Introduction to Database Systems
CSE 414

Lecture 6: SQL Subqueries

Announcements

- HW2 and WQ2 released
  - Both due next Tuesday

- Please fill in the Azure questionnaire by tonight!
  - See HW2 writeup for details

Simple Aggregations

Five basic aggregate operations in SQL

```
select count(*) from Purchase
select sum(quantity) from Purchase
select avg(price) from Purchase
select max(quantity) from Purchase
select min(quantity) from Purchase
```

Except `count`, all aggregations apply to a single attribute

Everything in SELECT must be
either a GROUP-BY attribute, or an aggregate

Need to be Careful…

```
SELECT product,
       max(quantity)
FROM   Purchase
GROUP BY product
```

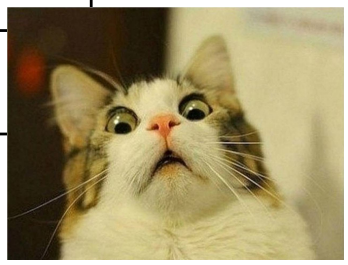
```
SELECT   product, quantity
FROM     Purchase
GROUP BY product
-- what does this mean?
```

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

Product	Max(quantity)
Bagel	20
Banana	50

Product	Quantity
Bagel	20
Banana	??

INF



Semantics of SQL With Group-By

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

FWGHOS

Evaluation steps:

1. Evaluate FROM-WHERE using Nested Loop Semantics
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

```
Purchase(pid, product, price, quantity, month)
```

Exercise

FWGHOS

Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”

```
Purchase(pid, product, price, quantity, month)
```

Exercise

FWGHOS

Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”

```
FROM        Purchase
```

```
Purchase(pid, product, price, quantity, month)
```

Exercise

FWGHOS

Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”

```
FROM        Purchase
GROUP BY    month
```

Purchase(pid, product, price, quantity, month)

Exercise

FWGHOS

Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”

```
FROM        Purchase
GROUP BY    month
HAVING      sum(quantity) < 10
```

Purchase(pid, product, price, quantity, month)

Exercise

FWGHOS

Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”

```
SELECT   month, sum(price*quantity),
         sum(quantity) as TotalSold
FROM     Purchase
GROUP BY month
HAVING   sum(quantity) < 10
```

Purchase(pid, product, price, quantity, month)

Exercise

FWGHOS

Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”

```
SELECT     month, sum(price*quantity),
           sum(quantity) as TotalSold
FROM       Purchase
GROUP BY   month
HAVING     sum(quantity) < 10
ORDER BY   sum(quantity)
```

WHERE vs HAVING

- WHERE condition is applied to individual rows
  - The rows may or may not contribute to the aggregate
  - No aggregates allowed here

- HAVING condition is applied to the entire group
  - Only applicable if GROUP BY is involved
  - Entire group is returned, or removed
  - May use aggregate functions on the group

```
Product(pid,pname,manufacturer)
Purchase(id,product_id,price,month)
```

Aggregate + Join

For each manufacturer, compute how many products with price > $100 they sold

```
Product(pid,pname,manufacturer)
Purchase(id,product_id,price,month)
```

Aggregate + Join

For each manufacturer, compute how many products with price > $100 they sold

Problem: manufacturer is in Product, price is in Purchase...

```
Product(pid,pname,manufacturer)
Purchase(id,product_id,price,month)
```

Aggregate + Join

For each manufacturer, compute how many products with price > $100 they sold

Problem: manufacturer is in Product, price is in Purchase...

```
-- step 1: think about their join
SELECT ...
FROM Product x, Purchase y
WHERE x.pid = y.product_id
  and y.price > 100
```

manu facturer	...	price	...
Hitachi		150	
Canon		300	
Hitachi		180	

```
Product(pid,pname,manufacturer)
Purchase(id,product_id,price,month)
```

Aggregate + Join

For each manufacturer, compute how many products with price > $100 they sold

Problem: manufacturer is in Product, price is in Purchase...

```
-- step 1: think about their join
SELECT ...
FROM Product x, Purchase y
WHERE x.pid = y.product_id
  and y.price > 100
```

manu facturer	...	price	...
Hitachi		150	
Canon		300	
Hitachi		180	

```
-- step 2: do the group-by on the join
SELECT x.manufacturer, count(*)
FROM Product x, Purchase y
WHERE x.pid = y.product_id
  and y.price > 100
GROUP BY  x.manufacturer
```

manu facturer	count(*)
Hitachi	2
Canon	1
...	

Product(pid,pname,manufacturer)
Purchase(id,product_id,price,month)

Aggregate + Join

Variant:
For each manufacturer, compute how many products with price > $100 they sold in each month

```
SELECT x.manufacturer, y.month, count(*)
FROM Product x, Purchase y
WHERE x.pid = y.product_id
  and y.price > 100
GROUP BY  x.manufacturer, y.month
```

manu facturer	month	count(*)
Hitachi	Jan	2
Hitachi	Feb	1
Canon	Jan	3
...		

FWGHOS

Including Empty Groups

- In the result of a group by query, there is one row per group in the result

Count(*) is never 0

```
SELECT x.manufacturer, count(*)
FROM Product x, Purchase y
WHERE x.pname = y.product
GROUP BY x.manufacturer
```

Including Empty Groups

```
SELECT x.manufacturer, count(y.pid)
FROM Product x LEFT OUTER JOIN Purchase y
ON x.pname = y.product
GROUP BY x.manufacturer
```

Count(pid) is 0 when all pid's in the group are NULL

What we have in our SQL toolbox

- Projections (`SELECT * / SELECT c1, c2, …`)
- Selections (aka filtering) (`WHERE cond`)
- Joins (inner and outer)
- Aggregates
- Group by
- Inserts, updates, and deletes

Make sure you read the textbook!

Subqueries

- A subquery is a SQL query nested inside a larger query
- Such inner-outer queries are called nested queries
- A subquery may occur in:
  - A `SELECT` clause
  - A `FROM` clause
  - A `WHERE` clause

- Rule of thumb: avoid nested queries when possible
  - But sometimes it's impossible, as we will see

FWGHOS

Subqueries…

- Can return a single value to be included in a `SELECT` clause
- Can return a relation to be included in the `FROM` clause, aliased using a tuple variable
- Can return a single value to be compared with another value in a `WHERE` clause
- Can return a relation to be used in the `WHERE` or `HAVING` clause under an existential quantifier

1. Subqueries in SELECT

```
Product (pname,  price, cid)
Company (cid, cname, city)
```

For each product return the city where it is manufactured

```
SELECT X.pname, (SELECT Y.city
                 FROM Company Y
                 WHERE Y.cid=X.cid) as City
FROM   Product X
```

"correlated subquery"

What happens if the subquery returns more than one city?

We get a runtime error

(and SQLite simply ignores the extra values…)

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

1. Subqueries in SELECT

Whenever possible, don't use a nested queries:

```
SELECT X.pname, (SELECT Y.city
                 FROM Company Y
                 WHERE Y.cid=X.cid) as City
FROM   Product X
```

=

```
SELECT X.pname, Y.city
FROM   Product X, Company Y
WHERE  X.cid=Y.cid
```

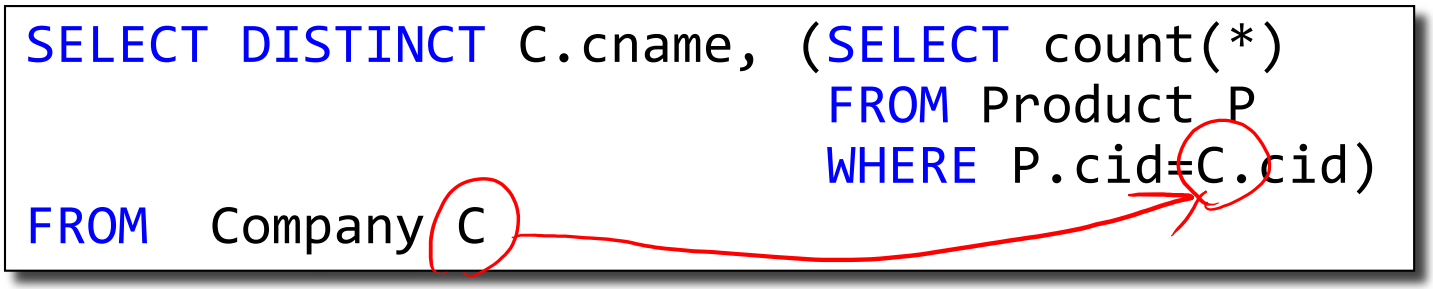
We have "unnested" the query

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

1. Subqueries in SELECT

Compute the number of products made by each company

```
SELECT DISTINCT C.cname, (SELECT count(*)
                          FROM Product P
                          WHERE P.cid=C.cid)
FROM   Company C
```



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

1. Subqueries in SELECT

Compute the number of products made by each company

```
SELECT DISTINCT C.cname, (SELECT count(*)
                          FROM Product P
                          WHERE P.cid=C.cid)
FROM  Company C
```

Better: we can unnest using a GROUP BY

```
SELECT C.cname, count(*)
FROM   Company C, Product P
WHERE  C.cid=P.cid
GROUP BY C.cname
```

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

1. Subqueries in SELECT

But are these really equivalent?

```
SELECT DISTINCT C.cname, (SELECT count(*)
                          FROM Product P
                          WHERE P.cid=C.cid)
FROM  Company C
```

```
SELECT C.cname, count(*)
FROM   Company C, Product P
WHERE  C.cid=P.cid
GROUP BY C.cname
```

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

1. Subqueries in SELECT

But are these really equivalent?

```
SELECT DISTINCT C.cname, (SELECT count(*)
                          FROM Product P
                          WHERE P.cid=C.cid)
FROM  Company C
```

```
SELECT C.cname, count(*)
FROM   Company C, Product P
WHERE  C.cid=P.cid
GROUP BY C.cname
```

No! Different results if a company has no products

```
SELECT C.cname, count(pname)
FROM Company C LEFT OUTER JOIN Product P
ON   C.cid=P.cid
GROUP BY C.cname
```

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

2. Subqueries in FROM

Find all products whose prices is > 20 and < 500

```
SELECT X.pname
FROM (SELECT *
      FROM Product AS Y
      WHERE price > 20) as X
WHERE X.price < 500
```

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

2. Subqueries in FROM

Find all products whose prices is > 20 and < 500

```
SELECT X.pname
FROM (SELECT *
      FROM Product AS Y
      WHERE price > 20) as X
WHERE X.price < 500
```

Try unnest this query !

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

2. Subqueries in FROM

Find all products whose prices is > 20 and < 500

```
SELECT X.pname
FROM (SELECT *
      FROM Product AS Y
      WHERE price > 20) as X
WHERE X.price < 500
```

Side note: This is not a correlated subquery. (why?)

Try unnest this query !

2. Subqueries in FROM

Sometimes we need to compute an intermediate table only to use it later in a SELECT-FROM-WHERE

- Option 1: use a subquery in the FROM clause
- Option 2: use the WITH clause
  - See textbook for details

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

2. Subqueries in FROM

```
SELECT X.pname
FROM (SELECT *
      FROM Product AS Y
      WHERE price > 20) as X
WHERE X.price < 500
```

=

A subquery whose result we called myTable

```
WITH myTable AS (SELECT * FROM Product AS Y WHERE price > 20)
  SELECT X.pname
  FROM myTable as X
  WHERE X.price < 500
```