Oracle7 Spatial Data Option™ Application Developer's Guide

Version 7.3.2

Part No. A43695–1

ORACLE®

New Dimensions in Transparent Data Sharing

Oracle7 Spatial Data Option™ Application Developer's Guide, Release 1.3.5

Part No. A43695–1

Author: Meg Hennington
Contributors: Nipun Agarwal, John Barranco, Guy Binns, Frank Bishop, Jesse Driver, Linda Feng, Gwendolyn Gall, Mike Galluchon, Edric Keighan, Cynthia Kibbe, Monnia Oropeza, Nancy Raabe, Jim Rawlings, Tim Robertson, Jon Veach, Peter Vretanos.

Preface

The *Oracle7 Spatial Data Option Application Developer's Guide* provides a step–by–step account of how to use the Spatial Data Option with Oracle7 to design and set up a database, and query and manipulate spatial data. It also includes a description of the demos available with the Spatial Data Option.

The following topics are included in this preface:

- audience
- how this guide is organized
- related publications
- document conventions
- customer support
- documentation sales and client relations
- reader comments

Audience

This manual is intended for developers and users of the Spatial Data Option.

Readers of this manual are assumed to be familiar with Oracle7 Relational Database Management System (RDBMS) concepts and experienced in using SQL. They are also assumed to have fundamental knowledge of the operating system environment on which they are running Spatial Data Option.

As a prerequisite, all readers should read Part I, ”Concepts,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.* It is a comprehensive introduction to the concepts and terminology used throughout this guide.

How This Guide is Organized

The chapters in this guide are organized as follows:

Chapter 1, ”Introduction”

This chapter provides an overview of the *Oracle7 Spatial Data Option Application Developer’s Guide*.

Chapter 2, ”Designing a Spatial Database”

This chapter includes the information and decisions you must make before designing the database.

Chapter 3, ”Setting Up a Spatial Database”

This chapter presents the tasks required for table creation, data conversion, and loading of spatial tables.

Chapter 4, ”Data Manipulation and Query”

This chapter includes information about procedures on how to manipulate and query the data in the database.

Chapter 5, ”Demos”

This chapter describes how to run Spatial Data Option demos and SQL scripts.

Index

Related Publications

Oracle7 Server Documentation Set

In addition to this guide, general information about the Oracle7 Server for all operating systems is provided in the Oracle7 Server documentation set, which consists of the following manuals:

Oracle7 Server Concepts

This book describes all features of the Oracle7 Server. It provides a conceptual foundation for the practical information contained in the other Oracle7 Server documentation.

Oracle7 Server Administrator's Guide

This book describes how to manage the Oracle7 Server.

Oracle7 Server Tuning

This book describes how to enhance Oracle7 Server. database performance by adjusting database applications, the database itself, and the operating system.

Oracle7 Server Application Developer's Guide

This book describes features of the Oracle7 Server, and how to develop applications for Oracle7.

Oracle7 Server Reference

This book provides reference information about the Oracle7 Server, including initialization parameters, data dictionary views, database limits, and SQL scripts.

Oracle7 Server SQL Reference

This book provides a complete description of SQL, which is used to manage information in an Oracle7 database.

Oracle7 Server Utilities

This book describes how to use Oracle7 Server utilities for data transfer, maintenance, and database administration.

Oracle7 Server Messages

This book provides complementary information about messages generated by the Oracle7 Server and its integral parts such as PL/SQL, precompilers, and SQL*Loader.

PL/SQL User's Guide and Reference

This book explains how to use PL/SQL, Oracle Corporation's procedural language extension to SQL.

Oracle7 Spatial Data Option Documentation

In addition to this guide, information about the Spatial Data Option for all operating systems is provided in the following manuals:

Oracle7 Spatial Data Option Overview – Advances in Relational Database Technology for Spatial Data Management

This book provides a general explanation of the Spatial Data Option.

Oracle7 Spatial Data Option Reference and Administrator's Guide

This book provides concepts, reference, and administration information for the Spatial Data Option.

Document Conventions

Conventions used in this guide differ somewhat from those used in other Oracle documentation, including the generic Oracle7 Server publications listed previously. The following conventions are observed.

Special Conventions

Since many operating systems are case-sensitive, enter commands exactly as shown.

Bold	Bold type is used to denote directories and filenames, as in **init.ora**. Portions of the filename that may vary appear in italics, as in **sgadef*x*.dbf.** System privileges also appear in bold.
`*italics*`	Italicized words in Courier font represent a variable. Substitute an appropriate value.
underlining	Underlining is used to define syntax defaults.
UPPERCASE WORDS	Uppercase text is used to call attention to Oracle command keywords and statements.
[UPPERCASE]	Key names are represented by uppercase letters enclosed in square brackets, as in [RETURN].

Command Syntax

`Courier`	Courier font is used for text that must be entered exactly as shown, as in the following example: `ls –l`
Vertical lines |	Vertical lines are used for alternative choices. The set of alternative choices is enclosed by curly braces if one of the items is required, or by square brackets if the item is an optional alternative.
Curly braces { }	Curly braces are used for required items. Users must choose one of the alternatives as in the following example: `.DEFINE { macro1 | macro2 }`
Brackets [ ]	Square brackets are used for optional items, as in the following example: `cvtcrt termname [outfile]`
Ellipses	Ellipses are used for an arbitrary number of similar items, as in the following example: `CHKVAL fieldname value1 value2 ... valueN`

The following symbols should always be entered as they appear in the command format:

• period	.
• comma	,
• hyphen	–
• semicolon	;
• colon	:
• equal sign	=
• backslash	\
• single quote	’
• double quote	”
• parentheses	()

Special Icon

The following special icon is provided to alert you to particular information within the body of this guide.

Warning: The warning symbol highlights text that warns you of actions that could be particularly damaging or fatal to your system.

Other Conventions

Note that ”Oracle7” and ”Oracle” refer to the relational database server product from Oracle Corporation. The term **”oracle**” refers to an executable or account by that name.

Unless otherwise stated, examples use the C shell (**csh**(1)) syntax.

All references made throughout this book to specific chapters refer to chapters in this guide except where noted.

Customer Support

Oracle Corporation's Worldwide Support technical support answer line can be reached 24 hours a day. If you have followed the instructions in the documentation and you need further assistance, please call:

In the USA: **1.415.506.1500**

In Europe: **+ 44 1344 860160**

You will be asked a series of questions to help navigate you to the correct Oracle product support group. Be prepared to supply the following information:

- your CSI number, which helps us track problems recorded for each customer and is necessary to identify you as a supported customer
- version numbers of the Oracle7 Server and associated products
- operating system name and version number
- details of error numbers and descriptions (write down the exact errors—it will help Worldwide Support track down the problem more quickly)
- a description of the problem

Documentation Sales and Client Relations

To order hard copy documentation, call Documentation Sales at the following number:

1.800.252.0303

For shipping inquiries, product exchanges or returns, call Client Relations at the following number:

1.415.506.1500

Your Comments Are Welcome

We value your comments as a user of the Spatial Data Option. As we write, revise, and evaluate our documentation, your opinions are the most important input we receive. Please use the Reader's Comment Form at the back of this manual to tell us what you like and dislike about this manual. Alternatively, you can contact us at the following address:

Documentation Manager Government Products Division
Oracle Corporation
500 Oracle Parkway Box 659204
Redwood Shores, California 94065
Phone: 1.415.506.2503 FAX: 1.415.506.7408

Contents

CHAPTER

1

Introduction

This chapter is an introduction to the Spatial Data Option, and describes the following topics:

- Oracle7 Spatial Data Option
- suggested order for reading the Spatial Data Option documentation

Oracle7 Spatial Data Option

The Spatial Data Option is a powerful extension to Oracle's core RDBMS technology, Oracle7. It enables spatiotemporal data to be efficiently stored, accessed, and manipulated in the database in the same manner as structured data. Using Spatial Data Option, records that contain spatial references can be georeferenced and organized by their location in space. This revolutionary technology enables the manipulation of spatial features that occur in geographic databases.

Oracle7 with the Spatial Data Option delivers open, standards–based data management technology, based on a flexible client/server architecture. The world's most popular DBMS, Oracle7 delivers the highest levels of data integrity, security, recovery, and scalable performance for applications, using structured, text, multimedia, and spatial data.

The qualities that the Spatial Data Option possesses make it the perfect tool for geographical information systems. The Spatial Data Option's patent–pending technology, the HHCODE, allows users to define, organize, and manage data of up to 32 dimensions in one column. This capability gives systems designers the ability to treat non–coordinate data, such as time, temperature, and demographics, as dimensions for application use.

The Spatial Data Option permits the synchronization of spatial and attribute data by maintaining them in the same database. Even when data elements are sorted, the data retains its spatial organization. Because the spatial data is clustered predictably in the database, a spatial query can quickly access only the relevant data rather than having to search the entire database.

In today's enormous databases, often sized in terabytes, the Spatial Data Option's ability to search, retrieve, and analyze spatial data at speeds comparable to text or structured data is vital. This data is easily accessed, and because of the new structure, retrieval speeds using a window on the data depend only on the size of the retrieved data set, not the size of the entire database.

For more general information on the Spatial Data Option, see *Oracle7 Spatial Data Option Overview – Advances in Relational Database Technology for Spatial Data Management.*

Suggested Reading

This section provides guidelines on the order in which to read the documentation for the Spatial Data Option**.**

Getting Started

The user just getting started with the Spatial Data Option should read Part I, ”Concepts,” *Oracle7 Spatial Data Option Reference and Administrator's Guide*, as well as Chapter 1, ”Introduction,” and Chapter 2, ”Designing a Spatial Database,” of this guide.

Developing the Database

The user already acquainted with the Spatial Data Option and about to enter the development stage should read Chapter 2, ”Designing a Spatial Database,” and Chapter 3, ”Setting Up a Spatial Database,” of this guide.

Database Maintenance

The user who is responsible for maintaining a database should read Chapter 4, ”Data Manipulation and Query,”of this guide and Chapter 4, ”Administration,” *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

Definition of Utilities, Functions, and Procedures

For a definition of all the utilities, functions, and procedures associated with Spatial Data Option, see Part II, ”Reference,” *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

CHAPTER

2 Designing a Spatial Database

This chapter discusses the issues associated with designing a database utilizing the Spatial Data Option.

The following topics are included in this chapter:

- determine database parameters
- data conversion considerations
- data load considerations
- query considerations
- design decision table

This chapter is intended to be a mental exercise to aid in developing a spatial database. Determining database parameters provides helpful tips for defining a spatial table. Conversion, loading, and query requirements must be considered when designing an implementation plan for a spatial database. The decision table at the end of this chapter is provided as a tool to record design decisions.

Oracle Corporation recommends setting up a test database to assist in determining the appropriate design choices for a spatial database.

Determine Database Parameters

After determining the purpose of the database and what data to include, consider how the Spatial Data Option characteristics affect your design plan.

Database Parameter Tasks

Perform the following tasks before setting up a spatial table:

- Task 1: Choose table type.
- Task 2: Define data elements as dimensional or attribute.
- Task 3: Determine number and type of dimensions.
- Task 4: Determine dimension range.
- Task 5: For each dimension range, determine scale.
- Task 6: For partitioned tables, determine high water mark.
- Task 7: For partitioned tables, select partition key column.
- Task 8: Optional. For partitioned tables, select compute mode.
- Task 9: Determine sizing requirements.

Task 1: Choose Table Type

This section discusses considerations for choosing a partitioned or non–partitioned table type to store your data.

Partitioned Table

Create a partitioned table to store all or a majority of the data for the database.

Non–Partitioned Table

Create a non–partitioned table if there is supporting data that may need to be cross–referenced with data in the partitioned table. Non–partitioned tables are considered a temporary storage area used for maintenance, analysis, and administration.

For more information on partitioned and non–partitioned table types, see "Spatial Table Types," Chapter 1, "Oracle7 Spatial Data Option Components," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

Task 2: Determine Data Elements as Dimensional or Attribute

This section discusses considerations for determining whether data is to be dimensional or attribute.

Dimensional

Data is determined to be dimensional if it has the following characteristics:

- It is numeric.
- It is non–random.
- When defined as a dimension, it can improve access or analysis results.

Note: To include time as a dimension, it must be converted to a numeric unit. Once in a numeric unit such as Julian format, coding requirements are the same as for other dimensional data.

Attribute

Data is determined to be attribute data if it has the following characteristics:

- typically used as selection criteria in a WHERE clause
- used to perform computations in a SELECT statement
- considered associated descriptive information to be stored in separate columns

Example I If you want to use a lottery number as a data element within the database, designate it as an attribute. Lottery numbers are random and therefore not suitable for dimensional clustering.

Example II A terrain mapping application uses georeferenced coordinates for queries, and displays elevations with place names. Latitude, longitude, and elevation are candidates for dimensional data, but place names can be an attribute.

Example III A temporal dimension can be used to reflect the position of an object over time, or to differentiate multiple recordings of the same data collected at different times.

For more information on defining data elements, see "Spatial Table Architecture," Chapter 1, "Oracle7 Spatial Data Option Components," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

Task 3: Determine Number and Type of Dimensions

This section discusses considerations for determining the number and type of dimensions for the database.

Number

The maximum number of dimensions that can be defined in an HHCODE is 32. These dimensions can be coordinate values such as *x*, *y*, and *z* in your preferred coordinate system, or any other numeric data you define as dimensional.

Type

The following spatial types can be used to represent data:

- points
- lines

For consistent database design, keep in mind what your data represents or what the spatial types are.

Points

The HHCODE of an *n*–dimensional point is the *n*–dimensional area around the point. The size of the area is determined by the number of levels or length of the HHCODE.

The HHCODE functions HHIDPART, HHIDROWS, HHCELLSIZE, and HHDISTANCE can be used only on the points spatial type.

Lines

As illustrated in Figure 2 – 1, lines in two–dimensional space are represented by a four–dimensional HHCODE. Each of the coordinates is encoded into the HHCODE as a single value, representing the start and end points of the line.

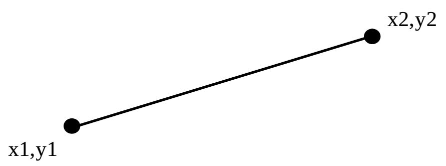


Figure 2 – 1 HHCODE Line Representation

The HHCODE functions HHIDLPART and HHIDLROWS can be used only on the lines spatial type.

Task 4: Determine Dimension Range

This section discusses considerations for determining dimension range.

Range

You must specify a range for every dimension coded into an HHCODE. The range of a dimension is a lower and an upper boundary. For better distribution of data in the partitions, the range should be set according to the lower and upper boundaries of the source data, to accommodate current and projected data ranges.

Note: If boundaries are too far apart, optimum partitioning might not occur. Data may not distribute evenly across partitions and can cause slow data retrieval. If boundaries are too close to allow for additional data to be added, the database may need to be redesigned to accommodate the new data.

Lower Boundary

The lower boundary should be a minimum numeric value that is less than the upper boundary. This lower boundary should never be exceeded by the dimension.

Upper Boundary

The upper boundary should be a maximum numeric value that is higher than the lower boundary. This upper boundary should never be exceeded by the dimension.

For example, if you have collected data every year from 1980 and expect the lifetime of the database to be 100 years, you would set the lower boundary of the temporal range to 1980 and the upper boundary to 2080. However, this could cause a potential problem if new data was introduced for the years before 1980. Once a range is set, the only way to input data that does not fall within the range is to redesign the database. This process can be quite time–intensive, depending on the amount of data you have.

Order

The lower boundary must always be entered before the upper boundary when setting up the database and during query procedures.

Example Two dimensions, latitude and longitude, with the following ranges in Cartesian coordinates are represented in Figure 2 – 2:

Latitude:	–90 to +90 degrees
Longitude:	–180 to +180 degrees

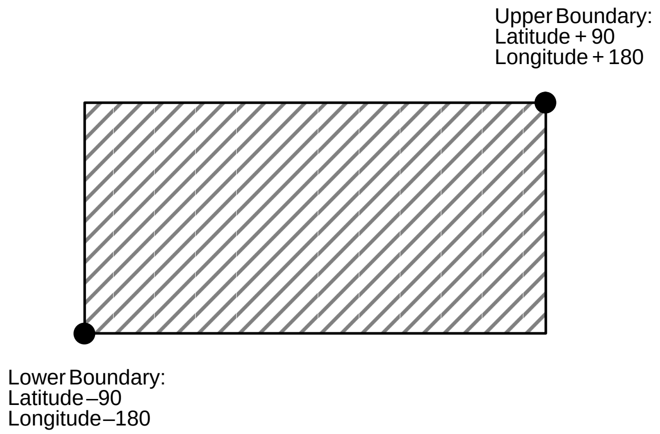


Figure 2 – 2 Defining the Boundaries of a Dimension

Task 5: For Each Dimension Range: Determine Scale

This section discusses considerations for determining the scale for each dimension encoded into the HHCODE.

Before you create a table, you must decide the scale at which to store the dimensional data. Scale refers to the number of places to the right of the decimal point. You must specify the scale for each dimension you encode in the HHCODE.

Observe the following requirements for scale:

- A dimension can have the same scale as the original data.
- A dimension can have a lower scale than the original data. However, this option can degrade the accuracy and detail of the original data.
- A dimension must use the same scale for the same dimension throughout the table.

Note: If a dimension cannot share the same scale as the original data, you can create an attribute column to record the discrepancies.

Task 6: For Partitioned Tables: Determine the High Water Mark

This section discusses considerations for determining the high water mark for partitioned tables.

The number that you choose for the high water mark determines the maximum number of rows to store in a partition before subdivision occurs, either manually or when data is loaded using SD*Loader. The number of subdivisions at each level can reach a maximum of 2^n where n is the number of dimensions. For example, if you have three dimensions on the first subdivision of data, you can have up to eight partitions. The optimum high water mark depends on factors that differ from environment to environment.

If working with a test database, try assigning different high water mark values to determine how data partitions, and how to balance the effects of extract times versus load times.

The goal for setting the high water mark is to make the partitions smaller than the area that will be queried. The following steps can help determine what the optimum high water mark is for the database:

1. Determine the size of the average query window.
2. Determine the number of records that would fall within the average query window.
3. Determine the high water mark to be a number less than the number of records that fall within the average query window.

For example, consider a database that stores topographical characteristics for the entire world, but where information is usually analyzed at 40 square mile intervals. The average query window size would be 40 square miles. The average number of records within each 40 square mile area is 100,000. To create enough partitions without slowing query times, the high water mark is set at 10,000.

Batch Loading

Ideally, the high water mark should be set low for the initial data load. This sets up the majority of partitions that will be needed and reduces time for subsequent loads.

Note: The high water mark can be changed after data has been loaded, but partitioning according to this change occurs only if or when data is loaded into a partition that reaches this new high water mark using SD*Loader. Existing partitions do not automatically reorganize themselves according to the change.

For more information on the high water mark and partitioning, see Chapter 1, ”Oracle7 Spatial Data Option Components,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.*

Task 7: For Partitioned Tables: Select Partition Key HHCODE Column

This section discusses considerations for selecting the partition key column for partitioned tables. You can have many HHCODE columns in the spatial table, but only one can be the partition key column.

The partition key column is the HHCODE column that forces partitioning to occur when data is loaded using SD*Loader. The column you select should be the one that organizes the data predictably based on retrieval requirements. It should have the following characteristics:

- Only one HHCODE column per table can function as the partition key.
- It should organize data in a predictable manner for queries.
- It must be established before using SD*Loader for proper partitioning.
- It should include at least one dimension that makes the HHCODE unique.

Note: Once you have loaded data into the table, you cannot change the partition key column without re–creating the table and reloading the data.

Organize Predictably

Dimensional data needs to be organized predictably. This means that data encoded in the partition key HHCODE column can be sorted to organize data quickly to determine where information is stored.

For example, paper files in an office file cabinet are usually organized in a predictable manner. Information is placed in a folder, which is in turn organized alphabetically. This predictable organization makes it easy to determine where certain information is stored.

Unique Data

In a table where the partition key HHCODE column is comprised of data that is not unique, data can continue to be placed in partitions that have exceeded the high water mark. For better query performance, each HHCODE column entry should be unique.

For example, if the partition key HHCODE column has two dimensions, latitude and longitude, and the same latitude and longitude data is collected year after year, all identical values are stored in the same corresponding partition. Even if the high water mark is exceeded for a particular partition, all identical values are still stored in the same partition, but the partition no longer subdivides. Adding a temporal dimension ensures the uniqueness of each HHCODE and eliminates this problem.

Task 8: Optional for Partitioned Tables: Select the Compute Mode

This section discusses considerations for selecting the compute mode for partitioned tables. Compute mode is optional.

The compute mode is the method SD*Loader uses to count the number of rows in a given partition when loading data into the spatial table. It is used by SD*Loader to determine if the high water mark has been reached and the partition should subdivide.

The compute mode can be set to ESTIMATE or EXACT.

If the compute mode is set to ESTIMATE, the number of rows is calculated using the following command:

```
SQL> ANALYZE TABLE partition_name ESTIMATE STATISTICS;
```

If the compute mode is set to EXACT, the number of rows is calculated exactly using the following command:

```
SQL> SELECT count(*) FROM partition;
```

Load times can vary depending on the machine, amount of data, and the fullness of existing partitions. If you have a test database, experiment using ESTIMATE versus EXACT to see which operates most efficiently with your dataset and environment.

Task 9: Sizing

Consider the amount of memory and disk space available for setting up a spatial database.

Many features unique to the Spatial Data Option require additional space. Consider the following before creating the database:

- triggers
- rollback segments
- system global area
- high water mark

Triggers

Triggers on a partitioned table are duplicated for each partition. As more partitions are created, the triggers created for them are stored in the SYSTEM tablespace.

For example, consider a trigger that is 200 bytes in size. If the spatial table has 10,000 partitions, then trigger storage takes up two Mb in the SYSTEM tablespace.

If triggers are required, the size of the SYSTEM tablespace should be increased. The trigger body definition should be kept small by encapsulating most of the SQL and PL/SQL statements in a separate package. The trigger body definition calls this common procedure. This method saves on database space because the common procedure is stored only once, instead of storing the trigger definition once for each partition.

Rollback Segments

Due to the structural nature of spatial tables, plan to create larger rollback segments than those used for standard Oracle7 tables. The size of packages used to install, back up files, and recover files requires larger rollback segments to ensure that information is not lost.

System Global Area

Make the system global area larger than for standard Oracle7 if you are loading and storing the Spatial Data Option packages in the system global area.

High Water Mark

Assuming that a partition does not first reach the Oracle7 size limit for a table as determined by the MAXEXTENTS parameter, the high water mark determines the maximum partition size.

Data Conversion Considerations

This section discusses the decisions you must make before converting data.

SLF files may not be portable across machines, and may need to be created on the machine used for the final database.

Conversion Consideration Tasks

Perform the following tasks before determining the conversion method best for you:

- Task 1: Determine source data format.
- Task 2: Choose a conversion method.
- Task 3: Choose when to sort.

Figure 2 – 3 is an illustrated overview of the Spatial Data Option conversion process.

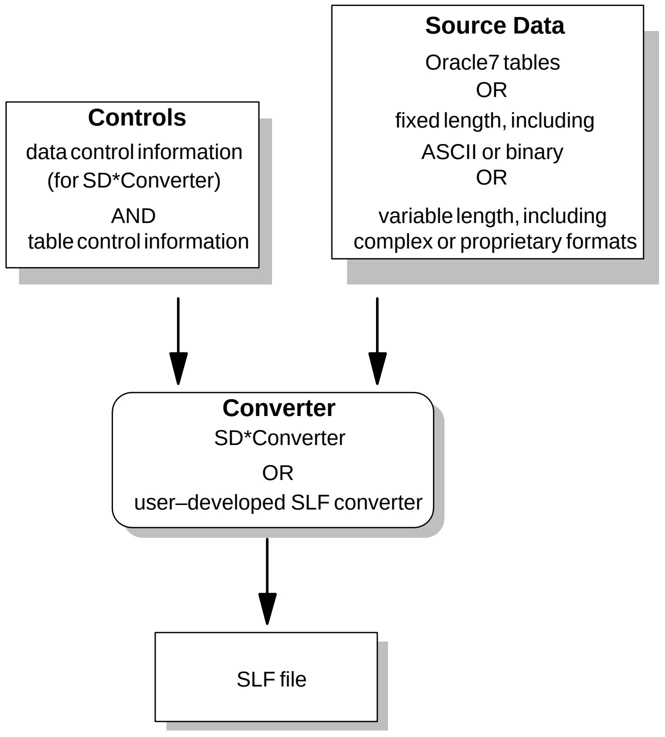


Figure 2 – 3 Data Conversion Overview

Task 1: Determine Source Data Format

This section discusses considerations for determining source data format.

Determining your data format helps determine which conversion method you employ. Use the following questions as guidelines:

- What is the current format of the data?
  - Oracle7 tables
  - fixed length, which includes ASCII or binary files, where all records are of similar format
  - variable length or complex format files, such as an existing proprietary format
- Does all similar data have the same record length?

Task 2: Choose a Conversion Method

This section discusses considerations for determining the best method for converting your data to SLF files.

The following are the basic conversion methods:

- conversion using the SD*Converter utility
- conversion using the SLF library functions to create your own converter

The SD*Converter utility converts data from Oracle7 tables or fixed format data files, such as ASCII or binary.

If converting from Oracle7 tables using SD*Converter, you must first create and register a spatial table in the Spatial Data Option data dictionary. During conversion, data control information is accessed from the Oracle7 data dictionary and table control information is accessed from the Spatial Data Option data dictionary.

If converting variable length or complex format data, such as proprietary format data, you have the following options:

- Use the SLF Library of functions to create your own converter.
- Write a C program to convert data to fixed length ASCII or binary and then use the SD*Converter utility.
- Convert and load data into an Oracle7 table and then use the SD*Converter utility.

Task 3: Choose When to Sort

This section discusses the option of when to sort source data. The sorting process takes approximately the same amount of time when performed as part of either conversion or loading. You can choose to sort data during the conversion or the loading process, depending on your requirements.

Some sorting considerations include the following:

- If fast conversion times are important, sorting should take place at the loading stage.
- If fast load times are important, sorting should take place at the conversion stage.
- If the data file contains fewer records than the high water mark, do not sort on conversion.
- If sorting is not done at the conversion stage, SD*Loader performs a sort before loading the data.
- Ensure that the speed of the CPU is adequate to achieve optimum sort times.
- Ensure that there is adequate space to accommodate the generated temporary files.

Data Load Considerations

This section discusses the decisions you must make before loading data.

Loading Consideration Tasks

Perform the following tasks before loading data:

- Task 1: Determine the data loading method.
- Task 2: Determine the high water mark for loading data.
- Task 3: Choose when to sort.
- Task 4: Save or discard temporary control files.

Task 1: Determine Data Loading Method

This section discusses data loading options.

SLF files can either be loaded all at once or in batches. Loading in batches with a low high water mark for the initial load forces most of the partitioning to occur during the first load. Forcing the partitioning reduces this time–intensive process for subsequent data loads.

Task 2: Determine High Water Mark for Loading Data

This section discusses considerations for determining the high water mark for different loading situations.

Single Data Load

If loading all the data at once, keep the high water mark at the level determined during the table creation process.

Batch Load

If loading data in batches, you can vary the high water mark for each data load. Setting the high water mark to a much lower number for the first batch causes most of the partitions to be created, which reduces the time spent on partition creation for subsequent loads.

Task 3: Choose When to Sort

This section discusses the option to sort during the load process.

Sorting occurs automatically during the load process if it was not performed during the conversion process. If possible, choose a machine that can perform sorts most efficiently.

Task 4: Decide to Save or Discard Temporary Control Files

This section discusses the option to save or discard temporary control files created during a load.

SD*Loader automatically creates temporary control files. Once the load is completed, the files are deleted. There is an option that can be used to save the files. Saving the files can be useful if you need to debug a load process.

Query Considerations

This section discusses the query decisions you must make before setting up the database.

Window Extract

One consideration is the possible shape of window extracts. The number of encoded dimensions can cause a restriction on the shape of the query window. Also, the units used to define the window extract areas must match the units used to define the dimension during table creation to receive correct returns.

Figure 2 – 4 illustrates each type of window extract and the number of dimensions for which they are applicable:

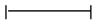
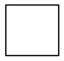
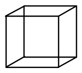
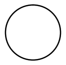
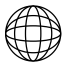
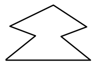
Extract Type	1–D	2–D	3–D	N–D
Range				Yes
Proximity	N/A			Yes
Polygon	N/A		N/A	N/A

Figure 2 – 4 Window Extract Types

Design Decision Table

The following table lists the key decisions you need to make before setting up the database. For each decision, record your selected option in the Choice column.

Decision	Option	Choice
Table Type	Partitioned Non–Partitioned	
Data Elements	Dimensional Attribute	
Dimension Number	Up to 32	
Dimension Type	Point Line	
Dimension Range	Upper Bound-ary Lower Bound-ary	
Dimension Scale		
High Water Mark	Numeric Value	
Partition Key HHCODE Column	Restricted to one column per table	
Compute Mode	ESTIMATE EXACT	
Conversion Method	SD*Converter User–Devel-oped Converter	
When to Sort	Conversion Load	
Data Loading Method	Single Batch	
High Water Mark	Single Batch initial subsequent	
Temporary Files	Save Discard	

Table 2 – 1 Design Summary Sheet

CHAPTER

3

Setting Up a Spatial Database

This chapter presents the tasks required for table creation, data conversion, and loading of Spatial Data Option tables. It includes the following topics:

- spatial table creation
- converting data into SLF files
- loading data from SLF files

Spatial Table Creation

This section provides instructions for creating partitioned or non–partitioned tables for the Spatial Data Option.

Spatial Table Creation Tasks

You need to perform the following tasks for both partitioned and non–partitioned tables. If you are creating a partitioned table you must additionally define the high water mark and the partition key column during tasks 3 and 4.

- Task 1: Optional. Determine available tablespace.
- Task 2: Create an empty table.
- Task 3: Register table. For partitioned tables, define partition parameters.
- Task 4: Add the HHCODE column. For partitioned tables, define partition parameters.
- Task 5: Optional. Provide tablespace.
- Task 6: Verify spatial table creation.

Task 1: Optional. Determine Available Tablespaces

Use the following procedure to determine what tablespaces are available for the spatial table.

1. For a list of available tablespaces, query the Oracle7 data dictionary view USER_TABLESPACES.
2. Optional. Create a tablespace for the spatial table.

Task 2: Create an Empty Table

Use the following procedure to create an empty table:

1. Define all HHCODE and attribute columns.
   - Define HHCODE columns as RAW (255).
   - Define attribute columns as you would for standard Oracle7 columns.
2. Execute the SQL CREATE TABLE command.

Example I The following example creates an Oracle7 table called POINTS with an HHCODE column LOCATION and attribute columns DTIME, DEPTH, and DESCRIPTION:

```
SQL> CREATE TABLE POINTS
2 (location       RAW (255),
3 dtime           DATE,
4 depth           NUMBER,
5 description     VARCHAR2 (30));
```

Example II The following example creates an Oracle7 table called LINES with an HHCODE column LOCATION and additional attribute columns:

```
SQL> CREATE TABLE LINES
2 (location       RAW (255),
3 seq_num         NUMBER (38,10),
4 featurecode     VARCHAR2 (12),
5 object           VARCHAR2 (30),
6 class           VARCHAR2 (30));
```

Example III The following example creates an Oracle7 table called TIME with an HHCODE column LOCATION and additional attribute columns:

```
SQL> CREATE TABLE TIME
2 (locationRAW (255),
3 dtime           DATE,
4 depth           NUMBER,
5 descr           VARCHAR2 (30));
```

Task 3: Register Table

Use the following procedure to register the table:

1. For partitioned tables, define the high water mark to activate partitioning.
2. Optional. For partitioned tables, define the compute mode. For optional parameters you must enter a NULL if you want to specify an optional parameter later in the procedure.
3. Execute the MD_DDL.REGISTER_MD_TABLE procedure to register the table in the Spatial Data Option data dictionary.

   The following is the syntax for this procedure:

   ```
   MD_DDL.REGISTER_MD_TABLE ([schema.|username,] sd_tablename
   [, high_water_mark, ['EXACT'|'ESTIMATE']])
   ```

Example I The following example registers the POINTS table:

```
SQL> EXECUTE MD_DDL.REGISTER_MD_TABLE('POINTS', –
> 10000,NULL,'EXACT');
```

Example II The following example registers the LINES table:

```
SQL> EXECUTE MD_DDL.REGISTER_MD_TABLE –
> ('LINES',1000);
```

Example III The following example registers the TIME table:

```
SQL> EXECUTE MD_DDL.REGISTER_MD_TABLE('TIME',1000,NULL,'EXACT');
```

For more information on MD_DDL.REGISTER_MD_TABLE, see Chapter 7, "SD*SQL Packages," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

Task 4: Add HHCODE Column

Use the following procedure to add an HHCODE column to the table:

1. Determine the spatial table name.
2. Define the HHCODE column name.
3. Define the partition key as FALSE or TRUE. There can be multiple HHCODE columns in a table, but only one can have the partition key parameter set to TRUE.
4. Define the lower and upper boundary for each dimension.
5. Define the scale for each dimension.
6. Execute the MD_DDL.ADD_HHCODE_COLUMN procedure to define the HHCODE column.

   The following is the syntax for this procedure:

   ```
   MD_DDL.ADD_HHCODE_COLUMN ([schema.|username,] sd_tablename,
   hhcode_column_name,[partition_key,] [not_null,] dimension_name,
   lower_boundary, upper_boundary, scale [, dimension_name,
   lower_boundary, upper_boundary, scale...])
   ```

Warning: HHCODE column definitions cannot be modified easily once they are created.

Example I The following example defines the partition key HHCODE column as LOCATION for the POINTS table:

```
SQL> EXECUTE MD_DDL.ADD_HHCODE_COLUMN ('POINTS','location', TRUE,–
>     'LON', –180, 180, 7, 'LAT', –90, 90, 7);
```

Example II The following example defines the partition key HHCODE column as LOCATION for the LINES table:

```
SQL> EXECUTE MD_DDL.ADD_HHCODE_COLUMN (’LINES’, ’location’, TRUE,–
>     ’LON1’, –180, 180, 7, ’LAT1’, –90, 90, 7, ’LON2’, –180,–
>     180, 7,’LAT2’, –90, 90, 7);
```

Example III The following example defines the partition key HHCODE column as LOCATION and includes a temporal dimension for the TIME table:

```
SQL> EXECUTE MD_DDL.ADD_HHCODE_COLUMN (’TIME’, ’location’,–
>     TRUE,’longitude’, –180, 180, 7, ’latitude’, –90, 90, 7,–
>     ’time’, MD.HHJLDATE (’1900–01–01’, ’YYYY–MM–DD’),–
>     MD.HHJLDATE (’5000–01–01’, ’YYYY–MM–DD’), 9);
```

In the preceding example, the time range is defined from 01–JAN–1900 to 01–JAN–5000 using the HHJLDATE function to calculate the Julian date for the start and end dates within the ADD_HHCODE_COLUMN procedure. Conversion can be performed separately, and the resulting values used in the ADD_HHCODE_COLUMN procedure.

For more information on MD_DDL.ADD_HHCODE_COLUMN, see Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.*

Task 5: Optional. Provide Tablespace

This task can maximize the use of available storage space.

Use the following procedure to allocate tablespace for partitioned tables:

1. Determine the spatial table name.
2. For a list of available tablespaces, query the Oracle7 data dictionary view USER_TABLESPACES.
3. Execute the MD_DDL.ALLOCATE_TABLESPACE procedure.

   The following is the syntax for this procedure:

   MD_DDL.ALLOCATE_TABLESPACE ([*schema.*|<u>*username,*</u>] *sd_tablename*, *tablespace_name*)

Example The following example allocates tablespace to the POINTS table:

```
SQL> EXECUTE MD_DDL.ALLOCATE_TABLESPACE (’POINTS’,–
>     ’tablespace4’);
```

For more information on MD_DDL.ALLOCATE_TABLESPACE, see Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.*

Task 6: Verify Spatial Table Creation

Use the following procedure to verify spatial table creation:

1. Check that the spatial table is defined in the Spatial Data Option data dictionary by accessing the data dictionary views.
2. Optional. If the table described is not correct, you can drop, re–create, and reregister the table.

Example I In the following example, the Spatial Data Option data dictionary view USER_MD_TABLES is checked to verify that the table was created as defined:

```
SQL> SELECT  md_table_name md_table, class, high_water_mark hwm,
2 count_mode
3 FROM user_md_tables
4 WHERE md_table_name = ’POINTS’;
```

The following output confirms the table’s creation as defined:

```
MD_TABLE   CLASS          HWM           COUNT_MODE
----------------------------------------------------
POINTS     PARTITIONED    10000         ESTIMATE
```

Example II In the following example, the Spatial Data Option data dictionary view USER_MD_DIMENSIONS is checked to verify that the dimensions were created as defined:

```
SQL> SELECT md_table_name md_table, column_name col,
2 dimension_name d_name, dimension_number d_num, lower_bound lb,
3 upper_bound ub, decimal_precision sc, recursion_level rl
4 FROM  user_md_dimensions
5 WHERE md_table_name = ’POINTS’
6 ORDER BY dimension_number;
```

The following output confirms the table’s creation as defined:

```
MD_TABLE   COLUMN         D_NAME  D_NUM   LB      UB      SC      RL
-----------------------------------------------------------------------
POINTS     LOCATION       LON     1       –180    180     7       32
POINTS     LOCATION       LAT     2       –90     90      7       31
```

Example III In the following example, the Spatial Data Option data dictionary view USER_MD_COLUMNS is checked to verify that the columns were created as defined:

```
SQL> SELECT column_name, data_type
2 FROM user_md_columns
3 WHERE md_table_name = ’POINTS’
4 ORDER BY column_id;
```

The following output confirms the table’s creation as defined:

```
COLUMN_NAME        DATA_TYPE
-----------------------------
LOCATION           HHCODE
DTIME              DATE
DEPTH              NUMBER
DESCRIPTION        VARCHAR2
```

Converting Data into SLF Files

Before you can load data into spatial tables, you must convert source data into SLF files.

Note: SLF files may not be portable across machines, and may need to be created on the machine used for the final database.

Figure 3 – 1 illustrates options for converting source data into an SLF file.

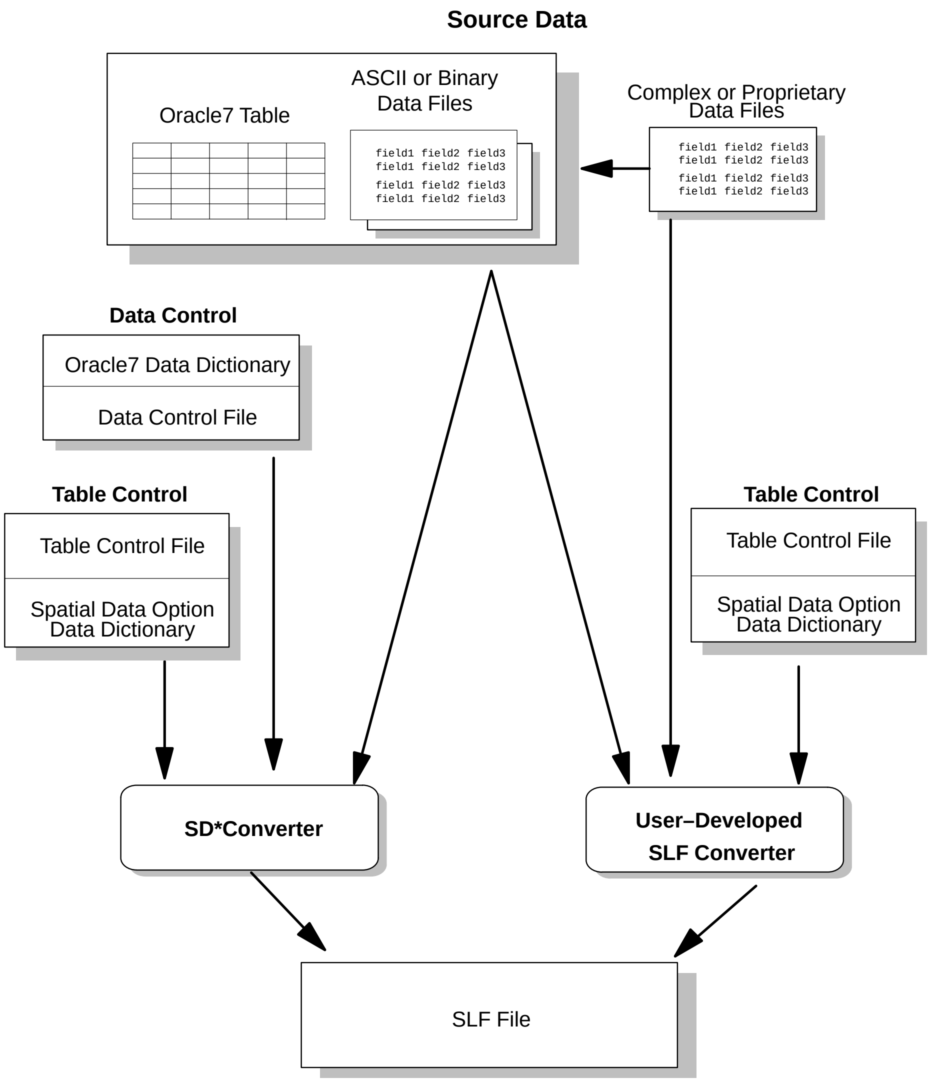


Figure 3 – 1 Data Conversion Process

Figure 3 – 2 is an example of a possible path to convert Oracle7 tables into an SLF file. The source data is an Oracle7 table. The data control is provided by the Oracle7 data dictionary. The table control is provided by the Spatial Data Option data dictionary.

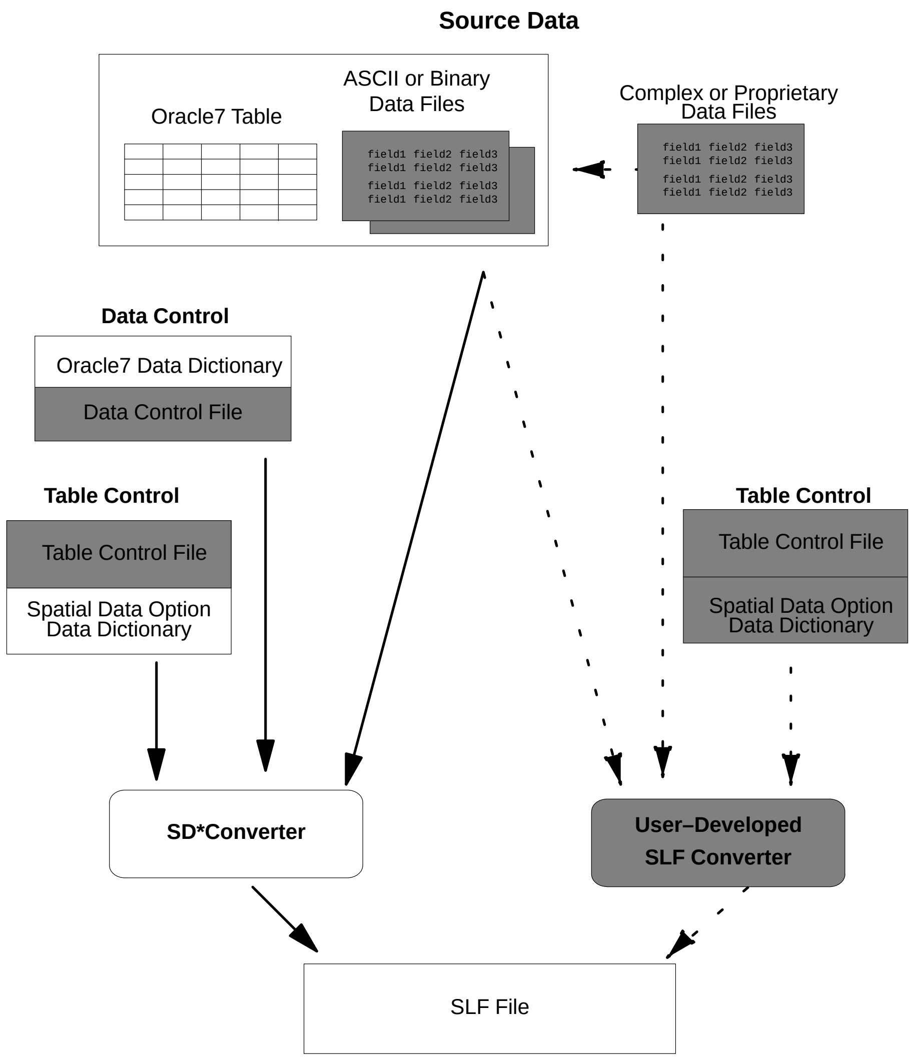


Figure 3 – 2 Conversion from Oracle7 Table to SLF File

Figure 3 – 3 is an example of a possible path to convert ASCII or binary source data into an SLF file. In this example, the source data is either an ASCII or binary file. The data control is provided by a data control file. The table control is provided by a table control file.

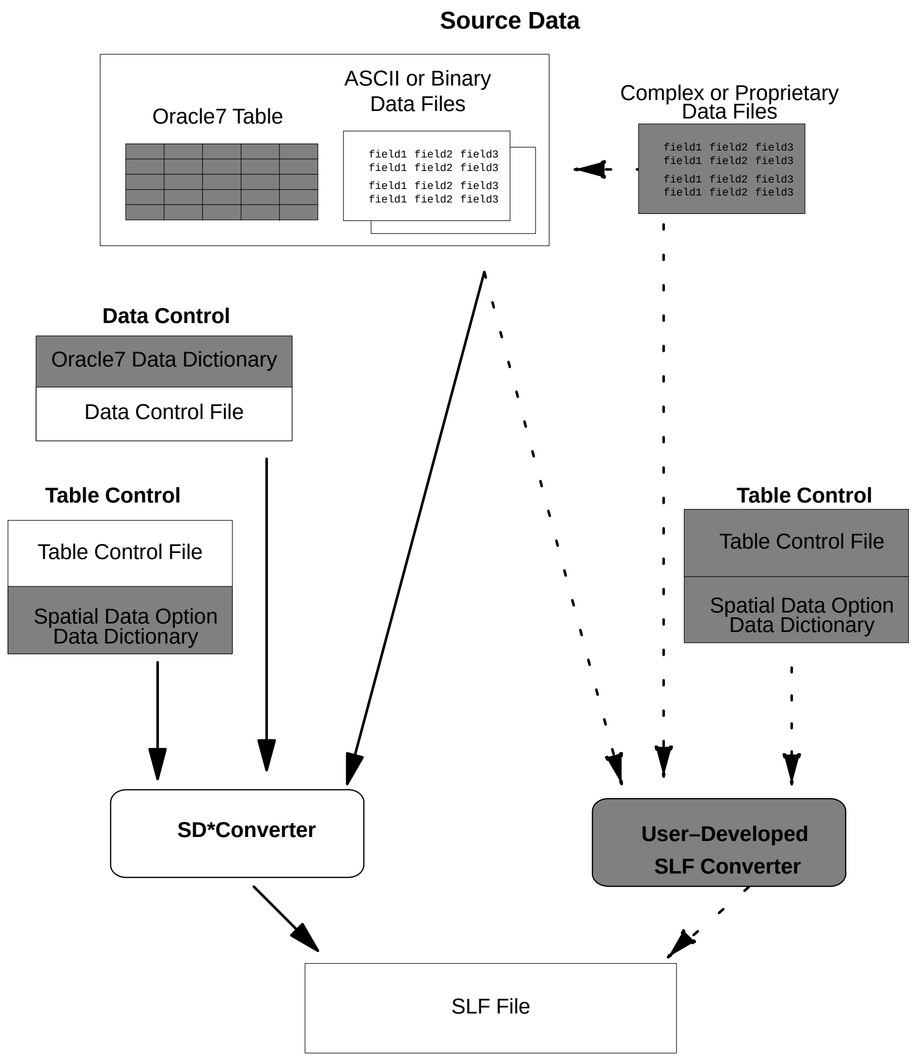


Figure 3 – 3 Conversion from ASCII or Binary Source Data to SLF File

Figure 3 – 4 is an example of a possible path to convert complex or proprietary source data into an SLF file. In this example, a user–developed converter is supplied with table control information from the Spatial Data Option data dictionary.

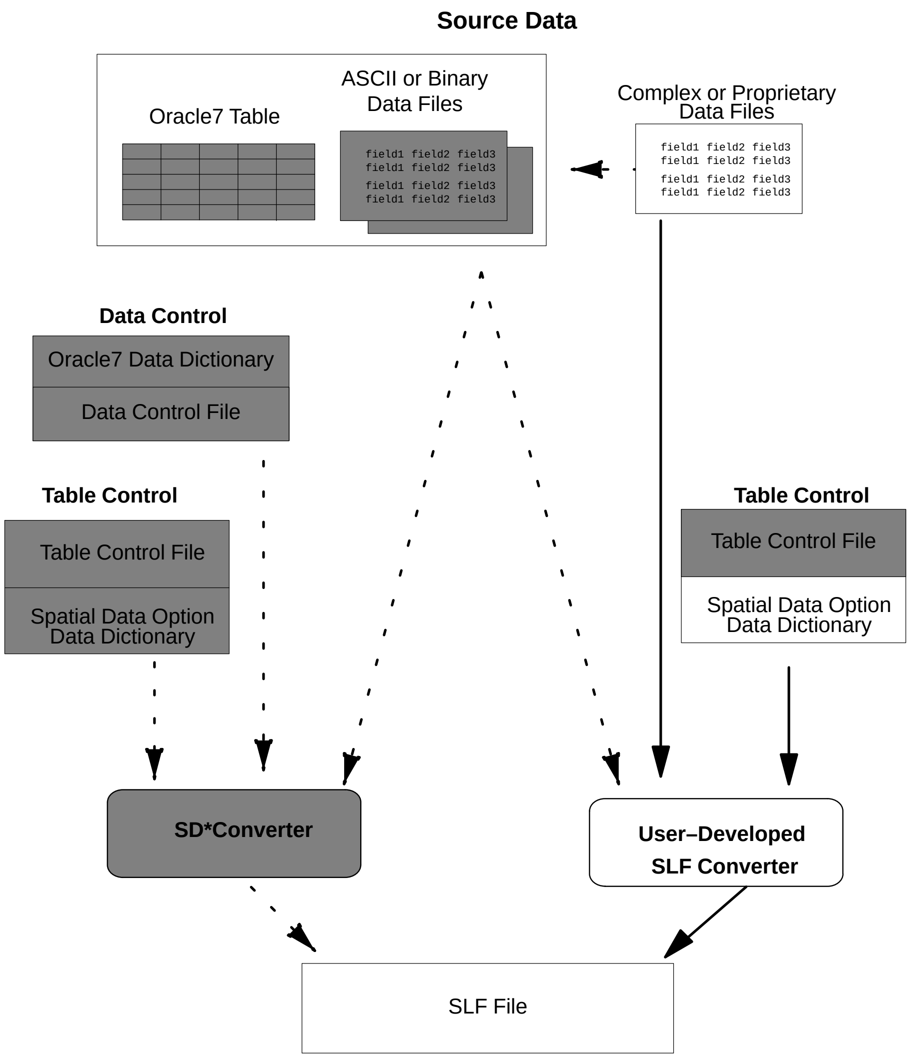


Figure 3 – 4 Conversion from Complex or Proprietary Source Data to SLF File

Data Conversion Tasks

Perform the following tasks to convert data to SLF files:

- Task 1: Determine the format of source data.
- Task 2: Provide data control information for SD*Converter.
- Task 3: Provide table control information.
- Task 4: Optional. Write user–developed converter.
- Task 5: Execute converter with optional sort.

For more information on converting data to SLF files, see Chapter 2, "Converting and Loading," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

Task 1: Determine Format of Source Data

The format of the source data must be determined before beginning the conversion process. Data should be one of the following categories:

- Oracle7 tables
- fixed length, ASCII or binary files, where all records are of similar format
- variable length or complex format files, such as an existing proprietary format
- combination of the preceding formats

For more information on conversion options, see "Choose a Conversion Method," Chapter 2, "Designing a Spatial Database."

Task 2: Provide Data Control Information

This section discusses the procedures to create a data control file or utilize data dictionary information for data conversion.

Oracle7 Data Dictionary

If converting from an Oracle7 table, the Oracle7 data dictionary is automatically accessed by the SD*Converter to provide the necessary data control information. Continue with Task 3.

Data Control File

Use the following procedure to create a data control file for source data formats other than Oracle7 tables:

1. For data control file requirements and guidelines, see "SD*Converter," Chapter 5, "Utilities," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*
2. Create a data control file for input to SD*Converter that defines the format of the conversion data. Format can be fixed length ASCII or binary.

   The following is the syntax for the data control file:

   ```
   {ASCII|BINARY}
   FIXED record_length
   COLUMN column_name POSITION {(number:number)|(number)}
   {DATE date_format_string|INTEGER|SMALLINT|FLOAT|DOUBLE|
   BYTEINT|RAW|CHAR} [NULLIF POSITION   {NE|!=|<>|EQ|==|=}
   'char_string'] DIMENSION dimension_name hhcode_column POSITION
   {(number:number)|(number)}
   {DATE date_format_string|INTEGER|SMALLINT|FLOAT|DOUBLE|BYTEINT}
   ```

Example I

The following example is a data control file called **points.ctl**:

```
Data Control File: points.ctl
-----------------------------------------------------------------
# Ctl file description of point file
ASCII
FIXED 33
DIMENSION  lon    location       POSITION  (2:11)   FLOAT
DIMENSION  lat    location       POSITION  (14:23)  FLOAT
COLUMN            depth          POSITION  (25:29)  INTEGER
-----------------------------------------------------------------
```

Example II

The following example is a data control file called **time.ctl**:

```
Data Control File: time.ctl
-----------------------------------------------------------------
# Ctl file description of time file
ASCII
FIXED 81
DIMENSION longitude    location POSITION (1:11)  FLOAT
DIMENSION latitude     location POSITION (15:25) FLOAT
DIMENSION time         location POSITION (50:71) DATE
                          'DD-MON-YY-HH-MI-SS-MLS'
COLUMN    dtime                 POSITION (29:37) DATE 'DD-MON-YY'
COLUMN    depth                 POSITION (42:47) INTEGER
COLUMN    descr                 POSITION (76:80) CHAR
-----------------------------------------------------------------
```

Task 3: Provide Table Control Information

This section discusses the procedures to create a table control file or utilize data dictionary information for source data conversion.

Data Dictionary Option

If a spatial table is already created and registered, you can substitute creating a table control file by utilizing the Spatial Data Option data dictionary table definition. The spatial table definition is accessed by specifying the spatial tablename as the TABLECONTROL parameter of the SD*Converter syntax.

Note: If converting from Oracle7 tables, you must use this option.

Table Control File

Create a table control file for the SD*Converter that defines the spatial table, using the following syntax to enter column and dimension information:

```
COLUMN column_name {VARCHAR2|NUMBER|DATE|RAW|CHAR|HHCODE
[PARTITION KEY]}
DIMENSION dimension_name hhcode_column (dimension_number,
lower_boundary, upper_boundary, scale)
```

For table control requirements and guidelines, see "SD*Converter," Chapter 5, "Utilities," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

Warning: Spatial table definitions used during conversion must match the spatial table definition of the table into which the data is loaded.

Example

The following example is a table control file called **mdpoints.ctl**:

```
Table Control File:       mdpoints.ctl
------------------------------------------------------------------
# Ctl file description of point table
COLUMN     location       HHCODE  PARTITION KEY
COLUMN     depth          NUMBER (38,10)
DIMENSION  lon            location (1, –180, 180, 7)
DIMENSION  lat            location (2, –90, 90, 7)
------------------------------------------------------------------
```

Task 4: Optional. Write User–Developed Converter

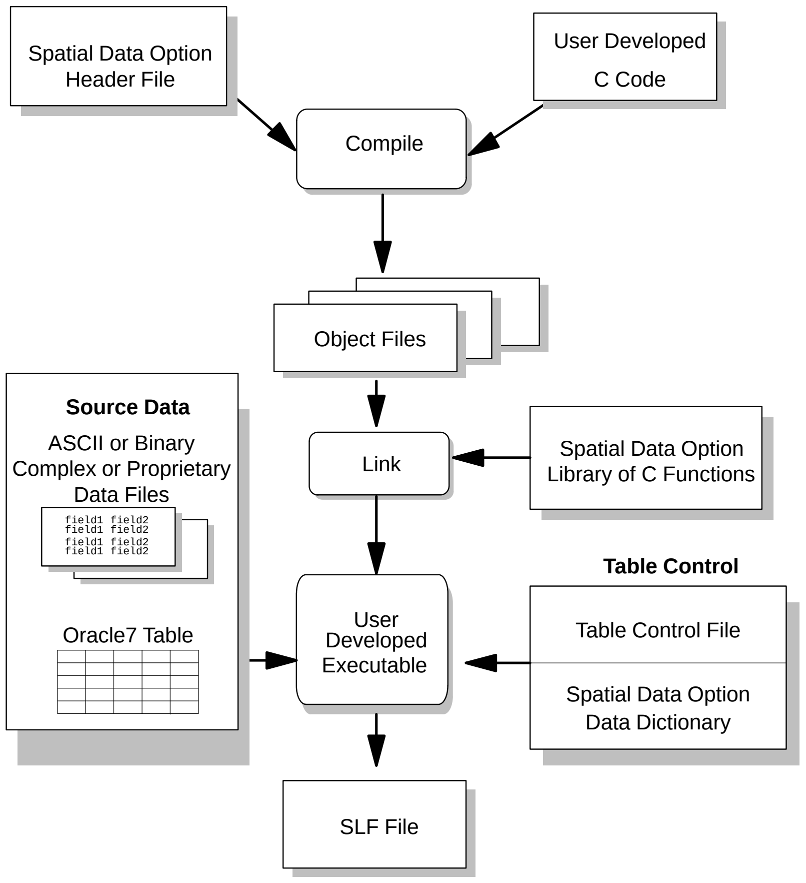
This section discusses the procedures used to write a user–developed converter for data conversion. However, you can also use the following methods as alternatives to writing your own converter:

- Write a C program to convert the source data into a fixed length ASCII or binary format and then use SD*Converter.
- Load the source data into a standard Oracle7 table and then use SD*Converter.

Use the following procedure to create a user–developed converter:

1. Include the SLF header file in the SLF file converter. The SLF header file is a C header file that describes Spatial Data Option data dictionary SLF data structures.
2. Link the SLF library file to the SLF file converter. The library file is a library of C functions.
3. Supply the following input to the converter program:
   - source data file
   - table control file or the table definition from the Spatial Data Option data dictionary

Figure 3 – 5 illustrates creating a user–developed converter and using it to convert source data to SLF files.



Spatial Data Option Header File
User Developed C Code
Compile
Object Files
Source Data
ASCII or Binary Complex or Proprietary Data Files
field1 field2
field1 field2
field1 field2
field1 field2
Oracle7 Table
Link
Spatial Data Option Library of C Functions
Table Control
Table Control File
Spatial Data Option Data Dictionary
User Developed Executable
SLF File

Figure 3 – 5 User–Developed Converter

For more information about the SLF header file and developing an SLF converter, see Chapter 8, "User–Developed SLF Converter," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

Task 5: Execute Converter

Use the following procedure to execute the SD*Converter utility or a user–developed converter:

1. If sorting on conversion, use the appropriate parameters.
2. Depending on the conversion method chosen, either execute the user–developed converter or the SD*Converter from the operating system command line.

   The syntax for the SD*Converter varies according to the input combination you choose. The following are some of the possible input combinations:

   - Oracle7 table
   - data control file, table control file, and source data file
   - data control file, Spatial Data Option data dictionary, and source data file

   When the input combination is an Oracle7 table, the syntax is as follows:

   ```
   SDCONV USERID=username/password TABLECONTROL=sd_tablename
   TABLE=Oracle7_tablename[SLF=Oracle7_tablename.slf|slf_filename]
   [LOG=log_filename] [BAD=bad_filename]
   [ERRORS=50|max_number_of_errors] [BINDSIZE=65536|bindsize]
   ```

   When the input combination is a data control file, a table control file, and a data file, the syntax is as follows:

   ```
   SDCONV DATACONTROL=data_control_filename
   TABLECONTROL=table_control_filename DATA=data_filename
   [SLF=data_filename.SLF|slf_filename] [LOG=log_filename]
   [BAD=bad_filename] [ERRORS=50|max_number_of_errors]
   [BINDSIZE=65536|bindsize] [SORT=YES|NO]
   ```

   When the input combination is a data control file, a Spatial Data Option data dictionary, and a data file, the syntax is as follows:

   ```
   SDCONV USERID=username/password
   DATACONTROL=data_control_filename TABLECONTROL=sd_tablename
   DATA=data_filename [SLF=data_filename.SLF|slf_filename]
   [LOG=log_filename] [BAD=bad_filename]
   [ERRORS=50|max_number_of_errors] [BINDSIZE=65536|bindsize]
   [SORT=YES|NO]
   ```

   For more information on SD*Converter, see Chapter 5, ”Utilities,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.*
3. If the conversion process fails, temporary files created may need to be removed.

⚠ **Warning**: SLF files created in the conversion process are considered temporary and should be deleted after the loading process. These files may not be portable across machines and should not be used to exchange data between machines.

Example I The following example creates an SLF file called **points2c.slf** from an Oracle7 table, **latlon_points**, accessing the table control information from the Spatial Data Option data dictionary, and the data control information from the Oracle7 data dictionary:

```
SDCONV userid=herman/vampire tablecontrol=points
table=latlon_points slf=points2c.slf
```

The following output appears on screen:

```
SD*Converter: Release 7.3.2.0.0 – Production on Mon Apr 24
 11:44:50 1995
Copyright (c) Oracle Corporation 1994. All rights reserved.
Converting Option:            Oracle Table to SLF File
------------------------------------------------------
Dictionary Control Table:     points

Input Data Table:             latlon_points
Output File:                  points2c.slf
Log File:                     NONE
Bad File:                     NONE

Bind Size (bytes):            65536
Errors Allowed:               50
Sort Option:                  Yes

Converting...
Sorting...

Total Errors encountered: 0
Total Bad lines skipped:  0
SD*Converter Finished
```

Example II The following example creates an SLF file called **points2b.slf** from a simple, fixed length data file, **points.dat**, using the table control information from the table control file, **mdpoints.ctl,** and the data control file, **points.ctl**:

```
SDCONV tablecontrol=mdpoints.ctl datacontrol=points.ctl
data=points.dat slf=points2b.slf
```

The following output appears on screen:

```
SD*Converter: Release 7.3.2.0.0 – Production on Mon Apr 24
11:57:59 1995
Copyright (c) Oracle Corporation 1994. All rights reserved.
Converter Option:             Data File to SLF File
```

```
---------------------------------------------------
Dictionary Control File:      mdpoints.ctl
Data Control File:            points.ctl

Input Data File:              points.dat
Output File:                  points2b.slf
Log File:                     NONE
Bad File:                     NONE

Bind Size (bytes):            65536
Errors Allowed:               50
Sort Option:                  Yes

Converting...
Sorting...

Total Errors encountered: 0
Total Bad lines skipped:  0
SD*Converter Finished
```

Example III The following example creates an SLF file called **points.slf** from a fixed length format, **points.dat**, accessing the table control information from the Spatial Data Option data dictionary, and the data control file **points.ctl**:

```
SDCONV userid=herman/vampire tablecontrol=points
datacontrol=points.ctl data=points.dat slf=points.slf
```

The following output appears on screen:

```
SD*Converter: Release 7.3.2.1.0 – Production on Mon Apr 24
11:57:59 1995
Copyright (c) Oracle Corporation 1994. All rights reserved.
Converter Option:             Data File to SLF File
---------------------------------------------------
Dictionary Control Table:     points
Data Control File:            points.ctl

Input Data File:              points.dat
Output File:                  points.slf
Log File:                     NONE
Bad File:                     NONE
Bind Size (bytes):            65536
Errors Allowed:               50
Sort Option:                  Yes
Converting...
Sorting...

Total Errors encountered: 0
Total Bad lines skipped:  0
SD*Converter Finished
```

Loading Data from SLF Files

The following section provides instructions for loading data into spatial tables from SLF files. These tasks should be performed only after successfully converting source data into SLF files.

During the load, the SLF file is sorted by partition, so that all the data for a partition is loaded at once. After each partition load, a commit is forced.

Loading Tasks

Perform the following tasks to load the SLF files into Spatial Data Option tables:

- Task 1: Ensure tablespace is adequate.
- Task 2: Perform initial load.
- Task 3: Verify spatial table creation.
- Task 4: Correct loading errors.
- Task 5: Perform any subsequent loads.

Task 1: Ensure Adequate Tablespace

Use the following procedure to ensure adequate tablespace for partitions during loading:

1. For a list of available tablespaces, query the Oracle7 data dictionary view USER_TABLESPACES.
2. Determine the spatial tablename.
3. Determine the names of the tablespaces allocated to the spatial table.
4. Optional. Use MD_DDL.ALLOCATE_TABLESPACE to create a new tablespace or MD_DDL.ACTIVATE_TABLESPACE to re–activate a deactivated tablespace.

   The following is the syntax for this procedure:

   MD_DDL.ACTIVATE_TABLESPACE ([*schema.*|*username,*] *sd_tablename*, *tablespace_name*)

For more information on MD_DDL.ACTIVATE_TABLESPACE, see Chapter 7, "SD*SQL Packages," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

Example I The following example activates TABLESPACE5:

```
SQL> EXECUTE md_ddl.activate_tablespace ('POINTS', 'tablespace5');
```

Task 2: Perform Initial Load

Use the following procedure to perform an initial data load:

1. Optional. To enhance performance of subsequent loads, set the high water mark lower than the level determined in Chapter 2, ”Designing a Spatial Database,” to force most of the partitioning to occur during the initial load. To do this, use the MD_DDL.ALTER_MD_TABLE_HWM procedure.

   For more information on MD_DDL.ALTER_MD_TABLE_HWM, see Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.*

   Note: Remember to raise the high water mark during subsequent loads.

2. Execute the SD*Loader utility as follows:

   ```
   SDLOAD USERID=username/password SLF=slf_filename
   SDTABLE=sd_tablename [LOG=log_filename]
   [BINDSIZE=65536|bindsize] [ROLLBACK=rollback_segment_name]
   [DIRECT=TRUE|FALSE] [CTLKEEP=TRUE|FALSE]
   ```

   For more information on keywords and values, see ”SD*Loader,” Chapter 5, ”Utilities,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.*

3. Optional. If the sort fails, manually remove the temporary files created.

4. Optional. After all data is loaded, delete SLF files.

Example I

You have sounding data for a region that was collected at 5 centimeter intervals, and another set of sounding data for the same region that was collected at 5 meter intervals. To improve loading time, load the dense (5 cm) data first; the partitions that will contain data are created. When the sparse dataset is loaded, data goes into the existing partitions.

Example II

The following example loads point data from the **points.slf** file into the POINTS table:

```
SDLOAD userid=herman/vampire slf=points.slf sdtable=points
```

The following output appears on screen. Only a portion of the output is shown:

```
SLF file points.slf contains 42545 records.
Starting load/partition process...
```

```
Commit point reached for POINTS_P000000001: 8680
Logical record count in SLF file: 8680

Commit point reached for POINTS_P000000002: 4773
Logical record count in SLF file: 13453

Commit point reached for POINTS_P000000003: 4817
Logical record count in SLF file: 18270
.
.
.

Load successfully completed.
```

Task 3: Verify Spatial Table Creation

To verify spatial table creation after the load process is complete, check the Spatial Data Option data dictionary by accessing the data dictionary views.

Example In the following example, the Spatial Data Option data dictionary views USER_MD_TABLES, USER_MD_PARTITIONS, USER_MD_COLUMNS, and USER_MD_DIMENSIONS are used to see new values after a load has finished:

```
SQL> SELECT md_table_name md_table, class, ptab_seq,
2 high_water_mark hwm
3 FROM  user_md_tables
4 WHERE md_table_name = 'POINTS';

MD_TABLE   CLASS           PTAB_SEQ  HWM
--------------------------------------------
POINTS     PARTITIONED    00000000G  10000

SQL> SELECT substr(partition_table_name,1,20)partition_table_name,
2 substr(common_hhcode,1,30) common_hhcode, common_level
3 FROM user_md_partitions
4 WHERE md_table_name = 'POINTS'
5 ORDER BY partition_table_name;

PARTITION_TABLE_NAME COMMN_HHCODE  COMMON_LEVEL
-------------------- ------------- ------------
POINTS_P000000001    7083800909020000         9
POINTS_P000000002    7083C00B0B020000        11
POINTS_P000000003    7083C40B0B020000        11
POINTS_P000000004    7083C80B0B020000        11
POINTS_P000000005    7083CC0B0B020000        11
POINTS_P000000006    7083D00A0A020000        10
```

```
POINTS_P000000007    7083E00B0B020000        11
POINTS_P000000008    7083E40B0B020000        11
POINTS_P000000009    7083E80B0B020000        11
POINTS_P00000000A    7083EC000D0D020000      13
POINTS_P00000000B    7083EC800D0D020000      13
POINTS_P00000000C    7083ECC00D0D020000      13
POINTS_P00000000D    7083ED0C0C020000        12
POINTS_P00000000E    7083EE540F0F020000      15
POINTS_P00000000F    70900606020000           6
15 rows selected.

SQL> SELECT substr(column_name,1,10) col_name, data_type
2 type,data_length length, data_precision prec, data_scale
3 scale, ndim, max_level, nullable n, partition_key p
4 FROM user_md_columns
5 WHERE md_table_name = 'POINTS'
6 ORDER BY column_name;

COL_NAME   TYPE   LENGTH  PREC   SCALE  NDIM   MAX_LEV  N P
---------- ----- ---------------- ---------- ---------- -----
DEPTH      NUMBER 22      38     10     0      0        Y N
LOCATION   HHCODE 13                    2      32       N Y

SQL> SELECT substr(column_name,1,10) col_name,
2 substr(dimension_name,1,5) dim,
3 dimension_number dim_num,lower_bound lb, upper_bound ub,
4 decimal_precision sc, recursion_level rl
5 FROM  user_md_dimensions
6 WHERE md_table_name = 'POINTS'
7 ORDER BY dimension_number;

COL_NAME   DIM    DIM_NUM        LB     UB     SC     RL
---------- ----- ---------------- ---------- ---------- ----
LOCATION   LON    1              –180   180    7      32
LOCATION   LAT    2              –90    90     7      31
```

Task 4: Correct Loading Errors

Use the following optional procedure to handle errors that occur during data loading:

1. For the current status of a load file, query the Spatial Data Option data dictionary view USER_MD_LOADER_ERRORS. This view indicates any files that failed to load.
2. Correct the problem as it is recorded by SD*Loader in the Spatial Data Option data dictionary.

3. Reload using the exact command line used for the original load. SD*Loader continues from where it halted.

Example In the following example, the Spatial Data Option data dictionary view USER_MD_LOADER_ERRORS is used to verify the status of a failed data load.

```
SQL> SELECT * FROM USER_MD_LOADER_ERRORS;

MD_TABLE_NAME      FILENAME       ROWS_LOADED
---------------------------------------------
POINTS             points.slf     610
```

Task 5: Perform Any Subsequent Loads

Use the following optional procedures to perform any subsequent data loads:

1. Use the MD_DDL.ALTER_MD_TABLE_HWM procedure to set the high water mark back to a higher value, if it was set low during the initial load, or to a higher level if there are to be more loads.
2. Execute the SD*Loader utility.

   To run multiple load sessions at the same time, use the following steps:

   2.1 Start a load process for one batch or data file on one CPU.

   2.2 Start a second load process for another batch or data file on another CPU.

   2.3 Repeat steps 2.1 and 2.2 until all available CPUs are utilized.
3. Optional. If sort during the load fails, manually remove the temporary files created.
4. Optional. After all data is loaded, delete all SLF files.

⚠ **Warning:** If more than one load process tries to load data into the same partition, one of the loads fails. Failed loads must be restarted later. For steps to correct this, see Task 4: Correct Loading Errors.

CHAPTER

4

Data Manipulation and Query

This chapter describes data manipulation and query methods.

The following topics are covered in this chapter:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- spatial data queries
- query modifiers

Data Definition Language (DDL)

This section discusses the procedures used to modify spatial tables, which includes the following:

- altering spatial table
- altering the HHCODE column
- dropping spatial table

Note: Some DDL commands are discussed throughout this manual. They are not included in this section to avoid repetition.

Altering Spatial Table

The method used to alter a spatial table differs, depending on whether the tables are partitioned or non–partitioned.

Partitioned Table

For a partitioned table, the following characteristics can be altered:

- compute mode
- high water mark
- attribute column

Alter Compute Mode: The following procedure alters the compute mode, which is used by SD*Loader when data is loaded into a partitioned table:

1. Determine the spatial tablename.
2. Invoke the MD_DDL.ALTER_MD_TABLE_CM procedure.

   The following is the syntax for this procedure:

   ```
   MD_DDL.ALTER_MD_TABLE_CM ([schema.|username,] sd_tablename,
   {'EXACT'|'ESTIMATE'})
   ```

Example

The following example changes the compute mode to ESTIMATE for the POINTS table:

```
SQL> EXECUTE MD_DDL.ALTER_MD_TABLE_CM ('POINTS', 'ESTIMATE');
```

For more information on altering partitioned tables, see Chapter 7, "SD*SQL Packages," *Oracle7 Spatial Data Option Reference and Administrator's Guide***.**

Alter High Water Mark: Altering the high water mark is effective only when new data is loaded using SD*Loader after the high water mark is changed. It does not redistribute data in existing partitions.

Use the following procedure to alter the high water mark:

1. Determine the spatial tablename.
2. Invoke the MD_DDL.ALTER_MD_TABLE_HWM procedure.

   The following is the syntax for this procedure:

   ```
   MD_DDL.ALTER_MD_TABLE_HWM ([schema.|username,] sd_tablename,
   high_water_mark)
   ```

Example I The following example changes the high water mark for the POINTS table to 1,500:

```
SQL> EXECUTE MD_DDL.ALTER_MD_TABLE_HWM('POINTS',1500);
```

Example II The following example alters the high water mark for the POINTS table to 200,000:

```
SQL> EXECUTE MD_DDL.ALTER_MD_TABLE_HWM ('POINTS',200000);
```

For more information on altering partitioned tables, see Chapter 7, "SD*SQL Packages," *Oracle7 Spatial Data Option Reference and Administrator's Guide***.**

Alter Attribute Column: Use the following procedure to alter an attribute column:

1. Determine the spatial tablename.
2. Generate the ALTER TABLE command template using the substitution variable %s to represent the partition tablename on which the command will operate.
3. Invoke the MD_DDL.ALTER_MD_TABLE procedure.

   The following is the syntax for this procedure:

   ```
   MD_DDL.ALTER_MD_TABLE ([schema.|username,] sd_tablename,–
   > %s sql_template [, continue_on_error])
   ```

Example I The following example adds the attribute column TEMPERATURE to the POINTS table:

```
SQL> EXECUTE MD_DDL.ALTER_MD_TABLE('herman','POINTS','alter –
> table %s add (temperature number (10,33))');
```

Example II The following example expands the DESCR column in the TIME table:

```
SQL> EXECUTE MD_DDL.ALTER_MD_TABLE('herman', 'TIME', 'alter –
> %s table modify (descr VARCHAR2 (60))');
```

For more information on altering partitioned tables, see Chapter 7, "SD*SQL Packages," *Oracle7 Spatial Data Option Reference and Administrator's Guide***.**

Non–Partitioned Table

Use the SQL ALTER TABLE command to alter a non–partitioned table. You can alter any column except the HHCODE column.

Altering HHCODE Column

To alter the format of an HHCODE column, use the following procedure:

1. Ensure that the data is in a form that can be reconverted and reloaded. If the spatial table has been modified, unload the data from the spatial table to either an Oracle7 table or the file system. If the spatial table has not been modified, use the original data sources to reconvert and reload.
2. Drop the spatial table using the MD_DDL.DROP_MD_TABLE procedure.

   For a description of the MD_DDL packages, see Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator's Guide.*
3. Re–create the table using the CREATE TABLE command.
4. Reregister the table using the MD_DDL.REGISTER_MD_TABLE procedure.
5. Redefine the HHCODE column using the MD_DDL.ADD_HHCODE_COLUMN.
6. Convert the data using the SD*Converter utility or a user–developed SLF converter.

   For a description of Spatial Data Option conversion and loading options, see Chapter 3, ”Setting Up a Spatial Database.”
7. Reload the data using the SD*Loader utility.

Dropping a Spatial Table

When a partitioned table is dropped, all associated partitions are dropped, followed by the spatial table. When a non–partitioned table is dropped, the spatial table is dropped.

If the MD_DDL.DROP_MD_TABLE procedure fails, an application error is raised.

Warning: Never use the SQL DROP TABLE command to drop a spatial table. If you do, the information in the Spatial Data Option data dictionary is not updated and is inconsistent with the state of the database.

Use the following procedure to drop a spatial table:

1. Determine the spatial table name.
2. Invoke the MD_DDL.DROP_MD_TABLE procedure.

   The following is the syntax for this procedure:

   ```
   MD_DDL.DROP_MD_TABLE ([schema.|username,] sd_tablename)
   ```

3. If the procedure fails, query the USER_MD_TABLES view. If the table still exists, re–execute the procedure.

For more information on MD_DDL.DROP_MD_TABLE, see Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.*

Example The following example drops the POINTS table and all associated partitions:

```
SQL> EXECUTE MD_DDL.DROP_MD_TABLE (’POINTS’);
```

Data Manipulation Language (DML)

This section discusses the commands used to query and manipulate data in spatial tables, which include the following:

- deleting data
- generating an HHCODE
- inserting data
- locking a partition or a table
- moving a record
- updating data

Note: To perform DML commands on spatial tables, you must use dynamic SQL with PL/SQL. To use DML commands in Pro*C, you must write a PL/SQL block in the Pro*C program.

Deleting Data

This section illustrates how to use the DELETE command for the following:

- attribute column of a partitioned table
- partition key column of a partitioned table
- non–partitioned table

Attribute Column of Partitioned Table

Use for following procedure to delete rows of data from a partitioned table based on the values in an attribute column.

1. Identify all partition tablenames and their status from the Spatial Data Option data dictionary views ALL_MD_PARTITIONS or USER_MD_PARTITIONS.
2. Raise an application error if the partition does not exist, and create the partition using the MD_PART.CREATE_INFERRED_PARTITION function.

   The following is the syntax for this function:

   ```
   MD_PART.CREATE_INFERRED_PARTITION ([schema.|username,]
   sd_tablename, hhcode_expression)
   ```

   For more information on exception handling, see ”MD_PART.CREATE_INFERRED PARTITION,” Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator's Guide.*
3. Use the SQL DELETE command to remove data from all the partitions identified in Step 1.

Partition Key Column of Partitioned Table

Use the following procedure to create a PL/SQL program that deletes rows of data from a partitioned table based on the values in a partition key column.

1. Declare PL/SQL variables for the partition key and the partition name.
2. Use MD_DML.GENHHCODE to generate an HHCODE of the partition key value for the data to be deleted.

   The following is the syntax for this function:

   ```
   MD_DML.GENHHCODE ([schema.|username,] sd_tablename,
   [hhcode_column_name,] dimension_1 [, dimension_n...]
   ```

   For more information on generating HHCODEs, see "MD_DML.GENHHCODE," Chapter 7, "SD*SQL Packages," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*
3. Determine the partition name that contains the HHCODE generated in Step 2 using the MD_PART.GET_PARTITION_NAME function.

   The following is the syntax for this function:

   ```
   MD_PART.GET_PARTITION_NAME ([schema.|username,] sd_tablename,
   hhcode_expression, wait_mode, partition_name)
   ```

   For more information on determining partition name, see "MD_PART.GET_PARTITION," Chapter 7, "SD*SQL Packages," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*
4. Raise an application error if the partition does not exist. Use the MD_PART.CREATE_INFERRED_PARTITION procedure to create the necessary partitions.
5. Use the SQL DELETE command with a WHERE clause.

Example

The following example of a PL/SQL program deletes the following records from the POINTS table:

Longitude:	–76.068641
Latitude:	46.14109

The following section of PL/SQL code declares variables.

```
DECLARE
   pk            RAW(255);      -- Partition Key
   pn            VARCHAR2(30);  -- Partition Name
   ps            INTEGER;       -- Partition Status
   crs           INTEGER;       -- Cursor
   rc            INTEGER;
BEGIN
```

The following section of PL/SQL code generates the HHCODE of the partition key value.

```
   pk := MD_DML.GENHHCODE('POINTS','location', –76.068641,
     46.14109);
```

The following section of PL/SQL code gets the partition name for the HHCODE of the partition key value.

```
   ps := MD_PART.GET_PARTITION_NAME('POINTS', pk, pn);
```

The following section of PL/SQL code raises an error if the partition is not found.

```
   IF (ps = MD_PART.NOTEXIST) THEN
     raise_application_error(–20000, 'Partition does not
          exist.');
   END IF;
```

The following section of PL/SQL code generates and executes the SQL DELETE command.

```
   crs := dbms_sql.open_cursor;
   dbms_sql.parse(crs, 'DELETE from  '|| pn||'  WHERE location =
     :pk', dbms_sql.v7);
   dbms_sql.bind_variable(crs, ':pk', pk);
   rc := dbms_sql.execute (crs);
   dbms_sql.close_cursor (crs);
COMMIT;
END;
```

Non–Partitioned Table

To delete data from a non–partitioned table generate and execute the SQL DELETE command following the same guidelines as for standard Oracle7 tables. To generate an HHCODE for the SQL DELETE command use the MD_DML.GENHHCODE function.

Generating HHCODE

Use the following procedure to generate an HHCODE value that can be used to insert data into a spatial table.

Note: This function returns a NULL if an error is encountered while generating the HHCODE.

1. Determine the spatial tablename.
2. Determine the spatial column name. If not specified, the HHCODE is generated using for the partition key column definition.
3. Invoke the MD_DML.GENHHCODE function.

   The following is the syntax for this function:

   ```
   MD_DML.GENHHCODE ([schema.|username,] sd_tablename,
   [hhcode_column_name], dimension_1 [, dimension_n...]
   ```

   For more information on generating HHCODEs, see ”MD_DML.GENHHCODE,” Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

Example

The following example of a PL/SQL program generates an HHCODE using the dimension definition of the partition key column of the POINTS table, which is LOCATION:

```
DECLARE
   hhc RAW (255);
BEGIN
   hhc := MD_DML.GENHHCODE
     ('POINTS', 'location', –76.1, 46.1);
END;
```

Inserting Data

This section illustrates how to use the SQL INSERT command to insert data into the following:

- partitioned table
- non–partitioned table

Partitioned Table

Use the following procedure to insert data into a partitioned table:

1. Determine the spatial table name.
2. Determine the spatial column name. If it is not specified, the HHCODE is generated using the partition key column definition.
3. Generate an HHCODE using the MD_DML.GENHHCODE function.

   The following is the syntax for this function:

   ```
   MD_DML.GENHHCODE ([schema.|username,] sd_tablename,
   [hhcode_column_name,] dimension_1 [, dimension_n...]
   ```

For more information on generating HHCODEs, see ”MD_DML.GENHHCODE,” Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.*

4. Determine the partition name in which to insert the HHCODE generated in Step 2 using the MD_PART.GET_PARTITION_NAME function.

   The following is the syntax for this function:

   ```
   MD_PART.GET_PARTITION_NAME ([schema.|username,] sd_tablename,
   hhcode_expression, wait_mode, partition_name)
   ```

   For more information on determining partition name, see ”MD_PART.GET_PARTITION,” Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.*

   The MD_PART.GET_PARTITION_NAME function also returns the status of the partition.

5. If the partition does not exist, create it using the MD_PART.CREATE_INFERRED_PARTITION procedure.

   The following is the syntax for this function:

   ```
   MD_PART.CREATE_INFERRED_PARTITION ([schema.|username,]
   sd_tablename, hhcode_expression)
   ```

   For more information on exception handling, see ”MD_PART.CREATE_INFERRED PARTITION,” Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.*

6. Use the SQL INSERT command to add data to the partition identified in Step 4.

   If you know that a block of records will be located in the same partition you can perform an array INSERT.

7. Optional. If the number of records in the partition exceeds the high water mark, you can subdivide the partition using the MD_PART.SUBDIVIDE_PARTITION procedure.

   The following is the syntax for this procedure:

   ```
   MD_PART.SUBDIVIDE_PARTITION ([schema.|username,]
   partition_name)
   ```

   For more information on subdividing partitions, see ”MD_DML.SUBDIVIDE_PARTITION,” Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.*

If the SD*Loader utility is used subsequently to insert data into a partition whose high water mark is exceeded, the utility subdivides the partition.

Example The following example of a PL/SQL program inserts the following data into the POINTS table:

Latitude:	126.25
Longitude:	77.2
Depth:	8888

The following section of PL/SQL code declares variables.

```
DECLARE
   pk           RAW(255);      -- Partition Key
   pn           VARCHAR2(30);  -- Partition Name
   ps           INTEGER;       -- Partition Status
   crs          INTEGER;       -- Cursor
   rc           INTEGER;       -- Return Code
   arch_flag    BOOLEAN := FALSE;
   cpart        VARCHAR2(30);
BEGIN
```

The following section of PL/SQL code generates the HHCODE of the partition key value.

```
   pk := MD_DML.GENHHCODE('POINTS','location', 126.25,
     77.2);
```

The following section of PL/SQL code gets the partition name for the HHCODE of the old partition key value.

```
   ps := MD_PART.GET_PARTITION_NAME('POINTS', pk, pn);
```

The following section of PL/SQL code creates an inferred partition if MD_PART.GET_PARTITION_NAME returns a status of NOTEXIST.

```
   IF (ps = MD_PART.NOTEXIST) THEN
     cpart := MD_PART.CREATE_INFERRED_PARTITION('POINTS', pk);
     ps := MD_PART.GET_PARTITION_NAME('POINTS', pk, pn);
   END IF;
```

The following section of PL/SQL code generates and executes the SQL INSERT command.

```
   crs := dbms_sql.open_cursor;
   dbms_sql.parse(crs, 'insert into  '|| pn||' (location,
     depth) '||' VALUES (:pk, 8888)', dbms_sql.v7);
   dbms_sql.bind_variable (crs, ':pk', pk);
   rc := dbms_sql.execute (crs);
   dbms_sql.close_cursor (crs);
   COMMIT;
END;
```

Non–Partitioned Table

To insert data into a non–partitioned table generate and execute the SQL INSERT command following the same guidelines as for standard Oracle7 tables. To generate an HHCODE for the SQL INSERT command use the MD_DML.GENHHCODE function.

Locking a Partition or Table

This section lists the following procedures that are used to lock and release spatial tables:

- locking a spatial table and all associated partitions
- locking a single partition
- releasing a lock

Note: If one of the partitions of the spatial table fails to lock, all other locks are released.

Locking a Spatial Table and All Associated Partitions

Use the following procedure to lock the spatial table and all associated partitions:

1. Determine the spatial table name.
2. Execute the MD_DML.LOCK_MD_TABLE procedure to lock the spatial table.

   The following is the syntax for this procedure:

   ```
   MD_DML.LOCK_MD_TABLE ([schema.|username,] sd_tablename
   [, 'EXCLUSIVE'|'ROW EXCLUSIVE'|'ROW SHARE'|'SHARE'|'SHARE
   UPDATE'|'SHARE ROW EXCLUSIVE', [lock_mode]])
   ```

   For more information on locking partitions, see "MD_DML.LOCK_MD_TABLE," Chapter 7, "SD*SQL Packages," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

Example

The following example locks the POINTS table:

```
EXECUTE MD_DML.LOCK_MD_TABLE('POINTS', 'EXCLUSIVE');
```

Locking a Single Partition

Use the following procedure to lock a single partition:

1. Determine the partition name you want to lock using the MD_PART.GET_PARTITION_NAME function.

   The following is the syntax for this function:

   ```
   MD_PART.GET_PARTITION_NAME ([schema.|username,] sd_tablename,
   hhcode_expression, wait_mode, partition_name)
   ```

   For more information on determining partition name, see "MD_PART.GET_PARTITION," Chapter 7, "SD*SQL Packages," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

2. Execute the SQL LOCK TABLE command.

Releasing a Lock

Use one of the following commands to release a lock:

- COMMIT saves changes made while a lock is held on a table.
- ROLLBACK discards changes made while a lock is held on a table.

Moving a Record

Updating the partition key column of a partitioned table can change the location of the data in space and can cause the data to move to a new partition. The MD_DML.MOVE_RECORD procedure updates the value of the partition key HHCODE column and, if needed, moves the record to a new partition.

Perform the following procedure to update the partition key column in a partitioned table and move data to another partition:

1. Determine the spatial table name.
2. Use the MD.HHENCODE function to generate an HHCODE for the old partition key value.

   The following is the syntax for this function:

   HHENCODE (*value*, *lower_boundary*, *upper_boundary*, *scale*
   [, *value*, *lower_boundary*, *upper_boundary*, *scale* ...])

   For more information on generating the partition key value, see "HHENCODE," Chapter 6, "SD*SQL Kernel Functions," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*
3. Use the MD.HHENCODE function to generate an HHCODE for the new partition key value.
4. Use the MD_PART.GET_PARTITION_NAME function to determine the partition name and partition status for the HHCODE of the new partition key value.

   The following is the syntax for this function:

   MD_PART.GET_PARTITION_NAME ([*schema.*|*username,*] *sd_tablename*, *hhcode_expression*, *wait_mode, partition_name*)

   For more information on determining partition name, see "MD_PART.GET_PARTITION," Chapter 7, "SD*SQL Packages," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*
5. If the partition does not exist create it using the MD_PART.CREATE_INFERRED_PARTITION procedure to create the necessary partitions.

   The following is the syntax for this function:

   MD_PART.CREATE_INFERRED_PARTITION ([*schema.*|*username,*]
   sd_tablename, *hhcode_expression*)

For more information on exception handling, see ”MD_PART.CREATE_INFERRED PARTITION,” Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator's Guide*.

6. Optional. Determine a WHERE clause based on attribute data in the spatial table.

7. Invoke the MD_DML.MOVE_RECORD procedure, specifying the optional attribute WHERE clause, to move the record from the old partition to the new.

   The following is the syntax for this procedure:

   ```
   MD_DML.MOVE_RECORD ([schema.|username,] sd_tablename,
   old_hhcode, new_hhcode [, attribute_where_clause])
   ```

   For more information on moving partitions, see ”MD_DML.MOVE_RECORD,” Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator's Guide*.

Example The following example moves a record from one partition to another in the POINTS table:

```
EXECUTE MD_DML.MOVE_RECORD('POINTS',
    MD.HHENCODE (–76.58976, –180, 180, 7, 45.996012, –90, 90, 7),
    MD.HHENCODE (–76.59, –180, 180, 7, 46.01, –90, 90, 7));
```

Updating Data

This section illustrates how to use the SQL UPDATE command for the following:

- attribute column of partitioned table
- partition key column of partitioned table
- non–partitioned table

Attribute Column of Partitioned Table

Use the following procedure to create a PL/SQL program that updates data in attribute columns of a partitioned table:

1. Declare PL/SQL variables.

2. Use the MD_DML.GENHHCODE function to generate an HHCODE of the partition key value for the data to be updated.

   The following is the syntax for this function:

   ```
   MD_DML.GENHHCODE ([schema.|username,] sd_tablename,
   [hhcode_column_name,] dimension_1 [, dimension_n...]
   ```

   For more information on generating HHCODEs, see ”MD_DML.GENHHCODE,” Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator's Guide*.

3. Use the MD_PART.GET_PARTITION_NAME function to determine the partition name and partition status for the HHCODE generated in Step 2.

   The following is the syntax for this function:

   ```
   MD_PART.GET_PARTITION_NAME ([schema.|username,] sd_tablename,
   hhcode_expression, wait_mode, partition_name)
   ```

   For more information on determining partition name, see ”MD_PART.GET_PARTITION,” Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.*

4. Raise an application error if the partition does not exist.

5. Generate and use the SQL UPDATE command.

Example The following example of a PL/SQL program changes the DEPTH value to 550 for the following records where:

Longitude:	–76.069794
Latitude:	46.157488

The following section of PL/SQL code declares variables.

```
DECLARE
   pk          RAW(255);         –– Partition Key
   pn             VARCHAR2(30);  –– Partition Name
   ps             INTEGER;       –– Partition Status
   crs            INTEGER;       –– Cursor
   rc             INTEGER;       –– Return Code
BEGIN
```

The following section of PL/SQL code generates the HHCODE of the partition key value.

```
   pk := MD_DML.GENHHCODE(’POINTS’,’location’, –76.069794,
     46.157488);
```

The following section of PL/SQL code gets the partition name for the HHCODE of the partition key value:

```
   ps := MD_PART.GET_PARTITION_NAME(’POINTS’, pk, pn);
   IF (ps = MD_PART.NOTEXIST) THEN
```

The following section of PL/SQL code raises an application error if the partition is not found:

```
     raise_application_error (–20000, ’Partition does not
          exist.’);
   END IF;
```

The following section of PL/SQL code generates and executes the SQL UPDATE command.

```
    crs := dbms_sql.open_cursor;
    dbms_sql.parse(crs, 'UPDATE  '|| pn||' SET depth=550 WHERE
      location = :pk ',dbms_sql.v7);
    dbms_sql.bind_variable(crs, ':pk', pk);
    rc := dbms_sql.execute (crs);
    dbms_sql.close_cursor (crs);
    COMMIT;
END;
```

Partition Key Column of Partitioned Table

Use the following procedure to create a PL/SQL program that updates data in the partition key column of a partitioned table:

1. Declare PL/SQL variables.
2. Use MD_DML.GENHHCODE to generate an HHCODE for the old partition key value.

   The following is the syntax for this function:

   `MD_DML.GENHHCODE ([`*`schema.`*`|`<u>*`username`*</u>*`,`*`]`*` sd_tablename`*`,`
   `[`*`hhcode_column_name,`*`]`*` dimension_1`*` [,`*` dimension_n...`*`]`

   For more information on generating HHCODEs, see "MD_DML.GENHHCODE," Chapter 7, "SD*SQL Packages," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

3. Use the MD_PART.GET_PARTITION_NAME function to determine the partition name and partition status for the HHCODE generated in Step 2.

   The following is the syntax for this function:

   `MD_PART.GET_PARTITION_NAME ([`*`schema.`*`|`<u>*`username`*</u>*`,`*`]`*` sd_tablename`*`,`
   `hhcode_expression``, `*`wait_mode, partition_name`*`)`

   For more information on determining partition name, see "MD_PART.GET_PARTITION," Chapter 7, "SD*SQL Packages," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

4. Raise an application error if the old partition does not exist.
5. Use MD_DML.GENHHCODE to generate an HHCODE for the old partition key value.

   The following is the syntax for this function:

   `MD_DML.GENHHCODE ([`*`schema.`*`|`<u>*`username`*</u>*`,`*`]`*` sd_tablename`*`,`
   `[`*`hhcode_column_name,`*`]`*` dimension_1`*` [,`*` dimension_n...`*`]`

For more information on generating HHCODEs, see ”MD_DML.GENHHCODE,” Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.*

6. Use the MD_PART.GET_PARTITION_NAME function to determine the partition name and partition status for the HHCODE of the new partition key value.

   The following is the syntax for this function:

   ```
   MD_PART.GET_PARTITION_NAME ([schema.|username,] sd_tablename,
   hhcode_expression, wait_mode, partition_name)
   ```

   For more information on determining partition name, see ”MD_PART.GET_PARTITION,” Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.*

7. If the partition does not exist create it using the MD_PART.CREATE_INFERRED_PARTITION procedure to create the necessary partitions.

   The following is the syntax for this function:

   ```
   MD_PART.CREATE_INFERRED_PARTITION ([schema.|username,]
   sd_tablename, hhcode_expression)
   ```

   For more information on exception handling, see ”MD_PART.CREATE_INFERRED PARTITION,” Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.*

8. If the old partition and the new partition are not the same, use the MD_DML.MOVE_RECORD procedure to update the partition key column.

   The following is the syntax for this procedure:

   ```
   MD_DML.MOVE_RECORD ([schema.|username,] sd_tablename,
   old_hhcode, new_hhcode [, attribute_where_clause])
   ```

   For more information on moving partitions, see ”MD_DML.MOVE_RECORD,” Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.*

9. If the old partition and the new partition are the same, use the SQL UPDATE command to update the partition key column.

Example The following example of a PL/SQL program updates the partition key value in the POINTS table as follows:

- old values:
  - Longitude: –76.05412
  - Latitude: 46.151302

- new values:
  - Longitude: 75
  - Latitude: –33

The following section of PL/SQL code declares variables.

```
DECLARE
   opk RAW(255);          –– Old Partition Key value
   opn VARCHAR2(30);      –– Old Partition Name
   ops INTEGER;           –– Old Partition Status
   npk RAW(255);          –– New Partition Key value
   npn VARCHAR2(30);      –– New Partition Name
   nps INTEGER;           –– New Partition Status
   crs INTEGER;           –– Cursor
   rc  INTEGER;           –– Return Code
BEGIN
```

The following section of PL/SQL code generates the HHCODE of the old partition key value.

```
   opk := MD_DML.GENHHCODE('POINTS', –76.05412, 46.151302);
```

The following section of PL/SQL code gets the partition name for the HHCODE of the old partition key value.

```
   ops := MD_PART.GET_PARTITION_NAME('POINTS',opk,opn);
```

The following section of PL/SQL code raises an application error if the partition is not found.

```
   IF (ops = MD_PART.NOTEXIST) THEN
     raise_application_error (–20000, 'Partition does not
          exist.');
   END IF;
```

The following section of PL/SQL code generates the HHCODE of the new partition key value.

```
   npk := MD_DML.GENHHCODE('POINTS', 75, –33);
```

The following section of PL/SQL code gets the partition name for the HHCODE of the new partition key value.

```
   nps := MD_PART.GET_PARTITION_NAME('POINTS', npk, npn);
```

The following section of PL/SQL code creates an inferred partition if MD_PART.GET_PARTITION_NAME returns a status of NOTEXIST.

```
   IF (nps = MD_PART.NOTEXIST) THEN
     npn := MD_PART.CREATE_INFERRED_PARTITION('POINTS', npk);
     nps := MD_PART.GET_PARTITION_NAME('POINTS', npk, npn);
   END IF;
```

The following section of PL/SQL code determines if the old partition and the new partition are the same. If they are not the same, it moves the data from the old partition to the new partition.

```
IF (opn != npn ) THEN
  MD_DML.MOVE_RECORD('POINTS', opk, npk);
```

The following section of PL/SQL code generates and executes the SQL UPDATE command to update the partition key column for data where the old partition and the new partition are the same.

```
    ELSE
      crs := dbms_sql.open_cursor;
      dbms_sql.parse (crs, 'UPDATE ' ||npn || ' set location =
           :npk '||'WHERE location = :opk ',
           dbms_sql.v7);
      dbms_sql.bind_variable (crs, ':npk', npk);
      dbms_sql.bind_variable (crs, ':opk', opk);
      rc := dbms_sql.execute (crs);
      dbms_sql.close_cursor (crs);
      COMMIT;
    END IF;
END;
```

Non–Partitioned Table

To update data in a non–partitioned table generate and execute the SQL UPDATE command following the same guidelines as for standard Oracle7 tables. To generate an HHCODE for the SQL UPDATE command use the MD_DML.GENHHCODE function.

Spatial Data Queries

The following query methods using Spatial Data Option are discussed in this section:

- package method
- custom method

A package or custom query can be performed for any HHCODE. However, if the HHCODE column is not the partition key, every partition must be searched to obtain the correct returns.

The package method requires use of a package that has all the necessary logic encapsulated, such as MD_WEX, to perform the query. Using this method, the following queries can be performed:

- query constraining only the partition key HHCODE column
- query constraining a partition key HHCODE column and an attribute column

The custom method requires the user to code the logic necessary to perform the query. Using this method the following queries can be performed:

- query constraining only the attribute column
- query constraining only the partition key HHCODE column
- query constraining a partition key HHCODE column and attribute column

Note: When querying, ranges must match the original definition or results will be inaccurate. The Spatial Data Option data dictionary USER_MD_DIMENSIONS view retrieves the original ranges.

Package Method

Use the following procedure to perform a query using the package method:

1. Optional. Clear all previous query specifications by executing the following procedure:

   ```
   MD_WEX.RESET_GLOBALS
   ```

2. Optional. Set the SQL filter using the following command where *sql_filter* represents the name of the SQL filter:

   ```
   MD_WEX.SET_SQL_FILTER (sql_filter)
   ```

   Default SQL filter is as follows:

   ```
   SELECT * FROM %s
   ```

3. Optional. Set the target type as table or view using the following procedure:

   ```
   MD_WEX.SET_TARGET_TYPE ({'TABLE'|'VIEW'})
   ```

   Default is 'TABLE.'

4. Optional. Set the HHCODE type as points, lines, or polygons using the following procedure:

   ```
   MD_WEX.SET_HHCODE_TYPE ('POINT'|'LINE')
   ```

   Default is 'POINT.'

5. Optional. Set the storage clause using the following procedure:

   ```
   MD_WEX.SET_STORAGE_CLAUSE (storage_clause_string)
   ```

6. Optional. Set the target tablespace using the following procedure:

   ```
   MD_WEX.SET_TARGET_TABLESPACE (tablespace_name)
   ```

   Default is the default tablespace defined in the user profile.

7. Optional. Set the dimension list indicating all of the HHCODE dimensions you want to include in the query, using the following procedure:

   ```
   MD_WEX.SET_DIMENSION_LIST (dimension_number
   [, dimension_number...])
   ```

   Default is all the dimensions defined in the Spatial Data Option data dictionary.

8. Set the RANGE, PROXIMITY, or POLYGON window using one of the following procedures:

   ```
   MD_WEX.SET_RANGE_WINDOW (lower_window_boundary_1,
   upper_window_boundary_1 [, lower_window_boundary_n,
   upper_window_boundary_n...])

   MD_WEX.SET_PROXIMITY_WINDOW (dimension_value_1,
   dimension_value_2, [ dimension_value_n,...] radius)
   MD_WEX.SET_POLYGON_WINDOW (x1,y1,x2,y2,x3,y3 [, xn, yn...])
   ```

9. Perform the extract using the EXTRACT command as follows.

   ```
   MD_WEX.EXTRACT ([schema.|username,] source_sd_tablename,
   [schema.|username,] target_object_name)
   ```

10. Optional. Drop the tables or views created by the query using the following procedure:

    ```
    MD_WEX.DROP_TARGET ([schema.|username,] target_object_name
    ```

Warning: SQL DROP TABLE cannot be used to drop the table or view created by MD_WEX.

11. Optional. To end the session or clean–up for a new query, reset all MD_WEX procedures back to their default values by using the following procedure:

    ```
    MD_WEX.RESET_GLOBALS
    ```

For more information on MD_WEX procedures, see Chapter 7, ”SD*SQL Packages,” *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

Example I

The following example illustrates a package proximity window query to select and view all POINTS records where:

depth:	50
center point1:	–76.1
center point2:	46.1
radius:	0.102
results table:	POINTS_TAB

```
EXECUTE MD_WEX.RESET_GLOBALS;
EXECUTE MD_WEX.SET_SQL_FILTER (’SELECT * FROM %s WHERE
      depth = 50’);
EXECUTE MD_WEX.SET_DIMENSION_LIST (1,2);
EXECUTE MD_WEX.SET_PROXIMITY_WINDOW (–76.1, 46.1, 0.102);
EXECUTE MD_WEX.EXTRACT (’herman’, ’POINTS’, ’herman’,
      ’POINTS_TAB’);
DESC POINTS_TAB
SELECT COUNT (*) FROM POINTS_TAB;
```

Example II

The following example extracts a two–dimensional range window of POINT data into a table and constrains the attribute column DEPTH to between 1000 and 2000:

```
EXECUTE MD_WEX.SET_DIMENSION_LIST(1,2);
EXECUTE MD_WEX.SET_HHCODE_TYPE( ’point’ );
EXECUTE MD_WEX.SET_RANGE_WINDOW(–76.1,–76.0,46.1,46.2);
EXECUTE MD_WEX.SET_SQL_FILTER(’SELECT * FROM %s ’||
    ’WHERE depth between 1000 and 2000’);
EXECUTE MD_WEX.EXTRACT(’herman’,’POINTS’,’herman’,
    ’POINTS_EXTRACT’ );
```

Custom Method

Use the following procedure to perform a query using the custom method:

1. Optional. Lock the spatial table.
2. Optional. Filter the HHCODE to include the dimensions you want to query.
3. Declare variables, if necessary.
4. Optional. Modify the WHERE clause of a SELECT statement.
5. If you are querying point data, identify the partition names by using the HHIDPART function defining the window as RANGE, PROXIMITY, or POLYGON as follows:

   ```
   HHIDPART ({'RANGE'|'PROXIMITY'|'POLYGON'}, COMMON_HHCODE,
   lower_boundary_1, upper_boundary_1, [lower_boundary_n,
   upper_boundary_n,...] window_definition)
   ```

   Define a range window definition as follows:

   ```
   lower_window_boundary_1, upper_window_boundary_1
   [, lower_window_boundary_n, upper_window_boundary_n...]
   ```

   Define a proximity window definition as follows:

   ```
   center_1, center_2, (center_n, ...) radius
   ```

   Define a polygon window definition as follows:

   ```
   x1, y1, x2, y2, x3, y3 [, xn, yn...]
   ```

6. If you are querying line data, identify the partition names by using the HHIDLPART function defining the window as RANGE, PROXIMITY, or POLYGON as follows:

   ```
   HHIDLPART ({'RANGE'|'PROXIMITY'|'POLYGON'}, COMMON_HHCODE,
   lower_boundary_1, upper_boundary_1, lower_boundary_2,
   upper_boundary_2, window_definition)
   ```

   Define a range window definition as follows:

   ```
   lower_window_boundary_1, upper_window_boundary_1
   [, lower_window_boundary_n, upper_window_boundary_n...]
   ```

   Define a proximity window definition as follows:

   ```
   center_1, center_2, radius
   ```

   Define a polygon window definition as follows:

   ```
   x1, y1, x2, y2, x3, y3 [, xn, yn...]
   ```

7. If you are querying point data and the partitions OVERLAP or ENCLOSE the query window, use the HHIDROWS or function to obtain appropriate returns as follows:

HHIDROWS ({'RANGE'|'PROXIMITY'|'POLYGON'}, *partition_key*, *lower_boundary_1*, *upper_boundary_1*, [*lower_boundary_n*, *upper_boundary_n*,...] *window_definition*)

Define a range window definition as follows:

lower_window_boundary_1, *upper_window_boundary_1*
[, *lower_window_boundary_n*, *upper_window_boundary_n...*]

Define a proximity window definition as follows:

center_1, center_2, (center_n, ...) radius

Define a polygon window definition as follows:

x1, *y1*, *x2*, *y2*, *x3, y3* [*, xn, yn*...]

8. If you are querying line data and the partitions OVERLAP or ENCLOSE the query window, use the HHIDLROWS or function to obtain appropriate returns as follows:

   HHIDLROWS ({'RANGE'|'PROXIMITY'|'POLYGON'}, *partition_key*, *lower_boundary_1*, *upper_boundary_1*, *lower_boundary_2*, *upper_boundary_2*, *window_definition*)

   Define a range window definition as follows:

   lower_window_boundary_1, *upper_window_boundary_1*, *lower_window_boundary_2*, *upper_window_boundary_2*

   Define a proximity window definition as follows:

   center_1, center_2, radius

   Define a polygon window definition as follows:

   x1, *y1*, *x2*, *y2*, *x3, y3* [*, xn, yn*...]

   Note: If the partition is INSIDE or EQUAL, all rows are included. You do not need to use the HHIDROWS or HHIDLROWS functions because all rows should be selected by the query.

   For more information on the query procedures, see Chapter 6, "SD*SQL Kernel Functions," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

Example I This example illustrates one method of implementing an attribute query across the multiple partitions of the POINTS table:

The following are some of the partitions belonging to the POINTS table.

```
POINTS_P000000001
POINTS_P000000002
POINTS_P000000003
POINTS_P000000004
POINTS_P000000005
```

```
POINTS_P000000006
POINTS_P000000007
POINTS_P000000008
```

The following creates a view that performs a UNION ALL on all partitions.

```
CREATE VIEW view1 AS
SELECT * FROM POINTS_p000000001
UNION ALL
SELECT * FROM POINTS_p000000002
UNION ALL
SELECT * FROM POINTS_p000000003
UNION ALL
SELECT * FROM POINTS_p000000004
UNION ALL
SELECT * from POINTS_p000000005
UNION ALL
SELECT * from POINTS_p000000006
UNION ALL
SELECT * from POINTS_p000000007
UNION ALL
SELECT * from POINTS_p000000008
.
.
```

The following executes the query against the view.

```
SELECT MIN(depth), MAX(depth)
   FROM view1;
```

Example II This example is a possible PL/SQL implementation for a UNION ALL query:

The following declares variables.

```
DECLARE
   hhc  RAW(255);
   pn VARCHAR2(30);  –– partition name
   crs INTEGER;
   rc INTEGER;
   depth number;
```

The following selects all the partition table names and status for the table POINTS.

```
   cursor c1 is
   SELECT partition_tablename,offline_status
     FROM user_md_partitions
     WHERE md_table_name = ’POINTS’;
BEGIN
   for rec in c1 loop
```

The following generates the SELECT statement and executes it using dynamic PL/SQL.

```
      crs := dbms_sql.open_cursor;
      dbms_sql.parse(crs,'SELECT location,depth FROM '||
      rec.partition_tablename,dbms_sql.v7);
      dbms_sql.define_column_raw(crs,1,hhc,255);
      dbms_sql.define_column(crs,2,depth);
      rc := dbms_sql.execute( crs );
      LOOP
           exit when dbms_sql.fetch_rows(crs) = 0;
           dbms_sql.column_value_raw(crs,1,hhc);
           dbms_sql.column_value(crs,2,depth);
           dbms_output.put_line( hhc||', '||to_char(depth) );
      END LOOP;
      dbms_sql.close_cursor( crs );
END;
```

The key point in this example is that there is an outer loop selecting all the partitions that belong to a spatial table and an inner loop selecting all the rows from each partition selected in the outer loop.

Example III The following example illustrates a custom query to select all POINTS records that fall within a RANGE window defined by the following boundaries:

dimension1:	–76.15	–76.05
dimension2:	45.8	46.185

```
SQL> col relationship format a20;
SQL> SELECT partition_tablename,
           HHIDPART('RANGE',common_hhcode,–180,180,–90,90,
           –76.15,–76.05,45.8,46.185) relationship
      FROM user_md_partitions
      WHERE md_table_name = 'POINTS';
```

The following determines how many rows from the POINTS_P000000001 partition lie within the same RANGE window defined by the following boundaries:

dimension1:	–76.15	–76.05
dimension2:	45.8	46.185

```
SELECT count(*)
FROM POINTS_p000000001
WHERE HHIDROWS('RANGE',location,–180,180,–90,90,
       –76.15,–76.05,45.8,46.185)  <> 'OUTSIDE';
```

Example IV The following example illustrates a custom query to select all POINTS records that fall within a RANGE window defined by the following boundaries:

dimension1:	–76.15	–76.05
dimension2:	45.8	46.185

The following declares variables.

```
DECLARE
   crs            INTEGER;
   rc             INTEGER;
```

The following checks whether a partition is either EQUAL or INSIDE a window separately from whether it ENCLOSES or OVERLAPS the window. If the partition is EQUAL or INSIDE, all rows in the partition are in the window and can be selected without applying the HHIDROWS function. If the partition is ENCLOSED or OVERLAPS the window, each row is checked individually with the HHIDROWS function.

```
cursor c1 is
   SELECT partition_tablename, offline_status
   FROM user_md_partitions
   WHERE md_table_name = ’POINTS’
   AND md.hhidpart(’RANGE’, common_hhcode,–180, 180, –90, 90
     –76.15, –76.05,45.8, 46.185) in (’EQUAL’, ’INSIDE’);
cursor c2 is
   SELECT partition_tablename, offline_status
   FROM user_md_partitions
   WHERE md_table_name = ’POINTS’
   AND md.hhidpart(’RANGE’, common_hhcode, –180, 180, –90, 90,
     –76.15, –76.05,45.8, 46.185) in (’ENCLOSES’, ’OVERLAP’);
hhc        RAW(255);
depth      number;
BEGIN
```

The following selects all the rows from the EQUAL or INSIDE partitions.

```
for rec in c1 LOOP
   crs := dbms_sql.open_cursor;
   dbms_sql.parse(crs, ’select location, depth
     FROM ’||rec.partition_tablename,dbms_sql.v7);
   dbms_sql.define_column_raw(crs,1,hhc,255);
   dbms_sql.define_column(crs, 2, depth);
   rc := dbms_sql.execute(crs);
   LOOP ––for all selected rows in current partition
     EXIT WHEN dbms_sql.fetch_rows(crs) = 0;
     dbms_sql.column_value_raw(crs,1,hhc);
```

```
      dbms_sql.column_value(crs,2,depth);
      dbms_output.put_line(hhc || ’ , ’ || to_char(depth));
    END LOOP;
    dbms_sql.close_cursor(crs);
END LOOP;
```

The following selects all the rows from the ENCLOSES or OVERLAPS partitions.

```
    crs := dbms_sql.open_cursor;
    dbms_sql.parse(crs, ’SELECT location, depth
      FROM ’||rec.partition_tablename||’
      WHERE hhidrows(’’RANGE’’,location, –180,180,–90,90,
           –76.15,–76.05, ’||’45.8, 46.185)
           <> ’’OUTSIDE’’’, dbms_sql.v7);
    dbms_sql.define_column_raw(crs,1,hhc,255);
    dbms_sql.define_column(crs, 2, depth);
    rc := dbms_sql.execute(crs);
LOOP
    EXIT WHEN dbms_sql.fetch_rows(crs) = 0;
    dbms_sql.column_value_raw(crs,1,hhc);
    dbms_sql.column_value(crs,2,depth);
    dbms_output.put_line(hhc || ’ , ’ || to_char(depth));
END LOOP;
dbms_sql.close_cursor(crs);
END LOOP;
END;
```

Example V The following example program assumes that the partitions being operated on do not subdivide during the execution of the program.

The following PL/SQL block performs this window query:

The following declares variables.

```
DECLARE
    crs INTEGER;
    rc  INTEGER;
```

The following identifies all the partitions that involved in the query; that is, all partitions that are INSIDE, OVERLAP, EQUAL, or ENCLOSE the window.

```
    cursor c1 is
      SELECT partition_tablename,offline_status
      FROM user_md_partitions
      WHERE md_table_name = ’POINTS’
      AND hhidpart (’RANGE’,common_hhcode,–180,180,–90,90,–76.1,
                    –76.0,46.1,46.2) != ’OUTSIDE’;
    hhc RAW(255);
    depth NUMBER;
BEGIN
    for rec in c1 LOOP
```

For each partition, the following selects all the rows that are INSIDE or on the BOUNDARY of the window.

```
      crs := dbms_sql.open_cursor;
      dbms_sql.parse(crs,
           'select location,depth from '||rec.partition_tablename
           || 'WHERE hhidrows (''RANGE'',location,–180,180,–90,90,
                  –76.1,–76.0,46.1,46.2) !=
                   ''OUTSIDE''',dbms_sql.v7);
      dbms_sql.define_column_raw(crs,1,hhc,255);
      dbms_sql.define_column(crs,2,depth);
      rc := dbms_sql.execute( crs );
      LOOP
           EXIT WHEN dbms_sql.fetch_rows(crs) = 0;
           dbms_sql.column_value_raw(crs,1,hhc);
           dbms_sql.column_value(crs,2,depth);
           dbms_output.put_line( hhc||', '||to_char(depth) );
      END LOOP;
      dbms_sql.close_cursor( crs );
    END LOOP;
END;
```

Query Modifiers

An essential feature of any query that has a dimensional component is the ability to manipulate data for querying. The SD*SQL HHCODE functions provide this access to the dimensional data.

Some functions are used with the selection of windows of data during a query, for further manipulation and analysis.

In addition to the functions mentioned to perform a straightforward query using Spatial Data Option there are several functions that can be used to vary the results in the following ways:

- filter the HHCODE
- rearrange order or grouping of the HHCODE
- modify the WHERE clause of a SELECT command

The HHCODE functions or combination of functions used as part of these categories varies according to desired results.

For more information on HHCODE functions, see Chapter 6, ”SD*SQL Kernel Functions,” *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

Filter HHCODE

The user can use the HHCOLLAPSE or HHCOMPOSE functions to filter original dimensions not necessary to the current query. HHCOLLAPSE functions ignore the dimensions of the HHCODE that are not necessary to the query. HHCOMPOSE creates a new HHCODE containing the dimensions important to the query.

Use the following procedure to integrate filtering the HHCODE into a query:

1. Invoke either the HHCOLLAPSE or HHCOMPOSE function to keep the dimensions that are necessary for the query.

   The syntax for these functions is as follows:

   ```
   HHCOLLAPSE (hhcode_expression, dimension_number
   [, dimension_number...])
   HHCOMPOSE (hhcode_expression, dimension_number
   [, dimension_number...])
   ```

   For more information, see ”HHCOLLAPSE,” and ”HHCOMPOSE,” Chapter 6, ”SD*SQL Kernel Functions,” *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

2. Perform either a package or custom query on the resulting HHCODE.

For example, the HHCODE can consist of latitude, longitude, depth, and time, but all you want is the information for July 7, 1993. You could use HHCOLLAPSE to filter out latitude, longitude, and depth; thus leaving you with time as the only dimension to search for the query.

Rearrange Order or Grouping

Sometimes it is useful to summarize or aggregate data having a common feature rather than looking at individual rows. This is done using the GROUP BY HHGROUP function in a SQL command. You can present rows in a sorted order using the SQL ORDER BY HHORDER command. In both cases, the HHGROUPBY and HHORDER functions are necessary to convert the HHCODE column to something the base SQL operator can use.

Use the following procedure to integrate rearranging order or grouping into a query:

1. Determine how data is currently ordered or grouped.
2. Determine what data is to be selected; for example, common code, minimum value, or maximum value.
3. Determine the spatial tablename.
4. Determine how data should be ordered or grouped to provide the desired query results.
5. Invoke the SQL SELECT command including either a GROUP BY HHGROUP or ORDER BY HHORDER clause as follows:

   The syntax for these functions is as follows:

   HHGROUP (*hhcode_expression*)
   HHORDER (*hhcode_expression*)

   For more information, see ”HHGROUP” and ”HHORDER,” Chapter 6, ”SD*SQL Kernel Functions,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.*
6. Perform either a package or custom query.

Note: The HHCODE functions are not recognized by the Oracle kernel as GROUP BY functions. As a result, an ORA–00979 error is returned if you attempt to use the HHCODE functions in a GROUP BY clause.

Example I

The following example demonstrates the ORA_00979 error and how to correctly use the GROUP BY HHGROUP clause:

```
SQL> SELECT COUNT(depth),HHSUBSTR(location,1,17) hhsubstr
             FROM POINTS_P000000001
             WHERE depth BETWEEN 50 AND 70
             GROUP BY HHGROUP(HHSUBSTR(location,1,17));
```

The preceding example returns the following error message:

```
SELECT COUNT(depth) count, HHSUBSTR(location,1,17) hhsubstr
                                                  *
             ERROR at line 1:
ORA–00979: not a GROUP BY expression
```

To correctly execute a SQL query containing an HHCODE expression in the SELECT list, you must surround the HHCODE expression with a SQL group function such as MIN, as demonstrated in the following example:

```
SQL> SELECT COUNT(depth) count,
             MIN(HHSUBSTR(location,1,17)) hhsubstr
             FROM POINTS_P000000001
             WHERE depth BETWEEN 50 AND 70
             GROUP BY HHGROUP(HHSUBSTR(location,1,17));
```

Example II

The following example illustrates a query on the POINTS_P000000001 partition to retrieve tiles of a size determined by an HHCODE substring of 17. The query should return the aggregate statistics of the number of records, the minimum, the maximum, and the average depth values.

```
SELECT count(depth), min(depth), max(depth), avg(depth)
FROM POINTS_p000000001
GROUP BY HHGROUP(HHSUBSTR(location,1,17))
   HAVING abs(max(depth)–min(depth)) < 75 ;
```

Modify WHERE Clause

HHCOMMONCODE, HHMATCH, and HHDISTANCE can be used to modify the WHERE clause to constrain the rows returned as follows:

- HHCOMMONCODE returns all HHCODEs that share the characteristics that were queried.
- HHMATCH returns the number of levels that match.
- HHDISTANCE is used to compute the Euclidean or Manhattan distance from a specified point.

  For example, if you want to know what the land characteristics are for a 1 mile distance from 45 degrees latitude and 60 degrees longitude, use the HHDISTANCE function to select the data in that area.

Use the following procedure in a custom query after obtaining returns from the HHIDPART or HHIDLPART command to improve efficiency of the HHIDROWS and HHIDLROWS commands:

1. Determine partition names essential to the query using the HHIDPART or HHIDLPART functions.

2. Determine the spatial tablename.
3. Determine which of the following functions works best in the SELECT command:
   - HHCOMMONCODE
   - HHMATCH
   - HHDISTANCE

   The syntax for HHCOMMONCODE is as follows:

   ```
   HHCOMMONCODE (hhcode_expression, hhcode_expression)
   ```

   The syntax for HHMATCH is as follows:

   ```
   HHMATCH (hhcode_expression, hhcode_expression)
   ```

   The syntax for HHDISTANCE is as follows:

   ```
   HHDISTANCE ({'EUCLID'|'MANHATTAN'}, hhcode_expression_1,
   hhcode_expression_2, lower_boundary_1, upper_boundary_1
   [, lower_boundary_n, upper_boundary_n...])
   ```

   For more information, see "HHCOMMONCODE," "HHMATCH," and "HHDISTANCE," Chapter 6, "SD*SQL Kernel Functions," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*
4. Invoke the SQL SELECT command with the modified WHERE clause.

Example The following example returns the common code between the minimum and maximum LOCATION HHCODEs in the POINTS_P000000001 partition.

```
SELECT HHCOMMONCODE(min (location), max (location))
    FROM POINTS_p000000001;
```

CHAPTER

5

Demos

This chapter describes how to run Spatial Data Option demos and SQL scripts.

The following topic is described in this chapter:

- Oracle7 Spatial Data Option demos and SQL scripts

Oracle7 Spatial Data Option Demos and SQL Scripts

This release of Spatial Data Option includes demos illustrating some possible uses of the product, and several examples of SQL scripts. The demo programs are designed to illustrate Spatial Data Option features and characteristics for the following distinct datasets: points, lines, plane, and time.

This section provides information about the following topics:

- demo and SQL script location
- prerequisites for running demos
- running Oracle7 Spatial Data Option demos
- removing Oracle7 Spatial Data Option demos

Demo and SQL Script Location

The Spatial Data Option demos and SQL scripts are installed in subdirectories as follows:

`$ORACLE_HOME/md/demo/`*`sub_directory`*

where *sub_directory* is one of the following:

points	is the directory where the points dataset is located.
lines	is the directory where the lines dataset is located.
plane	is the directory where the plane dataset is located.
time	is the directory where the time dataset is located.

Prerequisites

The following prerequisites must be met before you can run the demos or SQL scripts:

1. The demo user Herman must exist.

   The user Herman is created automatically during the Spatial Data Option installation process. The user Herman’s default tablespace is SYSTEM; however, you can specify a different default tablespace using the ALTER USER command.

   For syntax of the ALTER USER command, see the *Oracle7 Server SQL Reference.*

2. To run all demos, the user Herman must have at least 20 Mb free tablespace available.

3. To run the **pointsallp** demo, the Spatial Data Option PL/SQL packages must be pinned. For more information on pinning PL/SQL packages, see Chapter 4, "Administration," *Oracle7 Spatial Data Option Reference and Administrator's Guide.*

Running Oracle7 Spatial Data Option Demos

Each dataset contains standalone demos that automatically set up necessary dependencies. This section describes how to perform the following tasks:

- execute demo datasets
- execute demos individually

Execute Demo Datasets

You can execute all demos for a dataset at one time. To run a demo dataset, enter the following commands, substituting a dataset type and dataset name for *dataset_type* and *dataset_name* from Table 5 – 1:

```
% cd $ORACLE_HOME/md/demo/dataset_type
% make –f ins_dataset_type.mk dataset_name
```

Dataset Type	Dataset Name	Action
points	**pointsallp**	executes partitioned POINTS table demos
	pointsalln	executes non–partitioned POINTS table demos
lines	**linesall**	executes lines demos
plane	**planeall**	executes plane demos
time	**timeall**	executes time demos

Table 5 – 1 Dataset Type, Name, and Action

For example, entering the following commands executes all the partitioned POINTS table demos in turn:

```
% cd $ORACLE_HOME/md/demo/points
% make –f ins_points.mk pointsallp
```

The preceding command performs the following actions:

- creates a partitioned POINTS table
- registers the POINTS table
- adds an HHCODE column to POINTS table
- runs several examples of SD*Converter to convert the datafile to an SLF file
- runs several examples that demonstrate how to use the SLF library to convert a datafile to an SLF file

- runs SD*Loader to load one of the converted SLF files into the partitioned POINTS table
- runs SQL scripts that use HHCODE functions such as HHLEVELS, HHPRECISION, HHLENGTH, and HHCOMPOSE
- runs SD*SQL scripts
- deletes the partitioned POINTS demo tables and SLF files

For more information on Spatial Data Option utilities such as SD*Converter and SD*Loader, see Chapter 5, ”Utilities,” *Oracle7 Spatial Data Option Reference and Administrator’s Guide.*

Execute Demos Individually

Each of the demos in this section can be executed independently to view a specific feature of Spatial Data Option. The following tables list the demo name, the demo action, and what function of Spatial Data Option is illustrated by the demo.

points Demos: Enter the following commands, substituting a demo name for *demo* from Table 5 – 2:

```
% cd $ORACLE_HOME/md/demo/points
% make –f ins_points.mk demo
```

points Demos and SQL Scripts		
Demo Name	**Demo Action**	**Demonstrate**
points1	create partitioned POINTS table	SQL command
points2	convert using filedata/dicttable method	SD*Converter
points2b	convert using filedata/dictfile method	SD*Converter
points2c	convert using tabledata method	SD*Converter
points2d	convert using SLF library and dicttable	SLF library
points2e	convert using SLF library and dictfile	SLF library
points3	load SLF file	SD*Loader
points7	HHCODE functions	HHCODE functions
points8	SD*SQL scripts	SQL scripts

Table 5 – 2 points Demos and SQL Scripts

SQL Scripts: The SQL scripts in Table 5 – 3 illustrate Spatial Data Option INSERT, UPDATE, DELETE, and partition maintenance operations.

To execute all the SQL scripts at one time, enter the following commands:

```
% cd $ORACLE_HOME/md/demo/points
% make –f ins_points.mk points8
```

The **points8** demo runs all the SQL scripts listed in Table 5 – 3.

To execute each SQL script individually, perform the following tasks.

1. Execute the **points3** demo, which creates the partitioned POINTS table, by entering the following commands:

   ```
   % cd $ORACLE_HOME/md/demo/points
   % make –f ins_points.mk points3
   ```

2. Execute the **ptsdml.sq**l script, which creates the POINTS_DML PL/SQL package, by entering the following commands:

   ```
   % cd $ORACLE_HOME/md/demo/points
   % sqlplus herman/vampire @ptsdml.sql
   ```

3. Execute each SQL script by entering the following commands, substituting a SQL script name for *script* from Table 5 – 3:

   ```
   % cd $ORACLE_HOME/md/demo/points
   % sqlplus herman/vampire @script
   ```

Spatial Data Option SQL Scripts		
SQL Script	**Action**	**Dependencies**
ptsins.sql	INSERT into partition	
ptsupdep.sql	UPDATE attribute column	
ptsdel.sql	DELETE from partition	
ptsuploc.sql	UPDATE HHCODE column (move record between partitions)	
ptssubd.sql	subdivide partition	
ptscrein.sql	create inferred partition	

Table 5 – 3 Spatial Data Option SQL Scripts

lines Demos: Enter the following commands, substituting a demo name for *demo* from Table 5 – 4:

```
% cd $ORACLE_HOME/md/demo/lines
% make –f ins_lines.mk demo
```

lines Demos		
Demo Name	**Demo Action**	**Demonstrates**
lines1	create partitioned LINES table	SQL command
lines2	convert using filedata/dicttable method	SD*Converter
lines2b	convert using filedata/dictfile method	SD*Converter
lines2c	convert using tabledata method	SD*Converter
lines2d	convert using SLF library and dicttable method	SLF library
lines2e	convert using SLF library and dictfile method	SLF library
lines3	load SLF file	SLF library

Table 5 – 4 lines Demos

plane Demos: Enter the following commands, substituting a demo name for *demo* from Table 5 – 5:

```
% cd $ORACLE_HOME/md/demo/plane
% make –f ins_plane.mk demo
```

plane Demos		
Demo Name	**Demo Action**	**Demonstrates**
plane1	create partitioned PLANE table	SQL command
plane2	convert using filedata/dicttable method	SD*Converter
plane2b	convert using filedata/dictfile method	SD*Converter
plane2c	convert using tabledata method	SD*Converter
plane2d	convert using SLF library and dicttable method	SLF library
plane2e	convert using SLF library and dictfile method	SLF library
plane3	load SLF file	SLF library

Table 5 – 5 plane Demos

time Demos: Enter the following commands, substituting a demo name for *demo* from Table 5 – 6:

```
% cd $ORACLE_HOME/md/demo/time
% make –f ins_time.mk demo
```

time Demos		
Demo Name	**Demo Action**	**Demonstrates**
time1	create partitioned TIME table	SQL command
time2	convert using filedata/dicttable method	SD*Converter
time2b	convert using filedata/dictfile method	SD*Converter
time2c	convert using tabledata method	SD*Converter
time2d	convert using SLF library and dicttable method	SLF library
time3	load SLF file	SLF library

Table 5 – 6 time Demos

Removing Oracle7 Spatial Data Option Demo s

Enter the appropriate command from this section to remove all files created by the demos and to drop demo tables.

points Demos

Enter the following commands to change directories and remove extra files created during the **points** demos:

```
% cd $ORACLE_HOME/md/demo/points
% make –f ins_points.mk ptscln
```

lines Demos

Enter the following commands to change directories and remove extra files created during the **lines** demos:

```
% cd $ORACLE_HOME/md/demo/lines
% make –f ins_lines.mk lnscln
```

plane Demos

Enter the following commands to change directories and remove extra files created during the **plane** demos:

```
% cd $ORACLE_HOME/md/demo/plane
% make –f ins_plane.mk plncln
```

time Demos

Enter the following commands to change directories and remove extra files created during the **time** demos:

```
% cd $ORACLE_HOME/md/demo/time
% make –f ins_time.mk timcln
```

Index

Reader's Comment Form

Oracle7 Spatial Data Option™ Application Developer's Guide
Part No. A43695–1

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the topic, chapter, and page number below:

____

____

____

____

____

____

____

____

Please send your comments to:

Documentation Manager Government Products Division
Oracle Corporation
500 Oracle Parkway Box 659204
Redwood Shores, California 94065
Phone: 1.415.506.2503 FAX: 1.415.506.7408

If you would like a reply, please give your name, address, and telephone number below:

____

____

____

Thank you for helping us improve our documentation.