Views
Views

• A **view** is a relation defined by a query.
• The query defining the view is called the **view definition**
• For example:

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

• A view may be:
  - **virtual**: the view relation is defined, but not computed or stored.
    • Computed only on-demand – slow at runtime
    • Always up to date
  - **materialized**: the view relation is computed and stored in system.
    • Pre-computed offline – fast at runtime
    • May have stale data
Virtual view example

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)
View Example

We can use the view in a query as we would any other relation:

```
SELECT name, store
FROM Seattle-view, Product
WHERE Seattle-view.product = Product.name AND
      Product.category = "shoes"
```
Querying a virtual view

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

“View expansion”

```
SELECT name, Purchase.store
FROM Person, Purchase, Product
WHERE Person.city = "Seattle"    AND
    Person.name = Purchase.buyer    AND
    Purchase.product = Product.name  AND
    Product.category = "shoes"
```
The great utility of views

• Data independence
• Efficient query processing
  - materializing certain results can improve query execution
• Controlling access
  - Grant access to views only to filter data
• Data integration
  - Combine data sources using views
View-related issues

1. View selection
   • which views to materialize, given workload

2. View maintenance
   • when base relations change, (materialized) views need to be refreshed.

3. Updating virtual views
   • can users update relations that don’t exist?

4. Answering queries using views
   • when only views are available, what queries over base relations are answerable?
View Maintenance

• Two steps:
  ▪ **Propagate:** Compute changes to view when data changes.
  ▪ **Refresh:** Apply changes to the materialized view table.

• **Maintenance policy:** Controls when we do refresh.
  ▪ **Immediate:** As part of the transaction that modifies the underlying data tables. (+ Materialized view is always consistent; - updates are slowed)
  ▪ **Deferred:** Some time later, in a separate transaction. (- View becomes inconsistent; + can scale to maintain many views without slowing updates)
Deferred Maintenance

- Three flavors:
  - **Lazy**: Delay refresh until next query on view; then refresh before answering the query.
  - **Periodic (Snapshot)**: Refresh periodically. Queries possibly answered using outdated version of view tuples. Widely used, especially for asynchronous replication in distributed databases, and for warehouse applications.
  - **Event-based**: E.g., Refresh after a fixed number of updates to underlying data tables.
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(ssn, name, department, project, salary)
VALUES(NULL, "Joe", "Development", "Optimizer", NULL)
Non-Updatable Views

Person(name, city)
Purchase(buyer, seller, product, store)

```
CREATE VIEW City-Store AS
    SELECT Person.city, Purchase.store
    FROM Person, Purchase
    WHERE Person.name = Purchase.buyer
```

How can we add the following tuple to the view?

(“Seattle”, “Nine West”)

We don’t know the name of the person who made the purchase; cannot set to NULL.
Troublesome examples

CREATE VIEW OldEmployees AS
   SELECT name, age
   FROM Employee
   WHERE age > 30

INSERT INTO OldEmployees VALUES(“Joe”, 28)

If this tuple is inserted into view, it won’t appear. Allowed by default in SQL!
Ambiguous updates

<table>
<thead>
<tr>
<th>Name</th>
<th>group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>fac</td>
</tr>
<tr>
<td>Bob</td>
<td>fac</td>
</tr>
<tr>
<td>Bob</td>
<td>cvs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>group</th>
<th>file</th>
</tr>
</thead>
<tbody>
<tr>
<td>fac</td>
<td>foo.txt</td>
</tr>
<tr>
<td>fac</td>
<td>bar.txt</td>
</tr>
<tr>
<td>cvs</td>
<td>foo.txt</td>
</tr>
</tbody>
</table>

Join

Delete ("Alice", "foo.txt")
Updating views in practice

- Updates on views highly constrained:
  - SQL-92: updates only allowed on single-table views with projection, selection, no aggregates.
  - SQL-99: takes into account primary keys; updates on multiple table views may be allowed.
  - SQL-99: distinguishes between updatable and insertable views