

10

Creating Other Schema Objects

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

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

- Create simple and complex views
- Retrieve data from views
- Create, maintain, and use sequences
- Create and maintain indexes
- Create private and public synonyms

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

In this lesson, you are introduced to the view, sequence, synonym, and index objects. You are taught the basics of creating and using views, sequences, and indexes.

Database Objects

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Database Objects

There are several other objects in a database in addition to tables. In this lesson, you learn about views, sequences, indexes, and synonyms.

With views, you can present and hide data from tables.

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 View?

EMPLOYEES table

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000
4	103	Alexander	Hunold					9000
5	104	Bruce	Ernst					6000
6	107	Diana						4200
7	124	Kevin						5800
8	141	Trenna						3500
9	142	Curtis						3100
10	143	Randall						2600
11	144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	2500
12	149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-00	SA_MAN	10500
13	174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-96	SA_REP	11000
14	176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-98	SA_REP	8600
15	178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000
16	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400
17	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	13000
18	202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	6000
19	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000
20	206	William	Gietz	WGIEZT	515.123.8181	07-JUN-94	AC_ACCOUNT	8300

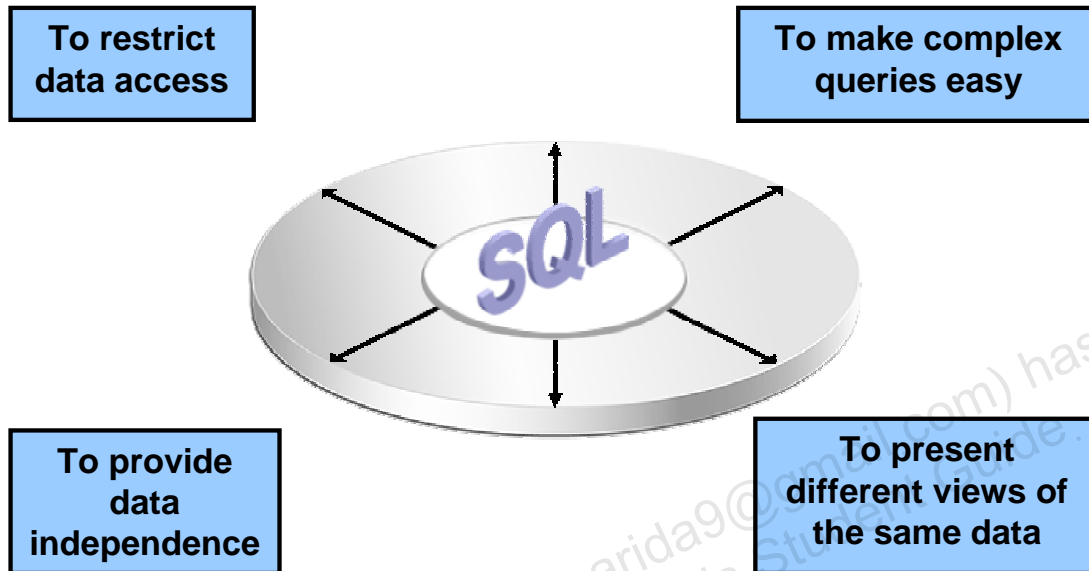
ORACLE

Copyright © 2009, Oracle. All rights reserved.

What Is a View?

You can present logical subsets or combinations of data by creating views of tables. A view is a logical table based on a table or another view. A view contains no data of its own but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called *base tables*. The view is stored as a `SELECT` statement in the data dictionary.

Advantages of Views



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Advantages of Views

- Views restrict access to the data because the view can display selected columns from the table.
- Views can be used to make simple queries to retrieve the results of complicated queries. For example, views can be used to query information from multiple tables without the user knowing how to write a join statement.
- Views provide data independence for ad hoc users and application programs. One view can be used to retrieve data from several tables.
- Views provide groups of users access to data according to their particular criteria.

For more information, see “CREATE VIEW” in the *Oracle SQL Reference*.

Simple Views and Complex Views

Feature	Simple Views	Complex Views
Number of tables	One	One or more
Contain functions	No	Yes
Contain groups of data	No	Yes
DML operations through a view	Yes	Not always

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Simple Views and Complex Views

There are two classifications for views: simple and complex. The basic difference is related to the DML (INSERT, UPDATE, and DELETE) operations.

- A simple view is one that:
 - Derives data from only one table
 - Contains no functions or groups of data
 - Can perform DML operations through the view
- A complex view is one that:
 - Derives data from many tables
 - Contains functions or groups of data
 - Does not always allow DML operations through the view

Creating a View

- You embed a subquery in the CREATE VIEW statement:

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
  [(alias[, alias]...)]
  AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

- The subquery can contain complex SELECT syntax.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a View

You can create a view by embedding a subquery in the CREATE VIEW statement.

In the syntax:

OR REPLACE	Re-creates the view if it already exists
FORCE	Creates the view regardless of whether or not the base tables exist
NOFORCE	Creates the view only if the base tables exist (This is the default.)
<i>view</i>	Is the name of the view
<i>alias</i>	Specifies names for the expressions selected by the view's query (The number of aliases must match the number of expressions selected by the view.)
<i>subquery</i>	Is a complete SELECT statement (You can use aliases for the columns in the SELECT list.)
WITH CHECK OPTION	Specifies that only those rows that are accessible to the view can be inserted or updated
<i>constraint</i>	Is the name assigned to the CHECK OPTION constraint
WITH READ ONLY	ensures that no DML operations can be performed on this view

Creating a View

- Create the EMPVU80 view, which contains details of employees in department 80:

```
CREATE VIEW empvu80
AS SELECT employee_id, last_name, salary
FROM employees
WHERE department_id = 80;
CREATE VIEW succeeded.
```

- Describe the structure of the view by using the DESCRIBE command:

```
DESCRIBE empvu80
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a View (continued)

The example in the slide creates a view that contains the employee number, last name, and salary for each employee in department 80.

You can display the structure of the view by using the DESCRIBE command.

Name	Null	Type
-----	-----	-----
EMPLOYEE_ID	NOT NULL	NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
SALARY		NUMBER(8,2)
3 rows selected		

Guidelines for Creating a View

- The subquery that defines a view can contain complex SELECT syntax, including joins, groups, and subqueries.
- If you do not specify a constraint name for a view created with the WITH CHECK OPTION, the system assigns a default name in the format SYS_Cn.
- You can use the OR REPLACE option to change the definition of the view without dropping and re-creating it or regranteeing object privileges previously granted on it.

Creating a View

- Create a view by using column aliases in the subquery:

```
CREATE VIEW    salvu50
  AS SELECT    employee_id ID_NUMBER, last_name NAME,
              salary*12 ANN_SALARY
    FROM      employees
    WHERE     department_id = 50;
CREATE VIEW succeeded.
```

- Select the columns from this view by the given alias names:

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a View (continued)

You can control the column names by including column aliases in the subquery.

The example in the slide creates a view containing the employee number (EMPLOYEE_ID) with the alias ID_NUMBER, name (LAST_NAME) with the alias NAME, and annual salary (SALARY) with the alias ANN_SALARY for every employee in department 50.

As an alternative, you can use an alias after the CREATE statement and before the SELECT subquery. The number of aliases listed must match the number of expressions selected in the subquery.

```
CREATE OR REPLACE VIEW    salvu50 (ID_NUMBER, NAME, ANN_SALARY)
  AS SELECT    employee_id, last_name, salary*12
    FROM      employees
    WHERE     department_id = 50;
CREATE VIEW succeeded.
```

Retrieving Data from a View

```
SELECT *  
FROM salvu50;
```

	ID_NUMBER	NAME	ANN_SALARY
1	124	Mourgos	69600
2	141	Rajs	42000
3	142	Davies	37200
4	143	Matos	31200
5	144	Vargas	30000

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Retrieving Data from a View

You can retrieve data from a view as you would from any table. You can display either the contents of the entire view or just specific rows and columns.

Modifying a View

- Modify the EMPVU80 view by using a CREATE OR REPLACE VIEW clause. Add an alias for each column name:

```
CREATE OR REPLACE VIEW empvu80
(id_number, name, sal, department_id)
AS SELECT  employee_id, first_name || ' '
           || last_name, salary, department_id
FROM      employees
WHERE     department_id = 80;
CREATE VIEW succeeded.
```

- Column aliases in the CREATE OR REPLACE VIEW clause are listed in the same order as the columns in the subquery.

ORACLE



Copyright © 2009, Oracle. All rights reserved.

Modifying a View

With the OR REPLACE option, a view can be created even if one exists with this name already, thus replacing the old version of the view for its owner. This means that the view can be altered without dropping, re-creating, and regranteeing object privileges.

Note: When assigning column aliases in the CREATE OR REPLACE VIEW clause, remember that the aliases are listed in the same order as the columns in the subquery.

Rules for Performing DML Operations on a View

- You can usually perform DML operations on simple views. 
- You cannot remove a row if the view contains the following:
 - Group functions
 - A GROUP BY clause
 - The DISTINCT keyword
 - The pseudocolumn ROWNUM keyword

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Performing DML Operations on a View

You can perform DML operations on data through a view if those operations follow certain rules.

You can remove a row from a view unless it contains any of the following:

- Group functions
- A GROUP BY clause
- The DISTINCT keyword
- The pseudocolumn ROWNUM keyword

Rules for Performing DML Operations on a View

You cannot modify data in a view if it contains:

- Group functions
- A `GROUP BY` clause
- The `DISTINCT` keyword
- The pseudocolumn `ROWNUM` keyword
- Columns defined by expressions

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, is positioned on the right side of a red horizontal bar.

Copyright © 2009, Oracle. All rights reserved.

Performing DML Operations on a View (continued)

You can modify data through a view unless it contains any of the conditions mentioned in the previous slide or columns defined by expressions (for example, `SALARY * 1.2`).

Rules for Performing DML Operations on a View

You cannot add data through a view if the view includes:

- Group functions
- A `GROUP BY` clause
- The `DISTINCT` keyword
- The pseudocolumn `ROWNUM` keyword
- Columns defined by expressions
- `NOT NULL` columns in the base tables that are not selected by the view

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Performing DML Operations on a View (continued)

You can add data through a view unless it contains any of the items listed in the slide. You cannot add data to a view if the view contains `NOT NULL` columns without default values in the base table. All required values must be present in the view. Remember that you are adding values directly to the underlying table *through* the view.

For more information, see “`CREATE VIEW`” in the *Oracle SQL Reference*.

Using the WITH CHECK OPTION Clause

- You can ensure that DML operations performed on the view stay in the domain of the view by using the WITH CHECK OPTION clause:

```
CREATE OR REPLACE VIEW empvu20
AS SELECT      *
   FROM        employees
   WHERE       department_id = 20
   WITH CHECK OPTION CONSTRAINT empvu20_ck ;
CREATE VIEW succeeded.
```

- Any attempt to change the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using the WITH CHECK OPTION Clause

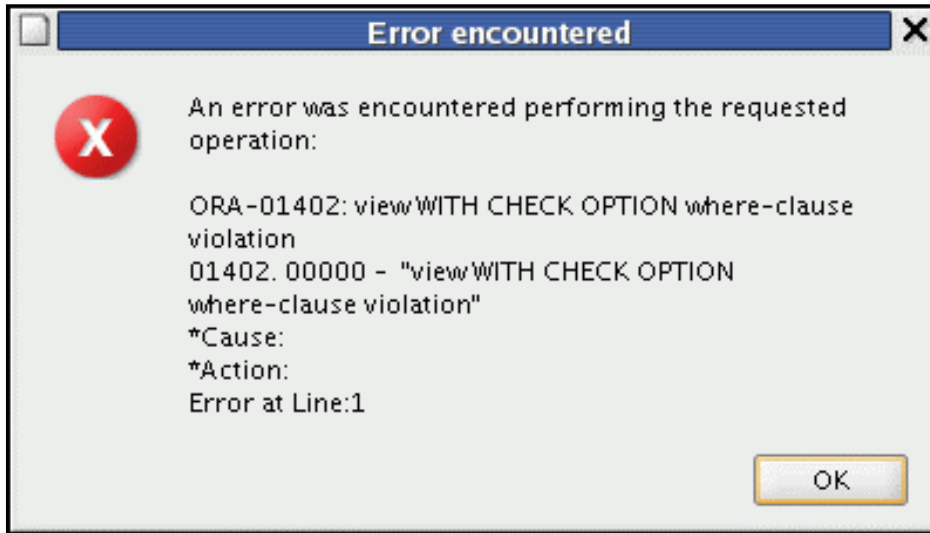
It is possible to perform referential integrity checks through views. You can also enforce constraints at the database level. The view can be used to protect data integrity, but the use is very limited.

The WITH CHECK OPTION clause specifies that INSERTs and UPDATEs performed through the view cannot create rows that the view cannot select, and therefore it enables integrity constraints and data validation checks to be enforced on data being inserted or updated.

Note: The WITH CHECK OPTION clause relates to inserts and updates on the view only. It has no effect on deletes.

If there is an attempt to perform DML operations on rows that the view has not selected, an error is displayed, along with the constraint name if that has been specified.

```
UPDATE empvu20
SET    department_id = 10
WHERE  employee_id = 201;
```


Using the WITH CHECK OPTION Clause (continued)

Note: No rows are updated because if the department number were to change to 10, the view would no longer be able to see that employee. With the WITH CHECK OPTION clause, therefore, the view can see only employees in department 20 and does not allow the department number for those employees to be changed through the view.

Denying DML Operations

- You can ensure that no DML operations occur by adding the `WITH READ ONLY` option to your view definition.
- Any attempt to perform a DML operation on any row in the view results in an Oracle server error.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Denying DML Operations

You can ensure that no DML operations occur on your view by creating it with the `WITH READ ONLY` option. The example in the next slide modifies the `EMPVU10` view to prevent any DML operations on the view.

Denying DML Operations

```
CREATE OR REPLACE VIEW empvu10
  (employee_number, employee_name, job_title)
AS SELECT      employee_id, last_name, job_id
  FROM        employees
  WHERE       department_id = 10
  WITH READ ONLY ;
CREATE VIEW succeeded.
```

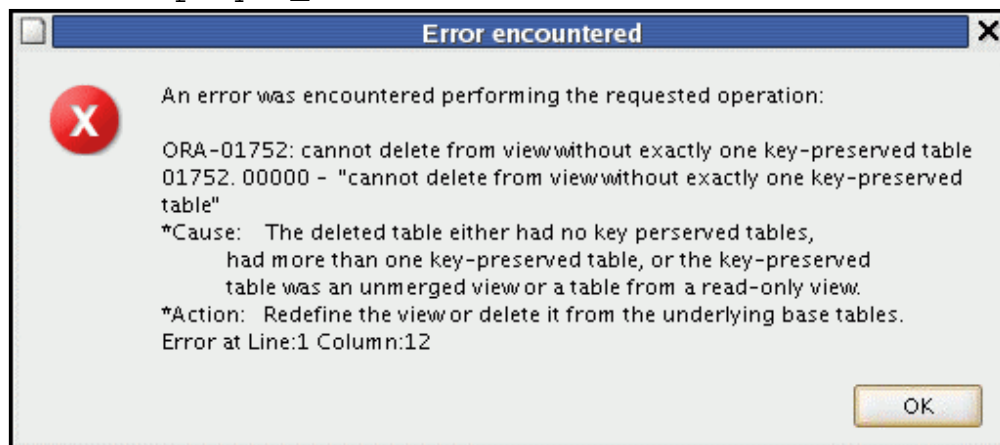
ORACLE

Copyright © 2009, Oracle. All rights reserved.

Denying DML Operations (continued)

Any attempt to remove a row from a view with a read-only constraint results in an error:

```
DELETE FROM empvu10
WHERE employee_number = 200;
```



Any attempt to insert a row or modify a row using the view with a read-only constraint results in an Oracle server error:

```
01733: virtual column not allowed here.
```

Removing a View

You can remove a view without losing data because a view is based on underlying tables in the database.

```
DROP VIEW view;
```

```
DROP VIEW empvu80;  
DROP VIEW empvu80 succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Removing a View

You use the DROP VIEW statement to remove a view. The statement removes the view definition from the database. Dropping views has no effect on the tables on which the view was based. Views or other applications based on deleted views become invalid. Only the creator or a user with the DROP ANY VIEW privilege can remove a view.

In the syntax:

view is the name of the view

Practice 10: Overview of Part 1

This practice covers the following topics:

- Creating a simple view
- Creating a complex view
- Creating a view with a check constraint
- Attempting to modify data in the view
- Removing views

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, is positioned on the right side of a red horizontal bar.

Copyright © 2009, Oracle. All rights reserved.

Practice 10: Overview of Part 1

Part 1 of this lesson's practice provides you with a variety of exercises in creating, using, and removing views.

Complete questions 1–6 at the end of this lesson.

Sequences

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

ORACLE

Copyright © 2009, Oracle. All rights reserved.

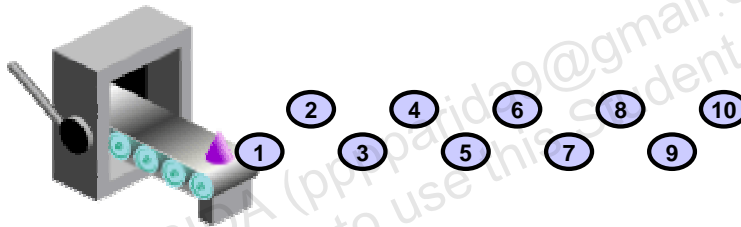
Sequences

A sequence is a database object that creates integer values. You can create sequences and then use them to generate numbers.

Sequences

A sequence:

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

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

Sequences (continued)

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

You can define a sequence to generate unique values or to recycle and use the same numbers again.

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.

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}];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a Sequence

Automatically generate sequential numbers by using the CREATE SEQUENCE statement.

In the syntax:

<i>sequence</i>	Is the name of the sequence generator
INCREMENT BY <i>n</i>	Specifies the interval between sequence numbers, where <i>n</i> is an integer (If this clause is omitted, the sequence increments by 1.)
START WITH <i>n</i>	Specifies the first sequence number to be generated (If this clause is omitted, the sequence starts with 1.)
MAXVALUE <i>n</i>	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 <i>n</i>	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.)

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;
CREATE SEQUENCE succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a Sequence (continued)

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 keeps in memory (By default, the Oracle server caches 20 values.)

The example in 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 “CREATE SEQUENCE” in the *Oracle SQL Reference*.

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.

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.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

NEXTVAL and CURRVAL Pseudocolumns

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.

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 you reference *sequence*.CURRVAL, the last value returned to that user's process is displayed.

NEXTVAL and CURRVAL Pseudocolumns (continued)**Rules for Using NEXTVAL and CURRVAL**

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 “Pseudocolumns” and “CREATE SEQUENCE” in *Oracle SQL Reference*.

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;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using a Sequence

The example in the slide inserts a new department in the DEPARTMENTS table. It uses the DEPT_DEPTID_SEQ sequence to generate a new department number as follows.

You can view the current value of the sequence:

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

	CURRVAL
1	120

Suppose that you now 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 to generate new employee numbers.

Caching Sequence Values

- 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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Caching Sequence Values

You can 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 memory, 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.

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;
ALTER SEQUENCE dept_deptid_seq succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Modifying 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 “ALTER SEQUENCE” in the *Oracle SQL Reference*.

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.
- To remove a sequence, use the `DROP` statement:

```
DROP SEQUENCE dept_deptid_seq;
DROP SEQUENCE dept_deptid_seq succeeded.
```

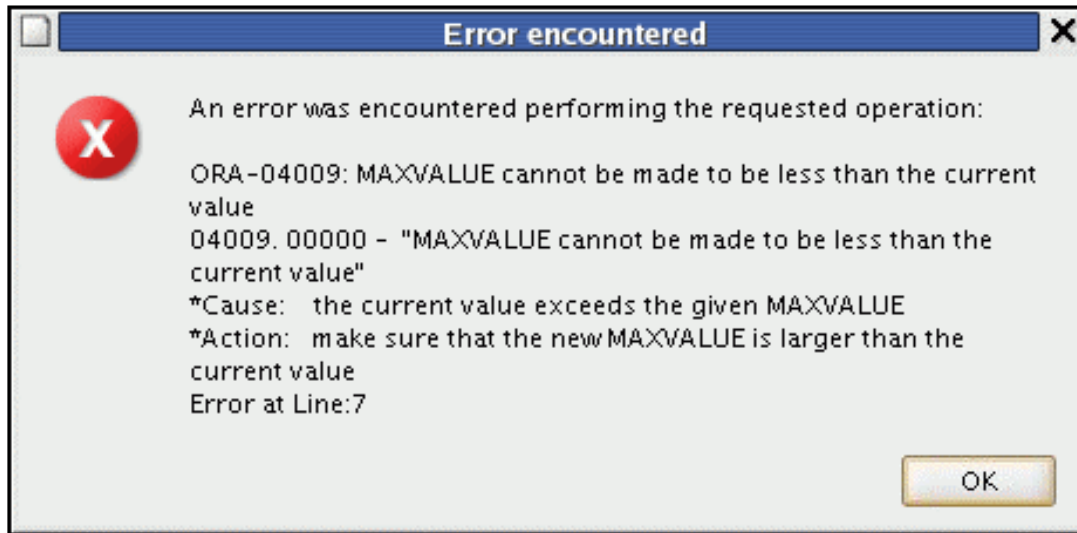
ORACLE

Copyright © 2009, Oracle. All rights reserved.

Guidelines for Modifying a Sequence

- You must be the owner or have the `ALTER` privilege for the sequence to modify it. You must be the owner or have the `DROP ANY SEQUENCE` privilege to remove 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 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;
```

Guidelines for Modifying a Sequence (continued)

Indexes

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

ORACLE

Copyright © 2009, Oracle. All rights reserved.

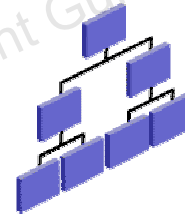
Indexes

Indexes are database objects that you can create to improve the performance of some queries. Indexes can also be created automatically by the server when you create a primary key or unique constraint.

Indexes

An index:

- Is a schema object
- Can be 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 that it indexes
- Is used and maintained automatically by the Oracle server



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Indexes (continued)

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. After an index is created, no direct activity is required by the user.

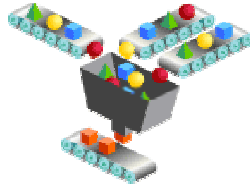
Indexes are logically and physically independent of the table that they index. This means that they can be created or dropped at any time and have no effect on the base tables or other indexes.

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

For more information, see “Schema Objects: Indexes” in *Database Concepts*.

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.

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

Types of Indexes

Two types of indexes can be created.

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 that is given to the constraint.

Nonunique index: This is an index that 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);
CREATE INDEX succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating an Index

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

In the syntax:

<i>index</i>	Is the name of the index
<i>table</i>	Is the name of the table
<i>column</i>	Is the name of the column in the table to be indexed

For more information, see “CREATE INDEX” in *Oracle SQL Reference*.

Index Creation Guidelines

Create an index when:	
✓	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 <code>WHERE</code> clause or a join condition
✓	The table is large and most queries are expected to retrieve less than 2% to 4% of the rows in the table
Do not create an index when:	
✗	The columns are not often used as a condition in the query
✗	The table is small or most queries are expected to retrieve more than 2% to 4% of the rows in the table
✗	The table is updated frequently
✗	The indexed columns are referenced as part of an expression

ORACLE

Copyright © 2009, Oracle. All rights reserved.

More Is Not Always Better

Having more indexes on a table does not produce faster queries. Each DML operation that is committed on a table with indexes means that the indexes must be updated. The more indexes that 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% to 4% of the rows

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

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 emp_last_name_idx;  
DROP INDEX emp_last_name_idx succeeded.
```

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

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

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Synonyms

Synonyms are database objects that enable you to call a table by another name. You can create synonyms to give an alternative name to a table.

Synonyms

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

- Create an easier reference to a table that is owned by another user
- Shorten lengthy object names

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

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a Synonym for an Object

To refer to a table that is 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:

<code>PUBLIC</code>	Creates a synonym that is accessible to all users
<code><i>synonym</i></code>	Is the name of the synonym to be created
<code><i>object</i></code>	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 that are owned by the same user.

For more information, see “CREATE SYNONYM” in the *Oracle SQL Reference*.

Creating and Removing Synonyms

- Create a shortened name for the DEPT_SUM_VU view:

```
CREATE SYNONYM d_sum
FOR dept_sum_vu;
CREATE SYNONYM succeeded.
```

- Drop a synonym:

```
DROP SYNONYM d_sum;
DROP SYNONYM succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a Synonym

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

The database administrator can create a public synonym that is 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;
CREATE SYNONYM succeeded.
```

Removing a Synonym

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

```
DROP PUBLIC SYNONYM dept;
DROP SYNONYM succeeded.
```

For more information, see "DROP SYNONYM" in the *Oracle SQL Reference*.

Summary

In this lesson, you should have learned how to:

- Create, use, and remove views
- Automatically generate sequence numbers by using a sequence generator
- Create indexes to improve query retrieval speed
- Use synonyms to provide alternative names for objects

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, is positioned on the right side of a red horizontal bar.

Copyright © 2009, Oracle. All rights reserved.

Summary

In this lesson, you should have learned about database objects such as views, sequences, indexes, and synonyms.

Practice 10: Overview of Part 2

This practice covers the following topics:

- Creating sequences
- Using sequences
- Creating nonunique indexes
- Creating synonyms

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, is positioned on the right side of a red horizontal bar.

Copyright © 2009, Oracle. All rights reserved.

Practice 10: Overview of Part 2

Part 2 of this lesson's practice provides you with a variety of exercises in creating and using a sequence, an index, and a synonym.

Complete questions 7–10 at the end of this lesson.

Practice 10**Part 1**

1. The staff in the HR department want to hide some of the data in the EMPLOYEES table. They want a view called EMPLOYEES_VU based on the employee numbers, employee names, and department numbers from the EMPLOYEES table. They want the heading for the employee name to be EMPLOYEE.
2. Confirm that the view works. Display the contents of the EMPLOYEES_VU view.

	EMPLOYEE_ID	EMPLOYEE	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
4	205	Higgins	110
5	206	Gietz	110
6	100	King	90
7	101	Kochhar	90
8	102	De Haan	90
9	103	Hunold	30
10	104	Ernst	30
11	107	Lorentz	30
12	124	Mourgos	50
13	141	Rajs	50
14	142	Davies	50
15	143	Matos	50
16	144	Vargas	50
17	149	Zlotkey	80
18	174	Abel	80
19	176	Taylor	80
20	178	Grant	(null)

3. Using your EMPLOYEES_VU view, write a query for the HR department to display all employee names and department numbers.

	EMPLOYEE	DEPARTMENT_ID
1	Whalen	10
2	Hartstein	20

...

19	Taylor	80
20	Grant	(null)

Practice 10 (continued)

4. Department 50 needs access to its employee data. Create a view named DEPT50 that contains the employee numbers, employee last names, and department numbers for all employees in department 50. You are asked to label the view columns EMPNO, EMPLOYEE, and DEPTNO. For security purposes, do not allow an employee to be reassigned to another department through the view.
5. Display the structure and contents of the DEPT50 view.

Name	Null	Type
EMPNO	NOT NULL	NUMBER(6)
EMPLOYEE	NOT NULL	VARCHAR2(25)
DEPTNO		NUMBER(4)

	EMPNO	EMPLOYEE	DEPTNO
1	124	Mourgos	50
2	141	Rajs	50
3	142	Davies	50
4	143	Matos	50
5	144	Vargas	50

6. Test your view. Attempt to reassign Matos to department 80.

Practice 10 (continued)**Part 2**

7. You need a sequence that can be used with the primary key column of the DEPT table. The sequence should start at 200 and have a maximum value of 1,000. Have your sequence increment by 10. Name the sequence DEPT_ID_SEQ.
8. To test your sequence, write a script to insert two rows in the DEPT table. Name your script lab_10_08.sql. Be sure to use the sequence that you created for the ID column. Add two departments: Education and Administration. Confirm your additions. Run the commands in your script.
9. Create a nonunique index on the NAME column in the DEPT table.
10. Create a synonym for your EMPLOYEES table. Call it EMP.