Database Programming

Indexes and Synonyms

Objectives

This lesson covers the following objectives:

- Define an index and its use as a schema object
- Define ROWID and its use in locating information in a database
- Name the conditions that cause an index to be created automatically
- Create and execute a CREATE INDEX and DROP INDEX statement
- Create and execute a function-based index
- Create private and public synonyms

Purpose

Can you imaging going to a book store or a library and finding all the books stacked on shelves row after row in no real order?

The only way to locate what you're looking for would be to examine every book in every row! Not very efficient, wouldn't you agree?

You may be surprised to know that retrieval of data from a database is exactly like the rows of books. For every query, a full table scan happens.

Purpose (cont.)

Fortunately, finding data in a database is more efficient than having to examine every row. Oracle uses an index to speed up the retrieval of rows. In this lesson, you will learn how and when to create an index as well as how to delete an index.

Also, in this lesson, you will learn how to create user-friendly names for database objects. Much like Internet Web addresses that eliminate having to know the Internet protocol address, synonyms provide a way to give an object a simpler name. (Sounds like aliases, doesn't it?)

Indexes

An Oracle Server index is a schema object that can speed up the retrieval of rows by using a pointer. Indexes can be created explicitly or automatically. If you do not have an index on the column you're selecting, then a full table scan occurs.

An index provides direct and fast access to rows in a table. Its purpose is to reduce the necessity of disk I/O (input/output) by using an indexed path to locate data quickly. The index is used and maintained automatically by the Oracle Server. Once an index is created, no direct activity is required by the user.

Indexes (cont.)

A ROWID is a base 64 string representation of the row address containing block identifier, row location in the block, and the database file identifier. Indexes use ROWID's because they are the fastest way to access any particular row.

Indexes (cont.)

Indexes are logically and physically independent of the table they index. This means that they can be created or dropped at any time and have no effect on the base tables or other indexes.

ID	TITLE	DURATION	ARTIST	TYPE_CODE
45	It's Finally Over	5 min	The Hobbits	12
46	I'm Going to Miss My Teacher	2 min	Jane Pop	12
47	Hurrah for Today	3 min	The Jubilant Trio	77
48	Meet Me At the Altar	6 min	Bobby West	1
49	Let's Celebrate	8 min	The Celebrants	77
50	All These Years	10 min	Diana Crooner	88

Indexes (cont.)

Note: When you drop a table, corresponding indexes are also dropped.

ID	TITLE	DURATION	ARTIST	TYPE_CODE
45	It's Finally Over	5 min	The Hobbits	12
46	I'm Going to Miss My Teacher	2 min	Jane Pop	12
47	Hurrah for Today	3 min	The Jubilant Trio	77
48	Meet Me At the Altar	6 min	Bobby West	1
49	Let's Celebrate	8 min	The Celebrants	77
50	All These Years	10 min	Diana Crooner	88

Types of Indexes

Two types of indexes can be created:

1. Unique index: The Oracle Server automatically creates this index when you define a column in a table to have a PRIMARY KEY or a UNIQUE KEY constraint. The name of the index is the name given to the constraint. Although you can manually create a unique index, it is recommended that you create a unique constraint in the table, which implicitly creates a unique index.

Types of Indexes (cont.)

Two types of indexes can be created (cont.):

2. Nonunique index: This is an index that a user can create to speed up access to the rows. For example, to optimize joins, you can create an index on the FOREIGN KEY column, which speeds up the search to match rows to the PRIMARY KEY column.

Creating an Index

Create an index on one or more columns by issuing the CREATE INDEX statement:

```
CREATE INDEX index_name
ON  table_name( column...,column)
```

To create an index in your schema, you must have the CREATE TABLE privilege. To create an index in any schema, you need the CREATE ANY INDEX privilege or the CREATE TABLE privilege on the table on which you are creating the index. Null values are not included in the index.

Creating an Index (cont.)

For example, to improve the speed of query access to the TITLE column in the DJs on Demand D_CDS table:

```
CREATE INDEX d_cds_idx
ON d_cds(title);
```

When to Create an Index

An index should be created only if:

- The column contains a wide range of values
- A column contains a large number of null values
- One or more columns are frequently used together in a WHERE clause or a join condition
- The table is large and most queries are expected to retrieve less than 2-4% of the rows.

When Not to Create an Index

When deciding whether or not to create an index, more is not always better. Each DML operation (INSERT, UPDATE, DELETE) that is performed on a table with indexes means that the indexes must be updated.

The more indexes you have associated with a table, the more effort it takes to update all the indexes after the DML operation.

When Not to Create an Index (cont.)

It is usually not worth creating an index if:

- The table is small
- The columns are not often used as a condition in the query
- Most queries are expected to retrieve more than 2-4 % of the rows in the table
- The table is updated frequently
- The indexed columns are referenced as part of an expression

Composite Index

A composite index (also called a "concatenated" index) is an index that you create on multiple columns in a table. Columns in a composite index can appear in any order and need not be adjacent in the table.

Composite indexes can speed retrieval of data for SELECT statements in which the WHERE clause references all or the leading portion of the columns in the composite index.

Composite Index (cont.)

Null values are not included in the composite index.

To optimize joins, you can create an index on the FOREIGN KEY column, which speeds up the search to match rows to the PRIMARY KEY column.

The optimizer does not use an index if the WHERE clause contains the IS NULL expression.

Confirming Indexes

Confirm the existence of indexes from the USER_INDEXES data dictionary view. You can also check the columns involved in an index by querying the USER_IND_COLUMNS view.

The query shown on the next slide is a join between the USER_INDEXES table (names of the indexes and their uniqueness) and the USER_IND_COLUMNS (names of the indexes, table names, and column names) table.

Confirming Indexes (cont.)

```
SELECT ic.index_name,
               ic.column_name,
               ic.column_position col_pos,
               ix.uniqueness
FROM user_indexes ix, user_ind_columns ic
WHERE ic.index_name = ix.index_name
AND ic.table_name = 'EMPLOYEES';
```

INDEX NAME	COLUMN NAME	COL POS	UNIQUENESS
EMP_EMAIL_UK	EMAIL	1	UNIQUE
EMP_EMP_ID_PK	EMPLOYEE_ID	1	UNIQUE
EMP_DEPARTMENT_IX	DEPARTMENT_ID	1	NONUNIQUE
EMP_JOB_IX	JOB_ID	1	NONUNIQUE
EMP_MANAGER_IX	MANAGER_ID	1	NONUNIQUE
EMP_NAME_IX	LAST_NAME	1	NONUNIQUE
EMP_NAME_IX	FIRST_NAME	2	NONUNIQUE

Function-based Indexes

A function-based index stores the indexed values and uses the index based on a SELECT statement to retrieve the data.

A function-based index is an index based on expressions.

The index expression is built from table columns, constants, SQL functions, and user-defined functions.

Function-based Indexes (cont.)

Function-based indexes are useful when you don't know in what case the data was stored in the database. For example, you can create a function-based index that can be used with a SELECT statement using UPPER in the WHERE clause. The index will be used in this search.

```
CREATE INDEX upper_last_name_idx
ON employees (UPPER(last_name));

SELECT *
FROM employees
WHERE UPPER(last_name) = 'KING';
```

Function-based Indexes (cont.)

Function-based indexes defined with the UPPER(column_name) or LOWER(column_name) keywords allow case-insensitive searches.

If you don't know how the employee last names were entered into the database, you could still use the index by entering uppercase in the SELECT statement.

When a query is modified using an expression in the WHERE clause, the index won't use it unless you create a function-based index to match the expression.

Function-based Indexes (cont.)

For example, the following statement allows for case-insensitive searches using the index:

```
CREATE INDEX upper_last_name_idx
ON d_partners (UPPER(last_name));

SELECT *
FROM d_partners
WHERE UPPER(last_name) = 'CHO';
```

Function-based Indexes (cont.)

To ensure that the Oracle Server uses the index rather than performing a full table scan, be sure that the value of the function is not null in subsequent queries. For example, the following statement is guaranteed to use the index, but without the WHERE clause the Oracle Server may perform a full table scan:

```
SELECT *
FROM d_partners
WHERE UPPER (last_name) IS NOT NULL
ORDER BY UPPER (last_name);
```

The Oracle Server treats indexes with columns marked DESC as function-based indexes. The columns marked DESC are sorted in descending order.

Function-based Indexes (cont.)

All of these examples use the UPPER and LOWER functions, but it is worth noticing that while these two are very frequently used in Function Based Indexes, the Oracle database is not limited to them.

Any valid Oracle Built-in function can be used, as can Database Functions that you write yourself. If you are writing your own functions to use in a Function Based Index, you must include the key word DETERMINISTIC in the function header.

Function-based Indexes (cont.)

In mathematics, a **deterministic system** is a system in which no randomness is involved in the development of future states of the system. Deterministic models therefore produce the same output for a given starting condition.

In Oracle, Deterministic declares that a function, when given the same inputs, will always return the exact same output. You must tell Oracle that the function is DETERMINISTIC and will return a consistent result given the same inputs.

Function-based Indexes (cont.)

The built-in SQL functions UPPER, LOWER, and TO_CHAR are all defined as deterministic by Oracle so you can create an index on a column using any of these three functions.

Function-based Indexes (cont.)

Another example of Function Based Indexes is shown here. The d_events table is queried to find any events planned for the Month of May.

```
SELECT *
FROM d_events
WHERE TO_CHAR(event_date,'mon') = 'may'
```

Results | Explain | Describe | Saved SQL | History

Query Plan

Operation	Options	Object	Rows	Time	Cost	Bytes	Filter Predicates *	Access Predicates
SELECT STATEMENT			1	1	3	79		
TABLE ACCESS	FULL	D_EVENTS	1	1	3	79	TO_CHAR(INTERNAL_FUNCTION("EVENT_DATE"),'mon') = 'may'	

* Unindexed columns are shown in red

As you can see, this query results in a Full Table Scan, which can be a very expensive operation if the table is big. Even though the event_date column is indexed, the index is not used due to the TO_CHAR expression.

Function-based Indexes (cont.)

Once we create the following Function Based Index, we can run the same query, but this time avoid the expensive Full Table Scan.

```
CREATE INDEX d_evnt_dt_indx ON d_events (to_char(event_date,'mon'))
```

```
SELECT *
FROM d_events
WHERE TO_CHAR(event_date,'mon') = 'may'
```

Results | Explain | Describe | Saved SQL | History

Query Plan

Operation	Options	Object	Rows	Time	Cost	Bytes	Filter Predicates *	Access Predicates
SELECT STATEMENT			1	1	2	79		
TABLE ACCESS	BY INDEX ROWID	D_EVENTS	1	1	2	79		
INDEX	RANGE SCAN	D_EVNT_DT_INDX	1	1	1			TO_CHAR(INTERNAL_FUNCTION("EVENT_DATE"),'mon') = 'may'

* Unindexed columns are shown in red

Now, Oracle can use the index on the events_date column.

Removing an Index

You cannot modify indexes. To change an index, you must drop it and then re-create it. Remove an index definition from the data dictionary by issuing the DROP INDEX statement.

To drop an index, you must be the owner of the index or have the DROP ANY INDEX privilege. If you drop a table, indexes and constraints are automatically dropped, but views and sequences remain.

Removing an Index (cont.)

In the syntax:

index is the name of the index.

```
DROP INDEX upper_last_name_idx;
DROP INDEX index;
DROP INDEX d_cds_idx;
```

SYNONYM

In SQL, as in language, a synonym is a word or expression that is an accepted substitute for another word.

Synonyms are used to simplify access to objects by creating another name for the object. Synonyms can make referring to a table owned by another user easier and shorten lengthy object names.

SYNONYM (cont.)

For example, to refer to the amys_copy_d_track_listings table in your classmate's schema, you can prefix the table name with the name of the user who created it followed by a period and then the table name, as in USMA_SBHS_SQL01_S04.amy.

SYNONYM (cont.)

Creating a synonym eliminates the need to qualify the object name with the schema and provides you with an alternative name for a table, view, sequence, procedure, or other object. This method can be especially useful with lengthy object names, such as views.

The database administrator can create a public synonym accessible to all users and can specifically grant the CREATE PUBLIC SYNONYM privilege to any user, and that user can create public synonyms.

SYNONYM (cont.)

```
CREATE [PUBLIC] SYNONYM synonym
FOR object;
```

In the syntax:

- PUBLIC: creates a synonym accessible to all users
- synonym: is the name of the synonym to be created
- object: identifies the object for which the synonym is created

```
CREATE SYNONYM dj_titles
FOR d_cds;
```

SYNONYM Guidelines

Guidelines:

- The object cannot be contained in a package.
- A private synonym name must be distinct from all other objects owned by the same user.

To remove a synonym:

```
DROP [PUBLIC] SYNONYM name_of_synonym

DROP SYNONYM dj_titles;
```

Confirming a SYNONYM

The existence of synonyms can be confirmed by querying the USER_SYNONYMS data dictionary view.

Column Name	Contents
Synonym_name	Name of the synonym.
Table_name	Owner of the object referenced by the synonym.
Table_owner	Name of the object referenced by the synonym.
Db_link	Database link referenced in a remote synonym.

Terminology

Key terms used in this lesson included:

- Composite index
- Confirming index
- CREATE PUBLIC SYNONYM
- DROP INDEX
- Function-based index
- Non-unique index
- Synonym
- Unique index

Summary

In this lesson, you should have learned how to:

- Define an index and its use as a schema object
- Define ROWID and its use in locating information in a database
- Name the conditions that cause an index to be created automatically
- Create and execute a CREATE INDEX and DROP INDEX statement
- Create function-based indexes
- Create private and public synonyms