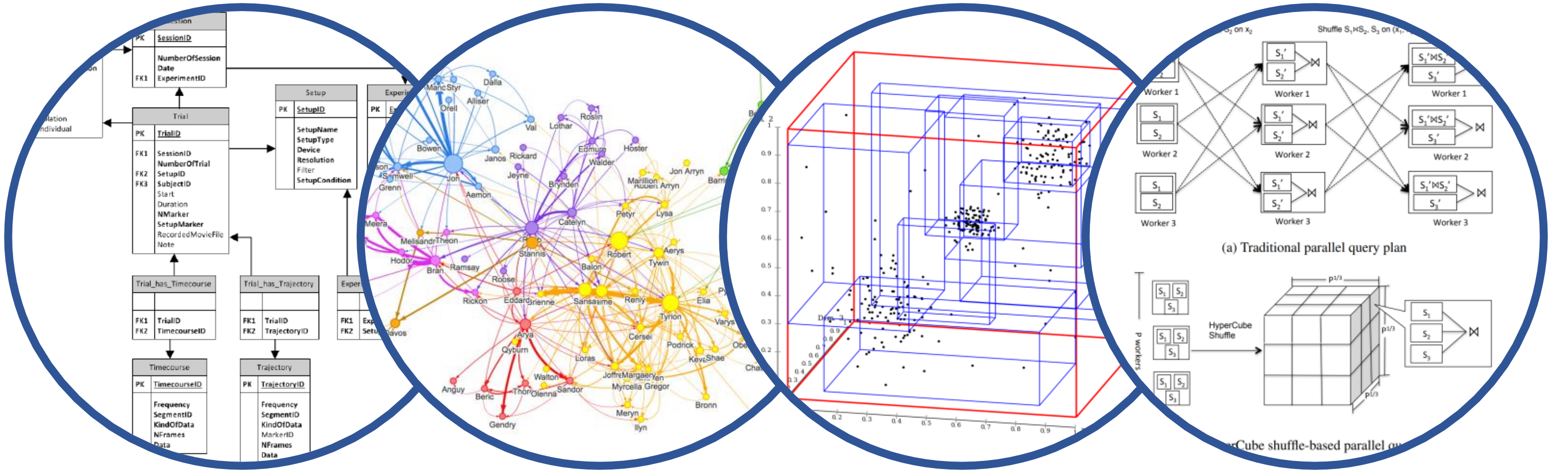


Introduction to Data Management

query in a query in a query in… RA?

Shana Hutchison

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

Coming soon…

- Hw3 uses Microsoft Azure
  - Run queries on a cloud database!
  - Everyone will have their own database
  - Same teams hw2 & hw3
- Look out for email from invites@microsoft.com

Recap – Subqueries I

- Witnessing problem again and again
- Subqueries
  - Set/bag UNION, INTERSECT, EXCEPT
  - FROM
  - SELECT

Today – Subqueries II & SQL-RA

- WHERE/HAVING subqueries
- More
  - Decorrelation: correlated to uncorrelated subquery
  - Un-nesting: eliminate subquery
- Universal quantification queries
- SQL to RA
- RA to SQL
- RA to RA

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

- 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

SQLite does not support ANY or ALL

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 → ∀
  - (NOT) IN → (∉) ∈
  - (NOT) EXISTS →

Use with a condition (=, >, <, …)

Check if a tuple is (not) part of a relation

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 → $\exists$
  - ALL → $\forall$

    Use with a condition (=, >, <, …)
  - (NOT) IN → ($\notin$) $\in$

    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)
```

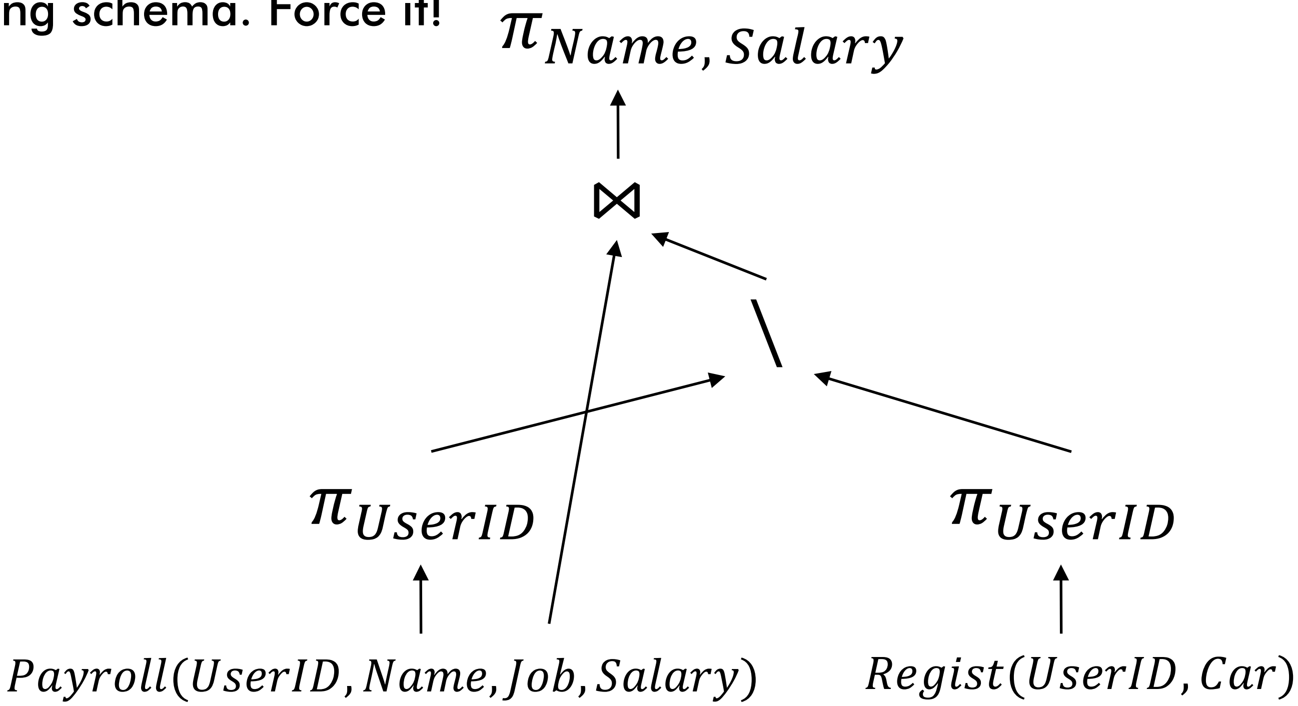
- This query takes Payroll UserIDs and subtracts Regist UserIDs
- Subtraction = EXCEPT
- EXCEPT requires matching schema. Force it!

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!



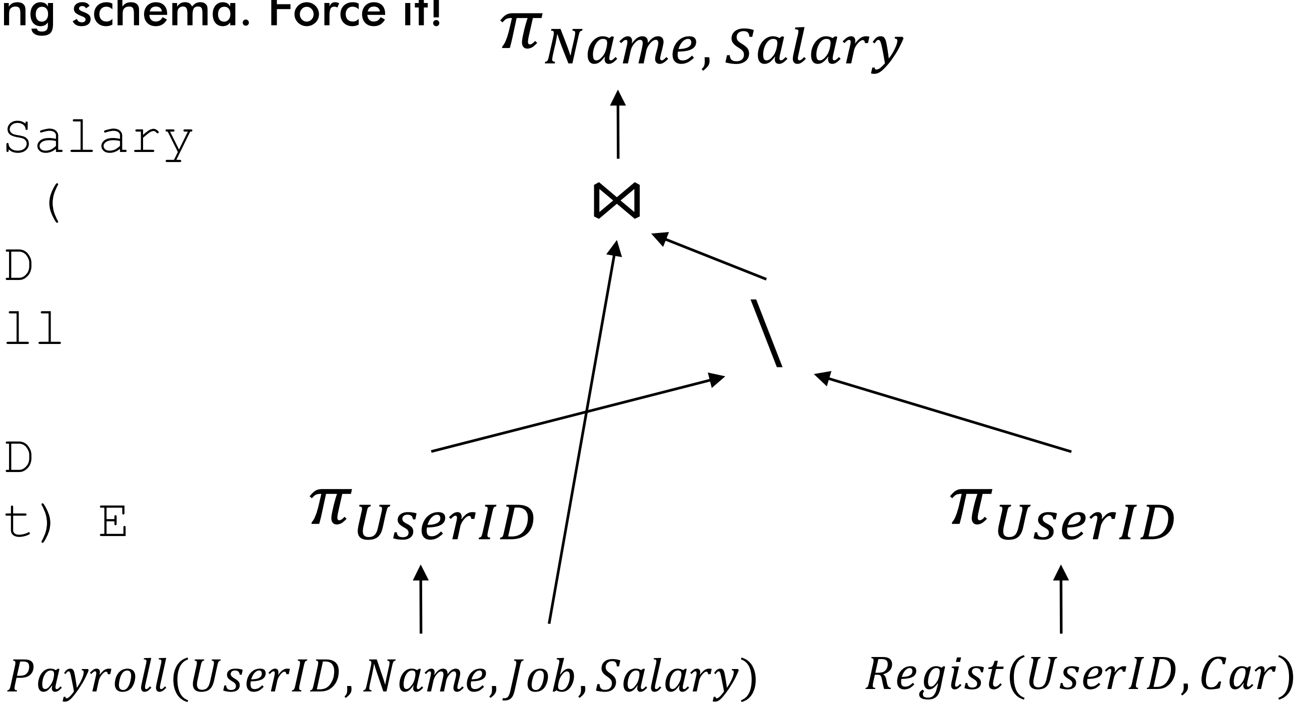
Subqueries in WHERE/HAVING

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
```



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

$$\neg\forall x, f(x) = \exists x, \neg f(x)$$

It’s okay if you haven’t seen logic symbols ∀, ∃, ¬ before.

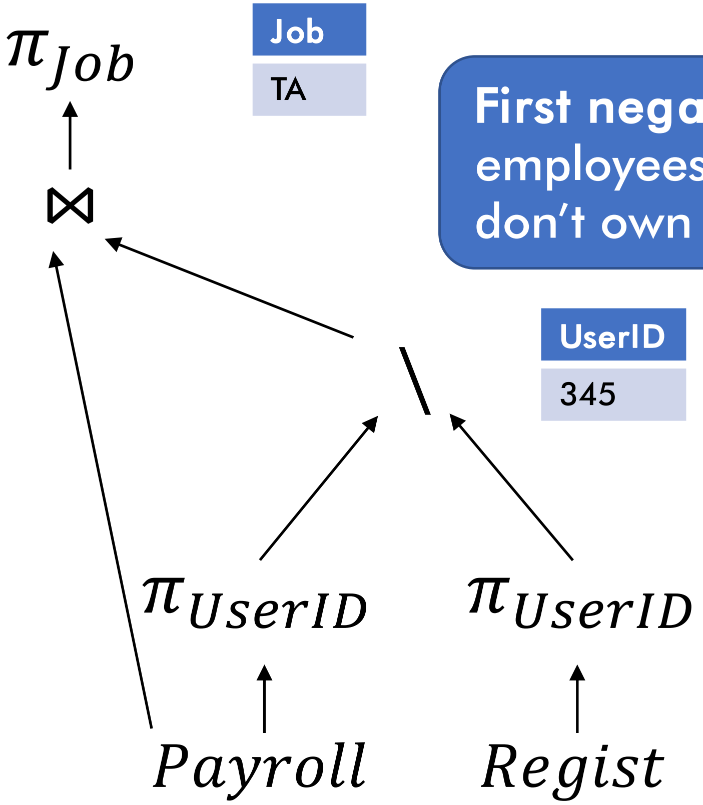
Think about how to logically negate an English sentence

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

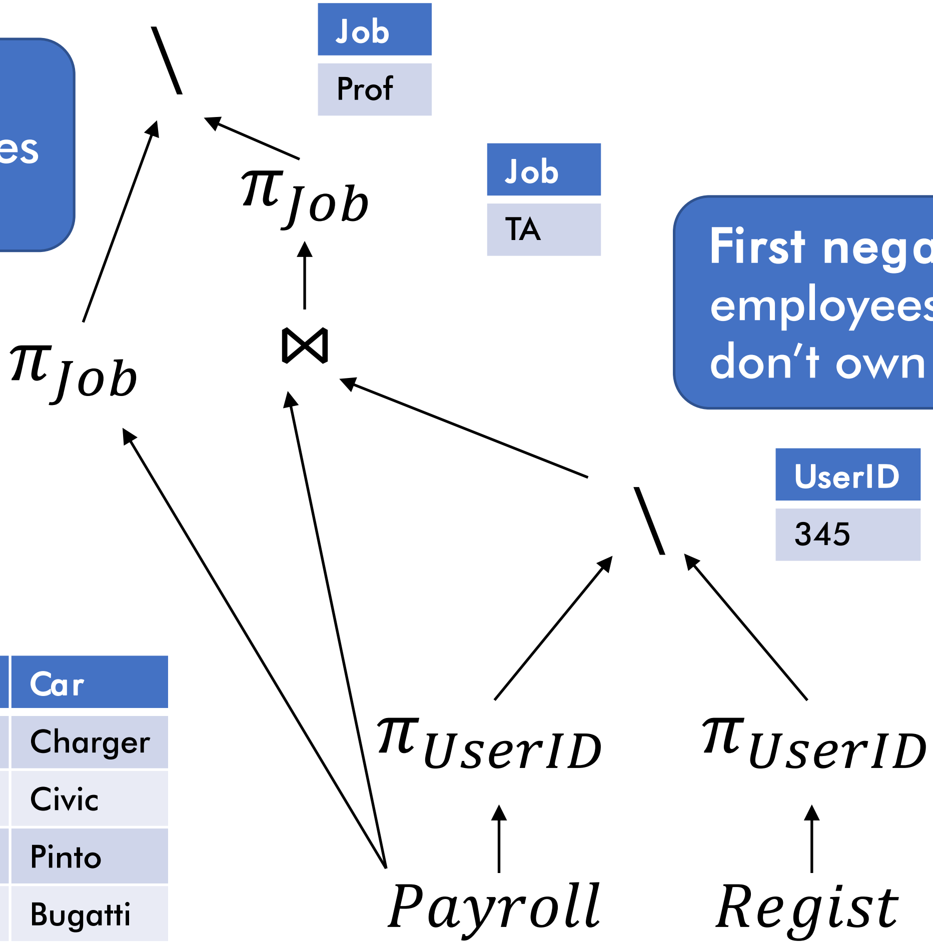
Hard Cases: Universal Quantifiers

Find jobs whose employees *all* own cars

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

Second negation: jobs whose employees all own a car

Double negation pattern!



Job
Prof

Job
TA

UserID
345

First negation: employees who don't own a car

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

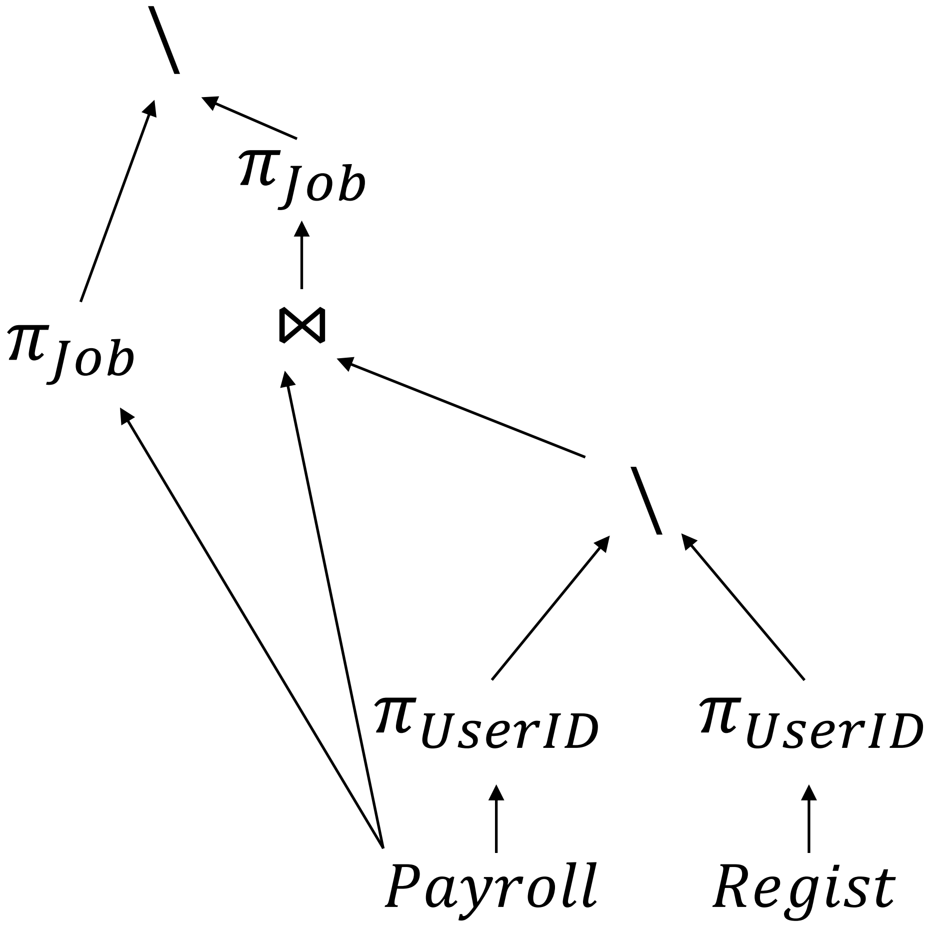
- Watch out for universal quantifiers
  - Require more complex answer
- *Double negation* pattern often works
  - aka the "not not rule"

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

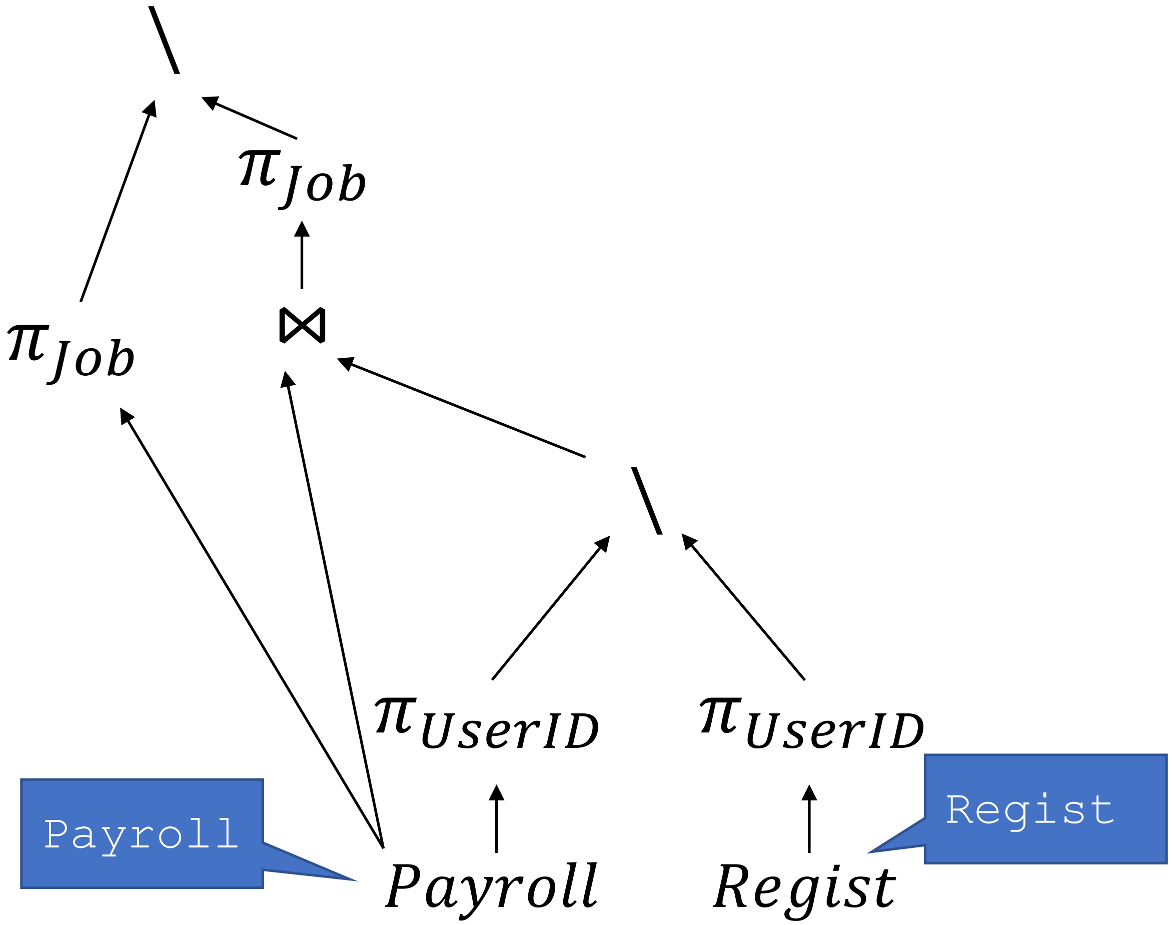
General SQL to RA

- Gee, converting SQL w/ subquery to RA is hard
  - Correlated EXISTS, IN, ANY, ALL
  - Universal quantifiers
- Is there a general algorithm?
- Well…
  - (Fun paper) Translating SQL into the Relational Algebra
  - (Hardcore 2015 paper for the mathematically inclined) Unnesting Arbitrary Queries by Neumann et al
- Advice for this class: Think!
  - What is the SQL doing?
  - Describe it in words
  - Can we use big RA operators like ∪, ∩, \ ?

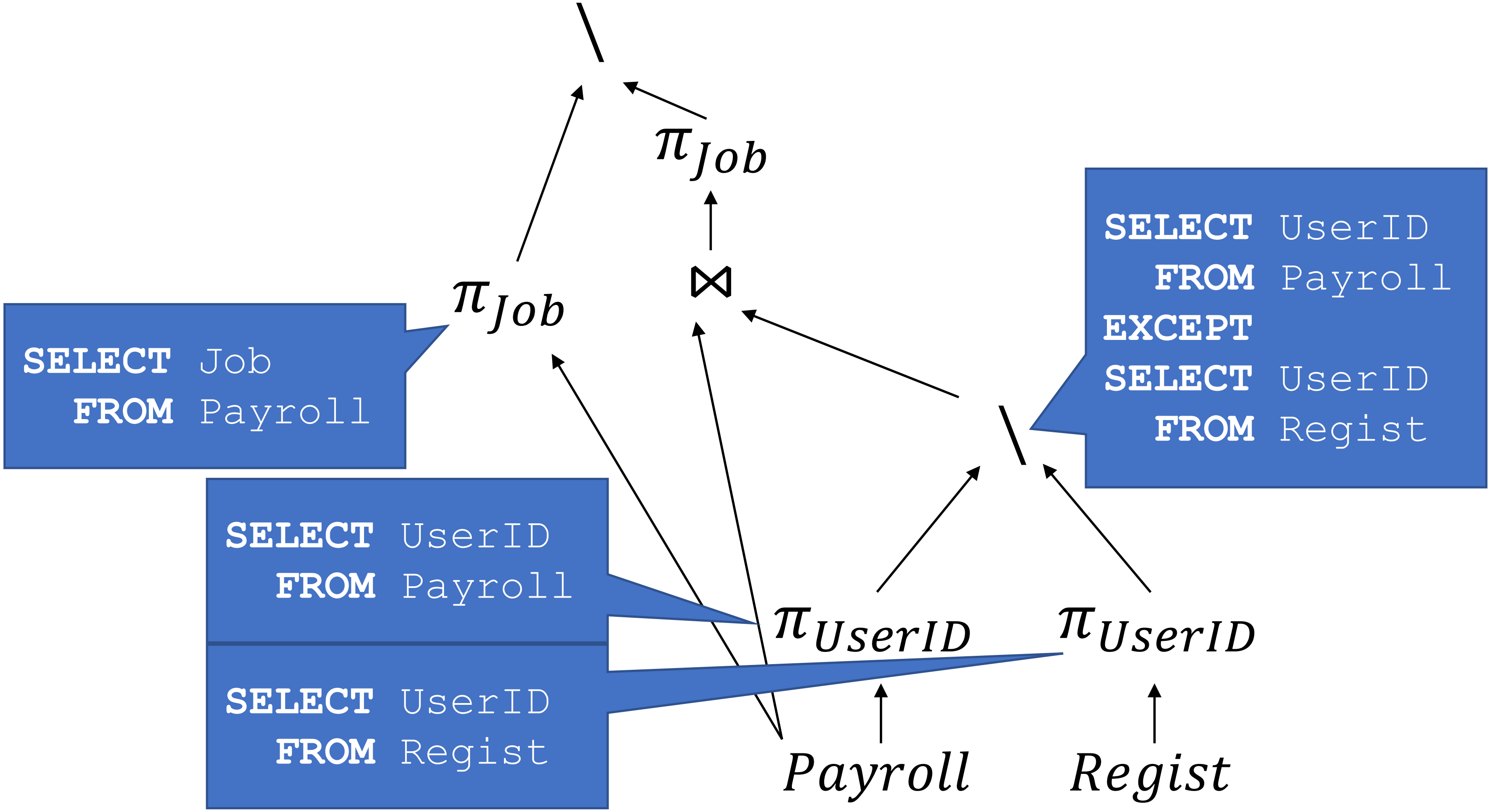
RA to SQL: Bottom-up



RA to SQL: Bottom-up



RA to SQL: Bottom-up



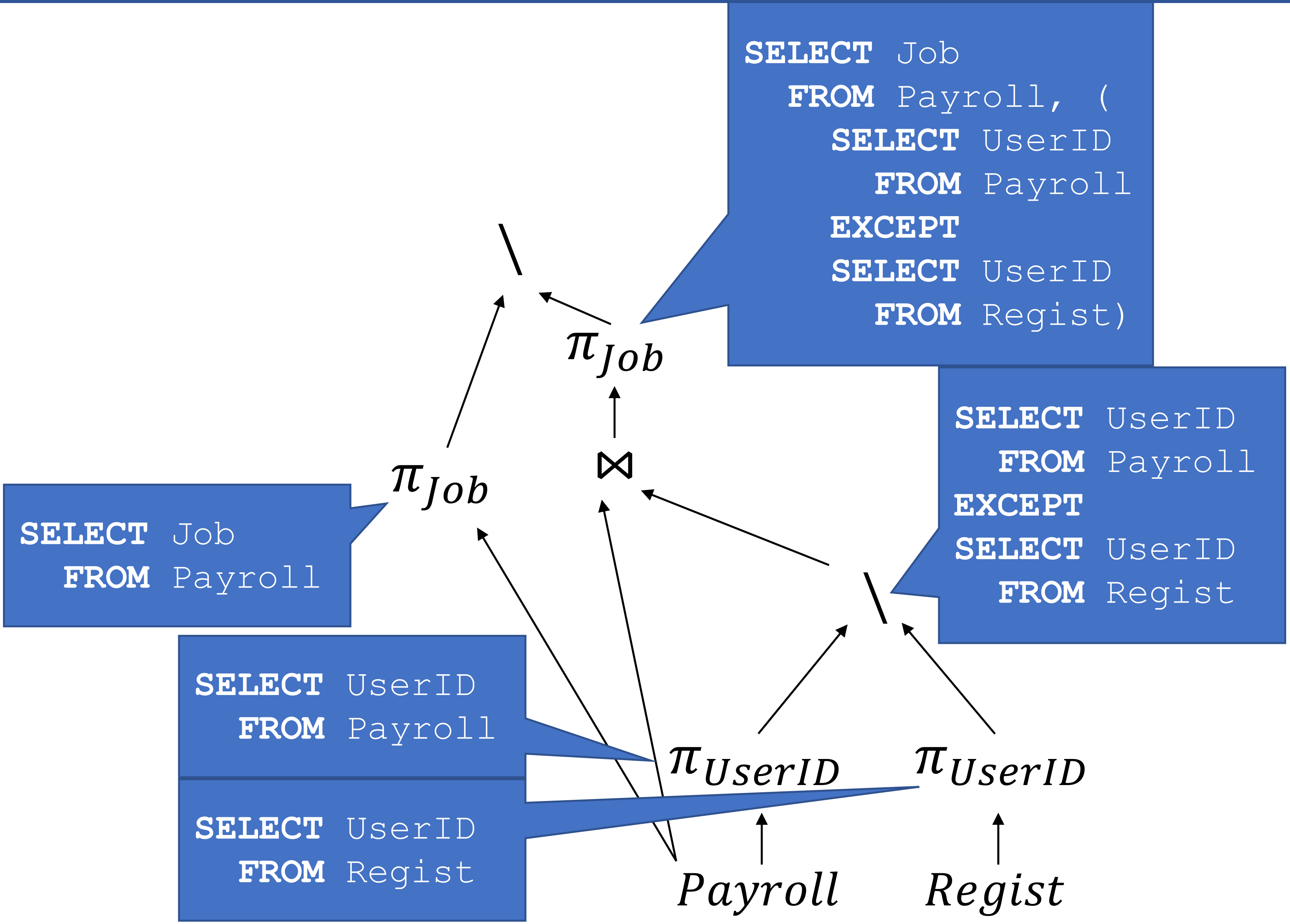
```
SELECT UserID
  FROM Payroll
EXCEPT
SELECT UserID
  FROM Regist
```

```
SELECT Job
  FROM Payroll
```

```
SELECT UserID
  FROM Payroll
```

```
SELECT UserID
  FROM Regist
```

RA to SQL: Bottom-up



```
SELECT Job
  FROM Payroll, (
    SELECT UserID
      FROM Payroll
    EXCEPT
    SELECT UserID
      FROM Regist)
```

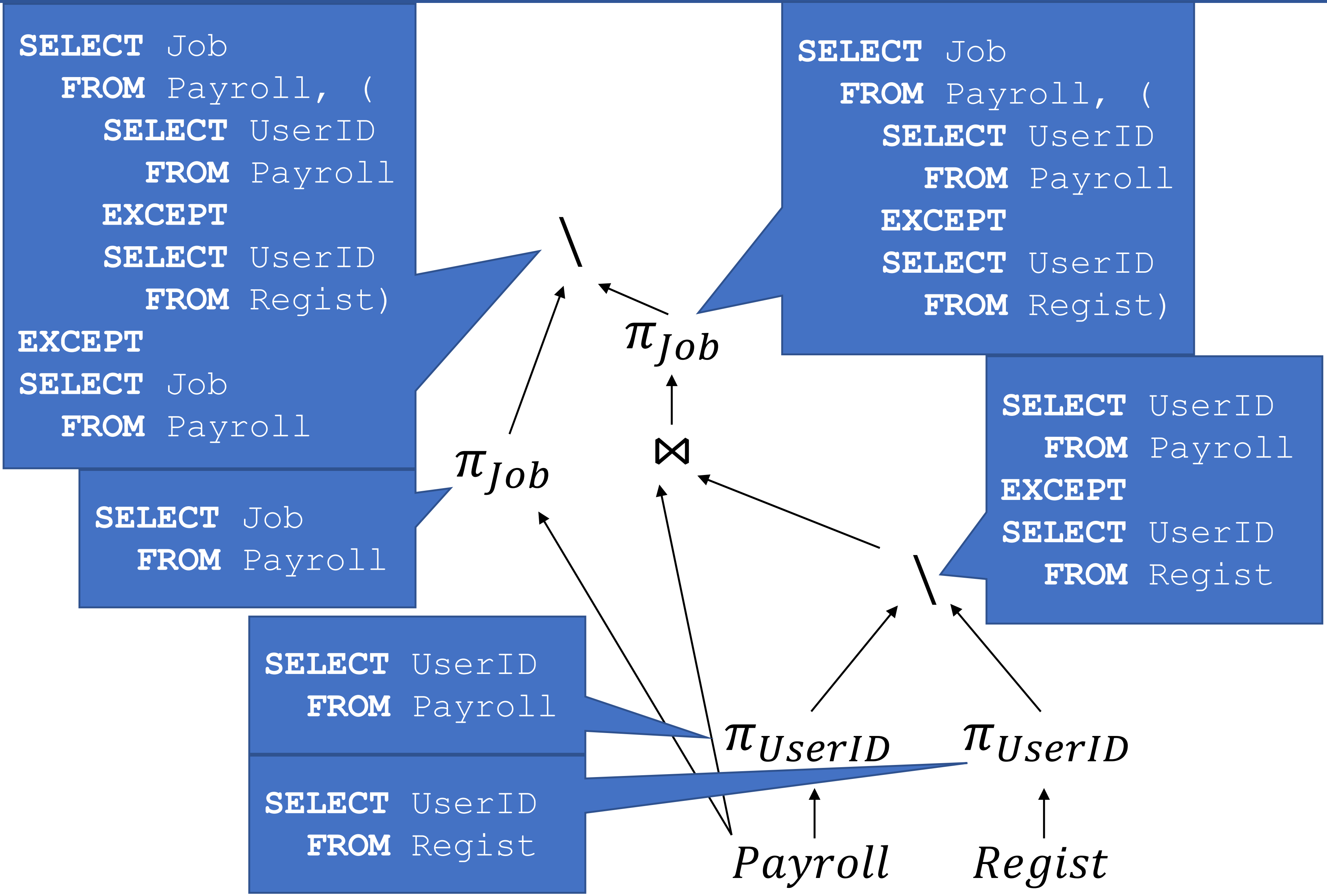
```
SELECT UserID
  FROM Payroll
EXCEPT
SELECT UserID
  FROM Regist
```

```
SELECT Job
  FROM Payroll
```

```
SELECT UserID
  FROM Payroll
```

```
SELECT UserID
  FROM Regist
```

RA to SQL: Bottom-up



```
SELECT Job
  FROM Payroll, (
    SELECT UserID
      FROM Payroll
    EXCEPT
    SELECT UserID
      FROM Regist)
EXCEPT
SELECT Job
  FROM Payroll
```

```
SELECT Job
  FROM Payroll, (
    SELECT UserID
      FROM Payroll
    EXCEPT
    SELECT UserID
      FROM Regist)
```

```
SELECT UserID
  FROM Payroll
EXCEPT
SELECT UserID
  FROM Regist
```

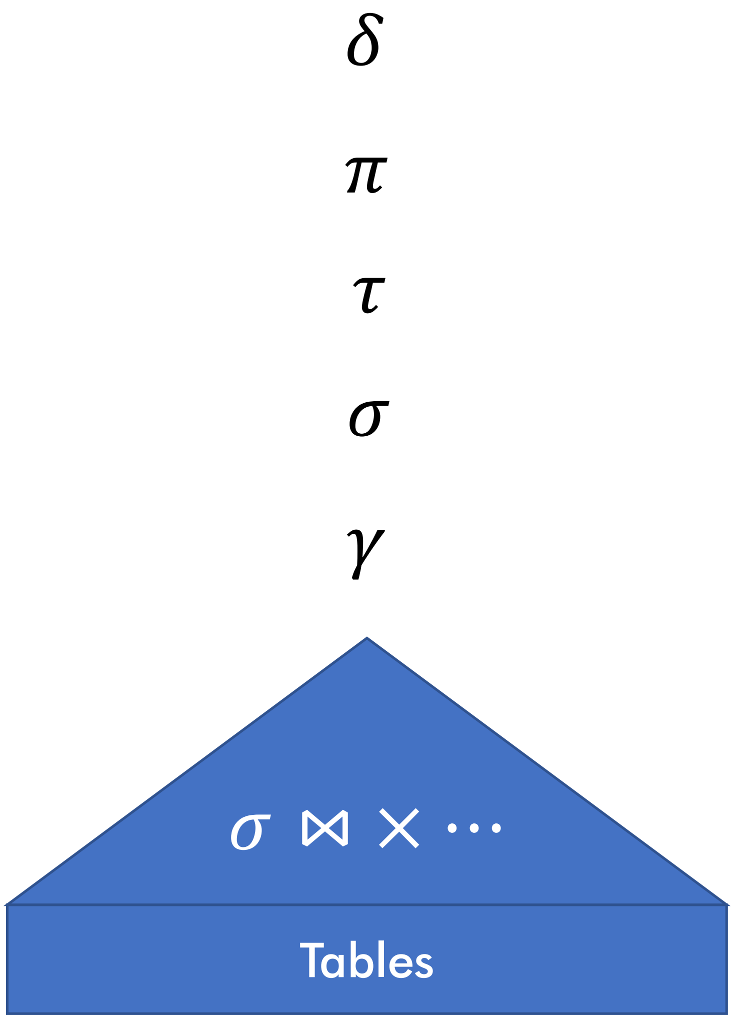
```
SELECT Job
  FROM Payroll
```

```
SELECT UserID
  FROM Payroll
```

```
SELECT UserID
  FROM Regist
```

General RA to SQL

- Easy! Build SQL bottom-up w/ subqueries
- Simplify: a single query can capture FWGHOS

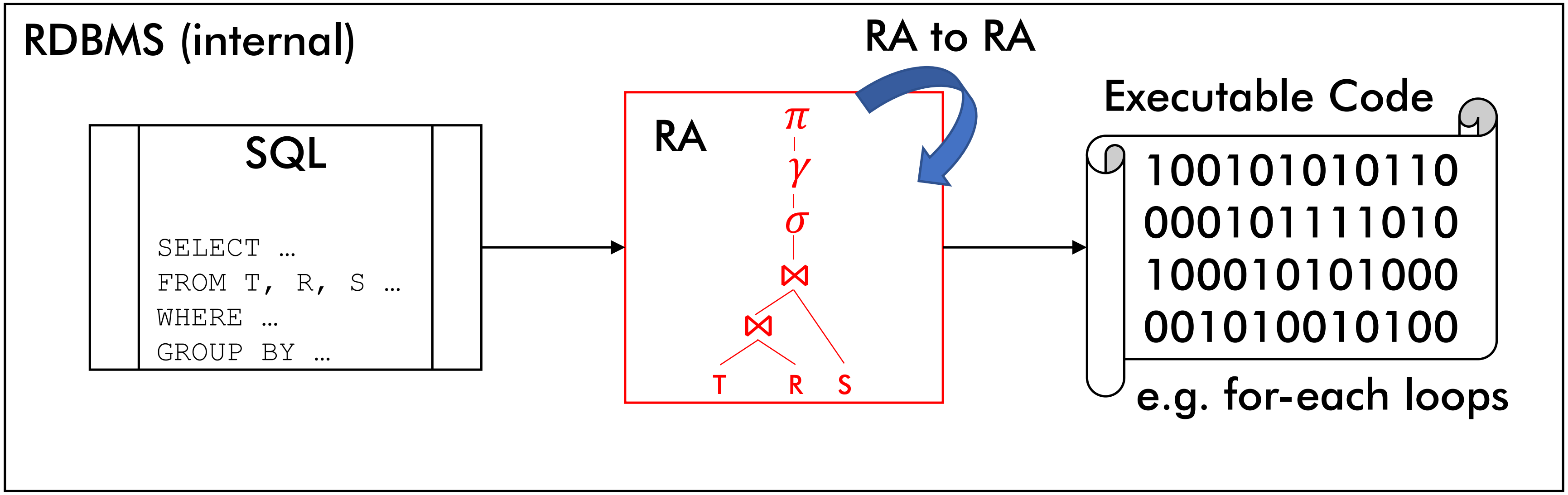


Hey, What's the Point of RA?

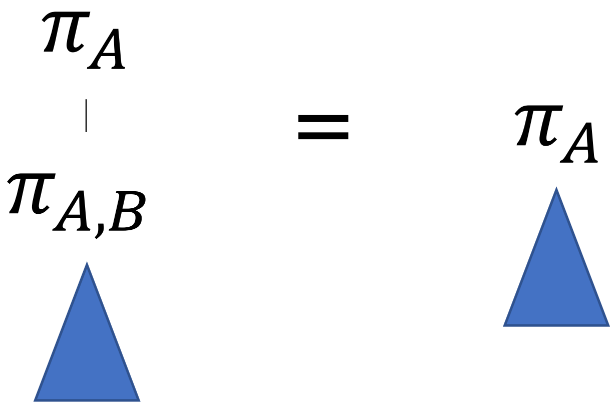
Overview of query optimization

1. RDBMS converts SQL to RA
2. Explore equivalent RA plans
3. Find the RA plan with cheapest estimated cost
4. Convert RA to code and execute

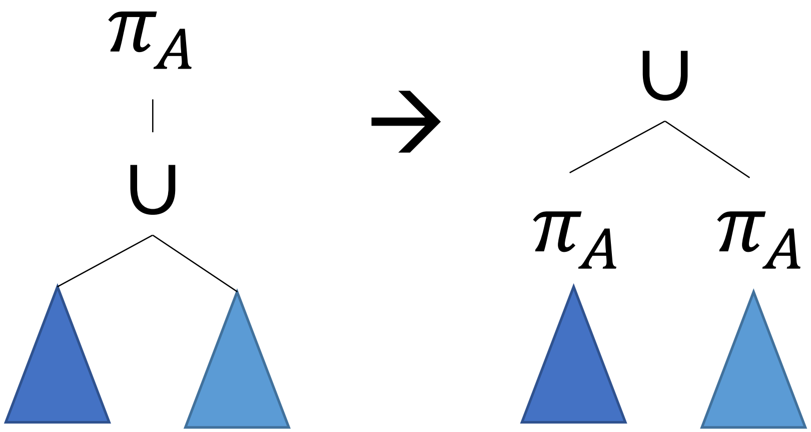
Today: RA to RA
Later: Estimate cost for an RA Plan



RA to RA: SELECT



RA to RA: SELECT-UNION

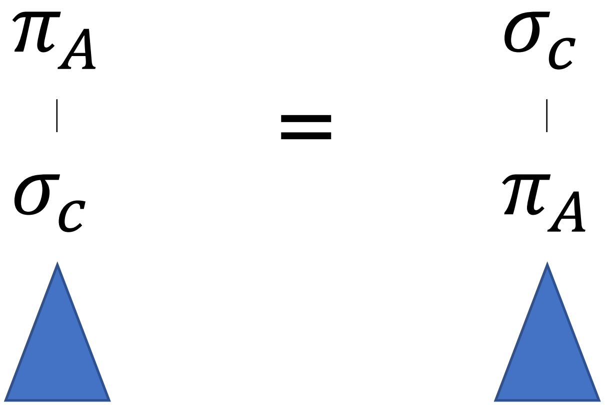


For ←, ensure schema matches

NOT for intersection, difference

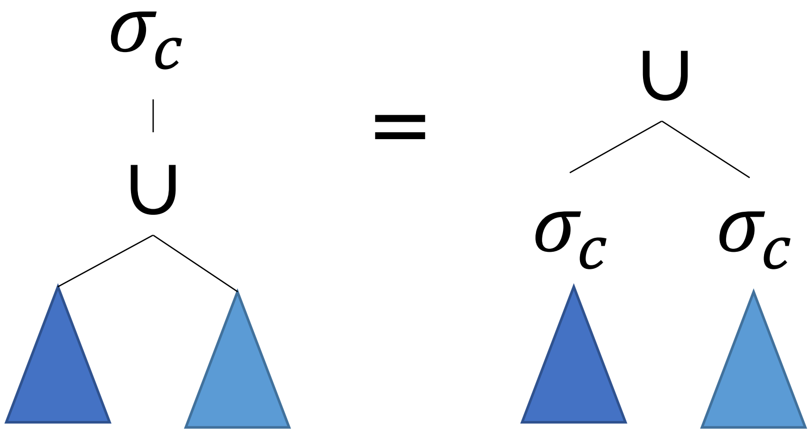
Might need ⊎ if duplicates involved, be careful

RA to RA: SELECT-WHERE

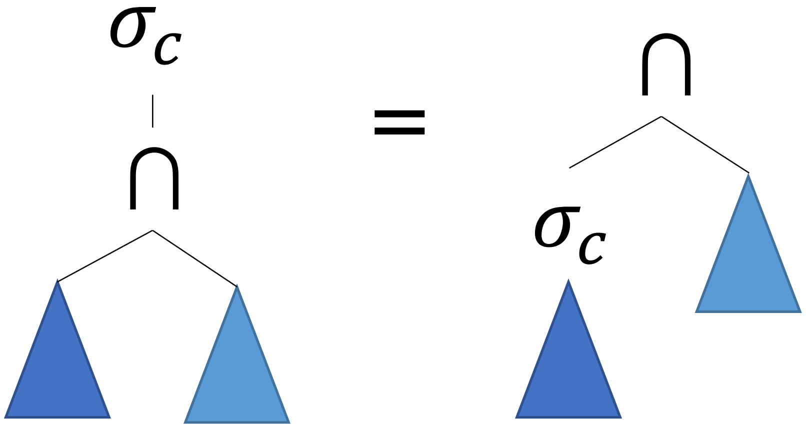


If c only references attributes in A

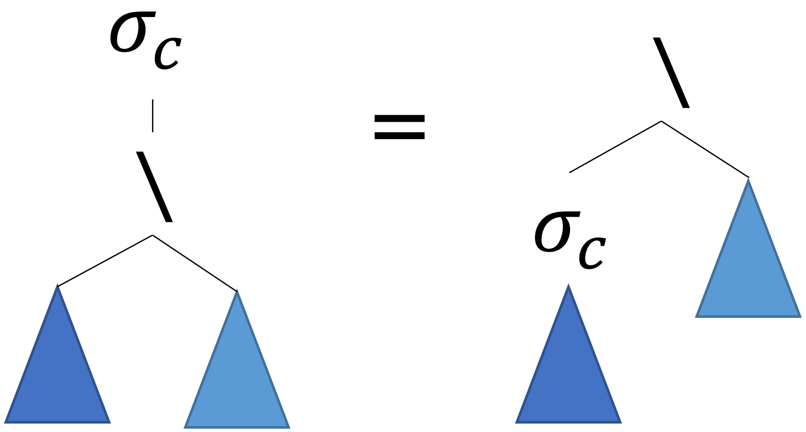
RA to RA: WHERE-UNION



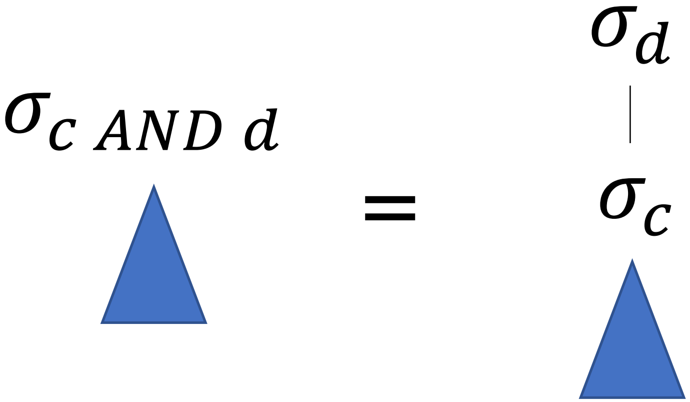
RA to RA: WHERE-INTERSECT



RA to RA: WHERE-EXCEPT

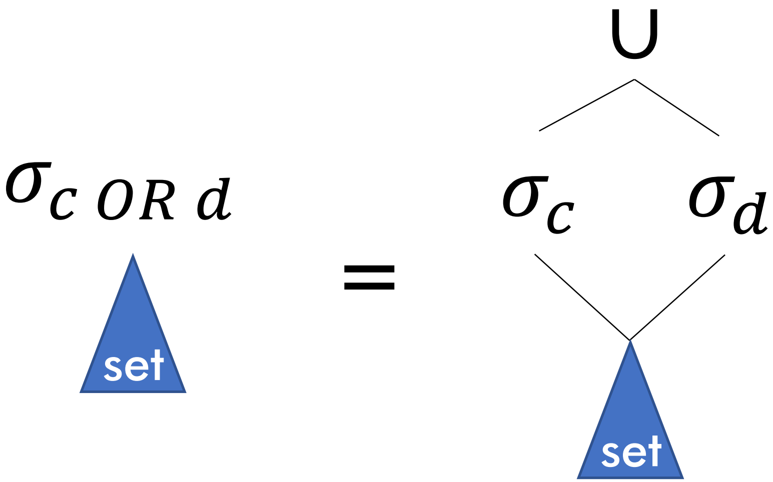


RA to RA: WHERE-AND



$\sigma_{c\ AND\ d}$ $=$ σ_d σ_c

RA to RA: WHERE-OR



$\sigma_{c\ OR\ d}$

set

=

∪

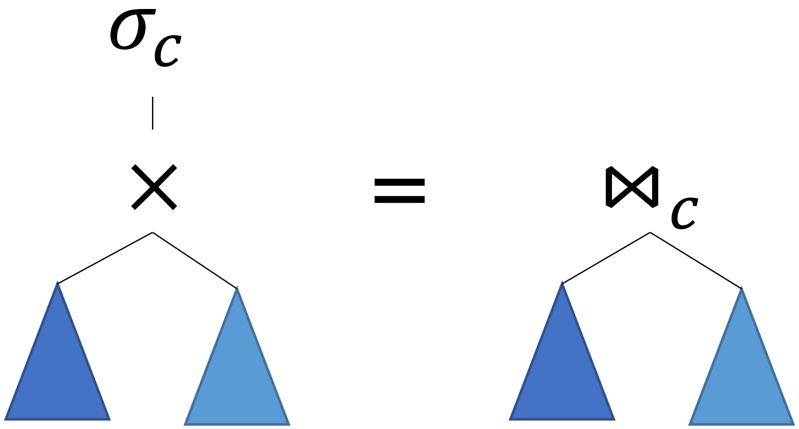
σ_c

σ_d

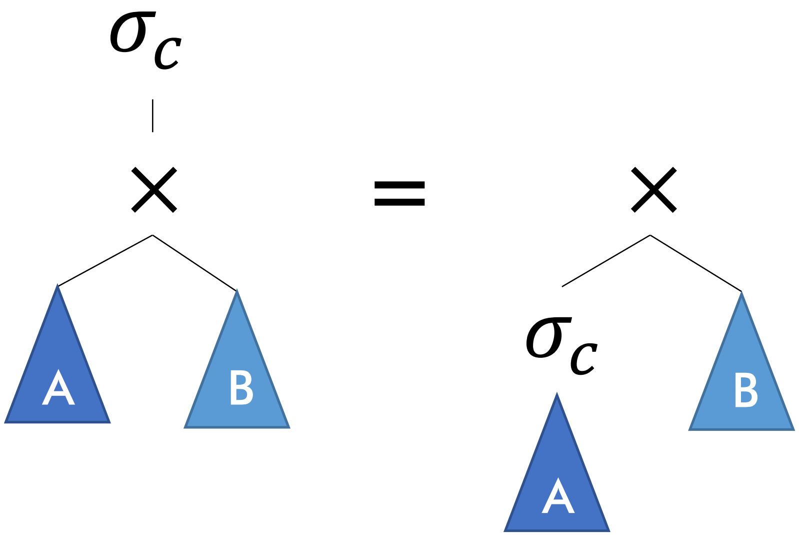
set

Watch out for duplicates

RA to RA: CARTESIAN-JOIN

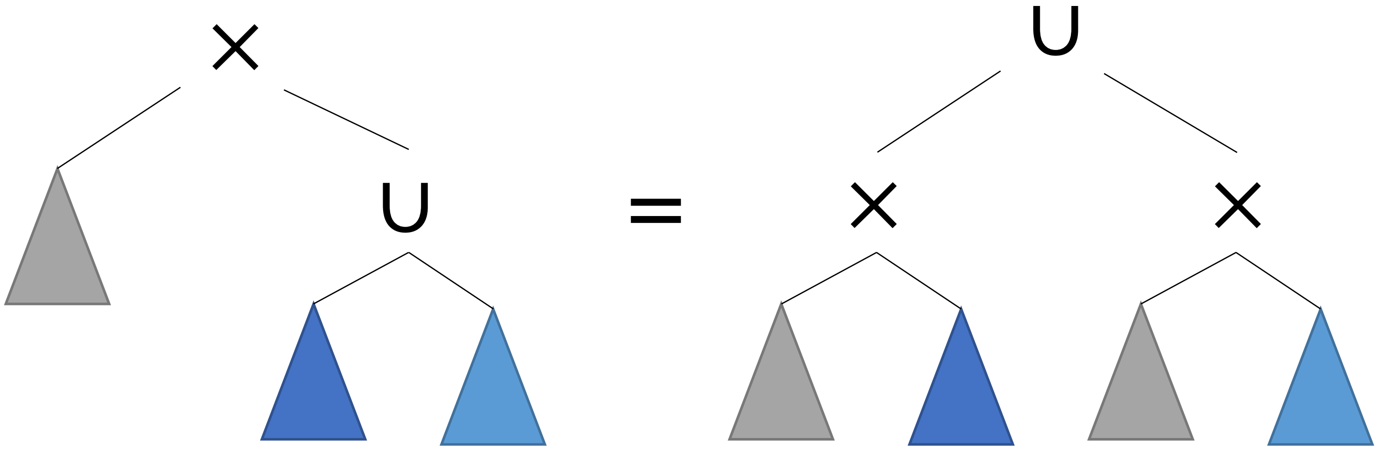


RA to RA: WHERE-JOIN



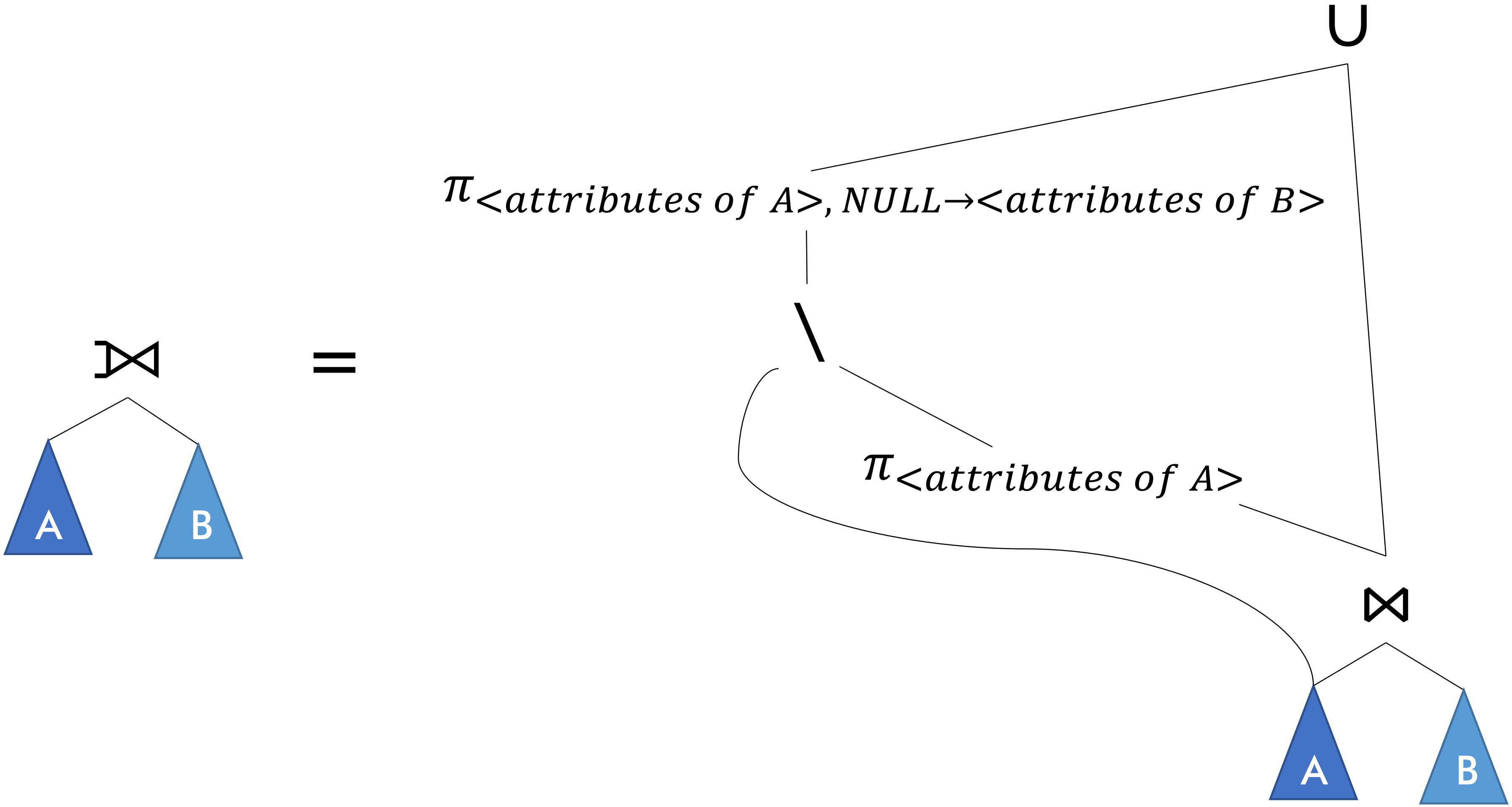
If c only references attributes in A

RA to RA: JOIN-UNION



Same for INTERSECT, EXCEPT

RA to RA: (LEFT) OUTER JOIN



$⟕$ $=$

$\cup$

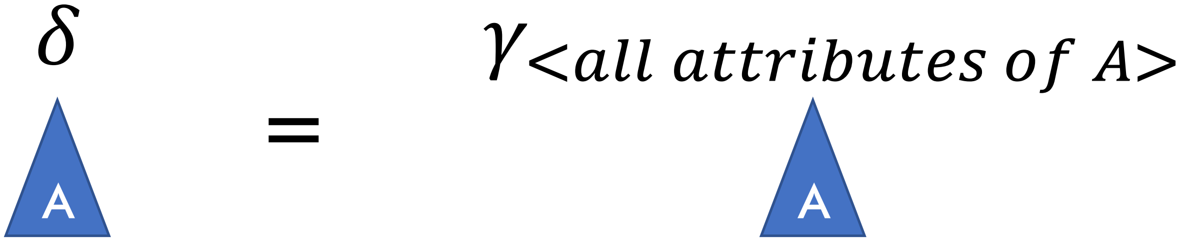
$\pi_{<attributes\ of\ A>,\ NULL \rightarrow <attributes\ of\ B>}$

$\setminus$

$\pi_{<attributes\ of\ A>}$

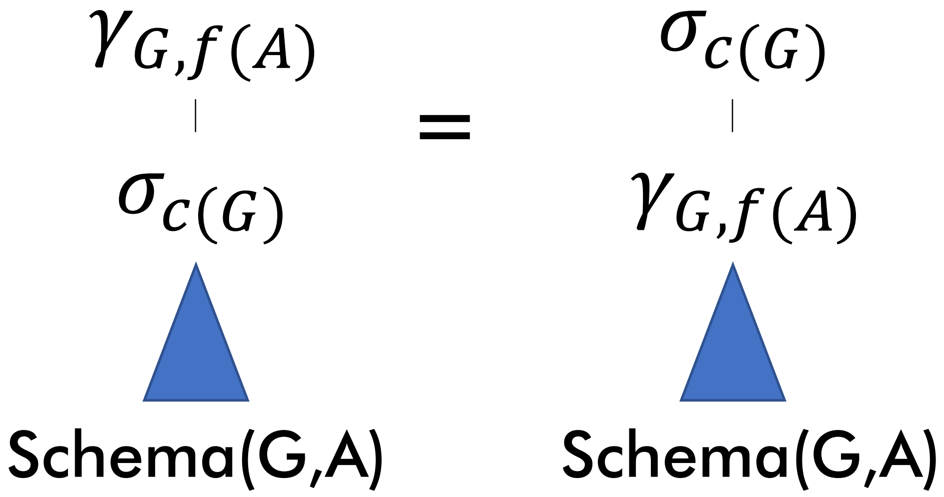
$\bowtie$

RA to RA: DISTINCT-GROUP BY



$$\delta = \gamma_{<all\ attributes\ of\ A>}$$

RA to RA: WHERE-HAVING



$\gamma_{G,f(A)}$

$\sigma_{c(G)}$

Schema(G,A)

$=$

$\sigma_{c(G)}$

$\gamma_{G,f(A)}$

Schema(G,A)

c only references attributes in G

RA to RA

- Plenty more equivalences
- How to remember?
  - **Think!**
  - Use the definitions

All Together

How many partnerships between TAs & Profs are possible where at least one member owns a car?

Payroll

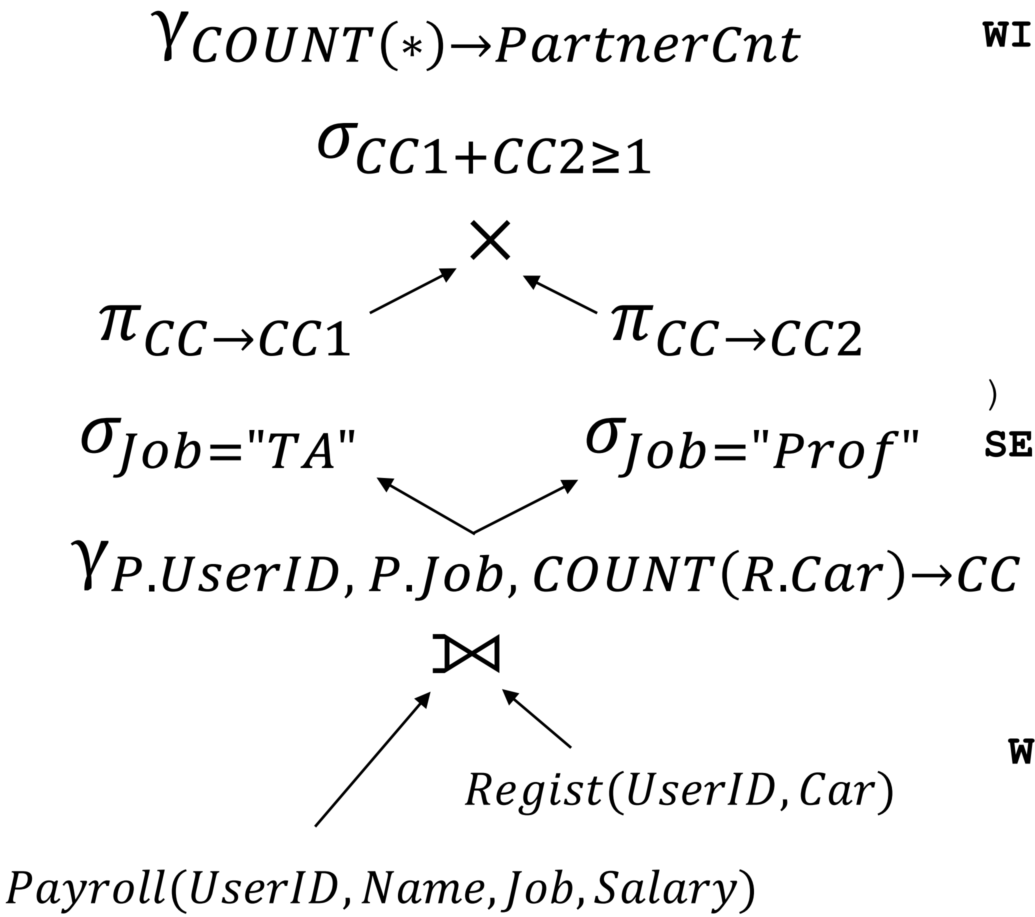
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

All Together

How many partnerships between TAs & Profs are possible where at least one member owns a car?



$\gamma_{COUNT(*) \rightarrow PartnerCnt}$

$\sigma_{CC1+CC2 \geq 1}$

$\times$

$\pi_{CC \rightarrow CC1}$

$\pi_{CC \rightarrow CC2}$

$\sigma_{Job="TA"}$

$\sigma_{Job="Prof"}$

$\gamma_{P.UserID, P.Job, COUNT(R.Car) \rightarrow CC}$

$\bowtie$

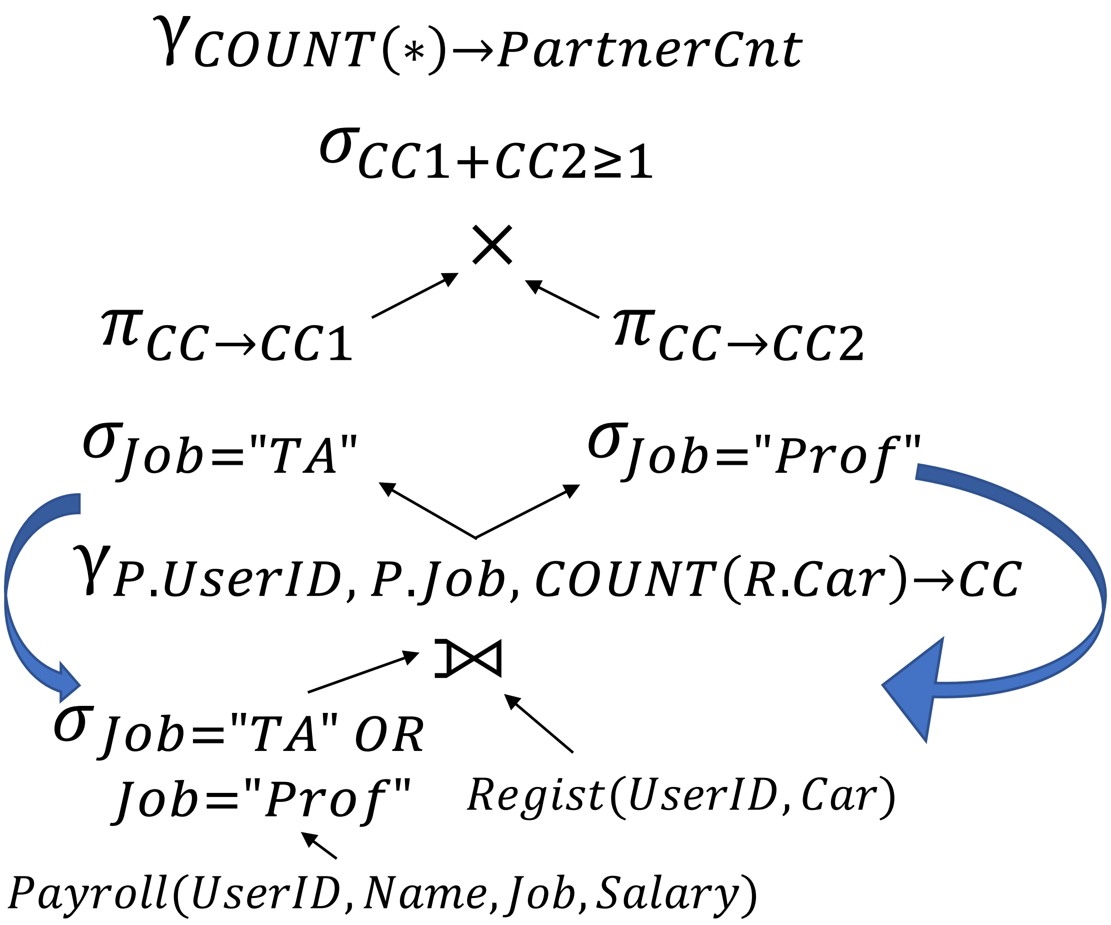
$Regist(UserID, Car)$

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

```
WITH Part1 AS (
   SELECT P.UserID, P.Job,
          COUNT(R.Car) AS CC
     FROM Payroll P
     LEFT OUTER JOIN Regist R
     ON P.UserID = R.UserID
    GROUP BY P.Job, P.UserID
)
SELECT COUNT(*) AS PartnerCount
   FROM
     (SELECT CC AS CC1 FROM Part1
       WHERE Job='TA') A,
     (SELECT CC AS CC2 FROM Part1
       WHERE Job='Prof') B
  WHERE A.CC1+B.CC2 >= 1
```

All Together

How many partnerships between TAs & Profs are possible where at least one member owns a car?



$\gamma_{COUNT(*)\rightarrow PartnerCnt}$

$\sigma_{CC1+CC2\geq 1}$

$\times$

$\pi_{CC\rightarrow CC1}$ $\pi_{CC\rightarrow CC2}$

$\sigma_{Job="TA"}$ $\sigma_{Job="Prof"}$

$\gamma_{P.UserID, P.Job, COUNT(R.Car)\rightarrow CC}$

$\bowtie$

$\sigma_{Job="TA" \ OR \ Job="Prof"}$

$Regist(UserID, Car)$

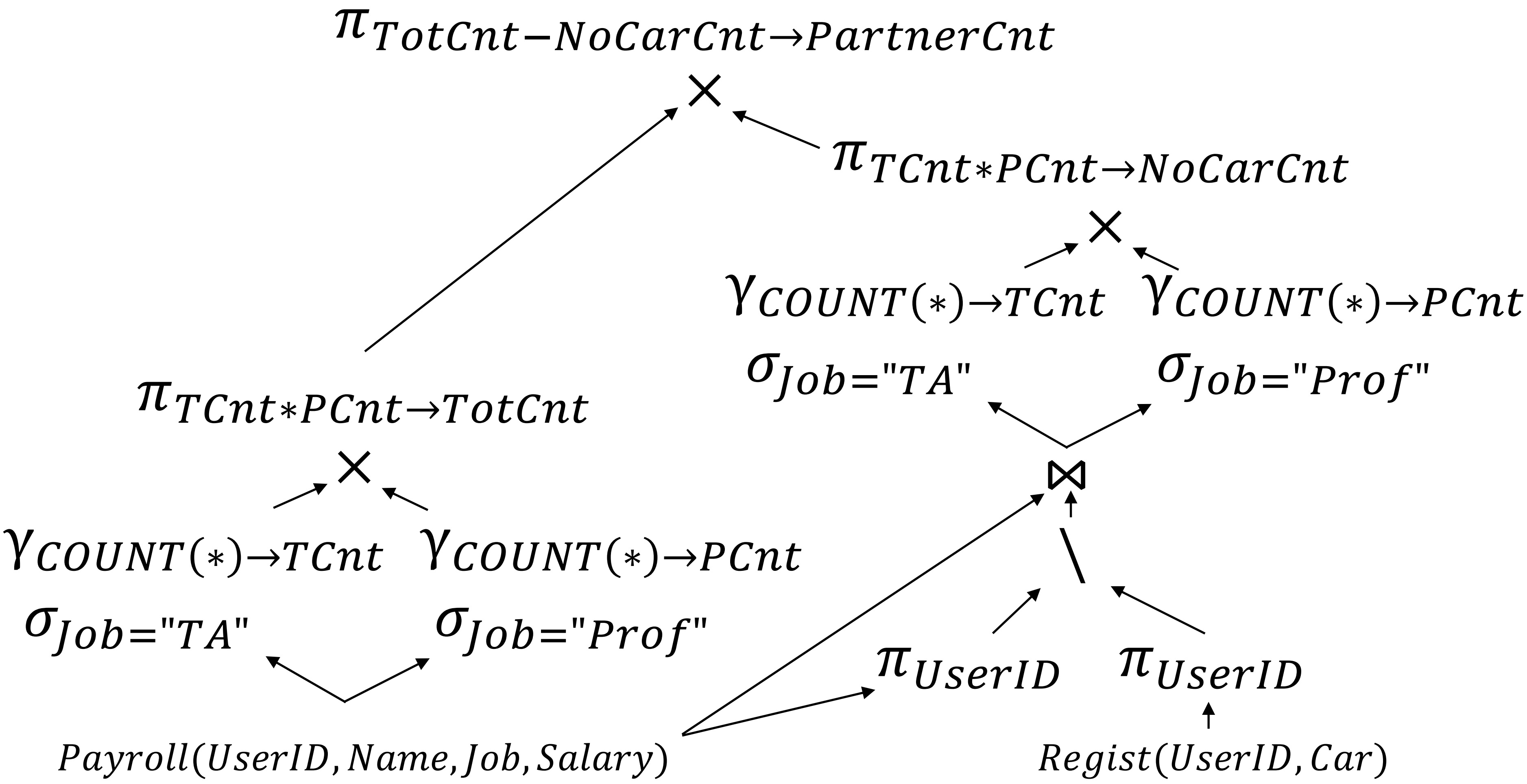
$Payroll(UserID, Name, Job, Salary)$

These σs commute through γ and $\bowtie$

Way more rewrites possible…

All Together

Another approach: take total # of partnerships and subtract those where neither owns a car



$\pi_{TotCnt-NoCarCnt \rightarrow PartnerCnt}$

$\times$

$\pi_{TCnt*PCnt \rightarrow NoCarCnt}$

$\times$

$\gamma_{COUNT(*) \rightarrow TCnt}$ $\gamma_{COUNT(*) \rightarrow PCnt}$

$\sigma_{Job="TA"}$ $\sigma_{Job="Prof"}$

$\pi_{TCnt*PCnt \rightarrow TotCnt}$

$\times$

$\bowtie$

$\gamma_{COUNT(*) \rightarrow TCnt}$ $\gamma_{COUNT(*) \rightarrow PCnt}$

$\setminus$

$\sigma_{Job="TA"}$ $\sigma_{Job="Prof"}$

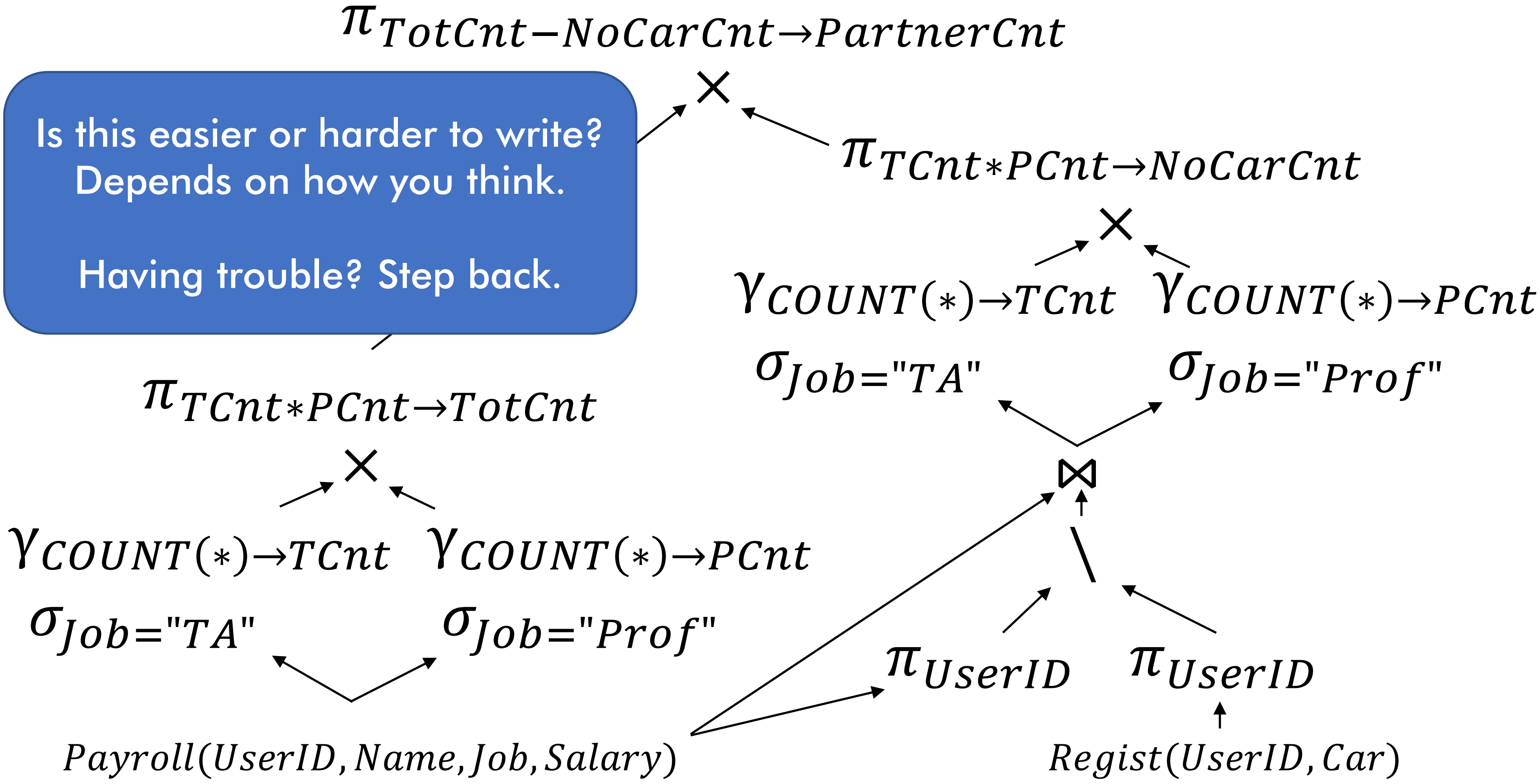
π_{UserID} π_{UserID}

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

$Regist(UserID, Car)$

All Together

Another approach: take total # of partnerships and subtract those where neither owns a car



$\pi_{TotCnt - NoCarCnt \rightarrow PartnerCnt}$
$\times$
$\pi_{TCnt * PCnt \rightarrow NoCarCnt}$
$\times$
$\gamma_{COUNT(*) \rightarrow TCnt}$
$\gamma_{COUNT(*) \rightarrow PCnt}$
$\sigma_{Job = "TA"}$
$\sigma_{Job = "Prof"}$
$\bowtie$
$\setminus$
π_{UserID}
π_{UserID}
$Regist(UserID, Car)$
$\pi_{TCnt * PCnt \rightarrow TotCnt}$
$\times$
$\gamma_{COUNT(*) \rightarrow TCnt}$
$\gamma_{COUNT(*) \rightarrow PCnt}$
$\sigma_{Job = "TA"}$
$\sigma_{Job = "Prof"}$
$Payroll(UserID, Name, Job, Salary)$

Is this easier or harder to write?
Depends on how you think.

Having trouble? Step back.

All Together

```
WITH Total AS (
  SELECT A.TCnt*B.PCnt AS TotCnt
    FROM (SELECT COUNT(*) AS TCnt FROM Payroll
           WHERE Job='TA') A,
         (SELECT COUNT(*) AS PCnt FROM Payroll
           WHERE Job='TA') B
), NoCarUsers AS (
  SELECT P.UserID, P.Job
    FROM Payroll P, (SELECT UserID FROM Payroll
                     EXCEPT
                     SELECT UserID FROM Regist) B
   WHERE P.UserID = B.UserID
), NoCar AS (
  SELECT A.TCnt*B.PCnt AS NoCarCnt
    FROM (SELECT COUNT(*) AS TCnt FROM NoCarUsers
           WHERE Job='TA') A,
         (SELECT COUNT(*) AS PCnt FROM NoCarUsers
           WHERE Job='TA') B
)
SELECT T.TotCnt-N.NoCarCnt AS PartnerCnt
  FROM Total T, NoCar N;
```

SQL & RA

- Play with your operators
- Might be easier to think in RA, then generate SQL
- RA Equivalences
  - Simplify queries
  - Make queries faster (optimization)
    - More coming later!