Views in SQL

- A **View** in SQL as a logical subset of data from one or more tables.
- Views are used to restrict data access.
- A View contains no data of its own but it is like a window through which data from tables can be viewed or changed.
- The table on which a View is based is called BASE Tables.
- There are 2 types of Views in SQL:

  Simple View and Complex View.

  Simple views can only contain a single base table.

  Complex views can be constructed on more than one base table. In particular, complex views can contain: join conditions, a group by clause, order by clause.

Working of views:

- A call to view in SQL query refers to the database and finds definition of views which is already stored in database.
- Then the DBMS convert this call of view into equal request on the base tables of the view and carries out the operations written in view definition and returns result set to query from which view is called.

View Definition

- A view is defined using the **create view** statement which has the form

  create view *v* **as** < query expression >

  where <query expression> is any legal SQL expression. The view name is represented by *v.*
- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- View definition is not the same as creating a new relation by evaluating the query expression
  - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.

Example Views

- A view of instructors without their salary

```
create view faculty as
   select ID, name, dept_name
   from instructor
```

- Find all instructors in the Biology department

```
select name
from faculty
where dept_name = ‘Biology’
```

- Create a view of department salary totals

```
 create view departments_total_salary(dept_name, total_salary) as
      select dept_name, sum (salary)
      from instructor
     group by dept_name;
```

The key differences between Simple and Complex types of Views are as follows:

S. No.	Simple View	Complex View
1.	Contains only one single base table or is created from only one table.	Contains more than one base table or is created from more than one table.
2.	We cannot use aggregate functions like MAX(), COUNT(), etc.	We can use all aggregate functions.
3.	DML operations could be performed through a simple view.	DML operations could not always be performed through a complex view.
	INSERT, DELETE and UPDATE are directly possible on a simple view.	We cannot apply INSERT, DELETE and UPDATE on complex view directly.
4.	Simple view does not contain group by, distinct, pseudocolumn like rownum, columns defined by expressions.	It can contain group by, distinct, pseudocolumn like rownum, columns defined by expressions.
7.	Does not include NOT NULL columns from base tables.	NOT NULL columns that are not selected by simple view can be included in complex view.
8.	In simple view, no need to apply major associations because of only one table.	In complex view, because of multiple tables involved general associations required to be applied such as join condition, group by or a order by clause.

9.	**Example:**
	CREATE VIEW Employee AS
	SELECT Empid, Empname
	FROM Employee
	WHERE Empid = ‘030314’;
		Example:
		CREATE VIEW EmployeeByDepartment AS
		SELECT e.emp_id, d.dept_id, e.dept_name FROM Employee e, Department d WHERE e.dept_id=d.dept_id;

Views Defined Using Other Views

-
  ```
  create view physics_fall_2009 as
     select course.course_id, sec_id, building, room_number
     from course, section
     where course.course_id = section.course_id
              and course.dept_name = ’Physics’
              and section.semester = ’Fall’
              and section.year = ’2009’;
  ```

-
  ```
  create view physics_fall_2009_watson as
      select course_id, room_number
      from physics_fall_2009
      where building= ’Watson’;
  ```

View Expansion

- Expand use of a view in a query/another view

```
create view physics_fall_2009_watson as
(select course_id, room_number
from (select course.course_id, building, room_number
        from course, section
        where course.course_id = section.course_id
            and course.dept_name = ’Physics’
            and section.semester = ’Fall’
            and section.year = ’2009’)
where building= ’Watson’;
```

Update of a View

- Add a new tuple to *faculty* view which we defined earlier

  insert into *faculty* **values** ('30765', 'Green', 'Music');

  This insertion must be represented by the insertion of the tuple

  ('30765', 'Green', 'Music', null)

  into the *instructor* relation

Instructor

ID	Name	Dept_name	title
('30765',	'Green',	'Music',	null)

Faculty

ID	Name	Dept_name
12345	John	singing
30765	Green	Music

```
create view faculty as
   select ID, name, Dept_name
   from instructor
```

Some Updates cannot be Translated Uniquely

- **create view** *instructor_info* **as**
  select *ID*, *name*, *building*
  from *instructor*, *department*
  where *instructor.dept_name= department.dept_name*;

- **insert into** *instructor_info* **values** ('69987', 'White', 'Taylor');
  - which department, if multiple departments in Taylor?
  - what if no department is in Taylor?

- Most SQL implementations allow updates only on simple views
  - The **from** clause has only one database relation.
  - The **select** clause contains only attribute names of the relation, and does not have any expressions, aggregates, or **distinct** specification.
  - Any attribute not listed in the **select** clause can be set to null
  - The query does not have a **group** by or **having** clause.

Instructor

ID	Name	Dept_name	title

Department

Dept_name	building	location	phone

ID	Name	Building
12345	John	XYZ
30765	Green	Taylor

And Some Not at All

- **create view** *history_instructors* **as**
  select *
  from *instructor*
  where *dept_name*= ’History’;
- What happens if we insert (’25566’, ’Brown’, ’Biology’, 100000) into *history_instructors?*

Materialized Views

- **Materializing a view**: create a physical table containing all the tuples in the result of the query defining the view
- If relations used in the query are updated, the materialized view result becomes out of date
    - Need to **maintain** the view, by updating the view whenever the underlying relations are updated.

Authorization

Forms of authorization on parts of the database:

- **Read** - allows reading, but not modification of data.
- **Insert** - allows insertion of new data, but not modification of existing data.
- **Update** - allows modification, but not deletion of data.
- **Delete** - allows deletion of data.

Forms of authorization to modify the database schema

- **Index** - allows creation and deletion of indices.
- **Resources** - allows creation of new relations.
- **Alteration** - allows addition or deletion of attributes in a relation.
- **Drop** - allows deletion of relations.

Authorization Specification in SQL

- The **grant** statement is used to confer authorization

  grant <privilege list>

  on <relation name or view name> **to** <user list>[with grant option]

- <user list> is:
  - a user-id
  - **public**, which allows all valid users the privilege granted
  - A role (more on this later)
- Granting a privilege on a view does not imply granting any privileges on the underlying relations.
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).

Privileges in SQL

- **select:** allows read access to relation,or the ability to query using the view
  - Example: grant users U_1, U_2, and U_3 **select** authorization on the *instructor* relation:

grant select on *instructor* **to** U_1, U_2, U_3

- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **all privileges**: used as a short form for all the allowable privileges

Revoking Authorization in SQL

- The **revoke** statement is used to revoke authorization.

  revoke <privilege list>

  on <relation name or view name> **from** <user list> [cascade/restrict]

- Example:

  revoke select on *branch* **from** U_1, U_2, U_3

- <privilege-list> may be **all** to revoke all privileges the revokee may hold.
- If <revokee-list> includes **public,** all users lose the privilege except those granted it explicitly.
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.
- All privileges that depend on the privilege being revoked are also revoked.

Roles

- **create role** instructor;
- **grant** *instructor* **to Amit;**
- Privileges can be granted to roles:
  - **grant select on** *takes* **to** *instructor*;
- Roles can be granted to users, as well as to other roles
  - **create role** *teaching_assistant*
  - **grant** *teaching_assistant* **to** *instructor*;
    - *Instructor* inherits all privileges of *teaching_assistant*
- Chain of roles
  - **create role** *dean*;
  - **grant** *instructor* **to** *dean*;
  - **grant** *dean* **to** Satoshi;

Authorization on Views

- **create view** *geo_instructor* **as**
  (**select** *
  from *instructor*
  where *dept_name* = ’Geology’);
- **grant select on** *geo_instructor* **to** *geo_staff*
- Suppose that a *geo_staff* member issues
  - **select** *
    from *geo_instructor*;
- What if
  - *geo_staff* does not have permissions on *instructor?*
  - creator of view did not have some permissions on *instructor?*

Other Authorization Features

- **references** privilege to create foreign key
  - **grant reference** (*dept_name*) **on** *department* **to** Mariano;
  - why is this required?
- transfer of privileges
  - **grant select on** *department* **to** Amit **with grant option**;
  - **revoke select on** *department* **from** Amit, Satoshi **cascade**;
  - **revoke select on** *department* **from** Amit, Satoshi **restrict**;
- Etc. read Section 4.6 for more details we have omitted here.

```
Dba-----create table student(rno int primary key,
sname varchar(30),
address varchar(40),
phone varchar(10)                  );
create user u1 with password 'u1';
create user u2 with password 'u2';
create user u3 with password 'u3';
grant select on student to u1;
create role instructor;
grant all on student to instructor ;
grant instructor to u1 with grant option;
U1-----grant select, delete on student to u2 with grant option;
DBA-----revoke instructor from u1 ;
revoke instructor from u1 cascae ;

select *from student;
insert into student values(2,'abc','mumbai',12345654);                  insert into
student values(3,'pqr','mumbai',12345654);                            insert into student
values(4,'pqr','mumbai',12345654);                          insert into student
values(5,'pqr','mumbai',12345654);                          insert into student
values(9,'pqr','mumbai',12345654);
grant insert on student to u1;
create user u12 with password 'u12';
grant all on student to u12
revoke select on student from u2 cascade
grant select on student to u3;
revoke insert on student from u2 restrict
grant all on database postgres to instructor;
```

Triggers (Active database)

- Trigger: A procedure that starts automatically if specified changes occur to the DBMS

- Three parts:
  - Event (activates the trigger)
  - Condition (tests whether the triggers should run) [Optional]
  - Action (what happens if the trigger runs)

- Semantics:
  - When event occurs, and condition is satisfied, the action is performed.

```
 create table department( dno int primary key,
                 dname varchar(30),
                 dlocation varchar(40),
                 dstartdate date);
 create table employee ( eid int primary key,
               ename varchar(20),
               dob date,
               address varchar(5),
               salary int,
               experience int,
               dno int,
                foreign key (dno) references department);
insert into department values(101,'mumbai', '23-10-2022')
insert into department values(102,'pune', '20-10-2021')


insert into employee values(1, 'john','12-01-2010','mum',20000,4,101)
 insert into employee values(2, 'smit','11-04-2010','mum',20000,5,101)
  insert into employee values(3, 'mery','12-01-2015','del',10000,3,102)
   insert into employee values(4, 'jana','12-01-2007','cz',10000,9,102)

   create view emp_sal as select eid, ename, salary, dno from employee
   select *from emp_sal
   select * from emp_sal where salary >1000
 select dno,count(*) as tot_emp from emp_sal group by dno

insert into employee values(5, 'jeny','12-01-2010','mum',15000,4,101)

insert into employee values(6, 'jerry','12-01-2010','mum',20000,4,101)


create view emp_dno as select eid, dno from emp_sal

select *from emp_dno

select *from emp_dno where dno=101


create view emp_dept as select eid,salary, address, employee.dno, dname from employee,
department where employee.dno=department.dno

select * from emp_dept
```