Guide to the Optical Subsystem

Informix Dynamic Server
Informix Dynamic Server, Developer Edition
Informix Dynamic Server, Workgroup Edition

Version 7.3
February 1998
Part No. 000-4375

Published by INFORMIX® Press

Informix Software, Inc.
4100 Bohannon Drive
Menlo Park, CA 94025-1032

Documentation Team: Smita Joshi, Geeta Karmarkar, Jennifer Leland, Barbara Nomiyama

Table of Contents

Introduction

Chapter 1 An Overview of the Optical Subsystem

Chapter 2 Installation and Initial Configuration

Chapter 3 Using the Optical Subsystem

Chapter 4 SQL for the Optical Subsystem

Index

Introduction

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

About This Manual

This manual describes the Optical Subsystem, a configuration of Informix Dynamic Server that supports the storage of TEXT and BYTE data types, also known as *simple large objects*, on optical platters. It describes the Optical Subsystem interface for an optical-storage subsystem and the set of SQL statements that you use exclusively with optical platters.

Types of Users

This manual is for the following users who want to take advantage of write-once-read-many (WORM) optical media for storage of simple large objects:

- Database administrators
- Database server administrators

This manual assumes that you have the following background:

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

If you have limited experience with relational databases, SQL, or your operating system, refer to *Getting Started with Informix Dynamic Server* for a list of supplementary titles.

Software Dependencies

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

- Informix Dynamic Server, Version 7.3
- Informix Dynamic Server, Developer Edition, Version 7.3
- Informix Dynamic Server, Workgroup Edition, Version 7.3

Assumptions About Your Locale

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

This manual assumes that you are using the default locale, **en_us.8859-1**. This locale supports U.S. English format conventions for dates, times, and currency. In addition, this locale supports the ISO 8859-1 code set, which includes the ASCII code set plus many 8-bit characters such as é, è, and ñ.

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

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

Demonstration Database

The DB-Access utility, which is provided with your Informix database server products, includes a demonstration database called **stores7** that contains information about a fictitious wholesale sporting-goods distributor. You can use SQL scripts provided with DB-Access to derive a second database, called **sales_demo**. This database illustrates a dimensional schema for data-warehousing applications. Sample command files are also included for creating and populating these databases.

Many examples in Informix manuals are based on the **stores7** demonstration database. The **stores7** database is described in detail and its contents are listed in this manual.

The scripts that you use to install the demonstration databases reside in the **$INFORMIXDIR/bin** directory on UNIX platforms and the **%INFORMIXDIR%\bin** directory on Windows NT platforms. For a complete explanation of how to create and populate the **stores7** demonstration database, refer to the *DB-Access User Manual*. For an explanation of how to create and populate the **sales_demo** database, refer to the *Informix Guide to Database Design and Implementation*.

New Features

Most of the new features for Version 7.3 of Informix Dynamic Server fall into five major areas:

- Reliability, availability, and serviceability
- Performance
- Windows NT-specific features
- Application migration
- Manageability

Several additional features affect connectivity, replication, and the Optical Subsystem. For a comprehensive list of new features, see the release notes for your database server.

This manual includes information about shared-library support for the Informix Optical Subsystem. You no longer need to order the Optical Subsystem as a separate product.

Documentation Conventions

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

The following conventions are covered:

- Typographical conventions
- Icon conventions
- Syntax conventions
- Sample-code conventions

Typographical Conventions

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

Convention	Meaning
KEYWORD	All keywords appear in uppercase letters in a serif font.
italics	Within text, new terms and emphasized words appear in italics. Within syntax diagrams, values that you are to specify appear in italics.
boldface	Identifiers (names of classes, objects, constants, events, functions, program variables, forms, labels, and reports), environment variables, database names, filenames, table names, column names, icons, menu items, command names, and other similar terms appear in boldface.
`monospace`	Information that the product displays and information that you enter appear in a monospace typeface.
KEYSTROKE	Keys that you are to press appear in uppercase letters in a sans serif font.
♦	This symbol indicates the end of feature-, product-, platform-, or compliance-specific information.

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

Icon Conventions

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

Comment Icons

Comment icons identify warnings, important notes, or tips. This information is always displayed in italics.

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

Feature, Product, and Platform Icons

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

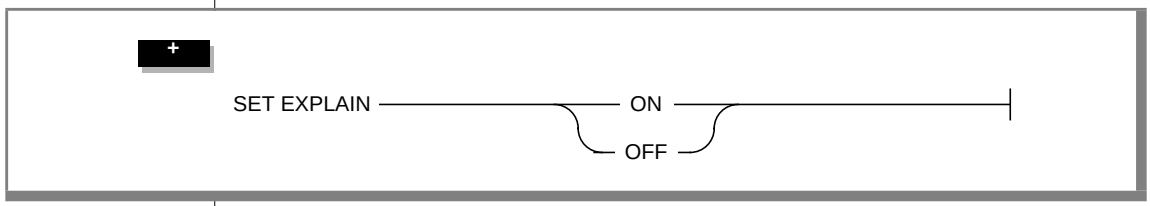
Icon	Description
GLS	Identifies information that relates to the Informix GLS feature.
IDS	Identifies information that is specific to Dynamic Server and its editions. However, in some cases, the identified section applies only to Informix Dynamic Server and not to Informix Dynamic Server, Workgroup and Developer Editions. Such information is clearly identified.
UNIX	Identifies information that is specific to the UNIX platform.
W/D	Identifies information that is specific to Informix Dynamic Server, Workgroup and Developer Editions.
WIN NT	Identifies information that is specific to the Windows NT environment.

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

Syntax Conventions

This section describes conventions for syntax diagrams. Each diagram displays the sequences of required and optional keywords, terms, and symbols that are valid in a given statement or segment, as Figure 1 shows.

Figure 1
Example of a Simple Syntax Diagram



Each syntax diagram begins at the upper-left corner and ends at the upper-right corner with a vertical terminator. Between these points, any path that does not stop or reverse direction describes a possible form of the statement.

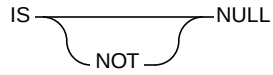
Syntax elements in a path represent terms, keywords, symbols, and segments that can appear in your statement. The path always approaches elements from the left and continues to the right, except in the case of separators in loops. For separators in loops, the path approaches counterclockwise from the right. Unless otherwise noted, at least one blank character separates syntax elements.

Elements That Can Appear on the Path

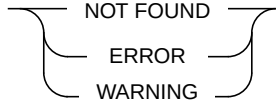
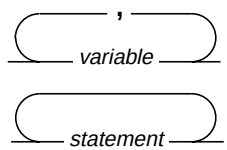
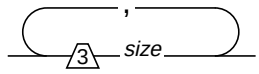
You might encounter one or more of the following elements on a path.

Element	Description
KEYWORD	A word in UPPERCASE letters is a keyword. You must spell the word exactly as shown; however, you can use either uppercase or lowercase letters.
(. , ; @ + * - /)	Punctuation and other nonalphanumeric characters 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. A table immediately following the diagram explains the value.
ADD Clause p. 1-14 ADD Clause	A reference in a box represents a subdiagram. Imagine that the subdiagram is spliced into the main diagram at this point. When a page number is not specified, the subdiagram appears on the same page.
Table Name see SQLS	A reference to SQLS in this manual refers to the *Informix Guide to SQL: Syntax*. Imagine that the subdiagram is spliced into the main diagram at this point.
E/C	An icon is a warning that this path is valid only for some products, or only under certain conditions. Characters on the icons indicate what products or conditions support the path. These icons might appear in a syntax diagram: D/B This path is valid only for DB-Access. E/C This path is valid only for INFORMIX-ESQL/C. EXT This path is valid for external routines.

(1 of 3)

Element	Description
	GLS This path is valid only if your database or application uses a nondefault GLS locale.
	OP This path is valid only for the Optical Subsystem.
	SPL This path is valid only if you are using Informix Stored Procedure Language (SPL).
	SQLE This path is valid for the SQL Editor.
	+ 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 runtime, or if you compiled with the **-ansi** flag, warning flags are set in the **sqlwarn** structure.
	A shaded option is the default action.
	Syntax that is enclosed between a pair of arrows is a subdiagram.
	The vertical line terminates the syntax diagram.
	A branch below the main path indicates an optional path. (Any term on the main path is required, unless a branch can circumvent it.)

(2 of 3)

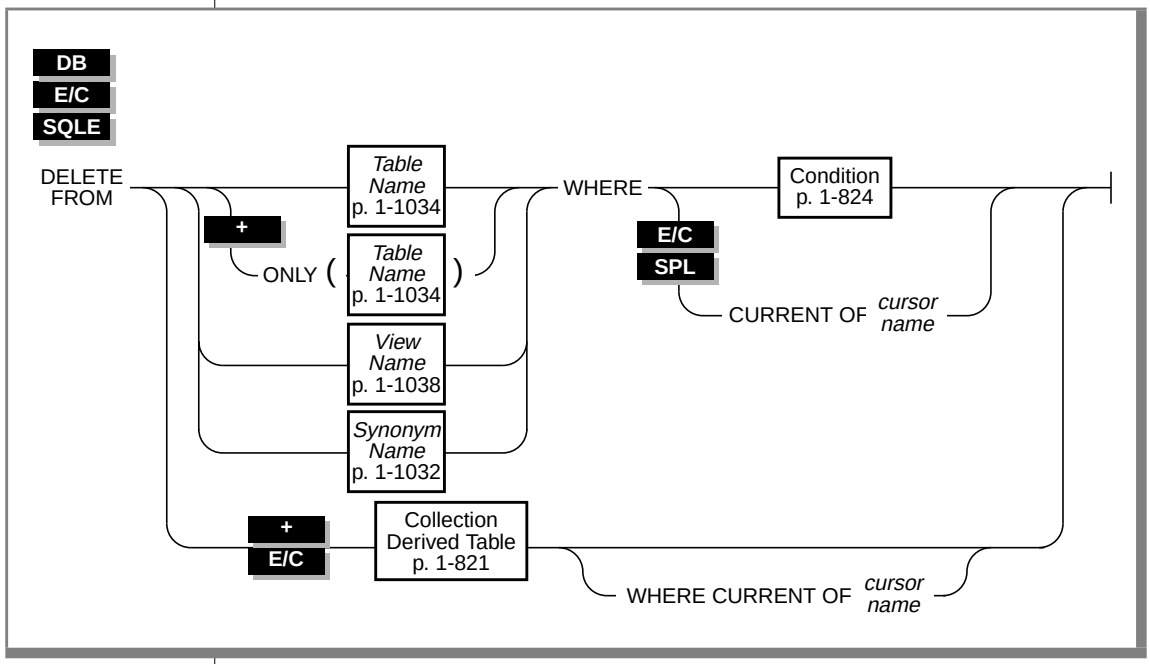
Element	Description
	A set of multiple branches indicates that a choice among more than two different paths is available.
	A loop indicates a path that you can repeat. Punctuation along the top of the loop indicates the separator symbol for list items. If no symbol appears, a blank space is the separator.
	A gate (3) on a path indicates that you can only use that path the indicated number of times, even if it is part of a larger loop. Here you can specify *size* no more than three times.

(3 of 3)

How to Read a Syntax Diagram

Figure 2 shows a syntax diagram that uses many of the elements that are listed in the previous table.

Figure 2
Example of a Syntax Diagram



The three icons at the top left of this diagram indicate that you can construct this statement if you are using DB-Access, ESQL/C, or the SQL Editor. To use the diagram to construct a statement, begin at the far left with the keywords DELETE FROM. Then follow the diagram to the right, proceeding through the options that you want.

To construct a DELETE statement

1. You must type the words DELETE FROM.
2. If you are using DB-Access, ESQL/C, or the SQL Editor, you can delete a table, view, or synonym:
   - Follow the diagram by typing the table name, view name, or synonym, as desired. Refer to the appropriate segment for available syntax options.
   - You must type the keyword WHERE.
   - If you are using DB-Access or the SQL Editor, you must include the Condition clause to specify a condition to delete. To find the syntax for deleting a condition, go to the "Condition" segment on page 1-803.
   - If you are using ESQL/C or SPL, you can include either the Condition clause to delete a specific condition or the CURRENT OF *cursorname* clause to delete a row from the table.
3. If you are using ESQL/C, you can also choose to delete from a collection-derived table:
   - Follow the diagram by going to the segment "Collection Derived Table" on page 1-800. Follow the syntax for the segment.
   - You can stop, taking the direct route to the terminator, or you can include the WHERE CURRENT OF *cursorname* clause to delete a row from a collection-derived table.
4. Follow the diagram to the terminator. Your DELETE statement is complete.

Sample-Code Conventions

Examples of SQL code occur throughout this manual. Except where noted, the code is not specific to any single Informix application development tool. If only SQL statements are listed in the example, they are not delimited by semicolons. For instance, you might see the code in the following example:

```
CONNECT TO stores7
.
.
.
DELETE FROM customer
    WHERE customer_num = 121
.
.
.
COMMIT WORK
DISCONNECT CURRENT
```

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

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

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

Additional Documentation

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

- On-line manuals
- Printed manuals
- Error message files
- Documentation notes, release notes, and machine notes

On-Line Manuals

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

Printed Manuals

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

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

Error Message Files

Informix software products provide ASCII files that contain all of the Informix error messages and their corrective actions. For a detailed description of these error messages, refer to *Informix Error Messages* in Answers OnLine.

UNIX

To read the error messages on UNIX, use the following commands.

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

♦

WIN NT

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

Documentation Notes, Release Notes, Machine Notes

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

UNIX

On UNIX platforms, the following on-line files appear in the **$INFORMIXDIR/release/en_us/0333** directory.

On-Line File	Purpose
OPTICALDOC_7.3	The documentation-notes file for your version of this manual describes features that are not covered in the manual or that have been modified since publication.
SERVERS_7.3	The release-notes file describes feature differences from earlier versions of Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.
IDS_*7.3*	The machine-notes file describes any special actions that are required to configure and use Informix products on your computer. Machine notes are named for the product described.

♦

WIN NT

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

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

Machine notes do not apply to Windows NT platforms. ♦

Compliance with Industry Standards

The American National Standards Institute (ANSI) has established a set of industry standards for SQL. Informix SQL-based products are fully compliant with SQL-92 Entry Level (published as ANSI X3.135-1992), which is identical to ISO 9075:1992. In addition, many features of Informix database servers comply with the SQL-92 Intermediate and Full Level and X/Open SQL CAE (common applications environment) standards.

Informix Welcomes Your Comments

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

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

Write to us at the following address:

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

If you prefer to send email, our address is:

doc@informix.com

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

650-926-6571

We appreciate your feedback.

Chapter

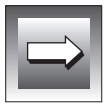
1

An Overview of the Optical Subsystem

Optical Subsystem is a functionality of Informix database servers that supports the storage of TEXT and BYTE data on an optical platter.

For more information on TEXT and BYTE data types, refer to the *Informix Guide to SQL: Reference*.

The Optical Subsystem includes an interface for an optical-storage subsystem and supports a set of SQL statements that are exclusively for use with optical platters. These SQL statements support the efficient storage and retrieval of data to and from the optical-storage subsystem.



Important: *In the manuals for Dynamic Server with UD Option, the term simple large object is used to refer to either TEXT or BYTE data. This manual does not use the term simple large object. The Optical Subsystem does not store CLOB (Character Large Object) and BLOB (Binary Large Object) data types, also referred to as smart large objects. Smart large objects use a file interface to support the optical-storage subsystem. For more information on how to store CLOB and BLOB data types on optical disc, refer to the "INFORMIX-ESQL/C Programmer's Manual."*

This chapter describes the purpose and implementation of the Optical Subsystem. The following topics are covered:

- The advantages that optical media offers
- The components of an optical-storage subsystem
- The ways in which you can organize data on optical platters
- The sequential and clustered methods of storing TEXT and BYTE data on optical platters
- The operation of the Optical Subsystem application-programming interface (API), which supports the optical-storage subsystem

The Advantages of Optical Media

Optical media offer the following advantages for storing data over conventional magnetic disks:

- Mass storage capacity (on the order of gigabytes)
- Mountable / unmountable storage units
- Low cost per bit of storage
- Long media life
- High data stability

Because optical media are usually written to and read with a laser, a bit of data is written to a much smaller area on the optical *platter* (disc) than the area that is required to record a bit on a conventional magnetic disk. Optical platters offer a very high degree of stability because they are far less susceptible than magnetic disks to corruption by environmental influences such as electromagnetic fields. Furthermore, the possibility of a read / write head crashing on to the optical disk is extremely remote because the optical disc read / write head does not come as close to the platter as the magnetic disk read / write head.

The Optical Subsystem supports only the write-once-read-many (WORM) type of optical media. When data is written to WORM optical media, a permanent, virtually incorruptible mark is made on the platter. Thereafter, you can read the data an unlimited number of times. When you update the data to a WORM optical platter, it is written to a new location on the platter; the optical-storage subsystem cannot reuse the original location.

Optical Media and TEXT and BYTE Data

In a database environment, the vast storage capacity of optical media make it particularly attractive for storing TEXT and BYTE data such as text documents, source and object programs, and scanned images. TEXT and BYTE data theoretically has no maximum size, although in practice the Optical Subsystem allows a maximum size of about 2 gigabytes per TEXT and BYTE data object. The number of locks available in the system might impose an additional constraint on the size.

The trade-off for the capacity, lower cost, integrity, and security that optical media provide, however, is slower access. In general, accessing optical media is slower than accessing magnetic disks. Typically, an optical drive accesses data at speeds that are 25 to 50 percent slower than a magnetic disk drive.

Components of the Optical-Storage Subsystem

The optical-storage subsystem has the following components:

- The platter
- The jukebox (autochanger) or stand-alone drive
- The shelf
- Management software for the optical-storage subsystem

The optical media are the removable optical platters that contain data.

You can keep optical platters in an automated library called a *jukebox* or *autochanger*. The jukebox is a cabinet that contains one or more optical platter drives, slots that store the optical platters when they are not mounted, and a robotic arm that transfers platters between the slots and the drives. Optical-storage subsystem commands initiate transport of the platters to and from the drives and manage the input/output (I/O) operations within the optical-storage subsystem.

Jukeboxes are not essential to the optical-storage subsystem. A supported optical-storage subsystem can consist of a stand-alone optical drive that holds only one platter or several stand-alone drives.

You store optical platters off-line on *shelves*. When a platter is known to the optical-storage subsystem but is not stored in the jukebox where it can be accessed, the platter is said to be *on the shelf*. The operator is responsible for storage and retrieval of platters to and from the shelf.

The management software for the optical-storage subsystem performs various tasks related to the optical-storage subsystem. It creates optical families and volumes and tracks information about them, chooses which optical drives to use and the optical platters to mount, issues commands to the jukebox, and reads and writes data to the optical platter.

Optical Platter Organization

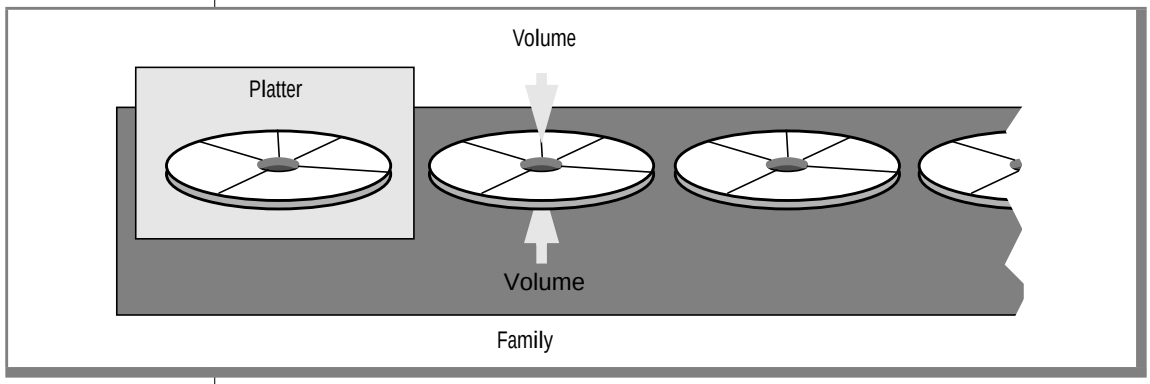
In a WORM optical-storage subsystem that the Optical Subsystem supports, the optical media are organized into units of storage called the *family* and the *volume*. Each side of the removable platter is called a volume. An optical family is theoretically an unlimited collection of volumes, although in practice the constraints of the optical-storage subsystem impose a limit on its size. When a volume becomes full, the optical-storage subsystem automatically allocates another volume from the family.

The optical-storage administrator creates an optical family with a utility that the vendor of the optical-storage subsystem provides. The administrator also uses optical-storage subsystem commands to add volumes to a family.

In a manner that is similar to an automated tape library, the optical-storage subsystem tracks the location of each TEXT and BYTE data object on a volume, each volume on a platter, and each platter in a family. Subsystem software schedules the optical drives and mounts platters as needed.

Figure 1-1 shows the relationship between the optical family, the volume, and the optical platter.

Figure 1-1
The Optical Family, the Volume, and the Optical Platter



Storage of TEXT and BYTE Data

You assign a TEXT or BYTE column to the optical-storage subsystem with the CREATE TABLE statement, which allows you to place the column data in an optical family. Prior to executing the CREATE TABLE statement, the administrator of the optical-storage subsystem must create an optical family name with a utility that the vendor of the optical-storage subsystem provides. You can then use the column-definition portion of the CREATE TABLE statement to assign the column to the optical family. For more information, see "Assigning TEXT and BYTE Columns to an Optical Platter" on page 3-4.

TEXT and BYTE data stored on optical media are not backed up (or archived). No Optical Subsystem backup and restore utilities exits. Consequently, you cannot restore them from backup media.

The Optical Subsystem can store TEXT and BYTE data on an optical volume either sequentially or in a cluster. The following sections describe each of these methods.

Storing TEXT and BYTE Data Objects Sequentially

Sequential storage means that the Optical Subsystem stores TEXT and BYTE data on the current volume of the family in the same sequence that they are inserted. When a volume becomes full, the optical-storage subsystem allocates the next volume in the family; no attempt is made to keep logically related TEXT and BYTE data objects together on the same volume or even on the same platter.

Storing TEXT and BYTE Data in a Cluster

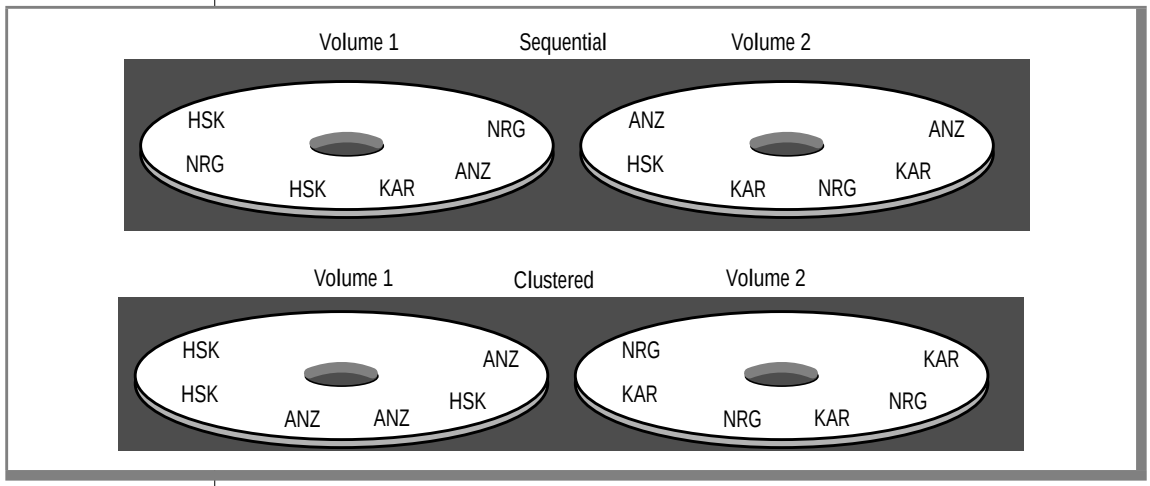
Optical clustering attempts to store logically related TEXT or BYTE data objects together on the same volume. In applications where related TEXT or BYTE data objects are often retrieved together, optical clustering minimizes time-consuming platter exchanges on the drives. This practice improves performance.

Optical clustering does not imply that TEXT or BYTE data objects that share a particular key value are stored next to each other on the optical platter. Instead, each optical cluster represents a reservation of some number of kilobytes (the clustersize) on a volume. However, the reserved kilobytes and the TEXT or BYTE data objects that are eventually stored might not be contiguous. The benefit of clustering is that you can find related TEXT or BYTE data objects on the same volume far faster than you can find them on separate volumes or platters.

Figure 1-2 illustrates the difference between sequential and clustered organization. In this example, the optical platters store TEXT or BYTE data objects that are scanned pictures of merchandise. The pictures belong to the **cat_descr** column in the **stores7** database. If you cluster the pictures, you must cluster them by reference to some other column, such as the manufacturer's name, as pictures do not have an inherent order themselves.

When stored sequentially, the optical-storage subsystem writes the pictures to the optical platter in the same order that they are inserted in the table. If you cluster them by the manufacturer code (the cluster key), the pictures for a given manufacturer are stored together. The clustered organization reduces platter exchanges on the optical drives.

Figure 1-2
Sequential Storage Versus Clustering by manu_code



Placing TEXT and BYTE Data in Optical Clusters

You create an optical cluster with the CREATE OPTICAL CLUSTER statement. For information on the syntax, refer to "CREATE OPTICAL CLUSTER" on page 4-6. As part of the syntax to define the cluster, specify the following column lists:

column list	is one or more logically related TEXT or BYTE data columns from a table. You want to store these columns together in the same cluster.
cluster-key column list	is one or more columns that provide a logical order for the TEXT or BYTE columns. The cluster-key columns must be from the same table as the TEXT or BYTE data columns.

For example, suppose you are a sporting-goods distributor and you have photographs taken of your stock to include in an electronic catalog. With Optical Subsystem, you create a table called **catalog**, which is similar to the **catalog** table provided with the **stores7** database but which stores the **cat_picture** column on an optical platter rather than a magnetic disk.

TEXT and BYTE Data Columns

In the CREATE TABLE statement for **catalog**, you define a BYTE column called **cat_picture** and use the IN portion of the column definition to place the column in the optical family **stock_photos**. For the complete syntax of the CREATE TABLE statement, see the *Informix Guide to SQL: Syntax*.

The Cluster-Key Column

Next, you decide that you want to keep the photographs for a given manufacturer together to avoid unnecessary platter exchanges during retrieval. To keep photographs for a given manufacturer together, use the CREATE OPTICAL CLUSTER statement and make the **manu_code** column (also a column in the **catalog** table) the **cluster-key** column. Because **manu_code** contains nine unique values (ANZ, HRO, HSK, KAR, NKL, NRG, PRC, SHM, SMT), the photographs are stored in nine unique clusters, each containing the photographs for a single manufacturer. You assign a cluster size of 18,000 kilobytes to the cluster for the following reason: each photograph requires 60 kilobytes of storage, and you want to allow for as many as 300 photographs per manufacturer.

How Optical Clustering Occurs

Insert the first photograph into the database. As the insert operation begins, the Optical Subsystem checks whether the data is stored on an optical platter and whether an optical cluster exists for it. If so, the Optical Subsystem uses an internal optical-cluster table to find the current volume for the cluster-key value. If the cluster-key (**manu_code**) value for the first insert is HSK, the Optical Subsystem asks the optical-storage subsystem to reserve 18,000 kilobytes on the current volume for this cluster key. This first photograph is then written onto the volume.

The Optical Subsystem adds the cluster-key value, HSK, and the location of the HSK cluster to the cluster-size information that is already recorded in its internal optical-cluster table. The Optical Subsystem also records the fact that 60 kilobytes in the HSK cluster are now used. When the next unique cluster-key value, NRG for example, is inserted, the optical-storage subsystem reserves another 18,000 kilobytes on the current volume for the data associated with this cluster key. This process is repeated each time the optical-storage subsystem encounters a unique cluster-key value.

As HSK and NRG photographs fill their respective clusters, the internal optical-cluster table tracks the amount of space used in each cluster. Before a photograph is written to optical platter, the Optical Subsystem compares the size of the object with the amount of space that remains in its assigned cluster. If the cluster is too full to receive the photograph, the optical-storage subsystem creates a new cluster on the volume.

If all space on the current volume is filled or reserved, the optical-storage subsystem finds the next available volume in the family (a stand-alone optical-storage subsystem prompts you to mount an empty platter), and creates a new cluster on it for the cluster-key value. When the new cluster is created, the internal optical-cluster table is updated with the new volume number and the new value for space that remains in the current cluster. If an additional volume is not available in the family, an error is returned.

Support for Multiple Optical Virtual Processors

The Optical Subsystem now supports multiple optical virtual processors (VPs). To execute optical I/O requests on multiple drives in parallel, configure multiple optical VPs. You can fine-tune the number of optical VPs based on your system configuration and usage.

You can set the number of optical VPs equal to or greater than the number of optical drives on a multiprocessor system. The optical VPs are not bound to a particular optical drive. An idle optical VP can service requests for I/O on any optical drive.

Guidelines for Determining the Number of Optical Virtual Processors

The following guidelines can help you determine the number of optical VPS to configure for your Optical Subsystem. The optimal number of optical VPs depends on the type of system usage.

Guideline One: The maximum number of optical VPs is equal to the number of clients performing optical I/O concurrently.

Guideline Two: However, if several clients are performing optical I/O concurrently, then use at least ONE optical VP per drive.

Scenario One: Clients are requesting writes on the current optical volume in the same family. Multiple optical VPs for each drive allow additional clients to complete the initial setup while the first client is performing its I/O. The goal is to keep all the drives busy and maximize performance. If the optical objects are large, two optical VPs per drive are sufficient. If the optical objects are tiny (as in the case of signature verification), you need several optical VPs to keep the drives busy.

Scenario Two: Clients are requesting writes on the current optical volume in many different families. The write requests are randomly distributed across families. If the number of write requests is high, you would gain performance by having more optical VPs because you are more likely to get an I/O request for an already mounted platter before it is dismounted. Multiple optical VPs reduce the number of platter exchanges, a time-consuming operation.

Scenario Three: In a read-intensive environment, you should use one optical VP per optical drive.

Scenario Four: If the number of concurrent optical I/O request rates is low, a single optical VP for all optical drives might be adequate.

Calculating Logical-Log Space for Optical Data

When you install the Optical Subsystem, you do not need to calculate any logical-log space for optical TEXT and BYTE data. Dynamic Server does not write optical TEXT and BYTE data to either the logical log or physical log. Optical blobs are NOT written to the logical-log backup tapes when the logical logs are backed up.

For information on estimating the size of your logical log and physical log for tblspace TEXT and BYTE data, refer to your *Administrator's Guide*.

The Optical Subsystem API

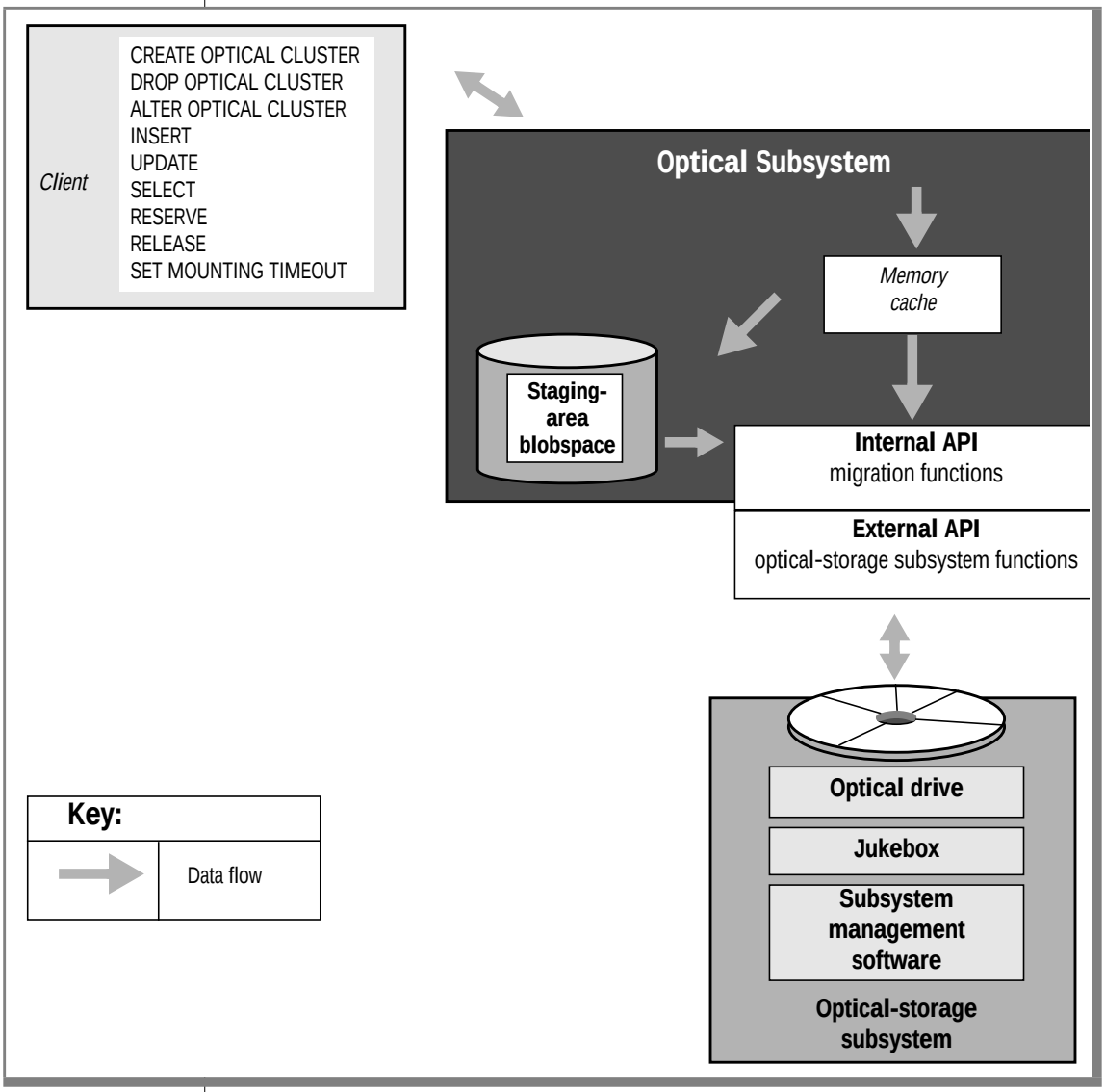
The API between the Optical Subsystem and the optical-storage subsystem consists of the following layers:

- An *internal* layer that represents the interaction between the Optical Subsystem and the API
- An *external* layer that represents the interaction between the API and the optical-storage subsystem

The optical-storage subsystem vendor provides the external layer of the interface in the form of a library that is linked to the Optical Subsystem during installation. For information on how to install the vendor library and how to link it with the Optical Subsystem, see Chapter 2, "Installation and Initial Configuration."

Figure 1-3 illustrates the relationships between the Optical Subsystem, the API, and the optical-storage subsystem.

Figure 1-3
Optical Subsystem, the API, the Staging-Area Blobspace, the Memory Cache, and the Optical-Storage Subsystem



The Memory Cache and Staging-Area Blobspace

Staging-area blobspace improves the performance of writing TEXT and BYTE data that is smaller than the cache size. The Optical Subsystem receives TEXT and BYTE data from the client application in 1-kilobyte pieces for a single row at a time. If the memory cache is full and cannot hold all pieces of TEXT and BYTE data, or if the cache is not being used, then the Optical Subsystem writes the TEXT and BYTE data to the staging-area blobspace. The Optical Subsystem uses the same memory cache for all client applications that are writing TEXT and BYTE data.

The system configuration parameter OPCACHEMAX, located in the ONCONFIG file, specifies the size of the memory cache in kilobytes. If the value is 0, the Optical Subsystem does not use the cache. You can set the client environment variable, **INFORMIXOPCACHE**, to restrict the amount of memory cache that the client application uses. The value for **INFORMIXOPCACHE** is specified in kilobytes and can be any value less than or equal to OPCACHEMAX. If **INFORMIXOPCACHE** is not set, the client application can use as much of the memory cache as is available.

The Optical Subsystem stores all TEXT or BYTE data for a given row in the memory cache until the last bit of TEXT or BYTE data in the row has been received or until the cache is filled. If TEXT or BYTE data in the row does not fit in the memory cache, it is flushed out of the cache to the staging-area blobspace. If the next column of TEXT or BYTE data in the row fits in the available space in the memory cache, it stays in the memory cache.

For example, imagine that you have a memory cache of 150 kilobytes, that the setting for the configuration parameter OPCACHEMAX is 150 kilobytes, and that the environment variable **INFORMIXOPCACHE** is not set. Your current row has three columns of TEXT data of the following sizes: 30 kilobytes, 180 kilobytes, and 70 kilobytes. The first TEXT data column is 30 kilobytes and the Optical Subsystem writes it to the memory cache. The second TEXT data column is 180 kilobytes and the Optical Subsystem tries to use the memory cache but the data is too large. The Optical Subsystem writes the TEXT data from the second column to the staging-area blobspace. The third column of TEXT data is 70 kilobytes and fits in the memory cache. The Optical Subsystem now outmigrates all three TEXT data columns in the row to the optical-storage subsystem in 50-kilobyte pieces in the order the TEXT data columns were processed: the first TEXT data from the memory cache; the second TEXT data from the staging-area blobspace; and the third TEXT data from the memory cache.

If you want to outmigrate a different number of bytes than the default of 50 kilobytes, you can configure the outmigration amount using the storage manager for the optical-storage subsystem. Figure 1-3 on page 1-14 illustrates the use of the memory cache and the staging area when TEXT and BYTE data is outmigrated to the optical-storage subsystem.

You create the staging-area blobspace through ON-Monitor, or with the **onspaces** utility, in the same way as any other Optical Subsystem blobspace. Before you use the optical-storage subsystem, you must perform the following tasks:

- Use ON-Monitor or a text editor to edit the ONCONFIG file and set the STAGEBLOB parameter to the name of the staging-area blobspace.
- Create the staging-area blobspace.
- Restart **oninit** to enable the Optical Subsystem to recognize the optical-storage subsystem.

Before you use the optical-storage subsystem, you might also want to perform the following tasks:

- Adjust the configuration parameter OPCACHEMAX to something other than the default (128 kilobytes).
- Specify the size for the **INFORMIXOPCACHE** environment variable in client environments.

Memory Cache

Memory for the memory cache is allocated as it is used. If OPCACHEMAX is set at 500 (500 kilobytes) and the largest TEXT or BYTE data is 50 kilobytes, the Optical Subsystem uses only 50 kilobytes for the memory cache.

Because the setting for the configuration parameter OPCACHEMAX is system wide, 100 percent of the memory cache is available if you are the only user. If additional users compete for use of the memory cache, more data is written to the staging-area blobspace. To properly assess the use of the memory cache, look at how many applications are processing TEXT or BYTE data objects and set the OPCACHEMAX to accommodate the largest data object.

The system configuration parameter OPCACHEMAX uses a default value of 128 kilobytes; however, you can set OPCACHEMAX to any value. If OPCACHEMAX is set to 0, the memory cache is not used and the Optical Subsystem writes all TEXT and BYTE data to the staging-area blobspace.

The Staging Area

The structure of the staging-area blobspace is the same as all other Optical Subsystem blobspaces. When it is created, the staging area consists of only one chunk, but more can be added as desired. Blobspace free-map pages manage the space. For information on how to create a blobspace, see your *Administrator's Guide*.

The optimal size for the staging-area blobspace depends on the following application factors:

- The frequency of data storage
- The frequency of data retrieval
- The average size of the data

To calculate the size of the staging-area blobspace, you must estimate the number of TEXT or BYTE data objects that you expect to reside in the staging-area blobspace simultaneously and multiply that number by the average size. In a single transaction, all TEXT or BYTE data that moves to the staging area is held in the staging area until the transaction completes. The number of TEXT or BYTE data objects that reside in the staging area simultaneously depends on the number of rows that are inserted at one time. The minimum size of the staging-area blobspace must be at least as large as the largest TEXT or BYTE data object that will reside in it. If the Optical Subsystem uses the staging-area blobspace, it writes an entire TEXT or BYTE data object to the staging area before it outmigrates the object to the optical-storage subsystem.

If a hardware failure occurs in the staging-area blobspace, the Optical Subsystem rolls the transaction back. The Optical Subsystem does not commit the transaction until it has written the data to the optical platter.

The staging-area blobspace cannot be mirrored.

The STAGEBLOB Parameter

You specify the stageblob blobspace with the STAGEBLOB onconfig parameter. The STAGEBLOB parameter has two values separated by a comma. The first value gives the name of the blobspace and the second value gives the number of optical VPs configured. In the following example, the *blobspace_name* is the name of the stageblob blobspace and 2 is the number of optical VPs configured. For information on configuring optical VPs, see "Support for Multiple Optical Virtual Processors" on page 1-11.

```
STAGEBLOB blobspace_name,2
```

In a UNIX environment, you can use either ON-Monitor or a text editor to edit the ONCONFIG file and set the STAGEBLOB parameter with the name of the staging-area blobspace. In a Windows NT environment, you can only use a text editor to edit the ONCONFIG file in order to set the STAGEBLOB parameter to a blobspace. For more information on setting the STAGEBLOB parameter, refer to your *Administrator's Guide*.

The presence of the STAGEBLOB parameter in the ONCONFIG file signals the Optical Subsystem that an optical-storage subsystem is present. If the Optical Subsystem does not find the STAGEBLOB parameter in the ONCONFIG file, it behaves as if no optical-storage subsystem existed. If the STAGEBLOB parameter is present but you have not created the staging-area blobspace, the Optical Subsystem displays the following message:

```
Invalid or missing name for Subsystem Staging Blobspace
```

You must create the staging-area blobspace and reinitialize the Optical Subsystem before it can migrate TEXT or BYTE data to the optical-storage subsystem. For the procedure to create the staging-area blobspace, refer to Chapter 2, "Installation and Initial Configuration."

The onstat -O Option

You can use the **-O** option of the **onstat** utility to monitor available and allocated resources for the memory cache and the staging-area blobspace. Figure 1-4 shows a sample of **onstat -O** output.

Figure 1-4
Example of onstat -O Option

```
Informix Dynamic Server Version 7.30.UN105 --On-Line-- Up 00:45:18 -- 11656 Kbytes

Optical StageBlob Cache
System Cache Totals:                                  System Blob Totals:
Size    Alloc.  Avail.              Number   Kbytes   Number  Kbytes
500     500     0                   1        20       3       1500

User Cache Totals:                                    User Blob Totals:
SID     User    Size                Number   Kbytes   Number  Kbytes
94      doug    250                 1        20       1       300
95      beth    500                 0        0        2       1200
```

The example shows that Doug wrote one 20-kilobyte TEXT or BYTE data to the memory cache. He also tried to write a 300-kilobyte TEXT or BYTE data object, but it did not fit because he is limited to 250 kilobytes of the memory cache. Beth can use the entire 500-kilobyte memory cache, but the two TEXT or BYTE data objects that she tried to write did not fit in the memory cache (1,200 kilobytes) and were written to the staging-area blobspace. No optical writes are in progress in Figure 1-4 (Avail = 0). No optical work was completed except for these two programs because the totals equal the sum of the two user entries (1,500 kilobytes). You can tell that the two programs are still connected because the entries are visible.

The API Internal Layer

The internal layer of the API manages the following operations:

- An *outmigration* operation that stores TEXT and BYTE data in the optical-storage subsystem
- An *inmigration* operation that retrieves TEXT and BYTE data from the optical-storage subsystem

Outmigration is the operation through which the Optical Subsystem directs the optical-storage subsystem to store TEXT and BYTE data on an optical platter. Outmigration activity occurs in the following sequence:

1. The Optical Subsystem stores the TEXT or BYTE data in the memory cache in 1-kilobyte pieces.
2. The Optical Subsystem stores TEXT or BYTE data that exceeds the memory cache in the staging-area blobspace.
3. The Optical Subsystem makes the size and cluster information about the TEXT or BYTE data available to the optical-storage subsystem.
4. The optical-storage subsystem finds and reserves space to store the TEXT or BYTE data.
5. The Optical Subsystem transfers the TEXT or BYTE data to the optical-storage subsystem.
6. The Optical Subsystem signals the end of data migration. Once the outmigration process ends, the staging-area blobpages that held the migrating data are marked *free*. In addition, the memory cache is flushed and available for use.
7. The family number, volume number, and byte-offset that the optical-storage subsystem uses to describe the storage location are passed back to the Optical Subsystem, where the information is stored in the TEXT or BYTE data descriptor in the data row. (The TEXT or BYTE data descriptor resides in a database table column. It is the pointer to the actual TEXT or BYTE data on the optical-storage subsystem.)

Because data cannot be rewritten to the same location on a WORM optical platter, when the Optical Subsystem updates TEXT or BYTE data, it outmigrates it to a new, clean location on the platter and updates the descriptor to reflect the new location. The optical-storage subsystem does not have the capability to reclaim space on the WORM platter after any TEXT or BYTE data on the platter is modified or deleted.

Inmigration is the operation through which the Optical Subsystem requests and gains access to TEXT or BYTE data that is stored in the optical-storage subsystem. Inmigration activity occurs in the following sequence:

1. The Optical Subsystem requests access to TEXT or BYTE data by supplying the optical-storage subsystem with the family name, volume number, and byte offset.
2. The optical-storage subsystem locates the TEXT or BYTE data and retrieves it in pieces. It deposits each portion of the TEXT or BYTE data in memory buffers that the Optical Subsystem can access.
3. The optical-storage subsystem signals the end of optical data migration.

Memory for the Optical Transfer Buffer

The memory for the optical transfer buffer is allocated from a special pool called *opool*. The optical transfer buffer transfers BYTE and TEXT data to and from the Optical Subsystem buffer.

To track memory usage, issue the **onstat -g mem opool** command. Figure 1-5 shows the output of the **onstat -g mem opool** command.

Figure 1-5
Example of onstat -g mem opool Option

```
Informix Dynamic Server Version 7.30.UN105--On-Line-Up 00:02:08--4968 Kbytes
Pool Summary:
           name   class  addr     totalsize  freesize  #allocfrag  #freefrag
           opool  V      a2ca018  8192       8072      1           1

           Blkpool Summary:
           name   class  addr     size       #blks
```

The API External Layer

The external layer of the API consists of a set of low-level functions that implement the interaction between the API and the optical-storage subsystem. These functions are compiled in a library called **libop.a** that the optical-storage subsystem vendor provides. The installation process links the **libop.a** library with the Optical Subsystem.

The Optical Subsystem Message File

Under certain exceptional conditions, the Optical Subsystem API writes messages to the message file. In general, the messages indicate that the Optical Subsystem made a request of the optical-storage subsystem and the optical-storage subsystem was unable to accomplish it. For the content of these messages and an appropriate action or response, if one is required, see the documentation from your optical-storage subsystem vendor. The MSGPATH variable in the ONCONFIG file specifies the UNIX pathname of the system message log file. The default value for MSGPATH is **/usr/informix/universal.log**. For more information about the message log file, refer to your *Administrator's Guide*.

The Optical Subsystem writes the following messages concerning the Optical Subsystem to the message log file:

```
oninit: invalid or missing name for Subsystem Staging
Blobspace
```

Cause: STAGEBLOB is specified in the configuration file, but the named dbspace does not exist. This message is normal the first time **oninit** is started with the Optical Subsystem.

Action: Use **onspaces** with the **-d** option to create the dbspace.

```
Optical Subsystem is running
```

Cause: You have set the value of the STAGEBLOB parameter in the configuration file, and the Optical Subsystem is communicating properly with the optical-storage subsystem.

Action None required.

`Optical Subsystem is not running`

Cause: You have set the value of the STAGEBLOB parameter in the configuration file, but the Optical Subsystem cannot detect the existence of the optical-storage subsystem.

Action: Check that the optical-storage subsystem is on-line.

`Optical Subsystem STARTUP Error`

Cause: The Optical Subsystem detects that the optical-storage subsystem is running, but the Optical Subsystem cannot communicate with it properly.

Action: Check your optical-storage subsystem for errors.

`Unable to initiate communication with the Optical Subsystem`

Cause: The optical driver that the optical-drive vendor supplies indicates that the drive is not accessible.

Action: Check the driver installation and the cabling between the computer and the drive.

Chapter

2

Installation and Initial Configuration

This chapter contains instructions for installing and configuring the Optical Subsystem. It contains the following sections:

- "Prerequisites" lists the procedures that your optical-storage subsystem vendor must complete before you can install the Optical Subsystem.
- "Installing the Optical Subsystem" provides detailed instructions on how to install the Optical Subsystem.
- "Creating the Staging Area" tells you how to set the STAGEBLOB parameter in the Optical Subsystem configuration file and how to create the staging-area blobspace.
- "Creating Optical Families" directs you to your vendor documentation for the optical-storage subsystem for instructions on how to create optical families and assign volumes to them.
- "Testing the Connection" directs you to perform two tests that indicate whether the connection between the Optical Subsystem and the optical-storage subsystem is operating correctly.

Prerequisites

You must obtain an optical-storage subsystem from an authorized optical-storage subsystem vendor before you can use your Informix Optical Subsystem. Contact your Informix sales representative to obtain a list of authorized optical-storage subsystem vendors.

Your vendor provides you with the support library for the optical-storage subsystem that is linked with the Optical Subsystem during the installation process. Your optical-storage subsystem vendor also provides you with optical-storage subsystem utilities that perform the following functions:

- Monitor the operation of the optical-storage subsystem
- Introduce and initialize new optical platters of the optical-storage subsystem
- Perform administrative and management tasks related to the optical-storage subsystem

Your optical-storage subsystem vendor should also perform the following tasks before you install the Optical Subsystem:

- Install the optical-storage subsystem hardware
- Configure the host-operating system to support the optical-storage subsystem
- Run diagnostics to ensure proper operation of the optical-storage subsystem on the host computer

Installing the Optical Subsystem

The Optical Subsystem is installed automatically when you install your database server. For instructions on how to install the database server, see your *Installation Guide*.

If you are installing your database server for the first time, configure and initialize the database server following the instructions that your *Administrator's Guide* gives.

UNIX

On a UNIX platform, the installation program makes a link from the optical shared library in **/usr/lib** to a dummy library in **$INFORMIXDIR/lib.** The name of the optical shared library differs from platform to platform. To find the name of the shared dummy library, refer to the machine notes in **$INFORMIXDIR/machine/en_us/0333 /**.

On the Sun Solaris platform, for example, the name of the shared library is **iosm07a.so**. You need to remove the link between the optical library **/usr/lib/iosm07a.so** and the dummy optical library in **$INFORMIXDIR/lib** and establish a link between **/usr/lib/iosm07a.so** and the shared library of your optical-storage subsystem. The following example assumes that you are using the Sun Solaris operating system. For the exact name of your optical shared library, refer to the vendor documentation for your optical-storage subsystem.

Become root:

```
su root
```

Change the directory to **/usr/lib**:

```
cd /usr/lib
```

Remove the link between **/usr/lib/iosm07a.so** and **$INFORMIXDIR/lib/iosm07a.so**:

```
rm iosm07a.so
```

Establish a link to the storage-manager shared library of your optical-storage subsystem:

```
ln -s /optical_storage_vendor_name/lib/shared_library_name
/usr/lib/iosm07a.so
```

♦

WIN NT

On a Windows NT platform, you would edit your ONCONFIG file to set the OPTICAL_LIB_PATH to the shared optical **dll** used. For example, if you use Plexus optical server manager installed on the **d** drive, you would set the following value in the ONCONFIG file:

```
OPTICAL_LIB_PATH d:\OSM\XDPSM\lib\libop.dll
```

♦

Continue with "Creating the Staging Area" before you access the optical-storage subsystem.

Creating the Staging Area

When TEXT or BYTE data moves from the Optical Subsystem environment to the optical-storage subsystem, the data is first directed to a memory cache or a blobspace on a magnetic disk, which is a *staging area*. Before using the optical-storage subsystem, the Optical Subsystem administrator must perform the following tasks:

- Name the staging-area blobspace
- Initialize the Optical Subsystem shared memory
- Create the staging-area blobspace

Naming the Staging Area on the STAGEBLOB Parameter

You must provide the name of the staging-area blobspace on the STAGEBLOB parameter.

UNIX

On a UNIX system, it is the first field of the value of the STAGEBLOB parameter in the ONCONFIG file:

```
STAGEBLOB blobspace_name,<num_optical_vps>
```

On a UNIX system, you can either use ON-Monitor to enter the name of the staging-area blobspace, or you can edit the ONCONFIG file using a text editor to set the value of the STAGEBLOB parameter. ♦

WIN NT

In a Windows NT environment, the name of the staging-area blobspace is the only value of the STAGEBLOB parameter in the ONCONFIG file:

```
STAGEBLOB blobspace_name
```

In a Windows NT environment, you can only edit the ONCONFIG file to set the STAGEBLOB parameter. For more information on setting ONCONFIG parameters, refer to your *Administrator's Guide*. ♦

UNIX

Configuring Optical Virtual Processors

On a UNIX platform, you can configure more than one optical VP. To set the number of optical VPs, use the second field of the value of the STAGEBLOB parameter in the ONCONFIG file. On a UNIX system, even though you can use either ON-Monitor or a text editor to edit the ONCONFIG file and set the STAGEBLOB parameter with the name of the staging-area blobspace, the only way to set the number of optical VPs is to use a text editor to edit the ONCONFIG file. The following example shows the format of the STAGEBLOB parameter:

```
STAGEBLOB blobspace_name,<num_optical_vps>
```

For example, if you want to use three optical VPs, specify the following values:

```
STAGEBLOB myblob,3
```

Do not leave spaces between the comma following the name of the blobspace and the number of optical VPs. For example, if you specify `STAGEBLOB myblob, 3`, the system will use the default of one optical VP instead of three optical VPs. For more information on setting the STAGEBLOB parameter, refer to your *Administrator's Guide*.

Initializing the Optical Subsystem

Initialize the Optical Subsystem shared memory. For specific information, see your *Administrator's Guide*. When you initialize the Optical Subsystem, the following error message appears on the screen:

```
Invalid or missing name for Subsystem Staging Blobspace
```

The Optical Subsystem is still initialized even though the staging-area blobspace has not yet been created. The staging-area blobspace name supplied as the STAGEBLOB parameter informs the Optical Subsystem that an optical-storage subsystem is present. You must create the staging-area blobspace, however, before you can outmigrate TEXT and BYTE data to the optical-storage subsystem.

Creating the Staging-Area Blobspace

The staging-area blobspace is the same as any other blobspace. On a UNIX system, you create the staging-area blobspace by using either ON-Monitor or the Optical Subsystem utility **onspaces**. In a Windows NT environment, you can use either the **onspaces** command-line utility or the **Spaces** icon in the Informix Enterprise Command Center. The name of the staging-area blobspace must be the same as the name that you provided on the STAGEBLOB parameter.

After you create the staging-area blobspace, you must reinitialize the Optical Subsystem.

For instructions on how to create a blobspace, see your *Administrator's Guide*.

UNIX

The following example shows the commands that you use to initialize the Optical Subsystem and create the staging-area blobspace on a UNIX system:

```
# Name the staging-area blobspace either through ON-Monitor
# or by editing the ONCONFIG file.
# Log in as informix
# Initialize the database server. If you are initializing your
# database server for the first time, do this using the -iy
# option.
# NOTE: Use oninit -iy ONLY on new systems. The -i option of
# oninit reinitializes Optical Subsystem and deletes all
# existing databases:
oninit -iy
# Wait for the database server to have a status of "online"
# before using onspaces. You can use onstat to check the
# status.
# Add staging-area blobspace:
touch /.../.../stageblob
onspaces -c -b stageblob -g 4 -p /stageblob -o 0 -s 10000
# Start new log:
onmode -l
# Backup logs:
ontape -a
# Do a level 0 archive:
ontape -s -L 0
# Bring down oninit:
onmode -yuk
# Bring up oninit:
oninit
```

♦

WIN NT

The following example shows the steps that you take to initialize the Optical Subsystem and create the staging-area blobspace on a Windows NT system:

```
# Name the staging-area blobspace by editing the ONCONFIG
# file.
# Log in as informix
# Initialize the database server.
Go to Control Panel and
double-click on the Services icon. Click on your database
server name in the Services window and click Start.
#If you are initializing the database server for the first
#time type the following parameters in the Startup Parameters
#window. (Use the -iy parameters ONLY on new systems. The -i
# parameter deletes all existing databases):
-iy
# Wait for database server to have a status of "Started"
# before using onspaces. You can use onstat to check the
# status.
# Create a file for the blobspace:
copy NUL c:\informixata\dbservername\stageblob
# Now create the blobspace:
onspaces -c -b stageblob -g 4 -p /stageblob -o 0 -s 10000
# Start new log:
onmode -l
# Backup logs:
ontape -a
# Do a level 0 archive
ontape -s -L 0
# Bring down oninit:
onmode -yuk
# Start the dbservername service:
Go to Control Panel and
double-click on the Services icon. Click on your database
server name in the Services window and click Start.
```

◆

Verifying the Presence of the Optical Subsystem

After configuration, check the Optical Subsystem message log file to make sure that your database server recognizes the Optical Subsystem. The following message should appear:

```
Optical subsystem is running.
```

If this message does not appear, check to make sure that the setup of the Optical Subsystem is correct and that the Optical Subsystem configuration is correct.

Creating Optical Families

TEXT and BYTE data that is outmigrated to the optical-storage subsystem is stored on a volume in an optical family that you assign using the CREATE TABLE statement. Before you can assign TEXT and BYTE data to an optical family, however, you must use the utilities that the vendor of your optical-storage subsystem provides to create an optical family and assign volumes to it. For the procedure to create an optical family, see the documentation that your optical-storage subsystem vendor provides.

Testing the Connection

The following tests ensure that you followed the relink and install procedures correctly and that the connection between the Optical Subsystem and the optical-storage system is operational.

Test One

Run the optical utilities of the optical-storage subsystem vendor that monitor the operation of the optical-storage subsystem and provide the options for platter management.

The Optical Subsystem log file, located either in the directory that **INFORMIXDIR** is set to or in a user-specified path, should indicate no abnormalities.

Test Two

Start the database server while the optical-storage subsystem is up. Both the log file for the Optical Subsystem and the log file for the optical-storage subsystem should indicate normal operations. The log file for the Optical Subsystem is located either in the directory that **INFORMIXDIR** is set to or the directory that the MSGPATH configuration parameter specifies in the **onconfig** file.

The message `Optical Subsystem is running` appears in the log file of the Optical Subsystem if the optical-storage subsystem is operating normally. For the name of the log file for the optical-storage subsystem and the messages that should appear in it, see the vendor documentation for your optical-storage subsystem.

Chapter

3

Using the Optical Subsystem

This chapter illustrates how the SQL extensions that the Optical Subsystem provides support access to WORM optical media. It provides examples that show you how to perform the following functions:

- Assign a TEXT or BYTE data column to an optical platter (CREATE TABLE).
- Create an optical cluster (CREATE OPTICAL CLUSTER).
- Reserve an optical volume (RESERVE).
- Release an optical volume (RELEASE).
- Set the mounting time-out (SET MOUNTING TIMEOUT).
- Locate the optical volume where a TEXT or BYTE column is stored (FAMILY() and VOLUME() functions).
- Use TEXT or BYTE data descriptors to enable multiple tables to share the same TEXT and BYTE data (DESCR() function).
- Migrate TEXT and BYTE data in an optical family from one Optical Subsystem to another (**HPL**, **dbexport** and **dbimport**).

The examples use the **cat_descr** and **cat_picture** TEXT and BYTE columns from the **catalog** table of the **stores7** database. You can use the optical SQL statements from DB-Access or from a program developed with one of the Informix SQL APIs, such as INFORMIX-ESQL/C.

Assigning TEXT and BYTE Columns to an Optical Platter

To tell the Optical Subsystem that a TEXT or BYTE column is to be stored on an optical platter, specify an optical *family name*, rather than a dbspace or blobspace name, in the column-definition portion of the CREATE TABLE statement. The administrator for the optical-storage subsystem must create the optical family name in the optical-storage subsystem before you can execute a CREATE TABLE statement that refers to it. For the procedure to create the optical family name, see your subsystem vendor documentation.

In the following example, the TEXT column **cat_descr** of the **catalog** table is stored with the table in a dbspace and the BYTE column **cat_picture** is stored in the optical family **catalog_stores7**:

```
CREATE TABLE catalog
     (
     catalog_num     SERIAL (10001),
     stock_num       SMALLINT NOT NULL,
     manu_code       CHAR(3) NOT NULL,
     cat_descr       TEXT IN TABLE,
     cat_picture     BYTE IN catalog_stores7,
     cat_advert      VARCHAR (255, 65)
     )
     IN dbspace10
```

When the statement is executed, the Optical Subsystem checks to determine whether **catalog_stores7** is a dbspace, a blobspace, or an optical family name. Either of the following conditions returns an error:

- The name refers to a blobspace or dbspace *and* an optical family name.
- The name does not refer to either a blobspace, dbspace, *or* optical family name.

Once you create the table, you can insert TEXT and BYTE data in an optical family in the following ways:

- With the LOAD statement (DB-Access) or the **dbload** utility
- From locator variables (INFORMIX-ESQL/C)

Reading TEXT and BYTE Data Columns from an Optical Platter

Use the SELECT statement to read a TEXT or BYTE data column from an optical platter in the same way that you read a TEXT or BYTE data column from a magnetic disk. The syntax of the SELECT statement does not change to support access to an optical-storage subsystem. However, the Optical Subsystem supports extensions to SQL that were developed exclusively for use with optical storage. These additional SQL statements accomplish the following tasks that affect how efficiently a TEXT or BYTE data column is read from the optical platter:

- Clustering TEXT and BYTE data
- Managing optical volumes

Clustering TEXT and BYTE Data

You can use the CREATE OPTICAL CLUSTER statement to store logically related TEXT and BYTE data on the same volume of an optical platter. Storing related TEXT and BYTE data together minimizes platter exchanges when the TEXT and BYTE data is retrieved. The CREATE OPTICAL CLUSTER statement requires you to perform the following steps:

- Assign a name to the optical cluster.
- Specify the TEXT or BYTE data columns in a table that are to be stored together in the optical cluster.
- Specify the cluster key.

  The cluster key consists of one or more columns in the table that define the logical grouping of the TEXT or BYTE data columns. The columns that make up the cluster key must not be TEXT or BYTE data columns.
- Specify the size of the cluster in kilobytes.

The following CREATE OPTICAL CLUSTER statement allocates a cluster of 18,000 kilobytes on an optical platter for the **cat_picture** BYTE column in the **catalog** table. It assigns the name **catalog_catpics** to the cluster.

```
CREATE OPTICAL CLUSTER catalog_catpics
    FOR catalog (cat_picture)
    ON (manu_code)
    CLUSTERSIZE 18000
```

The cluster key for **catalog_catpics** is **manu_code**. Therefore the **cat_picture** TEXT and BYTE data that have the same value in the **manu_code** column are stored together on the same volume or volumes, if more than one is required.

Choosing the Cluster Key

In the preceding example, the **manu_code** column was chosen for the cluster key because, of all the columns in the **catalog** table, it best meets the following criteria for choosing a cluster key:

- It provides a logical order for retrieving the TEXT and BYTE data.
- It contains duplicate values.

The first criterion for a cluster key, which can consist of more than one column, is that it provides a logical order for retrieving the TEXT and BYTE data. The purpose of the cluster key is to optimize retrieval of the TEXT and BYTE data by grouping related TEXT and BYTE data on the same volume. Grouping TEXT and BYTE data by a cluster key is only advantageous, however, if the TEXT and BYTE data is normally retrieved in the order that it is grouped. If the **cat_picture** TEXT and BYTE data is grouped by **stock_num** and then retrieved by **manu_code**, as the following example shows, the optical-storage subsystem might still need to perform multiple platter exchanges to retrieve the TEXT and BYTE data:

```
SELECT cat_picture FROM catalog WHERE manu_code = 'HRO'
```

The second criterion for a cluster key is that it must contain duplicate values. Of all the columns in the **catalog** table, only the first three, **catalog_num**, **stock_num**, and **manu_code**, provide logical sequences for accessing the table. Of these columns, **catalog_num** is not useful because it has a SERIAL data type, which produces a unique value for each row in the table. Clustering on this column produces a cluster for every row in the table where **cat_picture** is not null.

Assume, for example, that 26 rows in the **catalog** table have pictures in the **cat_picture** column, and that the **manu_code** column has seven unique values with occurrences as the following example shows:

```
manu_code       occurrences
ANZ                   7
HRO                   6
KAR                   1
NRG                   1
PRC                   4
SHM                   5
SMT                   2
```

For these characteristics, **manu_code** produces seven unique clusters with the same number of TEXT and BYTE data objects in each cluster per occurrence of each key. By contrast, assume that 26 unique values exist in the same 26 rows, only three of which occur multiple times (104 x 2, 110 x 4, 205 x 3). In this case, choosing **stock_num** as the cluster key produces 21 clusters, 17 of which contain only one TEXT and BYTE data object.

Choosing the Cluster Size

The cluster size is the size of the cluster expressed in kilobytes. It is based on the number of TEXT and BYTE data objects that you expect to store in each cluster and the average size of the TEXT and BYTE data.You can calculate it as follows:

```
cluster size = estimated number of TEXT and BYTE data objects *
    average TEXT and BYTE data size
```

If the average size of the **cat_picture** images is 60 kilobytes, and you want to store up to 300 TEXT and BYTE data objects per cluster, the cluster size is 18,000 kilobytes.

Altering the Cluster Size

You can use the ALTER OPTICAL CLUSTER statement to change the size of an optical cluster for all clusters that are created after the statement is executed. In the following example, the size of the **catalog_catpics** cluster is reduced to 6,000 kilobytes, a cluster size that stores up to one hundred 60-kilobyte TEXT or BYTE data objects per cluster:

```
ALTER OPTICAL CLUSTER catalog_catpics CLUSTERSIZE 6000
```

The ALTER OPTICAL CLUSTER statement changes only the cluster size for TEXT or BYTE data objects created after the statement is executed. Clusters that were created previously remain their original size.

Dropping an Optical Cluster

You can use the DROP OPTICAL CLUSTER statement to terminate optical clustering. The following example terminates clustering for **catalog_catpics**:

```
DROP OPTICAL CLUSTER catalog_catpics
```

The DROP OPTICAL CLUSTER statement does not affect the TEXT and BYTE data objects that were already stored in a cluster. It terminates clustering only for the affected columns in the future.

You can also use the DROP OPTICAL CLUSTER statement with the CREATE OPTICAL CLUSTER statement to change either the TEXT or BYTE data columns or the cluster-key columns for an optical cluster. For example, you can change the cluster key for **catalog_catpics** from **manu_code** to **stock_num** with the following statements:

```
DROP OPTICAL CLUSTER catalog_catpics;
CREATE OPTICAL CLUSTER catalog_catpics
    FOR catalog (cat_picture)
    ON (stock_num)
    CLUSTERSIZE 3000
```

These statements do not affect TEXT and BYTE data objects that were previously clustered on **manu_code** because you cannot alter data written to a WORM optical platter. These statements change the clustering only for future inserts or updates. If you want to recluster the TEXT and BYTE data objects that were clustered by **manu_code**, you must update them. The TEXT and BYTE data objects are then rewritten to the new cluster. The TEXT or BYTE data descriptors are updated with the new location of the TEXT and BYTE data objects. The space that these TEXT and BYTE data objects originally occupied is lost because you cannot reuse space on WORM optical media.

Managing Volumes on the Optical Drives

Your application can perform the following operations to manage the volumes on the optical drives and, consequently, affect the performance of the application:

- Reserve an optical volume to keep it mounted during a set of operations
- Release a reserve request to free an optical drive
- Set the time-out period for mounting an optical volume

Reserving an Optical Volume

You can use the RESERVE statement to keep a required optical volume mounted on an optical drive while you periodically access it during a set of operations. The RESERVE statement implies a request to mount the specified volume if it is not currently mounted.

When multiple users access the optical platters simultaneously, you might need to reserve a volume if your application accesses it consecutively. Otherwise, individual accesses are interleaved with requests from other applications so that each access by your application could require the volume to be remounted on the optical drive. Not reserving a volume results in an unsatisfactory response time from the optical-storage subsystem.

For each RESERVE request, the optical-storage subsystem adds the value one (1) to a reserve counter for that volume. If the value of the reserve counter is greater than zero, you cannot remove the volume. You can issue multiple RESERVE requests for a volume, but you must release each one before you can remove the volume from the drive. When you finish accessing the volume, use the RELEASE statement to release it. For details, see "Releasing a Reserved Optical Volume."

If all the optical drives are reserved when you issue a RESERVE statement, the optical-storage subsystem waits for the number of seconds that the SET MOUNTING TIMEOUT statement specifies or for the default time-out period.

You can use a SELECT statement to obtain the family name and volume number of the volume that you want to reserve. The FAMILY() and VOLUME() function expressions return the family name and volume number, respectively. In the following example, the SELECT statement uses the FAMILY() and VOLUME() functions to identify the volume that contains pictures of stock for manufacturer HRO. The RESERVE statement then reserves the volume, using the family name and volume number that the SELECT statement returned.

```
SELECT FAMILY(cat_picture), VOLUME(cat_picture)
    FROM catalog
    WHERE manu_code = 'HRO'
:
RESERVE 'catalog_1991' 013
```

Releasing a Reserved Optical Volume

You must release each RESERVE request for an optical volume with the RELEASE statement. The RELEASE statement cancels the latest RESERVE request for an optical volume by subtracting the value one (1) from the reserve counter for that volume. When the reserve counter is reduced to zero, you can remove the optical volume from the drive. The following statement releases volume 013 of the optical family:

```
RELEASE 'catalog_1991' 013
```

An error is returned if you issue a RELEASE statement for a volume that is not reserved.

The vendor of your optical-storage subsystem supplies a utility that you can also use to release a volume. The optical-storage subsystem administrator must release a volume with this utility when a program terminates prior to releasing a reserved volume.

Setting the Mounting Time-Out

The SET MOUNTING TIMEOUT statement lets you control the amount of time that your application waits for a volume to be mounted before it abandons the request to continue processing. When you request access to an optical volume, the optical-storage subsystem cannot mount it immediately if all the optical drives are busy processing requests from other users. In addition, one or more drives might be reserved at the time of your request. If a SET MOUNTING TIMEOUT statement is not executed, the optical-storage subsystem times out after five minutes (300 seconds) if it cannot mount the requested volume.

The SET MOUNTING TIMEOUT statement provides the following options for setting the time-out period:

- Wait indefinitely until a drive is available for the requested volume
- Abort the request for a volume immediately if it is not currently mounted on an optical drive
- Specify a number of seconds to wait for a volume to be mounted
- Wait for a volume to be mounted only if a drive is available

If it is mandatory that a particular optical volume be mounted before an application continues, it might be appropriate for the application to wait an indefinite amount of time for that volume to mount. If so, the following statement has that effect:

```
SET MOUNTING TIMEOUT TO WAIT
```

If multiple users and multiple applications are competing for the optical drives, you might not always want your application to wait for a volume that is not currently mounted. You might set the mounting time-out to NOT WAIT. For example, if your application retrieves data about an item of merchandise and the TEXT or BYTE data on an optical platter is desirable but not required, the application might not display the data if it is not immediately available.

The following example causes a request for a volume to abort immediately if the volume is not currently mounted:

```
SET MOUNTING TIMEOUT TO NOT WAIT
```

Normally, you should set the time-out period to the number of seconds that is appropriate for the application to wait for a volume. The following statement sets the mounting time-out period to 30 seconds. If the volume is not mounted within 30 seconds, the process times out and control is returned to the application.

```
SET MOUNTING TIMEOUT TO 30
```

Set the mounting time-out period to zero seconds to cause the process to wait for a volume to be mounted only if a drive is currently available, as follows:

```
SET MOUNTING TIMEOUT TO 0
```

When you specify a time-out period of zero, the optical-storage subsystem waits for a volume to mount if a drive is currently available for it. When you specify the NOT WAIT option, the optical-storage subsystem does not wait for the requested volume to mount if it is not currently on a drive.

Using TEXT and BYTE Data Descriptors

Data rows that include TEXT or BYTE data do not include the TEXT or BYTE data in the row itself. Instead, the data row contains a 56-byte TEXT or BYTE data descriptor that includes a forward pointer (rowid) to the location where the first segment of data is stored. The descriptor can point to a dbspace blobpage, a blobspace blobpage, or an optical-storage location.

You can use the DESCR() function expression to retrieve the TEXT or BYTE data descriptor for TEXT or BYTE data that is stored on WORM optical media. The DESCR() function returns the descriptor in an encoded format that is 112 bytes long. Using the DESCR() function as a column value in INSERT and UPDATE statements enables you to create additional pointers to existing TEXT and BYTE data objects on the optical media. Creating additional pointers to TEXT and BYTE data objects saves space on the optical platters by allowing multiple tables to refer to the same physical TEXT and BYTE data.

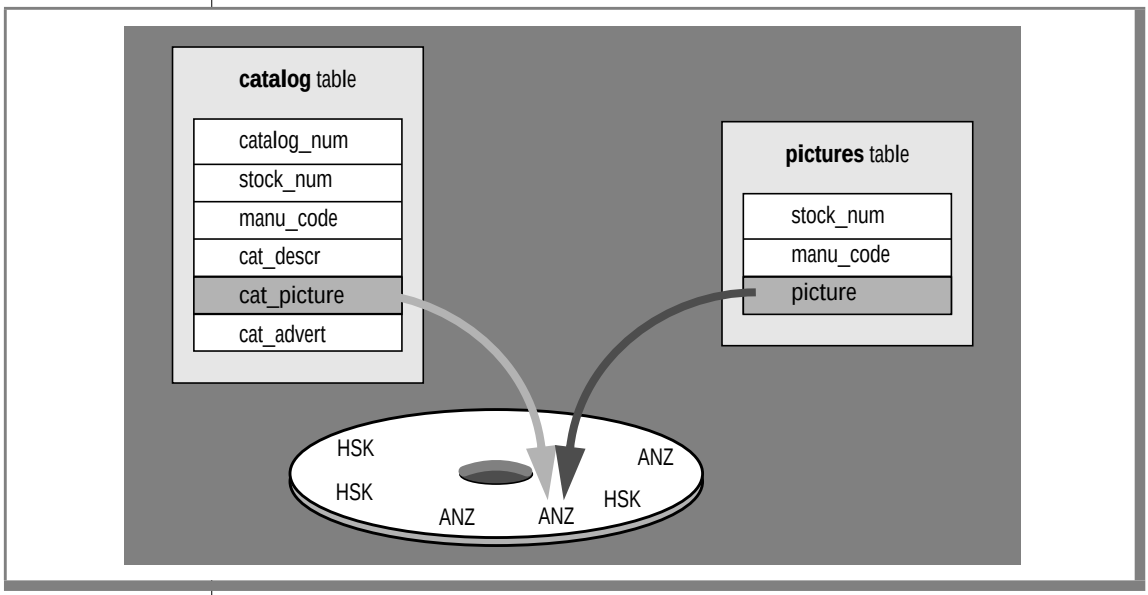
Before the TEXT or BYTE data descriptor is inserted in a data row, it is decoded and validated to verify consistency and legitimacy. The optical-storage subsystem performs all encoding, decoding, and validations through the Optical Subsystem calls to API functions. If the validation process fails, the insert operation fails.

The following example uses DESCR() in an INSERT statement to insert TEXT or BYTE data descriptors, or pointers to existing TEXT and BYTE data objects, in the following hypothetical **pictures** table. Assume that the **picture** column is a BYTE column assigned to an optical platter.

```
INSERT INTO pictures (stock_num, manu_code, picture)
    SELECT stock_num, manu_code, DESCR(cat_picture)
        FROM catalog
        WHERE manu_code = 'ANZ'
```

Figure 3-1 illustrates the result of the preceding INSERT statement for a row in the **pictures** table.

Figure 3-1
Two Descriptors That Point to the Same TEXT or BYTE Data Object on a WORM Platter



The following example uses DESCR() to update the **pictures** table with all pictures from the **cat_picture** column of the **catalog** table:

```
UPDATE pictures
    SET (picture) =
        (SELECT DESCR(cat_picture)
            FROM catalog
            WHERE pictures.stock_num = catalog.stock_num
            AND pictures.manu_code = catalog.manu_code
            AND cat_picture IS NOT NULL)
```

You can use the DESCR() function expression for TEXT or BYTE data columns that are stored on an optical platter. An error is returned if you use DESCR() on a nonoptical TEXT or BYTE data column.

Locating Columns Stored in Optical Families

To determine which columns in a database are stored on an optical platter, query the **sysblobs** table. The **sysblobs** table contains four columns that provide the following information:

- The space name (blobspace, dbspace, or family name) where the column is stored
- The media type (M = magnetic; O = optical)
- The table identification number
- The column number within the table

The following SELECT statement shows the locations of all TEXT or BYTE data columns in a given database:

```
SELECT * FROM sysblobs
```

The output that the preceding SELECT statement produces for the **stores7** database might indicate that column number 5 in table 109 is stored in the optical family **family1**.

```
spacename         type           tabid  colno
rootdbs             M             109      4
family1             O             109      5
```

With this information, you can query the **syscolumns** table to obtain the name of the column, as follows:

```
SELECT colname FROM syscolumns WHERE tabid = 109 AND colno = 5
```

That particular query produces the following output:

```
colname
cat_picture
```

The **oncheck** utility with the **-pD** option also tells you whether a TEXT or BYTE data column is stored on an optical platter. However, the information is displayed on a row-by-row basis**.** The following **oncheck** command displays information for rows in the **catalog** table of the **stores7** database**:**

```
oncheck -pD stores7:catalog
```

For more information about **oncheck,** refer to your *Administrator's Guide*.

Locating the Optical Volume Where TEXT and BYTE Data Is Stored

You can use the FAMILY() and VOLUME() function expressions in a SELECT statement to obtain the family name and volume number where a TEXT and BYTE data object is stored. The SELECT statement in the following example produces a list of the optical volumes that hold pictures of bicycle helmets (**stock_num** = 110):

```
SELECT FAMILY(cat_picture), VOLUME(cat_picture)
    FROM catalog
    WHERE stock_num = 110
```

Migrating a Database with TEXT and BYTE Data on an Optical Platter

When a database contains TEXT and BYTE data that is stored in an optical family, you can migrate the database from one Optical Subsystem to another by using the following utilities:

- **High-Performance Loader (HPL)** or **onpload**
- **dbexport** and **dbimport**

This section documents the operation of these programs with respect to migrating a database that contains TEXT and BYTE data stored on an optical platter. For a description of **dbexport** and **dbimport**, see the *Informix Migration Guide*. For a description of the **HPL** and **onpload** utilities, see the *Guide to the High-Performance Loader*.

If you are migrating a database that contains TEXT and BYTE data from one Optical Subsystem to another, check your vendor documentation for the compatibility requirements among optical-storage subsystems. For example, the destination optical-storage subsystem might need to include the same TEXT or BYTE data family name as the one that the source optical-storage subsystem uses.

Using HPL to Unload and Load Optical Data

You can use the **HPL** or the **onpload** utility to migrate optical TEXT and BYTE data. The **onpload** utility is the command-line version of the **HPL** graphical user interface (**ipload**). With **HPL** you can either load or unload a complete table or a partial table. For example, you can unload selected columns or rows that contain optical TEXT and BYTE data and insert them into a table in another database.

For instructions on how to use the **HPL** and **onpload** utilities, see the *Guide to the High-Performance Loader*.

The dbexport and dbimport Utilities

The **dbexport** utility unloads a database into ASCII files; the **dbimport** utility creates a database from ASCII files. You can use the **-d** option of the **dbexport** utility to specify that only the TEXT or BYTE data descriptor is written for a TEXT or BYTE data column that is stored on an optical platter. Migrating the descriptor permits a TEXT and BYTE data object that is stored on an optical platter to be shared in multiple databases, eliminating the need to store duplicate copies of it on an optical platter. The TEXT or BYTE data descriptor contains information about the location and size of the data. It holds all the information that is necessary to retrieve a TEXT or BYTE data object from an optical platter. If you do not specify the **-d** option, **dbexport** exports both the descriptor and the TEXT and BYTE data object.

If you want a different cluster arrangement in the destination database and you can afford the space on an optical platter, you can choose to duplicate the TEXT and BYTE data in a different cluster.

The following **dbexport** command exports the **stores7** database to the **./exstores7** directory, exporting only descriptors for tables that contain TEXT or BYTE data columns. (The **-c** instructs the program to ignore all errors except fatal errors, **-o** specifies the output directory, and **-q** suppresses the echo of SQL statements.)

```
dbexport -c -d -o ./exstores7 -q stores7
```

TEXT or BYTE data descriptors remain valid from one Optical Subsystem to another because each volume on an optical platter contains an internal tracking label. In addition to TEXT or BYTE data, each WORM volume contains a label with the following information:

- Family name
- Family number (also referred to as the optical-storage subsystem serial number)
- Volume number
- Optical-storage-subsystem identifier

Each optical-storage-subsystem device, whether a stand-alone drive or jukebox, has a unique optical-storage-subsystem identifier. The optical-storage subsystem uses the identifier and the family name to uniquely identify the family within the optical-storage subsystem. Using the identifier and the family name becomes an important issue in a distributed database environment where more than one optical-storage subsystem might be available.

Chapter

4

SQL for the Optical Subsystem

This chapter describes the syntax and usage of the individual SQL statements that the Optical Subsystem provides to support optical-storage subsystems. The syntax of each statement is illustrated in a diagram that shows the sequence of the required and optional elements. For a complete description of the conventions used in these diagrams, see the "Introduction." The SQL statements presented in this chapter are used exclusively with optical-storage subsystems. For additional information on the context in which each statement is used, see Chapter 3, "Using the Optical Subsystem."

This chapter describes the following SQL statements:

- ALTER OPTICAL CLUSTER
- CREATE OPTICAL CLUSTER
- DROP OPTICAL CLUSTER
- RELEASE
- RESERVE
- SET MOUNTING TIMEOUT

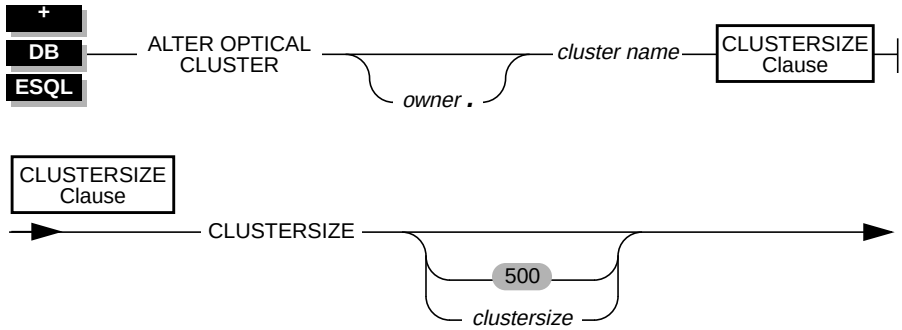
This chapter also describes the following function expressions:

- DESCR()
- FAMILY()
- VOLUME()

ALTER OPTICAL CLUSTER

Use the ALTER OPTICAL CLUSTER statement to alter the size of an optical cluster.

Syntax



Element	Purpose	Restrictions	Syntax
cluster name	The name of the cluster to alter	The cluster name must already exist	Identifier segment, see *Informix Guide to SQL: Syntax*
clustersize	Size of the cluster, in kilobytes, to be reserved for each unique cluster-key value	Must be less than the volume size	Literal Number segment, see *Informix Guide to SQL: Syntax*
owner	The user name of the owner of the cluster	The specified name must be a valid user name	Identifier segment, see *Informix Guide to SQL: Syntax*

Usage

To alter an optical cluster, you must be the owner of the cluster, have the Index privilege on the table, or have the DBA privilege.

In the following example, the CREATE OPTICAL CLUSTER statement creates **cat_clstr**, an optical cluster of 6,000 kilobytes for the **cat_picture** column in the **catalog** table. For each unique value of **stock_num** in the table, **cat_clstr** stores the associated data for this column. The following example changes the amount of space allocated for each **cat_clstr** from 6,000 kilobytes to 8,000 kilobytes. This change affects all clusters created after the ALTER OPTICAL CLUSTER statement is executed. Clusters created prior to an ALTER OPTICAL CLUSTER statement maintain their original size.

```
CREATE OPTICAL CLUSTER cat_clstr
     FOR catalog (cat_picture)
     ON (stock_num)
     CLUSTERSIZE 6000
ALTER OPTICAL CLUSTER cat_clstr CLUSTERSIZE 8000
```

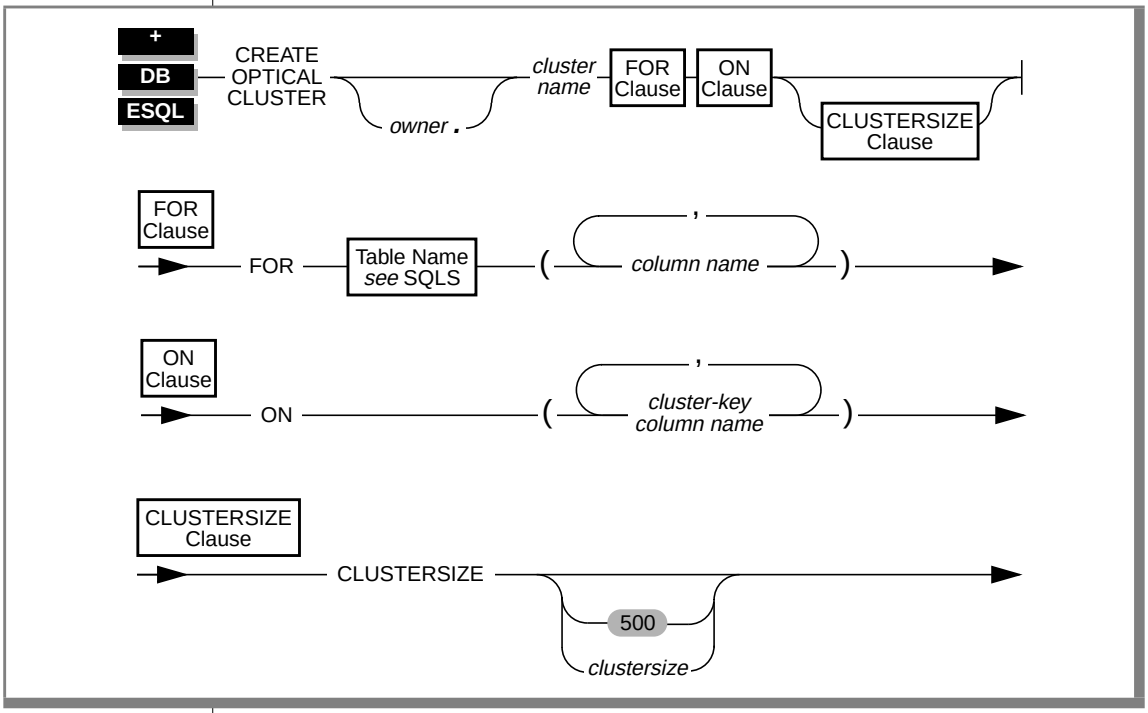
References

See the CREATE OPTICAL CLUSTER statement on page 4-6 and the DROP OPTICAL CLUSTER statement on page 4-10.

CREATE OPTICAL CLUSTER

Use the CREATE OPTICAL CLUSTER statement to create a cluster on an optical platter for one or more logically related TEXT or BYTE data columns.

Syntax



Element	Purpose	Restrictions	Syntax
column name	The name of the TEXT or BYTE data column to be clustered	You can assign up to 16 distinct columns to a cluster.	Identifier segment, see *Informix Guide to SQL: Syntax*
cluster-key column name	The name of the column that is the cluster key	You can specify up to 16 columns to create a composite cluster key. You cannot specify a TEXT or BYTE data column as part of a cluster key.	Identifier segment, see *Informix Guide to SQL: Syntax*
cluster name	The name of the optical cluster	The name must be unique.	Identifier segment, see *Informix Guide to SQL: Syntax*
clustersize	The amount of optical platter space to allocate for the cluster, specified in kilobytes. If no clustersize is specified, the size of the cluster defaults to 500 kilobytes.	The clustersize must be less than the volume size.	Expression segment, see *Informix Guide to SQL: Syntax*
owner	The valid user name of the cluster	The specified name must be a valid user name.	Identifier segment, see *Informix Guide to SQL: Syntax*

Usage

An optical cluster enables logically related TEXT or BYTE data to be stored on the same volume, which minimizes time-consuming platter exchanges during retrieval.

You can create (and, by default, become owner of) an optical cluster if you are the owner of the table, have the Resource privilege on the database and the Index privilege on the table, or have the DBA privilege on the database.

Only a person with DBA privileges can create the optical cluster and specify another user as the owner of the cluster.

TEXT and BYTE Data Columns

The following restrictions are placed on the TEXT or BYTE data columns that are being clustered on the optical platter:

- All columns within a single clustering strategy must reside on the same optical family.
- A single column cannot be clustered more than once.

Cluster-Key Columns

A cluster key can be a single column (for example, **stock_num**) or a composite of up to 16 columns (for example, **stock_num**, **manu_code**). You cannot specify a TEXT or BYTE data column as a cluster key. The maximum length of a cluster key is 256 bytes. An error message is returned if you submit a cluster key that is longer than 256 bytes.

You can gain better performance if you organize the columns in a composite cluster key so that the column with the fewest number of duplicate values is placed first.

Clustersize

The *clustersize* is the size, in kilobytes, of the space reserved on an optical volume for each cluster with a unique cluster-key value. The size of the cluster can be enlarged beyond this size, if necessary, but it cannot exceed the size of a volume. If you do not specify *clustersize*, the default size of the cluster is 500 kilobytes.

You can create more than one optical cluster for the same list of cluster-key columns. This option differs from the restrictions placed on indexes, each of which you must create from a unique set of index-key columns.

Example

Assume that the CREATE TABLE statement for the **catalog** table in the **stores7** database specifies that the columns **cat_descr** and **cat_picture** are to be stored in the same optical family. In the following example, the CREATE OPTICAL CLUSTER statement creates **cat_clstr**, an optical cluster of 6,000 kilobytes. A new **cat_clstr** is allocated to store the data for **cat_descr** and **cat_picture** each time a row is inserted with a unique value for **manu_code**.

```
CREATE OPTICAL CLUSTER cat_clstr
     FOR catalog (cat_picture, cat_descr)
     ON (manu_code)
     CLUSTERSIZE 6000
```

If the **stores7** database contains six unique values for **manu_code**, the statement lays the groundwork for the optical-storage subsystem to create six optical clusters, one for each value. If the table contains ten rows for one **manu_code**, then the **cat_clstr** for that value contains the **cat_descr** and **cat_picture** data for all ten rows.

To create the cluster, the optical-storage subsystem reserves 6,000 kilobytes on the current volume. If the current volume is too full to accept the space reservation, the cluster is created on the next available volume in the family. The location and size of each cluster on the **manu_code** cluster key is tracked in an internal optical-cluster table. The name of the optical cluster, the number of its TEXT and BYTE data columns, and the cluster-key column numbers are stored in the **sysopclstr** table.

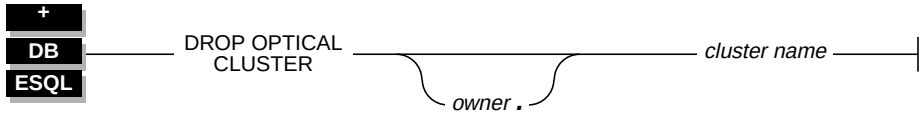
References

See the ALTER OPTICAL CLUSTER statement on page 4-4 and the DROP OPTICAL CLUSTER statement on page 4-10. In the *Informix Guide to SQL: Syntax*, see the CREATE TABLE statement.

DROP OPTICAL CLUSTER

Use the DROP OPTICAL CLUSTER statement to terminate optical clustering for the TEXT or BYTE data columns that are associated with the cluster name.

Syntax



Element	Purpose	Restrictions	Syntax
cluster name	The name of the optical cluster	The cluster name must already exist.	Identifier segment, see *Informix Guide to SQL: Syntax*
owner	The user name of the owner of the cluster	The specified name must be a valid user name.	Identifier segment, see *Informix Guide to SQL: Syntax*

Usage

The DROP OPTICAL CLUSTER statement terminates clustering on the optical platter for the TEXT or BYTE data columns associated with *cluster name* and drops the internal optical cluster table. After the DROP OPTICAL CLUSTER statement is executed, the data items that are associated with the cluster name are stored on the current volume; they are stored in the sequence that they are output to the optical-storage subsystem.

You must be the owner of the optical cluster or have the DBA privilege to drop the optical cluster.

To change the cluster-key columns of an optical cluster, drop the old optical cluster and create a new optical cluster with the new cluster-key columns. You can change your optical clustering strategy so that the **cat_picture** and **cat_descr** columns of the **catalog** table are clustered according to **stock_num** instead of **manu_code**, as follows:

```
DROP OPTICAL CLUSTER cat_clstr
CREATE OPTICAL CLUSTER cat_stock_clstr
     FOR catalog (cat_picture, cat_descr)
     ON (stock_num)
     CLUSTERSIZE 6000
```

After the new optical cluster, **cat_stock_clstr**, is created on **stock_num**, all data that is inserted into the database for the **cat_picture** and **cat_descr** columns is optically clustered according to the associated value of **stock_num**. However, the new cluster does not alter the clustering of TEXT and BYTE data already stored on the WORM optical platters. A new cluster affects only subsequent outmigrations. Therefore, in this example, the TEXT and BYTE data that **cat_clstr** previously clustered retains its original locations. In subsequent outmigrations, the data for the **cat_descr** and **cat_picture** columns is clustered by **cat_stock_clstr**, based on the value of **stock_num**. Changing the cluster key impacts how the data is stored on the optical platter but not on the integrity of the data.

Changing the clustering arrangement often can have an adverse affect on how efficiently data is retrieved. The TEXT and BYTE data that remains clustered by **manu_code** is effectively unclustered when **stock_num** accesses it. Accessing it by **stock_num** can result in frequent platter exchanges. To alleviate this problem, update the TEXT and BYTE data that **manu_code** previously stored so that it is rewritten in the new **stock_num** clusters. Because the optical media is WORM, however, the space where the TEXT and BYTE data was stored originally cannot be reused.

Administrators and programmers should be aware that the optical clustering strategy significantly affects the use of space on the optical platters.

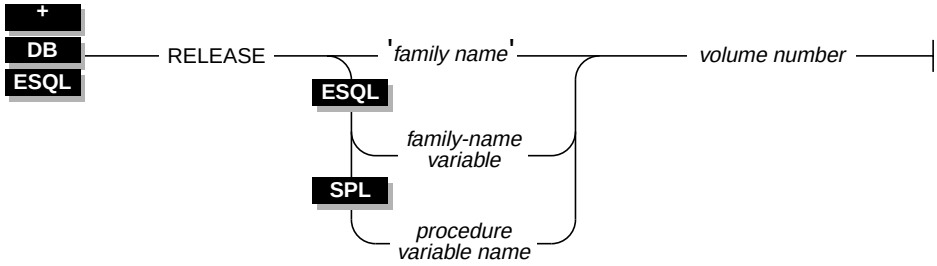
References

See the CREATE OPTICAL CLUSTER statement on page 4-6.

RELEASE

Use the RELEASE statement to cancel a reserve request for an optical volume.

Syntax



Element	Purpose	Restrictions	Syntax
family name	A quoted-string constant that specifies a family name in the optical-storage subsystem	Must be an existing family name	Quoted String segment, see the *Informix Guide to SQL: Syntax*
family-name variable	A CHARACTER or VARCHAR host variable that contains a family name	Must be an existing variable name	Variable name must conform to language-specific rules for variable names.
procedure variable name	The name of a variable defined in a stored procedure	Must be defined in the procedure and valid in the statement block	Identifier segment, see the *Informix Guide to SQL: Syntax*
volume number	The volume being released; specified as an integer	Must be reserved and mounted correctly	VOLUME() function, p. 4-20 RESERVE statement, p. 4-14 Expression segment, see the *Informix Guide to SQL: Syntax*

Usage

Each time that a RELEASE statement is executed, the value one (1) is subtracted from the reserve counter for the volume. When the value of the reserve counter is zero, you can unmount the volume.

The volume specified in the RELEASE statement must be reserved and the platter that contains the volume must be mounted currently. An error is returned if the platter is not mounted or if the value of the reserve counter for the volume is zero.

You can issue multiple reserve requests, but you must release each one before you can remove the volume from the optical drive.

The following example releases a reservation for volume 013 in the **catalog_1991** optical family:

```
RELEASE 'catalog_1991' 013
```

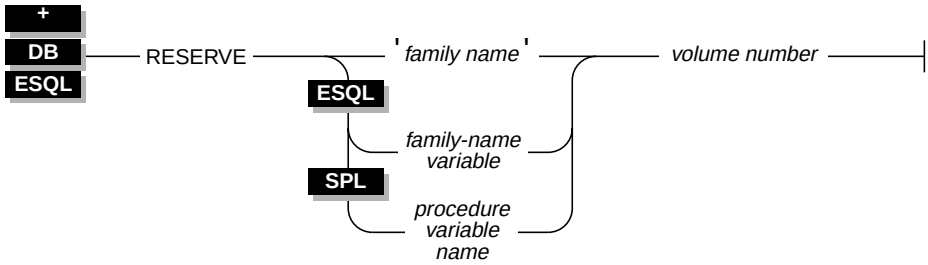
References

See the RESERVE statement on page 4-14.

RESERVE

Use the RESERVE statement to keep a particular optical volume mounted on the optical drive for the duration of a set of operations. The RESERVE statement implies a request to mount the specified volume if it is not currently mounted.

Syntax



Element	Purpose	Restrictions	Syntax
family name	A quoted-string constant that specifies a family name in the optical-storage subsystem	Must be a unique name	Quoted String segment, see the *Informix Guide to SQL: Syntax*
family-name variable	A CHARACTER or VARCHAR host variable that contains a family name	Must be a unique variable name	Variable name must conform to language-specific rules for variable names.
procedure variable name	The name of a variable defined in a stored procedure	Must be defined in the procedure and valid in the statement block	Identifier segment, see the *Informix Guide to SQL: Syntax*
volume number	The number of the volume being reserved; specified as an integer	None	VOLUME() function, p. 4-20 Expression segment, see the *Informix Guide to SQL: Syntax*

Usage

If the platter that contains the specified volume is not currently mounted on a drive, the optical-storage subsystem waits for a drive to become available and mounts the platter before it continues. The default waiting period for a mount attempt is five minutes (300 seconds). You can change this default period with the SET MOUNTING TIMEOUT statement.

You can issue multiple reserve requests for the same volume. Each time a RESERVE statement is executed for a volume, the value one (1) is added to the reserve counter for that volume. If no reserve requests have been issued for an optical volume, the value of the reserve counter is zero, indicating that you can remove the volume from the optical drive.

When you finish accessing the reserved volume, you can release it with the RELEASE statement. Each time a RELEASE statement is executed, the value one (1) is subtracted from the reserve counter for that volume. Thus, you can issue multiple reserve requests, but you must release each one before you can remove the volume from the optical drive.

You can use a SELECT statement with the FAMILY() and VOLUME() functions to obtain the family name and volume number of the volume that you want to reserve. Then, you can specify that optical volume in the RESERVE statement, as follows:

```
SELECT FAMILY(cat_picture), VOLUME(cat_picture)
     WHERE manu_code = HRO AND stock_num = 301
.
.
.
RESERVE 'catalog_1991' 013
```

References

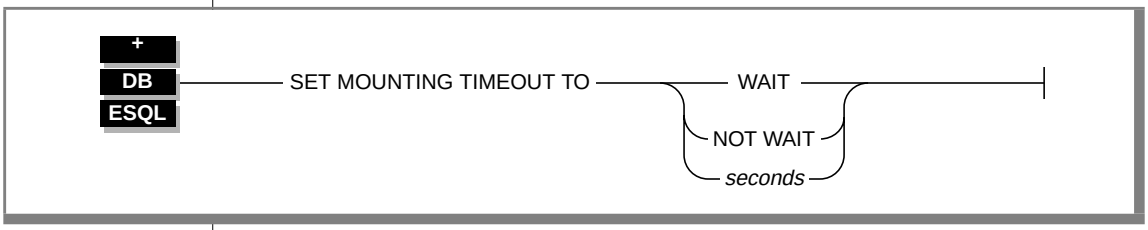
See the RELEASE statement on page 4-12.

SET MOUNTING TIMEOUT

When you request access to an optical volume, the optical-storage subsystem cannot mount it immediately if all the optical drives are busy processing requests from other users. In addition, one or more drives might be reserved at the time of your request.

Use the SET MOUNTING TIMEOUT statement to control the amount of time that your application waits for a volume to be mounted before it abandons the request to continue processing. The mounting timeout disregards the amount of time spent waiting for an optical VP to become available. If the SET MOUNTING TIMEOUT statement is not executed, the default value for a mounting timeout is five minutes (300 seconds).

Syntax



Element	Purpose	Restrictions	Syntax
WAIT	Wait until a volume is mounted	None	Literal Number segment, see *Informix Guide to SQL: Syntax*
NOT WAIT	Do not wait for a volume to be mounted	None	Literal Number segment, see *Informix Guide to SQL: Syntax*
seconds	The maximum number of seconds the process should wait for access to a volume. No default value exists for time-out within the statement.	Must be specified as a positive integer or zero. Mounting time of zero seconds is the same as ABORT	Literal Number segment, see *Informix Guide to SQL: Syntax*

Usage

If you set the time-out value to WAIT, all requests for a volume wait until the volume is mounted on an optical drive. The process that made the request waits indefinitely.

If you set the time-out value to NOT WAIT, all requests for a volume are aborted if that volume is not currently mounted on an optical drive.

If you set the time-out value to a positive integer, the integer specifies the number of seconds that the process waits for a volume to be mounted before the request is aborted.

If you set the time-out value to zero, processes are directed to wait for a volume to be mounted only if an optical drive is currently available. If a drive contains either a reserved volume or a platter that is being accessed, the request is aborted. If the drive is available, the process waits for the platter that contains the requested volume to be mounted. In general, mounting an optical platter from its slot to a drive in the jukebox takes about 15 seconds.

The following example sets the mounting time-out period to 45 seconds. When the application subsequently requests that a volume be mounted, it waits for up to 45 seconds for the platter with that volume to be mounted. If the platter with that volume cannot be mounted within the 45-second period, the process times out.

```
SET MOUNTING TIMEOUT TO 45
```

Function Expressions

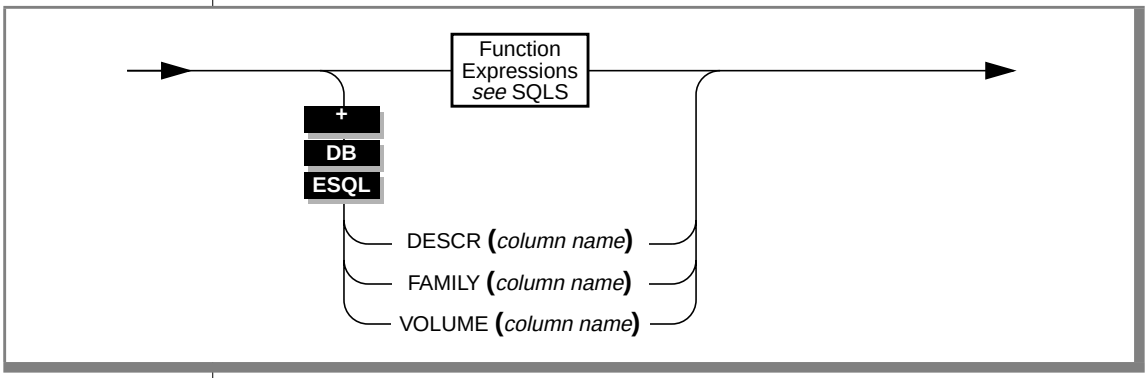
The following function expressions support the optical-storage subsystem. Use them with the SELECT statement.

- DESCR()
- FAMILY()
- VOLUME()

You can use a function expression to perform various operations on column data or to obtain information from the database server about the contents of one or more columns. A function expression requires an argument or parameter. For all three of these function expressions, the argument is the name of a column that is stored on an optical platter. For more information on function expressions and a complete list of other function expressions, see the *Informix Guide to SQL: Syntax*.

Use the DESCR(), FAMILY(), and VOLUME() function expressions to obtain information about TEXT or BYTE data columns that are stored on optical platters.

Syntax



Element	Purpose	Restrictions	Syntax
column name	The name of the TEXT or BYTE data column on which the function operates	The column name must already exist	Identifier segment, see *Informix Guide to SQL: Syntax*

Usage

The DESCR() function returns TEXT data that is the encoded descriptor for the TEXT or BYTE data column. The FAMILY() function returns the name of the optical family in which a column is stored. The VOLUME() function returns the number of the volume on which a column is stored. All three functions are valid *only* for data stored on an optical platter. They are valid whether the TEXT and BYTE data is stored sequentially or in clusters. An error message is returned if the functions are used on a column that is not stored on an optical platter.

DESCR()

The DESCR() function returns TEXT data, which is the encoded descriptor for the TEXT or BYTE data column. The encoded descriptor is 112 bytes. Use it as a column value in INSERT and UPDATE statements as a way to create additional pointers to existing TEXT and BYTE data on the optical platter. Before the descriptor is inserted in a data row, it is decoded and validated to verify its consistency and legitimacy. The optical-storage subsystem performs all encoding, decoding, and validations through calls by the Optical Subsystem to API functions. If the validation fails, the insert operation fails.

The DESCR() function is designed for WORM optical media only.

In the following example, the DESCR() function is used in an INSERT statement to fill the **picture** column of the **pictures** table with pointers to existing TEXT and BYTE data from the **catalog** table:

```
INSERT INTO pictures (stock_num, picture)
    SELECT stock_num, DESCR(cat_picture) FROM catalog
    WHERE manu_code = 'HRO'
```

FAMILY()

The FAMILY() function returns the name of the optical family in which a TEXT or BYTE data column is stored.

In the following example, the FAMILY() function obtains the *family name* of the optical platters that contain the data for the **cat_picture** column:

```
SELECT FAMILY(cat_picture) FROM catalog WHERE manu_code = 'HRO'
```

One row of output (*family name*) is generated for each row where **manu_code** is equal to HRO.

VOLUME()

The VOLUME() function returns the number of the volume where a TEXT or BYTE data column is stored.

Volumes are filled numerically. Once a volume is filled, or no room is available for a cluster, all subsequent writes occur on volumes with a higher number. When TEXT and BYTE data is stored sequentially, the highest volume number contains the most recently stored objects. However, this generalization is not true for clustered TEXT or BYTE data.

In the following example, the VOLUME() function generates a list of the volumes that contain data for the **cat_picture** column:

```
SELECT VOLUME(cat_picture) FROM catalog
```

References

For more information on using DESCR(), FAMILY(), and VOLUME(), see Chapter 3.

For the following information, see the *Informix Guide to SQL: Syntax*:

- A complete list of other function expressions
- The syntax of the column-definition portion of the CREATE TABLE statement, which is used to assign a TEXT or BYTE data column to an optical family
- The SELECT statement

Index

O

P

R

S

T

U

V

W

X