L06: SQL: Advanced

CS3200 Database design (fa18 s2)

https://northeastern-datalab.github.io/cs3200/

Version 9/24/2018

Announcements!

- Recall Exam 1 on THU Oct 4
  - Topic: SQL on your own computer and PostgreSQL instance
  - look at HW10 on BB before
  - 0 point practice exam on MON 1 (just practicing test modalities)
- HW3: you will have to draw your own schema
- Student feedback: "too much handholding"
- Ask questions in class if you can't follow

- Student poll: 2nd half of next class: SQL practice or database design?

A word on capitalization

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

Q: Find all US companies that manufacture products in the 'Gadgets' category!

```
SELECT   cname
FROM     Product P, Company
WHERE    country = 'USA'
AND      P.category = 'Gadgets'
AND      P.manufacturer = cname
```

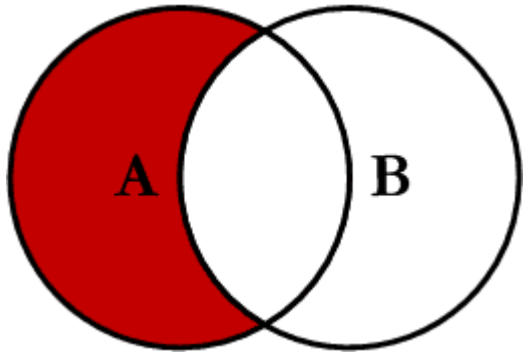
My recommendation for capitalization

1. SQL keywords in ALL CAPS,
2. Table names and aliases with Initial Caps
3. Column names all in lowercase.

PostgreSQL treats all in lowercase.
Except if you write:
create table "Product" (…)
This will preserve capitalization of table name
But … you need to always use quotations

More information: http://blog.lerner.co.il/quoting-postgresql/ , https://stackoverflow.com/questions/6331504/omitting-the-double-quote-to-do-query-on-postgresql

Let's practice




```
SELECT <select_list>
FROM A
LEFT JOIN B
ON A.key = B.key
WHERE B.key IS NULL
```

English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

```
SELECT eText, eid
FROM English
LEFT JOIN French
ON eid = fid
WHERE fid IS NULL
```

```
SELECT *
FROM English
WHERE eid NOT IN
    (SELECT fid
    FROM French)
```

Source: http://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins

Missing sales

Item(name, category)
Purchase(iName, store, month)

An "inner join":

```
SELECT  Item.name, Purchase.store
FROM    Item, Purchase
WHERE   Item.name = Purchase.iName
```

Same as:

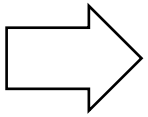
```
SELECT  Item.name, Purchase.store
FROM    Item JOIN Purchase ON
        Item.name = Purchase.iName
```

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9



Missing sales

Item(name, category)
Purchase(iName, store, month)

An "inner join":

```
SELECT  Item.name, Purchase.store
FROM    Item, Purchase
WHERE   Item.name = Purchase.iName
```

Same as:

```
SELECT  Item.name, Purchase.store
FROM    Item JOIN Purchase ON
        Item.name = Purchase.iName
```

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

Result

Name	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Products that never sold will be lost ☹

Missing sales

Item(<u>name</u>, category)
Purchase(iName, store, month)

An "inner join":

```
SELECT  Item.name, Purchase.store
FROM    Item, Purchase
WHERE   Item.name = Purchase.iName
```

Same as:

```
SELECT  Item.name, Purchase.store
FROM    Item INNER JOIN Purchase ON
        Item.name = Purchase.iName
```

"INNER JOIN" same as "JOIN"

What if you want to include never-sold products?

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

Result

Name	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Products that never sold will be lost ☹

Outer Joins

Item(name, category)
Purchase(iName, store, month)

"LEFT OUTER JOIN"
same as
"LEFT JOIN"

If we want to include the never-sold products, then we need an "outer join":

```
SELECT  Item.name, Purchase.store
FROM    Item LEFT OUTER JOIN Purchase ON
        Item.name = Purchase.iName
```

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

Result

Name	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz
OneClick	NULL

Now we include those products ☺

Outer Joins w/ selection

Item(name, category)
Purchase(iName, store, month)

Explanation: the filter ("month = 9") applies to the result of the outer join. Any tuple that has NULL as month, does not pass the filter

Same question, but now only for those sold in month = 9:

```
SELECT  Item.name, Purchase.store
FROM    Item LEFT OUTER JOIN Purchase ON
        Item.name = Purchase.iName
WHERE   month = 9
```

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

Result

Name	Store
Camera	Wiz

The products disappeared *despite* outer join ☹

Outer Joins w/ selection

Item(name, category)
Purchase(iName, store, month)

Explanation: now the filter ("month = 9") applies to the right side of the left join *before* joining. NULLs are appended only after filter, during join

Same question, but now only for those sold in month = 9:

```
SELECT  Item.name, Purchase.store
FROM    Item LEFT OUTER JOIN Purchase ON
        (Item.name = Purchase.iName
AND     month = 9)
```

parenthesis not required, and just for illustration

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9

Result

Name	Store
Camera	Wiz
Gizmo	NULL
OneClick	NULL

Now they are back again ☺

Outer Joins w/ selection

Item(name, category)
Purchase(iName, store, month)

Explanation: now the filter ("month = 9") applies to the right side of the left join *before* joining. NULLs are appended only after filter, during join

Same question, but now only for those sold in month = 9:

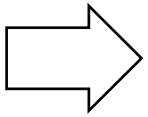
```
SELECT  Item.name, P.store
FROM    Item LEFT OUTER JOIN
        (SELECT iName, store FROM Purchase WHERE month = 9) P
on      Item.name = P.iName
```

Item

Name	Category
Gizmo	Gadget
Camera	Photo
OneClick	Photo

Purchase

iName	Store	Month
Gizmo	Wiz	8
Camera	Ritz	8
Camera	Wiz	9



Result

Name	Store
Camera	Wiz
Gizmo	NULL
OneClick	NULL

Now they are back again ☺

Empty Group Problem

Item(name, category)
Purchase(iName, store, month)

Compute, for each product, the total number of sales in Sept (= month 9)

```
SELECT    name, count(*)
FROM      Item, Purchase
WHERE     name = iName
    and   month = 9
GROUP BY  name
```

What's wrong?

Empty Group Problem

Item(name, category)
Purchase(iName, store, month)

Compute, for each product, the total number of sales in Sept (= month 9)

We need to use an attribute from "Purchase" to get the correct 0 count. Try "name" from "Item".

```
SELECT    name, count(store)
FROM      Item LEFT JOIN Purchase ON
          name = iName
    and   month = 9
GROUP BY  name
```

What happens if you add "sum(month)" to the SELECT clause?

Now we also get the products with 0 sales

Empty Group Problem

Item(name, category)
Purchase(iName, store, month)

Compute, for each product, the total number of sales in Sept (= month 9)

We need to use an attribute from "Purchase" to get the correct 0 count. Try "name" from "Item".

```
SELECT    x.name, count(y.store)
FROM      Item x LEFT OUTER JOIN Purchase y ON
          x.name = y.iName
    and   y.month = 9
GROUP BY  x.name
```

Now we also get the products with 0 sales

Coalesce function

M

a
1
2

N

a
2
3

```
SELECT M.a,
       COUNT(N.a) ct,
       SUM(N.a) su

FROM M
Left JOIN N ON M.a = N.a
GROUP BY M.a
```

Result

a	ct	su
1	0	NULL
2	1	2

Coalesce function

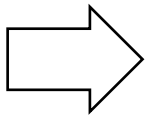
M

a
1
2

N

a
2
3

```
SELECT M.a,
       COUNT(N.a) ct,
       SUM(N.a) su,
       SUM(COALESCE(N.a, 0)) cosu
FROM M
Left JOIN N ON M.a = N.a
GROUP BY M.a
```



Result

a	ct	su	cosu
1	0	NULL	0
2	1	2	2

COALESCE: takes first non-NULL value

```
SELECT COALESCE(1, NULL)
```

```
SELECT COALESCE(NULL, 3)
```

```
SELECT COALESCE(1, 2)
```

Coalesce function

M

a
1
2

N

a
2
3

```
SELECT M.a, N.a, COALESCE(M.a, N.a) as b
FROM M
FULL JOIN N
ON M.a = N.a
```

Result

M.a	N.a	b
1	NULL	1
2	2	2
NULL	3	3

COALESCE: takes first non-NULL value

Natural Outer Join

M

a
1
2

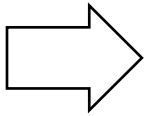
N

a
2
3

```
SELECT a
FROM M
NATURAL FULL JOIN N
```

```
NATURAL FULL JOIN
models "coalesce"
```

Result



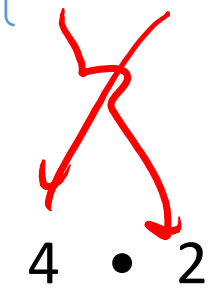
a
1
2
3

Commutativity & Associativity

Multiplication

$(3 \cdot 2) \cdot 4 = 24$

$3 \cdot (2 \cdot 4) = 24$



Multiplication is associative ☺

and commutative ☺

Matrix multiplication

$$\left(\begin{bmatrix} 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \right) \cdot \begin{bmatrix} 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 49 \end{bmatrix}$$

$$\begin{bmatrix} 11 & 16 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 \end{bmatrix} \cdot \left(\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 49 \end{bmatrix}$$

$$\begin{bmatrix} 5 \\ 13 \end{bmatrix}$$

Matrix multipl. is associative ☺

$$\begin{bmatrix} 3 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

#col ≠ #row

... but *not* commutative ☹

Commutativity & Associativity

Outer joins

R

A	B
1	2

S

B	C
2	3

T

A	C
4	5

```
SELECT A, B, C
FROM (R
NATURAL FULL JOIN S)
NATURAL FULL JOIN T
```

Result

A	B	C
5	NULL	4
1	2	3

```
SELECT  A, B, C
FROM R
NATURAL FULL JOIN (S
NATURAL FULL JOIN T)
```

Result

A	B	C
5	NULL	4
NULL	2	3
1	2	NULL

Outer Joins: summary

- Left (outer) join:
  - Include the left tuple even if there's no match
- Right (outer) join:
  - Include the right tuple even if there's no match
- Full (outer) join:
  - Include both left and right tuples even if there's no match

- (inner) join:
  - Include only the matches

Theta joins

Theta joins

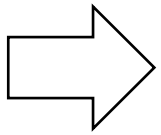
305

What do these queries compute?

R
a
1
2

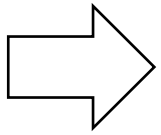
U
a
2
3
4

```
SELECT   R.a, U.a as b
FROM     R, U
WHERE    R.a < U.a
```



?

```
SELECT   R.a, U.a as b
FROM     R, U
WHERE    R.a >= U.a
```



?

A **Theta-join** allows for arbitrary comparison relationships (such as ≥).
An **equijoin** is a theta join using the equality operator.

Theta joins

305

R

a
1
2

U

a
2
3
4

What do these queries compute?

```
SELECT   R.a, U.a as b
FROM     R, U
WHERE    R.a < U.a
```

a	b
1	2
1	3
1	4
2	3
2	4

```
SELECT   R.a, U.a as b
FROM     R, U
WHERE    R.a >= U.a
```

a	b
2	2

A **Theta-join** allows for arbitrary comparison relationships (such as ≥).
An **equijoin** is a theta join using the equality operator.

Processing Multiple Tables–Joins

- **Join**: a relational operation that causes two or more tables with a common domain to be combined into a single table or view
- **Equi-join**: a join in which the joining condition is based on equality between values in the common columns; common columns appear redundantly in the result table
- A **Theta-join** allows for arbitrary comparison relationships (e.g., ≥). An equijoin is a theta join using the equality operator.
- **Natural join**: an equi-join in which one of the duplicate columns is eliminated in the result table

The common columns in joined tables are usually the primary key of the dominant table and the foreign key of the dependent table in 1:M relationships

Processing Multiple Tables–Joins

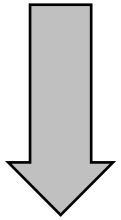
- **Left Outer join**: a join in which rows from the left table that do not have matching values in common columns are nonetheless included in the result table (as opposed to inner join, in which rows must have matching values in order to appear in the result table)

- Union join ("**Full outer join**"): includes all columns from each table in the join, and an instance for each row of each table

WITH clause

WITH clause: temporary relations


```
SELECT  pname, price
FROM    Product
WHERE   price =
        (SELECT  max(price)
        FROM     Product)
```



```
WITH    Max_price(value) as
        (SELECT  max(price)
        FROM     Product)
SELECT  pname, price
FROM    Product, Max_price
WHERE   price = value
```

Product (pname, price, cid)

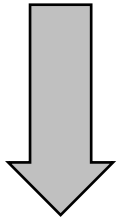
The **WITH** clause defines a temporary relation that is available only to the query in which it occurs. Sometimes easier to read. Very useful for queries that need to access the same intermediate result multiple times

WITH clause: temporary relations


```
SELECT  pname, price
FROM    Product
WHERE   price =
        (SELECT  max(price)
        FROM     Product)
```

Product (pname, price, cid)

The **WITH** clause defines a temporary relation that is available only to the query in which it occurs. Sometimes easier to read. Very useful for queries that need to access the same intermediate result multiple times



```
WITH    Max_price as
        (SELECT  max(price) as value
        FROM     Product)
SELECT  pname, price
FROM    Product, Max_price
WHERE   price = value
```

WITH Keyword

If you need to use a query and want it to run only once, the WITH keyword lets you define one without it running multiple times (which may happen with a subquery).

WITH {*subqueryName*} AS {subquery}

SELECT …

FROM … *subqueryName*

WHERE …

Witnesses

Motivation: What are these queries supposed to return?

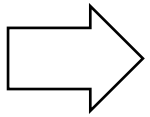
Product

PName	Price	cid
Gizmo	15	1
SuperGizmo	20	1
iTouch1	300	2
iTouch2	300	2

Company

cid	cname	city
1	GizmoWorks	Oslo
2	Apple	MountainView

Find for each company id, the maximum price amongst its products



?

Motivation: What are these queries supposed to return?

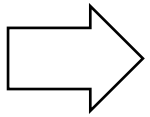
Product

PName	Price	cid
Gizmo	15	1
SuperGizmo	20	1
iTouch1	300	2
iTouch2	300	2

Company

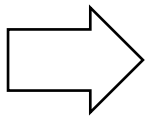
cid	cname	city
1	GizmoWorks	Oslo
2	Apple	MountainView

Find for each company id, the maximum price amongst its products



cid	mp
1	20
2	300

Find for each company id, the product with maximum price amongst its products



?

Motivation: What are these queries supposed to return?

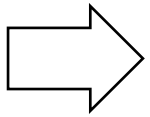
Product

PName	Price	cid
Gizmo	15	1
SuperGizmo	20	1
iTouch1	300	2
iTouch2	300	2

Company

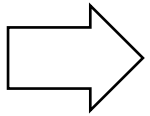
cid	cname	city
1	GizmoWorks	Oslo
2	Apple	MountainView

Find for each company id, the maximum price amongst its products



cid	mp
1	20
2	300

Find for each company id, the product with maximum price amongst its products (Recall that "group by cid" can just give us one single tuple per cid)



cid	mp	pname
1	20	SuperGizmo
2	300	iTouch1
2	300	iTouch2

Witnesses: simple (1/4)

Product (pname, price, cid)

Q: Find the most expensive product + its price
(Finding the maximum price alone would be easy)

Witnesses: simple (2/4)

Product (pname, price, cid)

Q: Find the most expensive product + its price
(Finding the maximum price alone would be easy)

Our Plan:

- 1. Compute max price in a subquery

1.
```
SELECT   max(P1.price)
FROM     Product P1
```

But we want the "witnesses," i.e. the product(s) with the max price. How do we do that?

Witnesses: simple (3/4)

Product (pname, price, cid)

Q: Find the most expensive product + its price
(Finding the maximum price alone would be easy)

Our Plan:

- 1. Compute max price in a subquery
- 2. Compute each product and its price...

2.
```
SELECT  P2.pname, P2.price
FROM    Product P2
```

1.
```
SELECT   max(P1.price)
FROM     Product P1
```

But we want the "witnesses," i.e. the product(s) with the max price. How do we do that?

Witnesses: simple (4/4)

Product (pname, price, cid)

Q: Find the most expensive product + its price
(Finding the maximum price alone would be easy)

Our Plan:

- 1. Compute max price in a subquery
- 2. Compute each product and its price...
  and compare the price with the max price

```
SELECT  P2.pname, P2.price
FROM    Product P2
WHERE   P2.price =
        (SELECT  max(P1.price)
        FROM     Product P1)
```

Witnesses: with joins (1/6)

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

Q: For each company, find the most expensive product + its price

Witnesses: with joins (2/6)

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

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company

Witnesses: with joins (2/6)

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

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company

1.
```
SELECT  max(P1.price)
FROM    Product P1
WHERE   P1.cid = 1
```

Witnesses: with joins (3/6)

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

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company
- 2. Compute each product and its price, per company

1.
```
SELECT  max(P1.price)
FROM    Product P1
WHERE   P1.cid = 1
```

Witnesses: with joins (3/6)

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

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company
- 2. Compute each product and its price, per company

2.
```
SELECT  C2.cname, P2.pname, P2.price
FROM    Company C2, Product P2
WHERE   C2.cid = P2.cid
```

1.
```
SELECT  max(P1.price)
FROM    Product P1
WHERE   P1.cid = 1
```

Witnesses: with joins (3/6)

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

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company
- 2. Compute each product and its price, per company
- 3. Compare the price with the max price

2.
```
SELECT  C2.cname, P2.pname, P2.price
FROM    Company C2, Product P2
WHERE   C2.cid = P2.cid
```

1.
```
SELECT  max(P1.price)
FROM    Product P1
WHERE   P1.cid = 1
```

Witnesses: with joins (4/6)

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

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company
- 2. Compute each product and its price, per company
- 3. Compare the price with the max price

```
SELECT  C2.cname, P2.pname, P2.price
FROM    Company C2, Product P2
WHERE   C2.cid = P2.cid
    and P2.price =
        (SELECT max(P1.price)
        FROM    Product P1
        WHERE   P1.cid = C2.cid)
```

How many aliases do we actually need?

Witnesses: with joins (5/6)

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

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute max price in a subquery for a given company
- 2. Compute each product and its price, per company
  and compare the price with the max price

```
SELECT  cname, pname, price
FROM    Company, Product
WHERE   Company.cid = Product.cid
    and price =
        (SELECT max(price)
        FROM    Product
        WHERE   cid = Company.cid)
```

We need no single alias here.

Next: can we eliminate the max operator in the inner query?

Witnesses: with joins (6/6)

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

Q: For each company, find the most expensive product + its price

Our Plan:

- 1. Compute all prices in a subquery, for a given company
- 2. Compute each product and its price, per company
  and compare the price with the all prices

```
SELECT  cname, pname, price
FROM    Company, Product
WHERE   Company.cid = Product.cid
    and price >= ALL
        (SELECT price
        FROM    Product
        WHERE   cid = Company.cid)
```

But: "ALL" does not work in SQLite ☹

Witnesses: with FROM (1/3)

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

Q: For each company, find the most expensive product + its price

Another Plan:

- 1. Create a table that lists the max price for each company id
- 2. Join this table with the remaining tables

F

1.
```
SELECT   cid, max(price) as MP
FROM     Product
GROUP    BY cid
```
) x, P, C

Finding the maximum price for each company was easy.
But we want the "witnesses", i.e. the products with max price.

Witnesses: with FROM (2/3)

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

Q: For each company, find the most expensive product + its price

Another Plan:

- 1. Create a table that lists the max price for each company id
- 2. Join this table with the remaining tables

2.
```
SELECT  C2.cname, P2.pname, X.MP
FROM    Company C2, Product P2,
        (SELECT  cid, max(price) as MP
        FROM     Product
        GROUP    BY cid) as X
WHERE   C2.cid = P2.cid
    and C2.cid = X.cid
    and P2.price = X.MP
```

Let's write the same query with a "WITH" clause

Witnesses: with FROM (3/3)

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

Q: For each company, find the most expensive product + its price

Another Plan with WITH:

- 1. Create a table that lists the max price for each company id
- 2. Join this table with the remaining tables

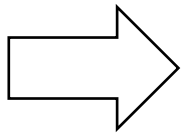
```
WITH    X(cid, MP) as
        (SELECT  cid, max(price)
        FROM     Product
        GROUP    BY cid)
SELECT  C2.cname, P2.pname, X.MP
FROM    Company C2, Product P2, X
WHERE   C2.cid = P2.cid
    and C2.cid = X.cid
    and P2.price = X.MP
```

Witnesses: with aggregates per group (1/8)

First: How to get the product that is sold with maximum price?

Purchase

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



Product	mp
Banana	4

```
SELECT   product, max(price) as mp
FROM
WHERE
GROUP BY
HAVING
```

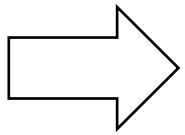
???

Witnesses: with aggregates per group (2/8)

First: How to get the product that is sold with maximum price?

Purchase *1) Find the maximum price*

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



(no name)
4

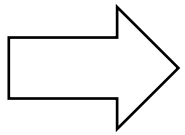
```
SELECT   max(price)
FROM     Purchase
```

Witnesses: with aggregates per group (3/8)

First: How to get the product that is sold with maximum price?

Purchase *2) Now you need to find product with price = maximum price*

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



Product	mp
Banana	4

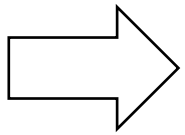
```
SELECT   P2.product, P2.price as mp
FROM     Purchase P2
WHERE    P2.price = (
             SELECT   max(price)
             FROM     Purchase
)
```

Witnesses: with aggregates per group (4/8)

First: How to get the product that is sold with maximum price?

Purchase *Another way to formulate this query*

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



Product	mp
Banana	4

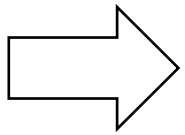
```
SELECT   P2.product, P2.price as mp
FROM     Purchase P2
WHERE    P2.price >= ALL (
             SELECT   price
             FROM     Purchase
)
```

Witnesses: with aggregates per group (5/8)

Second: How to get the product that is sold with max sales (=quanities sold).

Purchase

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



Product	sales
Banana	70

```
SELECT
FROM
WHERE
GROUP BY
HAVING
```

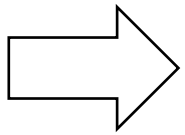
???

Witnesses: with aggregates per group (6/8)

Second: How to get the product that is sold with max sales (=quanities sold).

Purchase *1: find the total sales (sum of quantity) for each product*

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



Product	sales
Bagel	40
Banana	70

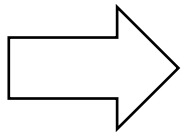
```
SELECT    product, sum(quantity) as sales
FROM      Purchase
GROUP BY product
```

Witnesses: with aggregates per group (7/8)

Second: How to get the product that is sold with max sales?

Purchase *2: we can use the same query as nested query*

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



(no name)
40
70

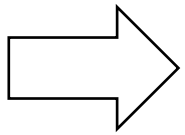
```
SELECT    sum(quantity)
FROM      Purchase
GROUP BY product
```

Witnesses: with aggregates per group (8/8)

Second: How to get the product that is sold with max sales?

Purchase *3: putting things together*

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



Product	sales
Banana	70

```
SELECT    product, sum(quantity) as sales
FROM      Purchase
GROUP BY product
HAVING    sum(quantity) >= ALL (
              SELECT    sum(quantity)
              FROM      Purchase
              GROUP BY product )
```

Next: Can you write the query without the "ALL" quanitfier?

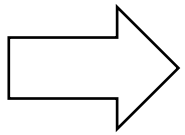
Witnesses: with aggregates per group (8/8)

Second: How to get the product that is sold with max sales?

Another way to formulate this query without "ALL"

Purchase

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



Product	sales
Banana	70

```
SELECT    product, sum(quantity) as sales
FROM      Purchase
GROUP BY  product
HAVING    sum(quantity) = 70
          (SELECT max (Q)
          FROM (SELECT    sum(quantity) Q
                FROM      Purchase
                GROUP BY  product) X )
```

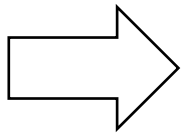
Witnesses: with aggregates per group (8/8)

Second: How to get the product that is sold with max sales?

Another way to formulate this query without "ALL"

Purchase

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



Product	sales
Banana	70

```
WITH X as
        (SELECT   product, sum(quantity) as sales
        FROM  Purchase
        GROUP BY product)
SELECT product, sales
FROM X
WHERE sales =
        (SELECT max (sales)
        FROM X)
```

Understanding nested queries

More SQL Queries

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

340

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 5.1 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 5.2 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 5.3 An Instance $B1$ of Boats

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

More nested Queries 1

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

340

Q: Find the names of sailors who have reserved a red boat.

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN
            ( SELECT R.sid
            FROM Reserves R
            WHERE R.bid IN
                        ( SELECT B.bid
                        FROM Boats B
                        WHERE B.color = 'red'))
```

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

More nested Queries 2

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

340

Q: Find the names of sailors who have reserved a boat that is not red.

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN
            ( SELECT R.sid
            FROM Reserves R
            WHERE R.bid not IN
                        ( SELECT B.bid
                        FROM Boats B
                        WHERE B.color = 'red'))
```

They must have reserved at least one boat in another color

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

More nested Queries 3

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

340

Q: Find the names of sailors who have not reserved a red boat.

```
SELECT S.sname
FROM Sailors S
WHERE S.sid not IN
            ( SELECT R.sid
            FROM Reserves R
            WHERE R.bid IN
                        ( SELECT B.bid
                        FROM Boats B
                        WHERE B.color = 'red'))
```

They can have reserved 0 or more boats in another color, but must not have reserved any red boat

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

More nested Queries 4

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

340

= Find the names of sailors who have reserved only red boats

Q: Find the names of sailors who have not reserved a boat that is not red.

```
SELECT S.sname
FROM Sailors S
WHERE S.sid not IN
            ( SELECT R.sid
            FROM Reserves R
            WHERE R.bid not IN
                        ( SELECT B.bid
                        FROM Boats B
                        WHERE B.color = 'red'))
```

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

More nested Queries 5

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

= Find the names of sailors who have reserved all red boats

Q: Find the names of sailors so there is no red boat that is not reserved by him.

```
SELECT S.sname
FROM Sailors S
WHERE not exists
        ( SELECT B.bid
        FROM Boats B
        WHERE B.color = 'red'
        AND not exists
                ( SELECT R.bid
                FROM Reserves R
                WHERE R.bid = B.bid
                AND R.sid = S.sid))
```

To understand semantics of nested queries, think of a nested loops evaluation: For each Sailors tuple, check the qualification by computing the subquery

Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

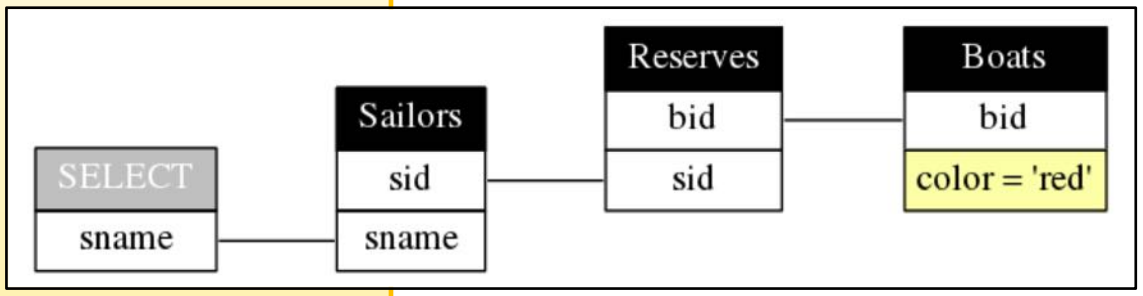
Once more: 1

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

340

Q: Find the names of sailors who have reserved a red boat.

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN
            ( SELECT R.sid
            FROM Reserves R
            WHERE R.bid IN
                        ( SELECT B.bid
                        FROM Boats B
                        WHERE B.color = 'red'))
```



Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

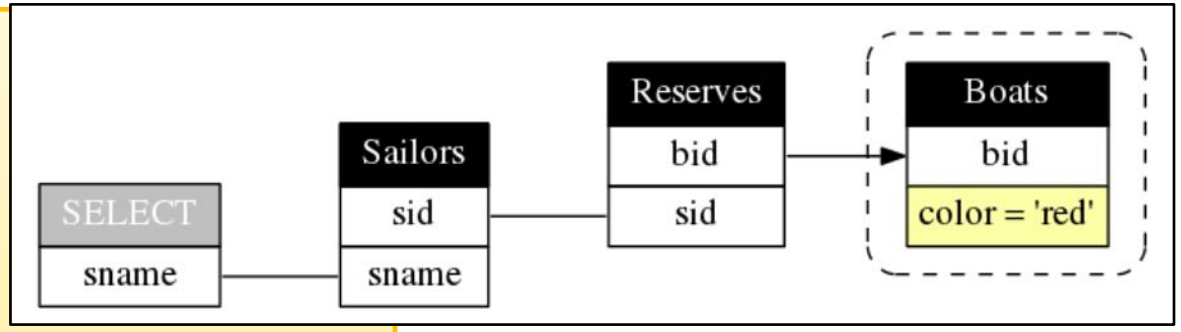
Once more: 2

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

340

Q: Find the names of sailors who have reserved a boat that is not red.

```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN
            ( SELECT R.sid
            FROM Reserves R
            WHERE R.bid not IN
                        ( SELECT B.bid
                        FROM Boats B
                        WHERE B.color = 'red'))
```



Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

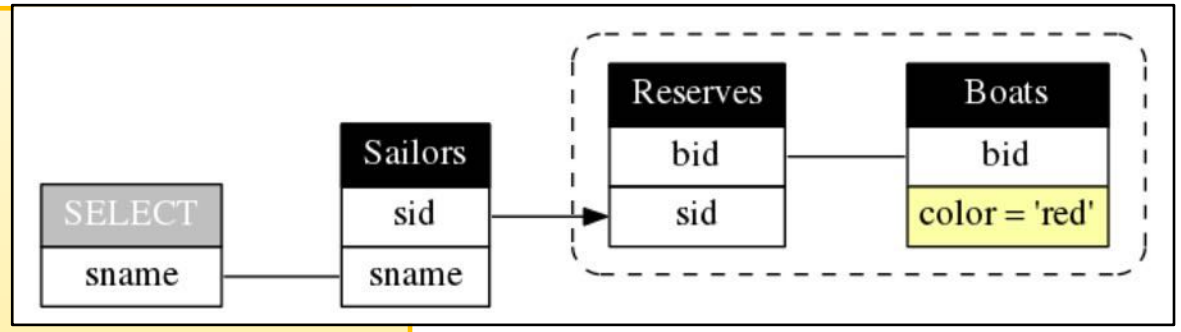
Once more: 3

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

340

Q: Find the names of sailors who have not reserved a red boat.

```
SELECT S.sname
FROM Sailors S
WHERE S.sid not IN
            ( SELECT R.sid
            FROM Reserves R
            WHERE R.bid IN
                        ( SELECT B.bid
                        FROM Boats B
                        WHERE B.color = 'red'))
```



Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

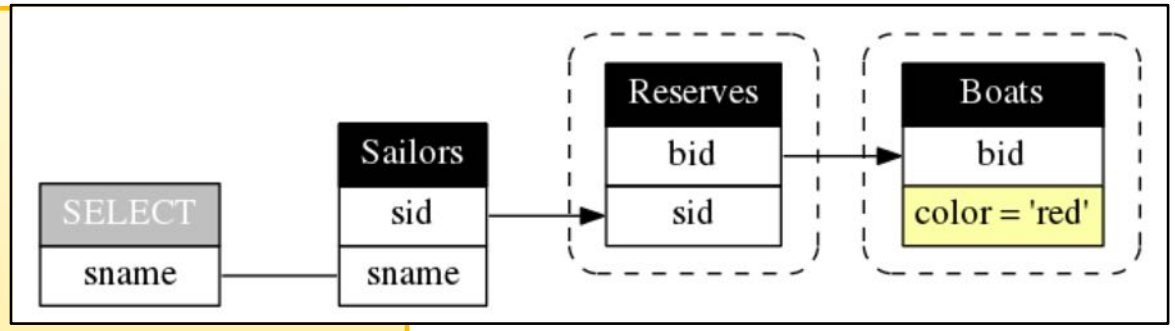
Once more: 4

Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

= Find the names of sailors who have reserved only red boats

Q: Find the names of sailors who have not reserved a boat that is not red.

```
SELECT S.sname
FROM Sailors S
WHERE S.sid not IN
            ( SELECT R.sid
            FROM Reserves R
            WHERE R.bid not IN
                        ( SELECT B.bid
                        FROM Boats B
                        WHERE B.color = 'red'))
```



Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

Once more: 5

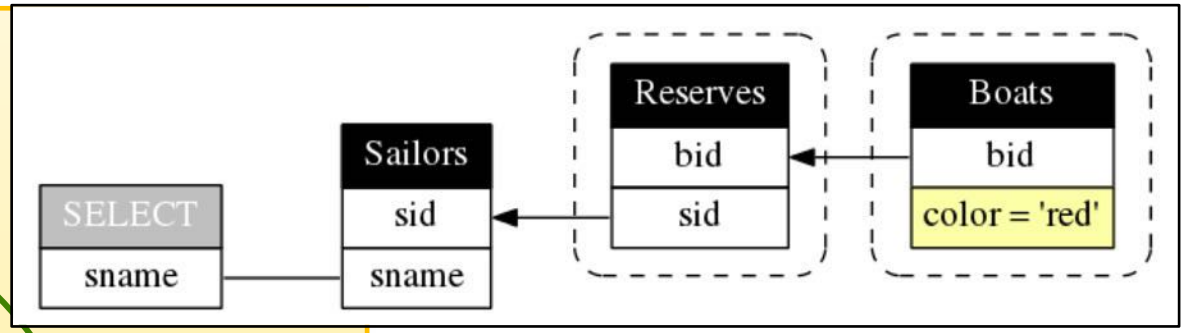
Sailors (sid, sname, rating, age)
Reserves (sid, bid, day)
Boats (bid, bname, color)

340

= Find the names of sailors who have reserved all red boats

Q: Find the names of sailors so there is no red boat that is not reserved by him.

```
SELECT S.sname
FROM Sailors S
WHERE not exists
        ( SELECT B.bid
        FROM Boats B
        WHERE B.color = 'red'
        AND not exists
                ( SELECT R.bid
                FROM Reserves R
                WHERE R.bid = B.bid
                AND R.sid = S.sid))
```



Query from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000)

http://queryviz.com

Input: Schema

Input Query

Output: Visualization

http://queryviz.com/online

http://www.youtube.com/watch?v=kVFnQRGAQls