

CREATING VIEWS

-- CHAPTER 12 --

- 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 another table or 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.
- Why use views? Views restrict access to the data because the view can display selective columns from the table. Views allow users to make simple queries to retrieve the results from complicated queries. For example, views allow users to view data from multiple tables without 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.
- There are two main types of view: Simple and Complex
- Simple View:
 - Derives data from only one table
 - Contains no functions or groups of data
 - Can perform DML through a view
- Complex View:
 - Derives data from many tables
 - Contains functions or groups of data
 - Does not always allow DML through the view
- Creating a View: You can create a subquery within the CREATE VIEW statement.

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

- OR REPLACE:
 - Re-creates the view if it already exists.
- FORCE:
 - Creates the view regardless of whether or not the base tables exists.
- NOFORCE:
 - Creates the view only if the base table exists (this is the default).
- View:
 - Is the name of the view.
- Alias:
 - 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).
- Subquery:
 - Is a complete SELECT statement. Note that this select statement cannot contain an ORDER BY clause.(You can use aliases for the columns in the SELECT list).
- WITH CHECK OPTION:
 - Specifies that only rows accessible to the view can be inserted or updated.
- Constraint:

- Is the name assigned to the CHECK OPTION constraint.
- WITH READ ONLY:
 - Ensures that no DML operations can be performed on this view.
- Here's an example of creating a view and using it to view its contents...

```
SQL> CREATE VIEW empvu10
  2 AS SELECT empno, ename, job
  3 FROM emp
  4 WHERE deptno=10;
```

View created.

```
SQL> select * from empvu10;
```

EMPNO	ENAME	JOB
7782	CLARK	MANAGER
7839	KING	PRESIDENT
7096	BLACK	SALESMAN
2296	AROMANO	SALESMAN
4296	AROMANO	SALESMAN
3354	AROMANO	SALESMAN

6 rows selected.

```
SQL> select *
  2 from empvu10
  3 where empno > 7000;
```

EMPNO	ENAME	JOB
7096	BLACK	SALESMAN
7782	CLARK	MANAGER
7839	KING	PRESIDENT

- Note that once the view is created, you may simply use a select statement to view its contents just as if the view was a table itself!
- You may also use the DESCRIBE statement to see the structure of the view as shown below:

```
SQL> describe empvu10;
Name                               Null?    Type
-----
EMPNO                               NOT NULL NUMBER(4)
ENAME                               VARCHAR2(10)
JOB                                  VARCHAR2(9)
```

- Also, once a view has been created, you can query the data dictionary table called USER_VIEWS to see the name of the view and the view definition. The text of the SELECT statement that constitutes your views is stored in a LONG column.
- Views can be modified by using the CREATE OR REPLACE VIEW clauses. The following example shows how to add an alias name to an existing view...

```
SQL> CREATE OR REPLACE VIEW empvu10
  2 (employee_number, employee_name, job_title)
  3 AS SELECT empno, ename, job
  4 FROM emp
  5 WHERE deptno = 10;
```

View created.

```
SQL> select * from empvu10;
```

EMPLOYEE_NUMBER	EMPLOYEE_N	JOB_TITLE
7782	CLARK	MANAGER
7839	KING	PRESIDENT
7096	BLACK	SALESMAN
2296	AROMANO	SALESMAN
4296	AROMANO	SALESMAN
3354	AROMANO	SALESMAN

- **Creating a Complex View:** The following example shows how to create a complex view that contains group functions to display values from two tables...

```
SQL> r
  1 CREATE VIEW dept_sum_vu
  2 (name, minsal, maxsal, avgsal)
  3 AS SELECT d.dname, MIN(e.sal), MAX(e.sal), AVG(e.sal)
  4 FROM emp e, dept d
  5 WHERE e.deptno = d.deptno
  6* GROUP BY d.dname
```

View created.

```
SQL> select * from dept_sum_vu;
```

NAME	MINSAL	MAXSAL	AVGSAL
ACCOUNTING	1000	3000	1650
DEVELOPMENT	99	4000	1978.8947
OPERATIONS	5000	5500	5250
RESEARCH	1500	2000	1750
SALES	1000	2000	1250

- **Rules for performing DML Operations on a View**
 - You can 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.
 - You cannot modify data in a view if it contains:
 - Any of the conditions mentioned in the previous rule
 - Columns defined by expressions
 - The ROWNUM pseudocolumn

- You cannot add data if:
 - The view contains any of the conditions mentioned above or in the previous rule
 - There are NOT NULL columns in the base tables that are not selected by the view.
- 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 are not allowed to create rows that the view cannot select, and therefore it allows integrity constraints and data validation checks to be enforced on data being inserted or updated.
- 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 on any row in the view will result in an ORACLE SERVER error.
- Removing a View: Remove a view without losing data because a view is based on underlying tables in the database.

```
SQL> select * from empvu10;
select * from empvu10
          *
ERROR at line 1:
ORA-00942: table or view does not exist
```

- Dropping views has no effect on the tables on which the views are based.
- Inline Views: An inline view is a subquery with an alias (correlation name) that you can use within a SQL statement. An inline view is similar to using a named subquery in the FROM clause of the main query. An inline view is not a schema object.
- “Top-N” Analysis
 - Top-N queries ask for the n largest or smallest values of all column
 - What are the ten best selling products?
 - What are the ten worst selling products?
 - Both largest values and smallest values sets are considered Top-N queries.
- Performing “Top-N” Analysis

```
SQL> SELECT [column_list], ROWNUM
FROM (SELECT [column_list] FROM table
ORDER BY Top-N_column)
WHERE ROWNUM <= N;
```

- Top-N queries use a consistent nested query structure with the elements described below:
 - Subquery or an inline view to generate the sorted list of data. The subquery or the inline-view includes the ORDER BY clause to ensure that the ranking is in the desired order. For results retrieving the largest values, DESC parameter is needed.
- Outer query to limit the number of rows in the final result set. The outer query includes the following components:
 - The ROWNUM pseudo-column, which assigns a sequential value starting with 1 to each of the rows returned from the query.
 - WHERE clause, which specifies the n rows to be returned. The outer WHERE clause must use a < or <= operator.

- The following is an example of a Top-N analysis...

```
SQL> r
 1 SELECT ROWNUM as RANK, ename, sal
 2 FROM (SELECT ename, sal FROM emp
 3        ORDER BY sal DESC)
 4* WHERE ROWNUM <= 10
```

RANK	ENAME	SAL
1	SI	
2	VIC_SANC	
3		
4	SHANK	5500
5	MOLITOR	5000
6	ALBERT	4000
7	BOBBY	3000
8	KING	3000
9	JONES	2000
10	CLARK	2000