Lecture 05: Views
Example

Purchase(customer, product, store)
Product(pname, price)

```
CREATE VIEW CustomerPrice AS
SELECT x.customer, y.price
FROM Purchase x, Product y
WHERE x.product = y.pname
```

"virtual table"
Example

```
SELECT u.customer, v.store
FROM CustomerPrice u, Purchase v
WHERE u.customer = v.customer
    and u.price > 100
```

Purchase(customer, product, store)
Product(pname, price)
CustomerPrice(customer, price)
Queries Over Views: Query Modification

Purchase(customer, product, store)
Product(pname, price)

View:
CREATE VIEW CustomerPrice AS
  SELECT x.customer, y.price
  FROM Purchase x, Product y
  WHERE x.product = y.pname

Query:
SELECT u.customer, v.store
FROM CustomerPrice u, Purchase v
WHERE u.customer = v.customer
  and u.price > 100
CREATE VIEW CustomerPrice AS
(SELECT x.customer, y.price
FROM Purchase x, Product y
WHERE x.product = y.pname)

Modified query:

SELECT u.customer, v.store
FROM (SELECT x.customer, y.price
      FROM Purchase x, Product y
      WHERE x.product = y.pname)
      u, Purchase v
WHERE u.customer = v.customer
and u.price > 100
Queries Over Views: Query Modification

Modified and unnested query:

```
SELECT x.customer, v.store
FROM Purchase x, Product y, Purchase v,
WHERE x.customer = v.customer
  and y.price > 100
  and x.product = y.pname
```
Types of Views

- **Virtual views:**
  - Pros/Cons?

- **Materialized views**
  - Pros/Cons?
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
Data Integration

Local DB ... Local DB

Integrated data

Global as View

Local DB ... Local DB

Integrated data

Local as View

V

V1

Vk
Query Rewriting using Views

Suppose you have these views:

\[ V_1(x, y) \iff \text{black}(x), \text{edge}(x, y) \]
\[ V_2(x, y) \iff \text{edge}(x, y), \text{black}(y) \]

Can you rewrite this query in terms of the views?

\[ Q(x, y) \iff \text{edge}(x, z_1), \text{black}(z_1), \text{edge}(z_1, z_2), \text{edge}(z_2, z_3), \text{black}(z_3), \text{edge}(z_3, y) \]

Example from [Duschka&Genesereth'97]
Query Rewriting using Views

Suppose you have these views:

V1(x,y) :- black(x), edge(x,y)
V2(x,y) :- edge(x,y), black(y)

Can you rewrite this query in terms of the views?

Q(x,y) :- edge(x,z1), black(z1),
         edge(z1,z2), edge(z2,z3)
         black(z3), edge(z3,y)

Answer:

Q(x,y) :- V2(x,z1), V1(z1,z2), V2(z2,z3)
         V1(z3,y)
Query Rewriting using Views

Suppose you have these views:

\[
V1(x, y) \iff \text{black}(x), \text{edge}(x, y)
\]
\[
V2(x, y) \iff \text{edge}(x, y), \text{black}(y)
\]

What about this query?

\[
Q(x, y) \iff \text{black}(x), \text{edge}(x, z1), \text{black}(z1), \text{edge}(z1, z2), \text{black}(z2), \text{edge}(z2, z3), \text{black}(z3), \text{edge}(z3, y), \text{black}(y)
\]
Query Rewriting using Views

Suppose you have these views:

\[ V1(x,y) :- black(x), edge(x,y) \]
\[ V2(x,y) :- edge(x,y), black(y) \]

What about this query?

\[ Q(x,y) :- black(x), edge(x,z1), 
    black(z1), edge(z1,z2), 
    black(z2), edge(z2,z3), 
    black(z3), edge(z3,y), black(y) \]

Answer:

\[ Q(x,y) :- V1(x,z1), V1(z1,z2), V1(z2,z3), V1(z3,y), V2(z3,y) \]
Query Rewriting using Views

Suppose you have these views:

\[ V1(x, y) : \text{black}(x), \text{edge}(x, y) \]
\[ V2(x, y) : \text{edge}(x, y), \text{black}(y) \]

Can you rewrite this?

\[ Q(x, y) : \text{edge}(x, z1), \text{edge}(z1, z2), \]
\[ \text{edge}(z2, z3), \text{edge}(z3, y) \]
Query Rewriting using Views

Suppose you have these views:

\[
V1(x, y) \leftarrow \text{black}(x), \text{edge}(x, y) \\
V2(x, y) \leftarrow \text{edge}(x, y), \text{black}(y)
\]

Can you rewrite this?

\[
Q(x, y) \leftarrow \text{edge}(x, z1), \text{edge}(z1, z2), \text{edge}(z2, z3), \text{edge}(z3, y)
\]

No! Maximally contained rewrite is:

\[
Q(x, y) \leftarrow V1(x, z1), V2(z1, z2), V1(z2, z3), V2(z3, y) \\
Q(x, y) \leftarrow V2(x, z1), V2(z1, z2), V2(z2, z3), V2(z3, y) \\
Q(x, y) \leftarrow V2(x, z1), V1(z1, z2), V1(z2, z3), V2(z3, y) \\
\ldots \text{etc.}
\]
### Vertical Partitioning

<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>Amherst</td>
<td>Clob2...</td>
<td>Blob2...</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Amherst</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>

### Tables T1, T2, T3

**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>Amherst</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>
Vertical Partitioning

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

Why use vertical partitioning?

SELECT address
    FROM Resumes
    WHERE name = 'Sue'

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

When to do this:

- When some fields are large, and rarely accessed
  - E.g. Picture
- In distributed databases
  - Customer personal info at one site, customer profile at another
- In data integration
  - T1 comes from one source
  - T2 comes from a different source
Horizontal Partitioning

<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>Huston</td>
<td>USA</td>
</tr>
<tr>
<td>345345</td>
<td>Sue</td>
<td>Amherst</td>
<td>USA</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Amherst</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>

CustomersInHuston

<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>Huston</td>
<td>USA</td>
</tr>
</tbody>
</table>

CustomersInAmherst

<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>Amherst</td>
<td>USA</td>
</tr>
<tr>
<td>345343</td>
<td>Joan</td>
<td>Amherst</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
  CustomersInHuston
  UNION ALL
  CustomersInAmherst
  UNION ALL
... . . .

SELECT name
FROM Customers
WHERE city = ‘Amherst’

Which tables are inspected by the system?
Horizontal Partitioning

Better:

```sql
CREATE VIEW Customers AS
(SELECT * FROM CustomersInHuston
 WHERE city = 'Huston')
UNION ALL
(SELECT * FROM CustomersInAmherst
 WHERE city = 'Amherst')
UNION ALL
...
```
Horizontal Partitioning

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

```
SELECT name
FROM CustomersInAmherst
```
Horizontal Partitioning

- Optimizations:
  - E.g. archived applications and active applications

- Distributed databases

- Data integration
CREATE VIEW PublicCustomers
SELECT Name, Address
FROM Customers

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>Amherst</td>
<td>-240</td>
</tr>
<tr>
<td>Joan</td>
<td>Amherst</td>
<td>333.25</td>
</tr>
<tr>
<td>Ann</td>
<td>Portland</td>
<td>-520</td>
</tr>
</tbody>
</table>

Fred is not allowed to see this

Fred is allowed to see this
CREATE VIEW BadCreditCustomers
SELEC
FROM
WHERE

John is not allowed to see >0 balances
Views and Updates

Discussion:
- What happens when we insert a tuple to a view?
- Update a tuple from a view?
- Can we always/ever do this?
## View side-effects

<table>
<thead>
<tr>
<th>Query class</th>
<th>Deciding whether there is a side-effect-free deletion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queries involving PJ</td>
<td>NP-hard</td>
</tr>
<tr>
<td>Queries involving JU</td>
<td>NP-hard</td>
</tr>
<tr>
<td>SPU</td>
<td>P</td>
</tr>
<tr>
<td>SJ</td>
<td>P</td>
</tr>
</tbody>
</table>
# Source side-effects

<table>
<thead>
<tr>
<th>Query class</th>
<th>Finding the minimum source deletions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queries involving PJ</td>
<td>NP-hard</td>
</tr>
<tr>
<td>Queries involving JU</td>
<td>NP-hard</td>
</tr>
<tr>
<td>SPU</td>
<td>P</td>
</tr>
<tr>
<td>SJ</td>
<td>P</td>
</tr>
</tbody>
</table>