12

Other Database Objects

Schedule:	Timing	Topic
	20 minutes	Lecture
	20 minutes	Practice
	40 minutes	Total

Objectives

After completing this lesson, you should be able to do the following:

- **Create, maintain, and use sequences**
- **Create and maintain indexes**
- **Create private and public synonyms**

Lesson Aim

In this lesson, you learn how to create and maintain some of the other commonly used database objects. These objects include sequences, indexes, and synonyms.

Database Objects

Object	Description
Table	Basic unit of storage; composed of rows and columns
View	Logically represents subsets of data from one or more tables
Sequence	Generates primary key values
Index	Improves the performance of some queries
Synonym	Alternative name for an object

Database Objects

Many applications require the use of unique numbers as primary key values. You can either build code into the application to handle this requirement or use a sequence to generate unique numbers.

If you want to improve the performance of some queries, you should consider creating an index. You can also use indexes to enforce uniqueness on a column or a collection of columns.

You can provide alternative names for objects by using synonyms.

What Is a Sequence?

A sequence:

- Automatically generates unique numbers
- Is a sharable object
- Is typically used to create a primary key value
- Replaces application code
- Speeds up the efficiency of accessing sequence values when cached in memory

What Is a Sequence?

A sequence is a user created database object that can be shared by multiple users to generate unique integers.

A typical usage for sequences is to create a primary key value, which must be unique for each row. The sequence is generated and incremented (or decremented) by an internal Oracle routine. This can be a time-saving object because it can reduce the amount of application code needed to write a sequence-generating routine.

Sequence numbers are stored and generated independently of tables. Therefore, the same sequence can be used for multiple tables.

The CREATE SEQUENCE Statement Syntax

Define a sequence to generate sequential numbers automatically:

```
CREATE SEQUENCE sequence
       [INCREMENT BY n]
       [START WITH n]
       [{MAXVALUE n | NOMAXVALUE}]
       [{MINVALUE n | NOMINVALUE}]
       [{CYCLE | NOCYCLE}]
       [{CACHE n | NOCACHE}];
```

Creating a Sequence

Automatically generate sequential numbers by using the `CREATE SEQUENCE` statement.

In the syntax:

`sequence`	is the name of the sequence generator
`INCREMENT BY` *`n`*	specifies the interval between sequence numbers where *n* is an integer (If this clause is omitted, the sequence increments by 1.)
`START WITH` *`n`*	specifies the first sequence number to be generated (If this cla use is omitted, the sequence starts with 1.)
`MAXVALUE` *`n`*	specifies the maximum value the sequence can generate
`NOMAXVALUE`	specifies a maximum value of 10^27 for an ascending sequence and –1 for a descending sequence (This is the default option.)
`MINVALUE` *`n`*	specifies the minimum sequence value
`NOMINVALUE`	specifies a minimum value of 1 for an ascending sequence and –(10^26) for a descending sequence (This is the default option.)
`CYCLE | NOCYCLE`	specifies whether the sequence continues to generate values after reaching its maximum or minimum value (`NOCYCLE` is the default option.)
`CACHE` *`n`* `| NOCACHE`	specifies how many values the Oracle server preallocates and keep in memory (By default, the Oracle server caches 20

Creating a Sequence

- Create a sequence named **DEPT_DEPTID_SEQ** to be used for the primary key of the **DEPARTMENTS** table.
- Do not use the **CYCLE** option.

```
CREATE SEQUENCE dept_deptid_seq
                INCREMENT BY 10
                START WITH 120
                MAXVALUE 9999
                NOCACHE
                NOCYCLE;
Sequence created.
```

Creating a Sequence (continued)

The example on the slide creates a sequence named `DEPT_DEPTID_SEQ` to be used for the `DEPARTMENT_ID` column of the `DEPARTMENTS` table. The sequence starts at 120, does not allow caching, and does not cycle.

Do not use the `CYCLE` option if the sequence is used to generate primary key values, unless you have a reliable mechanism that purges old rows faster than the sequence cycles.

For more information, see *Oracle9i SQL Reference*, “`CREATE SEQUENCE`.”

Note: The sequence is not tied to a table. Generally, you should name the sequence after its intended use; however the sequence can be used anywhere, regardless of its name.

Instructor Note

If the `INCREMENT BY` value is negative, the sequence descends. Also, `ORDER` | `NOORDER` options are available. The `ORDER` option guarantees that sequence values are generated in order. It is not important if you use the sequence to generate primary key values. This option is relevant only with the Parallel Server option.
If sequence values are cached, they will be lost if there is a system failure.

Confirming Sequences

- **Verify your sequence values in the `USER_SEQUENCES` data dictionary table.**

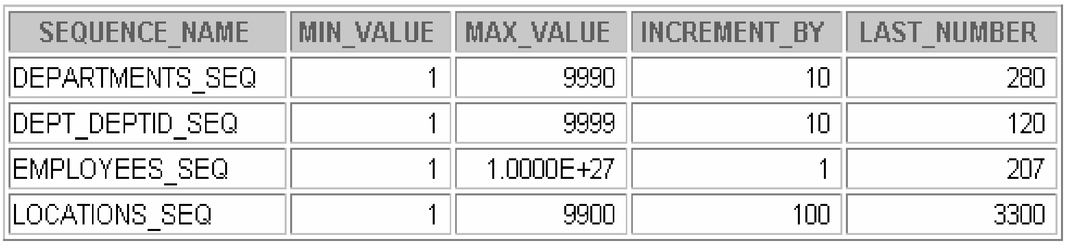
```
SELECT   sequence_name, min_value, max_value,
         increment_by, last_number
FROM     user_sequences;
```

- **The `LAST_NUMBER` column displays the next available sequence number if `NOCACHE` is specified.**

Confirming Sequences

Once you have created your sequence, it is documented in the data dictionary. Since a sequence is a database object, you can identify it in the `USER_OBJECTS` data dictionary table.

You can also confirm the settings of the sequence by selecting from the `USER_SEQUENCES` data dictionary view.



SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY	LAST_NUMBER
DEPARTMENTS_SEQ	1	9990	10	280
DEPT_DEPTID_SEQ	1	9999	10	120
EMPLOYEES_SEQ	1	1.0000E+27	1	207
LOCATIONS_SEQ	1	9900	100	3300

Instructor Note

Demo: `12_dd.sql`

Purpose: To illustrate the `USER_SEQUENCES` data dictionary view and its contents.

NEXTVAL and CURRVAL Pseudocolumns

- **NEXTVAL returns the next available sequence value. It returns a unique value every time it is referenced, even for different users.**
- **CURRVAL obtains the current sequence value.**
- **NEXTVAL must be issued for that sequence before CURRVAL contains a value.**

Using a Sequence

After you create your sequence, it generates sequential numbers for use in your tables. Reference the sequence values by using the NEXTVAL and CURRVAL pseudocolumns.

NEXTVAL and CURRVAL Pseudocolumns

The NEXTVAL pseudocolumn is used to extract successive sequence numbers from a specified sequence. You must qualify NEXTVAL with the sequence name. When you reference *sequence*.NEXTVAL, a new sequence number is generated and the current sequence number is placed in CURRVAL.

The CURRVAL pseudocolumn is used to refer to a sequence number that the current user has just generated. NEXTVAL must be used to generate a sequence number in the current user's session before CURRVAL can be referenced. You must qualify CURRVAL with the sequence name. When *sequence*.CURRVAL is referenced, the last value returned to that user's process is displayed.

You can use `NEXTVAL` and `CURRVAL` in the following contexts:

- The `SELECT` list of a `SELECT` statement that is not part of a subquery
- The `SELECT` list of a subquery in an `INSERT` statement
- The `VALUES` clause of an `INSERT` statement
- The `SET` clause of an `UPDATE` statement

You cannot use `NEXTVAL` and `CURRVAL` in the following contexts:

- The `SELECT` list of a view
- A `SELECT` statement with the `DISTINCT` keyword
- A `SELECT` statement with `GROUP BY`, `HAVING`, or `ORDER BY` clauses
- A subquery in a `SELECT`, `DELETE`, or `UPDATE` statement
- The `DEFAULT` expression in a `CREATE TABLE` or `ALTER TABLE` statement

For more information, see *Oracle9i SQL Reference*, “Pseudocolumns” section and “`CREATE SEQUENCE`.”

Instructor Note

Be sure to point out the rules listed on this page.

Using a Sequence

- **Insert a new department named “Support” in location ID 2500.**

```
INSERT INTO departments(department_id,
            department_name, location_id)
VALUES      (dept_deptid_seq.NEXTVAL,
            'Support', 2500);
1 row created.
```

- **View the current value for the `DEPT_DEPTID_SEQ` sequence.**

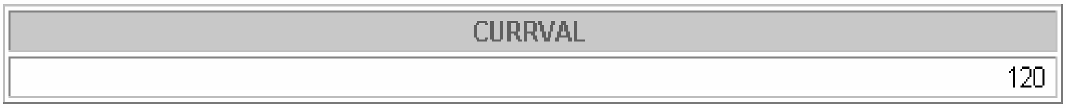
```
SELECT   dept_deptid_seq.CURRVAL
FROM     dual;
```

Using a Sequence

The example on the slide inserts a new department in the `DEPARTMENTS` table. It uses the `DEPT_DEPTID_SEQ` sequence for generating a new department number as follows:

You can view the current value of the sequence:

```
SELECT dept_deptid_seq.CURRVAL
FROM   dual;
```



```
                         CURRVAL
                                                    120
```

Suppose now you want to hire employees to staff the new department. The `INSERT` statement to be executed for all new employees can include the following code:

```
INSERT INTO employees (employee_id, department_id, ...)
VALUES (employees_seq.NEXTVAL, dept_deptid_seq .CURRVAL, ...);
```

Note: The preceding example assumes that a sequence called `EMPLOYEE_SEQ` has already been created for generating new employee numbers.

Using a Sequence

- **Caching sequence values in memory gives faster access to those values.**
- **Gaps in sequence values can occur when:**
  - **A rollback occurs**
  - **The system crashes**
  - **A sequence is used in another table**
- **If the sequence was created with `NOCACHE`, view the next available value, by querying the `USER_SEQUENCES` table.**

Caching Sequence Values

Cache sequences in memory to provide faster access to those sequence values. The cache is populated the first time you refer to the sequence. Each request for the next sequence value is retrieved from the cached sequence. After the last sequence value is used, the next request for the sequence pulls another cache of sequences into memory.

Gaps in the Sequence

Although sequence generators issue sequential numbers without gaps, this action occurs independent of a commit or rollback. Therefore, if you roll back a statement containing a sequence, the number is lost.

Another event that can cause gaps in the sequence is a system crash. If the sequence caches values in the memory, then those values are lost if the system crashes.

Because sequences are not tied directly to tables, the same sequence can be used for multiple tables. If you do so, each table can contain gaps in the sequential numbers.

Viewing the Next Available Sequence Value without Incrementing It

If the sequence was created with `NOCACHE`, it is possible to view the next available sequence value without incrementing it by querying the `USER_SEQUENCES` table.

Instructor Note

Frequently used sequences should be created with caching to improve efficiency. For cached sequences, there is no way to find out what the next available sequence value will be without actually obtaining, and using up, that value. It is recommended that users resist finding the next sequence value. Trust the system to provide a unique value each time a sequence is used in an `INSERT` statement.

Modifying a Sequence

Change the increment value, maximum value, minimum value, cycle option, or cache option.

```
ALTER SEQUENCE dept_deptid_seq
               INCREMENT BY 20
               MAXVALUE 999999
               NOCACHE
               NOCYCLE;
Sequence altered.
```

Altering a Sequence

If you reach the `MAXVALUE` limit for your sequence, no additional values from the sequence are allocated and you will receive an error indicating that the sequence exceeds the `MAXVALUE`. To continue to use the sequence, you can modify it by using the `ALTER SEQUENCE` statement.

Syntax

```
ALTER  SEQUENCE   sequence
       [INCREMENT BY n]
       [{MAXVALUE n | NOMAXVALUE}]
       [{MINVALUE n | NOMINVALUE}]
       [{CYCLE | NOCYCLE}]
       [{CACHE n | NOCACHE}];
```

In the syntax:

`sequence` is the name of the sequence generator

For more information, see *Oracle9i SQL Reference*, “`ALTER SEQUENCE`.”

Guidelines for Modifying a Sequence

- You must be the owner or have the `ALTER` privilege for the sequence.
- Only future sequence numbers are affected.
- The sequence must be dropped and re-created to restart the sequence at a different number.
- Some validation is performed.

Guidelines for Modifying Sequences

- You must be the owner or have the `ALTER` privilege for the sequence in order to modify it.
- Only future sequence numbers are affected by the `ALTER SEQUENCE` statement.
- The `START WITH` option cannot be changed using `ALTER SEQUENCE`. The sequence must be dropped and re-created in order to restart the sequence at a different number.
- Some validation is performed. For example, a new `MAXVALUE` that is less than the current sequence number cannot be imposed.

```
ALTER SEQUENCE dept_deptid_seq
      INCREMENT BY 20
      MAXVALUE 90
      NOCACHE
      NOCYCLE;
ALTER SEQUENCE dept_deptid_seq
*
ERROR at line 1:
ORA-04009: MAXVALUE cannot be made to be less than the
current
                value
```

Removing a Sequence

- Remove a sequence from the data dictionary by using the `DROP SEQUENCE` statement.
- Once removed, the sequence can no longer be referenced.

```
DROP SEQUENCE dept_deptid_seq;
Sequence dropped.
```

Removing a Sequence

To remove a sequence from the data dictionary, use the `DROP SEQUENCE` statement. You must be the owner of the sequence or have the `DROP ANY SEQUENCE` privilege to remove it.

Syntax

```
DROP    SEQUENCE     sequence;
```

In the syntax:

`sequence` is the name of the sequence generator

For more information, see *Oracle9i SQL Reference*, "`DROP SEQUENCE`."

What is an Index?

An index:

- **Is a schema object**
- **Is used by the Oracle server to speed up the retrieval of rows by using a pointer**
- **Can reduce disk I/O by using a rapid path access method to locate data quickly**
- **Is independent of the table it indexes**
- **Is used and maintained automatically by the Oracle server**

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, 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 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 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.

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

For more information, see *Oracle9i Concepts*, “Schema Objects” section, “Indexes” topic.

Instructor Note

The decision to create indexes is a global, high-level decision. Creation and maintenance of indexes is often a task for the database administrator.

Reference the column that has an index in the predicate `WHERE` clause without modifying the indexed column with a function or expression.
A `ROWID` is a hexadecimal string representation of the row address containing block identifier, row location in the block, and the database file identifier. The fastest way to access any particular row is by referencing its `ROWID`.

How Are Indexes Created?

- **Automatically: A unique index is created automatically when you define a `PRIMARY KEY` or `UNIQUE` constraint in a table definition.**
- **Manually: Users can create nonunique indexes on columns to speed up access to the rows.**

Types of Indexes

Two types of indexes can be created. One type is a 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.

The other type of index is a nonunique index, which a user can create. For example, you can create a `FOREIGN KEY` column index for a join in a query to improve retrieval speed.

Note: You can manually create a unique index, but it is recommended that you create a unique constraint, which implicitly creates a unique index.

Creating an Index

- **Create an index on one or more columns.**

```
CREATE INDEX index
ON table (column[, column]...);
```

- **Improve the speed of query access to the `LAST_NAME` column in the `EMPLOYEES` table.**

```
CREATE INDEX emp_last_name_idx
ON           employees(last_name);
Index created.
```

Creating an Index

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

In the syntax:

`index`	is the name of the index
`table`	is the name of the table
`column`	is the name of the column in the table to be indexed

For more information, see *Oracle9i SQL Reference*, "`CREATE INDEX`."

Instructor Note

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.

Another option in the syntax is the `UNIQUE` keyword. Emphasize that you should not explicitly define unique indexes on tables. Instead define uniqueness in the table as a constraint. The Oracle server enforces unique integrity constraints by automatically defining a unique index on the unique key.

When to Create an Index

You should create an index if:

- **A 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 to 4 percent of the rows**

More Is Not Always Better

More indexes on a table does not mean faster queries. Each DML operation that is committed on a table with indexes means that the indexes must be updated. The more indexes you have associated with a table, the more effort the Oracle server must make to update all the indexes after a DML operation.

When to Create an Index

Therefore, you should create indexes only if:

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

Remember that if you want to enforce uniqueness, you should define a unique constraint in the table definition. Then a unique index is created automatically.

Instructor Note

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.

When Not to Create an Index

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 to 4 percent of the rows in the table**
- **The table is updated frequently**
- **The indexed columns are referenced as part of an expression**

ORACLE

Instructor Note

Null values are not included in the 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

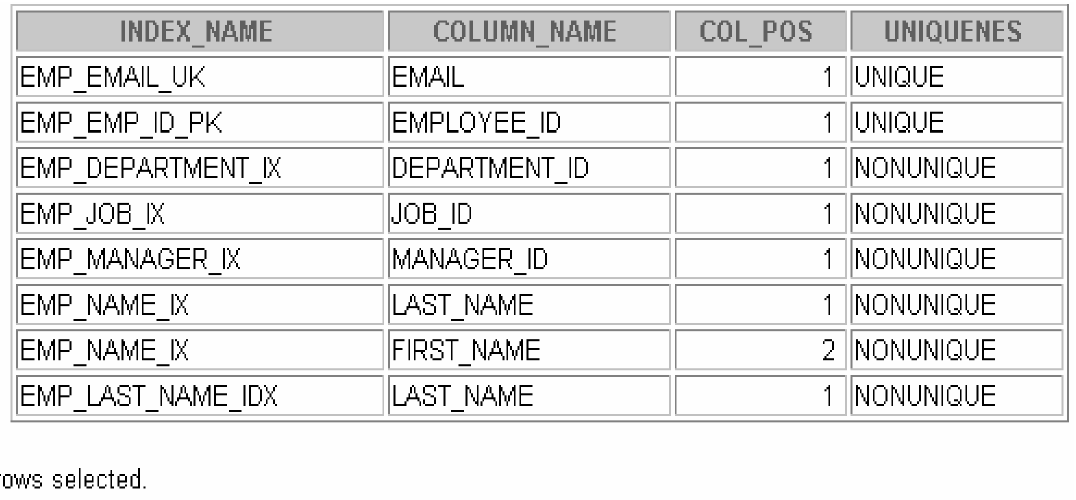
- **The `USER_INDEXES` data dictionary view contains the name of the index and its uniqueness.**
- **The `USER_IND_COLUMNS` view contains the index name, the table name, and the column name.**

```
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';
```

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 example on the slide displays all the previously created indexes, with the names of the affected column, and the index's uniqueness, on the `EMPLOYEES` table.



INDEX_NAME	COLUMN_NAME	COL_POS	UNIQUENES
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
EMP_LAST_NAME_IDX	LAST_NAME	1	NONUNIQUE

```
8 rows selected.
```

Function-Based Indexes

- **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.**

```
CREATE INDEX upper_dept_name_idx
ON departments(UPPER(department_name));

Index created.

SELECT *
FROM   departments
WHERE  UPPER(department_name) = 'SALES';
```

Function-Based Index

Function-based indexes defined with the `UPPER(`*`column_name`*`)` or `LOWER(`*`column_name`*`)` keywords allow case-insensitive searches. For example, the following index:

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

Facilitates processing queries such as:

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

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     employees
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.

Instructor Note

Let students know that to create a function-based index in your own schema on your own table, you must have the `CREATE INDEX` and `QUERY REWRITE` system privileges. To create the index in another schema or on another schema's table, you must have the `CREATE ANY INDEX` and `GLOBAL` `QUERY REWRITE` privileges. The table owner must also have the `EXECUTE` object privilege on the functions used in the function-based index.

Removing an Index

- Remove an index from the data dictionary by using the `DROP INDEX` command.

```
DROP INDEX index;
```

- Remove the `UPPER_LAST_NAME_IDX` index from the data dictionary.

```
DROP INDEX upper_last_name_idx;
Index dropped.
```

- To drop an index, you must be the owner of the index or have the `DROP ANY INDEX` privilege.

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.

In the syntax:

index	is the name of the index

Note: If you drop a table, indexes and constraints are automatically dropped, but views and sequences remain.

Synonyms

Simplify access to objects by creating a synonym (another name for an object). With synonyms, you can:

- **Ease referring to a table owned by another user**
- **Shorten lengthy object names**

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

Creating a Synonym for an Object

To refer to a table owned by another user, you need to prefix the table name with the name of the user who created it followed by a period. 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 objects. This method can be especially useful with lengthy object names, such as views.

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

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.

For more information, see *Oracle9i SQL Reference*, "`CREATE SYNONYM`."

Creating and Removing Synonyms

- **Create a shortened name for the `DEPT_SUM_VU` view.**

```
CREATE SYNONYM  d_sum
FOR  dept_sum_vu;
Synonym Created.
```

- **Drop a synonym.**

```
DROP SYNONYM d_sum;
Synonym dropped.
```

Creating a Synonym for an Object (continued)

The slide example creates a synonym for the `DEPT_SUM_VU` view for quicker reference.

The database administrator can create a public synonym accessible to all users. The following example creates a public synonym named `DEPT` for Alice's `DEPARTMENTS` table:

```
CREATE PUBLIC SYNONYM  dept
FOR    alice.departments;
Synonym created.
```

Removing a Synonym

To drop a synonym, use the `DROP SYNONYM` statement. Only the database administrator can drop a public synonym.

```
DROP PUBLIC SYNONYM  dept;
Synonym dropped.
```

For more information, see *Oracle9i SQL Reference*, "`DROP SYNONYM`."

Instructor Note

In the Oracle server, the DBA can specifically grant the `CREATE PUBLIC SYNONYM` privilege to any user, and that user can create public synonyms.

Summary

In this lesson, you should have learned how to:

- Automatically generate sequence numbers by using a sequence generator
- View sequence information in the `USER_SEQUENCES` data dictionary table
- Create indexes to improve query retrieval speed
- View index information in the `USER_INDEXES` dictionary table
- Use synonyms to provide alternative names for objects

ORACLE

Summary

In this lesson you should have learned about some of the other database objects including sequences, indexes, and views.

Sequences

The sequence generator can be used to automatically generate sequence numbers for rows in tables. This can save time and can reduce the amount of application code needed.

A sequence is a database object that can be shared with other users. Information about the sequence can be found in the `USER_SEQUENCES` table of the data dictionary.

To use a sequence, reference it with either the `NEXTVAL` or the `CURRVAL` pseudocolumns.

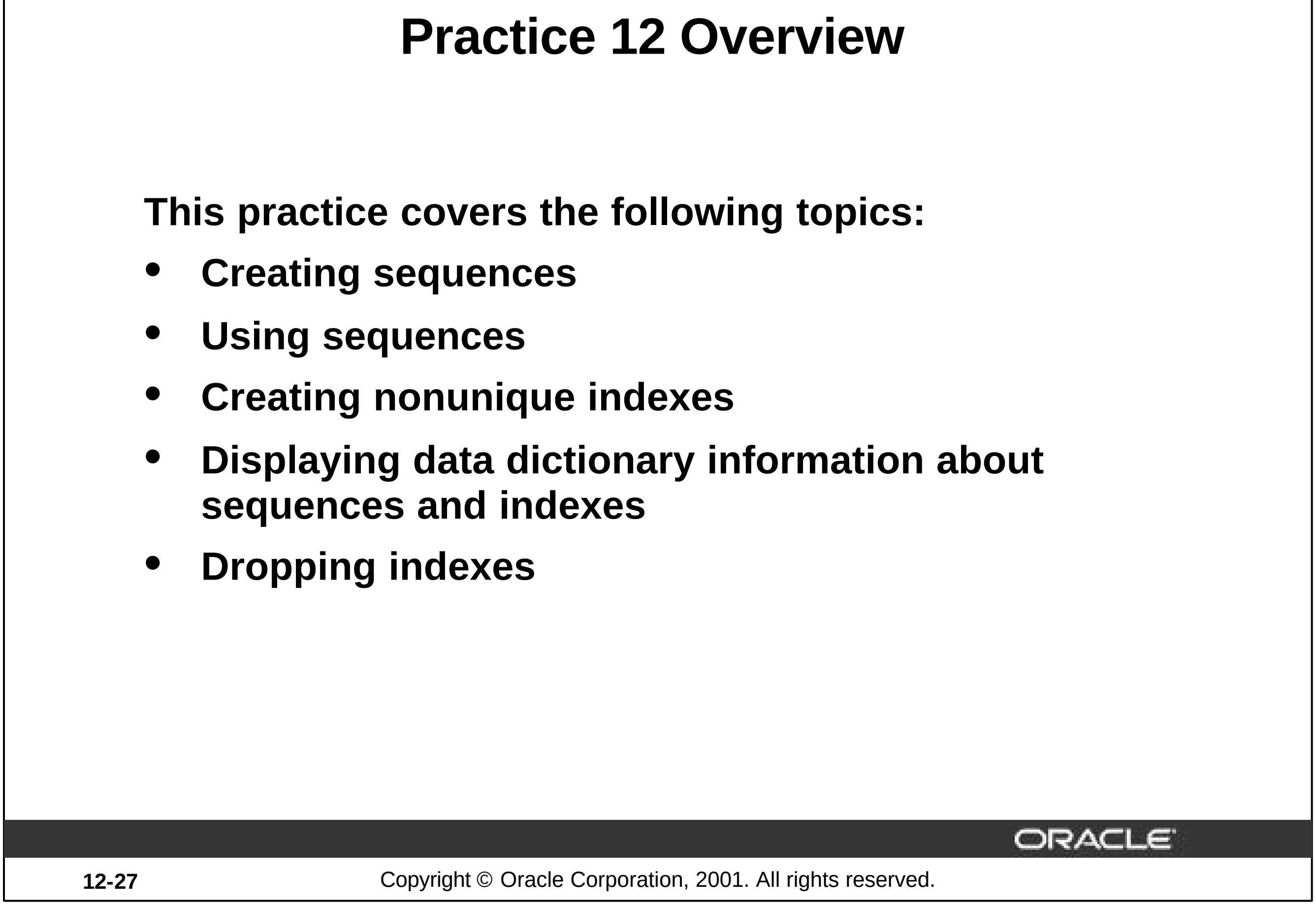
- Retrieve the next number in the sequence by referencing *`sequence`*`.NEXTVAL`.
- Return the current available number by referencing *`sequence`*`.CURRVAL`.

Indexes

Indexes are used to improve query retrieval speed. Users can view the definitions of the indexes in the `USER_INDEXES` data dictionary view. An index can be dropped by the creator, or a user with the `DROP ANY INDEX` privilege, by using the `DROP INDEX` statement.

Synonyms

Database administrators can create public synonyms and users can create private synonyms for convenience, by using the `CREATE SYNONYM` statement. Synonyms permit short names or alternative names for objects. Remove synonyms by using the `DROP SYNONYM` statement.



Practice 12 Overview

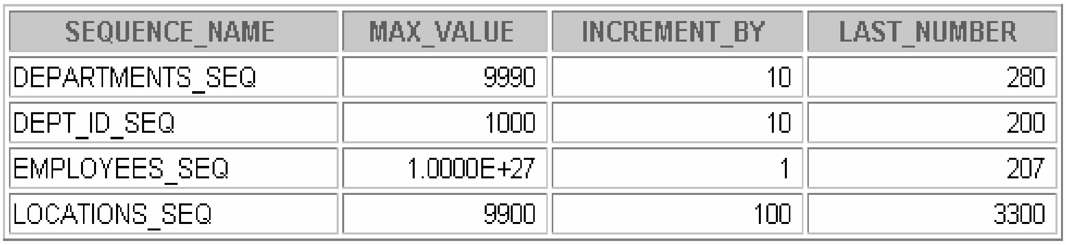
This practice covers the following topics:

- Creating sequences
- Using sequences
- Creating nonunique indexes
- Displaying data dictionary information about sequences and indexes
- Dropping indexes

Practice 12 Overview

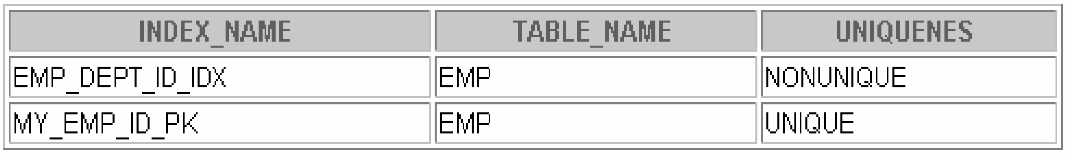
In this practice, you create a sequence to be used when populating your table. You also create implicit and explicit indexes.

1. Create a sequence to be used with the primary key column of the DEPT table. The sequence should start at 200 and have a maximum value of 1000. Have your sequence increment
   by ten numbers. Name the sequence DEPT_ID_SEQ.
2. Write a query in a script to display the following information about your sequences: sequence name, maximum value, increment size, and last number. Name the script lab12_2.sql. Run the statement in your script.



SEQUENCE_NAME	MAX_VALUE	INCREMENT_BY	LAST_NUMBER
DEPARTMENTS_SEQ	9990	10	280
DEPT_ID_SEQ	1000	10	200
EMPLOYEES_SEQ	1.0000E+27	1	207
LOCATIONS_SEQ	9900	100	3300

3. Write a script to insert two rows into the DEPT table. Name your script lab12_3.sql. Be sure to use the sequence that you created for the ID column. Add two departments named Education and Administration. Confirm your additions. Run the commands in your script.
4. Create a nonunique index on the foreign key column (DEPT_ID) in the EMP table.
5. Display the indexes and uniqueness that exist in the data dictionary for the EMP table. Save the statement into a script named lab12_5.sql.



INDEX_NAME	TABLE_NAME	UNIQUENES
EMP_DEPT_ID_IDX	EMP	NONUNIQUE
MY_EMP_ID_PK	EMP	UNIQUE