mode. When tables are locked in shared mode, the database server still has to acquire exclusive row or page locks when it inserts rows into the table. When you specify the **-k** option, the database server places an exclusive lock on the entire table. The **-k** option increases performance for large loads because the database server does not have to acquire exclusive locks on rows or pages as it inserts rows during the load operation. Table locking in exclusive mode reduces the number of locks needed during the load operation but reduces concurrency. If you are planning to load a large number of rows, use exclusive table locking and perform the load during nonpeak hours. You cannot use the **-k** option with the **-r** option because the **-r** option specifies that no tables are locked during the load operation.
-r	The **-r** option instructs **dbload** not to lock the tables during loading, enabling other users to update data in the tables during the load operation. If you do not specify the **-r** option, the tables specified in the command file are locked in shared mode during loading so that other users cannot update data in the tables. However, you can override this default locking mode by specifying the **-k** option. The **-k** option instructs **dbload** to lock the tables in exclusive mode rather than shared mode during the load operation. You cannot use the **-r** option with the **-k** option because the **-r** option specifies that the tables are not locked during the load operation while the **-k** option specifies that the tables are locked in exclusive mode.

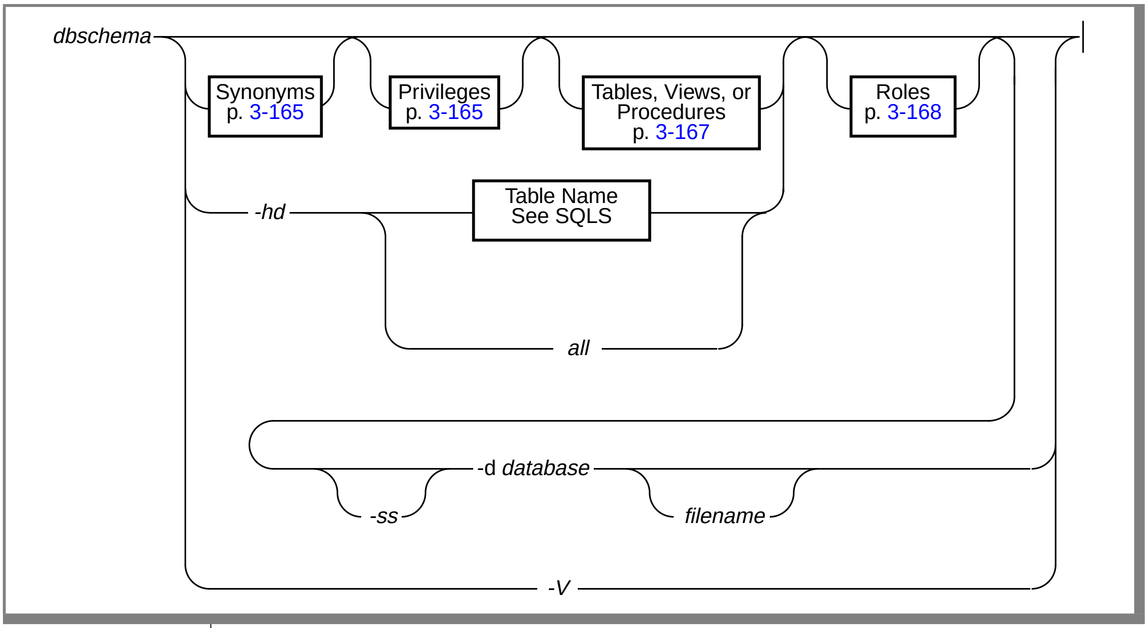
The dbschema Utility

The **dbschema** utility is changed in this release. The **dbschema** utility has been extended to display the SQL statements (the schema) required to replicate a role (as well as to replicate a database or a specific table, view, or procedure). In addition, the utility now supports object modes, violation detection, and fragment authorization. The entire description of the **dbschema** utility is included here.

Usage

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
- To display the SQL statements (the schema) required to replicate a role (as well as to replicate a database or a specific table, view, or procedure).



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 3-169.

-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 OnLine, 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 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," of the *Informix Guide to SQL: Reference*, **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:

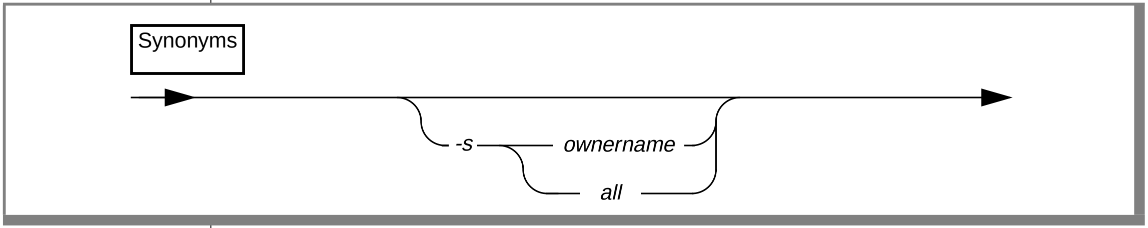
```
dbschema -d //finland/turku/stores7
```

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 owner of the original object owns the new object. 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.

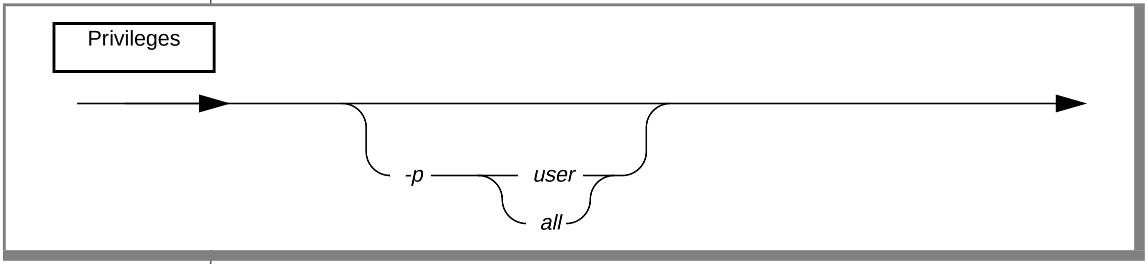
Output from **dbschema** that is executed with the specified option `-s alice` might appear, as shown in the following example:

```
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

The syntax for obtaining the privilege schema has been extended to display privilege information for roles.



-p *user*	displays the GRANT statements that grant privileges to a user, where *user* can be a user name or a role name. You can specify only one user or role.
-p all	displays the GRANT statements that grant privileges to all users for the table or view specified, or to all roles for the table specified.

You cannot specify a list of users or roles with the **-p** option. You can specify either one user or role, or all users and roles.

The following **dbschema** command and output show the privileges that were granted for the **calen** role:

```
sharky% dbschema -p calen -d stores7
```

```
DBSCHEMA Schema Utility       INFORMIX-SQL Version 7.10
Copyright (C) Informix Software, Inc., 1984-1995
Software Serial Number RDS#N000000

grant alter on table1 to 'calen'
```

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:

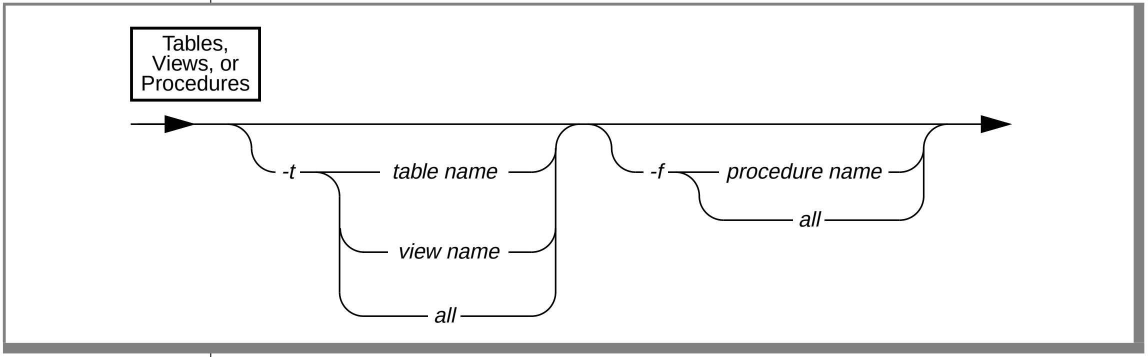
```
GRANT ALL ON 'tom'.customer TO 'claire' AS 'norma'
```

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 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

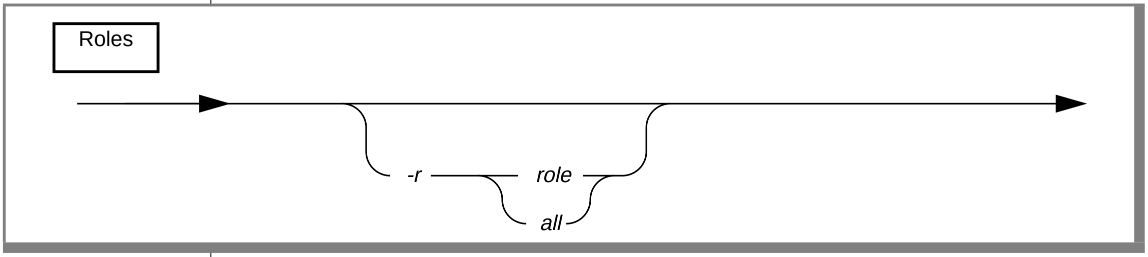


-f all	directs **dbschema** to limit the SQL statement output to the statements that are needed to replicate all procedures.
-f *procedure name*	directs **dbschema** to limit the SQL statement output to only the statements that are needed to replicate the specified procedure.
-t *table name*	directs **dbschema** to limit the SQL statement output to only the statements that are needed to replicate the specified table.
-t *view name*	directs **dbschema** to limit the SQL statement output to only the 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*.

Obtaining the Role Schema

The syntax of the **dbschema** utility has been extended to support roles.



-r *role*	directs **dbschema** to display the CREATE ROLE and GRANT statements that are needed to replicate and grant the specified role.
-r all	directs **dbschema** to display all CREATE ROLE and GRANT statements that are needed to replicate and grant all of the roles.

You cannot specify a list of users or roles with the **-r** option. You can specify either one role, or all roles.

The following **dbschema** command and output show that the role **calen** was created and was granted to **cathl**, **judith,** and **sallyc**:

```
sharky% dbschema -r calen -d stores7
```

```
DBSCHEMA Schema Utility       INFORMIX-SQL Version 7.10
Copyright (C) Informix Software, Inc., 1984-1995
Software Serial Number RDS#N000000
create role calen;

grant calen to cathl with grant option;
grant calen to judith ;
grant calen to sallyc ;
```

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 a fragmented table:

```
DBSCHEMA Schema Utility       INFORMIX-SQL Version 7.10
Copyright (C) Informix Software, Inc., 1984-1995
{ 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 table name, 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:

```
sharky% dbschema -hd invoices -d pubs_stores7
```

```
DBSCHEMA Schema Utility       INFORMIX-SQL Version 7.1
Copyright (C) Informix Software, Inc., 1984-1995
{

Distribution for cathl.invoices.invoice_num

Constructed on 03/10/1995

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 parentheses 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 parentheses 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 parentheses 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.

Support for Object Modes and Violation Detection

The following changes have been made to the output of **dbschema** to support the new object modes and violation-detection functionality in this release:

- The output now shows the names of not null constraints after the not null specifications. This change is necessary because you can use the output of the utility as input to create another database. If the same names were not used for not null constraints in both databases, problems could result.
- The output now shows the object mode of objects that are in the disabled state. These objects can be constraints, triggers, or indexes.
- The output now shows the object mode of objects that are in the filtering state. These objects can be constraints or unique indexes.
- The output now shows the violations and diagnostics tables that are associated with a base table (if violations and diagnostics tables have been started for the base table).

For more information about the new object modes and violation detection functionality in this release, see the following statements in this guide: SET, START VIOLATIONS TABLE, and STOP VIOLATIONS TABLE.

Support for Fragment Authorization

The output of **dbschema** has been changed to support the new fragment-authorization functionality in this release.

When you specify the **-p** or **-ss** option, the output now displays any GRANT FRAGMENT statements that are issued for a particular user or in the entire schema.

For more information about the new fragment authorization functionality in this release, see the following statements in this guide: GRANT FRAGMENT and REVOKE FRAGMENT.

Changes to the SQL Communications Area

This section describes changes to the SQL Communications Area (SQLCA).

The **sqlwarn** array within the SQLCA is changed in this release.

SQLWARN Array

This release includes changes to the SQLWARN array in the SQL Communications Area (SQLCA). For operations other than opening or connecting to a database, the **sqlwarn1** and **sqlwarn3** fields of this array are now set to `W` in the following situations:

- The **sqlwarn1** field is now set to `W` when a REVOKE statement with the ALL keyword does not revoke all seven table-level privileges.
- The **sqlwarn3** field is now set to `W` when a GRANT statement with the ALL keyword does not grant all seven table-level privileges.

The following list summarizes the complete behavior of the **sqlwarn1** and **sqlwarn3** fields for operations other than opening or connecting to a database:

sqlwarn1	This field is set to `W` when a column value is truncated as it is fetched into a host variable. *This field is also set to `W` when a REVOKE statement with the ALL keyword does not revoke all table-level privileges.*
sqlwarn3	This field is set to `W` when, on a SELECT or on opening a cursor, the number of items in the select list is not the same as the number of host variables given in the INTO clause to receive them. *This field is also set to `W` when a GRANT statement with the ALL keyword does not grant all table-level privileges.*

The italicized text in this list signifies behavior that is new in this release.

Chapter

ON-Archive Feature Enhancements

4

This chapter contains information about the enhancements for ON-Archive introduced in this release. The new and changed features simplify database administration.

Understanding ON-Archive Enhancements and Changes

ON-Archive is a recovery and backup system that enables you to back up data and subsequently restore it if your current data becomes corrupt or inaccessible. The causes of data corruption or loss can range from a program error to a disk crash to a disaster that damages the entire facility. ON-Archive enables you to recover data that you have lost due to mishaps. For information about ON-Archive, see the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*. The following sections describe the new features and changes to ON-Archive.

Features Added to ON-Archive

- Four new qualifiers, IMMEDIATE, NOIMMEDIATE, AUTOVOP, and NOAUTOVOP, are now added to the ARCHIVE, BACKUP, COPY/REQUEST, COPY/VSET, MODIFY/COMMAND, RETRIEVE/DBSPACESET, and RETRIEVE/LOGFILE commands.
- A new command, LIST/RECOVERY, is added to the **onarchive** utility. The LIST/RECOVERY command generates a report that provides recovery information.
- The **onautovop** utility can now execute a specific request.
- When you invoke the **onautovop** and **onarchive** utilities, the **oncatlgr** utility automatically executes.

- A sample script is now part of the ON-Archive product. When you write programs, you can use this sample script as a model for programs that catch log-full event alarms for automated backups of logical logs.
- When you use the **ondatartr** utility, you now have the option to proceed with, or cancel, an interruption of a logical restore when you press CTRL-C.
- You can add an ON-Archive activity log for the ARCHIVE, BACKUP, and RESTORE commands. This log allows you to log activities associated with those events.

Change to PRIVILEGE Parameter

The ON-Archive PRIVILEGE parameter, one of several ON-Archive security features, no longer supports OWNER mode.

Using New ON-Archive Qualifiers

The following new qualifiers are added to some of the **onarchive** utility commands:

- IMMEDIATE
- NOIMMEDIATE
- AUTOVOP
- NOAUTOVOP

Using the IMMEDIATE Qualifier

The IMMEDIATE qualifier saves time by allowing you to create and execute a request using a single command.

In earlier versions of ON-Archive, when you invoked the ARCHIVE, BACKUP, COPY/REQUEST, COPY/VSET, MODIFY/COMMAND, RETRIEVE/DBSPACESET, or RETRIEVE/LOGFILE commands, you created a request entry in the archive catalog for that command. Then, ON-Archive displayed the request ID associated with that request entry.

Now, when you specify IMMEDIATE, ON-Archive displays the request ID and immediately executes the request.

Example

The following example shows how to use the IMMEDIATE qualifier in the ARCHIVE command at the **onarchive** command-line prompt:

```
ARCHIVE/DBSPACESET=dbset1/VSET=arcvset/IMMEDIATE
```

The example shows how to archive of the **dbset1** dbspaceset and set the **arcvset** volume immediately.

Using the NOIMMEDIATE Qualifier

The NOIMMEDIATE qualifier is the default setting for the ARCHIVE, BACKUP, COPY/VSET, MODIFY/COMMAND, RETRIEVE/DBSPACESEST, or RETRIEVE/LOGFILE commands. If you do not specify the IMMEDIATE qualifier in those **onarchive** commands, the commands default to NOIMMEDIATE. The NOIMMEDIATE qualifier ensures that those commands function as in earlier versions.

Typically, you do not need to specify the NOIMMEDIATE qualifier. However, if you specify IMMEDIATE in a personal default file, the NOIMMEDIATE qualifier allows you to override the IMMEDIATE setting to ensure that the affected **onarchive** commands function as in earlier versions.

Example

The following example shows how to use the NOIMMEDIATE qualifier in the ARCHIVE command at the **onarchive** command-line prompt:

```
ARCHIVE/DBSPACESET=dbset1/VSET=arcvset/NOIMMEDIATE
```

The example shows how to archive the **dbset1** dbspaceset and the **arcvset** volume set with no immediate execution of the ARCHIVE request.

Using the AUTOVOP Qualifier

You use the AUTOVOP qualifier when you want to proceed to other tasks without waiting for a request to finish. The AUTOVOP qualifier simplifies the execution of requests by allowing you to take the following actions:

1. Use the **onautovop** utility to satisfy requests
2. Proceed to other tasks without waiting for a request to finish.

In earlier versions of ON-Archive, when you invoked the ARCHIVE, BACKUP, COPY/REQUEST, COPY/VSET, MODIFY/COMMAND, RETRIEVE/DBSPACESEST, or RETRIEVE/LOGFILE commands, you created a request entry in the archive catalog for that command. Then, ON-Archive displayed the request ID associated with that request entry.

Now, when you specify the AUTOVOP qualifier while invoking the ARCHIVE, BACKUP, COPY/REQUEST, COPY/VSET, MODIFY/COMMAND, RETRIEVE/DBSPACESET, or RETRIEVE/LOGFILE commands, **onarchive** calls the **onautovop** utility to satisfy that request when the request is executed.

Example

The following example shows how to use the AUTOVOP qualifier in the COPY/VSET command at the **onarchive** command-line prompt:

```
COPY/VSET=arcvset/DEST=tapevset/AUTOVOP
```

The example shows how to use the COPY command to copy the **arcvset** source volume set to the **tapevset** destination volume set and then release you to proceed to other tasks.

Using the NOAUTOVOP Qualifier

The NOAUTOVOP qualifier is the default setting for the ARCHIVE, BACKUP, COPY / VSET, MODIFY / COMMAND, RETRIEVE / DBSPACESET, or RETRIEVE / LOGFILE commands. If you do not specify the AUTOVOP qualifier in those **onarchive** commands, the commands default to NOAUTOVOP. The NOAUTOVOP qualifier ensures that those commands function as in earlier versions.

Typically, you do not need to specify the NOAUTOVOP qualifier. However, if you specify AUTOVOP in a personal default file, the NOAUTOVOP qualifier allows you to override the AUTOVOP setting to ensure that the affected **onarchive** commands function as in earlier versions.

Example

The following example shows how to use the NOAUTOVOP qualifier in the COPY / VSET command at the **onarchive** command-line prompt:

```
COPY/VSET=arcvset/DEST=tapevset/NOAUTOVOP
```

The COPY command copies the **arcvset** source volume set to the **tapevset** destination volume set. You cannot proceed to other tasks while that event occurs.

Using Command Qualifiers with Each Other

When you specify the AUTOVOP qualifier together with the NOIMMEDIATE qualifier, the **onautovop** utility performs the request when you issue the EXECUTE command for that request. Similarly, when you specify the AUTOVOP qualifier together with the IMMEDIATE qualifier, the **onautovop** utility performs the request immediately.

New ON-Archive Command

Use the LIST/RECOVERY command to list the volumes that are required to restore a full installation and dbspaces.

You can now use the LIST/RECOVERY command to provide a guide to restoring data. In earlier versions of ON-Archive, performing a cold restore of data required you to keep careful records of archives and backups and forced you to properly interpret those records during restore operations.

For information on the syntax and use of the LIST/RECOVERY command, see "The LIST/RECOVERY Command" on page 4-26.

Important: *Informix recommends that you use LIST/RECOVERY as part of the archive procedure in the event of a system crash.*

Using Enhanced ON-Archive Utilities

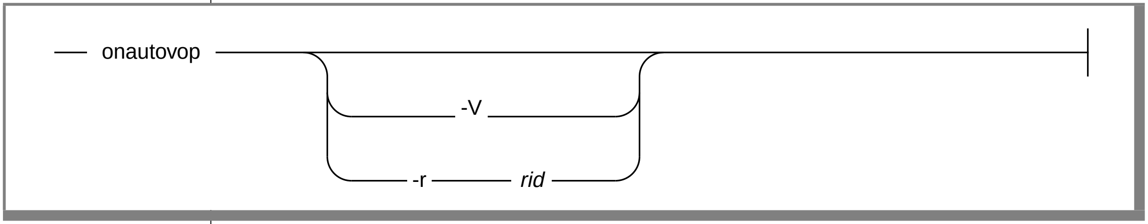
Several additional enhancements are now part of ON-Archive. Those enhancements exist to provide you with greater administrative control, accident prevention during restores, and improved monitoring capability.

The following list provides a detailed description of ON-Archive utility enhancements:

- Compared to earlier versions of ON-Archive, the **-r** option of the **onautovop** utility provides better control over command execution. Now, **onautovop** can execute a specific request when you use the **-r** option.
- Before you run **onarchive** or **onautovop**, you must first start the **oncatlgr** process. Now, when you invoke **onarchive** and **onautovop**, those utilities start the **oncatlgr** process for you.
- When you interrupt a logical restore by pressing CTRL-C, you must start over. Now, the **ondatartr** utility warns you when you press CTRL-C and allows you to cancel the interrupt.

Utility Enhancement for onautovop

When you use the **onautovop** utility, the new **-r** command-line qualifier allows you to execute a specific request. The **onautovop** utility executes only the specified request ID and then exits. The following diagram shows the syntax of the **onautovop** utility.



-r	executes the request specified by the *rid* qualifier.
rid	represents the request ID number.
-V	prints the version number and exits.

The following example shows how to use the **-r** option with **onautovop**:

```
onautovop -r 5
```

In the example, the value 5 represents a request.

Utility Enhancements That Automatically Start oncatlgr

Running the **onarchive** or **onautovop** utilities automatically starts the **oncatlgr** process.

Both **onarchive** or **onautovop** require a running **oncatlgr** process to operate. When either of those utilities start, they check for a running **oncatlgr** process. If that process is not running, either utility automatically starts an **oncatlgr** process.

Interrupt Enhancement to the ondatartr Utility

Interrupting a restore forces you to restart the entire process from the beginning. When you interrupt a logical restore by pressing CTRL-C, **ondatartr** now verifies whether you want to proceed with that interrupt. The verification prompt asks you if you want to interrupt **ondatartr** and provides you with a last chance to prevent the termination of a logical restore.

When you press the CTRL-C key sequence during a logical restore, the **ondatartr** utility displays the following prompt:

```
Interrupt of ondatartr during logical restore will require a
restart of the entire cold restore. Really interrupt the
restore? (y/n)
```

Type `Y` to terminate **ondatartr**. Type `N` to continue the logical restore without interruption.

See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for information on what to do if you press an incorrect key during a restore.

Automating Backups Using an Event Alarm Script

When a logical log changes state, an event alarm occurs. The new event-alarm feature allows you to automatically execute backups. This feature permits you to back up logical logs without using continuous backups. In addition, you are not required to monitor the state of logical logs to know when to start an automatic backup. Informix provides a sample script to help you understand this process.

Understanding the Sample Script

The event-alarm mechanism has the facility to call only one program. That program must handle *all* event alarms that the database administrator wants to capture. To help you deal with event alarms, Informix has provided a sample script called **logevent.sh**. You can find that sample script in the **$INFORMIXDIR/etc** directory where the other ON-Archive scripts are stored. You can use that script as a basis or model for automating a logical-log backup. You must edit **logevent.sh** before you can use it. The script has one limitation in that it does not know which volume sets you are using for log backups.

In the sample script, you can configure the following items:

- The percentage of full logs that trigger a logical-log backup
- The volume set used for the backup

When an event occurs, the database server can execute the script providing that you specify the full pathname of that script in the ONCONFIG file. When you place that script in the **$INFORMIXDIR /etc** directory, you gain an enhanced ability to administer the script.

To use an event-alarm script, perform the following steps:

1. Create a new script or configure the sample script for the correct environment.
2. Place the script in the **$INFORMIXDIR/etc** directory or any directory that you choose.
3. Specify the pathname of the script in the ONCONFIG file by using the ALARMPROGRAM configuration parameter. The pathname that you specify for the ALARMPROGRAM parameter must contain the full pathname of the sample script.

Adding an ON-Archive Activity Log to Log Archive Events

ON-Archive produces the ON-Archive activity log that contains a history of all the ON-Archive actions. You can find an example of the ON-Archive activity log in Figure 4-1 on page 4-13.

Before you start using ON-Archive, produce the ON-Archive activity log using the following steps:

1. The ON-Archive activity log can be large. Make sure the ON-Archive activity log resides in a directory that is appropriate for that file.
2. Make sure appropriate file permissions exist so that you can use the directory where the activity log resides.
3. Edit the **config.arc** configuration file. If you are not using the **config.arc** file, edit the file specified by the ARC_CONFIG environment variable.
4. Insert an ACTIVITYLOG parameter into the specified file.
5. Make sure the ACTIVITYLOG parameter holds the full pathname of the ON-Archive activity log file. The following example shows how to set the ACTIVITYLOG parameter.

   ```
   ACTIVITYLOG = /usr/informix/etc/onarchive.log
   ```

The ACTIVITYLOG configuration parameter points to the ON-Archive activity log. The ON-Archive activity log records all significant events generated by ON-Archive. The information in the ON-Archive activity log includes the beginning and end of all ARCHIVE, BACKUP, COPY, and RESTORE events.

In addition, the ON-Archive activity log contains information on dbspacesets, volume sets, volumes, and requests. The file records only commands that change the state of ON-Archive. The file does not record usage or syntax errors.

Using the ON-Archive Activity Log

Because several ON-Archive processes can run in parallel (for example, **onarchive** and **onautovop** can run simultaneously), the ON-Archive activity log conforms to a format that allows you to trace the history of the ON-Archive catalogs and follow different threads of execution.

Figure 4-1
Example of data within an ON-Archive activity log

```
Apr 19 1994 09:30 #00000000# <8734> onarchive (informix) defined dbs1and2:
define/dbspaceset=dbs1and2/dbspaces=(dbs1, dbs2)
Apr 19 1994 09:35 #00000000# <8734> onarchive (informix) defined vset1: define/vset=vset1/
access=5/device_type=tape/class=system/driver=tape
Apr 19 1994 09:37 #00000000# <9762> onarchive (informix) vset1:0001: define/volume/vset-vset1
Apr 19 1994 10:01 #00000000# <9762> onarchive (informix) created request #000000005#:
archive/dbspaceset=dbs1and2
Apr 19 1994 10:02 #00000005# <9762> onarchive (informix) begin to archive to vset1
            10:04 #00000005# <9762> processing dbs1 to vset1:0001
Apr 19 1994 10:04 #00000000# <9762> onarchive (informix) defined vset1:0002: define/volume/vset=vset1
            10:06 #00000005# <9762> processing dbs1 to vset1:000210:09 #00000005# <9762> processing dbs2
to vset1:0002
            10:22 #00000005# <9762> failure detected: ARC-01026E Cannot close the file on tape
Apr 19 1994 10:25 #00000005# <9762> end archive: FAILED
Apr 19 1994 10:30 #00000000# <9764> onarchive (root) created request #000000006#: backup
continuous/immediate/autovop
Apr 19 1994 10:30 #00000006# <9765> onautovop (root) begin continuous backup
            10:35 #00000006# <9765> processing logfile1 to vsetlogs:0001
            10:40 #00000006# <9765> processing logfile2 to vsetlogs:0001
            10:46 #00000006# <9765> processing logfile2 to vsetlogs:0002
Apr 19 1994 11:05 #00000000# <863> ondatartr (root) begin retrieve
            11:06 #00000000# <863> processing logfile3 to ONDATARTRLOG:0001 /dev/rst0
            11:09 #00000000# <863> retrieved dbs1 #00000005# from vset1:0001,000
            11:15 #00000000# <863> retrieved dbs2 #00000005# from vset1:0002
            11:15 #00000000# <863> retrieved logfile1#00000006# from vsetlogs:000
            11:17 #00000000# <863> retrieved logfile2 #00000006# from vsetlogs:0001,0002
            11:20 #00000000# <863> retrieved logfile3 #00000863# from ONDATARTRLOG:0001
```

Figure 4-1 on page 4-13 shows several significant features:

- Date and time printed for the following events:
  - Discrete events

    ```
    Apr 19 1994 09:30 #00000000# <8734> onarchive (informix)
    defined dbsland2: define/dbspaceset=dbsland2/dbspaces=(dbs1, dbs2)
    ```
  - Begin marks for events that occur over a time interval

    ```
    10:30 #00000006# <9765> onautovop (root) begin continuous backup
    ```
  - End marks for events that occur over a time interval

    ```
    10:25 #00000005# <9762> end archive: FAILED
    ```
- The name of the program that generated the event
- The user ID of the person who executed the program that generated the event

  ```
  09:37 #00000000# <9762> onarchive (informix) vset1:0001:
  define/volume/vset-vset1
  ```
- Request and process IDs printed after the timestamps, which show all events appropriate to ON-Archive activity

  Hash marks (#) enclose the request ID. Greater than and less than signs (><) enclose the process ID. This information allows you to search easily for the sequence of events for a particular process, a particular request, or both.

  ```
  10:04 #00000005# <9762> processing dbs1 to vset1:0001
  ```
- The entire command issued to create, modify, or delete an ON-Archive object

  This information allows for easy re-creation of the ON-Archive catalogs and helps you understand what happens during execution events.

  ```
  10:30 #00000000# <9764> onarchive (root) created request
  #000000006#: backup continuous/immediate/autovop
  ```

- Events that take time to complete, such as ARCHIVE, BACKUP, COPY, and RETRIEVE.

  Those events are bracketed by lines that contain the request ID and the words *begin* and *end*. The *end* events include the status of the request and are preceded by error messages when failure occurs.

  ```
  10:02 #00000005# <9762> onarchive (informix) begin to archive to vset1
  10:04 #00000005# <9762> processing dbs1 to vset1:0001
  Apr 19 1994 10:04 #00000000# <9762> onarchive (informix) defined vset1:0002:
  define/volume/vset=vset1
  10:06 #00000005# <9762> processing dbs1 to vset1:000210:09 #00000005#
  <9762> processing dbs2 to vset1:0002
  10:22 #00000005" <9762> failure detected: ARC-01026E Cannot close the file
  on tape
  Apr 19 1994 10:25 #00000005# <9762> end archive: FAILED
  ```

- All significant changes in state during the execution of ARCHIVE, BACKUP, COPY, and RETRIEVE events.

  These events include changing tapes or dbspaces during an archive or restore.

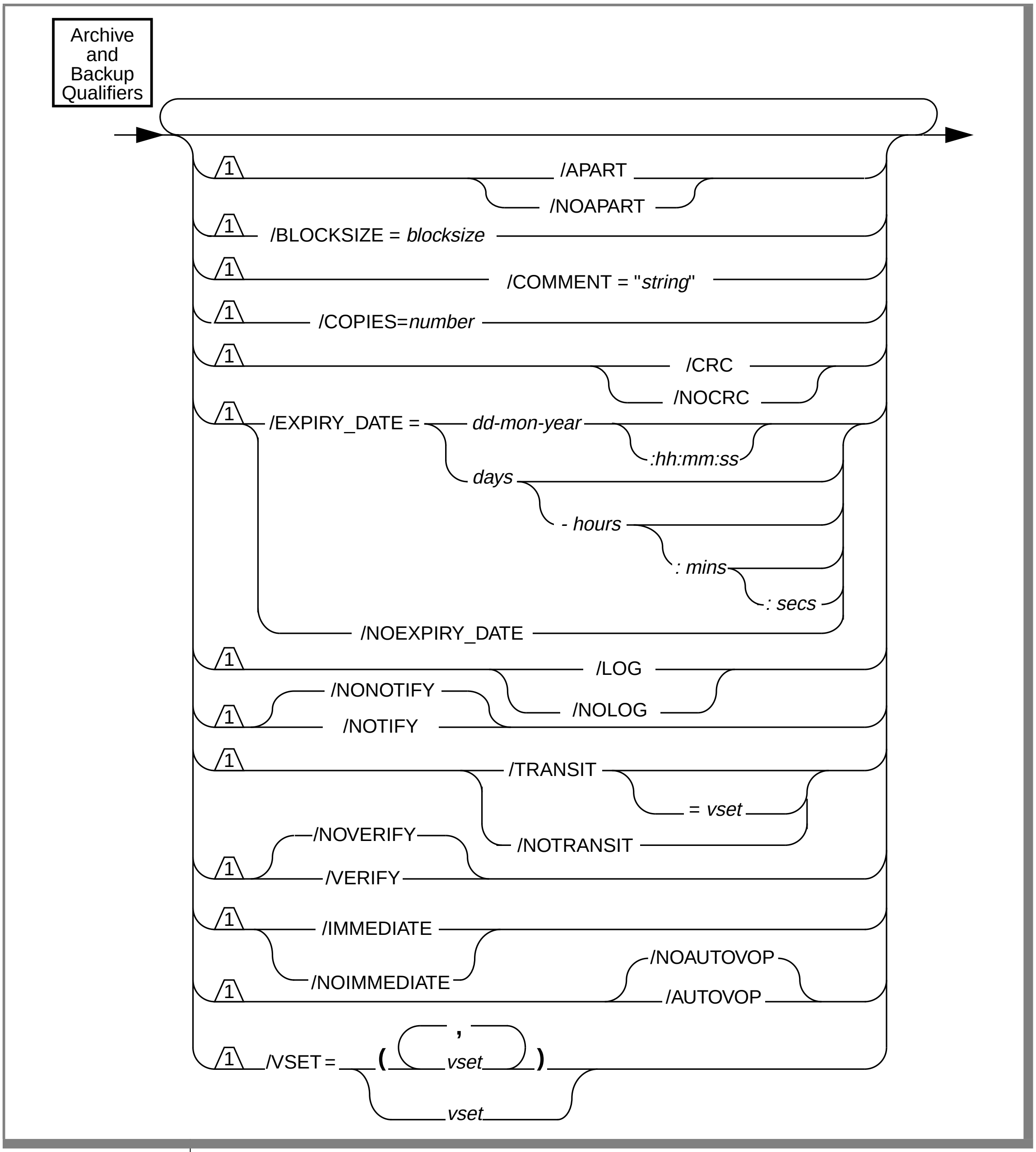
```
10:04 #00000005# <9762> processing dbs1 to vset1:0001
```

Change to the PRIVILEGE Configuration Parameter

You set ON-Archive configuration parameters in the **config.arc** file. In the current version of ON-Archive, you cannot use the OWNER option of the PRIVILEGE parameter. For more information on PRIVILEGE parameter modes, see the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.

ARCHIVE and BACKUP Qualifiers

This section describes qualifiers that were referenced in the ARCHIVE and BACKUP syntax diagrams for some commands. See the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for those diagrams. The new qualifiers are AUTOVOP, NOAUTOVOP, IMMEDIATE, and NOIMMEDIATE.


Archive and Backup Qualifiers
1
/APART
/NOAPART
/BLOCKSIZE = blocksize
/COMMENT = "string"
/COPIES=number
/CRC
/NOCRC
/EXPIRY_DATE =
dd-mon-year
:hh:mm:ss
days
- hours
: mins
: secs
/NOEXPIRY_DATE
/LOG
/NOLOG
/NONOTIFY
/NOTIFY
/TRANSIT
= vset
/NOTRANSIT
/NOVERIFY
/VERIFY
/IMMEDIATE
/NOIMMEDIATE
/NOAUTOVOP
/AUTOVOP
/VSET =
(
,
vset
)
vset

APART	specifies that the save set must reside on a separate volume from other save sets. That is, the save set is written to a blank volume, and no other save sets are written to the volume. Whenever you write to a remote tape device, you must use the APART qualifier. See "Keeping Archives on Separate Volumes" in Chapter 5 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.
NOAPART	indicates that the save set can be stored on volumes with other save sets. It is the default.
AUTOVOP	specifies when you want to proceed to other tasks without waiting for a request to finish.
NOAUTOVOP	specifies the default setting for ARCHIVE and BACKUP qualifiers. Allows you to override an AUTOVOP default setting, specified in a personal default file, to ensure that the ARCHIVE and BACKUP qualifiers function as in earlier versions.
BLOCKSIZE	specifies the block size in bytes when writing to tape volumes or, with NB_DISK_SPACE_EXTENT, the size of disk-space allocations when writing to disk. See "Disk-Space Allocation During Concurrent Operations" in Chapter 2 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*. The internal default value is 64 kilobytes as specified in the operator default file. See "Where Qualifier Default Values Are Specified" in Chapter 4 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*. *blocksize* is an integer from 8197 to 65,024 specifying the block size in bytes.
COMMENT	specifies a text string containing a comment about the archive or backup operation. The comment is stored with the request in the ON-Archive catalog. *string* is a text string containing a comment about the archive. The string cannot exceed 80 characters.

COPIES	specifies the number of copies (including the original) that must be made of the save set created by the request. Each copy is created on a different vset for greater security. See "Creating Multiple Copies of an Archive" in Chapter 5 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.
	number is an integer from 1 to 5. You must specify the same number of vsets with the VSET qualifier.
CRC	adds a cyclic-redundancy check (CRC) field at the end of each data block during the backup. This check ensures that the data on the tape is still valid at the time of the restore.
NOCRC	bypasses the operation of writing the CRC field at the end of each data block during the backup. It is the default.
EXPIRY_DATE	specifies an expiration date for a request. The **onautovop** utility removes all information regarding the request from the ON-Archive catalog when the operating system date equals (or is greater than) the expiration date. See "Specifying When Archive Data Is Obsolete" in Chapter 5 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.
	You can also remove specific requests manually (and thus render save sets obsolete) using the REMOVE/REQUEST command shown in Chapter 8 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.

dd	is an integer between 1 and 31 representing a specific day in a month.
mon	is a three-letter abbreviation for a month.
year	is an integer representing a year.
hh	is an integer representing the hour.
mm	is an integer representing the minute.
ss	is an integer representing the seconds in a minute.

	days	is an integer between 0 and 9999 specifying a number of days. The value is added to the request-creation date (not the execution date) to determine the expiration date.
	hours	is an integer between 0 and 24 specifying a number of hours.
	mins	is an integer between 0 and 60 specifying a number of minutes.
	secs	is an integer between 0 and 59 specifying a number of seconds.
NOEXPIRY_DATE	specifies that no expiration date must be set. It is the default.	
IMMEDIATE	simplifies the process of creating and executing a request. The IMMEDIATE qualifier saves time by allowing you to create and execute a request using a single command. It displays the request ID associated with that request entry.	
NOIMMEDIATE	specifies the default setting for ARCHIVE and BACKUP qualifiers. Allows you to override an IMMEDIATE default setting, specified in a personal default file, to ensure that the ARCHIVE and BACKUP qualifiers function as in earlier versions.	
LOG	specifies that a log file must be generated for this request. The log file is created in the directory where the **onarchive** or **onautovop** command was started. The log file is named **arc***rid***.log**, where *rid* is the archive or backup request id.	
NOLOG	specifies that a log file must not be generated for this request. It is the default.	
NOTIFY	specifies that after the request executes, electronic mail is sent to the user who created the request.	
NONOTIFY	specifies that no mail is sent after the request executes. It is the default.	

TRANSIT	specifies whether a transit vset must be used. See "What Is a Transit Volume Set?" in Chapter 2 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*. If TRANSIT is used without any parameters, ON-Archive selects a transit vset. It searches first through the user's transit vsets for an available transit vset. When none is available, ON-Archive then searches the system-transit vsets. *vset* is the name of a transit vset.
NOTRANSIT	specifies that a transit vset must not be used. It is the default.
VERIFY	instructs ON-Archive to reread each save set after writing it to tape. Rereading each save set ensures that the data on the tape can later be restored.
NOVERIFY	instructs ON-Archive not to reread each save set after writing it to the tape. NOVERIFY is the default.
VSET	specifies the vset(s) to use for this operation. No default value exists. When a vset is not specified, one is selected automatically. See "How On-Archive Selects Volume Sets, Devices, and Volumes" in Chapter 4 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*. More than one vset is usually only specified when the COPIES qualifier is used. *vset* is the name of a vset.

ON-Archive Syntax Enhancements

This section contains syntax diagrams and descriptions for commands now enhanced to support new qualifiers. The following list shows the enhanced commands:

- COPY / VSET and COPY / VSET / REQUEST
- MODIFY / COMMAND
- RETRIEVE / DBSPACESET
- RETRIEVE / LOGFILE

This section also contains the syntax diagram and description for a new command called LIST / RECOVERY.

The COPY/VSET and COPY/VSET/REQUEST Commands

The COPY command creates a new request that, when executed, copies the contents of one vset (or volume) onto another vset. Both the source and the destination vsets must be previously defined.

The COPY command is a powerful storage-management tool. It can be used for the following tasks:

- Media refreshment

  Copy an entire vset into another with similar definition, then delete the old one.
- Reclaiming storage space

  Copy only unremoved and unexpired save sets from one vset into another with a similar definition, then delete the old one.
- Media conversion

  Copy a vset defined on one type of media into another vset defined on another type of media (for example, converting a volume from disk to tape).

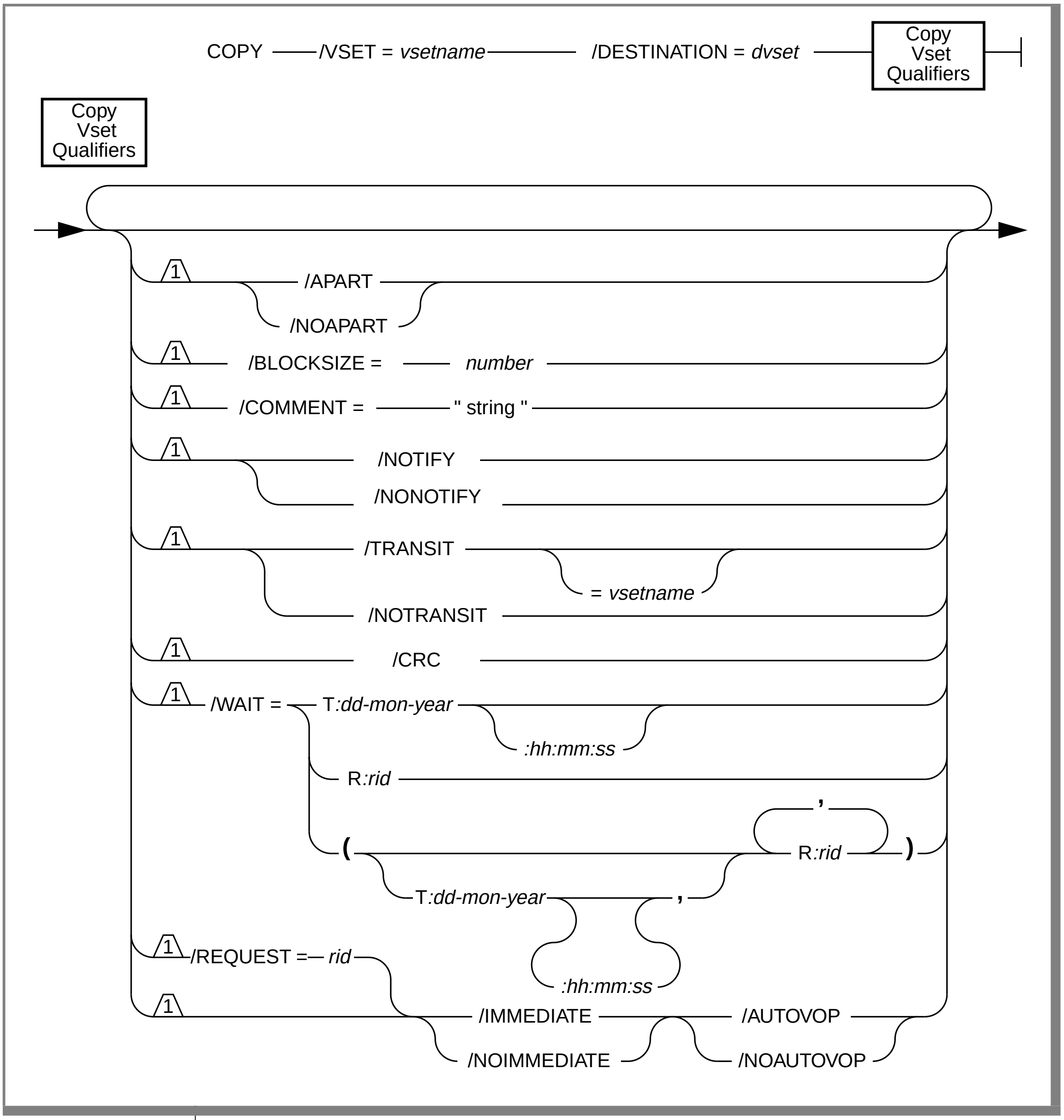
- Creating an extra copy of a specific save set

  Copy a save set from one vset to another.
- Data separation

  Copy all save sets that belong to one user from one vset into another.

The following restrictions apply to the COPY command:

- To copy a save set, you must have access to the volume on which the save set is stored, and to both source and destination vsets.
- The save sets selected from the input vset must not already exist in the destination vset.
- If ON-Archive is running in OPERATOR privilege mode, users **root** and **informix** can copy save sets, even if they did not create the request that created the save set.
- If ON-Archive is running in GROUP privilege mode, users in the **super_archive**, **informix**, or **root** groups can copy save sets, even if they did not create the request that created the save set.
- Only **informix** or **root** can copy an entire vset, regardless of the privilege mode in which ON-Archive is running.
- If the COPY request copies an entire vset and, therefore, generates child requests to copy the individual save sets, the child requests do not execute automatically; you must execute them. See "REQUEST" in Chapter 8 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for more information. The new COPY/VSET qualifiers are AUTOVOP, NOAUTOVOP, IMMEDIATE, and NOIMMEDIATE.


COPY
/VSET = vsetname
/DESTINATION = dvset
Copy Vset Qualifiers
Copy Vset Qualifiers
1
/APART
/NOAPART
/BLOCKSIZE =
number
/COMMENT =
" string "
/NOTIFY
/NONOTIFY
/TRANSIT
= vsetname
/NOTRANSIT
/CRC
/WAIT =
T:dd-mon-year
:hh:mm:ss
R:rid
(
T:dd-mon-year
,
:hh:mm:ss
,
R:rid
)
/REQUEST = rid
/IMMEDIATE
/NOIMMEDIATE
/AUTOVOP
/NOAUTOVOP

APART	specifies that the save set must be kept on a volume separate from other save sets (that is, the save set is written to a blank volume and no other save sets are written to that volume). Whenever you write to a remote tape device, you must use the APART qualifier.
NOAPART	indicates that the save set can be stored on volumes with other save sets. It is the default.
AUTOVOP	specifies when you want to proceed to other tasks without waiting for a request to finish.
NOAUTOVOP	specifies the default setting for the COPY/VSET and COPY/VSET/REQUEST commands. Allows you to override an AUTOVOP default setting, specified in a personal default file, to ensure that the COPY/VSET and COPY/VSET/REQUEST commands function as in earlier versions.
BLOCKSIZE	specifies the block size in bytes when writing to tape volumes or, with NB_DISK_SPACE_EXTENT, the size of disk space allocations when writing to disk. See "Disk-Space Allocation During Concurrent Operations" in Chapter 2 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.
	The internal default value is 64 kilobytes as specified in the operator default file. See "Where Qualifier Default Values Are Specified" in Chapter 4 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.
	number is an integer specifying the block size in bytes. The valid block-size range is from 8197 to 65,024.
COMMENT	specifies a text string containing a comment about the copy operation. The comment is stored with the save set in the ON-Archive catalog.
	string is a text string. It cannot exceed 80 characters.
CRC	instructs ON-Archive to add a cyclic-redundancy check (CRC) at the end of each save-set block.
DESTINATION	specifies the name of a vset to copy to.
	dvset is the name of an existing vset.

IMMEDIATE	simplifies the process of creating and executing a request. The IMMEDIATE qualifier saves time by allowing you to create and execute a request using a single command. Displays the request ID associated with that request entry.
NOIMMEDIATE	specifies the default setting for the COPY/VSET and COPY/VSET/REQUEST commands. Allows you to override an IMMEDIATE default setting, specified in a personal default file, to ensure that the COPY/VSET and COPY/VSET/REQUEST commands function as in earlier versions.
NOTIFY	specifies that after the request executes, electronic mail is sent to the user who created the request.
NONOTIFY	specifies that no mail is sent after the request executes. It is the default.
REQUEST	specifies that only save sets created by the specified request are to be copied. *rid* is an existing request id. If you do not use the REQUEST qualifier to specify particular save sets, the entire vset is copied to the destination vset. ON-Archive does this by generating a new child request for each save set in the vset to copy it to the destination vset. Those requests must then be executed.
TRANSIT	controls whether a transit vset must be used. See "What Is a Transit Volume Set?" in Chapter 2 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*. If TRANSIT is used without any parameters, ON-Archive selects a transit vset. It searches first through the user's transit vsets for an available transit vset. When no transit vsets are available, ON-Archive then searches the system-transit vsets. See the CLASS qualifier for the DEFINE/VSET command in Chapter 4 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for more information on system and user vsets. A transit vset is necessary for COPY when the source and destination vset require the same device type, and only one device of that type is available. *vsetname* is the name of a transit vset.

NOTRANSIT	specifies that a transit vset must not be used. It is the default.
VSET	specifies the name of a vset to copy from. *vsetname* is the name of an existing vset.
WAIT	controls whether a request must wait for certain events to occur before it can start. See “Wait and Repeat Qualifiers” in Chapter 8 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide.*

COPY/VSET Example

The following example creates a request to copy vset **may93** to vset **may93bkp**:

```
Onarchive> COPY/VSET=may93/DESTINATION=may93bkp
```

```
Request 00000041 registered in the catalog.
```

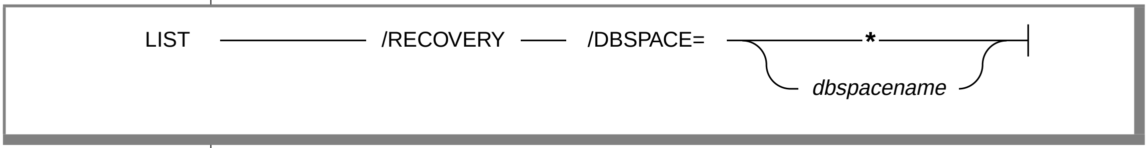
The LIST/RECOVERY Command

The LIST/RECOVERY command improves the data-restoration process and generates a report that displays data-restoration information. The LIST/RECOVERY command provides a simple mechanism for printing or saving. In addition, the LIST/RECOVERY command generates a report that does the following tasks:

- Sorts volumes in the mount order required for restore
- Provides an indication of the volumes that contain critical data
- Serves as a guide during cold restores to make OnLine available as quickly and easily as possible.

The LIST/RECOVERY command sorts the data in the order required to restore the data.When you specify a dbspace, or a list of dbspaces, LIST/RECOVERY displays only the data required to perform a warm restore of the dbspaces.

When you run ON-Archive in OPERATOR mode, you can list the recovery report only if you log in as **informix** or **root**. When you run ON-Archive in GROUP mode, only users in the **super_archive**, **informix**, or **root** groups can generate a LIST/RECOVERY report.



*	is a wildcard character that represents all dbspace names.
DBSPACE	instructs the LIST command to display dbspace information.
dbspacename	is the name of a specific dbspace.
RECOVERY	instructs the LIST command to display data-restoration information.

Execute the LIST/RECOVERY command after the following events:

- Archiving the entire installation
- Archiving critical and noncritical dbspaces separately
- Archiving some critical and noncritical dbspaces together, but without archiving the entire installation
- Archiving level 0, changing data, then archiving again at levels 1 and 2
- Archiving, changing data, and backing up the logical logs
- Archiving successfully, trying another archive that fails, then archiving successfully
- Making incremental archives on specific dbspacesets

When those events finish, verify the contents of the ON-Archive activity log. In addition, make sure that you check the contents of the ON-Archive activity log after you complete the following tasks:

- Explicitly define dbspacesets, volume sets, and volumes
- Create requests to archive and back up without executing those requests
- Restore a dbspace
- Enact a cold restore

For more information on the ON-Archive activity log, see "Adding an ON-Archive Activity Log to Log Archive Events" on page 4-12.

LIST/RECOVERY Example

The following example shows one way to display recovery information using the LIST/RECOVERY command:

```
LIST/RECOVERY/DBSPACE=*
```

The output displays recovery information for all dbspaces in an installation that performs whole archives daily at approximately 1 a.m., level 0 archives monthly, level 1 archives Sunday mornings, level 2 archives daily, and continuous log backups (three logs per saveset) between archives.

```
onarchive> LIST/RECOVERY
Recovery requirements as of April 18, 1994 at 09:20 for * + = required for minimal restore
Vol  Save Set VSet              Label Device Date          Level
0100 00000100 March_Monthly     Mar0M 8mmTape01-APR-1994 01:130 dbs1, dbs2, dbs3, rootdbs+
0101 00000100 March_Monthly     Mar1M 8mmTape01-APR-1994 01:130 dbs1, dbs2, dbs3, rootdbs+
0102 00000100 March_Monthly     Mar2M 8mmTape01-APR-1994 01:130 dbs1, dbs2, dbs3, rootdbs+
0134 00000130 April_Week_2      Apr0W28mmTape17-APR-1994 01:091 dbs1, dbs2, dbs3, rootdbs+
0135 00000132 April_Week_2      Apr1W28mmTape17-APR-1994 01:252 dbs1, dbs2, dbs3, rootdbs+
0137 00000132 April-Day_18_Logs Apr18L8mmTape18-APR-1994 05:47Backup LF00001234, LF00001235, LF00001236
0140 00000134 April-Day_18_Logs Apr18L8mmTape18-APR-1994 09:15Backup LF00001237, LF00001238, LF00001239
```

The MODIFY/COMMAND Command

The MODIFY/COMMAND command enables you to modify previously entered ARCHIVE, BACKUP, COPY, REMOVE (with WAIT), and RETRIEVE requests that have a status of NEW, FAILED, CANCELLED, or UNCOMPLETED. The MODIFY/COMMAND command enables you to add, replace, or delete the qualifiers in the qualifier lists of these commands.

Modifying a request whose status is NEW simply updates its qualifier list with the specified changes. Modifying requests with any of the other permitted status values generates a new request with the updated qualifier list.

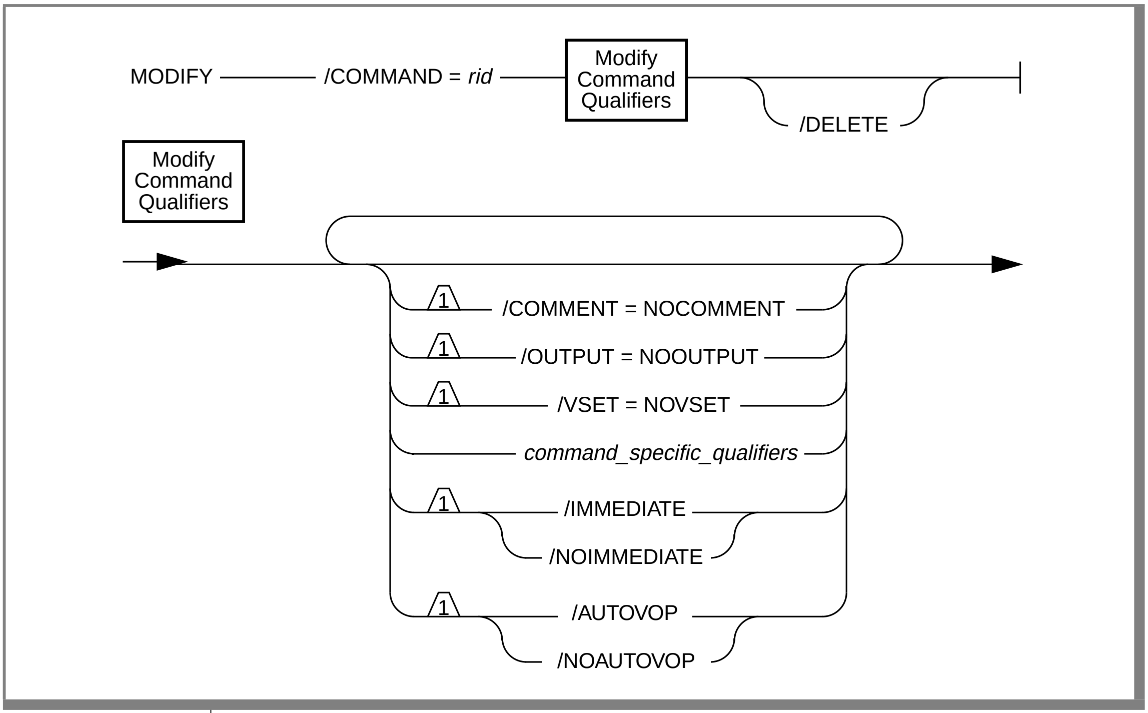
You can use the DELETE qualifier when you modify requests of FAILED, CANCELLED, or UNCOMPLETED status.

When you use the **onarchive** menu interface, you can delete qualifiers from a command by simply blanking out the field on the menu.

The following restrictions apply to the MODIFY/COMMAND command:

- If ON-Archive is running in OPERATOR privilege mode, a user must be **informix** or **root** to modify a request. Users **informix** and **root** can modify any other user’s requests.
- If ON-Archive is running in GROUP privilege mode, users can only modify their own requests. Users who are members of the **super_archive** group can modify any other user’s requests.

The new MODIFY/COMMAND qualifiers are AUTOVOP, NOAUTOVOP, IMMEDIATE, and NOIMMEDIATE.



AUTOVOP	specifies when you want to proceed to other tasks without waiting for a request to finish.
NOAUTOVOP	specifies the default setting for the MODIFY/COMMAND command. Allows you to override an AUTOVOP default setting, specified in a personal default file, to ensure that the MODIFY/COMMAND command functions as in earlier versions.
COMMAND	specifies the original request to modify. *rid* is a request in the catalog.
command-specific-qualifiers	The qualifiers permitted with this command depend on the type of request being modified. For example, if an ARCHIVE request is being modified, only the qualifiers permitted with that command are valid.
COMMENT= NOCOMMENT	removes the comment from the specified request.

DELETE	can only be used with requests with the status FAILED, CANCELLED, or UNCOMPLETED. This qualifier deletes the original request.
IMMEDIATE	simplifies the process of creating and executing a request. The IMMEDIATE qualifier saves time by allowing you to create and execute a request using a single command. Displays the request ID associated with that request entry.
NOIMMEDIATE	specifies the default setting for the MODIFY/COMMAND command. Allows you to override an IMMEDIATE default setting, specified in a personal default file, to ensure that the MODIFY/COMMAND command functions as in earlier versions.
OUTPUT= NOOUTPUT	removes the OUTPUT qualifier from the specified request.
VSET=NOVSET	removes the VSET qualifier from the specified request.

See “The REMOVE/FAILED_REQUEST Command,” “The REMOVE/REQUEST Command,” and “Groups of Qualifiers” in Chapter 8 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide* for more information on removing requests from the ON-Archive catalog.

MODIFY/COMMAND Examples

The following example removes the comment from request 11:

```
Onarchive> MODIFY/COMMAND=11/COMMENT=NOCOMMENT
```

```
      Request 00000011 modified
```

The following command creates a new request, using request 21 as a base, modifying the DBSPACESET qualifier in request 21. It also removes request 21 from the catalog.

```
Onarchive> MODIFY/COMMAND=21/DBSPACESET=*/DELETE
```

```
   Request 00000034 registered in the catalog
   Request 00000021 removed from the catalog
```

The following command removes the cancelled request 13 from the catalog:

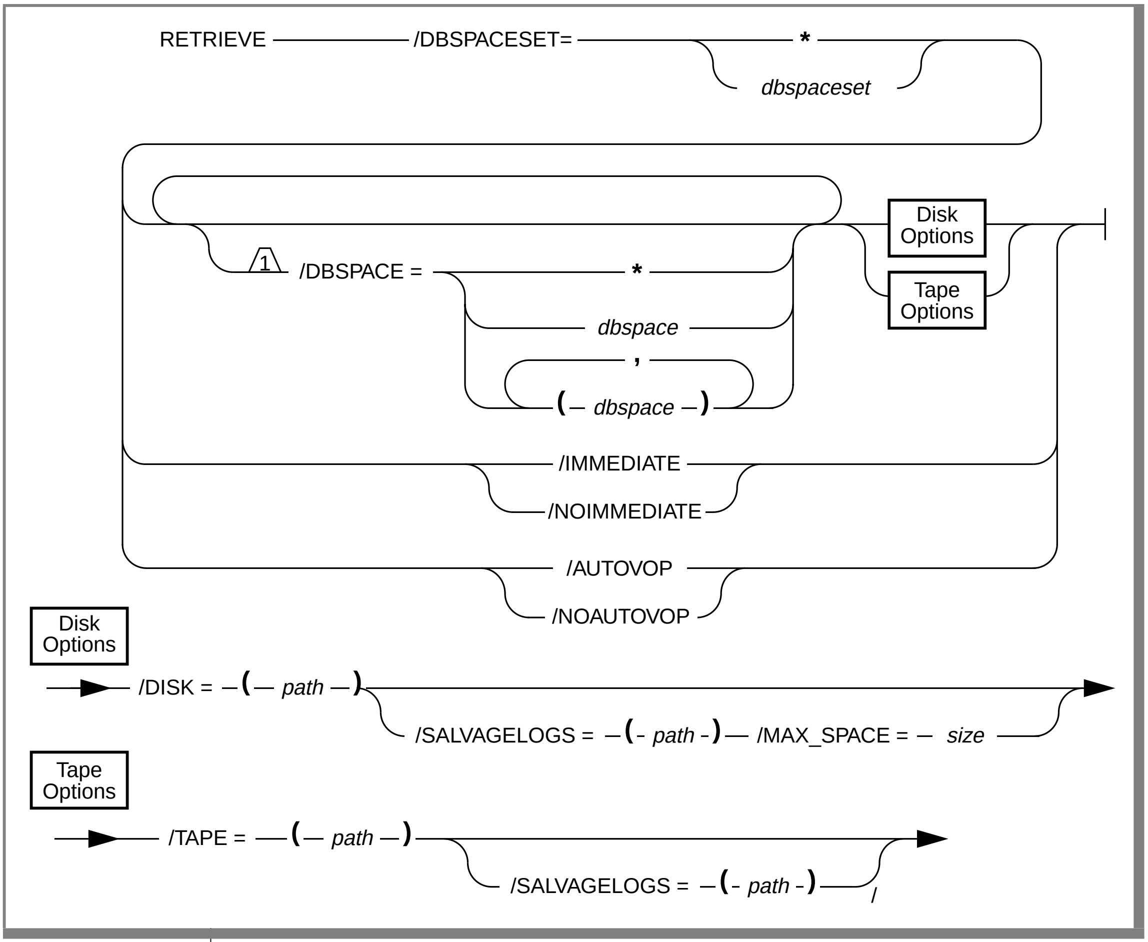
```
Onarchive> MODIFY/COMMAND=13/DELETE
```

The following command negates the APART, LOG, and EXPIRY_DATE qualifiers for request 30:

```
Onarchive> MODIFY/COMMAND=30/NOAPART/NOLOG/NOEXPIRY_DATE
```

RETRIEVE/DBSPACESET Command

The RETRIEVE/DBSPACESET command performs a physical restore of the dbspace set or dbspaces specified. The root dbspace must be among the first dbspaces restored. You can only use this command when OnLine is off-line. The new RETRIEVE/DBSPACESET qualifiers are AUTOVOP, NOAUTOVOP, IMMEDIATE, and NOIMMEDIATE. For information on how to use the RETRIEVE/DBSPACESET command with the **ondatartr** utility, see Chapter 7 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide.*


RETRIEVE
/DBSPACESET=
*
dbspaceset
1
/DBSPACE =
*
dbspace
,
(dbspace)
Disk Options
Tape Options
/IMMEDIATE
/NOIMMEDIATE
/AUTOVOP
/NOAUTOVOP
Disk Options
/DISK = (path)
/SALVAGELOGS = (path) /MAX_SPACE = size
Tape Options
/TAPE = (path)
/SALVAGELOGS = (path)

AUTOVOP	specifies when you want to proceed to other tasks without waiting for a request to finish.
NOAUTOVOP	specifies the default setting for the RETRIEVE/DBSPACESET command. Allows you to override an AUTOVOP default setting, specified in a personal default file, to ensure that the RETRIEVE/DBSPACESET command functions as in earlier versions.
DBSPACE	specifies particular dbspaces to be restored within a dbspace set. When you omit this qualifier, all the dbspaces within the dbspace set are restored. *dbspace* is any dbspace that was archived with this dbspace set. * specifies all dbspaces for the dbspace set.
DBSPACESET	specifies the dbspace set or sets to be restored. The root dbspace must be among the first dbspaces restored. If * is specified, it refers to a restore of all the dbspaces managed by OnLine. To use * for a restore, you must have created the archive that you want to restore using the * parameter with the ARCHIVE/DBSPACESET command. *dbspaceset* is any dbspace set that was archived in this save set.
DISK	specifies the pathname of the disk volume from which the dbspaceset is being restored. *path* is the pathname to a file.
IMMEDIATE	simplifies the process of creating and executing a request. The IMMEDIATE qualifier saves time by allowing you to create and execute a request using a single command. Displays the request ID associated with that request entry.
NOIMMEDIATE	specifies the default setting for the RETRIEVE/DBSPACESET command. Allows you to override an IMMEDIATE default setting, specified in a personal default file, to ensure that the RETRIEVE/DBSPACESET command functions as in earlier versions.

MAX_SPACE		specifies the size of the disk volume in 512-byte blocks where the salvaged logs are written. MAX_SPACE is required when the DISK qualifier is present.
	size	is an integer between 0 and 99999999. The minimum usable size of MAX_SPACE is 16, which is ((page size * 3) / 512) + 4.
SALVAGELOGS		instructs **ondatartr** to back up any logical-log files that are currently not backed up. It backs them up to the specified volume before a level-0 physical restore occurs. The device type (DISK or TAPE) is assumed to be the same as is specified for the retrieve operation, although the pathname to the volume device can be different. The volume created is in the vset ONDATARTRLOG and is given a volume number 1. You can later use the CAT-ALOG command to catalog the volume.
	path	is the pathname to either a file or a tape device. Pathnames used here can differ from the pathnames specified in the DISK and TAPE qualifiers but must be of the same type.
TAPE		specifies the pathname of the device that has mounted on it the tape volume from which the dbspaceset is being restored.
	path	is the pathname to a tape device.

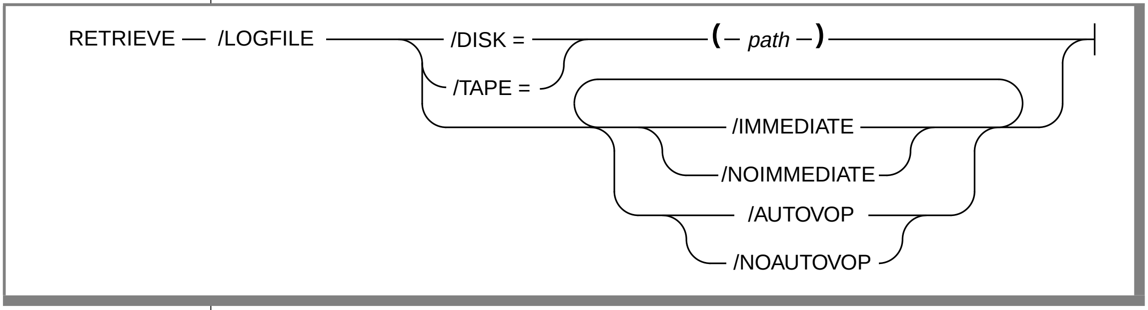
As described in “Start Parallel Cold Physical Restores” in Chapter 7 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*, once OnLine is in recovery mode, you can use a different **ondatartr** process to start a physical restore of another save set (or sets).

As described in “Perform a Logical Restore with ondatartr” in Chapter 7 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*, once the physical restore is completed, you can use **ondatartr** to start a logical restore, or you can bring OnLine to on-line mode without restoring the logical-log files. If you do not restore logical-log files, your OnLine data is restored only to its state at the time of the archive.

The RETRIEVE/LOGFILE Command

The RETRIEVE/LOGFILE command determines which logical-log files were backed up after the last archive, tells you the numbers, and prompts you to enter the request IDs of commands to retrieve them.

OnLine must be running (in recovery mode) to execute this command. The **oncatalgr** utility, however, cannot be running. After the log files are retrieved, OnLine is in quiescent mode. The new RETRIEVE/LOGFILE qualifiers are AUTOVOP, NOAUTOVOP, IMMEDIATE, and NOIMMEDIATE. For information on how to use the RETRIEVE/LOGFILE command with the **ondatartr** utility, see Chapter 7 of the *INFORMIX-OnLine Dynamic Server Archive and Backup Guide*.



AUTOVOP	specifies when you want to proceed to other tasks without waiting for a request to finish.
NOAUTOVOP	specifies the default setting for the RETRIEVE/LOGFILE command. Allows you to override an AUTOVOP default setting, specified in a personal default file, to ensure that the RETRIEVE/LOGFILE command functions as in earlier versions.
DISK	specifies the pathname of the disk volume from which the log files are being restored.
	path is the pathname to a directory.
IMMEDIATE	simplifies the process of creating and executing a request. The IMMEDIATE qualifier saves time by allowing you to create and execute a request using a single command. Displays the request ID associated with that request entry.

NOIMMEDIATE	specifies the default setting for the RETRIEVE/LOGFILE command. Allows you to override an IMMEDIATE default setting, specified in a personal default file, to ensure that the RETRIEVE/LOGFILE command functions as in earlier versions.
LOGFILE	specifies that you start a logical-restore operation.
TAPE	specifies the pathname of the device that has mounted on it the tape volume from which the log files are being restored.
	path is the pathname to a tape device.

Chapter

SQL API Enhancements

5

This chapter describes the new features available within this release of the Informix SQL application-programming interface (API) products.

Flagging Informix Extensions

If you are writing an application that conforms to the ANSI SQL-92 standard, you can check an ESQL program for syntax containing Informix extensions to this standard by performing one of the following actions:

- Set the **DBANSIWARN** environment variable to check for Informix extensions in the static SQL statements (at compile time) and the dynamic SQL statements (at run time).
- Specify the **-ansi** command-line option of your ESQL preprocessor to check for Informix extensions *only* in the static SQL statements (at compile time).

With this release, the ESQL preprocessors (**esql** and **esqlcobol**) flag only the features that do *not* conform to the ANSI SQL-92 entry level. Therefore, these preprocessors no longer flag the following 7.1 features as Informix extensions:

- Use of delimited identifiers

  For more information on delimited identifiers in ESQL/C, see Chapter 1, "Programming in ESQL/C" in the *INFORMIX-ESQL/C Programmer's Manual*. Also see the Identifier segment in the *Informix Guide to SQL: Syntax*.
- Use of the AS keyword in the SELECT statement for labelling column expressions

  For more information on this keyword, see the entry for SELECT in the *Informix Guide to SQL: Syntax*.

For more information on the ANSI flagging of SQL syntax, see the descriptions of the SELECT statement and the Identifier segment in Chapter 3, "SQL Enhancements," of this supplement.

Identifying New SQL Statements

When you execute an SQL statement dynamically, you can use the DESCRIBE statement to identify the type of SQL statement being executed. DESCRIBE sets the global SQLCODE variable to an integer value that identifies the SQL statement. The **sqlstype.h** header file provides a set of defined constants that you can use in your code instead of the actual constant values.

Figure 5-1 shows the defined constants added to the **sqlstype.h** file for the SQL statements that are new with this release.

Figure 5-1
New SQL statement constants

SQL Statement	ESQL Defined Constant	Constant Value
SET	SQ_SETOBJMODE	76
START VIOLATIONS TABLE	SQ_START	77
STOP VIOLATIONS TABLE	SQ_STOP	78
GRANT FRAGMENT	SQ_GRANT (same as the GRANT statement)	18
REVOKE FRAGMENT	SQ_REVOKE (same as the REVOKE statement)	19

Within your ESQL application, use the constants named in Figure 5-1 instead of the numeric values to identify these SQL statements when these statements are executed dynamically.

For more information on the use of the DESCRIBE statement within an ESQL/C application, see Chapter 10, “Dynamic SQL in INFORMIX-ESQL/C,” of the *INFORMIX-ESQL/C Programmer’s Manual*. For information about the use of DESCRIBE within an ESQL/COBOL application, see Chapter 6, “Dynamic Management in INFORMIX-ESQL/COBOL,” of the *INFORMIX-ESQL/COBOL Programmer’s Manual*. In addition, see the entry for DESCRIBE in the *Informix Guide to SQL: Syntax*.

New Warning Values

With this release, the GRANT and REVOKE statements now return a warning in the following cases:

- You attempt to grant ALL privileges to a user when you do not have all seven table-level privileges yourself.

  The GRANT statement successfully grants the privileges that you have but sets a warning condition to indicate that some privileges have not been granted.
- You attempt to revoke ALL privileges from a user who has not been granted all seven table-level privileges.

  The REVOKE statement successfully revokes the privileges that the revokee has but sets a warning condition to indicate that some privileges have not been revoked.

In an ESQL application, you can check for this warning condition using either the SQLSTATE variable or the SQLCA structure, as shown in the following table:

Condition	SQLSTATE	SQLCA Structure: ESQL/C	SQLCA Structure: ESQL/COBOL
GRANT ALL does not grant all seven table-level privileges	'01007'	In **sqlca.sqlwarn**: **sqlwarn3** is set to "W"	In SQLWARN OF SQLCA: SQLWARN3 is set to "W"
REVOKE ALL does not revoke all seven table-level privileges	'01006'	In **sqlca.sqlwarn**: **sqlwarn1** is set to "W"	In SQLWARN OF SQLCA: SQLWARN1 is set to "W"

For more information, see the description of the GRANT and REVOKE statements in Chapter 3, "SQL Enhancements," of this supplement. Chapter 3 also contains information on the new SQLWARN settings.

New ESQL/C Function

This version of the ESQL / C library contains a new function called **sqgetdbs()**. This section contains the following information about this function:

- A function description that describes the syntax, arguments, and return values as well as a general description of this function
- A sample program (in the file **sqgetdbs.ec**) that uses **sqgetdbs()** to display a list of available databases

sqgetdbs()

Purpose

The **sqgetdbs()** function returns the names of databases that a database server can access.

Syntax

```
int sqgetdbs(ret_fcnt, dbnarray, dbnsize, dbnbuffer, dbnbufsz)
       int *ret_fcnt;
       char **dbnarray;
       int dbnsize;
       char *dbnbuffer;
       int dbnbufsz;
```

ret_fcnt	is a pointer to the number of database names returned.
dbnarray	is a user-defined array of character pointers.
dbnsize	is the size of the *dbnarray* user-defined array.
dbnbuffer	is a pointer to a user-defined buffer that contains the names of the databases returned by the function.
dbnbufsz	is the size of the *dbnbuffer* user-defined buffer.

You must provide the following user-defined data structures to the **sqgetdbs()** function:

- The *dbnbuffer* buffer holds the names of the null-terminated database names that **sqgetdbs()** returns.
- The *dbnarray* array holds pointers to the database names stored in the *dbnbuffer* buffer. For example, *dbnarray*[0] points to the first character of the first database name returned (in *dbnbuffer*), *dbnarray*[1] points to the first character of the second database name, and so on.

If the application is connected to a database server, a call to **sqgetdbs()** returns the names of the databases that are available in the database server of the current connection. Otherwise, it returns the database names available in the default database server (indicated by the **INFORMIXSERVER** environment variable). If you use the **DBPATH** environment variable to identify additional database servers that contain databases, **sqgetdbs()** also lists the databases that are available on these database servers. It first lists the databases that are available through **DBPATH** and then the databases that are available through the **INFORMIXSERVER** environment variable.

Return Codes

0	Successfully obtained database names
<0	Unable to obtain database names

Example

This sample program is contained in the **sqgetdbs.ec** file in the **demo** directory.

```
/*
   * sqgetdbs.ec *

   This program lists the available databases in the database server
   of the current connection.
*/

#include <stdio.h>

/* Defines used with exception-handling function: exp_chk() */
#define WARNNOTIFY     1
#define NOWARNNOTIFY   0

/* Defines used for user-defined data structures for sqgetdbs() */
#define BUFFSZ          256
#define NUM_DBNAMES      10

main()
{
    char db_buffer[ BUFFSZ ];     /* buffer for database names */
    char *dbnames[ NUM_DBNAMES ]; /* array of pointers to database
                                       names in 'db_buffer' */
    int num_returned;             /* number of database names returned */
    int ret, i;

    printf("SQGETDBS Sample ESQL Program running.\n\n");

    EXEC SQL connect to default;
    exp_chk("CONNECT TO default server", NOWARNNOTIFY);
    printf("Connected to default server.\n");
    ret = sqgetdbs(&num_returned, dbnames, NUM_DBNAMES,
        db_buffer, BUFFSZ);
    if(ret < 0)
        {
        printf("Unable to obtain names of databases.\n");
        exit(1);
        }

    printf("\nNumber of database names returned = %d\n", num_returned);

    printf("Databases currently available:\n");
    for (i = 0; i < num_returned; i++)
        printf("\t%s\n", dbnames[i]);

    printf("\nSQGETDBS Sample Program over.\n\n");
}

/*
 * The exp_chk() file contains the exception handling functions to
 * check the SQLSTATE status variable to see if an error has occurred
 * following an SQL statement. If a warning or an error has
 * occurred, exp_chk() executes the GET DIAGNOSTICS statement and
 * displays the detail for each exception that is returned.
 */
EXEC SQL include exp_chk.ec;
```

For a listing of the **exp_chk()** exception-handling function, see Chapter 8, “Exception Handling,” of the *INFORMIX-ESQL/C Programmer’s Manual*.

Example Output

The output you see from the **sqgetdbs** sample program depends on the settings of your **INFORMIXSERVER** and **DBPATH** environment variables. The following sample output assumes that the **INFORMIXSERVER** environment variable is set to **mainserver** and that this database server contains three databases called **stores7**, **sysmaster**, and **tpc**. This output also assumes that the **DBPATH** environment is *not* set.

```
SQGETDBS Sample ESQL Program running.

Connected to default server.

Number of database names returned = 3
Databases currently available:
    stores7@mainserver
    sysmaster@mainserver
    tpc@mainserver

SQGETDBS Sample Program over.
```

Chapter

DB-Access Enhancements

6

This chapter describes the new features of the DB-Access utility that is packaged with INFORMIX-OnLine Dynamic Server and the INFORMIX-SE database server. For additional information, see the *DB-Access User Manual*.

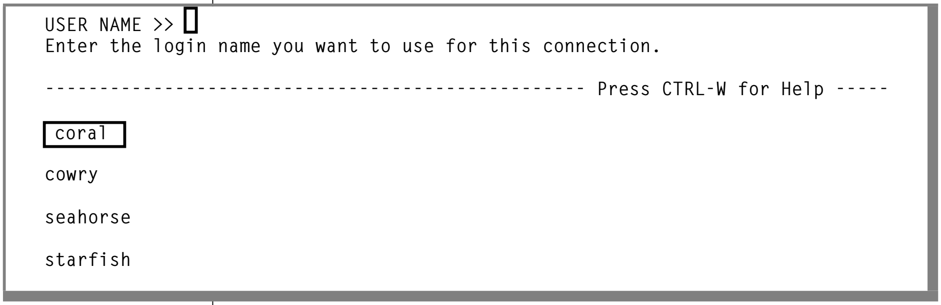
USER Clause of CONNECT Statement

DB-Access supports the USER clause of the CONNECT statement through the CONNECTION and SQL menus and in interactive non-menu mode. This feature allows you to specify a user ID or a user ID and password before you connect to a database environment.

The CONNECTION Menu

On the DB-Access main menu, press the C key or highlight the Connection option and press RETURN to call up the CONNECTION menu. To connect to a database server, press the C key on the CONNECTION menu or press RETURN.

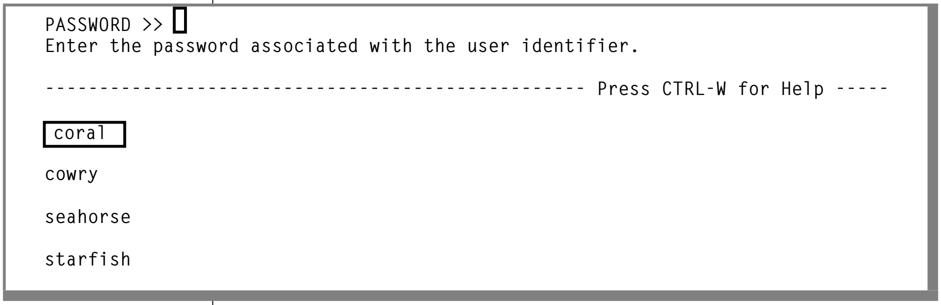
DB-Access displays a list of available database servers and prompts you to make a selection. Select a database server and DB-Access prompts you to enter a user name, as shown in Figure 6-1.



```
USER NAME >> 
Enter the login name you want to use for this connection.

-------------------------------------------------- Press CTRL-W for Help -----

coral

cowry

seahorse

starfish
```

Figure 6-1
The USER NAME prompt screen

- If you do not specify a user identifier on the USER NAME screen and simply press RETURN, you see the standard SELECT DATABASE screen listing databases on the chosen database server.
- If you enter the login name that you want DB-Access to use when connecting to the target database server, DB-Access displays the PASSWORD screen, as shown in Figure 6-2.



```
PASSWORD >> 
Enter the password associated with the user identifier.

-------------------------------------------------- Press CTRL-W for Help -----

coral

cowry

seahorse

starfish
```

Figure 6-2
The PASSWORD prompt screen

Enter on the PASSWORD screen a password associated with the user identifier, or press RETURN if you do not want to enter a password. For security reasons, the password that you enter on the screen is not displayed.

If the user identifier and password combination is valid, you connect to the target database server. You can then select a database on that database server.

For more information about using the CONNECTION menu to connect to a database environment, see Chapter 6, "The Connection and Session Menu Options," in the *DB-Access User Manual*.

The SQL Menu

You can enter the SQL statement CONNECT with the USER clause through the SQL menu in DB-Access. Select the Query-language menu from the DB-Access main menu and then select the SQL menu to enter SQL statements. Other options on the Query-language menu let you execute, modify, or redirect the output for your CONNECT statement or select an existing command file containing a CONNECT statement.

Tip: *Do not include the USING clause in your CONNECT statement. If you do, you will see error message -32412.*

For information on the Query-language features of DB-Access, see Chapter 3, "The Query-language Menu Option," in the *DB-Access User Manual*. For the syntax of the CONNECT ... USER statement, see the *Informix Guide to SQL: Syntax*. For additional information on SQL statements, see Chapter 3, "SQL Enhancements," in this supplement.

Interactive Non-Menu and Background Modes

You can use the CONNECT ... USER syntax in SQL statements that are issued in interactive mode; however, DB-Access does not support the USING *password* syntax of the CONNECT statement in certain situations.

For complete information on CONNECT statement syntax, see the *Informix Guide to SQL: Syntax*. For information on using DB-Access in interactive non-menu mode, see Chapter 1, “Working with DB-Access,” in the *DB-Access User Manual*.

Connecting in Interactive Non-Menu Mode

When you include the USER clause in a CONNECT statement in interactive mode, DB-Access prompts you to enter a password. You can either enter a user identifier or press the RETURN key. If you type in a password, you cannot see it on the screen.

The following command examples show how to connect to a database server in interactive mode.

The first example uses the CONNECT statement without a USING clause:

```
dbaccess - -

> connect to '@calserve';

     Disconnected.

     Connected.
```

If you include the USER clause in a CONNECT statement, as shown in the second example, DB-Access prompts you for a password, using echo suppression:

```
> connect to '@calserve' user 'dianne';

     ENTER PASSWORD:

     Connected.
```

For security reasons, do not type the password on the screen where it can be seen. To avoid displaying the password, also do not include the USING clause in a CONNECT statement in interactive mode, as shown in the third example command:

```
> connect to '@calserve' user 'dianne' using senat0r;
```

DB-Access cannot prompt you in background mode and returns the following error:

```
32412 USING clause unsupported. DB-Access will prompt you for
a password.
```

Connecting with a File or Shell File in Background Mode

You can execute the USER clause of a CONNECT statement in a DB-Access file or shell script and can include the USING clause.

The following example uses a command file that contains a CONNECT statement with a USER clause to connect to a database server:

```
dbaccess - connfile.sql
```

Important: *You can explicitly include the USING password clause of a CONNECT statement in a DB-Access command file, but, for security reasons, you should make the command file read-only by the user.*

The following example uses a shell file to connect to a database server. DB-Access prompts you for a password.

```
dbaccess - - <<\!
connect to '@calserve' user 'dianne';
!

ENTER PASSWORD:
```

Error Messages

OnLine Error Messages

None of the new OnLine features generate numbered error messages. Instead, OnLine sends new and updated error message to the message log. See Chapter 1, "OnLine Enhancements," for more information about OnLine error messages.

SQL Error Messages

Most of the following SQL messages are new in this release. A few of the messages are existing messages that have been revised in this release. See the Version 7.1 *Informix Error Messages* manual for further information on SQL messages.

-322 Cannot create a trigger on, alter, rename view *view-name*.

You can only create a trigger on a table. Consider creating the trigger on the table from which the view is derived, or consider creating view *view-name* as a table and then creating the trigger on it.

You can also receive this message if you issue the START VIOLATIONS TABLE statement or the STOP VIOLATIONS TABLE statement for a view. You must specify the name of a base table in both of these statements.

-388 No resource permission.

If you issued a CREATE TABLE, CREATE INDEX, or CREATE PROCEDURE statement, you cannot execute this statement because your account has not been granted Resource privilege in this database. You need the Resource privilege to create permanent tables, indexes on permanent tables, and procedures.

If you issued a SET statement, START VIOLATIONS TABLE statement, or STOP VIOLATIONS TABLE statement, you cannot execute this statement because your account has not been granted Resource privilege in this database. You need the Resource privilege to execute the SET statement for a constraint, trigger, or index defined on a table in the current database. You also need the Resource privilege to execute the START VIOLATIONS TABLE or STOP VIOLATIONS TABLE statement on a base table in the current database.

To recover from this error, contact a person who has the DBA privilege on this database and ask to be granted the Resource privilege on the database.

-525 Failure to satisfy referential constraint *constraint-name*.

During an ALTER TABLE or SET statement, you have added or re-enabled a referential constraint that is violated by the data in the table. Check that the data in the referencing column (child key) exists in the referenced column (parent key).

-710 Table *table-name* has been dropped, altered, or renamed.

Error -710 can occur with explicitly prepared statements. These statements have the following form:

```
PREPARE statement id FROM quoted string
```

After a statement has been prepared in the database server and before the user executes it, the table has been renamed or altered, possibly changing the structure of the table. Problems might occur as a result.

Error -710 can also occur with stored procedures. Before executing a new stored procedure for the first time, the database server optimizes the code (statements) in the stored procedure. Optimization makes the code depend on the structure of the tables that the procedure references. If the table structure changes after the procedure is optimized and before it is executed, error -710 can occur.

Each stored procedure is optimized the first time that it is run (not when it is created). This behavior means that a stored procedure might succeed the first time that it is run and yet fail later under virtually identical circumstances. The failure of a stored procedure can also be intermittent because failure during one execution forces an internal warning to reoptimize the procedure before the next execution.

The database server keeps a list of tables that the stored procedure references explicitly. Whenever any of these explicitly referenced tables is modified, the database server reoptimizes the procedure the next time that the procedure is executed.

However, if the stored procedure depends on a table that is referenced only indirectly, the database server cannot detect the need to reoptimize the procedure after that table is changed. For example, a table can be referenced indirectly if the stored procedure invokes a trigger. If a table that is referenced by the trigger (but not directly by the stored procedure) is changed, the database server does not know that it should reoptimize the stored procedure before running it. When the procedure is run after the table has been changed, error -710 can occur.

You can prevent error -710 by forcing reoptimization of the stored procedure. You can force reoptimization by executing the following statement:

```
UPDATE STATISTICS FOR PROCEDURE procedure name
```

You can add this statement to your program in either of the following ways:

- Place the UPDATE STATISTICS statement after each statement that changes the mode of an object.
- Place the UPDATE STATISTICS statement before each execution of the stored procedure.

It is most efficient to place the UPDATE STATISTICS statement with the action that occurs less frequently in the program (change of object mode or execution of the procedure). In most cases, the change of object mode occurs less frequently.

You must execute the UPDATE STATISTICS statement for each stored procedure that indirectly references the changed tables unless the procedure also references the tables explicitly.

You can use either of the following methods to recover from error -710:

- You can force reoptimization of the stored procedure by issuing the UPDATE STATISTICS statement, as described in the preceding paragraphs.
- You can rerun the stored procedure.
- The first time that the stored procedure fails, the database server marks the procedure as in need of reoptimization. The next time that you run the procedure, the database server reoptimizes the procedure before running it. However, running the stored procedure twice might be neither practical nor safe. Forcing reoptimization of the procedure by using the UPDATE STATISTICS statement is a safer choice.

-859 "Distributions Only" is not meaningful in an update statistics LOW request.

You cannot specify the DISTRIBUTIONS ONLY option in the LOW mode of the UPDATE STATISTICS statement.

-886 Cannot drop table or view because of existing dependencies.

When you issue a DROP TABLE or DROP VIEW statement, you cannot drop the table or view if you specify the RESTRICT option and a view or foreign-key constraint exists that depends on that table or view.

You also cannot drop a table if you specify the RESTRICT option and a violations and diagnostics table exist for that table.

-891 Temporary table objects can only be enabled.

You cannot change the object mode of a temporary table object to the disabled or filtering object mode.

-892 Cannot disable object *object-name* due to other active objects using it.

Other objects are using this object. If the object being disabled is an index, then a unique constraint, primary constraint, or referential constraint might be using that object. If the object is a unique or a primary-key constraint, then a referential constraint might be using that object.

-893 Cannot activate / create object *object-name* because of its dependencies.

The user has issued a SET statement to set a database object to the enabled or filtering object mode, or the user has issued a CREATE INDEX, CREATE TRIGGER, or CREATE TABLE statement to create a database object in the enabled or filtering object mode. However, this object needs other disabled objects. For example, before enabling a referential constraint on a table, the user must first enable the indexes that the constraint needs.

-894 Cannot find object *object-name*.

The object name that the user specified explicitly in the SET statement is not found in the database.

-895 Cannot create violations / diagnostics table.

The user has issued a START VIOLATIONS TABLE statement for a target table. The database server is not able to create the violations and diagnostics tables for this target table. Any one of the following situations can be the reason for the failure:

- The target table already has a violations and diagnostics table.
- The names specified for the violations and diagnostics table in the START VIOLATIONS TABLE statement are not valid. For example, if you omitted the USING clause from the statement and the number of characters in the target table name plus four characters is longer than the maximum identifier length, the generated names of the violations and diagnostics tables would be longer than the maximum identifier length. If the names of the violations and diagnostics tables are invalid for this reason, the user can rectify the problem by giving explicit names to the violations and diagnostics tables in the USING clause of the START VIOLATIONS TABLE statement.
- The names that were specified for the violations and diagnostics tables in the START VIOLATIONS TABLE statement match the names of existing tables in the database.
- The target table contains columns with the names **informix_tupleid** or **informix_optype**. Because these two column names would duplicate the **informix_tupleid** or **informix_optype** columns in the violations table, the database server cannot create the violations table.
- The target table is a temporary table.

- The target table is serving as a violations or diagnostics table for some other table.
- The target table is a system catalog table.

-896 Violations table is not started for the target table.

If a violations and diagnostics table have not been started for the target table, and an INSERT, DELETE, or UPDATE statement fails to satisfy any filtering-mode object on the target table, the user who issued the INSERT, DELETE, or UPDATE statement receives this message.

To recover from this error, you must start a violations and diagnostics table for the target table. Then, when users issue INSERT, DELETE, or UPDATE statements that fail to satisfy filtering-mode objects defined on the table, they do not receive this message.

-897 Cannot modify / drop a violations / diagnostics table.

The user has tried to alter or drop a table that is serving as a violations table or a diagnostics table for another table.

-898 Cannot alter a table which has associated violations / diagnostics tables.

The user has tried to add, drop, or modify a column in a table that has a violations and diagnostics table associated with it.

-899 Too many violations.

The number of records in the diagnostics table either exceeds or will exceed the limit specified in the MAX ROWS clause of the START VIOLATIONS TABLE statement. When a single statement on the target table (such as an INSERT or SET statement) causes more records to be inserted into the diagnostics table than the limit specified by the MAX ROWS clause, this error is returned to the user who issued the statement on the target table.

-971 Integrity violations detected.

The user has attempted to change the object mode of a disabled constraint or disabled unique index to the enabled or filtering mode, but the SET statement fails because the table contains data that violates the constraint or the unique-index requirement. If a violations table has been started for the table that contains the inconsistent data, this message is returned to the user. The message is returned whether or not the SET statement included the WITH ERROR option.

Similarly, when an INSERT, DELETE, or UPDATE statement causes some records to be added to the violations table because the statement violates a filtering mode object, this message is returned to the user if the following two conditions are true:

- The SET statement or CREATE statement that specified the filtering object mode for the object included the WITH ERROR option.
- No other errors have been encountered during the execution of the INSERT, DELETE, or UPDATE statement.

-973 Cannot insert from the violations table to the target table.

The user has issued a statement that attempts to insert rows from the violations table into the target table. For example, the user enters the following statement:

```
INSERT INTO mytable SELECT * FROM mytable_vio
```

If the target table has some filtering mode objects, this error is returned to the user. The user can recover in any of the following ways:

- Set the object mode of the filtering objects to some other mode
- Stop the violations table
- Insert rows from the violations table into a temporary table, and then insert rows from the temporary table into the target table

-974 Cannot drop not null constraint on the serial column.

The user has issued a command to drop a NOT NULL constraint on a column that has a SERIAL data type. Such constraints can be disabled, but they cannot be dropped before the column is dropped.

-975 Invalid object and object mode combination.

The user has tried to create a new object in an object mode that does not apply to that object type, or the user has set the object mode of an existing object to a mode that does not apply to that type of object. For example, if the user tries to create a trigger in the filtering mode or set an existing trigger to the filtering mode, the user receives this error.

-976 Table must be fragmented by expression to grant fragment authority.

The user attempted to grant fragment-level privileges on an unfragmented table or on a table that is not fragmented by expression.

-977 No permission on fragment (*dbspace-name*).

The user does not have the required fragment-level privilege on the table fragment. This message is always followed by another message that identifies the privilege that the user lacks. If an INSERT statement fails, the second message is -271. If an UPDATE statement fails, the second message is -346. If a DELETE statement fails, the second message is -240.

-978 No insert permission on the violations / diagnostics tables.

The user has issued an INSERT, DELETE, or UPDATE statement on a table with filtering-mode objects. The user receives this message because the user lacks the Insert privilege on the violations or diagnostics tables associated with this table. The user must have the Insert privilege on the violations and diagnostics tables before the database server can write rows into the violations and diagnostics tables for that user.

Similarly, if the user has issued a SET statement to change the object mode of a disabled constraint or a disabled unique index to the enabled or filtering mode, and if a violations table has been started for the target table, the user receives this message if the user lacks the Insert privilege on the violations or diagnostics table associated with the target table.

-19800 Role name already exists as a user or role.

You cannot create a role name identical to any user known to the system or a user or role known to the database. Change the name of the role.

-19801 Role name cannot be <*reserved word*>.

You cannot create a role name that is a reserved word. The reserved words are *connect*, *resource*, *dba, select*, *update, delete*, *insert*, *index*, *references, alter, execute*, *default*, *none*, *null, public*. Change the name of the role.

-19802 Name cannot appear as both the role granted and the role grantee.

A role cannot be granted to itself, either directly or indirectly. Verify if the role granted and grantee are the correct roles. Check for roles already granted to the role being granted.

-19803 Only the DBA, or a user granted the role with the WITH GRANT OPTION can grant, revoke, or drop a role.

Check your privileges and permissions. You must be the DBA or have been granted the role with the WITH GRANT OPTION to grant, revoke, or drop a role.

-19804 The role does not exist.

For a role to exist, the DBA must first create the role using the CREATE ROLE statement.

-19805 No privilege to set to the role.

Check your privileges and permissions. A user or role must be granted a role using the GRANT statement before the role can be set.

-19806 Cannot grant database privileges to a role.

A role cannot be granted database-level privileges. Use the GRANT statement to grant database-level privileges to the user or to PUBLIC.

-19807 Cannot grant privileges to a role WITH GRANT OPTION.

A user granted a role with the WITH GRANT OPTION cannot in turn grant the role to a user and include the WITH GRANT OPTION. Use the GRANT statement without the WITH GRANT OPTION.

-19808 User name already exists as a *rolename* in the database.

A user cannot open a database as a role name that is the same as the user name. Role names and user names must be unique in the database.

–32507 Cannot set session authorization.

You must obtain the DBA privilege when you execute the SET SESSION AUTHORIZATION statement. Otherwise, refer to the accompanying error message for more information.

–32508 Statement is invalid within a transaction.

Abort or commit a transaction before you issue the SET SESSION AUTHORIZATION statement.

–32509 Bad session authorization format.

The user name supplied as an argument to the SET SESSION AUTHORIZATION statement is invalid. You cannot use PUBLIC as a user name. Supply the user name of a valid user.

-32532 Illegal data type for VARIANCE or STDEV.

Internal error. You cannot compute VARIANCE and STDEV on character, date/time, or blob columns.

DB-Access Error Messages

-32412 USING clause unsupported. DB-Access will prompt you for a password.

DB-Access does not support the USING *password* clause in a CONNECT ... USER statement when it violates security. For example, do not type a password on the screen where it can be seen or include it in a command file that is readable by someone other than the user. To maintain security, DB-Access prompts you to enter the password on the screen and uses echo suppression to hide it from view.

ON-Archive Error Messages

ARC-01848E INFORMIXDIR not set in environment.

One of the ON-Archive processes failed in an attempt to get the value of the **INFORMIXDIR** environment variable. Make sure the environment that executes the ON-Archive process first sets the **INFORMIXDIR** environment variable.

ARC-01849E A dbspace in the list is a temporary dbspace and cannot be archived.

One of the dbspaces listed for the dbspaceset is a temporary dbspace. Change the definition of the dbspaceset to exclude any temporary dbspaces.

SQL API Error Messages

None of the new features in ESQL/C and ESQL/COBOL generate numbered error messages. For errors generated by new SQL features, see "SQL Error Messages."

Connectivity Error Messages

-27004 Illegal sqlhosts file option / parameter, *parameter*, for dbservername, *servername*.

You specified an invalid option or parameter in your **sqlhosts** file. Check the option ID and parameter associated with *servername* in the **sqlhosts** file.

-27005 Illegal **sqlexecd** daemon option, *option_value*.

You specified an invalid **sqlexecd** daemon option. Check the option *option_value* and its parameter.

-27006 Network driver cannot establish listen endpoint.

You have specified stream pipe (`ipcstr`) as the network communication type for this database server. The database server is not able to create the stream pipe. The most common cause of this error is that another database server on your network is already using the service name for this database server.

Make sure that the **servicename** field in the **$INFORMIXDIR/etc/sqlhosts** file is unique across all database servers on your network. If the service name is unique, check the accompanying ISAM code for additional error information.

Index