# Introduction to Database Systems
# CSE 414

## Lecture 5: SQL Aggregates

Product(pname, price, category, manufacturer)
Company(cname, country)

# Joins in SQL

| pname | price | category | manufacturer |
|---|---|---|---|
| MultiTouch | 199.99 | gadget | Canon |
| SingleTouch | 49.99 | photography | Canon |
| Gizom | 50 | gadget | GizmoWorks |
| SuperGizmo | 250.00 | gadget | GizmoWorks |

| cname | country |
|---|---|
| GizmoWorks | USA |
| Canon | Japan |
| Hitachi | Japan |

Retrieve all Japanese products that cost < $150

```
SELECT P.pname, P.price
FROM   Product as P, Company as C
WHERE  P.manufacturer=C.cname AND
       C.country='Japan' AND C.price < 150
```

Join Predicate

Selection predicates

# (Inner) Joins

```
SELECT   x1.a1, x2.a2, … xm.am
FROM     R1 as x1, R2 as x2, … Rm as xm
WHERE    Cond
```

tuple in R1

```
for x1 in R1:
  for x2 in R2:
    ...
      for xm in Rm:
        if Cond(x1, x2…):
          output(x1.a1, x2.a2, … xm.am)
```

This is called nested loop semantics since we are interpreting what a join means using a nested loop

# Self Join Example

```
Product(pname, price, category, manufacturer)
Company(cname, country)
-- manufacturer is foreign key to Company
```

Find US companies that manufacture both ‘gadgets’ and ‘photo’ products

```
SELECT DISTINCT z.cname
FROM Product x, Product y, Company z
WHERE z.country = ’USA’
  AND x.manufacturer = z.cname
  AND y.manufacturer = z.cname
  AND x.category = 'gadget’
  AND y.category = 'photography;
```

Need to include Product twice!

# Joins in SQL

- The join we have just seen is sometimes called an inner join
  - Each row in the result **must come from both tables in the join**
- Sometimes we want to include rows from only one of the two table: outer join

Employee(id, name)
Sales(employeeID, productID)

# Outer Join

left

right

Employee

| id | name |
|---|---|
| 1 | Joe |
| 2 | Jack |
| 3 | Jill |

Sales

| employeeID | productID |
|---|---|
| 1 | 344 |
| 1 | 355 |
| 2 | 544 |

## Retrieve employees and their sales

```
SELECT *
FROM   Employee E
       LEFT OUTER JOIN
       Sales S
    ON E.id = S.employeeID
```

Jill is present

| id | name | empolyeeID | productID |
|---|---|---|---|
| 1 | Joe | 1 | 344 |
| 1 | Joe | 1 | 355 |
| 2 | Jack | 2 | 544 |
| 3 | Jill | NULL | NULL |

# Outer Joins

```
tableA (LEFT/RIGHT/FULL) OUTER JOIN tableB ON p
```

- Left outer join:
  - Include tuples from `tableA` even if no match
- Right outer join:
  - Include tuples from `tableB` even if no match
- Full outer join:
  - Include tuples from both even if no match

- In all cases:
  - Patch tuples without matches using `NULL`

# Aggregates in SQL

# 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

# Demo

# Aggregates and NULL Values

Null values are not used in aggregates

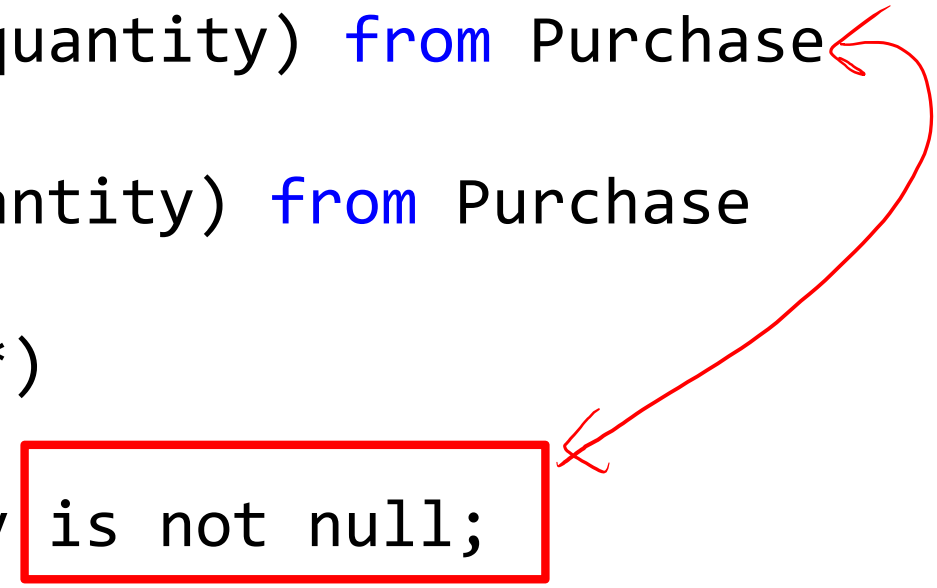
```
insert into Purchase
values(12, 'gadget', NULL, NULL, 'april')
```

Let's try the following

```
select count(*) from Purchase
select count(quantity) from Purchase

select sum(quantity) from Purchase

select count(*)
from Purchase
where quantity is not null;
```



# Counting Duplicates

`COUNT` applies to duplicates, unless otherwise stated:

```
SELECT count(product)
FROM   Purchase
WHERE  price > 4.99
```

same as `count(*)` if no nulls

We probably want:

```
SELECT count(DISTINCT product)
FROM   Purchase
WHERE  price > 4.99
```

# More Examples

```
SELECT   Sum(P.price * P.quantity)
FROM     Purchase as P
```

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

What do they mean ?

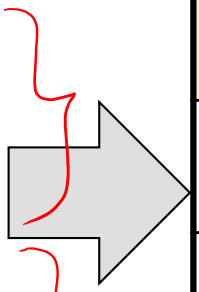
# Grouping and Aggregation

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

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

# Grouping and Aggregation

| 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
```

# Other Examples

Compare these two queries:

```
SELECT   product, count(*)
FROM     Purchase
GROUP BY product
```

```
SELECT   month, count(*)
FROM     Purchase
GROUP BY month
```

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

What does it return?

# Need to be Careful…

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

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

# 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 |

# 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 |

# 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 | ?? |

# Need to be Careful…

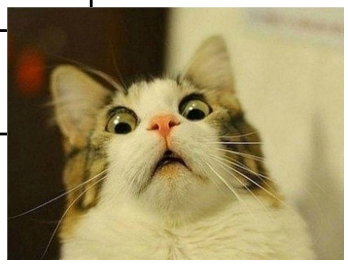
```
SELECT product,
       max(quantity)
FROM   Purchase
GROUP BY product
```

```
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 | ?? |



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

# Need to be Careful…

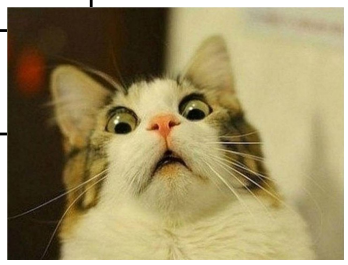
```
SELECT product,
       max(quantity)
FROM   Purchase
GROUP BY product
```

```
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 | ?? |



# 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
```

How is this query processed?

# 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
```

Do these queries return the same number of rows? Why?

```
SELECT   product, Sum(quantity) AS TotalSales
FROM     Purchase
GROUP BY product
```

# 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
```

Do these queries return the same number of rows? Why?

```
SELECT   product, Sum(quantity) AS TotalSales
FROM     Purchase
GROUP BY product
```

Rows where price > 1are removed, so first query may return fewer groups

# Grouping and Aggregation

1. Compute the FROM and WHERE clauses.

2. Group by the attributes in the GROUPBY

3. Compute the SELECT clause:
   grouped attributes and aggregates.

FWGS™

# 1,2: From, Where

FWGS

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

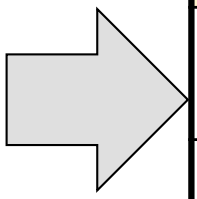
WHERE price > 1

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

# 3,4. Grouping, Select

FWGS

| 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
```

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

# Ordering Results

```
SELECT product, sum(price*quantity) as rev
FROM    Purchase
GROUP BY product
ORDER BY rev desc
```

FWGOS™

Note: some SQL engines
want you to say `ORDER BY sum(price*quantity) desc`

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

# HAVING Clause

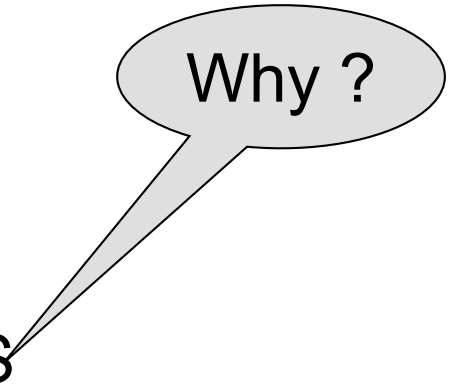
Same query as before, except that we consider only products that had at least 30 sales.

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

HAVING clause contains conditions on aggregates.

# General form of Grouping and Aggregation

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




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 and on attributes $a_1,\ldots,a_k$

# 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