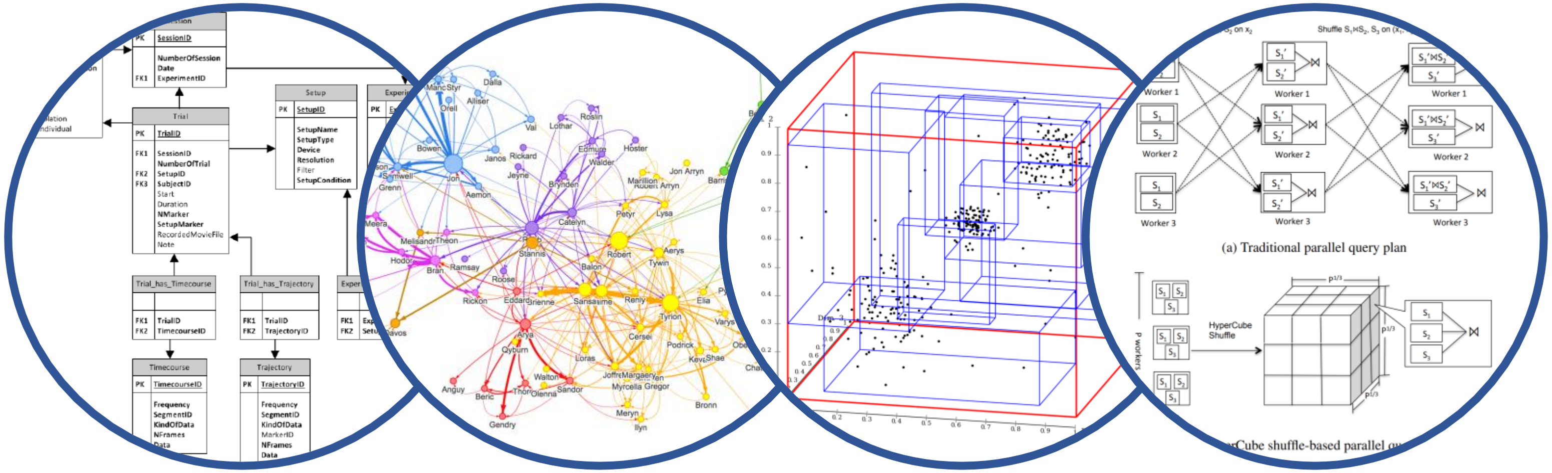
CSE 414, Winter 2020



Introduction to Data Management

query in a query in a query in a …

Shana Hutchison

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

Recap – FWGHOS™

- Our SQL toolbox grows!
  - GROUP BY – grouping and aggregating
  - HAVING – post-group selection filter
- SQL "executed" in the following order
  1. FROM
  2. WHERE
  3. GROUP BY
  4. HAVING
  5. ORDER BY
  6. SELECT (DISTINCT)

Recap – The Witnessing Problem

- A question pattern that asks for data associated with a maxima/minima 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 Problem
  - (last lecture) with fancy self-join
  - building up subqueries from sub-plans
  - WITH clause to abstract useful intermediate result
- Subquery mechanics
  - Set/bag operations
  - FROM
  - SELECT
  - WHERE/HAVING
- Decorrelation, equivalences along the way
- Universal quantification queries

The Witnessing Problem Simplified

- Wanted to join respective maxima
  - GROUP BY technique was interesting
  - Remember the suggestion last Friday that we **compute the maxima first then join & filter**?

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

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

The Witnessing Problem Simplified

- Wanted to join respective maxima
  - GROUP BY technique was interesting
  - Remember the suggestion last Friday that we **compute the maxima first then join & filter**?

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 Simplified

```
SELECT Job,
       MAX(Salary) AS maxima
  FROM Payroll
GROUP BY Job
```

Job	maxima
TA	60000
Prof	100000

↑

$\gamma_{Job,\, MAX(P.Salary) \rightarrow maxima}$

↑

Payroll

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

The Witnessing Problem Simplified

```
SELECT Job,
       MAX(Salary) AS maxima
  FROM Payroll
GROUP BY Job
```

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

⋈

"Natural Join"
Join on matching attributes

Job	maxima
TA	60000
Prof	100000

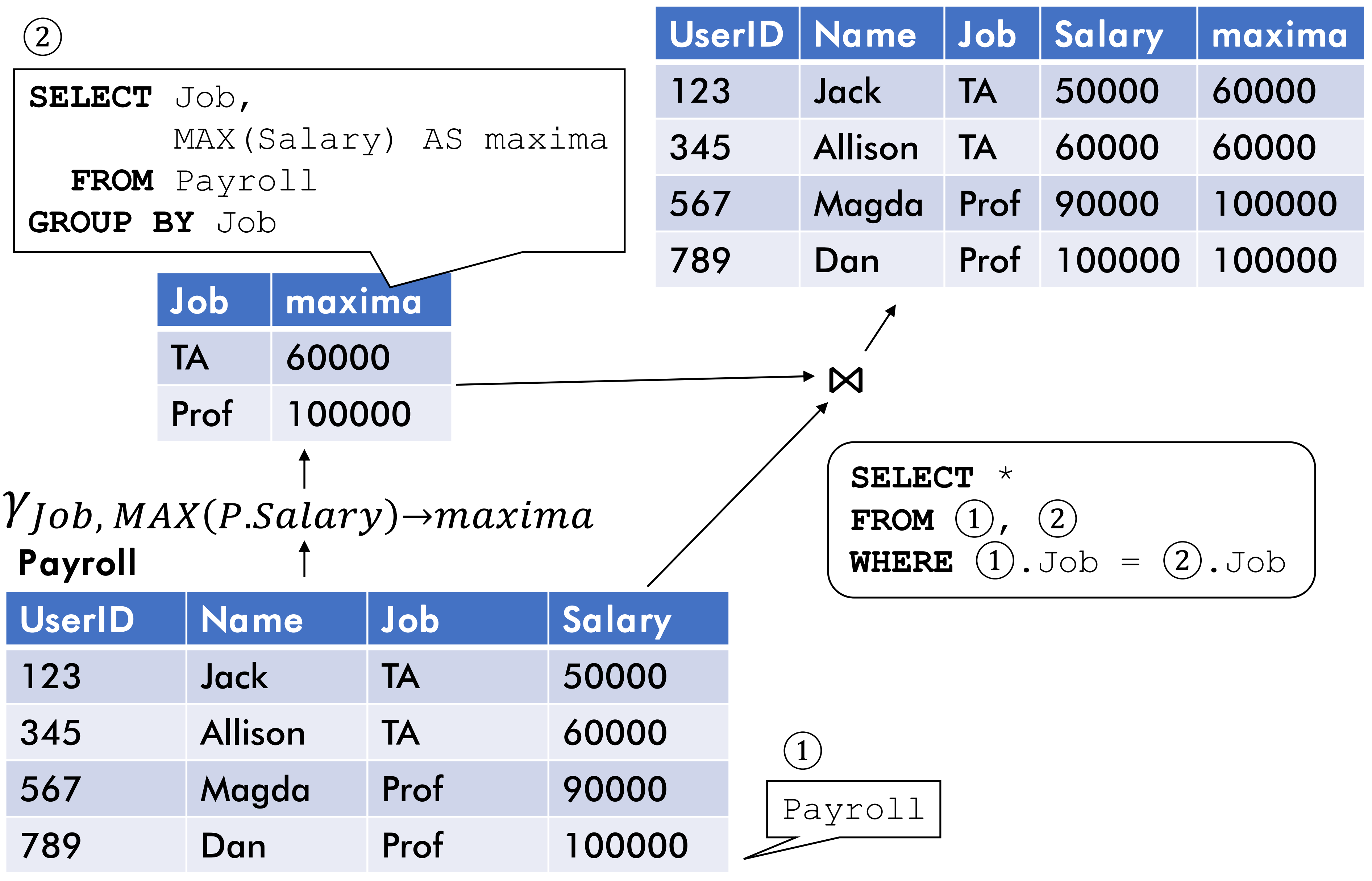
$\gamma_{Job,\, MAX(P.Salary) \to maxima}$

Payroll

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

Okay, but how do we write SQL for this beast?

The Witnessing Problem Simplified



②

```
SELECT Job,
       MAX(Salary) AS maxima
  FROM Payroll
GROUP BY Job
```

Job	maxima
TA	60000
Prof	100000

$\gamma_{Job,\ MAX(P.Salary) \rightarrow maxima}$

Payroll

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

$\bowtie$

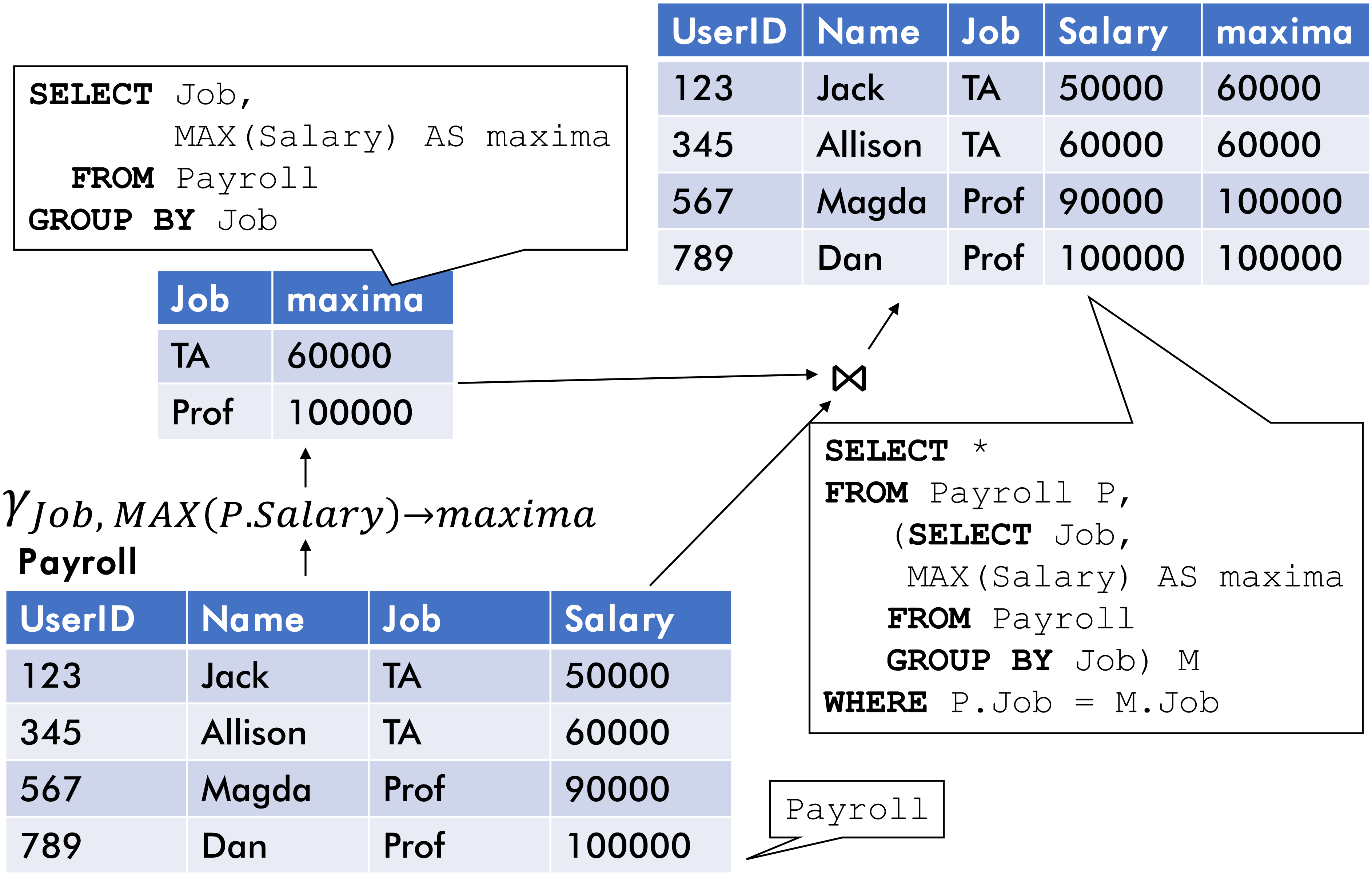
UserID	Name	Job	Salary	maxima
123	Jack	TA	50000	60000
345	Allison	TA	60000	60000
567	Magda	Prof	90000	100000
789	Dan	Prof	100000	100000

```
SELECT *
FROM ①, ②
WHERE ①.Job = ②.Job
```

①

```
Payroll
```

The Witnessing Problem Simplified



```
SELECT Job,
       MAX(Salary) AS maxima
  FROM Payroll
GROUP BY Job
```

Job	maxima
TA	60000
Prof	100000

$\gamma_{Job,\, MAX(P.Salary) \rightarrow maxima}$

Payroll

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

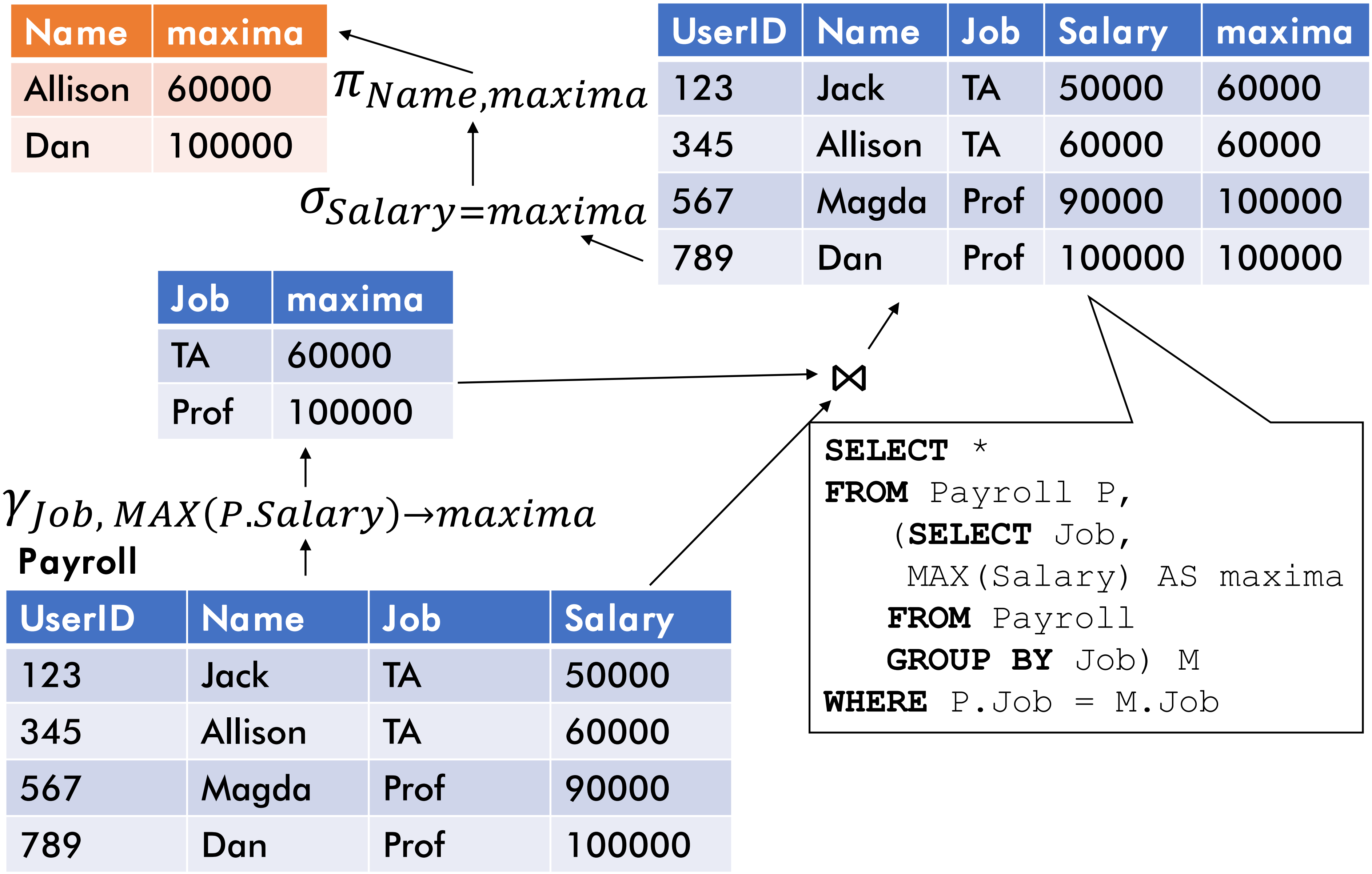
$\bowtie$

```
SELECT *
FROM Payroll P,
    (SELECT Job,
     MAX(Salary) AS maxima
    FROM Payroll
    GROUP BY Job) M
WHERE P.Job = M.Job
```

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

```
Payroll
```

The Witnessing Problem Simplified



Name	maxima
Allison	60000
Dan	100000

$\pi_{Name,maxima}$

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

$\sigma_{Salary=maxima}$

Job	maxima
TA	60000
Prof	100000

$\bowtie$

$\gamma_{Job,\, MAX(P.Salary) \rightarrow maxima}$

Payroll

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

```
SELECT *
FROM Payroll P,
    (SELECT Job,
     MAX(Salary) AS maxima
    FROM Payroll
    GROUP BY Job) M
WHERE P.Job = M.Job
```

The Witnessing Problem Simplified

Name	maxima
Allison	60000
Dan	100000

$\pi_{Name,maxima}$

$\sigma_{Salary=maxima}$

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

```
SELECT Name, maxima
FROM ③
WHERE Salary = maxima
```

$\bowtie$

Prof	100000

③

```
SELECT *
FROM Payroll P,
    (SELECT Job,
     MAX(Salary) AS maxima
    FROM Payroll
    GROUP BY Job) M
WHERE P.Job = M.Job
```

$\gamma_{Job,\, MAX(P.Salary)\rightarrow maxima}$

Payroll

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

The Witnessing Problem Simplified

Name	maxima
Allison	60000
Dan	100000

$\pi_{Name,maxima}$

$\sigma_{Salary=maxima}$

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

$\bowtie$

```
SELECT Name, maxima
FROM (SELECT *
    FROM Payroll P,
         (SELECT Job,
          MAX(Salary) AS maxima
         FROM Payroll
         GROUP BY Job) M
    WHERE P.Job = M.Job)
WHERE Salary = maxima
```

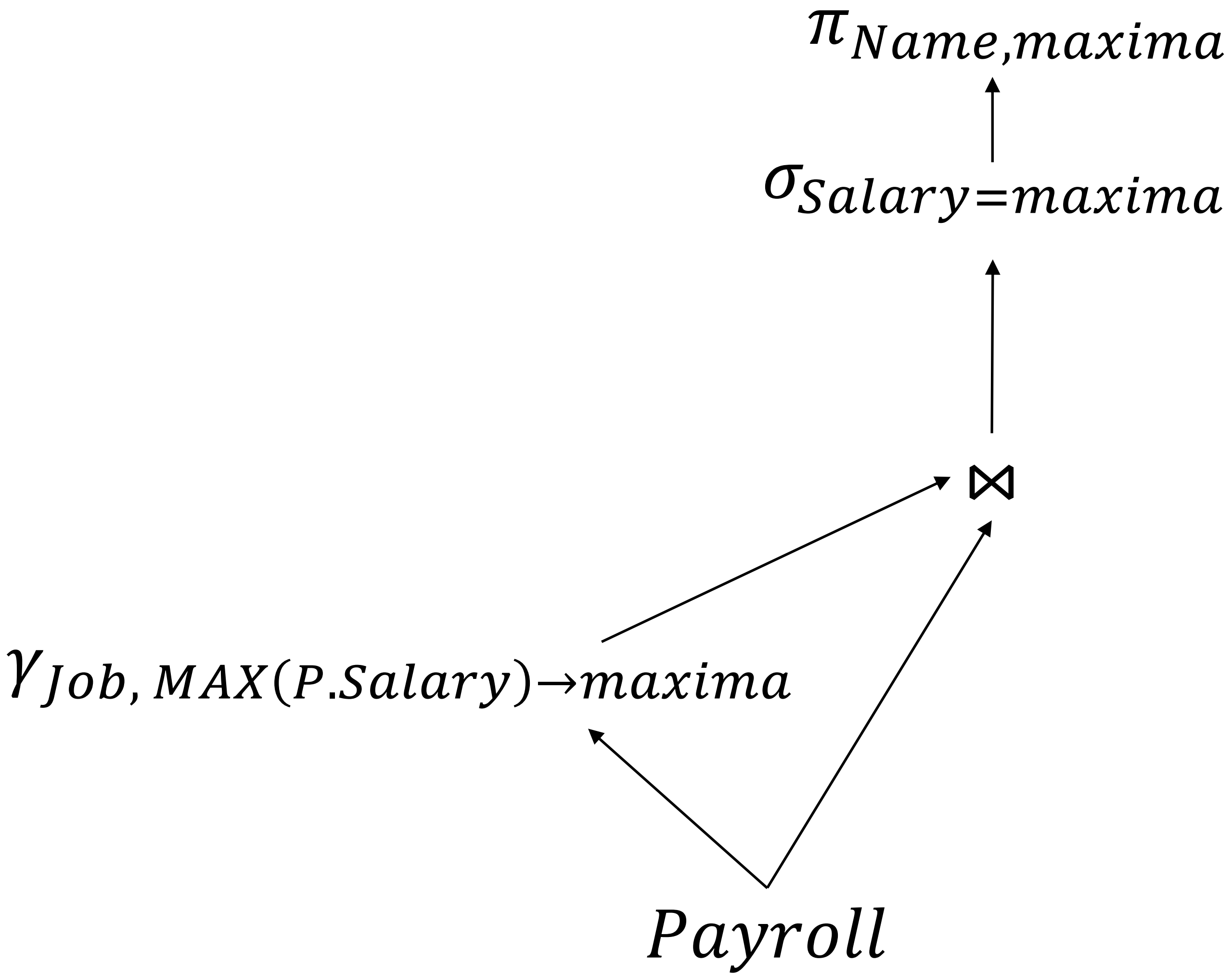
```
SELECT *
FROM Payroll P,
    (SELECT Job,
     MAX(Salary) AS maxima
    FROM Payroll
    GROUP BY Job) M
WHERE P.Job = M.Job
```

γ

P

U			y
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

The Witnessing Problem Simplified



$\pi_{Name, maxima}$

$\sigma_{Salary = maxima}$

$\bowtie$

$\gamma_{Job,\, MAX(P.Salary) \rightarrow maxima}$

$Payroll$

On A Practical Note

Split your problem into sub-problems!

Solve them separately,
then compose them!

Easier to debug a small query than a big one…

The Witnessing Problem [More] Simplified

- Hold up, too complicated!
- 1st & 2nd level
  can be combined

```
SELECT Name, maxima
FROM (SELECT *
   FROM Payroll P,
      (SELECT Job,
       MAX(Salary) AS maxima
      FROM Payroll
      GROUP BY Job) M
   WHERE P.Job = M.Job)
WHERE Salary = maxima
```

The Witnessing Problem [More] Simplified

- Hold up, too complicated!
- 1st & 2nd level can be combined

```
SELECT Name, maxima
FROM (SELECT *
   FROM Payroll P,
      (SELECT Job,
       MAX(Salary) AS maxima
      FROM Payroll
      GROUP BY Job) M
   WHERE P.Job = M.Job)
WHERE Salary = maxima
```

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

- Ahh, that’s better. Can we combine further?
- Well, yes, but it results in the self-join query we started with…
  - Need 2nd copy of Payroll
  - Outer WHERE becomes HAVING (for filter after aggregation)

The Witnessing Problem [More] Simplified

- Hold up, too complicated!
- 1st & 2nd Level

```
SELECT Name, maxima
FROM (SELECT *
    FROM Payroll P,
         (SELECT Job,
          MAX(Salary) AS maxima
          FROM Payroll
          GROUP BY Job) M
    WHERE P.Job = M.Job)
WHERE Salary = maxima
```

Tradeoff between:

1. Complicated query w/o subqueries
2. “Simpler” query w/ subqueries

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

- Ahh, that’s better. Can we combine further?
- Well, yes, but it results in the self-join query we started with…
  - Need 2nd copy of Payroll
  - Outer WHERE becomes HAVING (for filter after aggregation)

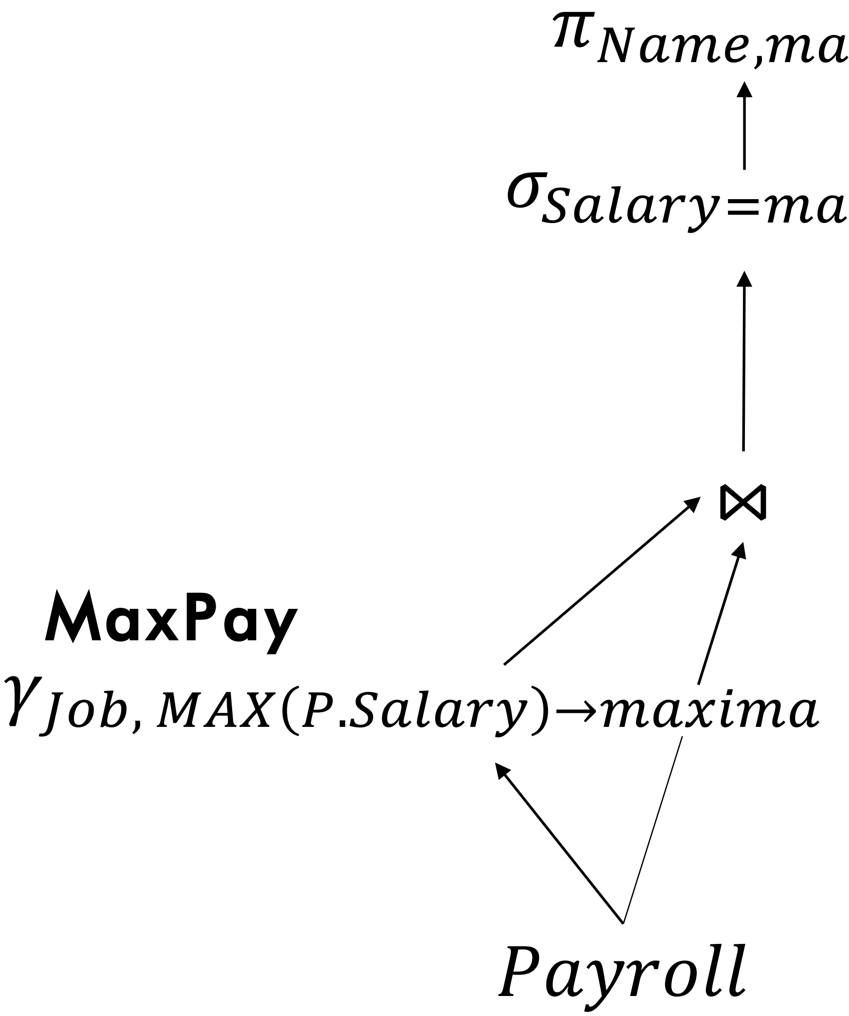
The Witnessing Problem Abstracted

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

- **Option #3: Save an intermediate result**
  - Good for abstraction
  - Good for reuse

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

The Witnessing Problem Abstracted



$\pi_{Name,maxima}$

$\sigma_{Salary=maxima}$

$\bowtie$

MaxPay

$\gamma_{Job,\,MAX(P.Salary)\rightarrow maxima}$

$Payroll$

MaxPay

Job	Salary
TA	60000
Prof	100000

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

The Witnessing Problem Summary

Option 1:
Build larger query from subqueries

```
SELECT Name, maxima
FROM (SELECT *
   FROM Payroll P,
      (SELECT Job, MAX(Salary) AS maxima
      FROM Payroll
      GROUP BY Job) M
   WHERE P.Job = M.Job)
WHERE Salary = maxima
```

Option 2:
Write a fancy big query

```
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)
```

Option 3:
Abstract a useful intermediate result

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

The Witnessing Problem Summary

```
SELECT Name, maxima
FROM (SELECT *
    FROM Payroll P,
        (SELECT Job, MAX(Salary) AS maxima
        FROM Payroll
        GROUP BY Job) M
    WHERE P.Job = M.Job)
WHERE Salary = maxima
```

```
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)
```

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

The Witnessing Problem Summary

psst… later we learn a 4th option:

correlated subquery in the WHERE clause

So many options!

- With a perfect optimizer, all options are equivalent in speed; choose the easiest to write
- In practice, how you write a query affects speed

General Subqueries

- Let’s step back
- Subqueries can be interpreted as
  - **single values**
    - Single-tuple, single-attribute (1x1) relation
    - Returned as part of a tuple
  - **whole relations**
    - Used as input for another query
    - Checked for containment of a value

Set Operations: SQL

- SQL / RA mimics set theory
  - 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: RA

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

Bag operators in RA are rare.

I've seen $\uplus$ for bag union.

2-arg **Union** op (Symbol "cup")
Compute set union
on matching schema

$$A \cup B$$

2-arg **Intersection** op (Symbol "cap")
Compute set intersection
on matching schema

$$A \cap B$$

2-arg **Difference** op (Backslash)
Compute set difference
on matching schema

$$A \setminus B$$

Union

{ (123, Jack, TA, 50000),
(345, Allison, TA, 60000),
(567, Magda, Prof, 90000),
(789, Dan, Prof, 100000) }

∪

{ (987, Gibbs, TA, 60000),
(423, Shana, Prof, 60000) }

=

{ (123, Jack, TA, 50000),
(345, Allison, TA, 60000),
(567, Magda, Prof, 90000),
(789, Dan, Prof, 100000),
(987, Gibbs, TA, 60000),
(423, Shana, Prof, 60000) }

{ (123, Jack, TA, 50000),
(345, Allison, TA, 60000),
(567, Magda, Prof, 90000),
(789, Dan, Prof, 100000) }

∪

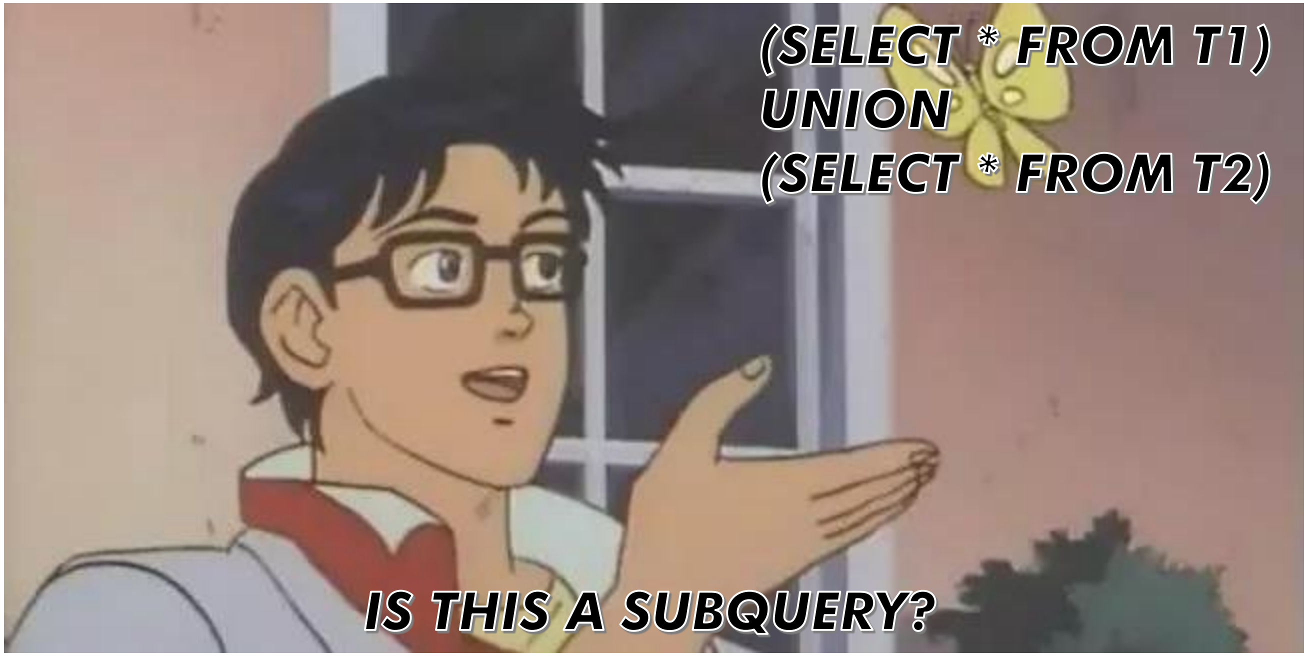
{ (987, Gibbs),
(423, Shana) }

= ???

Schema mismatch!

Set Operations

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



Subqueries in FROM

- Equivalent to a WITH subquery
- Use: Solve sub-problem to later join / evaluate

```
 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 SELECT

- Must return a single value
  - single-tuple single-attribute

Uncorrelated Subqueries in SELECT

Uncorrelated = independent of outer query

```
SELECT Name,
    (SELECT AVG(Salary)
       FROM Payroll) AS AvgSal
  FROM Payroll
```

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

Uncorrelated Subqueries in SELECT

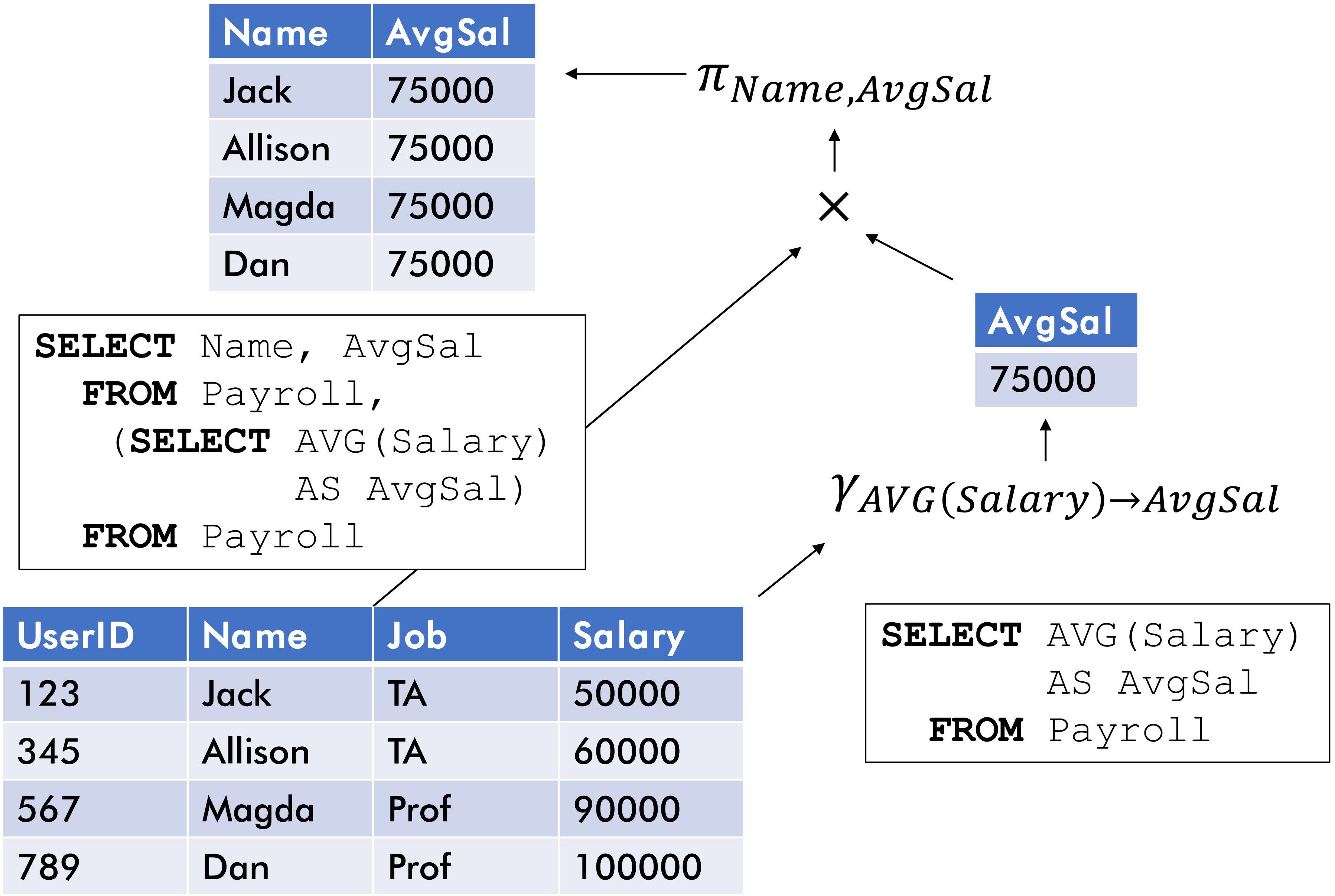
Name	AvgSal
Jack	75000
Allison	75000
Magda	75000
Dan	75000

- Hold up, we've seen this before!
- Transform to FROM clause

```
SELECT Name,
    (SELECT AVG(Salary)
       FROM Payroll) AS AvgSal
  FROM Payroll
```

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

Uncorrelated Subqueries in SELECT



Name	AvgSal
Jack	75000
Allison	75000
Magda	75000
Dan	75000

$\pi_{Name,AvgSal}$

$\times$

```
SELECT Name, AvgSal
  FROM Payroll,
    (SELECT AVG(Salary)
            AS AvgSal)
  FROM Payroll
```

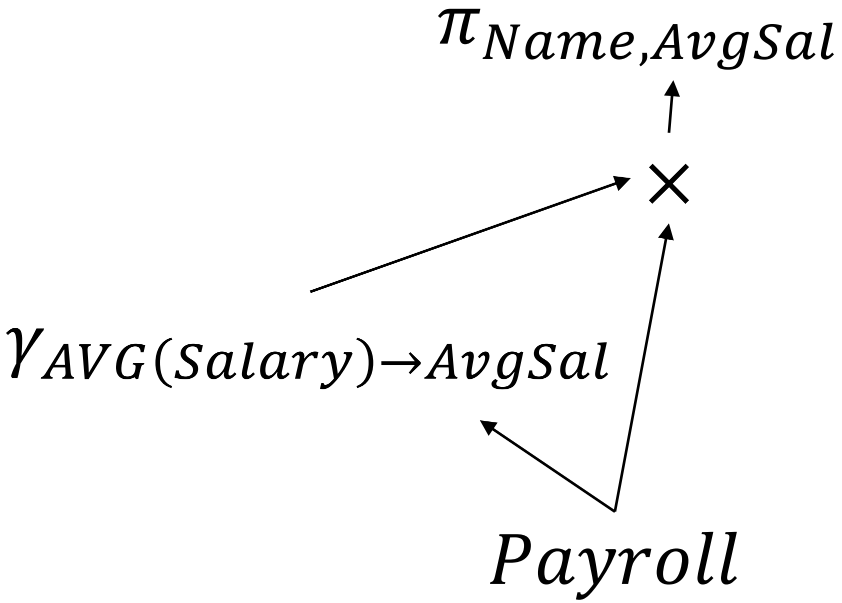
AvgSal
75000

$\gamma_{AVG(Salary)\rightarrow AvgSal}$

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

```
SELECT AVG(Salary)
       AS AvgSal
  FROM Payroll
```

Uncorrelated Subqueries in SELECT



$\pi_{Name,AvgSal}$

$\times$

$\gamma_{AVG(Salary) \rightarrow AvgSal}$

$Payroll$

- Uncorrelated SELECT subqueries don’t add much
- Can convert to FROM subqueries

Correlated Subqueries in SELECT

- Must return a single value
- Used to compute an associated value

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)
                AS AvgSal
  FROM Payroll AS P
```

Correlated Subqueries in SELECT

- Must return a single value
- Used to compute an associated value

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)
                AS AvgSal
  FROM Payroll AS P
```

Correlated subquery!
The entire subquery is recomputed for each tuple

Correlated Subqueries in SELECT

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

Output

Name	AvgSal

Payroll P

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

Correlated Subqueries in SELECT

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

Output

Name	AvgSal

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

Correlated Subqueries in SELECT

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

Output

Name	AvgSal

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

Correlated Subqueries in SELECT

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

Output

Name	AvgSal
Jack	55000

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

Correlated Subqueries in SELECT

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

Output

Name	AvgSal
Jack	55000

Correlated Subqueries in SELECT

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

55000

Output

Name	AvgSal
Jack	55000
Allison	55000

Correlated Subqueries in SELECT

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

Output

Name	AvgSal
Jack	55000
Allison	55000
Magda	**95000**

Correlated Subqueries in SELECT

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

Output

Name	AvgSal
Jack	55000
Allison	55000
Magda	95000
Dan	**95000**

Correlated Subqueries in SELECT

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

Output

Name	AvgSal
Jack	55000
Allison	55000
Magda	95000
Dan	95000

Correlated Subqueries in SELECT

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

Output

Name	AvgSal
Jack	55000
Allison	55000
Magda	95000
Dan	95000

- Hold up! How do we draw an RA Plan?
- There is no “evaluate a different subquery for each tuple” operator…
- Solution: de-correlate by adding a join

Think About This

Design an equivalent query without a SELECT subquery

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) AS AvgSal
  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

→

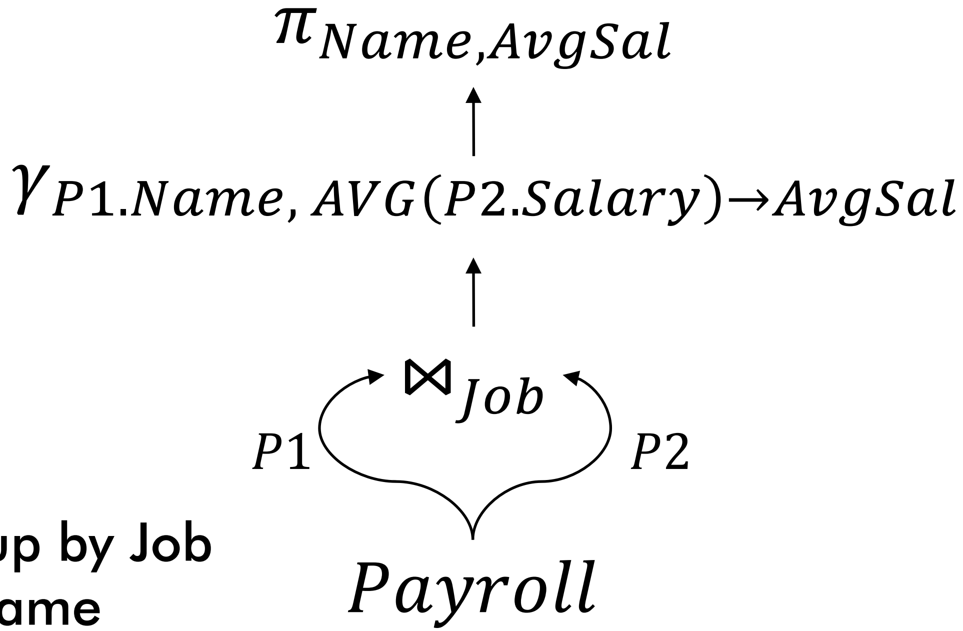
Output

Name	AvgSal
Jack	55000
Allison	55000
Magda	95000
Dan	95000

Solution #1

For each person find the average salary of their job

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



$\pi_{Name,AvgSal}$

$\gamma_{P1.Name,\,AVG(P2.Salary)\rightarrow AvgSal}$

$\bowtie_{Job}$

$P1$ $P2$

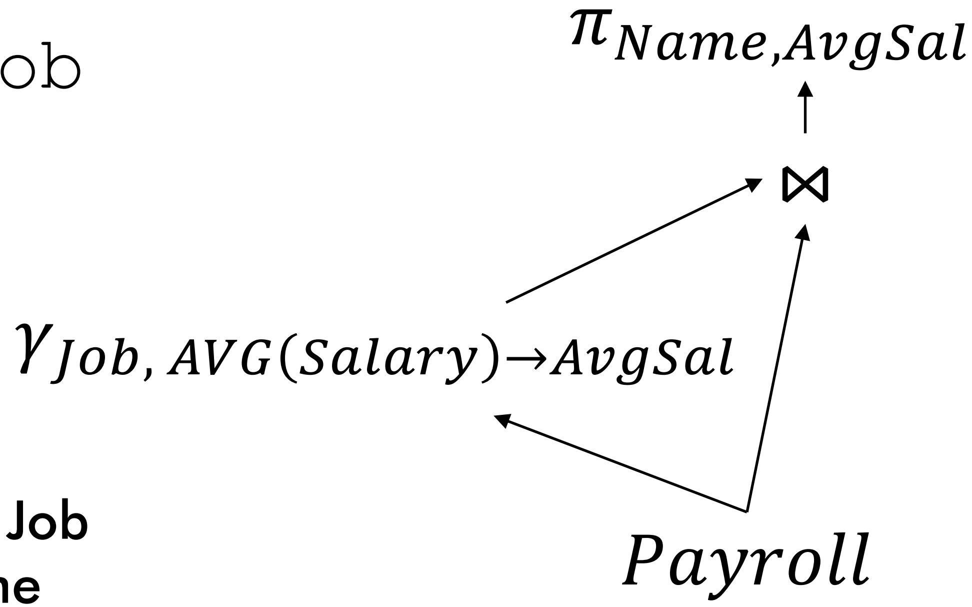
$Payroll$

Idea:
1. Self-join Payroll
2. Use one copy to aggregate, group by Job
3. Use one copy to keep original Name

Solution #2

For each person find the average salary of their job

```
SELECT P.Name, M.AvgSal
  FROM Payroll P,
    (SELECT Job, AVG(Salary) AS AvgSal
        FROM Payroll
       GROUP BY Job) M
 WHERE P.Job = M.Job
```



Idea:

1. Compute aggregate, group by Job
2. Join to original Payroll for Name

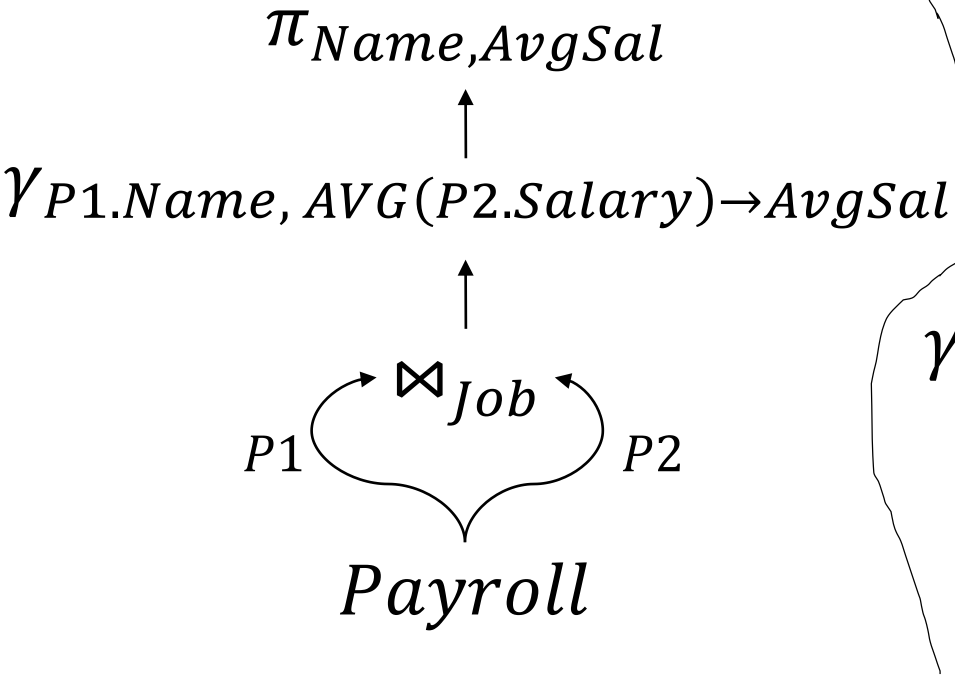
Preview: RA Equivalence

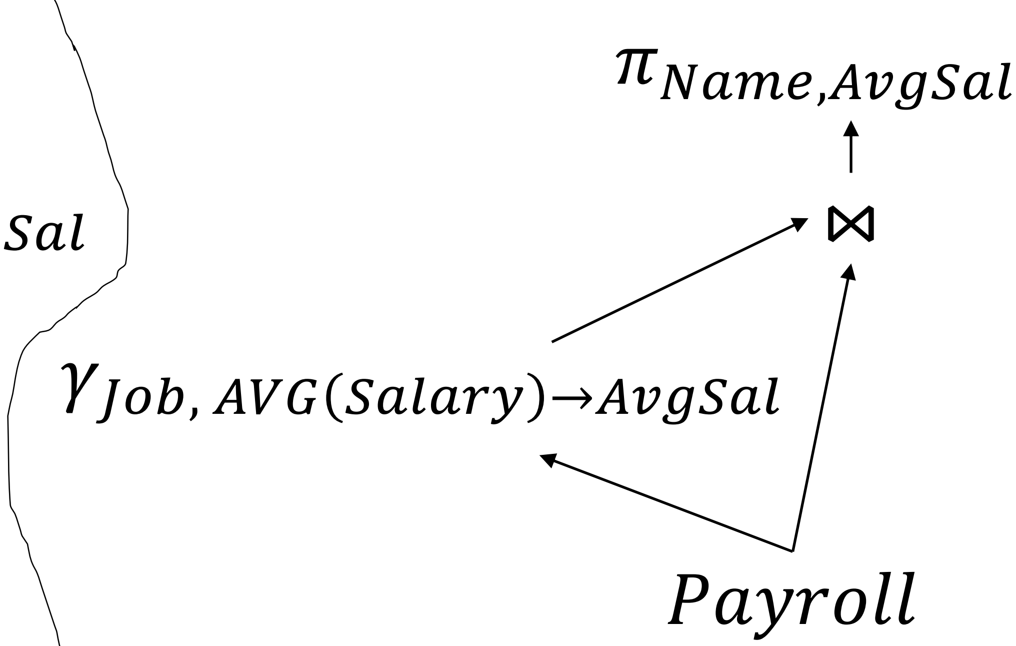
For each person find the average salary of their job

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

```
SELECT P.Name, M.AvgSal
  FROM Payroll P,
    (SELECT Job, AVG(Salary) AS AvgSal
       FROM Payroll
      GROUP BY Job) M
 WHERE P.Job = M.Job
```

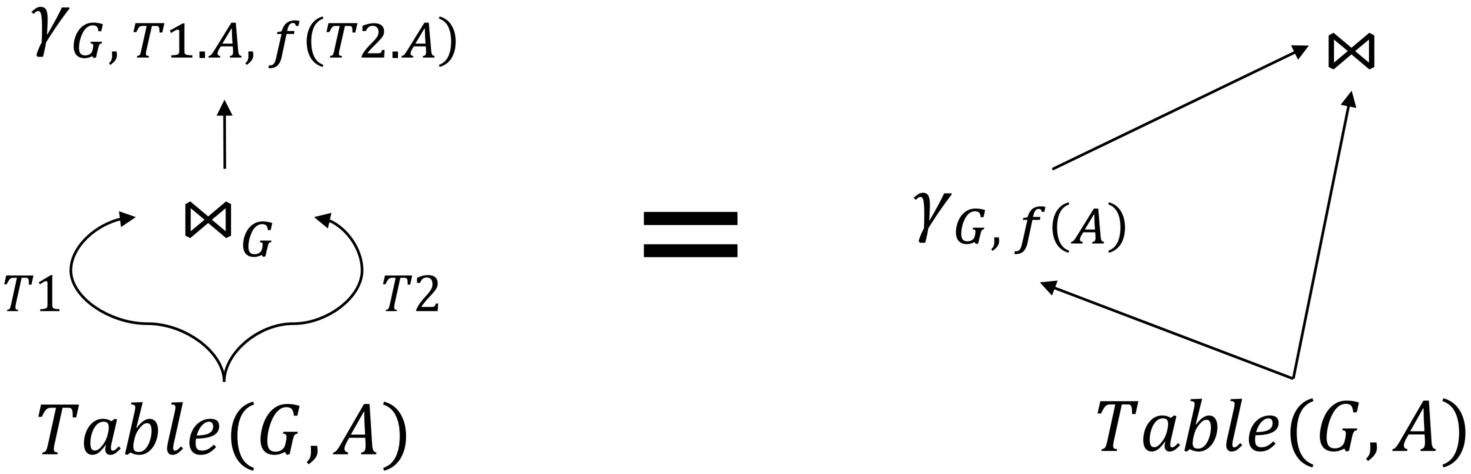
=





Preview: RA Equivalence

- General form: “Push aggregate into join” rewrite
- Speedup
- Proof using formal RA



Subqueries in SELECT: Special Cases

For each person find the number of cars they drive

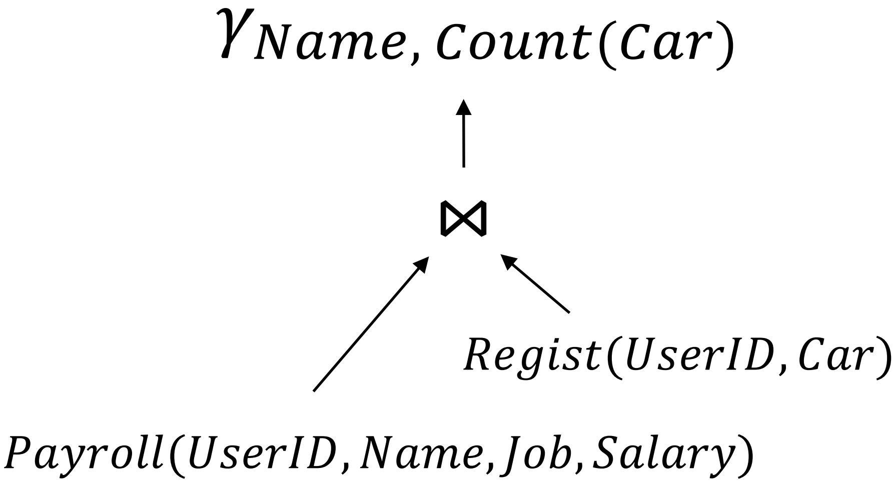
```
SELECT P.Name,
    (SELECT COUNT(R.Car)
       FROM Regist AS R
      WHERE P.UserID = R.UserID)
  FROM Payroll AS P
```

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

=?

Same? Discuss!

(RA impossible with
correlated SELECT subquery)


$\gamma_{Name, Count(Car)}$
$\bowtie$
$Regist(UserID, Car)$
$Payroll(UserID, Name, Job, Salary)$

Payroll

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

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

Subqueries in SELECT: Special Cases

For each person find the number of cars they drive

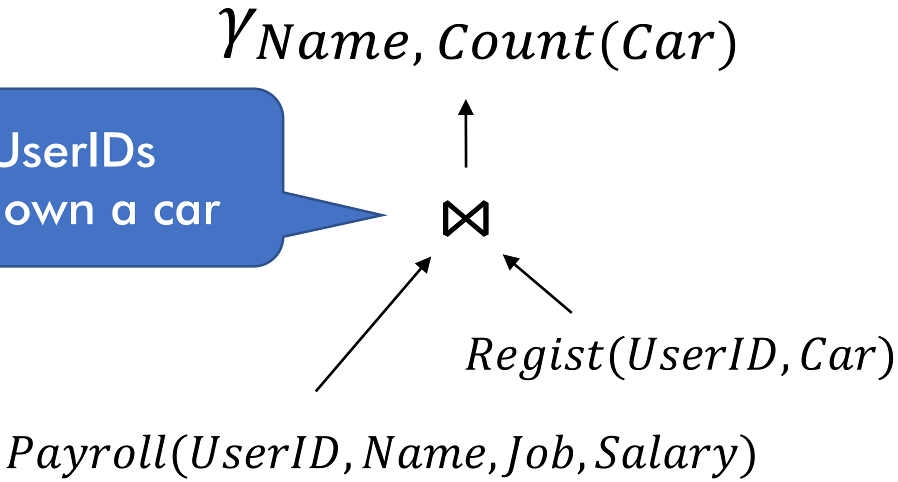
```
SELECT P.Name,
   (SELECT COUNT(R.Car)
      FROM Regist AS R
     WHERE P.UserID = R.UserID)
  FROM Payroll AS P
```

Returns 0 for UserIDs that don't own a car

≠

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

Drops UserIDs that don't own a car


$\gamma_{Name, Count(Car)}$
$\bowtie$
$Regist(UserID, Car)$
$Payroll(UserID, Name, Job, Salary)$

User OUTER JOIN to retain UserIDs that don't own a car

Subqueries in SELECT: Special Cases

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
```

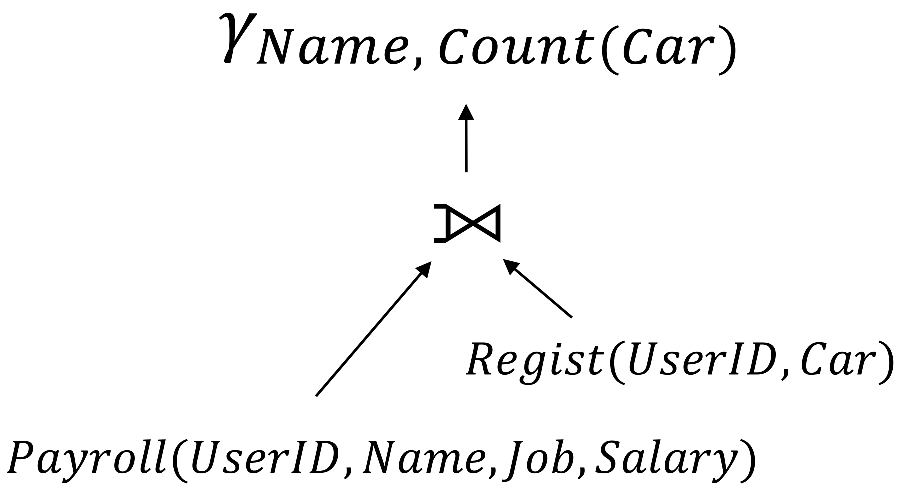
```
SELECT P.Name, COUNT(R.Car)
 FROM Payroll AS P
 LEFT OUTER JOIN Regist AS R
   ON P.UserID = R.UserID
GROUP BY P.Name
```

UserID	COUNT(Car)
123	1
345	0
567	2
789	0

=

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

UserID	Car
123	Charger
567	Civic
567	Pinto



$\gamma_{Name, Count(Car)}$

$\bowtie$

$Regist(UserID, Car)$

$Payroll(UserID, Name, Job, Salary)$

Subqueries in WHERE/HAVING

- Basic use: compare to scalar (single tuple single attribute)

Find the name who earns the highest salary

```
SELECT P.Name
  FROM Payroll P
 WHERE P.Salary = (SELECT MAX(P1.Salary)
                     FROM Payroll P1)
```

General strategy:
Join the condition into
a new column

- Uncorrelated subquery
- Need to separate before can draw RA
- We've seen this before: simple witness problem

Subqueries in WHERE/HAVING

- Basic use: compare to scalar (single tuple single attribute)

Find the name who earns the highest salary

for each job type

```
SELECT P.Name
  FROM Payroll P
 WHERE P.Salary = (SELECT MAX(P1.Salary)
                     FROM Payroll P1
                    WHERE P.Job = P1.Job)
```

- Correlated subquery
  - Like SELECT subquery, evaluated per tuple
- Like the witness problem

Subqueries in WHERE/HAVING

- Advanced keywords:
  - ANY → ∃
  - ALL → ∀

    Use with a condition (=, >, <, …)
  - (NOT) IN → (∉) ∈ Check if a tuple is (not) part of a relation
  - (NOT) EXISTS → Check if a relation is (not) empty

Subqueries in WHERE/HAVING

SQLite does not support ANY or ALL

- Advanced keywords:
  - ANY → ∃
  - ALL → ∀

    Use with a condition (=, >, <, …)
  - (NOT) IN → (∉) ∈ Check if a tuple is (not) part of a relation
  - (NOT) EXISTS → Check if a relation is (not) empty

Again: Find the name who earns the highest salary *for each job type*

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

Subqueries in WHERE/HAVING

- Advanced keywords:
  - ANY → ∃
  - ALL → ∀

    Use with a condition (=, >, <, …)
  - (NOT) IN → (∉) ∈

    Check if a tuple is (not) part of a relation
  - (NOT) EXISTS →

    Check if a relation is (not) empty

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

Subqueries in WHERE/HAVING

- Advanced keywords:
  - ANY → ∃
  - ALL → ∀

    Use with a condition (=, >, <, …)
  - (NOT) IN → (∉) ∈ Check if a tuple is (not) part of a relation
  - (NOT) EXISTS → Check if a relation is (not) empty

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)
```

De-correlated! But not enough to draw RA (no "NOT IN" operator)

Subqueries in WHERE/HAVING

- Advanced keywords:
  - ANY → ∃
  - ALL → ∀

Use with a condition (=, >, <, …)

  - (NOT) IN → (∉) ∈ — Check if a tuple is (not) part of a relation
  - (NOT) EXISTS → — Check if a relation is (not) empty

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

To convert to an RA Plan,
rewrite using UNION, INTERSECT, or EXCEPT

```
WHERE P.UserID NOT IN (SELECT UserID
                        FROM Regist)
```

De-correlated! But not enough to draw RA (no "NOT IN" operator)

Subqueries in WHERE/HAVING

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)
```

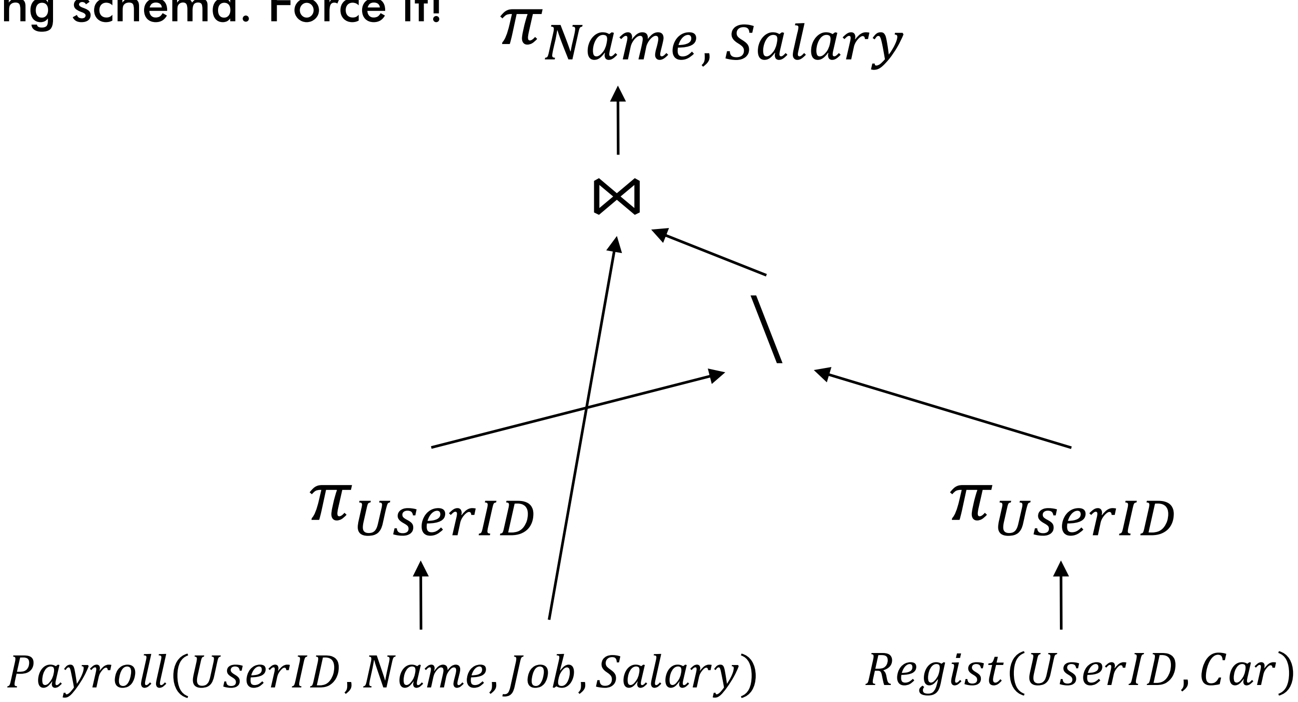
- This query takes Payroll UserIDs and subtracts Regist UserIDs
- Subtraction = EXCEPT
- EXCEPT requires matching schema. Force it!

Subqueries in WHERE/HAVING

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)
```

- This query takes Payroll UserIDs and subtracts Regist UserIDs
- Subtraction = EXCEPT
- EXCEPT requires matching schema. Force it!



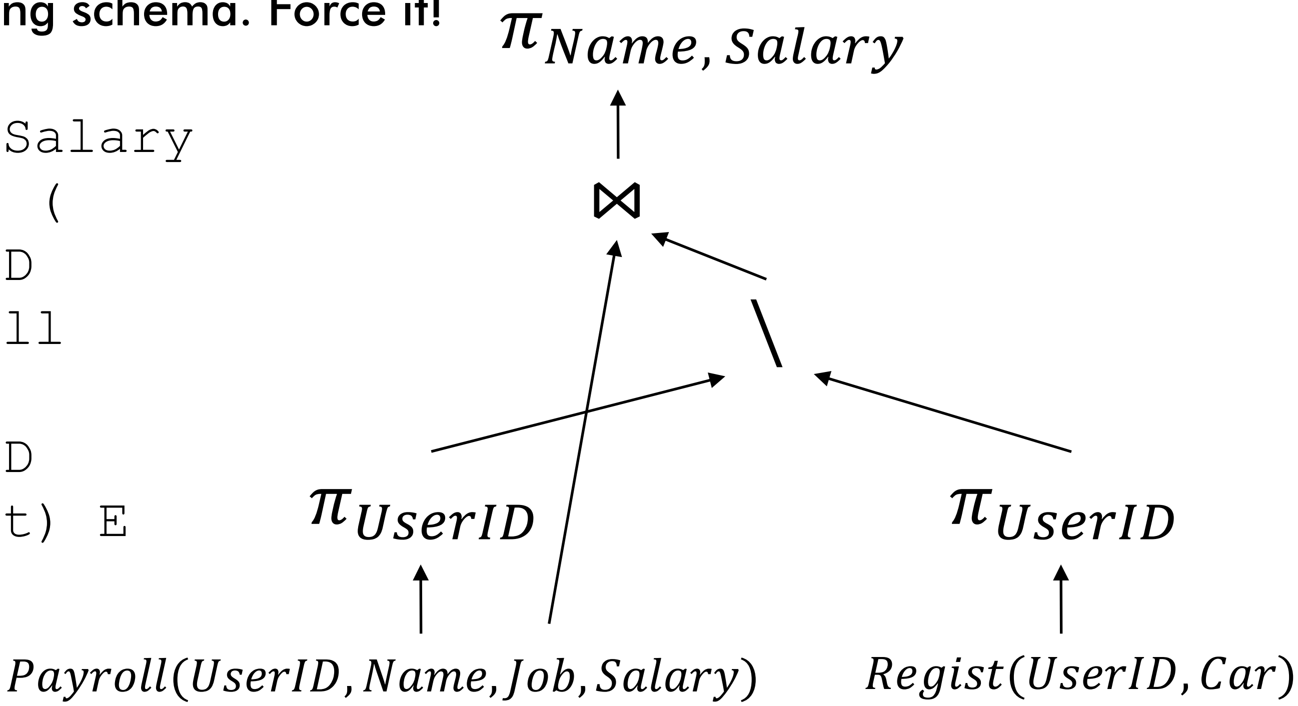
Subqueries in WHERE/HAVING

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)
```

- This query takes Payroll UserIDs and subtracts Regist UserIDs
- Subtraction = EXCEPT
- EXCEPT requires matching schema. Force it!

```
SELECT P.Name, P.Salary
  FROM Payroll P, (
     SELECT UserID
       FROM Payroll
     EXCEPT
     SELECT UserID
       FROM Regist) E
 WHERE P.UserID
     = E.UserID
```



$\pi_{Name, Salary}$

$\bowtie$

$\setminus$

π_{UserID}

π_{UserID}

$Payroll(UserID, Name, Job, Salary)$

$Regist(UserID, Car)$

Hard Cases: Universal Quantifiers

Find jobs whose employees **all** own cars

- All = “every employee must own a car”
- Hard to compute directly
- Try computing the negation!

Payroll

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

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto
789	Bugatti

Output

Job
Prof

Hard Cases: Universal Quantifiers

Find jobs whose employees **all** own cars

→ ∀ employee e ∈ job, e owns a car

∀ = "for all"

Try computing the negation!

→ ¬(∀ employee e ∈ job, e owns a car)

¬ = "not"

→ ∃ employee e ∈ job, ¬(e owns a car)

→ ∃ employee e ∈ job, e doesn't own a car

Find jobs with **an** employee who **doesn't** own a car

∃ = "there exists (at least one)"

De Morgan's Law

¬(∀x, c) = ∃x, ¬c

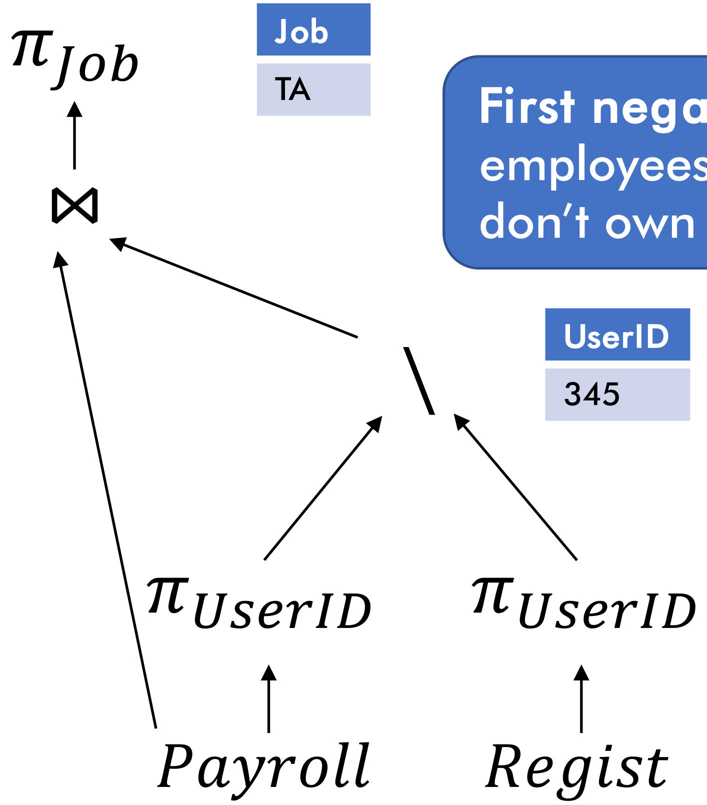
It's okay if you haven't seen logic symbols ∀, ∃, ¬ before. Think about how to logically negate an English sentence

Hard Cases: Universal Quantifiers

Find jobs whose employees *all* own cars

➔ Find jobs with **an** employee who **doesn't** own a car

Jobs of employees who don't own a car



Job
TA

First negation: employees who don't own a car

UserID
345

Payroll

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

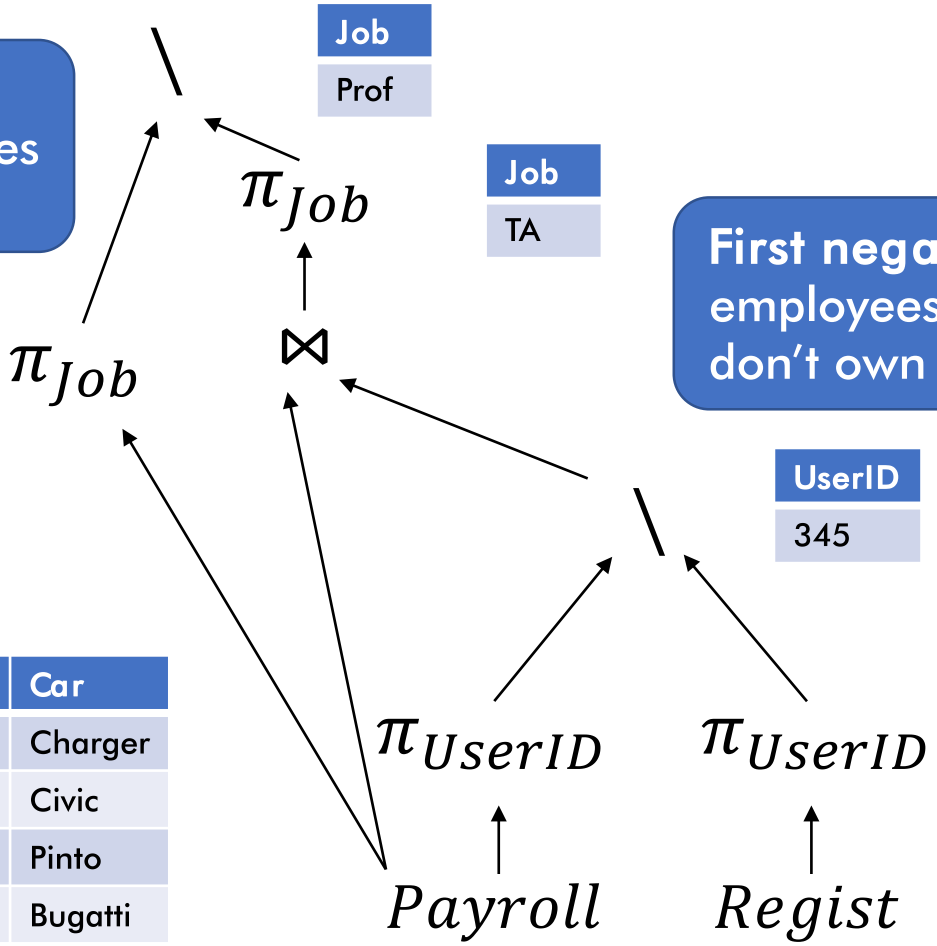
Regist

UserID	Car
123	Charger
567	Civic
567	Pinto
789	Bugatti

Hard Cases: Universal Quantifiers

Find jobs whose employees *all* own cars

→ Find jobs with **an** employee who **doesn't** own a car



Job
Prof

Job
TA

UserID
345

Second negation: jobs whose employees all own a car

First negation: employees who don't own a car

Double negation pattern!

Payroll

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

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto
789	Bugatti

Hard Cases: Universal Quantifiers

- Watch out for universal quantifiers
  - Require more complex answer
- *Double negation* pattern often works

$$\forall = \neg \exists \neg$$