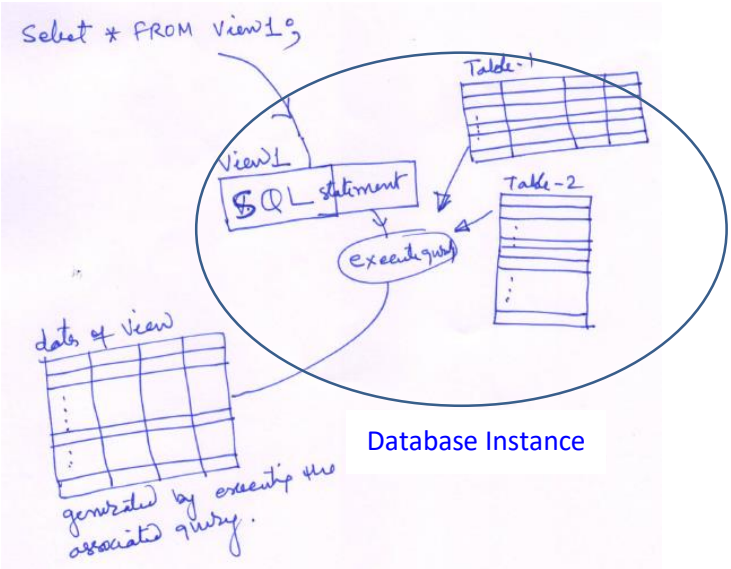
09. Programming Databases

Base Relations and Virtual Relations

Relations that are defined using CREATE TABLE statement actually hold the data, and are called "stored relations", "base relations", or Tables.

There is another type of relations that can exist in relational databases is, *Virtual Relation*. These are the relations that actually do not store data. They instead store query expression for generating data for the relation.



- These are "virtual" in the sense that they do not store tuples, and their tuples are generated by evaluating associated query expression.
- SQL refers virtual relations as **Views**; while term "Table" is used for base relations.
- Views can be used in query expressions in same manner base relations are used.

For example a "virtual relation" that is expressed as (that is associated query is) "SELECT * FROM EMPLOYEE NATURAL JOIN DEPARTMENT", and named as EMP, can be queried as SELECT * FROM EMP;

Creating and using a View

Views are created using **CREATE VIEW** statement

Example:

```
CREATE VIEW research_projects AS
    SELECT PNO, PNAME, PLOCATION FROM project NATURAL JOIN department
        WHERE dname = 'Research';
```

Query from: SELECT * FROM research_projects;

The query associated with view gets executed and result is given to FROM clause in this query. Once created, a view can be used like any other base relation in queries. For example

```
SELECT employee.* FROM research_projects NATURAL JOIN WORKS_ON
                    JOIN EMPLOYEE ON essn = ssn;

SELECT avg(salary) FROM research_projects NATURAL JOIN WORKS_ON
                    JOIN EMPLOYEE ON essn = ssn;
```

- When you create a view its definition is stored as an element in database instance. A databases instance will have a number of tables, views, and constraints (and some more … later).

- A variation to create view is "CREATE OR REPLACE VIEW"- allows to replace existing definition of view. There are certain limitations though while replacing an existing view definition by other.

  CREATE OR REPLACE VIEW research_projects AS ….

- Another example EMP view:

```
CREATE OR REPLACE VIEW EMP AS
  SELECT SSN, FNAME || ' ' || MINIT || '. ' || LNAME AS NAME, DNAME,
      CAST(SALARY/12 AS INT) AS SAL FROM EMPLOYEE NATURAL JOIN DEPARTMENT;
```

- You can rename name of attributes while creating views-

```
CREATE OR REPLACE VIEW EMP(SSN,NAME,DEPARTMENT, SALARY) AS
   SELECT SSN, FNAME || ' ' || MINIT || '. ' || LNAME AS NAME, DNAME,
       CAST(SALARY/12 AS INT) FROM EMPLOYEE NATURAL JOIN DEPARTMENT;
```

Dropping View

```
DROP VIEW PAY;
```

Benefits of Views? Why do we need Views?

1. To hide data from users: hiding full database structure from applications serves quite a few objectives.
   - Most applications do not require seeing full view of database tables
   - Some data are to be hidden from some users. For example we may not like to return salary column when SELECT * FROM EMPLOYEE is requested. Instead they do not know the name of actual relation and what they know is name of a view that does not include salary column. Sometimes user even may not know the relation that they are querying is Base Relation or Virtual Relation.
2. Views can be used to encapsulate complex queries: Complex queries are created by experts and saved as views. Application developer (like java developer) just write simple queries on views.
3. Views as reusable unit of business functionality in various applications.
4. Adds to "logical data independence": if applications deal with views, we can change the underlying schema without affecting applications
   - Making Database Schema independent of applications. If all database access is through views, structure of database can be independently changed without affecting application code.

Updatable views

Views are relations; virtual relations but sometimes application developer may not know if it is a base relation or a virtual relation. Motivated with this often database designers may allow user to update databases through views. This done by allowing user to apply INSERT, UPDATE and DELETE operations on views.

Challenge here is Views as such do not store data and we are attempting to update them?

Views on which update operations are allowed are called "Updatable Views". Since views has no data, the updating a view should actually update the "underlying base relation". For example our research projects can be an updatable view. We may be adding a tuple to it, modifying its tuple, or so -

```
UPDATE research_projects set pname = 'Process Automation' where pno = 1;
//this may update the corresponding tuple in Project base relation.

INSERT INTO research_projects VALUES (11, "New Initiatives", "Mumbai");
//this may in fact may insert a tuple  into Project relation with "appropriate DNO"
```

Note the issue in INSERT here; the insert should be inserting a tuple in underlying project relation; the Project relation has an attribute dno, which needs to be set to dno of research. Therefore, making this view updatable requires customizing the insertion operation.

We can also see some more issues associated with making views updatable through our EMP view, created earlier (and shown below, also)

```
CREATE OR REPLACE VIEW EMP AS
  SELECT SSN, FNAME || ' ' || MINIT || '. ' || LNAME AS NAME, DNAME,
      CAST(SALARY/12 AS INT) AS SAL FROM EMPLOYEE NATURAL JOIN DEPARTMENT;
```

Recall EMPLOYEE and DEPARTMENT relations from Company database as underlying base relations for view EMP. Given this, try figuring out what is expected to be updated in following updates to EMP view? What base attribute should that affect?

```
(U1) UPDATE EMP SET salary = 23456 where ssn=1234;

(U2) INSERT INTO EMP VALUES(…)

(U3) UPDATE EMP SET department = 'Marketing' where ssn=1234;
```

Update U1 may require salary to be multiplied by 12 before update.

Update U2 may require few mapping (from view attribute to base relation attribute) and salary conversion to annual.

Update U3 may require many things to be resolved, supplied new dname should be found in relation department, if found corresponding dno is to be placed in the relevant tuple in EMPLOYEE base relation.

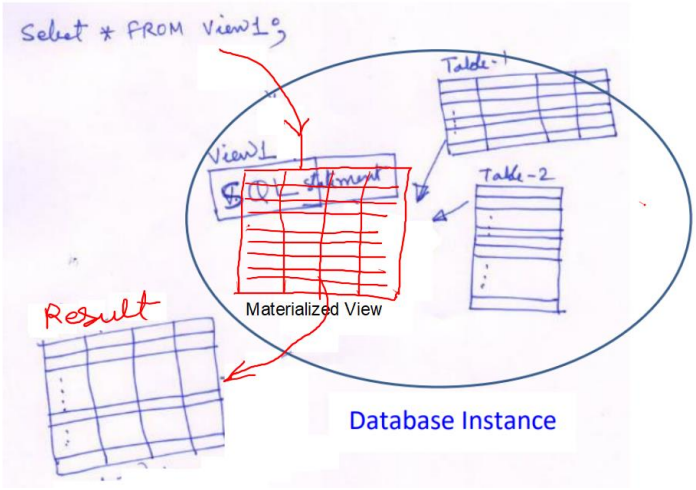
Actually making views updatable is a complex task.

Still it needs to be done!

SQL does provide certain rules: rules are complex; still in nutshell it is as following - a view can become automatically updatable if attributes in a view unambiguously maps to some attribute of some underlying base relation otherwise does not.

PostgreSQL views, by default, never updatable. We can make a postgresql view updatable through "INSTEAD OF triggers". INSTEAD OF Trigger typically is a "stored procedure" that actually gets executed instead of performing the requested update operation on the view/relation.

Materialized Views



- Materialized Views are views that compute the associated query once, and store the result as base relation.
- Materialized views are useful where source tables are huge, and computation of view takes longer. Data warehouses basically contain a large number of materialized views. For example, consider amazon management requiring various aggregated summary reports generated from sales/revenue all over the world. Answer of such queries might be implemented as materialized views.
- Materialized views may no longer be in sync with underlying base relations, as base relations might get updated since view was "materialized"
- Materialized views are periodically recomputed by executing associated SQL query. In the Amazon's example here, re-computation of views could be every Daily, Weekly, Monthly, Yearly, or so.

Views Summary

- Virtual relations. Relations that do not hold data, they have a query associated instead;
- Virtual in the sense that its tuples are generated by executing associated query.
- SQL refers virtual relations as Views; while term "Table" is used for base relations.
- Views can be used in query expressions in same manner base relations are used.
- If we can apply update operations (INSERT/UPDATE/DELETE) on views then it is updatable.
- There are certain rules that govern if a view can be updatable by default – if all attributes of view unambiguously maps to attributes in a base relation then views can be updatable. Otherwise there are other ways like "INSTEADOF trigger" to make a view updatable.
- Sometimes view is materialized, that is computed result of the view query is stored as base relations. As underlying base relations get updated, a materialized view can become "obsolete", and periodically can be recomputed.