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.

\[
\text{SELECT product, Sum(quantity) AS TotalSales FROM Purchase WHERE price > 1 GROUP BY product}
\]

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

\[
\text{SELECT product, Sum(quantity) AS TotalSales FROM Purchase GROUP BY product}
\]

Empty groups are removed, hence 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.
### 1,2: FROM, WHERE

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

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>Bagel</td>
<td>1.50</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>0.5</td>
<td>50</td>
</tr>
<tr>
<td>Banana</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Banana</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

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

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

Note: some SQL engines want you to say ORDER BY sum(price*quantity) desc
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 R_1, ..., R_n
WHERE C_1
GROUP BY a_1, ..., a_k
HAVING C_2

S = may contain attributes a_1, ..., a_k and/or any aggregates but NO OTHER ATTRIBUTES
C_1 = is any condition on the attributes in R_1, ..., R_n
C_2 = is any condition on aggregate expressions and on attributes a_1, ..., a_k
SEMANTICS OF SQL WITH GROUP-BY

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

Evaluation steps:

1. Evaluate FROM-WHERE using Nested Loop Semantic
2. Group by the attributes a₁,...,aₖ
3. Apply condition C₂ to each group (may have aggregates)
4. Compute aggregates in S and return the result
EXERCISE

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

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
EXERCISE

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

\[
\text{FROM Purchase}
\]
\[
\text{GROUP BY month}
\]
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
```
Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”

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

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

```sql
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
• Occasionally, some groups become empty and are removed

HAVING condition is applied to the entire group

• Entire group is returned, or removed
• May use aggregate functions on the group
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
```
What do they compute?

\[
\text{SELECT} \quad \text{month, sum(quantity), max(price)} \\
\text{FROM} \quad \text{Purchase} \\
\text{GROUP BY} \quad \text{month}
\]

\[
\text{SELECT} \quad \text{month, sum(quantity)} \\
\text{FROM} \quad \text{Purchase} \\
\text{GROUP BY} \quad \text{month}
\]

\[
\text{SELECT} \quad \text{month} \\
\text{FROM} \quad \text{Purchase} \\
\text{GROUP BY} \quad \text{month}
\]

Lesson: DISTINCT is a special case of GROUP BY
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 Purchase, price is in Product...
AGGREGATE + JOIN

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

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

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

<table>
<thead>
<tr>
<th>manufacturer</th>
<th>...</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hitachi</td>
<td></td>
<td>150</td>
</tr>
<tr>
<td>Canon</td>
<td></td>
<td>300</td>
</tr>
<tr>
<td>Hitachi</td>
<td></td>
<td>180</td>
</tr>
</tbody>
</table>
AGGREGATE + JOIN

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

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

-- step 1: think about their join

```
SELECT ...
FROM Product x, Purchase y
WHERE x.pid = y.product_id
and y.price > 100
```

-- 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
```
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
```
INCLUDING EMPTY GROUPS

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

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

Count(*) is never 0
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
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
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

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

What happens if the subquery returns more than one city?

We get a runtime error

(and SQLite simply ignores the extra values…)
Whenever possible, don’t use a nested queries:

\[
\text{SELECT } X.pname, (\text{SELECT } Y.city \\
\text{ FROM Company } Y \\
\text{ WHERE } Y.cid=X.cid) \text{ as City} \\
\text{FROM } Product X
\]

We have “unnested” the query.
Compute the number of products made by each company

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

But are these really equivalent?

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

No! Different results if a company has no products

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

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

Find all products whose prices is > 20 and < 500

```sql
SELECT X.pname
FROM (SELECT *
    FROM Product AS Y
    WHERE price > 20) as X
WHERE X.price < 500
```
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!
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
Product (pname, price, cid)
Company (cid, cname, city)

2. SUBQUERIES IN FROM

```sql
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

```sql
WITH myTable AS (SELECT *
                 FROM Product AS Y
                 WHERE price > 20)
SELECT X.pname
FROM myTable as X
WHERE X.price < 500
```
Find all companies that make some products with price < 200
Find all companies that make *some* products with price < 200
Find all companies that make some products with price < 200

Using **EXISTS**:  

```sql
SELECT DISTINCT C.cname
FROM Company C
WHERE EXISTS (SELECT *
               FROM Product P
               WHERE C.cid = P.cid AND P.price < 200)
```
3. SUBQUERIES IN WHERE

Find all companies that make some products with price < 200

Using **IN**

```
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid IN (SELECT P.cid
                 FROM Product P
                 WHERE P.price < 200)
```
3. SUBQUERIES IN WHERE

Find all companies that make some products with price < 200

Using ANY:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE 200 > ANY (SELECT price
FROM Product P
WHERE P.cid = C.cid)
```
3. SUBQUERIES IN WHERE

Find all companies that make some products with price < 200

Using ANY:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE 200 > ANY (SELECT price
FROM Product P
WHERE P.cid = C.cid)
```

Existential quantifiers

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

Not supported in sqlite
3. SUBQUERIES IN WHERE

Find all companies that make some products with price < 200

Now let’s unnest it:

```sql
SELECT DISTINCT C.cname
FROM Company C, Product P
WHERE C.cid = P.cid and P.price < 200
```
3. SUBQUERIES IN WHERE

Find all companies that make some products with price < 200

Existential quantifiers

Now let’s unnest it:

```
SELECT DISTINCT C.cname
FROM Company C, Product P
WHERE C.cid = P.cid and P.price < 200
```

Existential quantifiers are easy! 😊
3. SUBQUERIES IN WHERE

Find all companies s.t. all their products have price < 200

same as:

Find all companies that make only products with price < 200
3. SUBQUERIES IN WHERE

Find all companies s.t. all their products have price < 200

same as:

Find all companies that make only products with price < 200

Product (pname, price, cid)
Company (cid, cname, city)
3. SUBQUERIES IN WHERE

Find all companies s.t. all their products have price < 200

same as:

Find all companies that make only products with price < 200

Universal quantifiers are hard!
3. SUBQUERIES IN WHERE

Find all companies s.t. all their products have price < 200

1. Find the other companies that make some product ≥ 200

```sql
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid IN (SELECT P.cid
                  FROM Product P
                  WHERE P.price >= 200)
```
3. SUBQUERIES IN WHERE

Find all companies s.t. **all** their products have price < 200

1. Find *the other* companies that make **some** product ≥ 200

```
SELECT DISTINCT C.cname  
FROM Company C  
WHERE C.cid IN (SELECT P.cid  
                     FROM Product P  
                     WHERE P.price >= 200)
```

2. Find all companies s.t. **all** their products have price < 200

```
SELECT DISTINCT C.cname  
FROM Company C  
WHERE C.cid NOT IN (SELECT P.cid  
                     FROM Product P  
                     WHERE P.price >= 200)
```
3. SUBQUERIES IN WHERE

Find all companies s.t. all their products have price < 200

Using EXISTS:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE NOT EXISTS (SELECT *
FROM Product P
WHERE P.cid = C.cid AND P.price >= 200)
```
3. SUBQUERIES IN WHERE

Find all companies s.t. all their products have price < 200

Using **ALL**:

```sql
SELECT DISTINCT C.cname 
FROM Company C 
WHERE 200 >= ALL (SELECT price 
FROM Product P 
WHERE P.cid = C.cid)
```
3. Subqueries in WHERE

Find all companies s.t. all their products have price < 200

Using **ALL**:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE 200 >= ALL (SELECT price
                    FROM Product P
                    WHERE P.cid = C.cid)
```

Not supported in sqlite
QUESTION FOR DATABASE THEORY FANS AND THEIR FRIENDS

Can we unnest the *universal quantifier* query?

We need to first discuss the concept of *monotonicity*
MONOTONE QUERIES

Definition A query $Q$ is **monotone** if:

- Whenever we add tuples to one or more input tables, the answer to the query will not lose any of the tuples

Product $(\text{pname}, \text{price}, \text{cid})$
Company $(\text{cid}, \text{cname}, \text{city})$
MONOTONE QUERIES

Definition: A query $Q$ is **monotone** if:

- Whenever we add tuples to one or more input tables, the answer to the query will not lose any of the tuples.

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pname</strong></td>
<td><strong>cid</strong></td>
</tr>
<tr>
<td>Gizmo</td>
<td>c001</td>
</tr>
<tr>
<td>Gadget</td>
<td>c004</td>
</tr>
<tr>
<td>Camera</td>
<td>c003</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>pname</strong></th>
<th><strong>city</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Lyon</td>
</tr>
<tr>
<td>Camera</td>
<td>Lodtz</td>
</tr>
</tbody>
</table>
Monotone Queries

Definition: A query $Q$ is **monotone** if:

- Whenever we add tuples to one or more input tables, the answer to the query will not lose any of the tuples.

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pname</strong></td>
<td><strong>cid</strong></td>
</tr>
<tr>
<td>Gizmo</td>
<td>c001</td>
</tr>
<tr>
<td>Gadget</td>
<td>c004</td>
</tr>
<tr>
<td>Camera</td>
<td>c003</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pname</strong></td>
<td><strong>city</strong></td>
</tr>
<tr>
<td>Gizmo</td>
<td>Lyon</td>
</tr>
<tr>
<td>Camera</td>
<td>Lodtz</td>
</tr>
</tbody>
</table>

So far it looks monotone...
MONOTONE QUERIES

Definition A query $Q$ is **monotone** if:

- Whenever we add tuples to one or more input tables, the answer to the query will not lose any of the tuples.
MONOTONE QUERIES

**Theorem**: If $Q$ is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it is monotone.
MONOTONE QUERIES

Theorem: If $Q$ is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it is monotone.

Proof. We use the nested loop semantics: if we insert a tuple in a relation $R_i$, this will not remove any tuples from the answer.

```
SELECT $a_1, a_2, \ldots, a_k$
FROM $R_1$ AS $x_1$, $R_2$ AS $x_2$, \ldots, $R_n$ AS $x_n$
WHERE Conditions
```

```
for $x_1$ in $R_1$ do
  for $x_2$ in $R_2$ do
    \ldots
    for $x_n$ in $R_n$ do
      if Conditions
        output $(a_1,\ldots,a_k)$
```
MONOTONE QUERIES

The query:

Find all companies s.t. all their products have price < 200

is not monotone
MONOTONE QUERIES

The query:

Find all companies s.t. all their products have price < 200

is not monotone

<table>
<thead>
<tr>
<th>pname</th>
<th>price</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>c001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>c001</td>
<td>Sunworks</td>
<td>Bonn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunworks</td>
</tr>
</tbody>
</table>
MONOTONE QUERIES

The query:

Find all companies s.t. all their products have price < 200

Consequence: If a query is not monotonic, then we cannot write it as a SELECT-FROM-WHERE query without nested subqueries.
QUERIES THAT MUST BE NESTED

Queries with universal quantifiers or with negation
QUERIES THAT MUST BE NESTED

Queries with universal quantifiers or with negation

Queries that use aggregates in certain ways

- `sum(..) and count(*)` are NOT monotone, because they do not satisfy set containment
- `select count(*) from R` is not monotone!
ANNOUNCEMENTS

You received invitation email to @cs
You will be prompted to choose passwd

- Problems with existing account?
- In the worst case we will ask you to create a new @outlook account just for this class

If OK, create the database server

- Choose cheapest pricing tier!

Remember: WQ2 due on Friday
GROUP BY V.S.
NESTED QUERIES

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  y.price > 1)
                AS  TotalSales
FROM      Purchase  x
WHERE     x.price > 1

Why twice?
Author(login, name)
Wrote(login, url)

MORE UNNESTING

Find authors who wrote ≥ 10 documents:
MORE UNNESTING

Find authors who wrote $\geq 10$ documents:

Attempt 1: with nested queries

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

This is SQL by a novice.
MORE UNNESTING

Find authors who wrote \( \geq 10 \) documents:

**Attempt 1:** with nested queries

**Attempt 2:** using `GROUP BY` and `HAVING`

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

This is SQL by an expert.
Product (pname, price, cid)
Company (cid, cname, city)

FINDING WITNESSES

For each city, find the most expensive product made in that city
Product (pname, price, cid)  
Company (cid, cname, city)

**FINDING WITNESSES**

For each city, find the most expensive product made in that city

Finding the maximum price is easy...

```
SELECT x.city, max(y.price)  
FROM Company x, Product y  
WHERE x.cid = y.cid  
GROUP BY x.city;
```

But we need the witnesses, i.e., the products with max price
Product (pname, price, cid)
Company (cid, cname, city)

FINDING WITNESSES

To find the witnesses, compute the maximum price in a subquery (in FROM or in WITH)

WITH CityMax AS
  (SELECT x.city, max(y.price) as maxprice
   FROM Company x, Product y
   WHERE x.cid = y.cid
   GROUP BY x.city)
SELECT DISTINCT u.city, v.pname, v.price
FROM Company u, Product v, CityMax w
WHERE u.cid = v.cid
  and u.city = w.city
  and v.price = w.maxprice;
To find the witnesses, compute the maximum price in a subquery (in FROM or in WITH)

```
SELECT DISTINCT u.city, v.pname, v.price
FROM Company u, Product v,
  (SELECT x.city, max(y.price) as maxprice
   FROM Company x, Product y
   WHERE x.cid = y.cid
   GROUP BY x.city) w
WHERE u.cid = v.cid
  and u.city = w.city
  and v.price = w.maxprice;
```
Product (pname, price, cid)
Company (cid, cname, city)

FINDING WITNESSES

Or we can use a subquery in where clause

```sql
SELECT u.city, v.pname, v.price
FROM Company u, Product v
WHERE u.cid = v.cid
  and v.price >= ALL (SELECT y.price
                       FROM Company x, Product y
                       WHERE u.city=x.city
                       and x.cid=y.cid);
```

CSE 344 - 2017au
There is a more concise solution here:

```sql
SELECT u.city, v.pname, v.price
FROM Company u, Product v, Company x, Product y
WHERE u.cid = v.cid AND u.city = x.city
    AND x.cid = y.cid
GROUP BY u.city, v.pname, v.price
HAVING v.price = max(y.price)
```