

Introduction to Database Systems

SQL Aggregation & Grouping


Aggregation in SQL

```
>sqlite3 lecture04
```

```
sqlite> create table Purchase(  
    pid int primary key,  
    product text,  
    price float,  
    quantity int,  
    month varchar(15));
```

```
sqlite> -- download data.txt
```

```
sqlite> .import lec04-data.txt Purchase
```



Other DBMSs have
other ways of
importing data

Comment about SQLite

- One cannot load NULL values such that they are actually loaded as null values
- So we need to use two steps:
 - Load null values using some type of special value
 - Update the special values to actual null values

```
update Purchase  
  set price = null  
 where price = 'null'
```

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 value

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 sum(quantity)
```

```
from Purchase
```

```
where quantity is not null;
```

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  
    -- NULL is counted in count(*)  
select count(quantity) from Purchase  
    -- NULL is ignored in count(quantity)
```

```
select sum(quantity) from Purchase
```

```
select sum(quantity)  
from Purchase  
where quantity is not null;  
    -- "is not null" is redundant
```

Counting Duplicates

COUNT applies to duplicates, unless otherwise stated:

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

same as Count(*) if no nulls

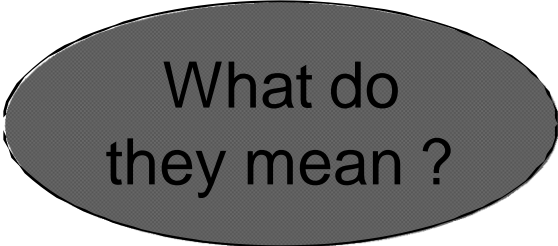
We probably want:

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

More Examples

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

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



What do
they mean ?

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'
```



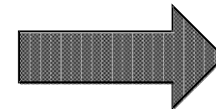
90 (= 60+30)

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)

More Examples

How can we find the average revenue per sale?

```
SELECT sum(price * quantity) / count(*)  
FROM   Purchase  
WHERE  product = 'bagel'
```

How can we find the average price of a bagel sold?

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

More Examples

```
SELECT sum(price * quantity) / count(*)  
FROM   Purchase  
WHERE  product = 'bagel'
```

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

What happens if there are NULLs in price or quantity?

Lesson: disallow NULLs unless you need to handle them

Grouping and Aggregation

Purchase(product, price, quantity)

Find number of bagels sold for more than \$1

SELECT	Sum(quantity) as TotalSold
FROM	Purchase
WHERE	price > 1 and product = 'bagel'

Grouping and Aggregation

Purchase(product, price, quantity)

Find number sold for more than \$1 **for each product**

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

Let's see what this means...

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.

FWGS

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

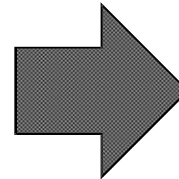
FWGS

WHERE price > 1

3. SELECT

FWGS

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



Product	sum(quantity)
Bagel	40
Banana	20

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

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

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
```

How about
this one?

Need to be Careful...

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

```
SELECT product, 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

sqlite allows this
query to be executed
with strange
behavior.

Better DBMS (e.g., SQL
Server) gives an error

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

Ordering Results

```
SELECT product, sum(price*quantity)
FROM Purchase
GROUP BY product
ORDER BY sum(price*quantity) DESC
```

FWGOS

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)

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

HAVING Clause

Same query as earlier, 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
```

FWGHOS

HAVING clause contains conditions on groups.

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

Exercise

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)
```

FWGHOS

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
 - Entire group is returned, or not at all
 - May use aggregate functions in the group

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

Mystery Query

What do they compute?

```
SELECT    month, sum(quantity), max(price)
FROM      Purchase
GROUP BY  month
```

```
SELECT    month, sum(quantity)
FROM      Purchase
GROUP BY  month
```

```
SELECT    month
FROM      Purchase
GROUP BY  month
```

Lesson:
DISTINCT is
a special case
of GROUP BY

Aggregates and Joins

```
create table Product(  
    pid int primary key,  
    pname varchar(15),  
    manufacturer varchar(15));  
  
insert into product values(1,'bagel','Sunshine Co.');
```

pid	pname	manufacturer
1	bagel	Sunshine Co.
2	banana	BusyHands
3	gizmo	GizmoWorks
4	gadget	BusyHands
5	powerGizmo	PowerWorks

```
insert into product values(2,'banana','BusyHands');
```

pid	pname	manufacturer
1	bagel	Sunshine Co.
2	banana	BusyHands
3	gizmo	GizmoWorks
4	gadget	BusyHands
5	powerGizmo	PowerWorks

```
insert into product values(3,'gizmo','GizmoWorks');
```

pid	pname	manufacturer
1	bagel	Sunshine Co.
2	banana	BusyHands
3	gizmo	GizmoWorks
4	gadget	BusyHands
5	powerGizmo	PowerWorks

```
insert into product values(4,'gadget','BusyHands');
```

pid	pname	manufacturer
1	bagel	Sunshine Co.
2	banana	BusyHands
3	gizmo	GizmoWorks
4	gadget	BusyHands
5	powerGizmo	PowerWorks

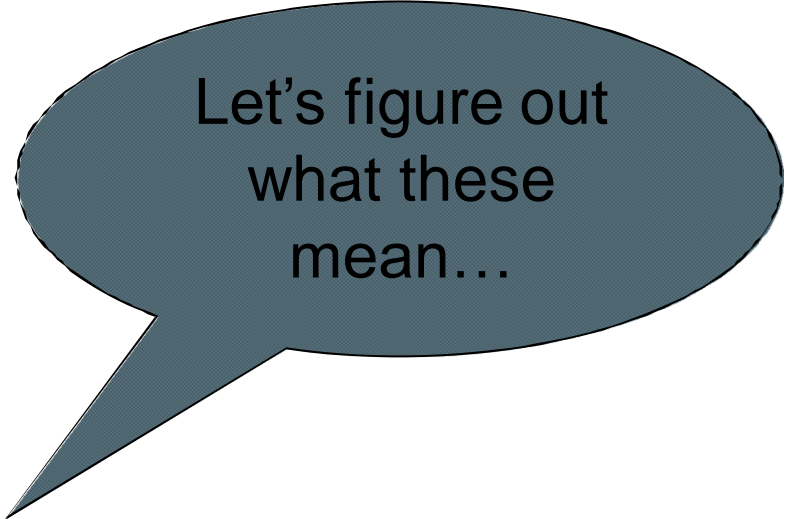
```
insert into product values(5,'powerGizmo','PowerWorks');
```

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

Product(pid, pname, manufacturer)

Aggregate + Join Example

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



Let's figure out
what these
mean...

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

Nested Loop Semantics for SFW

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

for x1 in R1:

 for x2 in R2:

 ...

 for xm in Rm:

 if Cond(x1, x2...):

 output(x1.a1, x2.a2, ... xm.am)

Nested loop
semantics

Semantics for SFWGH

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

S = may contain attributes a_1, \dots, a_k and/or any aggregates, but NO OTHER ATTRIBUTES

C1 = is any condition on the attributes in R_1, \dots, R_n

C2 = is any condition on aggregate expressions and on attributes a_1, \dots, a_k



Why ?

Semantics for SFWGH

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

Evaluation steps:

1. Evaluate FROM-WHERE using Nested Loop Semantics
2. Group by the attributes a_1, \dots, a_k
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result

Semantics for SFWGH

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

Execution order:

FWGHOS

Evaluation steps:

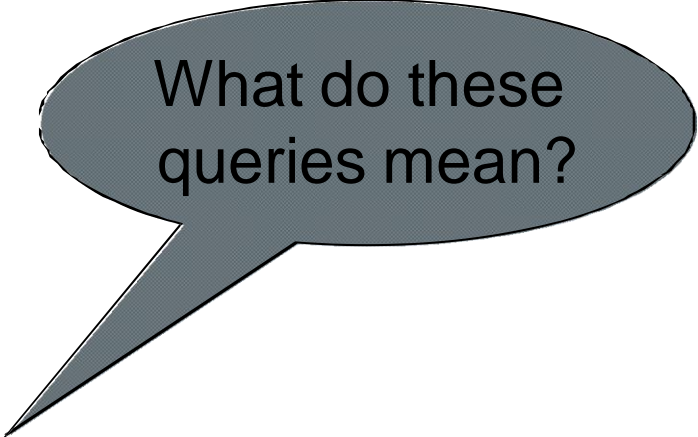
1. Evaluate FROM-WHERE using Nested Loop Semantics
2. Group by the attributes a_1, \dots, a_k
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result

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

Product(pid, pname, manufacturer)

Aggregate + Join Example

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



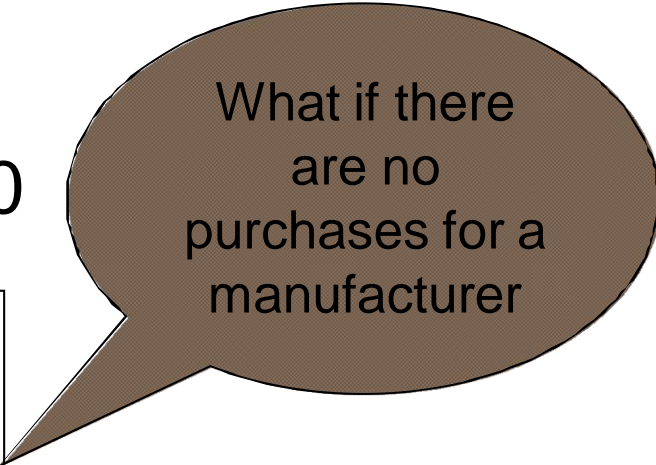
What do these queries mean?

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


Empty Groups

- In the result of a group by query, there is one row per group in the result
- No group can be empty!
- In particular, count(*) is never 0

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



What if there
are no
purchases for a
manufacturer

Empty Group Solution: Outer Join

```
SELECT manufacturer, count(quantity)
FROM Product LEFT OUTER JOIN Purchase
ON pname = product
GROUP BY manufacturer
```



Why not count (*) ?

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

Product(pid, pname, manufacturer)

Exercise:

Find all manufacturers with more than 10 items sold.
Return manufacturer name and number of items sold.

```
SELECT manufacturer, sum(quantity)
FROM Product, Purchase
WHERE pname = product
GROUP BY manufacturer
HAVING sum(quantity) > 10
```

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

Product(pid, pname, manufacturer)

Exercise:

Find all manufacturers with more than 1 distinct product sold
Return the name of the manufacturer and
number of distinct products sold

```
SELECT manufacturer, count(distinct product)
FROM Product, Purchase
WHERE pname = product
GROUP BY manufacturer
HAVING count(distinct product) > 1
```

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

Product(pid, pname, manufacturer)

Exercise:

Find all products with more than 2 purchases

Return the name of the product and max price it was sold

```
SELECT pname, max(price)
FROM Product, Purchase
WHERE pname = product
GROUP BY pname
HAVING COUNT(*) > 2
```

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

Product(pid, pname, manufacturer)

Exercise:

Find all manufacturers with at least 5 purchases in one month
Return manufacturer name, month, and number of items sold

```
SELECT manufacturer, month, sum(quantity)
FROM Product, Purchase
WHERE pname = product
GROUP BY manufacturer, month
HAVING count(*) >= 5
```