**Notes on Updatable Views :**

The following notes apply to updatable views:

An updatable view is one you can use to insert, update, or delete base table rows. You can create a view to be inherently updatable, or you can create an **INSTEAD OF** trigger on any view to make it updatable.

For a view to be inherently updatable, the following conditions must be met:

* Each column in the view must map to a column of a single table. For example, if a view column maps to the output of a **TABLE** clause (an unnested collection), then the view is not inherently updatable.
* The view must not contain any of the following constructs:

A set operator

A **DISTINCT** operator

An aggregate or analytic function

A **GROUP** **BY**, **ORDER** **BY**, **MODEL**, **CONNECT** **BY**, or **START** **WITH** clause

A collection expression in a **SELECT** list

A subquery in a **SELECT** list

A subquery designated **WITH READ ONLY**

* In addition, if an inherently updatable view contains pseudocolumns or expressions, then you cannot update base table rows with an **UPDATE** statement that refers to any of these pseudocolumns or expressions.
* If you want a join view to be updatable, then all of the following conditions must be true:
  + The DML statement must affect only one table underlying the join.
  + For an **INSERT** statement, the view must not be created **WITH** **CHECK** **OPTION**, and all columns into which values are inserted must come from a **key-preserved table**. A key-preserved table is one for which every primary key or unique key value in the base table is also unique in the join view.
  + For an **UPDATE** statement, the view must not be created **WITH** **CHECK** **OPTION**, and all columns updated must be extracted from a key-preserved table.
* For a **DELETE** statement, if the join results in more than one key-preserved table, then Oracle Database deletes from the first table named in the **FROM** clause, whether or not the view was created **WITH** **CHECK** **OPTION**.

Restrictions on DML operations for views use the following criteria in the order listed:

* If a view is defined by a query that contains SET or DISTINCT operators, a GROUP BY clause, or a group function, then rows cannot be inserted into, updated in, or deleted from the base tables using the view.
* If a view is defined with WITH CHECK OPTION, a row cannot be inserted into, or updated in, the base table (using the view), if the view cannot select the row from the base table.
* If a NOT NULL column that does not have a DEFAULT clause is omitted from the view, then a row cannot be inserted into the base table using the view.
* If the view was created by using an expression, such as DECODE(deptno, 10, "SALES", ...), then rows cannot be inserted into or updated in the base table using the view.

**Creating a View:**

**Example**The following statement creates a view of the sample table **employees** named **emp\_view**. The view shows the employees in department 20 and their annual salary:

CREATE VIEW emp\_view AS

SELECT last\_name, salary\*12 annual\_salary

FROM employees

WHERE department\_id = 20;

The view declaration need not define a name for the column based on the expression **salary\*12**, because the subquery uses a column alias (**annual\_salary**) for this expression.

**Creating a View with Constraints:**

**Example**The following statement creates a restricted view of the sample table **hr.employees** and defines a unique constraint on the **email** view column and a primary key constraint for the view on the **emp\_id** view column:

CREATE VIEW emp\_sal (emp\_id, last\_name,

email UNIQUE Key, PRIMARY KEY (emp\_id)) AS SELECT employee\_id, last\_name, email FROM employees;

**Creating an Updatable View:**

**Example**The following statement creates an updatable view named **clerk** of all clerks in the **employees** table. Only the employees' IDs, last names, department numbers, and jobs are visible in this view, and these columns can be updated only in rows where the employee is a kind of clerk:

CREATE VIEW clerk AS

SELECT employee\_id, last\_name, department\_id, job\_id

FROM employees

WHERE job\_id = 'PU\_CLERK'

or job\_id = 'SH\_CLERK'

or job\_id = 'ST\_CLERK';

This view lets you change the **job\_id** of a purchasing clerk to purchasing manager (**PU\_MAN**):

UPDATE clerk SET job\_id = 'PU\_MAN' WHERE employee\_id = 118;

The next example creates the same view **WITH** **CHECK** **OPTION**. You cannot subsequently insert a new row into **clerk** if the new employee is not a clerk. You can update an employee's **job\_id** from one type of clerk to another type of clerk, but the update in the preceding statement would fail, because the view cannot access employees with non-clerk **job\_id**.

CREATE VIEW clerk AS

SELECT employee\_id, last\_name, department\_id, job\_id

FROM employees

WHERE job\_id = 'PU\_CLERK'

or job\_id = 'SH\_CLERK'

or job\_id = 'ST\_CLERK'

WITH CHECK OPTION;

**Creating a Join View: Example**A join view is one whose view subquery contains a join. If at least one column in the join has a unique index, then it may be possible to modify one base table in a join view. You can query **USER\_UPDATABLE\_COLUMNS** to see whether the columns in a join view are updatable. For example:

CREATE VIEW locations\_view AS

SELECT d.department\_id, d.department\_name, l.location\_id, l.city

FROM departments d, locations l

WHERE d.location\_id = l.location\_id;

SELECT column\_name, updatable

FROM user\_updatable\_columns

WHERE table\_name = 'LOCATIONS\_VIEW';

COLUMN\_NAME UPD

------------------------------ ---

DEPARTMENT\_ID YES

DEPARTMENT\_NAME YES

LOCATION\_ID NO

CITY NO

In the preceding example, the primary key index on the **location\_id** column of the **locations** table is not unique in the **locations\_view** view. Therefore, **locations** is not a key-preserved table and columns from that base table are not updatable.

INSERT INTO locations\_view VALUES

(999, 'Entertainment', 87, 'Roma');

INSERT INTO locations\_view VALUES

\*

ERROR at line 1:

ORA-01776: cannot modify more than one base table through a join view

You can insert, update, or delete a row from the **departments** base table, because all the columns in the view mapping to the **departments** table are marked as updatable and because the primary key of **departments** is retained in the view.

INSERT INTO locations\_view (department\_id, department\_name)

VALUES (999, 'Entertainment');

1 row created.

What is the difference between & and &&?

"&" is used to create a *temporary substitution variable* that will prompt you for a value every time it is referenced. Example:

SQL> SELECT sal FROM emp WHERE ename LIKE '&NAME';

Enter value for name: SCOTT

old 1: SELECT sal FROM emp WHERE ename LIKE '&NAME'

new 1: SELECT sal FROM emp WHERE ename LIKE 'SCOTT'

SAL

----------

3000

SQL> /

Enter value for name: SCOTT

old 1: SELECT sal FROM emp WHERE ename LIKE '&NAME'

new 1: SELECT sal FROM emp WHERE ename LIKE 'SCOTT'

SAL

----------

3000

"&&" is used to create a *permanent substitution variable*. Once you have entered a value (defined the variable) its value will used every time the variable is referenced. Example:

SQL> SELECT sal FROM emp WHERE ename LIKE '&&NAME';

Enter value for name: SCOTT

old 1: SELECT sal FROM emp WHERE ename LIKE '&&NAME'

new 1: SELECT sal FROM emp WHERE ename LIKE 'SCOTT'

SAL

----------

3000

SQL> /

old 1: SELECT sal FROM emp WHERE ename LIKE '&&NAME'

new 1: SELECT sal FROM emp WHERE ename LIKE 'SCOTT'

SAL

----------

3000

The "&&" will actually define the variable similarly to what the DEFINE command or OLD\_VALUE/ NEW\_VALUE clauses of a COLUMN statement would have done.

SQL> define

DEFINE NAME = "SCOTT" (CHAR)

[[1]](#footnote-1)

1. s [↑](#footnote-ref-1)