Database design and implementation

Lecture 05: Views
Example

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

“virtual table”

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

- Really important to speed up processing time

```
SELECT * 
FROM Person 
WHERE name = 'Smith'
```

```
CREATE INDEX Pname ON Person(name)
```

More on types of Indexes in later lecture
Creating Indexes

- They can be on more than one attribute

```sql
CREATE INDEX doubleIndex ON Person(age, city)
```

- For which of these queries is this index helpful?

```sql
SELECT * FROM Person WHERE age = 55
```

```sql
SELECT * FROM Person WHERE age = 55 and city = 'Amherst'
```

```sql
SELECT * FROM Person WHERE city = 'Amherst'
```
Creating Indexes

- They can be on more than one attribute

```
CREATE INDEX doubleIndex ON Person(age, city)
```

- For which of these queries is this index helpful?

<table>
<thead>
<tr>
<th>Query</th>
<th>Index Helpful</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SELECT * FROM Person WHERE age = 55</code></td>
<td>YES</td>
</tr>
<tr>
<td><code>SELECT * FROM Person WHERE city = 'Amherst'</code></td>
<td>NO</td>
</tr>
<tr>
<td><code>SELECT * FROM Person WHERE age = 55 AND city = 'Amherst'</code></td>
<td>YES</td>
</tr>
</tbody>
</table>
Indexes are Materialized Views

CREATE INDEX W
ON Person(age)

CREATE INDEX P
ON Person(city)

If W and P are views, what is their schema?
Indexes are Materialized Views

CREATE INDEX W
ON Person(age)

CREATE INDEX P
ON Person(city)

CREATE VIEW W
SELECT age, pid
FROM Person

CREATE VIEW P
SELECT city, pid
FROM Person

If W and P are views, what is their schema?

(index value, record id)
Indexes are Materialized Views

CREATE INDEX W ON Person(age)
CREATE INDEX P ON Person(city)

CREATE VIEW W
SELECT age, pid
FROM Person

CREATE VIEW P
SELECT city, pid
FROM Person

SELECT age, city
FROM Person
WHERE age > 22 and city LIKE 'A%'

SELECT x.age, y.city
FROM W x, P y
WHERE x.age > 22 and y.city LIKE 'A%' and x.pid = y.pid

Covering indexes
Data Integration

Local DB \ldots Local DB \rightsquigarrow \text{Integrated data} \leftarrow V \rightarrow \text{Global as View}

Local DB \ldots Local DB \rightsquigarrow \text{Integrated data} \leftarrow V_1 \rightarrow \text{Local as View} \leftarrow V_k
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)

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

Suppose you have these views:

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

Can you rewrite this query in terms of the views?

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

Answer:

\[ Q(x, y) \ :- \ V_2(x, z_1), \ V_1(z_1, z_2), \ V_2(z_2, z_3) \]
\[ V_1(z_3, y) \]

\[ \begin{align*}
  \text{x} & \rightarrow \text{y} \\
  \text{x} & \rightarrow \text{gray y} \\
  \text{x} & \rightarrow \text{any color} \\
  \text{any color} & \rightarrow \text{any color} \\
  \text{gray} & \rightarrow \text{gray} \\
  \text{gray} & \rightarrow \text{any color} \\
\end{align*} \]
Query Rewriting using Views

Suppose you have these views:

\[
\begin{align*}
V1(x,y) &: \text{black}(x), \text{edge}(x,y) \\
V2(x,y) &: \text{edge}(x,y), \text{black}(y)
\end{align*}
\]

What about this query?

\[
\begin{align*}
Q(x,y) &: \text{black}(x), \text{edge}(x,z1), \\
& \quad \text{black}(z1), \text{edge}(z1,z2), \\
& \quad \text{black}(z2), \text{edge}(z2,z3), \\
& \quad \text{black}(z3), \text{edge}(z3,y), \text{black}(y)
\end{align*}
\]
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:

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

Can you rewrite this?

\[ Q(x,y) \leftarrow \text{edge}(x,z_1), \text{edge}(z_1,z_2), \text{edge}(z_2,z_3), \text{edge}(z_3,y) \]
Query Rewriting using Views

Suppose you have these views:

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

Can you rewrite this?

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

No! Maximally contained rewrite is:

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

**Resumes**

<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>

**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

## 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>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>

## 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>

## 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
  CustomersInHouston
    UNION ALL
  CustomersInAmherst
    UNION ALL
  ...

SELECT name
FROM Customers
WHERE city = 'Amherst'

Which tables are inspected by the system?
Horizontal Partitioning

Better:

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

\[
\text{SELECT } \text{name} \\
\text{FROM } \text{Customers} \\
\text{WHERE } \text{city} = '\text{Amherst}'
\]

\[
\text{SELECT } \text{name} \\
\text{FROM } \text{CustomersInAmherst}
\]
Horizontal Partitioning

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

- Distributed databases

- Data integration
CREATE VIEW PublicCustomers

SELECT Name, Address

FROM 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
    SELECT *
    FROM Customers
    WHERE Balance < 0

<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>

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?
Propagating annotations

The annotation \( p \cdot r \) means joint use of the data annotated by \( p \) and the data annotated by \( r \)
Propagating annotations (2)

The notation \( p + r \) means alternative use of the data annotated by \( p \) and the data annotated by \( r \).
Propagating Annotations (3)

\[ R \]

\[ \begin{array}{ccc}
A & B & C \\
... \\
a & b & c_1 \\
... \\
a & b & c_2 \\
... \\
a & b & c_3 \\
\end{array} \]

\[ \pi_{AB} R \]

\[ \begin{array}{cc}
A & B \\
... \\
a & b \\
... \\
\end{array} \]

\[ p + r + s \]

+ denotes alternative use of data

Project
The view deletion problem

- $D$ a database instance and $V = Q(D)$ a view defined over $D$.
  - Find a set of tuples $\Delta D$ to remove from $D$ so that a specific tuple $t$ is removed from the view

- Minimize the number of side-effects in the view
  - View side-effect problem
    - Hard: queries with joins and projection or union
    - PTIME: the rest

- Minimize the number of tuples deleted from $D$
  - Source side-effect problem
    - Same dichotomy

[BunemanKhannaTan. PODS02]
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>