Introduction to Database Systems
CSE 444

Lecture 3: SQL (part 2)

Outline

- Aggregations (6.4.3 – 6.4.6)
- Examples, examples, examples…
- Nulls (6.1.6 - 6.1.7) [Old edition: 6.1.5-6.1.6]
- Outer joins (6.3.8)

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

Aggregation: Count Distinct

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
```

Simple Aggregation 1/2

Purchase (product, price, quantity)

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

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

What do these queries mean?

Simple Aggregation 2/2

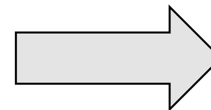
Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

3 * 20 = 60
2 * 20 = 40
sum: 100

SQL creates attribute name

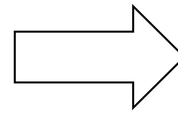
```
SELECT  sum (price * quantity)
FROM    Purchase
WHERE   product = 'Bagel'
```



(No column name)
100

Grouping and Aggregation

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
~~Banana~~	~~1~~	~~50~~
Banana	2	10
Banana	4	10

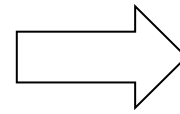


Product	TotalSales
Bagel	40
Banana	20

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

From → Where → Group By → Select

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
~~Banana~~	~~1~~	~~50~~
Banana	2	10
Banana	4	10

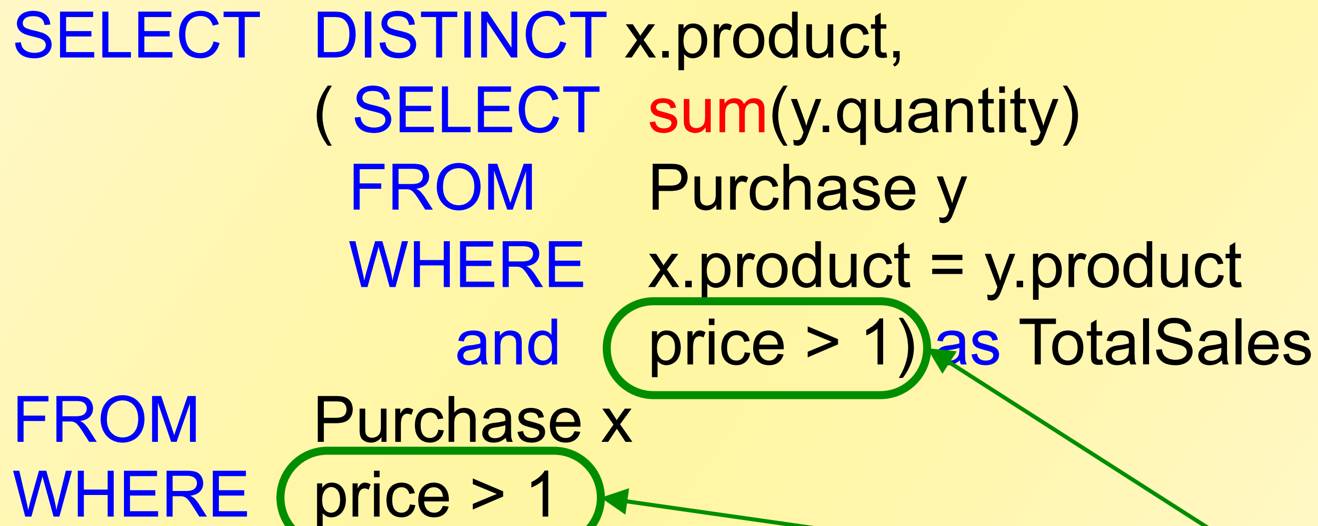


Product	TotalSales
Bagel	40
Banana	20

Select contains
- grouped attributes
- and aggregates

```
4  SELECT     product, sum(quantity) as TotalSales
1  FROM       Purchase
2  WHERE      price > 1
3  GROUP BY   product
```

Group By v.s. Nested Queries

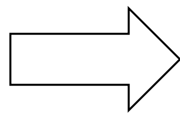


Why twice?

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

Another Example

```
SELECT   product,
         sum(quantity) as SumQuantity,
         max(price) as MaxPrice
FROM     Purchase
GROUP BY product
```



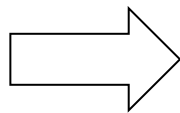
Product	TotalSales	MaxPrice
Bagel	40	3
Banana	70	4

Next, focus only on products with at least 50 sales

HAVING Clause

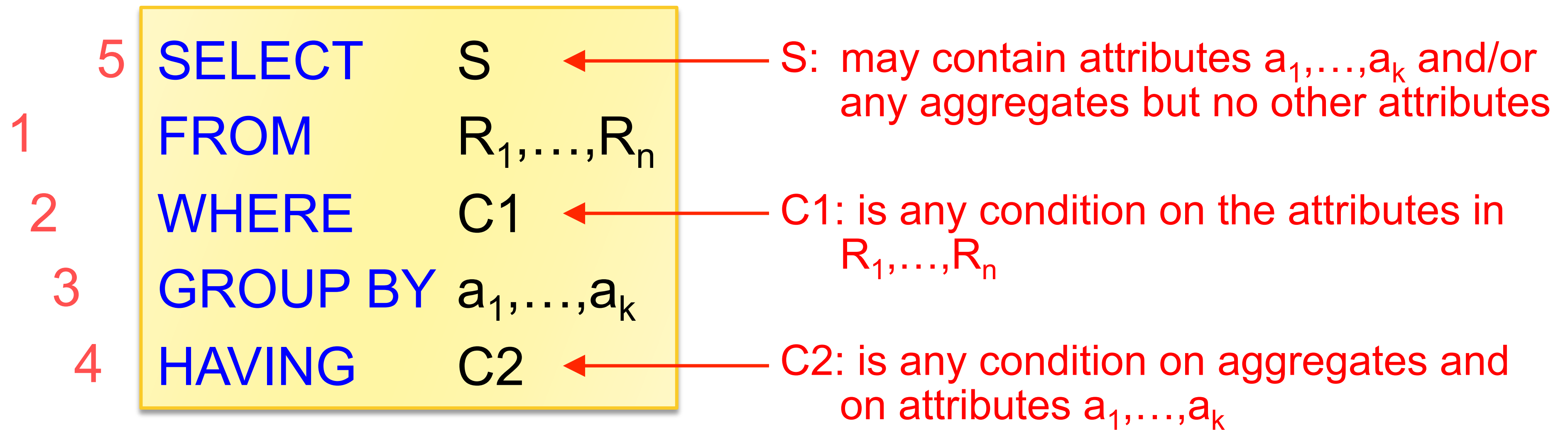
Q: Similar to before, but only products with at least 30 sales.

```
SELECT   product,
         sum(quantity) as SumQuantity,
         max(price) as MaxPrice
FROM     Purchase
GROUP BY product
HAVING   sum(quantity) > 50
```



Product	TotalSales	MaxPrice
Banana	70	4

General form of Grouping and Aggregation



Evaluation

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

Advanced SQLizing

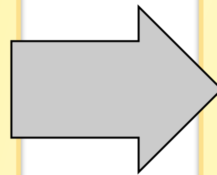
1. Getting around INTERSECT and EXCEPT
2. Unnesting Aggregates
3. Finding witnesses

INTERSECT and EXCEPT*

```
(SELECT R.A, R.B
 FROM    R)

INTERSECT

(SELECT S.A, S.B
 FROM   S)
```



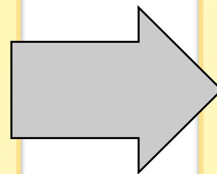
```
SELECT R.A, R.B
FROM   R
WHERE
    EXISTS(SELECT  *
           FROM    S
           WHERE  R.A=S.A
              and  R.B=S.B)
```

Can be unnested. How?

```
(SELECT R.A, R.B
 FROM    R)

EXCEPT

(SELECT S.A, S.B
 FROM   S)
```



```
SELECT R.A, R.B
FROM   R
WHERE
  NOT  EXISTS(SELECT  *
              FROM    S
              WHERE  R.A=S.A
                 and  R.B=S.B)
```

*Not in all DBMSs

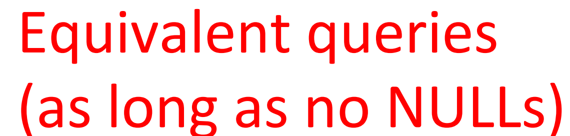
Unnesting Aggregates

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

Find the number of companies in each city

```
SELECT DISTINCT city, (SELECT   count(*)
                       FROM     Company Y
                       WHERE    X.city = Y.city)
FROM  Company X
```

```
SELECT      city, count(*)
FROM        Company
GROUP BY city
```



Note: no need for DISTINCT
(DISTINCT *is the same* as GROUP BY)

Unnesting Aggregates

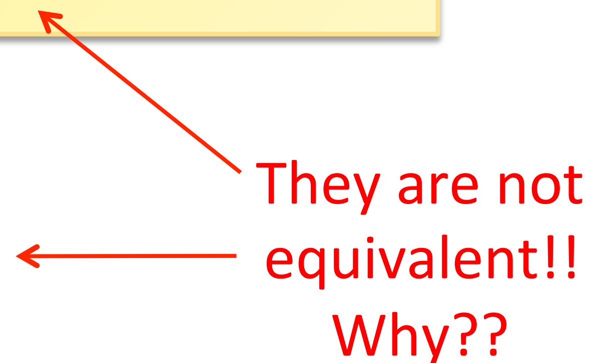
```
Product ( pname,  price, company)
Company(cname, city)
```

What if there are no products for a city? ☺

Find the number of products made in each city

```
SELECT DISTINCT X.city, (SELECT count(*)
                         FROM   Product Y, Company Z
                         WHERE Z.cname = Y.company
                              and Z.city = X.city)
FROM  Company X
```

```
SELECT      X.city, count(*)
FROM        Company X, Product Y
WHERE       X.cname = Y.company
GROUP BY  X.city
```



More on Unnesting

- Find all authors who wrote at least 10 documents:

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

Works, but this is bad style

- Second attempt (no nesting):

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

Much better!

Finding Witnesses

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

Q: For each store, find its most expensive products

Finding the maximum price is easy…

```
SELECT Store.sid, max(Product.price)
FROM    Store, Product
WHERE  Store.sid = Product.sid
GROUP BY  Store.sid
```

But we want the “witnesses”, i.e. the products with max price

Finding Witnesses

Plan:

- Compute max price in a subquery
- Compare it with each product price

```
SELECT Store.sname, Product.pname
FROM Store, Product,
          (SELECT Store.sid as sid, max(Product.price) as p
           FROM    Store, Product
           WHERE  Store.sid = Product.sid
            GROUP BY  Store.sid) X
WHERE  Store.sid = Product.sid
      and Store.sid = X.sid
      and Product.price = X.p
```

Finding Witnesses

There is a more concise solution here:

```
SELECT Store.sname, 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)
```

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 it can be NULL (*nullable* attribute) or not

- How does SQL cope with tables that have NULLs ?

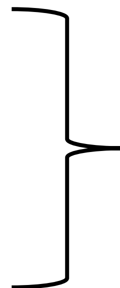
Null Values

- If x= NULL then
  - Arithmetic operations produce NULL. E.g: 4*(3-x)/7
  - Boolean conditions are also NULL. E.g: x='Joe'

- In SQL there are three boolean values:

  FALSE, TRUE, UNKNOWN

- Reasoning:

FALSE = 0		x AND y = min(x,y)
TRUE = 1		x OR y = max(x,y)
UNKNOWN = 0.5		NOT x = (1 – x)

Null Values: example

```
SELECT   *
FROM     Person
WHERE  (age < 25)
     and  (height > 6 or weight > 190)
```

Age	Height	Weight
20	NULL	200
~~NULL~~	~~6.5~~	~~170~~

Rule in SQL:
include only tuples that
yield TRUE

```
SELECT   *
FROM     Person
WHERE  age < 25  or age >= 25
```

← Unexpected behavior

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

Test NULL
explicitly

Outerjoins

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

An “inner join”:

```
SELECT Product.name, Purchase.store
FROM    Product, Purchase
WHERE   Product.name = Purchase.prodName
```

Same as:

```
SELECT Product.name, Purchase.store
FROM    Product JOIN Purchase ON
                 Product.name = Purchase.prodName
```

But Products that never sold will be lost !

Outerjoins

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

If we want the never-sold products, we need an "outerjoin":

```
SELECT Product.name, Purchase.store
FROM   Product LEFT OUTER JOIN Purchase ON
                    Product.name = Purchase.prodName
```

Product

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Result

Name	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz
OneClick	NULL

Inner join does not produce this tuple

Example

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

- Compute, for each product, the total number of sales in 'September'

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

What's wrong?

Example

Product(<u>name</u>, category)
Purchase(prodName, month, store)

- Compute, for each product, the total number of sales in 'September'

We need to use the attribute to get the correct 0 count. (§6.4.6)

```
SELECT    Product.name, count(store)
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 with 0 sales

Outer Joins: summary

- 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 both left and right tuples even if there's no match