VIEWS

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.

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, It will return *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.

Contd..

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.**

Example

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

View created.

Contd..

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.

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;
```

View Created.

VIEWS

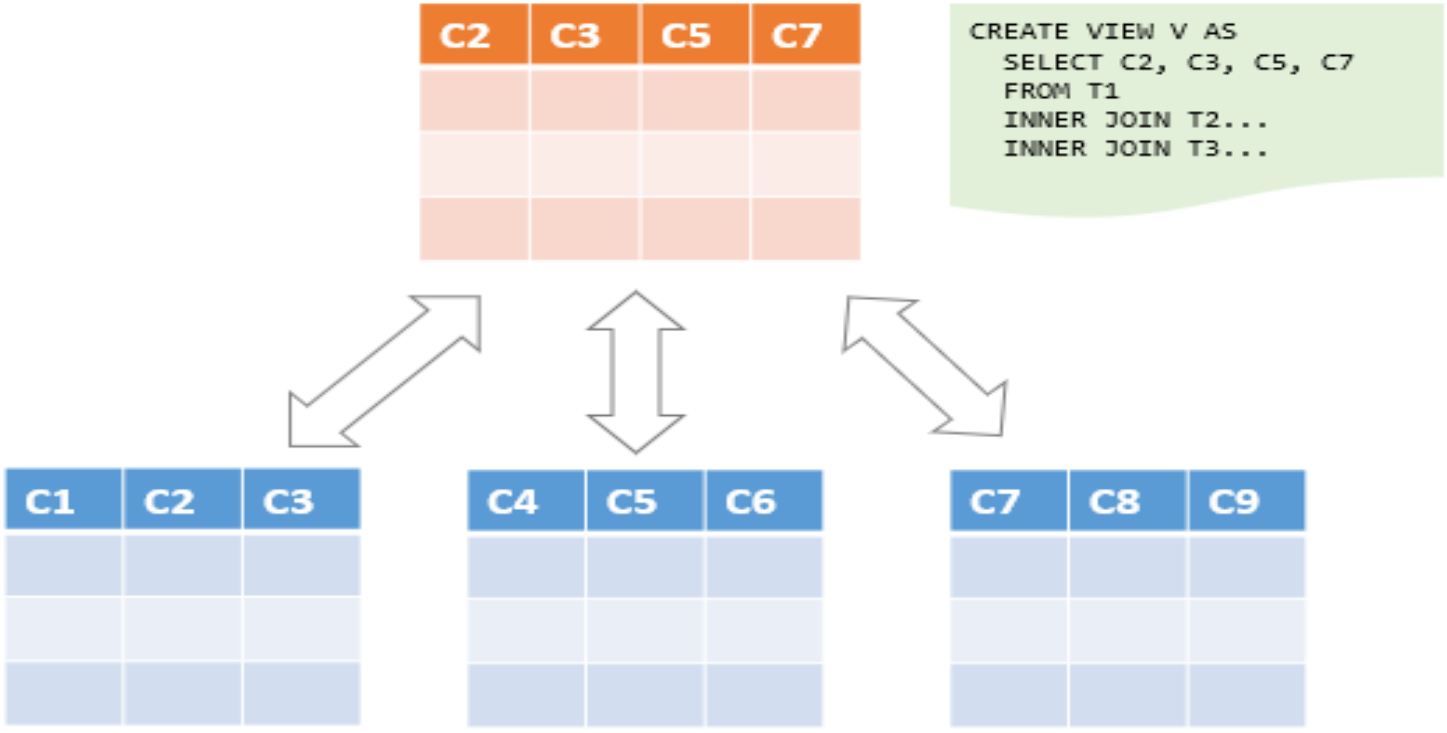
```
SELECT *
FROM employee_parking;

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.

View can be created using multiple tables



Types of Views

- Some Views are used only for looking at table data. Other Views can be used to Insert, Update and Delete table data as well as View data.
- If a View is used to only look at table data and nothing else the View is called a **Read-Only View**.
- A View that is used to look at table data as well as Insert, Update and Delete table data is called an **Updateable View.**

Types of views :

Read-only View : Allows only SELECT operations.

Updateable View : Allows SELECT as well as INSERT , UPDATE and DELETE operations.

The reasons why views are created are:

- When **Data redundancy** is to be kept to the minimum while maintaining **data security.**
- A database view allows you to simplify complex queries A database view helps limit data access to specific users. A database view enables computed columns.
- A database view enables backward compatibility.
- When **Data security** is required .

CREATING A VIEW

Syntax:

```
CREATE <OR REPLACE> VIEW <ViewName> AS
   SELECT <ColumnName1 >, <ColumnName2>
   FROM <TableName> WHERE <ColumnName> =
   < Expression List> <WITH       READ ONLY> ;
```

Contd...

- This statements creates a view based on query specified in SELECT statement.
- OR REPLACE option recreates the view if it is already existing maintaning the privileges granted to view viewname.
- WITH READ ONLY option creates readonly view.

Read-Only VIEW

We can create a view with read-only option to restrict access to the view.

Syntax to create a view with Read-Only Access

CREATE or **REPLACE FORCE VIEW** view_name **AS**

SELECT column_name(s)

FROM table_name

WHERE condition **WITH read-only;**

The above syntax will create view for read-only purpose, we cannot Update or Insert data into read-only view. It will throw an error.

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.

Updateable Views :

Views can also be used for data manipulation . Views on which data manipulation can be done are called Updateable Views.

When an updateable view name is given in an Insert Update, or Delete SQL statement, modifications to data in the view will be immediately passed to the underlying table.

Contd..

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

- Views defined from Single table
- If the user wants to INSERT records with the help of a view, then the PRIMARY KEY column(s) and all the NOT NULL columns must be included in the view .
- The user can UPDATE, DELETE records with the help of a view even if the PRIMARY KEY column and NOT NULL column(s) are excluded from the view definition .

Contd..

- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain summary functions.
- The SELECT clause may not contain set functions
- The SELECT clause may not contain an ORDER
- BY clause.
- The FROM clause may not contain multiple tables.

Contd...

- The WHERE clause may not contain subqueries. The query may not contain GROUP BY or HAVING.
  - Calculated columns may not be updated.
  - All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

EXAMPLE:

Run SQL Command Line

```
SQL>update stu set name='xyz' where enroll=4866;

1 Row updated.

SQL>select * from stu;

ENROLL          NAME
-----------   ----------
      4866       xyz
      4546       BDSG
```

Inserting Rows into a View

Rows of data can be inserted into a view.

Here, we cannot insert rows in the CUSTOMERS_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in a similar way as you insert them in a table.

Deleting Rows into a View

Rows of data can be deleted from a view.

Following is an example to delete a record having AGE = 22.

```
SQL > DELETE FROM CUSTOMERS_VIEW
   WHERE age = 22;
```

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');
```

Contd...

```
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.

Contd..

```
SELECT *
FROM department
WHERE dpt_no >= 5;

 DPT_NO   DPT_NAME            DPT_MGRSS DPT_MGR_S
-----------  --------------------- ------------------ -----------------
         7        Production            999444444     22-MAY-
        18        Department 18         98
        19        Department 19
```

more rows are displayed…

More Examples

DELETE dept_view

WHERE dpt_no = 18 OR dpt_no = 19;

2 rows deleted.

Contd..

```
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
```

The WITH CHECK OPTION

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.

If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.

Contd...

The following code block has an example of creating same view CUSTOMERS_VIEW with the WITH CHECK OPTION.

```
CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM
CUSTOMERS
WHERE age IS NOT NULL
 WITH CHECK OPTION;
```

The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

Force **VIEW** Creation

FORCE keyword is used while creating a view, forcefully. This keyword is used to create a View even if the table does not exist. After creating a force View if we create the base table and enter values in it, the view will be automatically updated.

Syntax for forced View is,

CREATE or **REPLACE FORCE VIEW** view_name **AS**

SELECT column_name(s)

FROM table_name

WHERE condition;

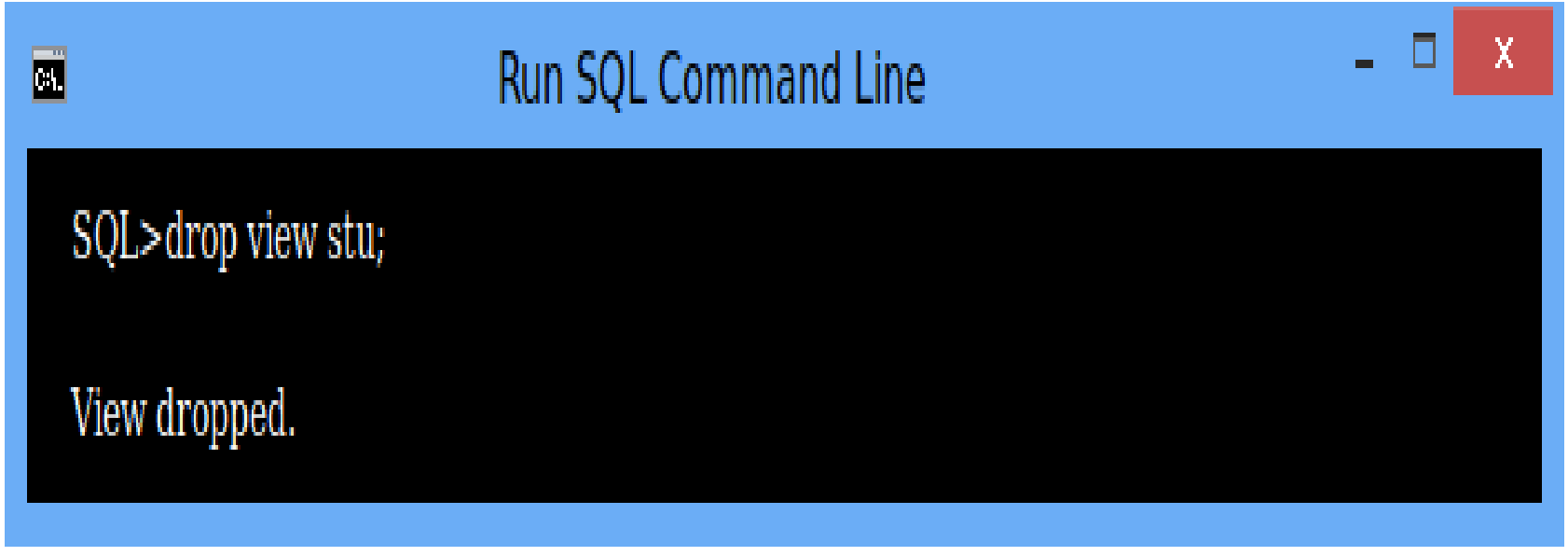
Destroying a View :

The drop command drops the specified view.

Syntax :

DROP VIEW Viewname;

EXAMPLE:



Why views won’t allow order by clause?

Views behave like tables whose contents are determined by the results of a query.

Tables don't have order; they're just bags of rows.

Tables... are just bags of rows - and then viwes are just virtual bags of virtual rows - views "don't exist" - e.g. there's no data stored for them at all - they're just "stored definitions of a query to be executed", basically.

FUNCTIONS AND VIEWS – A JOIN VIEW

- In addition to specifying columns from existing tables, you can use single row functions consisting of number, character, date, and group functions as well as expressions to create additional columns in views.

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.

Contd..

```
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 view "VIEW_NAME" has errors error message.

CREATING A VIEW WITH ERRORS

- In the CREATE VIEW command shown below, the table named *divisions* does not exist and the view is created with errors. Oracle returns an appropriate warning message.

```
CREATE FORCE VIEW div_view AS
 SELECT *
FROM divisions;
Warning: View created with
compilation errors.
```

- If we now create a table named divisions, a query of the invalid div_view view will execute, and the view is automatically recompiled and becomes valid.

Disadvantages of database view

Performance

Tables dependency

A Different View

Person(name, city)
Purchase(buyer, seller, product, store)
Product(name, maker, category)

```
CREATE VIEW    Seattle-view  AS
        SELECT  buyer, seller, product, store
        FROM    Person, Purchase
        WHERE  Person.city = "Seattle"     AND
                                    Person.name = Purchase.buyer
```

We have a new virtual table:
Seattle-view(buyer, seller, product, store)

A Different View

We can later use the view:

```
SELECT   name, store
 FROM    Seattle-view, Product
WHERE    Seattle-view.product =
Product.name  AND
                Product.category = “shoes”
```

What Happens When We Query a View ?

```
SELECT    name, Seattle-view.store
FROM      Seattle-view, Product
WHERE         Seattle-view.product = Product.name AND
            Product.category = “shoes”;
```

```
SELECT   name, Purchase.store
FROM     Person, Purchase,
Product
WHERE        Person.city = “Seattle”  AND
             Person.name = Purchase.buyer  AND
             Purchase.poduct = Product.name
             AND  Product.category = “shoes”;
```

Types of Views

- Virtual views:
  - Used in databases
  - Computed only on-demand – slow*er* at runtime
  - Always up to date
- Materialized views
  - Used in data warehouses
  - Precomputed offline – fast*er* at runtime
  - May have stale data

View	Materialized View
View is nothing but a set a sql statements together which join single or multiple tables and shows the data	Materialized view is a concept mainly used in Datawarehousing,
views does not store the data themselves but point to the data.	Materialized view store the data. Reason being it is easier/faster to access the data
View is a logical or virtual memory which is based on select query	materialized view is physical duplicate data in a table,it works faster than simple, Its works as snap shot and used for security purposes
simple view is the view in which we can not make DML command if the view is created by multiple tables	we can make DML command in materialize view
A view takes the output of a query and makes it appear like a virtual table	Materialized views are schema objects that can be used to summarize, precompute, replicate, and distribute data. E.g. to construct a data warehouse.

Contd..

You can use a view in most places where a table can be used	A materialized view can be stored in the same database as its base table(s) or in a different database.
All operations performed on a view will affect data in the base table and so are subject to the integrity constraints and triggers of the base table.	A materialized view provides indirect access to table data by storing the results of a query in a separate schema object. Unlike an ordinary view, which does not take up any storage space or contain any data.

Updating Views

How can I insert a tuple into a table that doesn't exist?

Employee(ssn, name, department, project, salary)

```
CREATE VIEW     Developers
  AS  SELECT name, project
FROM        Employee
WHERE department =
  "Development"
```

If we make the following insertion:

```
INSERT INTO    Developers
VALUES("Joe", "Optimizer")
```

It becomes:

```
INSERT INTO    Employee
VALUES(NULL, "Joe", NULL, "Optimizer", NULL)
```

Non-Updatable Views

```
CREATE VIEW Seattle-view  AS
    SELECT seller, product, store  FROM
    Person, Purchase

    WHERE        Person.city = "Seattle"  AND
                 Person.name = Purchase.buyer
```

How can we add the following tuple to the view?

("Joe","Shoe Model 12345","Nine West")

We need to add "Joe" to Person first, but we don't have all its attributes

Reusing a Materialized View

- Suppose I have **only** the result of SeattleView:

```
SELECT buyer, seller, product, store
FROM   Person, Purchase
WHERE  Person.city = 'Seattle'
       AND Person.per-name = Purchase.buyer
```

- and I want to answer the query

```
SELECT buyer, seller
FROM   Person, Purchase
WHERE  Person.city = 'Seattle'
       AND Person.per-name = Purchase.buyer AND
       Purchase.product='gizmo'.
```

Then, I can rewrite the query using the view.

Query Rewriting Using Views

Rewritten query:

```
SELECT      buyer, seller
FROM        SeattleView
WHERE       product= ‘gizmo’
```

Original query:

```
SELECT      buyer, seller
FROM        Person, Purchase
WHERE       Person.city = ‘Seattle’     AND
            Person.per-name = Purchase.buyer AND
            Purchase.product=‘gizmo’.
```

Another Example

- I still have **only** the result of SeattleView:

```
SELECT  buyer, seller, product, store
FROM    Person, Purchase
WHERE   Person.city = 'Seattle'    AND
        Person.per-name = Purchase.buyer
```

- but I want to answer the query

```
SELECT      buyer, seller
FROM    Person, Purchase
WHERE   Person.city = 'Seattle' AND  Person.per-name
        = Purchase.buyer AND  Person.PhoneLIKE
        '206 543 %'.
```

And Now?

- I still have **only** the result of SeattleView:

```
SELECT      buyer, seller, product, store
FROM        Person, Purchase, Product
WHERE       Person.city = 'Seattle'    AND
            Person.per-name = Purchase.buyer AND
            Purchase.product = Product.name
```

- but I want to answer the query

```
SELECT      buyer, seller
FROM        Person, Purchase
WHERE       Person.city = 'Seattle'    AND
            Person.per-name = Purchase.buyer.
```

And Now?

- I still have **only** the result of:

```
SELECT seller, buyer, Sum(Price)
FROM     Purchase
WHERE   Purchase.store = ‘The Bon’
Group By seller, buyer
```

- but I want to answer the query

```
SELECT seller, Sum(Price)
FROM     Purchase
WHERE   Person.store = ‘The Bon’
Group By seller
```

And what if it’s the other way around?

Finally…

- I still have **only** the result of:

```
SELECT      seller, buyer, Count(*)
FROM    Purchase
WHERE   Purchase.store = ‘The Bon’
Group By seller, buyer
```

- but I want to answer the query

```
SELECT      seller, Count(*)
FROM    Purchase
WHERE   Person.store = ‘The Bon’
Group By seller
```

Advantages and Disadvantages

- **Advantages**
  - Useful for summarizing, pre-computing, replicating an distributing data
  - Faster access for expensive and complex joins
  - Transparent to end-users

    MVs can be added/dropped without invalidating coded SQL

- **Disadvantages**
  - Performance costs of maintaining the views
  - Storage costs of maintaining the views

View	Materialized View
View is nothing but a set a sql statements together which join single or multiple tables and shows the data	Materialized view is a concept mainly used in Datawarehousing,
views does not store the data themselves but point to the data.	Materialized view store the data. Reason being it is easier/faster to access the data
View is a logical or virtual memory which is based on select query	materialized view is physical duplicate data in a table,it works faster than simple, Its works as snap shot and used for security purposes
simple view is the view in which we can not make DML command if the view is created by multiple tables	we can make DML command in materialize view
A view takes the output of a query and makes it appear like a virtual table	Materialized views are schema objects that can be used to summarize, precompute, replicate, and distribute data. E.g. to construct a data warehouse.

View	Materialized View
View is nothing but a set a sql statements together which join single or multiple tables and shows the data	Materialized view is a concept mainly used in Datawarehousing,
views does not store the data themselves but point to the data.	Materialized view store the data. Reason being it is easier/faster to access the data
View is a logical or virtual memory which is based on select query	materialized view is physical duplicate data in a table,it works faster than simple, Its works as snap shot and used for security purposes
simple view is the view in which we can not make DML command if the view is created by multiple tables	we can make DML command in materialize view
A view takes the output of a query and makes it appear like a virtual table	Materialized views are schema objects that can be used to summarize, precompute, replicate, and distribute data. E.g. to construct a data warehouse.