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

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT   Pname AS prodName, Price AS askPrice
FROM     Product
WHERE    Price > 100
```

Query with renaming

prodName	askPrice
SingleTouch	$149.99
MultiTouch	$203.99

Union, Intersection, Difference

```
(SELECT  name
 FROM     Person
 WHERE    City="Seattle")

   UNION

(SELECT  name
 FROM     Person, Purchase
 WHERE   buyer=name AND store="The Bon")
```

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


```
(SELECT DISTINCT R.A
FROM R)
INTERSECT ( (SELECT S.A FROM S)
                  UNION
                  (SELECT T.A FROM T))
```

Conserving Duplicates

```
(SELECT  name
 FROM     Person
 WHERE    City="Seattle")

   UNION  ALL

(SELECT  name
 FROM     Person, Purchase
 WHERE   buyer=name AND store="The Bon")
```

7

Subqueries

A subquery producing a single value:

```
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.

8

Can say the same thing without a subquery:

```
SELECT Purchase.product
FROM    Purchase, Person
WHERE  buyer = name AND ssn = '123456789'
```

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

9

Subqueries Returning Relations

Find companies who manufacture products bought by Joe Blow.

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

Here the subquery returns a set of values: no more runtime errors.

10

Subqueries Returning Relations

Equivalent to:

```
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 !

11

Removing Duplicates

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

←Multiple copies

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

← Single copies

12

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')
```

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

Now they are equivalent

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')
```


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

Conditions on Tuples

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

May not work in SQL server...

Correlated Queries

Movie (title, year, director, length)
Find movies whose title appears more than once.

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

correlation

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

Complex Correlated Query

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

- Find products (and their manufacturers) that are more expensive than all products made by the same manufacturer before 1972

```
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

19

Aggregation: Count

```
SELECT  Count(*)
FROM    Product
WHERE   year > 1995
```

Except COUNT, all aggregations apply to a single attribute

20

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
```

21

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'
```

22

Simple Aggregations

Purchase

Product	Date	Price	Quantity
Bagel	10/21	0.85	15
Banana	10/22	0.52	7
Banana	10/19	0.52	17
Bagel	10/20	0.85	20

23

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…

24

Grouping and Aggregation

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

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

25

First compute the FROM-WHERE clauses (date > "9/1") then GROUP BY product:

Product	Date	Price	Quantity
Banana	10/19	0.52	17
Banana	10/22	0.52	7
Bagel	10/20	0.85	20
Bagel	10/21	0.85	15

26

Then, aggregate

Product	TotalSales
Bagel	$29.75
Banana	$12.48

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

27

GROUP BY v.s. Nested Quereis

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

```
SELECT DISTINCT  x.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

Product	SumSales	MaxQuantity
Banana	$12.48	17
Bagel	$29.75	20

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
```

29

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) > 30
```

HAVING clause contains conditions on aggregates.

30

General form of Grouping and Aggregation

SELECT S
FROM $R_1,\ldots,R_n$
WHERE C1
GROUP BY $a_1,\ldots,a_k$
HAVING C2

Why ?

S = may contain attributes $a_1,\ldots,a_k$ and/or any aggregates but NO OTHER ATTRIBUTES
C1 = is any condition on the attributes in $R_1,\ldots,R_n$
C2 = is any condition on aggregate expressions

31

General form of Grouping and Aggregation

SELECT S
FROM $R_1,\ldots,R_n$
WHERE C1
GROUP BY $a_1,\ldots,a_k$
HAVING C2

Evaluation steps:

1. Compute the FROM-WHERE part, obtain a table with all attributes in $R_1,\ldots,R_n$
2. Group by the attributes $a_1,\ldots,a_k$
3. Compute the aggregates in C2 and keep only groups satisfying C2
4. Compute aggregates in S and return the result

32

Aggregation

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

33

- Find all authors who wrote at least 10 documents:
- Attempt 1: with nested queries

This is SQL by a novice

```
SELECT DISTINCT Author.name
FROM Author
WHERE count(SELECT Wrote.url
            FROM Wrote
            WHERE Author.login=Wrote.login)
      > 10
```

34

- Find all authors who wrote at least 10 documents:
- Attempt 2: SQL style (with GROUP BY)

This is SQL by an expert

```
SELECT Author.name
FROM Author, Wrote
WHERE Author.login=Wrote.login
GROUP BY Author.name
HAVING count(wrote.url) > 10
```

No need for DISTINCT: automatically from GROUP BY

35

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

```
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
```

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

36