# Database Management Systems
# CSEP 544

## Lecture 2: SQL

# Announcements

- HW1 due tonight (11:59pm)
- PA2 & HW2 released
- Fill out HW3 email account form by tonight!
- Final information posted on piazza
- Check website for up to date OH info

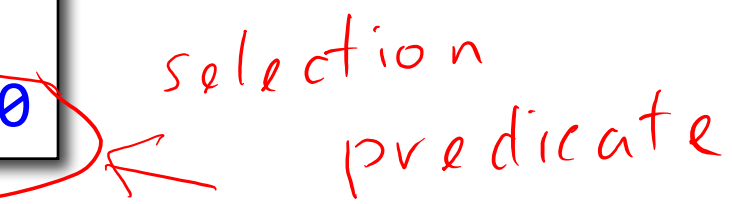
# Review

- Data models
  - Instance
  - Schema
  - Language
- Relational data model
  - Relations are flat
  - Tuples are not ordered
- Logical and physical data independence

# Reading Assignment 1

# Selections in SQL

```
SELECT *
FROM   Product
WHERE  price > 100.0
```




# Projections in SQL

```
SELECT CName
FROM   Product
```

```
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

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 pname, price
FROM   Product, Company
WHERE  ...
```

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 pname, price
FROM   Product, Company
WHERE  manufacturer=cname AND
       country='Japan' AND price < 150
```

join predicate

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
```

tuple vars

P.

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 USA companies that manufacture "gadget" products

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 USA companies that manufacture "gadget" products

```
SELECT DISTINCT cname
FROM   Product, Company
WHERE  country='USA' AND category = 'gadget'
       AND manufacturer = cname
```

Why DISTINCT?

# Joins in SQL

- The standard join in SQL 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

# Joins and Aggregates

# (Inner) joins

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

```
SELECT DISTINCT cname
FROM   Product, Company
WHERE  country='USA' AND category = 'gadget'
       AND manufacturer = cname
```

# (Inner) joins

```
SELECT DISTINCT cname
FROM   Product, Company
WHERE  country='USA' AND category = 'gadget'
       AND manufacturer = cname
```

- Product

| pname | category | manufacturer |
|---|---|---|
| Gizmo | gadget | GizmoWorks |
| Camera | Photo | Hitachi |
| OneClick | Photo | Hitachi |

- Company

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

# (Inner) joins

```
SELECT DISTINCT cname
FROM   Product, Company
WHERE  country='USA' AND category = 'gadget'
       AND manufacturer = cname
```

- Product

| pname | category | manufacturer |
|---|---|---|
| Gizmo | gadget | GizmoWorks |
| Camera | Photo | Hitachi |
| OneClick | Photo | Hitachi |

- Company

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

# (Inner) joins

```
SELECT DISTINCT cname
FROM   Product, Company
WHERE  country='USA' AND category = 'gadget'
       AND manufacturer = cname
```

- Product

| pname | category | manufacturer |
|---|---|---|
| Gizmo | gadget | GizmoWorks |
| Camera | Photo | Hitachi |
| OneClick | Photo | Hitachi |

- Company

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

# (Inner) joins

```
SELECT DISTINCT cname
FROM   Product, Company
WHERE  country='USA' AND category = 'gadget'
       AND manufacturer = cname
```

- Product

| pname | category | manufacturer |
|---|---|---|
| Gizmo | gadget | GizmoWorks |
| Camera | Photo | Hitachi |
| OneClick | Photo | Hitachi |

- Company

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

| pname | category | manufacturer | cname | country |
|---|---|---|---|---|
| Gizmo | gadget | GizmoWorks | GizmoWorks | USA |

# (Inner) joins

```
SELECT DISTINCT cname
FROM   Product, Company
WHERE  country='USA' AND category = 'gadget'
       AND manufacturer = cname
```

- Product

| pname | category | manufacturer |
|---|---|---|
| Gizmo | gadget | GizmoWorks |
| Camera | Photo | Hitachi |
| OneClick | Photo | Hitachi |

- Company

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

# (Inner) joins

```
SELECT DISTINCT cname
FROM   Product, Company
WHERE  country='USA' AND category = 'gadget'
       AND manufacturer = cname
```

- Product

| pname | category | manufacturer |
|---|---|---|
| Gizmo | gadget | GizmoWorks |
| Camera | Photo | Hitachi |
| OneClick | Photo | Hitachi |

- Company

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

# (Inner) joins

```
SELECT DISTINCT cname
FROM   Product, Company
WHERE  country='USA' AND category = 'gadget'
       AND manufacturer = cname
```

```
SELECT DISTINCT cname
FROM   Product JOIN Company ON
       country = 'USA' AND category = 'gadget'
       AND manufacturer = cname
```

where

# (Inner) Joins

```
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)
```

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

# Another example

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

Retrieve all Japanese companies that manufacture products in both 'gadget' and 'photography' categories

# Another example

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

Retrieve all Japanese companies that manufacture products in both ‘gadget’ and ‘photography’ categories

```
SELECT DISTINCT cname
FROM Product P1, Product P2, Company
WHERE country = 'Japan' AND P1.category = 'gadget'
      AND P2.category = 'photography'
      AND P1.manufacturer = cname
      AND P2.manufacturer = cname;
```

# Self-Joins and Tuple Variables

- Find all companies that manufacture both products in the 'gadgets' and 'photo' category
- Joining Product with Company is insufficient: need to join Product, with Product, and with Company

- When a relation occurs twice in the FROM clause we call it a *self-join*
  - in that case we must use tuple variables (why?)

# Self-joins

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

## Product

| pname | category | manufacturer |
|---|---|---|
| Gizmo | gadget | GizmoWorks |
| SingleTouch | photo | Hitachi |
| MultiTouch | Photo | GizmoWorks |

## Company

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

# Self-joins

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

Product

x

| pname | category | manufacturer |
|---|---|---|
| Gizmo | gadget | GizmoWorks |
| SingleTouch | photo | Hitachi |
| MultiTouch | Photo | GizmoWorks |

Company

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

# Self-joins

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

Product

x

y

| pname | category | manufacturer |
|---|---|---|
| Gizmo | gadget | GizmoWorks |
| SingleTouch | photo | Hitachi |
| MultiTouch | Photo | GizmoWorks |

Company

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

# Self-joins

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

Product

x

y

| pname | category | manufacturer |
|---|---|---|
| Gizmo | gadget | GizmoWorks |
| SingleTouch | photo | Hitachi |
| MultiTouch | Photo | GizmoWorks |

Company

z

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

# Self-joins

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

Product

x

y

| pname | category | manufacturer |
|---|---|---|
| Gizmo | gadget | GizmoWorks |
| SingleTouch | photo | Hitachi |
| MultiTouch | Photo | GizmoWorks |

Company

z

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

# Self-joins

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

Product

x

y

| pname | category | manufacturer |
|---|---|---|
| Gizmo | gadget | GizmoWorks |
| SingleTouch | photo | Hitachi |
| MultiTouch | Photo | GizmoWorks |

Company

z

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

# Self-joins

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

z.cname

Product

| pname | category | manufacturer |
|---|---|---|
| Gizmo | gadget | GizmoWorks |
| SingleTouch | photo | Hitachi |
| MultiTouch | Photo | GizmoWorks |

Company

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

| x.pname | x.category | x.manufacturer | y.pname | y.category | y.manufacturer | z.cname | z.country |
|---|---|---|---|---|---|---|---|
| Gizmo | gadget | GizmoWorks | MultiTouch | Photo | GizmoWorks | GizmoWorks | USA |

# Self-joins

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

Product

x

y

| pname | category | manufacturer |
|---|---|---|
| Gizmo | gadget | GizmoWorks |
| SingleTouch | photo | Hitachi |
| MultiTouch | Photo | GizmoWorks |

Company

z

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

| x.pname | x.category | x.manufacturer | y.pname | y.category | y.manufacturer | z.cname | z.country |
|---|---|---|---|---|---|---|---|
| Gizmo | gadget | GizmoWorks | MultiTouch | Photo | GizmoWorks | GizmoWorks | USA |

# Outer joins

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

-- prodName is foreign key
```

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

We want to include products that are never sold, but some are not listed! Why?

# Outer joins

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

-- prodName is foreign key
```

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

```
SELECT Product.name, Purchase.store
FROM   Product 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 |

```
SELECT Product.name, Purchase.store
FROM   Product 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 |

```
SELECT Product.name, Purchase.store
FROM   Product 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 |

Output

| Name | Store |
|---|---|
| Gizmo | Wiz |

```
SELECT Product.name, Purchase.store
FROM   Product 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 |

Output

| Name | Store |
|---|---|
| Gizmo | Wiz |

```
SELECT Product.name, Purchase.store
FROM   Product 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 |

Output

| Name | Store |
|---|---|
| Gizmo | Wiz |

```
SELECT Product.name, Purchase.store
FROM   Product 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 |

Output

| Name | Store |
|---|---|
| Gizmo | Wiz |

```
SELECT Product.name, Purchase.store
FROM   Product 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 |

Output

| Name | Store |
|---|---|
| Gizmo | Wiz |
| Camera | Ritz |

```
SELECT Product.name, Purchase.store
FROM   Product 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 |

Output

| Name | Store |
|---|---|
| Gizmo | Wiz |
| Camera | Ritz |
| Camera | Wiz |

```
SELECT Product.name, Purchase.store
FROM   Product 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 |

Output

| Name | Store |
|---|---|
| Gizmo | Wiz |
| Camera | Ritz |
| Camera | Wiz |

```
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 |

Output

| Name | Store |
|---|---|
| Gizmo | Wiz |
| Camera | Ritz |
| Camera | Wiz |

```
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 |

Output

| Name | Store |
|---|---|
| Gizmo | Wiz |
| Camera | Ritz |
| Camera | Wiz |
| OneClick | NULL |

```
SELECT Product.name, Purchase.store
FROM   Product FULL 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 |
| Phone | Foo |

Output

| Name | Store |
|---|---|
| Gizmo | Wiz |
| Camera | Ritz |
| Camera | Wiz |
| OneClick | NULL |
| NULL | Foo |

# 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`

# Comment about SQLite

- 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 attribute

# Aggregates and NULL Values

Null values are not used in aggregates

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

Try the following at home:

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

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

What do they mean ?

# Grouping and Query Evaluation

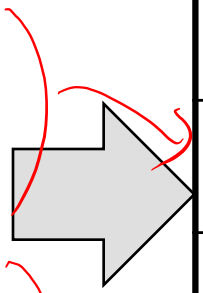
# Grouping and Aggregation

```
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 mean ?

# 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 |

```
SELECT   product, quantity
FROM     Purchase
GROUP BY product
```

# Need to be Careful…

```
SELECT   product
FROM     Purchase
GROUP BY product
```

```
SELECT   quantity
FROM     Purchase
```

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

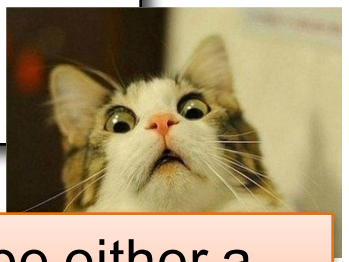
| Product |
|---|
| Bagel |
| Banana |

+

???

| Quantity |
|---|
| 20 |
| 20 |
| 50 |
| 10 |
| 10 |

```
SELECT   product, quantity
FROM     Purchase
GROUP BY product
```



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
```

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

sqlite is WRONG on this query.

Advanced DBMS (e.g. SQL Server) gives an error

# 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

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 |

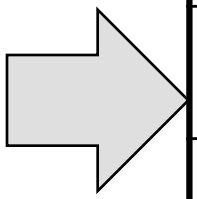
WHERE price > 1

```
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

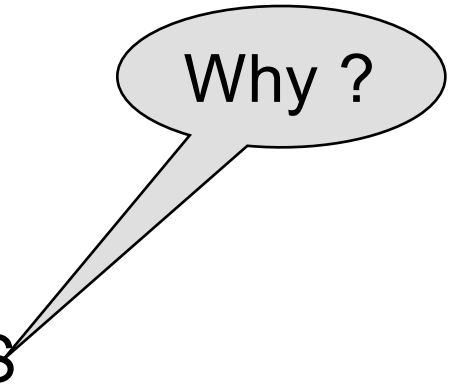
Same query as before, except that we consider only products that had at least 30 sales.

```
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

```
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”

```
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”

```
FROM      Purchase
```

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”

```
FROM        Purchase
GROUP BY    month
```

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"

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

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
```

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)
```

# 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
```

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

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

```
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: price is in Purchase, manufacturer 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
```

| 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: price is in Purchase, manufacturer 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
```

| 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

# Nested Queries

# What have we learned so far

- Data models
- Relational data model
  - Instance: relations
  - Schema: table with attribute names
  - Language: SQL

# What have we learned so far

- SQL features
  - Projections
  - Selections
  - 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 writing nested queries when possible
  - But sometimes it's impossible, as we will see

# Subqueries…

- Can appear as computed values in a `SELECT` clause
- Can appear in `FROM` clauses and aliased using a tuple variable that represents the tuples in the result of the subquery
- Can return a single constant to be compared with another value in a `WHERE` clause
- Can return relations to be used in `WHERE` clauses

# 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 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
```

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

Try unnest this query !

# 2. Subqueries in FROM

At the end of the lecture we will see that sometimes we really need a subquery and one option will be to put it in the `FROM` clause.

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

# 3. Subqueries in WHERE

Find all companies that make some products with price < 200

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

# 3. Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers

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

# 3. Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers

---

Using EXISTS:

```
SELECT DISTINCT  C.cname
FROM    Company C
WHERE   EXISTS (SELECT *
                FROM Product P
                WHERE C.cid = P.cid and P.price < 200)
```

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

# 3. Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers

---

Using IN

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

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

# 3. Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers

---

Using ANY:

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

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

# 3. Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers

---

Using ANY:

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

Not supported in sqlite

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

# 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
```

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

# 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! ☺

```
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

```
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

```
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

Universal quantifiers are hard! ☹

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

# 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)
```

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

# 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)
```

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

# 3. Subqueries in WHERE

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

Universal quantifiers

---

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)
```

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

# 3. Subqueries in WHERE

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

Universal quantifiers

---

Using ALL:

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

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

# 3. Subqueries in WHERE

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

Universal quantifiers

---

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*

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

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

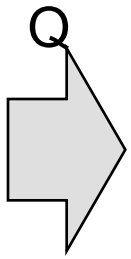
---

Product

| pname | price | cid |
|---|---|---|
| Gizmo | 19.99 | c001 |
| Gadget | 999.99 | c004 |
| Camera | 149.99 | c003 |

Company

| cid | cname | city |
|---|---|---|
| c002 | Sunworks | Bonn |
| c001 | DB Inc. | Lyon |
| c003 | Builder | Lodtz |




| pname | city |
|---|---|
| Gizmo | Lyon |
| Camera | Lodtz |

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

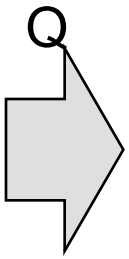
---

Product

| pname | price | cid |
|---|---|---|
| Gizmo | 19.99 | c001 |
| Gadget | 999.99 | c004 |
| Camera | 149.99 | c003 |

Company

| cid | cname | city |
|---|---|---|
| c002 | Sunworks | Bonn |
| c001 | DB Inc. | Lyon |
| c003 | Builder | Lodtz |




| pname | city |
|---|---|
| Gizmo | Lyon |
| Camera | Lodtz |

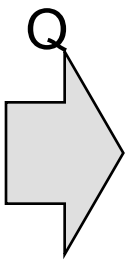
---

Product

| pname | price | cid |
|---|---|---|
| Gizmo | 19.99 | c001 |
| Gadget | 999.99 | c004 |
| Camera | 149.99 | c003 |
| iPad | 499.99 | c001 |

Company

| cid | cname | city |
|---|---|---|
| c002 | Sunworks | Bonn |
| c001 | DB Inc. | Lyon |
| c003 | Builder | Lodtz |




| pname | city |
|---|---|
| Gizmo | Lyon |
| Camera | Lodtz |
| iPad | Lyon |

# 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 a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions
```

```
for x1 in R1 do
  for x2 in R2 do
    …
    for xn in Rn do
      if Conditions
      output (a1,…,ak)
```

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

# Monotone Queries

- The query:

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

is not monotone

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

# Monotone Queries

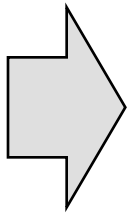
- The query:

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

is not monotone

| pname | price | cid |
|---|---|---|
| Gizmo | 19.99 | c001 |

| cid | cname | city |
|---|---|---|
| c001 | Sunworks | Bonn |



| cname |
|---|
| Sunworks |

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

# Monotone Queries

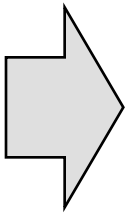
- The query:

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

is not monotone

| pname | price | cid |
|---|---|---|
| Gizmo | 19.99 | c001 |

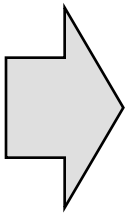
| cid | cname | city |
|---|---|---|
| c001 | Sunworks | Bonn |



| cname |
|---|
| Sunworks |

| pname | price | cid |
|---|---|---|
| Gizmo | 19.99 | c001 |
| Gadget | 999.99 | c001 |

| cid | cname | city |
|---|---|---|
| c001 | Sunworks | Bonn |



| cname |
|---|
| |

- 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 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!