Lecture 04: Views and Constraints

October 6, 2010

Guest lecturer: Dan Suciu
Views

Views are relations, except that they may not be physically stored.

For presenting different information to different users

Employee(ssn, name, department, project, salary)

CREATE VIEW Developers AS
    SELECT name, project
    FROM Employee
    WHERE department = 'Development'

Payroll has access to Employee, others only to Developers
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

CustomerPrice(customer, price)  “virtual table”
SELECT u.customer, v.store
FROM CustomerPrice u, Purchase v
WHERE u.customer = v.customer AND u.price > 100
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 (read book)
Queries Over Views:
Query Modification

**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
```
Queries Over Views: Query Modification

Modified query:

```sql
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
```
Another Example

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

\[
\begin{align*}
\text{SELECT DISTINCT} & \quad u.\text{customer}, v.\text{store} \\
\text{FROM} & \quad \text{CustomerPrice } u, \text{ Purchase } v \\
\text{WHERE} & \quad u.\text{customer} = v.\text{customer} \quad \text{AND} \\
& \quad u.\text{price} > 100
\end{align*}
\]

\[
\begin{align*}
\text{SELECT DISTINCT} & \quad x.\text{customer}, v.\text{store} \\
\text{FROM} & \quad \text{Purchase } x, \text{ Product } y, \text{ Purchase } v, \\
\text{WHERE} & \quad x.\text{customer} = v.\text{customer} \quad \text{AND} \\
& \quad y.\text{price} > 100 \quad \text{AND} \\
& \quad x.\text{product} = y.\text{pname}
\end{align*}
\]
Applications of Virtual Views

• Physical data independence. E.g.
  – Vertical data partitioning
  – Horizontal data partitioning

• Security
  – The view reveals only what the users are allowed to know
## 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>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>
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

When do we use vertical partitioning?
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>Huston</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>

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>

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
   CustomersInHuston
   UNION ALL
   CustomersInSeattle
   UNION ALL
   ...

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 CustomersInHuston
 WHERE city = 'Huston')
UNION ALL
(SELECT * FROM CustomersInSeattle
 WHERE city = 'Seattle')
UNION ALL
...
```
Horizontal Partitioning

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

↓

```sql
SELECT name
FROM CustomersInSeattle
```
Horizontal Partitioning

Applications:

• Optimizations:
  – E.g. archived applications and active applications
• Distributed databases
• Data integration
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>

CREATE VIEW PublicCustomers
SELECT Name, Address
FROM Customers
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>

CREATE VIEW BadCreditCustomers
SELECT *
FROM Customers
WHERE Balance < 0

John is not allowed to see >0 balances
Constraints in SQL

Constraints in SQL:
- Keys, foreign keys
- Attribute-level constraints
- Tuple-level constraints
- Global constraints: assertions

The more complex the constraint, the harder it is to check and to enforce
Keys

CREATE TABLE Product (  
    name CHAR(30) PRIMARY KEY,  
    category VARCHAR(20))

OR:

Product(name, category)

CREATE TABLE Product (  
    name CHAR(30),  
    category VARCHAR(20)  
    PRIMARY KEY (name))
Keys with Multiple Attributes

```
CREATE TABLE Product (  
    name CHAR(30),  
    category VARCHAR(20),  
    price INT,  
    PRIMARY KEY (name, category))
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>10</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>20</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Photo</td>
<td>30</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>40</td>
</tr>
</tbody>
</table>

Product(name, category, price)
Other Keys

CREATE TABLE Product (
  productID CHAR(10),
  name CHAR(30),
  category VARCHAR(20),
  price INT,
  PRIMARY KEY (productID),
  UNIQUE (name, category))

There is at most one PRIMARY KEY; there can be many UNIQUE
Foreign Key Constraints

CREATE TABLE Purchase (  
    prodName CHAR(30)  
    REFERENCES Product(name),  
    date DATETIME)

prodName is a foreign key to Product(name)  
name must be a key in Product
<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ProdName</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Wiz</td>
</tr>
</tbody>
</table>
Foreign Key Constraints

```
CREATE TABLE Purchase (  
    prodName CHAR(30),  
    category VARCHAR(20),  
    date DATETIME,  
    FOREIGN KEY (prodName, category) REFERENCES Product(name, category)
)
```

• (name, category) must be a PRIMARY KEY
What happens during updates?

Types of updates:
• In Purchase: insert/update
• In Product: delete/update
What happens during updates?

- SQL has three policies for maintaining referential integrity:
  - **Reject** violating modifications (default)
  - **Cascade**: after a delete/update do a delete/update
  - **Set-null** set foreign-key field to NULL

READING ASSIGNMENT: 7.1.5, 7.1.6
Constraints on Attributes and Tuples

• Constraints on attributes:
  NOT NULL -- obvious meaning...
  CHECK condition -- any condition!

• Constraints on tuples
  CHECK condition
CREATE TABLE Purchase (  
    prodName CHAR(30)  
    CHECK (prodName IN  
       SELECT Product.name  
       FROM Product),  
    date DATETIME NOT NULL)
General Assertions

CREATE ASSERTION myAssert CHECK NOT EXISTS(
    SELECT Product.name
    FROM Product, Purchase
    WHERE Product.name = Purchase.prodName
    GROUP BY Product.name
    HAVING count(*) > 200)