Lecture 04: SQL

Wednesday, January 11, 2006

Outline

• Two Examples
• Nulls (6.1.6)
• Outer joins (6.3.8)
• Database Modifications (6.5)
Two Examples

Store(sid, sname)
Product(pid, pname, price, sid)

Find all stores that sell *only* products with price > 100

same as:

Find all stores s.t. all their products have price > 100)

```
SELECT Store.name
FROM Store, Product
WHERE Store.sid = Product.sid
GROUP BY Store.sid, Store.name
HAVING 100 < min(Product.price)
```

```
SELECT Store.name
FROM Store
WHERE 100 < ALL (SELECT Product.price
                FROM product
                WHERE Store.sid = Product.sid)
```

Almost equivalent…

```
SELECT Store.name
FROM Store
WHERE
  100 < ALL (SELECT Product.price
                FROM product
                WHERE Store.sid = Product.sid)
```

```
SELECT Store.name
FROM Store
WHERE Store.sid NOT IN (SELECT Product.sid
                        FROM Product
                        WHERE Product.price <= 100)
```

Almost equivalent…

```
SELECT Store.name
FROM Store
WHERE Store.sid NOT IN (SELECT Product.sid
                        FROM Product
                        WHERE Product.price <= 100)
```
Two Examples

Store\((sid, sname)\)
Product\((pid, pname, price, sid)\)

For each store,
find its most expensive product

```
Two Examples

This is easy but doesn’t do what we want:

\[
\text{SELECT } \text{Store.sname, max(Product.price)} \\
\text{FROM } \text{Store, Product} \\
\text{WHERE } \text{Store.sid = Product.sid} \\
\text{GROUP BY } \text{Store.sid, Store.sname}
\]

Better:

\[
\text{SELECT } \text{Store.sname, x.pname} \\
\text{FROM } \text{Store, Product x} \\
\text{WHERE } \text{Store.sid = x.sid and} \\
\text{x.price }\geq \text{ ALL (SELECT y.price} \\
\text{FROM Product y} \\
\text{WHERE Store.sid = y.sid)}
\]

But may return multiple product names per store
Two Examples

Finally, choose some pid arbitrarily, if there are many with highest price:

```sql
SELECT Store.sname, max(x.pname)
FROM Store, Product x
WHERE Store.sid = x.sid and
  x.price >=
    ALL (SELECT y.price
         FROM Product y
         WHERE Store.sid = y.sid)
GROUP BY Store.sname
```

NULLS in SQL

• Whenever we don’t have a value, we can put a NULL
• Can mean many things:
  – Value does not exists
  – Value exists but is unknown
  – Value not applicable
  – Etc.
• The schema specifies for each attribute if can be null (nullable attribute) or not
• How does SQL cope with tables that have NULLs?
Null Values

• If \( x = \text{NULL} \) then \( 4 \times (3-x)/7 \) is still \( \text{NULL} \)

• If \( x = \text{NULL} \) then \( x = “Joe” \) is UNKNOWN

• In SQL there are three boolean values:
  
  \[
  \begin{align*}
  \text{FALSE} & = 0 \\
  \text{UNKNOWN} & = 0.5 \\
  \text{TRUE} & = 1
  \end{align*}
  \]

Null Values

• \( C1 \text{ AND } C2 = \min(C1, C2) \)

• \( C1 \text{ OR } C2 = \max(C1, C2) \)

• \( \text{NOT } C1 = 1 - C1 \)

Rule in SQL: include only tuples that yield TRUE

E.g.
- age=20
- height=NULL
- weight=200

SELECT * FROM Person WHERE (age < 25) AND (height > 6 OR weight > 190)
Null Values

Unexpected behavior:

```sql
SELECT *
FROM Person
WHERE age < 25 OR age >= 25
```

Some Persons are not included!

Null Values

Can test for NULL explicitly:
- x IS NULL
- x IS NOT NULL

```sql
SELECT *
FROM Person
WHERE age < 25 OR age >= 25 OR age IS NULL
```

Now it includes all Persons
Outerjoins

Explicit joins in SQL = “inner joins”:

\[
\text{Product(name, category)}
\]
\[
\text{Purchase(prodName, store)}
\]

\[
\begin{align*}
\text{SELECT} & \quad \text{Product.name, Purchase.store} \\
\text{FROM} & \quad \text{Product JOIN Purchase ON} \\
& \quad \text{Product.name = Purchase.prodName}
\end{align*}
\]

Same as:

\[
\begin{align*}
\text{SELECT} & \quad \text{Product.name, Purchase.store} \\
\text{FROM} & \quad \text{Product, Purchase} \\
\text{WHERE} & \quad \text{Product.name = Purchase.prodName}
\end{align*}
\]

But Products that never sold will be lost!

Outerjoins

Left outer joins in SQL:

\[
\text{Product(name, category)}
\]
\[
\text{Purchase(prodName, store)}
\]

\[
\begin{align*}
\text{SELECT} & \quad \text{Product.name, Purchase.store} \\
\text{FROM} & \quad \text{Product LEFT OUTER JOIN Purchase ON} \\
& \quad \text{Product.name = Purchase.prodName}
\end{align*}
\]
Application

Compute, for each product, the total number of sales in ‘September’
Product(name, category)
Purchase(prodName, month, store)

```
SELECT Product.name, count(*)
FROM   Product, Purchase
WHERE  Product.name = Purchase.prodName
       and Purchase.month = ‘September’
GROUP BY Product.name
```

What’s wrong?
Application

Compute, for each product, the total number of sales in ‘September’

Product(name, category)
Purchase(prodName, month, store)

```
SELECT Product.name, count(*)
FROM Product LEFT OUTER JOIN Purchase ON
    Product.name = Purchase.prodName
    and Purchase.month = ‘September’
GROUP BY Product.name
```

Now we also get the products who sold in 0 quantity

Outer Joins

- Left outer join:
  - Include the left tuple even if there’s no match
- Right outer join:
  - Include the right tuple even if there’s no match
- Full outer join:
  - Include the both left and right tuples even if there’s no match
Modifying the Database

Three kinds of modifications
• Insertions
• Deletions
• Updates

Sometimes they are all called “updates”

Insertions

General form:

```
INSERT INTO R(A1,…., An) VALUES (v1,…., vn)
```

Example: Insert a new purchase to the database:

```
INSERT INTO Purchase(buyer, seller, product, store)
VALUES ('Joe', 'Fred', 'wakeup-clock-espresso-machine', 'The Sharper Image')
```

Missing attribute → NULL.
May drop attribute names if give them in order.
Insertions

```
INSERT INTO PRODUCT(name)
SELECT DISTINCT Purchase.product
FROM Purchase
WHERE Purchase.date > "10/26/01"
```

The query replaces the VALUES keyword. Here we insert many tuples into PRODUCT.

Insertion: an Example

Product(name, listPrice, category)

Purchase(prodName, buyerName, price)

prodName is foreign key in Product.name

Suppose database got corrupted and we need to fix it:

<table>
<thead>
<tr>
<th>Product</th>
<th>Purchase</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>listPrice</td>
</tr>
<tr>
<td>gizmo</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>prodName</th>
<th>buyerName</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>camera</td>
<td>John</td>
<td>200</td>
</tr>
<tr>
<td>gizmo</td>
<td>Smith</td>
<td>80</td>
</tr>
<tr>
<td>camera</td>
<td>Smith</td>
<td>225</td>
</tr>
</tbody>
</table>

Task: insert in Product all prodNames from Purchase.
Insertion: an Example

\[
\text{INSERT INTO Product(name)}
\]

\[
\text{SELECT DISTINCT prodName}
\]

\[
\text{FROM Purchase}
\]

\[
\text{WHERE prodName NOT IN (SELECT name FROM Product)}
\]

<table>
<thead>
<tr>
<th>name</th>
<th>listPrice</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>gizmo</td>
<td>100</td>
<td>Gadgets</td>
</tr>
<tr>
<td>camera</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Insertion: an Example

\[
\text{INSERT INTO Product(name, listPrice)}
\]

\[
\text{SELECT DISTINCT prodName, price}
\]

\[
\text{FROM Purchase}
\]

\[
\text{WHERE prodName NOT IN (SELECT name FROM Product)}
\]

<table>
<thead>
<tr>
<th>name</th>
<th>listPrice</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>gizmo</td>
<td>100</td>
<td>Gadgets</td>
</tr>
<tr>
<td>camera</td>
<td>200</td>
<td>-</td>
</tr>
<tr>
<td>camera??</td>
<td>225 ??</td>
<td>-</td>
</tr>
</tbody>
</table>

\[\text{\textup{← Depends on the implementation}}\]
Deletions

Example:

```
DELETE FROM PURCHASE
WHERE seller = 'Joe' AND
     product = 'Brooklyn Bridge'
```

Factoid about SQL: there is no way to delete only a single occurrence of a tuple that appears twice in a relation.

Updates

Example:

```
UPDATE PRODUCT
SET   price = price/2
WHERE Product.name IN
      (SELECT product
       FROM   Purchase
       WHERE  Date = 'Oct, 25, 1999');
```