Lecture 03: SQL
Friday, October 4, 2002

Outline

- Unions, intersections, differences (6.2.5, 6.4.2)
- Subqueries (6.3)
- Aggregations (6.4.3 – 6.4.6)

Hint for reading the textbook: read the entire chapter 6!

Reading assignment from “SQL for Nerds”: chapter 4, “More complex queries” (you will find it very useful for subqueries)

First Unintuitive SQLism

```
SELECT DISTINCT R.A
FROM R, S, T
WHERE R.A = S.A OR R.A = T.A
```

Looking for \( R \cap (S \cup T) \)

But what happens if T is empty?

Renaming Columns

```
SELECT P.name AS prodName, Price AS askPrice
FROM Product
WHERE Price > 100
```

Query with renaming

Union, Intersection, Difference

```
(SELECT name
FROM Person
WHERE City = "Seattle")
UNION
(SELECT name
FROM Person, Purchase
WHERE buyer = name AND store = "The Ben")
```

Similarly, you can use INTERSECT and EXCEPT. You must have the same attribute names (otherwise: rename).
Conserving Duplicates

\[(\text{SELECT name FROM Person WHERE City = "Seattle"}) \quad \text{UNION ALL} \quad (\text{SELECT name FROM Person, Purchase WHERE buyer = name AND store = "The Bon"})\]

Subqueries

A subquery producing a single value:

\[
\text{SELECT Purchase.product FROM Purchase WHERE buyer = (SELECT name FROM Person WHERE ssn = '123456789')}\]

In this case, the subquery returns one value.

If it returns more, it’s a run-time error.

Subqueries Returning Relations

Find companies who manufacture products bought by Joe Blow.

\[
\text{SELECT Company.name FROM Company, Product, Purchase WHERE Company.name = Product.maker AND Product.name = Purchase.product AND Purchase.buyer = 'Joe Blow'}\]

This is equivalent to the previous one when the ssn is a key and '123456789' exists in the database; otherwise they are different.

Subqueries Returning Relations

Equivalent to:

\[
\text{SELECT Company.name FROM Company, Product, Purchase WHERE Company.name = Product.maker AND Product.name = Purchase.product AND Purchase.buyer = 'Joe Blow'}\]

Is this query equivalent to the previous one?

Beware of duplicates!

Removing Duplicates

\[
\text{SELECT DISTINCT Company.name FROM Company, Product, Purchase WHERE Company.name = Product.maker AND Product.name = Purchase.product AND Purchase.buyer = 'Joe Blow'}\]

← Multiple copies

\[
\text{SELECT Company.name FROM Company, Product, Purchase WHERE Company.name = Product.maker AND Product.name = Purchase.product AND Purchase.buyer = 'Joe Blow'}\]

← Single copies
Removing Duplicates

```
SELECT DISTINCT Company.name
FROM Company, Product
WHERE Company.name = Product.maker
AND Product.name IN
(SELECT Purchase.product
FROM Purchase
WHERE Purchase.buyer = 'Joe Blow');
```

Now they are equivalent

```
SELECT DISTINCT Company.name
FROM Company, Product
WHERE Company.name = Product.maker
AND Product.name = Purchase.product
AND Purchase.buyer = 'Joe Blow';
```

Subqueries Returning Relations

You can also use:

```
s > ALL R
s > ANY R
EXISTS R
```

```
Product (pname, price, category, maker)
```

Find products that are more expensive than all those produced
By “Gizmo-Works”

```
SELECT name
FROM Product
WHERE price > ALL (SELECT price
FROM Purchase
WHERE maker = 'Gizmo-Works');
```

```
SELECT DISTINCT Company.name
FROM Company, Product, Purchase
WHERE Company.name = Product.maker
AND (Product.name, price) IN
(SELECT Purchase.product, Purchase.price
FROM Purchase
WHERE Purchase.buyer = 'Joe Blow');
```

Question for Database Fans and their Friends

- Can we express this query as a single SELECT-FROM-WHERE query, without subqueries?

- Hint: show that all SFW queries are monotone
  (figure out what this means). A query with ALL is not monotone

```
SELECT DISTINCT title
FROM Movie AS x
WHERE year <>
(SELECT year
FROM Movie
WHERE title = x.title);
```

May not work in SQL server...

```
SELECT DISTINCT title
FROM Movie AS x
WHERE year <>
(SELECT year
FROM Movie
WHERE title = x.title);
```

Correlated Queries

```
SELECT DISTINCT title
FROM Movie AS x
WHERE year <>
ANY
(SELECT year
FROM Movie
WHERE title = x.title);
```

Note (1) scope of variables (2) this can still be expressed as single SFW

```
SELECT DISTINCT title
FROM Movie AS x
WHERE year <>
ANY
(SELECT year
FROM Movie
WHERE title = x.title);
```

Complex Correlated Query

```
SELECT DISTINCT pname, maker
FROM Product AS x
WHERE price > ALL (SELECT price
FROM Product AS y
WHERE x.maker = y.maker AND y.year < 1972);
```

Powerful, but much harder to optimize!
Aggregation

SELECT Avg(price) FROM Product WHERE maker="Toyota"

SQL supports several aggregation operations:

SUM, MIN, MAX, AVG, COUNT

Aggregation: Count

SELECT Count(*) FROM Product WHERE year > 1995

Except COUNT, all aggregations apply to a single attribute

Aggregation: Count

COUNT applies to duplicates, unless otherwise stated:

SELECT Count(category) same as Count(*) FROM Product WHERE year > 1995

Better:

SELECT Count(DISTINCT category) FROM Product WHERE year > 1995

Simple Aggregation

Purchase(product, date, price, quantity)

Example 1: find total sales for the entire database

SELECT Sum(price * quantity) FROM Purchase

Example 1': find total sales of bagels

SELECT Sum(price * quantity) FROM Purchase WHERE product = 'bagel'

Grouping and Aggregation

Usually, we want aggregations on certain parts of the relation.

Purchase(product, date, price, quantity)

Example 2: find total sales after 9/1 per product.

SELECT product, Sum(price*quantity) AS TotalSales FROM Purchase WHERE date > "9/1" GROUPBY product

Let’s see what this means…

Simple Aggregations

<table>
<thead>
<tr>
<th>Product</th>
<th>Date</th>
<th>Price</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>10/21</td>
<td>0.85</td>
<td>15</td>
</tr>
<tr>
<td>Banana</td>
<td>10/22</td>
<td>0.52</td>
<td>7</td>
</tr>
<tr>
<td>Banana</td>
<td>10/19</td>
<td>0.52</td>
<td>17</td>
</tr>
<tr>
<td>Bagel</td>
<td>10/20</td>
<td>0.85</td>
<td>20</td>
</tr>
</tbody>
</table>
Grouping and Aggregation

1. Compute the FROM and WHERE clauses.
2. Group by the attributes in the GROUP BY.
3. Select one tuple for every group (and apply aggregation).

SELECT can have (1) grouped attributes or (2) aggregates.

Then, aggregate

<table>
<thead>
<tr>
<th>Product</th>
<th>TotalSales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>$29.75</td>
</tr>
<tr>
<td>Banana</td>
<td>$12.48</td>
</tr>
</tbody>
</table>

SELECT product, Sum(price * quantity) AS TotalSales
FROM Purchase
WHERE date > "9/1"
GROUP BY product

GROUP BY v.s. Nested Queries

SELECT product, Sum(price * quantity) AS TotalSales
FROM Purchase
WHERE date > "9/1"
GROUP BY product

SELECT product, (SELECT Sum(y.price * y.quantity)
FROM Purchase y
WHERE x.product = y.product
AND y.date > "9/1")
AS TotalSales
FROM Purchase x
WHERE x.date > "9/1"

Another Example

<table>
<thead>
<tr>
<th>Product</th>
<th>SumSales</th>
<th>MaxQuantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banana</td>
<td>$12.48</td>
<td>17</td>
</tr>
<tr>
<td>Bagel</td>
<td>$29.75</td>
<td>20</td>
</tr>
</tbody>
</table>

For every product, what is the total sales and max quantity sold?

SELECT product, Sum(price * quantity) AS SumSales
      Max(quantity) AS MaxQuantity
FROM Purchase
GROUP BY product

HAVING Clause

Same query, except that we consider only products that had at least 100 buyers.

SELECT product, Sum(price * quantity)
FROM Purchase
WHERE date > "9/1"
GROUP BY product
HAVING Sum(quantity) > 100

HAVING clause contains conditions on aggregates.
General form of Grouping and Aggregation

```sql
SELECT S
FROM R_1, ..., R_n
WHERE C_1
GROUP BY a_1, ..., a_k
HAVING C_2
```

- $S$ may contain attributes $a_1, ..., a_k$ and/or aggregates but no other attributes.
- $C_1$ is any condition on the attributes in $R_1, ..., R_n$.
- $C_2$ is any condition on aggregate expressions.

Why?

Evaluation steps:
1. Compute the FROM-WHERE part, obtain a table with all attributes in $R_1, ..., R_n$.
2. Group by the attributes $a_1, ..., a_k$.
3. Compute the aggregates in $C_2$ and keep only groups satisfying $C_2$.
4. Compute aggregates in $S$ and return the result.

Aggregation

- Author(login, name)
- Document(url, title)
- Wrote(login, url)
- Mentions(url, word)

- Find all authors who wrote at least 10 documents:
  - Attempt 1: with nested queries
    ```sql
    SELECT DISTINCT Author.name
    FROM Author
    WHERE Author.login=Wrote.login
      AND Wrote.url=MENTIONS.url
    GROUP BY Author.name
    HAVING count(DISTINCT MENTIONS.word) > 10000
    ```
    This is SQL by a novice.

  - Attempt 2: SQL style (with GROUP BY)
    ```sql
    SELECT Author.name
    FROM Author, Wrote, Mentions
    WHERE Author.login=Wrote.login AND Wrote.url=Mentions.url
    GROUP BY Author.name
    HAVING count(DISTINCT Mentions.word) > 10000
    ```
    No need for DISTINCT; automatically from GROUP BY.

- Find all authors who have a vocabulary over 10000 words:

Look carefully at the last two queries: you may be tempted to write them as a nested queries, but in SQL we write them best with GROUP BY.