(4)

19

## VIEWS

Logical data is how we want to see the current data in our database. Physical data is how this data is actually placed in our database.

Views are masks placed upon tables. This allows the programmer to develop a method via which we can display predetermined data to users according to our desire.

In a concurrent environment, where several people are querying a database different people will want to look at data differently, i.e. each group of people will want to see different fields of the same table

To make the querying of the table easier, Oracle provides for the generation of views. A view is created unique, according to the needs of each user, where the user can then, only access those fields of the database allowed by the view. This goes a long way in providing security for data within a table

The DBA treats a view just as it would treat a base table. Hence you can query a view exactly as though it were a base table. The query fired on a view would naturally run faster than if it were fired on the base table, as the view will be a subset of the total number of columns in the table.

It is a programming convention that a view name begins with vw to allow one to distinguish a view from a table when the name is used in the FROM clause of the SQL sentence.

Views may be created for the following reasons:

- The DBA stores the view as a definition only. Hence, there is no duplication of data.
- Simplifies queries.
- Can be queried as a base table itself.
- Provides data security.
- Avoids data redundancy.

### Creation of views :

<u>Syntax</u>:

```
CREATE VIEW viewname AS
        SELECT columnname, columnname
        FROM tablename
        WHERE columnname = expression list ;
```

<u>Example</u> : Create view on client_master for the admin department.

```
CREATE VIEW vw_clientadmin AS
        SELECT name, address1, address2  city, pincode, state
        FROM client_master ;
```

This creates a view by the name of *vw_clientadmin* based on the table *client_master*.

### Renaming the columns of a view :

<u>Example</u> :

```
CREATE VIEW vw_clientadmin AS
        SELECT name, addr1, addr2, city, pincode, state
        FROM client_master ;
```

Here the columns of the table are related to the view on a one-to-one relationship. The columns of the view can take on different names from the table columns, if required.

**Using Views:** (visual concept)

Table name : **Client_Master**(This is the base table from which the view is created)

| client_no | name | address1 | address2 | city | pincode | state |
|---|---|---|---|---|---|---|

Example :
```
CREATE VIEW vw_clientadmin AS
        SELECT name, address1, address2, city, pincode, state
        FROM client_master ;
```

vw *clientadmin*, the view created from *client master* will look as follows :

| name | address1 | address2 | city | pincode | state |
|---|---|---|---|---|---|

**Selecting a data set from a view :**

Example :
```
SELECT name, address1, address2, city, pincode, state
        FROM vw_clientadmin
        WHERE city IN ('BOMBAY', 'DELHI');
```

**Updateable Views :**

Views can also be used for data manipulation i.e. the user can perform the Insert, Update and the Delete operations on the view. The views on which data manipualtion can be done are called *Updateable Views*. Views that donot allow data manipuation are called *Reasonly* views. When you give a view name in the Update, Insert or Delete statement, the modifications to the data will be passed to the underlying table.

For the view to be updateable, it should meet the following criteria :

- The view must be created on a single table.
- The Primary key column of the table should be included in the view.
- Aggregate functions cannot be used in the select statement.
- The select statement used for creating a View should not include DISTINCT, GROUP BY or HAVING clause.
- The select statement used for creating a View should not include use subqueries for the creation of views.
- if a view is defined from another view, the second view should be updateable.
- It must not use constants, strings or value expressions like sell_price * 1.05.
- For insert, it should include all the NOT NULL fields.

**Destroying a view :**

A view can be dropped by using the DROP VIEW command.

Syntax : DROP VIEW viewname;

Example : DROP VIEW vw_clientadmin;

QUE : WRITE A SHORT NOTE ON SEQUENCES OR
WHEN AND HOW CAN WE CREATE A SEQUENCE ? EXPLAIN

ANS : Most applications require the automatic generation of a numeric value.
The technique used by some developers is to create table with two columns. One column would contain the value of CNAME and the other column would contain the value of CNUM
The table's CNUM value could be updated whenever the next highest value is requested.
ORACLE provides an automatic sequences generator of numeric values, which can have a maximum value of upto 38 digits. A sequence can be defined to
Generate numbers in ascending or descending.
Provide intervals between numbers.
Caching of sequence number in memory.

[illegible]

## CREATING SEQUENCES:

A sequence can be created by issuing the following syntax.

Syntax:

```
✓ CREATE SEQUENCES sequence name
  INCREMENT BY integer values
  START WITH integer values
  MAX VALUE integer values / NOMAX VALUE
  MIN VALUE integer values / NOMIN VALUE
  CYCLE / NOCYCLE
  CACHE integer values / NOCACHE
  ORDER / NOORDER
```

[illegible]

Ex:

```
Create sequence temp
Increment by 1
Start with    1
Max value    100
Cycle;
```

How to use sequence with table when we insert a record into table?

Create stu table with snó, name & city now insert the record into stu table with following command.

```
        Insert into stu (sno,name,city)values
                (temp.nextval,'Rohit','Modasa');
```

here we insert value of sno into stu table by use of temp sequence and here generate automatic next number

KEYWORDS AND PARAMETERS OF SEQUENCE

SEQUENCE NAME
sequence name indicate the name of sequence

INCREMENT BY:
Increment By specifies the interval between sequence number. It can be any positive or negative value but not zero, if this clause is omitted, the default values is 1.

MINVALUE
Specifies the sequences minimum value

MAXVALUE
Max value specifies the maximum value that a sequence can generate

STARAT WITH
Specifies the first sequence number to be generated . The default for an ascending sequence is the sequences minimum values and for descending it is the maximum value.

CYCLE
specifies that the sequence continue to generate values after reaching either its maximum or minimum value.

CACHE
Cache specifies how many values of the sequence ORACLE preallocates and keeps in memory for faster access. The minimum values for this parameter is two.

NOCACHE
Nocache specifies that the values of the sequence are not peallocated. If the cache/nocache clause is omitted oracle cache 20 sequence numbers by default.

ORDER
Order guarantees that sequence numbers are generated in order of request . This is only necessary if you area using exclusive mode as it will always be in order

NOORDER
Noorder does not guarantee sequence numbers are generated I order of request. If the order/ noorder clause is omitted it takes the noorder clause by default

## ALTERING A SEQUENCE:

Sequence can be altered by using the alter : equence statement

Syntax:

```
        Alter sequence sequence_name
        [increament by integervalue
        maxvalue integervalue/nomaxvalue
        minvalue integervalue/nominvalue
        cycle/nocycle
        cache integervalue/cocache
        order/noorder]
```

note: the starting value of the sequence cannot be altered

Ex:

Change the sequence temp to and interval between two number as 2

```
Alter sequence temp
Increment by 2
```

## DROPPING A SEQUENCE:

A sequence can be dropped by using the statement drop sequence

Syntax: Drop sequence sequence_name

Ex : Drop the sequence temp

QUE : WRITE A SHORT NOTE ON STORED PROCEDURES

ANS : WHAT ARE PROCEDURES ?

Procedures are named PL/SQL blocks that can take parameters, performs an action and pass value OR
A procedure is generally used to perform an action and to pass values.

There are three part of Procedures :

1. A declarative part,
2. An executable part, and
3. An option-handling part

DECLARATIVE PART:
The declarative part may contain declaration of cursors, constants, variables, exceptions and subprograms. These objects are local to the procedure.

EXECUTABLE PART:
The executable part contains a PL/SQL block consisting of statements assign values, control execution and manipulate ORACLE data. The action to be performed here and data returned back to the calling environment is also returned from here. Variables declared are put to use in this block.

EXCEPTION HANDLING PART:
This part contains code that performs an action to deal with exceptions raised during the execution of the executable part.

## ADVANTAGES OF PROCEDURES

1 Security : Stored procedures can help enforce data security.

2 Performance : It improves database performance in the following ways;
   1. Amount of information sent over a network is less.
   2. No complication step is required to execute the code.

3 Memory Allocation : Reduction in memory as stored procedures have shared memory capabilities so only one copy of procedure needs to be loaded for execution by multiple users.

4 Productivity : Increased development productivity, by writing a single procedure we can avoid redundant coding and productivity.

5 Integrity : Improves integrity, a procedure needs to be tested only once to guarantee that it returns an accurate result. Hence coding errors can be reduced.

## SYNTAX FOR CREATING STORED PROCEDURE

```
Syntax:     CREATE OR REPLACE
            PROCEDURE procedurename
                  (argument { IN,OUT,IN OUT}datatype,....){IS,AS}
                  variable declaration;
                  constant declaration;
            BEGIN
                  PL/SQL subprogram body;
            EXCEPTION
                  Exception PL/SQL block;
            END;
```

**KEYWORDS AND PARAMETERS :**

| | |
|---|---|
| REPLACE | recreates the procedure if it already exists. |
| Procedurename | is the name of the procedure to be created. |
| Argument | is the name of an argument to the procedure. Parentheses can be omitted if no arguments are present. |
| IN | specifies that you must specify a value for the argument when calling the procedure. |
| OUT | specifies that the procedure passes a value for this argument back to its calling environment after execution. |
| IN OUT | specifies that you must specify a value for the argument when calling the procedure and that the procedure passes a value for this argument back to its calling environment after execution. By default it takes IN. |
| Datatype | is the datatype of an argument. It supports any datatype supported by PL/SQL. |

EX : CREATE A PROCEDURE FOR CALCULATING THE SUM OF TWO NUMBER:

```
        CREATE  OR REPLACE PROCEDURE sum1
                (n1 IN number, n2  IN number , ans OUT number)
        IS
        BEGIN
                Ans := n1 + n2;
        END;
```

EXECUTE THE ABOVE PROCEDURE :

```
DECLARE
        Result   number;
BEGIN
        Sum1(&no1, &no2, result);
        dbms_output.put_line('the sum is ' ||result);
END;
```

## DELETING A STORED PROCEDURE

Syntex : DROP PROCEDURE procedurename;

Ex : DROP PROCEDURE sum1;

QUE EXPLAIN IN VS OUT ARGUMENT TYPE.

ANS : IN specifies that you must specify a value for the argument when calling the Procedure. OR

This argument accept the value from calling environment

OUT specifies that the procedure passes a value for this argument back to its calling environment after execution.

Ex :
```
CREATE OR REPLACE PROCEDURE sum1
        (n1 IN number, n2 IN number , ans OUT number)
IS
BEGIN
        Ans := n1 +.n2;
END;
```
Here n1 and n2 both are IN variable and ans OUT variable.

QUE : WRITE A SHORT NOTE ON INDEXES

ANS : An Index in an ordered list of a column or group of column in a table.

An index created on the single column of the table is called simple Index.
When multiple columns are included in the index it is called composite Index.

CREATING AN INDEX FOR A TABLE:

Simple Index : CREATE INDEX indexfilename
ON tablename (columnnme);

Ex : Create an index on the table customer, field cnum.

```
CREATE INDEX cust_idx
ON customer(cnum);
```

Composite Index : CREATE INDEX indexfilename
ON tablename (columnname, columnname);

Ex : Create a composite index on the salesman table for the columns snum and sname

```
CREATE INDEX sal_idx
ON salesman (snum, sname);
```

CREATE AN UNIQUE INDEX

Syntex : CREATE UNIQUE INDEX indexfilename
ON tablename(columnname);

Ex : Create a unique index on the table customer, field cnum

```
CREATE UNIQUE INDEX cuni_ndx
ON customer (cnum);
```

When the user defines a primary key or a unique key constraint, Oracle automatically creates unique indexes on the primary key or unique key.

DROPPING INDEXES :
An index can the be dropped by using the DROP INDEX command.

Syntax : DROP INDEX indexfilename;

Ex : Drop index cuni_idx;