Views

So far, we have looked at SQL tables
- Relations that are persistent
- Physically stored in the DBMS

It is also possible to have virtual relations, or views
- Defined by an expression which is a SQL query
- Do not exist physically in DBMS
  - Although it is possible to use materialized views

Views can be queried directly
- In some cases, it is also possible to modify views

Creating a view

View

```
CREATE VIEW RegionalSales (category, sales, state) AS
  SELECT P.category, S.sales, L.state
  FROM Products P, Sales S, Locations L
  WHERE P.pid = S.pid AND S.locid = L.locid
```

Defining Query (also referred to as View Subquery)
Base Tables

Equivalent Query (without views)

```
SELECT R.category, R.state, SUM(R.sales)
FROM RegionalSales R GROUP BY R.category, R.state
```

Views as subqueries

```
SELECT R.category, R.state, SUM(R.sales)
FROM (SELECT P.category, S.sales, L.state
      FROM Products P, Sales S, Locations L
      WHERE P.pid = S.pid AND S.locid = L.locid) R
GROUP BY R.category, R.state
```

Querying views

Querying Views

```
SELECT R.category, R.state, SUM(R.sales)
FROM RegionalSales R GROUP BY R.category, R.state
```

- Views are queried just like regular tables
  - A view is just another relation (albeit a virtual one)
  - Queries can involve both views and base tables
  - Helps to think of views in terms of analogy with window on data
Why are views useful? (1/3)

- **Usability**
  - Certain information must be retrieved from many tables
  - View abstraction can get all info in one (virtual) table
  - Queries are much easier to write on a single table
  - Subqueries that are often used can be included in queries without need for nesting

Why are views useful? (2/3)

- **Compatibility**
  - Shield users and application developer from changes
  - What if a schema changes? Define view that looks like the old schema
  - Users/applications access view, no changes needed in queries
  - “Obsolete” tables are preserved using views

Why are views useful? (3/3)

- **Security**
  - Restrict user access to certain data only
    - Managers and employees are given different “views” of same data
  - Both column- and row-level access control possible
  - Column-wise: students can only access Name and Age columns from a Student table
  - Row-wise: access only transactions above $10,000 value

Modifying views

- Is it possible to insert, update, delete tuples in a view?
  - Views are virtual …
  - … so modifications must be reflected in the base tables
- Why modifying views is a subtle issue?
  - Difficulty of translating view modifications in a unique way of updating base tables
  - Must be non-ambiguous in how to trace the base table tuple to update
- Views can be modified subject to restrictions
  - These are called updatable views
  - Still, many views are not updatable

Updatable Views

- SQL-92 provides formal definition of updatable view:
  1. View involves a single relation R. If R is a view, it must also be updatable (relaxed in SQL-99)
  2. Aggregate operations are not present in the view definition
  3. The DISTINCT keyword is not specified in SELECT clause
  4. All columns in subquery are simple columns, not expressions
  5. The WHERE clause must not contain a subquery involving R
  6. All attributes in R that are not in the SELECT clause of the view must not have both NOT NULL restriction and no default

Updatable Views (contd.)

- Insertion can be done directly on the base table
  - Other attributes in R set to NULL
- Deletion also possible
  - Delete tuple from base table
- Both insertion and deletion may cause problems!
Issues with insertion

**View Definition**

```sql
CREATE VIEW TopStudents (sname) AS
SELECT Name FROM Students S WHERE S.gpa > 3.0;
```

*Now let's insert a new student*

```sql
INSERT INTO TopStudents VALUES ('FirstLastName');
```

*GPA is set to NULL*

- Tuple falls outside view definition!
- Not a mistake, but update will not be reflected in view!
- **WITH CHECK OPTION** clause disallows such an insertion
- One solution is to include GPA in view definition

**Example**

Create view `ActorSummary` that lists for every actor the actor identifier, actor name, number of movies starred in, and the year of debut (i.e., the year of the earliest movie(s) the actor starred in). The view will have four columns with headings:

- **ID**
- **ActorName**
- **MovieCount**
- **DebutYear**

```sql
CREATE VIEW ActorSummary (ID, ActorName, MovieCount, DebutYear) AS
SELECT A.actor_id, A.name, COUNT(M.movie_id), MIN(M.year)
FROM Actors A, StarsIn S, Movies M
WHERE A.actor_id = S.actor_id AND S.movie_id = M.movie_id
GROUP BY A.actor_id, A.name;
```

**Example 2**

Create a view `ManagerSummary` that lists for every department the department name, manager ID and manager name, manager salary and the number of employees in that department. The view will have five columns with headings:

- **DeptName**
- **MgrID**
- **MgrName**
- **MgrSalary**
- **EmpCount**

```sql
CREATE VIEW ManagerSummary (DeptName, MgrID, MgrName, MgrSalary, EmpCount) AS
SELECT D.dname, D.managerid, E.ename, E.salary, COUNT(W.eid)
FROM Department D, Employee E, Works W
WHERE D.managerid = E.eid AND D.did = W.did
GROUP BY D.did, D.dname, D.managerid, E.ename, E.salary;
```