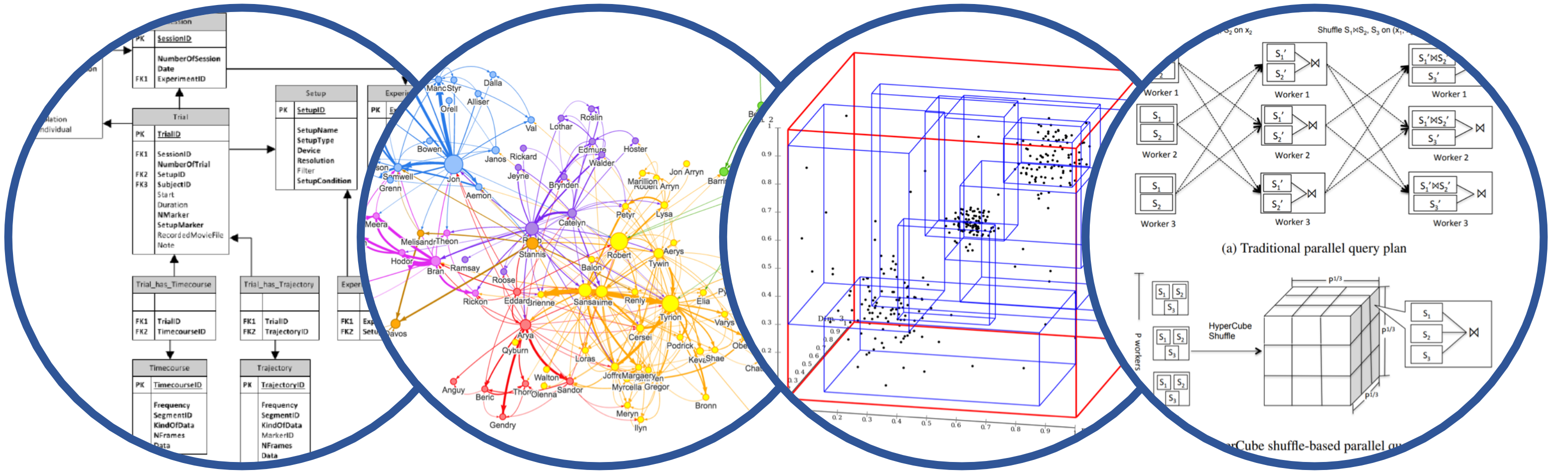


Introduction to Data Management

RA examples and Subqueries

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Announcements

- HW 2 due tomorrow

- Office hours adjusted
  - Shana has office hours Thursday at 3:30pm, Gates 150
  - TAs and I often available by appointment

- Midterm 2 weeks from today – will be added to calendar

Goals for Today

- Recap RA
- Use SQL queries to assist other SQL queries
- Subqueries give you 99% of the tools for queries you can think of

Recap RA Operators

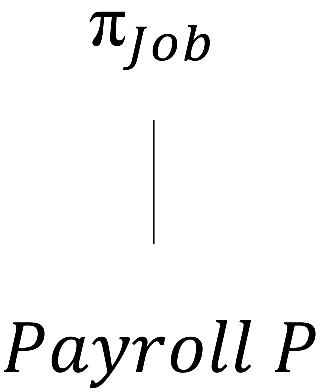
- These are all the operators you will see in this class
  - We'll profile these one at a time

$\bowtie$	Inner Join	γ	Grouping & Aggregation	τ	Sort
$\times$	Cartesian Product	$\cup$	Union	δ	Duplicate Elimination
σ	Selection	$\cap$	Intersection		
π	Projection	$-$	Difference		

Simple Example

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000



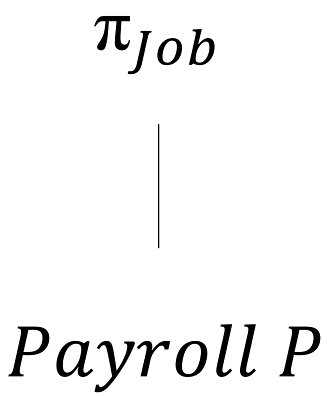
π_{Job}

$Payroll\ P$

Simple Example

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000



π_{Job}

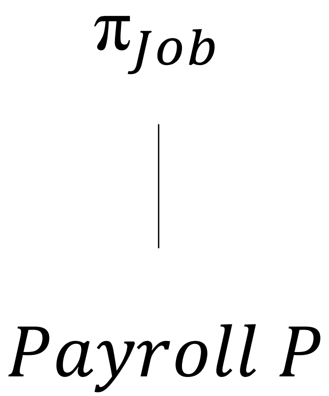
$Payroll\ P$

```
SELECT Job
FROM Payroll;
```

Simple Example

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000



```
SELECT Job
FROM Payroll;
```

Job
TA
TA
Prof
Prof

Simple Example

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Job
TA
Prof

```
SELECT DISTINCT Job
FROM Payroll;
```

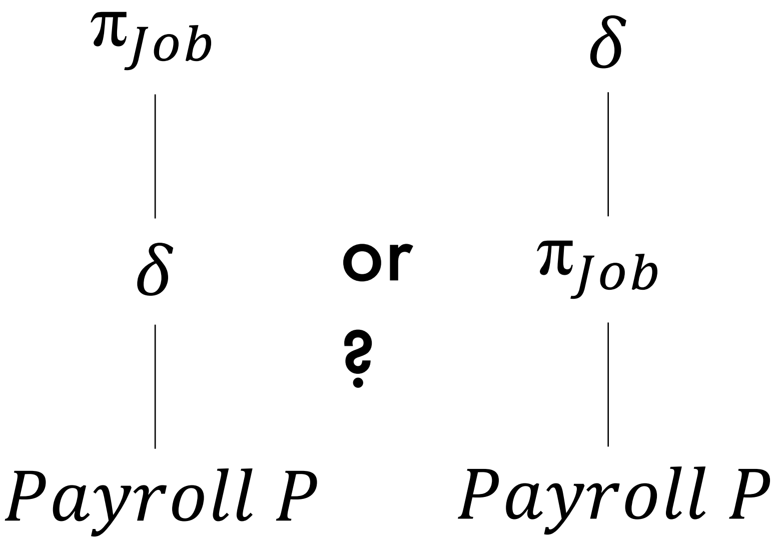
Simple Example

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Job
TA
Prof

```
SELECT DISTINCT Job
FROM Payroll;
```



π_{Job}

δ

$Payroll\ P$

or ?

δ

π_{Job}

$Payroll\ P$

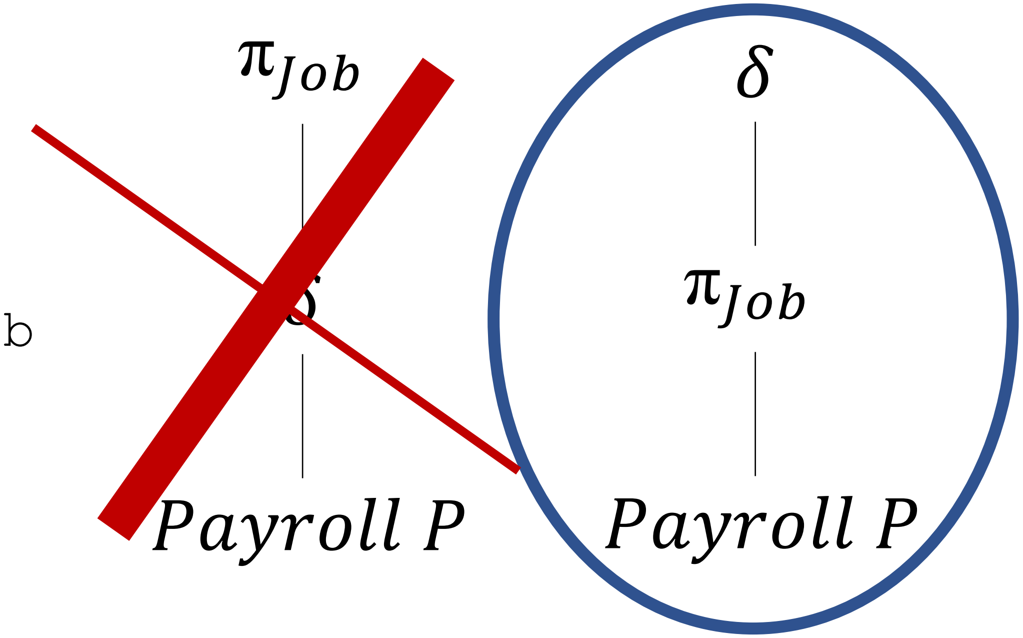
Simple Example

Payroll

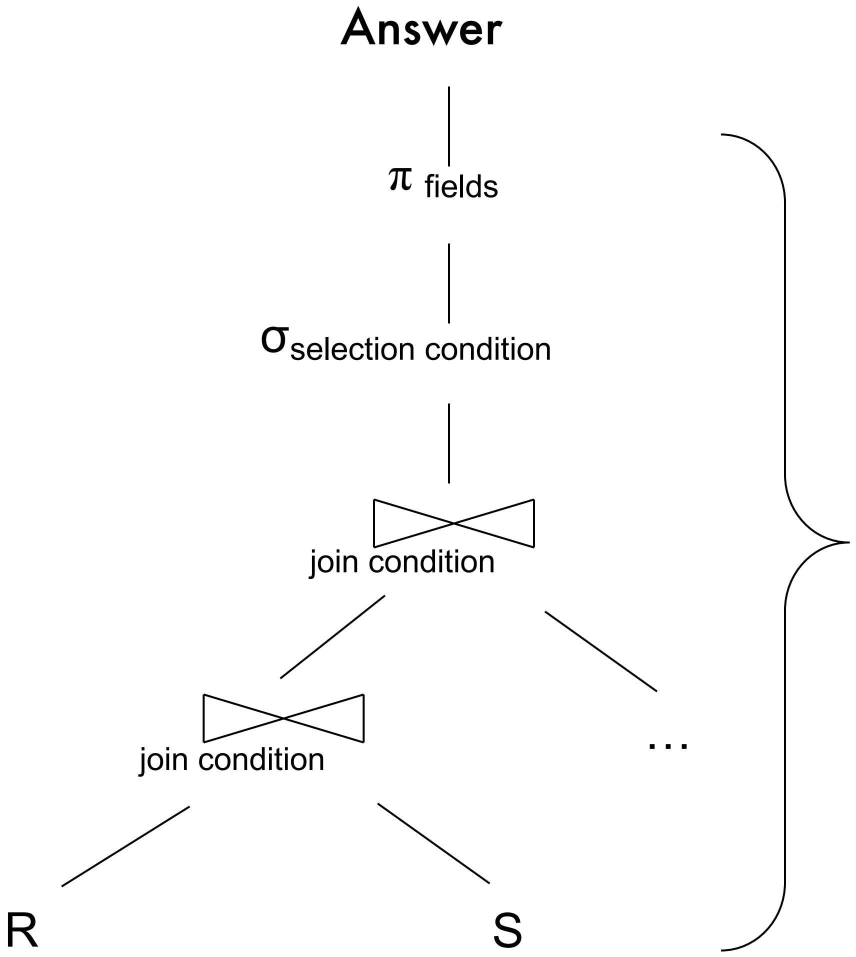
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Job
TA
Prof

```
SELECT DISTINCT Job
FROM Payroll;
```



Typical Plan for a Query



```
SELECT fields
FROM R, S, …
WHERE condition
```

SELECT-PROJECT-JOIN Query

Aggregation Order

How is aggregation processed internally?

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```

Aggregation RA

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```

$$\gamma_{Job,\, MAX(P.Salary)\rightarrow maxSal,\, MIN(P.Salary)\rightarrow minSal}$$

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```

Job	maxSal	minSal
TA	60000	50000
Prof	100000	90000

$$\gamma_{Job,\, MAX(P.Salary)\rightarrow maxSal,\, MIN(P.Salary)\rightarrow minSal}$$

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```

$$\sigma_{minSal>80000}$$

Job	maxSal	minSal
TA	60000	50000
Prof	100000	90000

$$\gamma_{Job,\, MAX(P.Salary) \to maxSal,\, MIN(P.Salary) \to minSal}$$

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```

Job	maxSal	minSal
Prof	100000	90000

$$\sigma_{minSal>80000}$$

Job	maxSal	minSal
TA	60000	50000
Prof	100000	90000

$$\gamma_{Job,\, MAX(P.Salary) \rightarrow maxSal,\, MIN(P.Salary) \rightarrow minSal}$$

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```

$$\pi_{Job,\, maxSal}$$

Job	maxSal	minSal
Prof	100000	90000

$$\sigma_{minSal>80000}$$

Job	maxSal	minSal
TA	60000	50000
Prof	100000	90000

$$\gamma_{Job,\, MAX(P.Salary)\rightarrow maxSal,\, MIN(P.Salary)\rightarrow minSal}$$

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```

Job	maxSal
Prof	100000

$$\pi_{Job,\, maxSal}$$

Job	maxSal	minSal
Prof	100000	90000

$$\sigma_{minSal>80000}$$

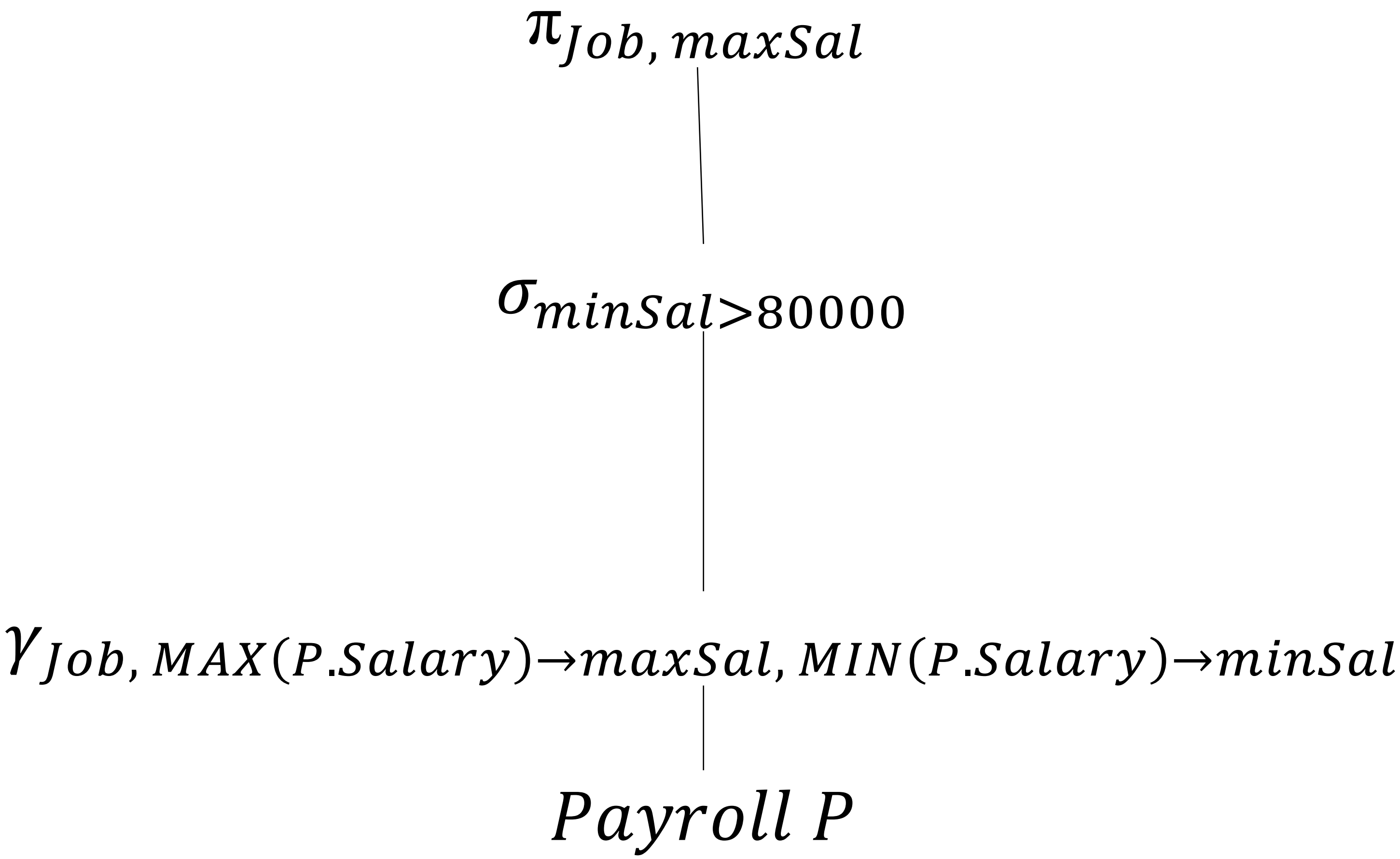
Job	maxSal	minSal
TA	60000	50000
Prof	100000	90000

$$\gamma_{Job,\, MAX(P.Salary)\rightarrow maxSal,\, MIN(P.Salary)\rightarrow minSal}$$

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```



$\pi_{Job,\, maxSal}$

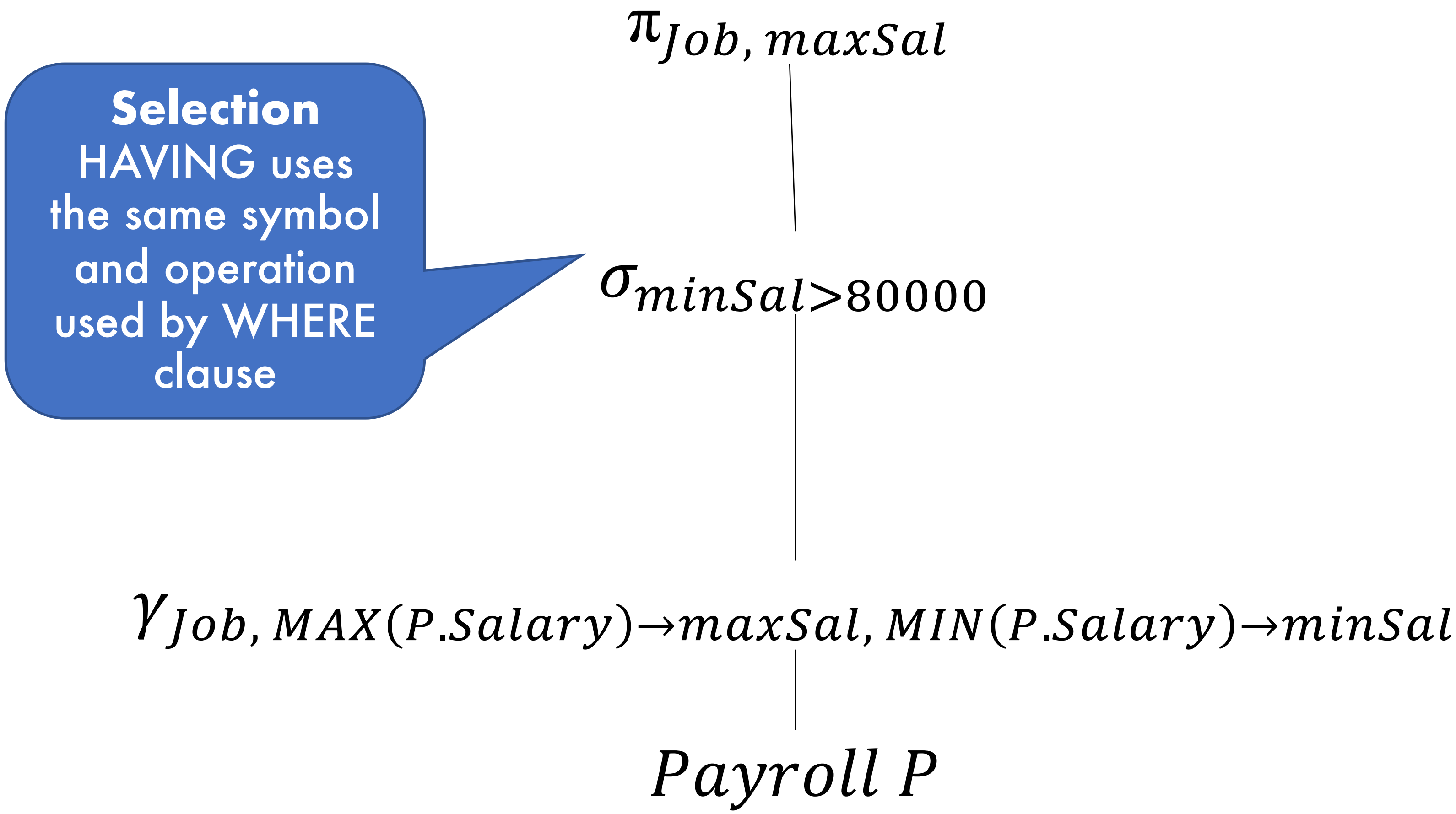
$\sigma_{minSal > 80000}$

$\gamma_{Job,\, MAX(P.Salary) \rightarrow maxSal,\, MIN(P.Salary) \rightarrow minSal}$

$Payroll\ P$

Aggregation RA

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```



Selection
HAVING uses the same symbol and operation used by WHERE clause

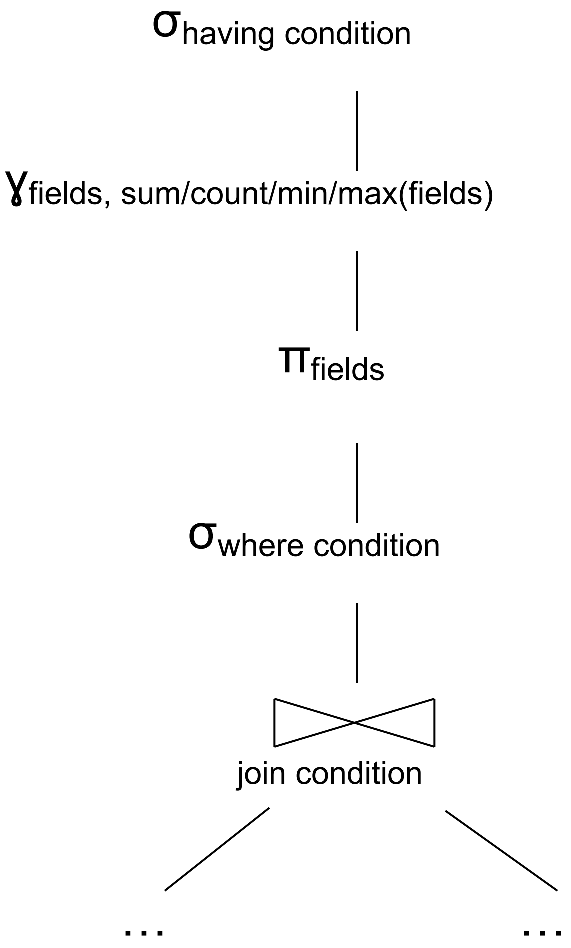
$$\pi_{Job,\, maxSal}$$

$$\sigma_{minSal>80000}$$

$$\gamma_{Job,\, MAX(P.Salary)\rightarrow maxSal,\, MIN(P.Salary)\rightarrow minSal}$$

$$Payroll\ P$$

Typical Plan for an Aggregate Query



$\sigma_{\text{having condition}}$

$\gamma_{\text{fields, sum/count/min/max(fields)}}$

π_{fields}

$\sigma_{\text{where condition}}$

$\bowtie$

join condition

…

…

```
SELECT fields
FROM R, S, …
WHERE condition
GROUP BY fields
HAVING condition
```

Recap – The Witnessing Problem

- A question pattern that asks for data associated with a maxima of some value
  - Observed how to do it with grouping
  - “Self join” on values you find the maxima for
  - GROUP BY to deduplicate one side of the join
  - HAVING to compare values with respective maxima

Outline

- Witnessing via subquery
- Subquery mechanics
    - Set/bag operations
    - SELECT
    - FROM
    - WHERE/HAVING
- Decorrelation and unnesting along the way

The Witnessing Problem Simplified

- Wanted to join respective maxima
  - GROUP BY technique was interesting
  - Last time people suggested that we can **compute the maxima first then join**

UserID	Name	Job	Salary	maxima
123	Jack	TA	50000	60000
345	Allison	TA	60000	60000
567	Magda	Prof	90000	100000
789	Dan	Prof	100000	100000

Return the person (or people) with the highest salary for each job type

The Witnessing Problem - Previously

UserID	Name	Job	Salary	maxima
123	Jack	TA	50000	60000
345	Allison	TA	60000	60000
567	Magda	Prof	90000	100000
789	Dan	Prof	100000	100000

Return the person with the highest salary for each job type

```
SELECT P1.Name, MAX(P2.Salary)
  FROM Payroll AS P1, Payroll AS P2
 WHERE P1.Job = P2.Job
 GROUP BY P2.Job, P1.Salary, P1.Name
HAVING P1.Salary = MAX(P2.Salary)
```

The Witnessing Problem - Previously

```
SELECT P1.Name, MAX(P2.Salary)
  FROM Payroll AS P1, Payroll AS P2
 WHERE P1.Job = P2.Job
 GROUP BY P2.Job, P1.Salary, P1.Name
HAVING P1.Salary = MAX(P2.Salary)
```

Join on "original" grouping attributes

P1				P2			
UserID	**Name**	**Job**	**Salary**	**UserID**	**Name**	**Job**	**Salary**
123	Jack	TA	50000	123	Jack	TA	50000
123	Jack	TA	50000	345	Allison	TA	60000
345	Allison	TA	60000	345	Allison	TA	60000
345	Allison	TA	60000	123	Jack	TA	50000
567	Magda	Prof	90000	567	Magda	Prof	90000
567	Magda	Prof	90000	789	Dan	Prof	100000
789	Dan	Prof	100000	789	Dan	Prof	100000
789	Dan	Prof	100000	567	Magda	Prof	90000

The Witnessing Problem - Previously

```
SELECT P1.Name, MAX(P2.Salary)
  FROM Payroll AS P1, Payroll AS P2
 WHERE P1.Job = P2.Job
 GROUP BY P2.Job, P1.Salary, P1.Name
HAVING P1.Salary = MAX(P2.Salary)
```

Group on additional attributes that you are argmax-ing for

P1				P2			
UserID	Name	Job	Salary	UserID	Name	Job	Salary
123	Jack	TA	50000	123	Jack	TA	50000
123	Jack	TA	50000	345	Allison	TA	60000
345	Allison	TA	60000	345	Allison	TA	60000
345	Allison	TA	60000	123	Jack	TA	50000
567	Magda	Prof	90000	567	Magda	Prof	90000
567	Magda	Prof	90000	789	Dan	Prof	100000
789	Dan	Prof	100000	789	Dan	Prof	100000
789	Dan	Prof	100000	567	Magda	Prof	90000

The Witnessing Problem - Previously

```
SELECT P1.Name, MAX(P2.Salary)
  FROM Payroll AS P1, Payroll AS P2
 WHERE P1.Job = P2.Job
 GROUP BY P2.Job, P1.Salary, P1.Name
HAVING P1.Salary = MAX(P2.Salary)
```

Group on additional attributes that you are argmax-ing for

P1				P2			
UserID	Name	Job	Salary	UserID	Name	Job	Salary
123	Jack	TA	50000	123	Jack	TA	50000
123	Jack	TA	50000	345	Allison	TA	60000
345	Allison	TA	60000	345	Allison	TA	60000
345	Allison	TA	60000	123	Jack	TA	50000
567	Magda	Prof	90000	567	Magda	Prof	90000
567	Magda	Prof	90000	789	Dan	Prof	100000
789	Dan	Prof	100000	789	Dan	Prof	100000
789	Dan	Prof	100000	567	Magda	Prof	90000

The Witnessing Problem - Previously

```sql
SELECT P1.Name, MAX(P2.Salary)
  FROM Payroll AS P1, Payroll AS P2
 WHERE P1.Job = P2.Job
 GROUP BY P2.Job, P1.Salary, P1.Name
HAVING P1.Salary = MAX(P2.Salary)
```

P1				P2			
UserID	**Name**	**Job**	**Salary**	**UserID**	**Name**	**Job**	**Salary**
123	Jack	TA	50000	123	Jack	TA	50000
123	Jack	TA	50000	345	Allison	TA	60000
345	Allison	TA	60000	345	Allison	TA	60000
345	Allison	TA	60000	123	Jack	TA	50000
567	Magda	Prof	90000	567	Magda	Prof	90000
567	Magda	Prof	90000	789	Dan	Prof	100000
789	Dan	Prof	100000	789	Dan	Prof	100000
789	Dan	Prof	100000	567	Magda	Prof	90000

The Witnessing Problem Simplified

```
SELECT P1.Name, MAX(P2.Salary)
  FROM Payroll AS P1, Payroll AS P2
 WHERE P1.Job = P2.Job
 GROUP BY P2.Job, P1.Salary, P1.Name
HAVING P1.Salary = MAX(P2.Salary)
```

We can compute the same thing!

```
WITH MaxPay AS
     (SELECT P1.Job AS Job,
             MAX(P1.Salary) AS Salary
        FROM Payroll AS P1
       GROUP BY P1.Job)
SELECT P.Name, P.Salary
  FROM Payroll AS P, MaxPay AS MP
 WHERE P.Job = MP.Job AND
       P.Salary = MP.Salary
```

The Witnessing Problem Simplified

MaxPay

Job	Salary
TA	60000
Prof	100000

Useful intermediate result!

```
WITH MaxPay AS
     (SELECT P1.Job AS Job,
             MAX(P1.Salary) AS Salary
        FROM Payroll AS P1
       GROUP BY P1.Job)
SELECT P.Name, P.Salary
  FROM Payroll AS P, MaxPay AS MP
 WHERE P.Job = MP.Job AND
       P.Salary = MP.Salary
```

The Witnessing Problem Simplified

```
WITH MaxPay AS
     (SELECT P1.Job AS Job,
             MAX(P1.Salary) AS Salary
        FROM Payroll AS P1
       GROUP BY P1.Job)
SELECT P.Name, P.Salary
  FROM Payroll AS P, MaxPay AS MP
 WHERE P.Job = MP.Job AND
       P.Salary = MP.Salary
```

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

MaxPay

Job	Salary
TA	60000
Prof	100000

The Witnessing Problem Simplified

```
WITH MaxPay AS
     (SELECT P1.Job AS Job,
             MAX(P1.Salary) AS Salary
        FROM Payroll AS P1
       GROUP BY P1.Job)
SELECT P.Name, P.Salary
  FROM Payroll AS P, MaxPay AS MP
 WHERE P.Job = MP.Job AND
       P.Salary = MP.Salary
```

Join predicate

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

MaxPay

Job	Salary
TA	60000
Prof	100000

The Witnessing Problem Simplified

```
WITH MaxPay AS
     (SELECT P1.Job AS Job,
             MAX(P1.Salary) AS Salary
        FROM Payroll AS P1
       GROUP BY P1.Job)
SELECT P.Name, P.Salary
  FROM Payroll AS P, MaxPay AS MP
 WHERE P.Job = MP.Job AND
       P.Salary = MP.Salary
```

Selection Predicate

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

MaxPay

Job	Salary
TA	60000
Prof	100000

The Witnessing Problem Simplified

```
WITH MaxPay AS
     (SELECT P1.Job AS Job,
             MAX(P1.Salary) AS Salary
        FROM Payroll AS P1
       GROUP BY P1.Job)
SELECT P.Name, P.Salary
  FROM Payroll AS P, MaxPay AS MP
 WHERE P.Job = MP.Job AND
       P.Salary = MP.Salary
```

Solving a subproblem can make your life easy

Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

MaxPay

Job	Salary
TA	60000
Prof	100000

The Punchline about Subqueries

- Subqueries can be interpreted as **single values** or as **whole relations**
  - A single value (a 1x1 relation) can be returned as part of a tuple
  - A relation can be:
    - Used as input for another query
    - Checked for containment of a value

Set Operations

- SQL mimics set theory in many ways
  - Bag = duplicates allowed
  - **UNION (ALL)** → set union (bag union)
  - **INTERSECT** (ALL) → set intersection (bag intersection)
  - **EXCEPT** (ALL) → set difference (bag difference)

- SQL Server Management Studio 2017
  - INTERSECT ALL not supported
  - EXCEPT ALL not supported

Set Operations

- SQL set-like operators basically slap two queries together (not really a subquery…)

```
SELECT * FROM T1
UNION
SELECT * FROM T2;
```

Subqueries in SELECT

- Must return a single value
- Uses:
    - Compute an associated value

Subqueries in SELECT

- Must return a single value
- Uses:
  - Compute an associated value

```
SELECT P.Name, (SELECT AVG(P1.Salary)
                  FROM Payroll AS P1
                 WHERE P.Job = P1.Job)
  FROM Payroll AS P
```

Subqueries in SELECT

- Must return a single value
- Uses:
  - Compute an associated value
- Example: For each employee, return their name and average job salary.

```
SELECT P.Name, (SELECT AVG(P1.Salary)
                  FROM Payroll AS P1
                 WHERE P.Job = P1.Job)
  FROM Payroll AS P
```

Subqueries in SELECT

- Must return a single value
- Uses:
  - Compute an associated value
- Example: For each employee, return their name and average job salary.

```
SELECT P.Name, (SELECT AVG(P1.Salary)
                  FROM Payroll AS P1
                 WHERE P.Job = P1.Job)
  FROM Payroll AS P
```

Subqueries in SELECT

- Must return a single value
- Uses:
  - Compute an associated value
- Example: For each employee, return their name and average job salary.

```
SELECT P.Name, (SELECT AVG(P1.Salary)
                  FROM Payroll AS P1
                 WHERE P.Job = P1.Job)
  FROM Payroll AS P
```

Correlated subquery! Semantics are that the entire subquery is recomputed for each tuple

Subqueries in SELECT

- Must return a single value
- Uses:
  - Compute an associated value
- Example: For each employee, return their name and average job salary.

```
SELECT P.Name, (SELECT AVG(P1.Salary)
                  FROM Payroll AS P1
                 WHERE P.Job = P1.Job)
  FROM Payroll AS P
```

Subqueries in SELECT

```
SELECT P.Name, (SELECT AVG(P1.Salary)
                  FROM Payroll AS P1
                 WHERE P.Job = P1.Job)
  FROM Payroll AS P
```

Payroll P

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Subqueries in SELECT

```
SELECT P.Name, (SELECT AVG(P1.Salary)
                 FROM Payroll AS P1
                WHERE P.Job = P1.Job)
  FROM Payroll AS P
```

Payroll P

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Payroll P1

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Subqueries in SELECT

```
SELECT P.Name, (SELECT AVG(P1.Salary)
                  FROM Payroll AS P1
                 WHERE P.Job = P1.Job)
  FROM Payroll AS P
```

Payroll P

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Payroll P1

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Subqueries in SELECT

```
SELECT P.Name, (SELECT AVG(P1.Salary)
                  FROM Payroll AS P1
                 WHERE P.Job = P1.Job)
  FROM Payroll AS P
```

Payroll P

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

55000

Payroll P1

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Subqueries in SELECT

```
SELECT P.Name, (SELECT AVG(P1.Salary)
                  FROM Payroll AS P1
                 WHERE P.Job = P1.Job)
  FROM Payroll AS P
```

Payroll P

UserID	Name	Job	Salary	
123	Jack	TA	50000	55000
345	Allison	TA	60000	
567	Magda	Prof	90000	
789	Dan	Prof	100000	

Subqueries in SELECT

```
SELECT P.Name, (SELECT AVG(P1.Salary)
                 FROM Payroll AS P1
                WHERE P.Job = P1.Job)
  FROM Payroll AS P
```

Payroll P

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

55000

Payroll P1

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Subqueries in SELECT

```
SELECT P.Name, (SELECT AVG(P1.Salary)
                 FROM Payroll AS P1
                WHERE P.Job = P1.Job)
  FROM Payroll AS P
```

Payroll P

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

55000

Payroll P1

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Subqueries in SELECT

```
SELECT P.Name, (SELECT AVG(P1.Salary)
                 FROM Payroll AS P1
                WHERE P.Job = P1.Job)
  FROM Payroll AS P
```

Payroll P

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

55000

55000

Payroll P1

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Subqueries in SELECT

```
SELECT P.Name,  (SELECT AVG(P1.Salary)
                  FROM Payroll AS P1
                 WHERE P.Job = P1.Job)
  FROM Payroll AS P
```

Payroll P

UserID	Name	Job	Salary	
123	Jack	TA	50000	55000
345	Allison	TA	60000	55000
567	Magda	Prof	90000	95000
789	Dan	Prof	100000	

Subqueries in SELECT

```
SELECT P.Name, (SELECT AVG(P1.Salary)
                 FROM Payroll AS P1
                WHERE P.Job = P1.Job)
  FROM Payroll AS P
```

Payroll P

UserID	Name	Job	Salary	
123	Jack	TA	50000	55000
345	Allison	TA	60000	55000
567	Magda	Prof	90000	95000
789	Dan	Prof	100000	95000

Subqueries in SELECT

For each person find the average salary of their job

```
SELECT P.Name, (SELECT AVG(P1.Salary)
                  FROM Payroll AS P1
                 WHERE P.Job = P1.Job)
  FROM Payroll AS P
```

Same (decorrelated and unnested)

```
SELECT P1.Name, AVG(P2.Salary)
  FROM Payroll AS P1, Payroll AS P2
 WHERE P1.Job = P2.Job
 GROUP BY P1.Name
```

Subqueries in SELECT

For each person find the number of cars they drive

Subqueries in SELECT

For each person find the number of cars they drive

```
SELECT P.Name,  (SELECT COUNT(R.Car)
                   FROM Regist AS R
                  WHERE P.UserID =
                        R.UserID)
  FROM Payroll AS P
```

Subqueries in SELECT

For each person find the number of cars they drive

```
SELECT P.Name, (SELECT COUNT(R.Car)
                  FROM Regist AS R
                 WHERE P.UserID =
                       R.UserID)
  FROM Payroll AS P
```

Same? **Discuss!**

```
SELECT P.Name, COUNT(R.Car)
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID
 GROUP BY P.Name
```

Subqueries in SELECT

For each person find the number of cars they drive

```
SELECT P.Name, (SELECT COUNT(R.Car)
                  FROM Regist AS R
                 WHERE P.UserID =
                       R.UserID)
  FROM Payroll AS P
```

0-count case not covered!

```
SELECT P.Name, COUNT(R.Car)
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID
 GROUP BY P.Name
```

Subqueries in SELECT

For each person find the number of cars they drive

```
SELECT P.Name, (SELECT COUNT(R.Car)
                  FROM Regist AS R
                 WHERE P.UserID =
                       R.UserID)
  FROM Payroll AS P
```

Still possible to decorrelate and unnest

Subqueries in SELECT

For each person find the number of cars they drive

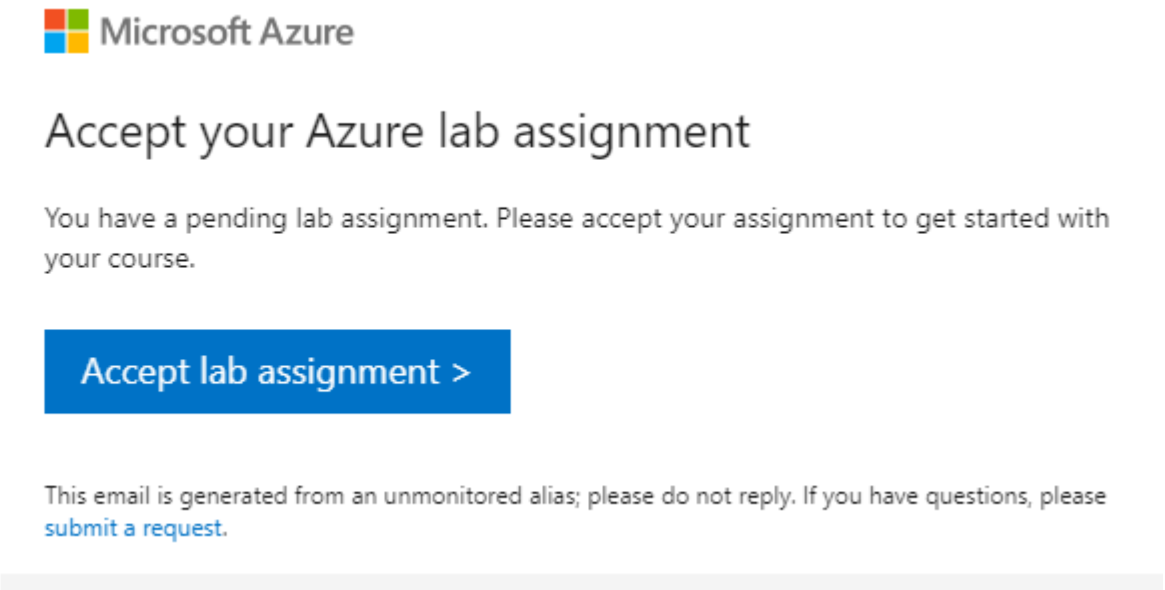
```
SELECT P.Name, (SELECT COUNT(R.Car)
                  FROM Regist AS R
                 WHERE P.UserID =
                       R.UserID)
  FROM Payroll AS P
```

Still possible to decorrelate and unnest

```
SELECT P.Name, COUNT(R.Car)
  FROM Payroll AS P LEFT OUTER JOIN
       Regist AS R ON P.UserID = R.UserID
 GROUP BY P.Name
```

Announcements

- HW 3 out today
- Azure credits have been issued ($75)
  - Sent to @uw.edu emails
  - Enter your @uw.edu email in sign-in
  - Post on Piazza if you have issues



Accept your Azure lab assignment

You have a pending lab assignment. Please accept your assignment to get started with your course.

This email is generated from an unmonitored alias; please do not reply. If you have questions, please submit a request.

Subqueries in FROM

- Equivalent to a WITH subquery
- Uses:
  - Solve subproblems that can be later joined/evaluated

```
  WITH MaxPay AS
       (SELECT P1.Job AS Job,
               MAX(P1.Salary) AS Salary
          FROM Payroll AS P1
         GROUP BY P1.Job)
SELECT P.Name, P.Salary
  FROM Payroll AS P, MaxPay AS MP
 WHERE P.Job = MP.Job AND
       P.Salary = MP.Salary
```

Syntactic sugar

```
SELECT P.Name, P.Salary
  FROM Payroll AS P, (SELECT P1.Job AS Job,
                             MAX(P1.Salary) AS Salary
                        FROM Payroll AS P1
                       GROUP BY P1.Job) AS MP
 WHERE P.Job = MP.Job AND
       P.Salary = MP.Salary
```

Subqueries in WHERE/HAVING

- Uses:
  - ANY → ∃
  - ALL → ∀
  - (NOT) IN → (∉) ∈
  - (NOT) EXISTS → (∅ = …) ∅ ≠ …

Subqueries in WHERE/HAVING

- Uses:
  - ANY → ∃
  - ALL → ∀
  - (NOT) IN → (∉) ∈
  - (NOT) EXISTS → (∅ = …) ∅ ≠ …

Find the name and salary of people who do not drive cars

```
SELECT P.Name, P.Salary
  FROM Payroll AS P
 WHERE P.UserID NOT IN (SELECT UserID
                          FROM Regist)
```

Decorrelated!

3. Subqueries in WHERE

- SELECT .... WHERE EXISTS (subquery);
- SELECT .... WHERE NOT EXISTS (subquery);
- SELECT .... WHERE attribute IN (subquery);
- SELECT .... WHERE attribute NOT IN (subquery);
- SELECT .... WHERE constant > ANY (subquery);
- SELECT .... WHERE constant > ALL (subquery);

3. Subqueries in WHERE

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

Find all companies that make some products with price < 200

3. Subqueries in WHERE

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

Find all companies that make some products with price < 200

Existential quantifiers

3. Subqueries in WHERE

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

Find all companies that make some products with price < 200

Existential quantifiers

Using EXISTS: EXISTS (subquery) returns true iff cardinality of subquery > 0

```
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

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

Find all companies that make some products with price < 200

Existential quantifiers

Using IN: `attr` IN (`subquery`) returns true iff value of `attr` is contained in `subquery`

```
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

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

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)
```

3. Subqueries in WHERE

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

Find all companies that make some products with price < 200

Existential quantifiers

Using ANY: const > ANY (sub) returns true if const > value for at least one value in sub

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

Not supported
in sqlite

3. Subqueries in WHERE

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

Find all companies that make some products with price < 200

Existential quantifiers

Now let’s unnest it:

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

3. Subqueries in WHERE

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

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
```

3. Subqueries in WHERE

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

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

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

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

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

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

3. Subqueries in WHERE

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

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! ☹

3. Subqueries in WHERE

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

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)
```

3. Subqueries in WHERE

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

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

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

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)
```

3. Subqueries in WHERE

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

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)
```

3. Subqueries in WHERE

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

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

Subqueries in WHERE/HAVING

- Uses:
  - ANY → ∃
  - ALL → ∀
  - (NOT) IN → (∉) ∈
  - (NOT) EXISTS → (∅ = …) ∅ ≠ …

Find the name and salary of people who do not drive cars

```
SELECT P.Name, P.Salary
  FROM Payroll AS P
 WHERE NOT EXISTS (SELECT *
                     FROM Regist AS R
                    WHERE P.UserID =
                          R.UserID)
```

Subqueries in WHERE/HAVING

- Uses:
  - ANY → ∃
  - ALL → ∀
  - (NOT) IN → (∉) ∈
  - (NOT) EXISTS → (∅ = …) ∅ ≠ …

Find the name and salary of people who do not drive cars

```
SELECT P.Name, P.Salary
  FROM Payroll AS P
 WHERE P.UserID NOT IN (SELECT UserID
                          FROM Regist)
```

Decorrelated!

Encoding Universal Quantifiers

- Could we ever encode a universal quantifier with a SELECT-FROM-WHERE query with no subqueries or aggregates?

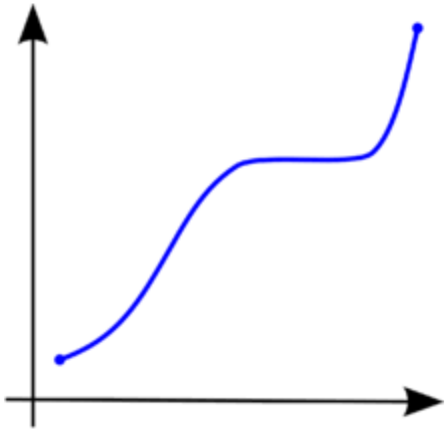
Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.



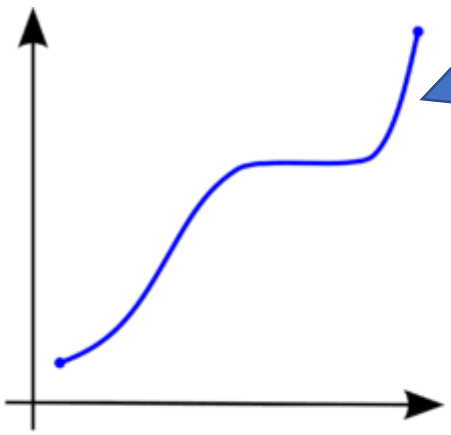
Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.



Monotone queries can be similar to monotonically increasing functions when considering cardinalities of results

Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.

```
SELECT P.Name, P.Car
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID
```

Is this query monotone?

Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.

```
SELECT P.Name, P.Car
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID
```

Is this query monotone? Yes!

Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.

I can't add tuples to Payroll or Regist that would "remove" a previous result

```
SELECT P.Name, P.Car
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID
```

Is this query monotone? Yes!

Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.

```
SELECT P.Name
  FROM Payroll AS P
 WHERE P.Salary >= ALL (SELECT Salary
                          FROM Payroll)
```

Is this query monotone?

Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.

```
SELECT P.Name
  FROM Payroll AS P
 WHERE P.Salary >= ALL (SELECT Salary
                          FROM Payroll)
```

Is this query monotone? No!

Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.

I can add a tuple to Payroll that has a higher salary value than any other

```
SELECT P.Name
  FROM Payroll AS P
 WHERE P.Salary >= ALL (SELECT Salary
                          FROM Payroll)
```

Is this query monotone? No!

Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.

```
SELECT P.Job, COUNT(*)
  FROM Payroll AS P
 GROUP BY P.Job
```

Is this query monotone?

Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.

```
SELECT P.Job, COUNT(*)
  FROM Payroll AS P
 GROUP BY P.Job
```

Is this query monotone? No!

Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.

Aggregates generally are sensitive to any new tuples

```
SELECT P.Job, COUNT(*)
  FROM Payroll AS P
 GROUP BY P.Job
```

Is this query monotone? No!

Monotonicity

- All SELECT-FROM-WHERE queries (without aggregates) are monotone
- **Queries with universal quantifiers are not generally monotone**
- You have to do something "complex" if you need to code a universal quantifier

Takeaways

- SQL is able to mirror logic over sets more or less directly
- The internal interpretation of nested queries can be quite involved
  - But our DBMS is able to derive such interpretations automagically
- We can reason about expressive power of certain queries.

Next Unit

- We are done with lectures on SQL queries!
- Up next: Database Design
  - Data modeling
  - Ethics and Security