TABLES FROM TABLES, VIEWS, SYNONYMS, AND SEQUENCES

Create Table from Existing Table

- CREATE TABLE AS statement to create a table from an existing table by copying the existing table's columns.
- It is important to note that when creating a table in this way, the new table will be populated with the records from the existing table (based on the SELECT Statement).

Create Table AS SELECT

- CREATE TABLE new_table AS (SELECT * FROM old_table);
- CREATE TABLE suppliers AS (SELECT * FROM companies WHERE company_id < 5000);
- CREATE TABLE new_table AS (SELECT column_1, column2, ... column_n FROM old_table);

VIEWS

- A database view is a *logical* or *virtual table* based on a query.
- It is useful to think of a *view* as a stored query.
- Views are created through use of a CREATE VIEW command that incorporates use of the SELECT statement.
- Views are queried just like tables.

Views

- They are NOT separate copies of the data; they reference the data in the underlying tables
- Database stores definition of view; the data is updated when the underlying tables are updated

VIEWS

```
CREATE VIEW employee_parking
  (parking_space, last_name,
     first_name, ssn) AS
SELECT emp_parking_space,
  emp_last_name, emp_first_name,
  emp_ssn
FROM employee
ORDER BY emp_parking_space;
```

View Created.

VIEWS

```
SELECT *
FROM employee_parking;

PARKING_SPACE LAST_NAME  FIRST_NAME   SSN
------------- ---------- -----------  ---------
            1 Bordoloi   Bijoy        999666666
            3 Joyner     Suzanne      999555555
           32 Zhu        Waiman       999444444
more rows are displayed…
```

- Notice that the only columns in the query are those defined as part of the view.

VIEWS

- Additionally, we have renamed the columns in the view so that they are slightly different than the column names in the underlying employee table.
- Further, the rows are sorted by *parking_space* column even though there is no ORDER BY in the SELECT command used to access the view.

CREATING A VIEW

- **CREATE VIEW Syntax**

  CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW <view name> [(column alias name….)] AS <query> [WITH [CHECK OPTION] [READ ONLY] [CONSTRAINT]];

- The OR REPLACE option is used to create a view that already exists. This option is useful for modifying an existing view without having to drop or grant the privileges that system users have acquired with respect to the view .
- If you attempt to create a view that already exists without using the OR REPLACE option, Oracle will return the ORA-00955: *name is already used by an existing object* error message and the CREATE VIEW command will fail.

CREATING A VIEW

- The FORCE option allows a view to be created even if a base table that the view references does not already exist.
- This option is used to create a view prior to the actual creation of the base tables and accompanying data. Before such a view can be queried, the base tables must be created and data must be loaded into the tables. This option can also be used if a system user does not currently have the privilege to create a view.
- The NOFORCE option is the opposite of FORCE and allows a system user to create a view if they have the required permissions to create a view, and if the tables from which the view is created already exist. This is the default option.

CREATING A VIEW

- The WITH READ ONLY option allows creation of a view that is read-only. You cannot use the DELETE, INSERT, or UPDATE commands to modify data for the view.
- The WITH CHECK OPTION clause allows rows that can be selected through the view to be updated. It also enables the specification of constraints on values.
- The CONSTRAINT clause is used in conjunction with the WITH CHECK OPTION clause to enable a database administrator to assign a unique name to the CHECK OPTION. If the DBA omits the CONSTRAINT clause, Oracle will automatically assign the constraint a system-generated name that will not be very meaningful.

Example

```
CREATE VIEW empview7 AS
SELECT emp_ssn, emp_first_name, emp_last_name
FROM employee
WHERE emp_dpt_number=7;
```

View created.

- A simple query of the *empview7* shows the following data.

```
SELECT *
FROM empview7;
EMP_SSN   EMP_FIRST_NAME          EMP_LAST_NAME
--------- ---------------------- ----------------------
999444444 Waiman                  Zhu
999111111 Douglas                 Bock
999333333 Dinesh                  Joshi
999888888 Sherri                  Prescott
```

Example

- It is also possible to create a view that has exactly the same structure as an existing database table.
- The view named *dept_view* shown next has exactly the same structure as *department* table.

```
CREATE VIEW dept_view AS
SELECT *
FROM department;
View created.
```

Example

- We can recreate the view by using the OR REPLACE clause to create a view that is *read-only* by specifying a WITH READ ONLY clause.
- The new version of *dept_view* will restrict data manipulation language operations on the view to the use of the SELECT command.

```
CREATE OR REPLACE VIEW dept_view AS
SELECT *
FROM department WITH READ ONLY CONSTRAINT
   vw_dept_view_read_only;
View created.
```

Example

```
CREATE OR REPLACE VIEW dept_salary
   (name, min_salary, max_salary, avg_salary) AS
SELECT d.dpt_name, MIN(e.emp_salary),
   MAX(e.emp_salary), AVG(e.emp_salary)
FROM employee e, department d
WHERE e.emp_dpt_number=d.dpt_no
GROUP BY d.dpt_name;
```

View created.

```
SELECT *
FROM dept_salary;
```

NAME	MIN_SALARY	MAX_SALARY	AVG_SALARY
Admin and Records	25000	43000	31000
Headquarters	55000	55000	55000
Production	25000	43000	34000

VIEW STABILITY

- A view does not actually store any data. The data needed to support queries of a view are retrieved from the underlying database tables and displayed to a result table whenever a view is queried. The result table is only stored temporarily.
- If a table that underlies a view is dropped, then the view is no longer valid. Attempting to query an invalid view will produce an ORA-04063: view "VIEW_NAME" has errors error message.

INSERTING , UPDATING, AND DELETING TABLE ROWS THROUGH VIEWS

- You can insert a row if the view in use is one that is updateable (not read-only).
- A view is updateable if the INSERT command does not violate any constraints on the underlying tables.
- This rule concerning constraint violations also applies to UPDATE and DELETE commands.

Example

```
CREATE OR REPLACE VIEW dept_view AS
SELECT dpt_no, dpt_name
FROM department;

INSERT INTO dept_view VALUES (18, 'Department 18');
INSERT INTO dept_view VALUES (19, 'Department 20');

SELECT *
FROM dept_view;
DPT_NO  DPT_NAME
------- ------------------
      7 Production
      3 Admin and Records
      1 Headquarters
     18 Department 18
     19 Department 20
```

Example

```
UPDATE dept_view SET dpt_name = 'Department 19‘
 WHERE dpt_no = 19;
1 row updated.

SELECT *
FROM department
WHERE dpt_no >= 5;
 DPT_NO   DPT_NAME       DPT_MGRSS DPT_MGR_S
----------     ------------------- ----------------- ----------------
         7       Production          999444444     22-MAY-98
        18       Department 18
        19       Department 19
more rows are displayed…
```

More Examples

```
DELETE dept_view
WHERE dpt_no = 18 OR dpt_no = 19;
2 rows deleted.

SELECT *
FROM department;
DPT_NO   DPT_NAME            DPT_MGRSS   DPT_MGR_S
-------- -------------------- ---------- ---------
       7 Production           999444444 22-MAY-98
       3 Admin and Records    999555555 01-JAN-01
       1 Headquarters         999666666 19-JUN-81
```

More Examples

```
CREATE VIEW management
    AS SELECT Emp_no, ename
        FROM employees
        WHERE emp_no IN
                (SELECT mgr
                FROM employees
                WHERE mgr IS NOT NULL);
```

- Using a view in a query causes the SELECT statement that defines the view to be executed and data values retrieved

  eg SELECT ename
  FROM management;

More Examples

- Views are often used to simplify writing of complex queries which involve multiple tables or group functions

```
Eg: CREATE VIEW deptsummary
AS SELECT d.deptno, dept_name, sum(sal) Salaries
                FROM departments d, employees e
                WHERE d.dept_no = e.dept_no(+)
                GROUP BY d.dept_no, dept_name;
SELECT * FROM deptsummary
      WHERE Salaries > 5000;
```

More Examples

- Views are also used to permit access to only certain rows and/or columns in a table to some user(s) and not permit access to the remaining columns and/or rows

Eg: CREATE VIEW emplist

```
    AS SELECT emp_no, ename, fname, job
    FROM employees
    WHERE LOWER(job)!='president'
    ORDER BY ename, fname;
SELECT * FROM emplist;
```

CREATING A VIEW WITH ERRORS

- If there are no syntax errors in a CREATE VIEW statement, Oracle will create a view even if the view-defining query refers to a non-existent table or an invalid column of an existing table.
- The view will also be created even if the system user does not have privileges to access the tables which a view references.
- The new view will be unusable and is categorized as “created with errors.”
- In order to create such a view, the system user must use the FORCE option of the CREATE VIEW command.

DROPPING VIEW

- A DBA or view owner can drop a view with the DROP VIEW command. The following command drops a view named *dept_view*.

```
DROP VIEW dept_view;
View dropped.
```

A Summary of VIEW Facts

- A view does not store data, but a view does display data through a SELECT query as if the data were stored in the view.
- A view definition as provided by the CREATE VIEW statement is stored in the database. Further, Oracle develops what is termed an "execution plan" that is used to "gather up" the data that needs to be displayed by a view. This execution plan is also stored in the database.
- A view can simplify data presentation as well as provide a kind of data security by limiting access to data based on a "need to know."

A Summary of VIEW Facts

- A view can display data from more than one table.
- Views can be used to update the underlying tables. Views can also be limited to read-only access.
- Views can change the appearance of data. For example, a view can be used to rename columns from tables without affecting the base table.
- A view that has columns from more than one table cannot be modified by an INSERT, DELETE, or UPDATE command if a grouping function, GROUP BY clause is part of the view definition.

A Summary of VIEW Facts

- A view cannot reference the *nextval* and *currval* pseudocolumns created through the use of sequences.
- A row cannot be inserted in a view in which the base table has a column with the NOT NULL or other constraint that cannot be satisfied by the new row data.

SYNONYMS

- A *synonym* is an *alias*, that is, a form of shorthand used to simplify the task of referencing a database object.

- **Creating Synonyms**
- The general form of the CREATE SYNONYM command is:

```
CREATE [PUBLIC] SYNONYM
synonym_name FOR object_name;
```

Synonym

- A synonym allows you to associate an alternate name with an object such as a table or view
- Eg CREATE SYNONYM emp

  FOR employees;

  SELECT * FROM emp;

SYNONYMS

- There are two categories of synonyms, *public* and *private*.
- A public synonym can be accessed by any system user.
- The individual creating a public synonym does not own the synonym – rather, it will belong to the PUBLIC user group that exists within Oracle.
- Private synonyms, on the other hand, belong to the system user that creates them and reside in that user's schema.

SYNONYMS

- A system user can grant the privilege to use private synonyms that they own to other system users.
- In order to create synonyms, you will need to have the CREATE SYNONYM privilege.
- This privilege will be granted to you by the DBA.
- You must have the CREATE PUBLIC SYNONYM privilege in order to create public synonyms.

Dropping Synonyms

- If you own a synonym, you have the right to drop (delete) the synonym. The DROP SYNONYM command is quite simple.

```
DROP SYNONYM synonym_name;
```

- In order to drop a public synonym you must include the PUBLIC keyword in the DROP SYNONYM command.
- In order to drop a public synonym, you must have the DROP PUBLIC SYNONYM privilege.

```
DROP PUBLIC SYNONYM
synonym_name;
```

Renaming Synonyms

- Private synonyms can be renamed with the RENAME SYNONYM command.
- All existing references to the synonym are automatically updated.
- Any system user with privileges to use a synonym will retain those privileges if the synonym name is changed.
- The syntax of the RENAME SYNONYM command is like that for the RENAME command for any other database object such as a view or table.

```
RENAME old_synonym_name TO
new_synonym_name;
```

Renaming Synonyms

- The RENAME SYNONYM command only works for private synonyms.
- If we attempt to rename a public synonym such as the *tblspaces* synonym, Oracle will return an ORA-04043: *object tblspaces does not exist* error message as is shown here.

```
RENAME tblspaces TO ts;
ORA-04043: object TBLSPACES
does not exist
```

SEQUENCES

- Oracle provides the capability to generate sequences of unique numbers, and they are called **sequences.**
- Just like tables, views, indexes, and synonyms, a sequence is a type of database object.
- Sequences are used to generate unique, sequential integer values that are used as primary key values in database tables.
- The sequence of numbers can be generated in either ascending or descending order.

Creating Sequences

- The syntax of the CREATE SEQUENCE command is fairly complex because it has numerous optional clauses.

```
CREATE SEQUENCE <sequence name>
[INCREMENT BY <number>]
[START WITH <start value number>]
[MAXVALUE <MAXIMUM VLAUE NUMBER>]
[NOMAXVALUE]
[MINVALUE <minimum value number>]
[CYCLE]
[NOCYCLE]
[CACHE <number of sequence value to cache>]
[NOCACHE]
[ORDER]
[NOORDER];
```

- **sequence_name:** Name of the sequence.
- **initial_value:** starting value from where the sequence starts. Initial_value should be greater than or equal to minimum value and less than equal to maximum value.
- **increment_value:** Value by which sequence will increment itself. Increment_value can be positive or negative.
- **minimum_value:** Minimum value of the sequence.
- **maximum_value:** Maximum value of the sequence.
- **cycle:** When sequence reaches its set_limit it starts from beginning.
- **nocycle:** An exception will be thrown if sequence exceeds its max_value.

Example

```
CREATE SEQUENCE order_number_sequence
INCREMENT BY 1
START WITH 1
MAXVALUE 100000000
MINVALUE 1
CYCLE
CACHE 10;
Sequence created.
```

Accessing Sequence Values

- Sequence values are generated through the use of two *pseudocolumns* named *currval* and *nextval*.
- A pseudocolumn behaves like a table column, but psuedocolumns are not actually stored in a table.
- We can select values from pseudocolumns but cannot perform manipulations on their values.
- The first time you select the *nextval* pseudocolumn, the initial value in the sequence is returned.
- Subsequent selections of the *nextval* pseudocolumn will cause the sequence to increment as specified by the INCREMENT BY clause and will return the newly generated sequence value.

Accessing Sequence Values

- The *currval* pseudocolumn returns the current value of the sequence, which is the value returned by the last reference to nextval.

- **Example**

```
CREATE TABLE sales_order (
  order_number NUMBER(9)
 CONSTRAINT pk_sales_order PRIMARY KEY,
 order_amount NUMBER(9,2));
```

Accessing Sequence Values

- The INSERT commands shown below insert three rows into the *sales_order* table. The INSERT commands reference the *order_number_sequence.nextval* pseudocolumn.

```
INSERT INTO sales_order
VALUES(order_number_sequence.nextval,
155.59 );
INSERT INTO sales_order
VALUES(order_number_sequence.nextval,
450.00 );
INSERT INTO sales_order
VALUES(order_number_sequence.nextval,
16.95);
```

Accessing Sequence Values

```
SELECT *
FROM sales_order;
```

```
ORDER_NUMBER ORDER_AMOUNT
------------ ------------
           1       155.59
           2          450
           3        16.95
```

Accessing Sequence Values

- Use of *currval.*

```
CREATE TABLE order_details (
            order_number        NUMBER(9),
      order_row           NUMBER(3),
            product_desc        VARCHAR2(15),
      quantity_ordered  NUMBER(3),
            product_price       NUMBER(9,2),
CONSTRAINT pk_order_details
            PRIMARY KEY (order_number, order_row),
CONSTRAINT fk_order_number FOREIGN KEY
(order_number) REFERENCES sales_order);
```

Accessing Sequence Values

- The order_details table has a FOREIGN KEY reference to the sales_order table through the order_number column.

```
DELETE FROM sales_order;
INSERT INTO sales_order
    VALUES ( order_number_sequence.nextval, 200.00 );
INSERT INTO order_details
    VALUES ( order_number_sequence.currval, 1, 'End
   Table',1, 100.00);
INSERT INTO order_details
    VALUES ( order_number_sequence.currval, 2, 'Table
   Lamp',2, 50.00);
```

Accessing Sequence Values

```
SELECT *
FROM sales_order;

ORDER_NUMBER ORDER_AMOUNT
------------ ------------
           5          200

SELECT *
FROM order_details;

ORDER_NUMBER ORDER_ROW  PRODUCT_DESC  QUANTITY_ORDERED PRODUCT_PRICE
------------ ---------- ------------ ---------------- -------------
           5          1 End Table                   1           100
           5          2 Table Lamp                  2            50
```

Altering a Sequence

- A sequence is usually altered when it is desirable to set or eliminate the values of the MINVALUE or MAXVALUE parameters, or to change the INCREMENT BY value, or to change the number of cached sequence numbers.
- The ALTER SEQUENCE command shown here changes the MAXVALUE of the order_number_sequence to 200,000,000.

```
ALTER SEQUENCE order_number_sequence MAXVALUE
200000000;
Sequence altered.
```

Altering a Sequence

- When specifying a MINVALUE clause, the specified value should be less than the MAXVALUE where a sequence generates ascending numbers.
- In the case of a descending sequence, the MAXVALUE should be less than the MINVALUE.

Viewing Sequence Properties

- You may need to review the names and properties of your sequences.
- You can do this by querying the USER_SEQUENCES system view with a SELECT command.This view is part of the database's data dictionary.

```
SELECT * FROM USER_SEQUENCES;

SEQUENCE_NAME      MIN_VAL MAX_VALUE   INCRE  C  O  CACHE_SIZE Last_N
---------------- ------ ---------- ----- -- -- --------- ----
  -
ORDER_NUMBER_SEQUENCE   1   200000000 1     Y  N         10     6
```

Dropping a Sequence

- DROP SEQUENCE command is used to drop sequences that need to be recreated or are no longer needed.
- The general format is shown here along with an example that drops the *order_number_sequence* object.

```
DROP SEQUENCE <sequence name>;
DROP SEQUENCE order_number_sequence;
Sequence dropped.
```

END