Introduction to Data Management
CSE 344

Lecture 17: Views
What is a View?

A view is a relation defined by a query

Customer(cid, name, city)
Purchase(cid, pid, store)
Product(pid, name, price)

CREATE VIEW StorePrice AS
SELECT x.store, y.price
FROM Purchase x, Product y
WHERE x.pid = y.pid

This is like a new table StorePrice(store,price)
How to Use a View?

• A "high end" store is a store that sold some product over 1000. For each customer, find all the high end stores that they visit. Return a set of (customer-name, high-end-store) pairs.

```
SELECT DISTINCT z.name, u.store
FROM Customer z, Purchase u, StorePrice v
WHERE z.cid = u.cid
AND u.store = v.store
AND v.price > 1000
```
Types of Views

• **Virtual views**
  – Used in databases
  – Computed only on-demand – slow at runtime
  – Always up to date

• **Materialized views**
  – Used in data warehouses
  – Pre-computed offline – fast at runtime
  – May have stale data
  – Indexes are materialized views
Queries Over Virtual Views: Query Modification

**View:**

```sql
CREATE VIEW StorePrice AS
SELECT x.store, y.price
FROM Purchase x, Product y
WHERE x.pid = y.pid
```

**Query:**

```sql
SELECT DISTINCT z.name, u.store
FROM Customer z, Purchase u, StorePrice v
WHERE z.cid = u.cid
AND u.store = v.store
AND v.price > 1000
```
Queries Over Virtual Views: Query Modification

Modified query:

```sql
SELECT DISTINCT z.name, u.store
FROM Customer z, Purchase u,
    (SELECT x.store, y.price
     FROM Purchase x, Product y
     WHERE x.pid = y.pid) v
WHERE z.cid = u.cid
AND u.store = v.store
AND v.price > 1000
```
Queries Over Virtual Views: Query Modification

Modified and unnested query:

```
SELECT DISTINCT z.name, u.store
FROM Customer z, Purchase u,
    Purchase x, Product y
WHERE z.cid = u.cid
AND u.store = x.store
AND y.price > 1000
AND x.pid = y.pid
```
Further Virtual Views
Optimizations

CREATE VIEW AcmePurchase AS
SELECT  x.cid, x.name as cname, x.city, z.pid, z.name as pname, z.price
FROM Customer x, Purchase y, Product z
WHERE x.cid = y.cid and y.store = 'ACME' and y.pid = z.pid

SELECT max(u.price)
FROM AcmePurchase u

SELECT max(z.price)
FROM Customer x, Purchase y, Product z
WHERE x.cid = y.cid and y.store = 'ACME' and y.pid = z.pid

First rewrite. Can we further optimize?
Applications of Virtual Views

• **Increased physical data independence.** E.g.
  – Vertical data partitioning
  – Horizontal data partitioning

• **Logical data independence.** E.g.
  – Change schemas of base relations (i.e., stored tables)

• **Security**
  – View reveals only what the users are allowed to know
### Vertical Partitioning

The image shows a table representing a database table with columns for SSN, Name, Address, Resume, and Picture. The table is vertically partitioned into three tables (T1, T2, T3) for clarity and performance optimization.

#### Original Table
<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
<th>Resume</th>
<th>Picture</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Mary</td>
<td>Huston</td>
<td>Clob1…</td>
<td>Blob1…</td>
</tr>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
<td>Clob2…</td>
<td>Blob2…</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Seattle</td>
<td>Clob3…</td>
<td>Blob3…</td>
</tr>
<tr>
<td>234234</td>
<td>Ann</td>
<td>Portland</td>
<td>Clob4…</td>
<td>Blob4…</td>
</tr>
</tbody>
</table>

#### T1
<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Mary</td>
<td>Huston</td>
</tr>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### T2
<table>
<thead>
<tr>
<th>SSN</th>
<th>Resume</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Clob1…</td>
</tr>
<tr>
<td>345345</td>
<td>Clob2…</td>
</tr>
</tbody>
</table>

#### T3
<table>
<thead>
<tr>
<th>SSN</th>
<th>Picture</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Blob1…</td>
</tr>
<tr>
<td>345345</td>
<td>Blob2…</td>
</tr>
</tbody>
</table>

This partitioning allows for faster query processing by reducing the amount of data that needs to be scanned.
CREATE VIEW Resumes AS

SELECT T1.ssn, T1.name, T1.address,
      T2.resume, T3.picture
FROM T1, T2, T3
WHERE T1.ssn=T2.ssn and T2.ssn=T3.ssn
Vertical Partitioning

```
SELECT address
FROM Resumes
WHERE name = 'Sue'
```

Which of the tables T1, T2, T3 will be queried by the system?

When do we use vertical partitioning?
Vertical Partitioning Applications

1. Can improve performance of some queries
   - When queries touch small fraction of columns
   - Only need to read desired columns from disk
   - Can produce big I/O savings for wide tables
   - Potential benefit in data warehousing applications

• But
  - Repeated key columns add a lot of overhead
  - Need expensive joins to reconstruct tuples
Vertical Partitioning Applications

2. When some fields are large and rarely accessed
   – E.g. Picture

3. In distributed databases
   – Customer personal info at one site, profile at another

4. In data integration
   – T1 comes from one source
   – T2 comes from a different source
## Horizontal Partitioning

### Customers

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>City</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Mary</td>
<td>Houston</td>
<td>USA</td>
</tr>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
<td>USA</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Seattle</td>
<td>USA</td>
</tr>
<tr>
<td>234234</td>
<td>Ann</td>
<td>Portland</td>
<td>USA</td>
</tr>
<tr>
<td>--</td>
<td>Frank</td>
<td>Calgary</td>
<td>Canada</td>
</tr>
<tr>
<td>--</td>
<td>Jean</td>
<td>Montreal</td>
<td>Canada</td>
</tr>
</tbody>
</table>

### CustomersInHouston

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>City</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>234234</td>
<td>Mary</td>
<td>Houston</td>
<td>USA</td>
</tr>
</tbody>
</table>

### CustomersInSeattle

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>City</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Seattle</td>
<td>USA</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Seattle</td>
<td>USA</td>
</tr>
</tbody>
</table>

### CustomersInCanada

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>City</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>Frank</td>
<td>Calgary</td>
<td>Canada</td>
</tr>
<tr>
<td>--</td>
<td>Jean</td>
<td>Montreal</td>
<td>Canada</td>
</tr>
</tbody>
</table>
Horizontal Partitioning

CREATE VIEW Customers AS
CustomersInHouston
UNION ALL
CustomersInSeattle
UNION ALL
...

CSE 344 - Fall 2011
Horizontal Partitioning

SELECT name
FROM Customers
WHERE city = ‘Seattle’

Which tables are inspected by the system?

WHY ???
Horizontal Partitioning

Better:

```sql
CREATE VIEW Customers AS
(SELECT * FROM CustomersInHouston
 WHERE city = 'Houston')
UNION ALL
(SELECT * FROM CustomersInSeattle
 WHERE city = 'Seattle')
UNION ALL
...
```

Other techniques exist: read DBMS documentation
Horizontal Partitioning

```
SELECT name
FROM Customers
WHERE city = 'Seattle'
```

```
SELECT name
FROM CustomersInSeattle
```
Horizontal Partitioning Applications

• Performance optimization
  – Especially for data warehousing
  – E.g. one partition per month
  – E.g. archived applications and active applications

• Distributed and parallel databases

• Data integration
Logical Data Independence

• Initially, we had schema:
  \texttt{Person(SSN, Name, Phone, City)}

• But we decide to change it to
  \texttt{P1(SSN, Name, City)}
  \texttt{P2(SSN, Phone)}

• But the application already uses the Person table in SQL queries. How can we avoid having to rewrite the application code?
Views and Security

Customers:

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Houston</td>
<td>450.99</td>
</tr>
<tr>
<td>Sue</td>
<td>Seattle</td>
<td>-240</td>
</tr>
<tr>
<td>Joan</td>
<td>Seattle</td>
<td>333.25</td>
</tr>
<tr>
<td>Ann</td>
<td>Portland</td>
<td>-520</td>
</tr>
</tbody>
</table>

CREATE VIEW PublicCustomers
SELECT Name, Address
FROM Customers

Fred is not allowed to see this

Fred is allowed to see this
Views and Security

Customers:

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Huston</td>
<td>450.99</td>
</tr>
<tr>
<td>Sue</td>
<td>Seattle</td>
<td>-240</td>
</tr>
<tr>
<td>Joan</td>
<td>Seattle</td>
<td>333.25</td>
</tr>
<tr>
<td>Ann</td>
<td>Portland</td>
<td>-520</td>
</tr>
</tbody>
</table>

John is not allowed to see >0 balances

CREATE VIEW BadCreditCustomers
    SELECT *
    FROM Customers
    WHERE Balance < 0
Levels of Abstraction

- **Physical Schema**: includes storage details, file organization, and indexes.
- **Conceptual Schema**: describes stored data in terms of a data model (a.k.a. logical schema).
- **External Schema**: schema seen by applications.
View Update Problem

• Since a view is like a table, can we update it?
• Example

  Purchase(cid, pid, store)
  Product(pid, name, price)

  CREATE VIEW StorePrice AS
  SELECT x.store, y.price
  FROM Purchase x, Product y
  WHERE x.pid = y.pid

  DELETE FROM StorePrice
  WHERE price < 10
View Update Problem

• **Updatable views (SQL-92)**
  – Defined on single base relation
  – No aggregation in definition
  – Inserts have NULL values for missing fields
  – Better if view definition includes primary key

• **Updatable views (SQL-99)**
  – May be defined on multiple tables

• **Messy issue in general**
Materialized Views

• Goal: Pre-compute content of view and store it
• Speeds-up query execution
• But materialized view maintenance is a problem
  – Every time someone updates the base tables
  – Need to re-compute the materialized view
• Materialized view selection is a physical tuning problem just like index selection
• Indexes are a type of materialized view