Recall the Three Levels of Abstraction

**Views**

**Definition (View)**

A view is a relation in the external schema whose instance is determined by instances of relations in the conceptual schema.

A view has many of the same properties as a base relation in the conceptual schema:
- its schema information appears in the database schema
- access controls can be applied to it
- other views can be defined in terms of it
Types of Views

**Virtual view:** Only the definition is stored
- evaluating statements concerning a virtual view means rewriting those statements based on the view definition

**Materialized view:** Instance is stored in the DBS
- i.e. corresponding objects exist on the physical level

Creating and Deleting Views using SQL

- General form:
  ```sql
  create [materialized] view <name> as <query>
  drop view <name>
  ```
- Example:
  ```sql
  create view ResearchProjects as
  ( select projno, projname, lastname as empname
    from project, employee
    where respemp = empid and type = 'Research' )
  ```

Querying Views

Query a view as if it were a base relation.
```sql
select projname
from ResearchProjects
where empname = 'Smith'
```

What happens when you query a virtual view?
Recall: Query Processing Steps (View Resolution)

- Parsing
- Query Validation
- View Resolution
- Optimization
- Plan Compilation
- Execution

As given in:

Querying Views (cont’d)

```
select projname
from ResearchProjects
where empname = 'Smith'
```

Substitution:

```
select projname
from ( select projno, projname, lastname as empname
        from project, employee
        where respemp = eid and type = 'Research' )
where empname = 'Smith'
```

Rewrite to:

```
select projname
from project, employee
where respemp = eid and type = 'Research'
    and lastname = 'Smith'
```

Updating a Database via Views

- Modifications to a view’s instance must be propagated back to instances of relations in conceptual schema.
- Some views cannot be updated unambiguously.

<table>
<thead>
<tr>
<th>Person</th>
<th>External Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ed</td>
<td>Hockey</td>
</tr>
<tr>
<td>Dave</td>
<td>American Hockey</td>
</tr>
<tr>
<td>Wes</td>
<td>Canadian Hockey</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>National Pastimes</th>
<th>Personal Pastimes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hockey</td>
<td>American Hockey</td>
</tr>
<tr>
<td>Curling</td>
<td>Canadian Curling</td>
</tr>
</tbody>
</table>

NOTES:

- What does it mean to insert (Darryl, Hockey)?
- What does it mean to delete (Dave, Curling)?
View Updates in SQL

According to SQL-92, a view is updatable only if its definition satisfies a variety of conditions:

- The query references exactly one table
- The query only outputs simple attributes (no expressions)
- There is no grouping/aggregation/distinct
- There are no nested queries
- There are no set operations

These rules are more restrictive than necessary.

Motivation for Materializing Views

- Recall our example query:
  ```sql
  select projname
  from ResearchProjects
  where empname = 'Smith'
  ```
- and the rewritten version:
  ```sql
  select projname
  from project, employee
  where respemp = eid and type = 'Research'
      and lastname = 'Smith'
  ```
- Suppose we frequently ask this type of query, varying employees.
- Then, materializing the view might be a good idea.
- Additionally creating a clustered index on (the view's) attribute `empname` might be even better.

Motivation for Materializing Views (cont'd)

<table>
<thead>
<tr>
<th>Advantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using materialized views may improve query execution performance significantly (in particular for views with a complex definition).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Downside</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need to maintain the view as the underlying base relations change.</td>
</tr>
</tbody>
</table>

Notes
Issues Related to Using Materialized Views

**View design:** What views should we materialize, and what indexes should we build on those views?

**View exploitation:** Given a query and a set of materialized views, can we use the materialized views to answer (parts of) the query more efficiently?
- Different uses may be possible (another dimension for the optimizer)

**View maintenance:** How and when to refresh the materialized views?

---

Maintenance Policies

- **Immediate:** Do the refresh as part of the transaction that modifies the underlying base relations.
  - **Pro:** Materialized view is always consistent
  - **Con:** Updates are slowed

- **Deferred:** Do the refresh some time later, in a separate transaction.
  - **Pro:** Can scale to maintain many views without slowing updates
  - **Con:** View may be out of date

  - **Lazy** Delay refresh until next query on view; then refresh before answering the query.
  - **Periodic** Refresh periodically. Widely used, especially for asynchronous replication in distributed databases, and for warehouse applications.
  - **Event-based** E.g., after a fixed number of updates to underlying base relations.

---

Summary and Outlook

- Views are relations in the external schema whose instances are determined by relations in the conceptual schema
- Types: virtual views, materialized views
- Updating a database via views may be ambiguous
- First topic for our discussion next week
- Materialized views may speed up query execution significantly
- Issues related to using materialized views:
  - View design
  - View maintenance
  - View exploitation ← second topic next week