Database Design and Implementation

CS 645

Views and constraints
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
```
Queries over views: query modification

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

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
Modified and unnested query:

```sql
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 ... Local DB

integrated data

v

integrated data

v1

vk

global as view

local as view
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]
Suppose you have these views:

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

Can you rewrite this query in terms of the views?

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

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) :- 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)
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?

\[ Q(x,y) :\text{edge}(x,z_1), \text{edge}(z_1,z_2), \]
\[ \text{edge}(z_2,z_3), \text{edge}(z_3,y) \]
Query rewriting using views

Suppose you have these views:

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

Can you rewrite this?

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

No! Maximally contained rewrite is:

Q(x,y) :- V1(x,z1), V2(z1,z2), V1(z2,z3), V2(z3,y)
Q(x,y) :- V2(x,z1), V2(z1,z2), V2(z2,z3), V2(z3,y)
Q(x,y) :- V2(x,z1), V1(z1,z2), V1(z2,z3), V2(z3,y)
Q(x,y) :- V2(x,z1), V1(z1,z2), V1(z2,z3), V2(z3,y)

...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>
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:

```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
Views and security

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
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>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>

CREATE VIEW BadCreditCustomers
    SELECT *
    FROM Customers
    WHERE Balance < 0

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?
- Delete a tuple from a view?
Propagating annotations

Join (on B)

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

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

\[ \pi_{AB} R \]

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

\[ \begin{array}{cc}
    \text{Project} \\
    p \\
    r \\
    s \\
\end{array} \]

\[ \begin{array}{cc}
    \ldots \\
    a & b \\
    \ldots \\
\end{array} \]

\[ p + r + s \]

+ denotes alternative use of data
The view deletion problem

\[ D \text{ a database instance and } V = Q(D) \text{ a view defined over } D. \]

\[ \text{Find a set of tuples } \Delta D \text{ to remove from } D \text{ so that a specific tuple } t \text{ is removed from the view} \]

\[ \text{Minimize the number of side-effects in the view} \]
\[ \text{View side-effect problem} \]
\[ \text{Hard: queries with joins and projection or union} \]
\[ \text{PTIME: the rest} \]

\[ \text{Minimize the number of tuples deleted from } D \]
\[ \text{Source side-effect problem} \]
\[ \text{Same dichotomy} \]

[Buneman, Khanna, Tan. PODS 2002]
### 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>
Constraints

Constraint: a property that we want our data to satisfy

Enforce by taking actions:
- Forbid an update
- Or perform compensating updates

Two approaches:
- Declarative constraints
- Triggers
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 enforce
CREATE TABLE Product (
    name CHAR(30) PRIMARY KEY,
    category VARCHAR(20)
)
Keys with multiple attributes

Product(name, category, price)

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>
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
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 (primary key or unique)  

may write just Product

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>

Purchase

<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>
<tr>
<td>CREATE TABLE Purchase (</td>
<td></td>
</tr>
<tr>
<td>prodName CHAR(30),</td>
<td></td>
</tr>
<tr>
<td>category VARCHAR(20),</td>
<td></td>
</tr>
<tr>
<td>date DATETIME,</td>
<td></td>
</tr>
<tr>
<td>store VARCHAR(30),</td>
<td></td>
</tr>
<tr>
<td>FOREIGN KEY (prodName, category)</td>
<td></td>
</tr>
<tr>
<td>REFERENCES Product(name, category)</td>
<td></td>
</tr>
</tbody>
</table>

Product(name, category, price)  
Purchase(proName, category, date |
What happens during updates?

Types of updates:
- In Purchase: insert/update
- In Product: delete/update

<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>
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

```sql
CREATE TABLE Purchase (
    prodName CHAR(30) 
    REFERENCES Product(name) 
    ON DELETE SET NULL
    ON UPDATE CASCADE
)
```
Constraints on attributes and tuples

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

```
CREATE TABLE Purchase (...  
  store VARCHAR(30) NOT NULL, ...)  
```

```
CREATE TABLE Product(...  
  price INT CHECK(price>0 and price <999))  
```

**Constraints on tuples**
- **CHECK** condition

```
... CHECK(price*quantity < 1000)...
```
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)

Often not implemented in DBMSs!
Semantic optimization with constraints

Product(name, price)
Purchase(buyer, seller, prodName, store)

SELECT Purchase.store
FROM Product, Purchase
WHERE Product.name = Purchase.prodName
Semantic optimization with constraints

Product(name, price)
Purchase(buyer, seller, prodName, store)

SELECT Purchase.store
FROM Product, Purchase
WHERE Product.name = Purchase.prodName

Yes, if Purchase.prodName is a foreign key, and not null