CIS 560 – Database System Concepts

Lecture 3

SQL

August 30, 2013

Credits for slides: Suciu, Chang, Ullman.

Announcements

- CIS MySQL accounts
- First SQL assignment to be posted tonight

Outline

Last time:
- Data in SQL
- Simple Queries in SQL (Section 6.1)
- Queries with more than one relation (Section 6.2)

Today:
- Subqueries (Section 6.3)
- Aggregations (Sections 6.4.3 - 6.4.6)

Next time:
- Nulls (Sections 6.1.6 - 6.1.7)
- Outer joins (Section 6.3.8)
- Views (Sections 8.1, 8.2, 8.3)

3

Review

Tables in SQL - terminology

Most general form of an SQL query

DISTINCT

ORDER BY

Keys and foreign keys

Joins

Tuple variables

4

Subqueries

- A *subquery* is another SQL query nested inside a larger query
- Such inner-outer queries are called *nested queries*
- A subquery may occur in:
  1. A SELECT clause
  2. A FROM clause
  3. A WHERE clause

Rule of thumb: avoid writing nested queries when possible; keep in mind that sometimes it's impossible

5

1. Subqueries in SELECT

Product (pname, price, company)
Company(cname, city)

For each product return the city where it is manufactured

```
SELECT X.pname, (SELECT Y.city
                 FROM Company Y
                 WHERE Y.cname=X.company)
FROM  Product X
```

What happens if the subquery returns more than one city ?

6

1. Subqueries in SELECT

Product (pname, price, company)
Company(cname, city)

Whenever possible, don't use a nested queries:

```
SELECT pname, (SELECT city FROM Company WHERE cname=company)
FROM  Product
```

=

```
SELECT pname, city
FROM   Product, Company
WHERE cname=company
```

We have "unnested" the query

7

2. Subqueries in FROM

Product (pname, price, company)
Company(cname, city)

Find all products whose prices is > 20 and < 30

```
SELECT X.pname
FROM (SELECT * FROM Product AS Y WHERE Y.price > 20) AS  X
WHERE X.price < 30
```

Unnest this query !

8

3. Subqueries in WHERE

Product (pname, price, company) Existential quantifiers
Company(cname, city)

Find all cities that make some products with price < 100

Using EXISTS:

```
SELECT DISTINCT  Company.city
FROM     Company
WHERE  EXISTS (SELECT *
                        FROM Product
                       WHERE company = cname and Product.price < 100)
```

9

3. Subqueries in WHERE

Product (pname, price, company) Existential quantifiers
Company(cname, city)

Find all cities that make some products with price < 100

Predicate Calculus (a.k.a. First Order Logic)

$$\{ y \mid \exists x.\ Company(x,y) \wedge (\exists z.\ \exists p.\ Product(z,p,x) \wedge p < 100) \}$$

10

3. Subqueries in WHERE

Product (pname, price, company) Existential quantifiers
Company(cname, city)

Find all cities that make some products with price < 100

Using IN

```
SELECT DISTINCT  Company.city
FROM     Company
WHERE Company.cname IN (SELECT Product.company
                        FROM Product
                        WHERE Product.price < 100)
```

11

3. Subqueries in WHERE

Product (pname, price, company) Existential quantifiers
Company(cname, city)

Find all cities that make some products with price < 100

Using ANY:

```
SELECT DISTINCT  Company.city
FROM     Company
WHERE 100 > ANY (SELECT price
                 FROM Product
                 WHERE company = cname)
```

12

3. Subqueries in WHERE

Product (pname, price, company) Existential quantifiers
Company(cname, city)

Find all cities that make some products with price < 100

Now let's unnest it:

3. Subqueries in WHERE

Product (pname, price, company) Existential quantifiers
Company(cname, city)

Find all cities that make some products with price < 100

Now let's unnest it:

```
SELECT DISTINCT  Company.cname
FROM     Company, Product
WHERE  Company.cname = Product.company and Product.price < 100
```

Existential quantifiers are easy ! ☺

3. Subqueries in WHERE

Product (pname, price, company)
Company(cname, city)

Universal quantifiers

Find all cities with companies
that make only products with price < 100

Universal quantifiers are hard ! ☹

3. Subqueries in WHERE

Product (pname, price, company)
Company(cname, city)

Universal quantifiers

Find all cities with companies
that make only products with price < 100

Predicate Calculus (a.k.a. First Order Logic)

$\{ y \mid \exists x.\ Company(x,y) \wedge (\forall z.\ \forall p.\ Product(z,p,x) \rightarrow p < 100) \}$

3. Subqueries in WHERE

De Morgan's Laws:

$\neg(A \wedge B) = \neg A \vee \neg B$
$\neg(A \vee B) = \neg A \wedge \neg B$
$\neg \forall x.\ P(x) = \exists x.\ \neg P(x)$
$\neg \exists x.\ P(x) = \forall x.\ \neg P(x)$

$\neg(A \rightarrow B) = A \wedge \neg B$

$\{ y \mid \exists x.\ Company(x,y) \wedge (\forall z.\ \forall p.\ Product(z,p,x) \rightarrow p < 100) \}$

=

$\{ y \mid \exists x.\ Company(x,y) \wedge \neg (\exists z \exists p.\ Product(z,p,x) \wedge p \geq 100) \}$

=

$\{ y \mid \exists x.\ Company(x,y)) \}$ -
$\{ y \mid \exists x.\ Company(x,y) \wedge (\exists z \exists p.\ Product(z,p,x) \wedge p \geq 100) \}$

3. Subqueries in WHERE

1. Find *the other* companies: i.e. s.t. <u>some</u> product ≥ 100

```
SELECT DISTINCT  Company.city
FROM     Company
WHERE  Company.cname IN (SELECT Product.company
                         FROM Product
                         WHERE Product.price >= 100)
```

2. Find all companies s.t. <u>all</u> their products have price < 100

```
SELECT DISTINCT  Company.city
FROM     Company
WHERE  Company.cname NOT IN (SELECT Product.company
                             FROM Product
                             WHERE Product.price >= 100)
```

3. Subqueries in WHERE

Product (pname, price, company) Universal quantifiers

Company(cname, city)

Find all cities with companies
that make only products with price < 100

Using EXISTS:

```
SELECT DISTINCT  Company.city
FROM     Company
WHERE NOT EXISTS (SELECT *
                  FROM Product
                  WHERE company = cname and Product.price >= 100)
```

19

3. Subqueries in WHERE

Product (pname, price, company) Universal quantifiers

Company(cname, city)

Find all cities with companies
that make only products with price < 100

Using ALL:

```
SELECT DISTINCT  Company.city
FROM     Company
WHERE 100 > ALL  (SELECT price
                  FROM Product
                  WHERE company = cname)
```

20

Question for Database Fans and their Friends

- Can we unnest the *universal quantifier* query ?

21

Monotone Queries

- A query Q is monotone if:
  - Whenever we add tuples to one or more of the tables…
  - … the answer to the query cannot contain fewer tuples
- **Fact**: all unnested queries are monotone
  - Proof: using the "nested for loops" semantics
- **Fact**: A query with universal quantifier is not monotone
- **Consequence**: we cannot unnest a query with a universal quantifier

22

Queries that must be nested

- Queries with universal quantifiers or with negation
- The drinkers-bars-beers example next
- This is a famous example by Ullman

23

The drinkers-bars-beers example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Challenge: write these in SQL

Find drinkers that frequent some bar that serves some beer they like.

x: $\exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$

Find drinkers that frequent only bars that serve some beer they like.

x: $\forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \wedge \text{Likes}(x,z))$

Find drinkers that frequent some bar that serves only beers they like.

x: $\exists y.\ \text{Frequents}(x, y) \wedge \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$

Find drinkers that frequent only bars that serve only beer they like.

x: $\forall y.\ \text{Frequents}(x, y) \Rightarrow \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$

24

Aggregation

```
SELECT  avg(price)
FROM    Product
WHERE   maker='Toyota'
```

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

SQL supports several aggregation operations:

sum, count, min, max, avg

Except count, all aggregations apply to a single attribute

25

Aggregation: Count

COUNT applies to duplicates, unless otherwise stated:

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

same as Count(*)

We probably want:

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

26

More Examples

Purchase(product, date, price, quantity)

```
SELECT  Sum(price * quantity)
FROM    Purchase
```

What do they mean ?

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

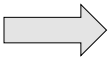
27

Simple Aggregations

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	1.50	20
Banana	0.5	50
Banana	2	10
Banana	4	10

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



90 (= 60+30)

28

Grouping and Aggregation

Purchase(product, price, quantity)

Find total quantity for all sales.

Find total quantity for all sales over $1.

Find total quantities for all sales over $1, by product.

29

Grouping and Aggregation

Purchase(product, price, quantity)

Find total quantities for all sales over $1, by product.

```
SELECT    product, Sum(quantity) AS TotalSales
FROM      Purchase
WHERE     price > 1
GROUP BY  product
```

Let's see what this means…

30

Grouping and Aggregation

1. Compute the FROM and WHERE clauses.
2. Group by the attributes in the GROUP BY
3. Compute the SELECT clause: grouped attributes and aggregates.

31

1&2. FROM-WHERE-GROUPBY

Product	Price	Quantity
Bagel	3	20
Bagel	1.50	20
Banana	~~0.5~~	~~50~~
Banana	2	10
Banana	4	10

32

3. SELECT

Product	Price	Quantity
Bagel	3	20
Bagel	1.50	20
Banana	~~0.5~~	~~50~~
Banana	2	10
Banana	4	10

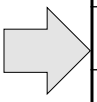
???

```
SELECT    product, Sum(quantity) AS TotalSales
FROM      Purchase
WHERE     price > 1
GROUP BY  product
```

33

3. SELECT

Product	Price	Quantity
Bagel	3	20
Bagel	1.50	20
Banana	~~0.5~~	~~50~~
Banana	2	10
Banana	4	10



Product	TotalSales
Bagel	40
Banana	20

```
SELECT    product, Sum(quantity) AS TotalSales
FROM      Purchase
WHERE     price > 1
GROUP BY  product
```

34

GROUP BY vs. Nested Queries

Find total quantities for all sales over $1, by product.

```
SELECT    product, Sum(quantity) AS TotalSales
FROM      Purchase
WHERE     price > 1
GROUP BY  product
```

```
SELECT DISTINCT  x.product, (SELECT Sum(y.quantity)
                             FROM    Purchase y
                             WHERE x.product = y.product
                                   AND price > 1)
                            AS TotalSales
FROM      Purchase x
WHERE     price > 1
```

Why twice ?

35

Another Example

What does it mean ?

```
SELECT    product,
          sum(quantity) AS SumSales
          max(price) AS MaxQuantity
FROM      Purchase
GROUP BY  product
```

36

HAVING Clause

Same query, except that we consider only products that had the overall quantity sold at least 30.

```
SELECT   product, Sum(quantity)
FROM     Purchase
WHERE    price > 1
GROUP BY product
HAVING   Sum(quantity) > 30
```

HAVING clause contains conditions on aggregates.

37

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

38

General form of Grouping and Aggregation

```
SELECT   S
FROM     R1,…,Rn
WHERE    C1
GROUP BY a1,…,ak
HAVING   C2
```

Evaluation steps:

1. Evaluate FROM-WHERE, apply condition C1
2. Group by the attributes $a_1,\ldots,a_k$
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result

39