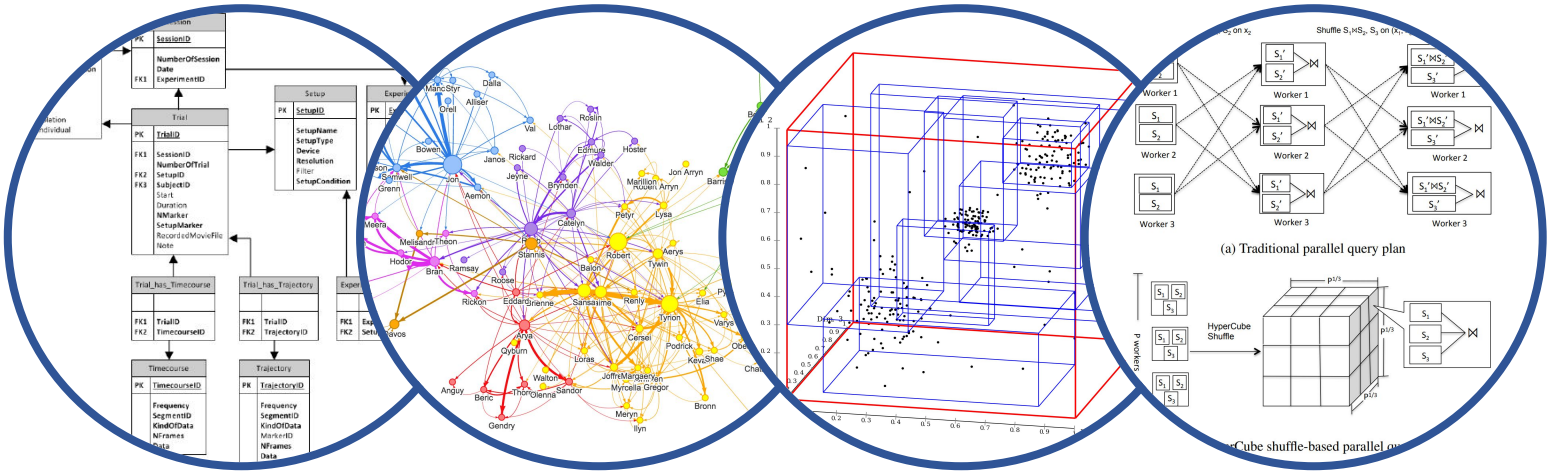


Introduction to Data Management

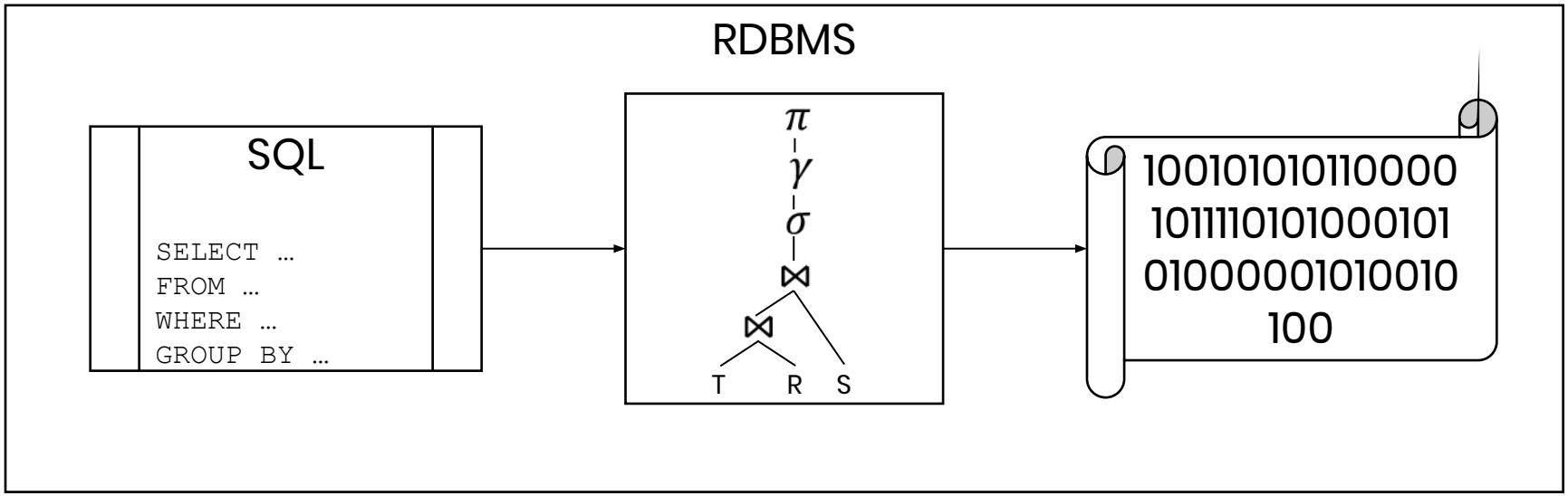
Relational Algebra

Alyssa Pittman
Based on slides by Jonathan Leang, Dan Suciu, et al

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

Recap – What's the Point of RA?

- Relational Algebra (RA) does the job
  - When processing your query, the **RDBMS will actually store an RA tree** (like a bunch of labeled nodes and pointers)
  - After some optimizations, the **RA tree is converted into instructions** (like a bunch of functions linked together)



Recap – RA Operators

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

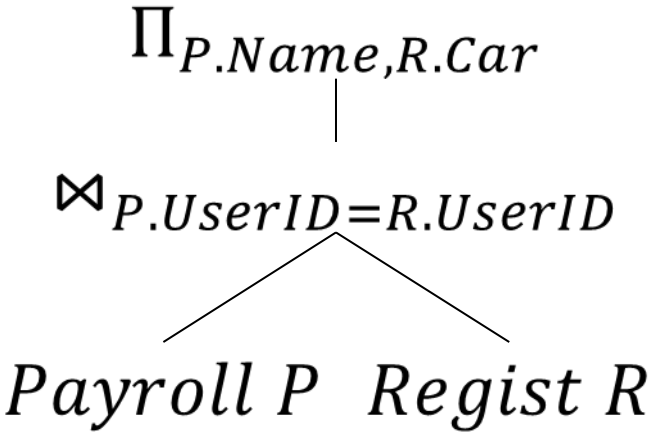
$\bowtie$	Join	$\cup$	Union	γ	Grouping & Aggregation
$\times$	Cartesian Product	$\cap$	Intersection	τ	Sort
σ	Selection	$-$	Difference	δ	Duplicate Elimination
π	Projection				

RA

Extended RA

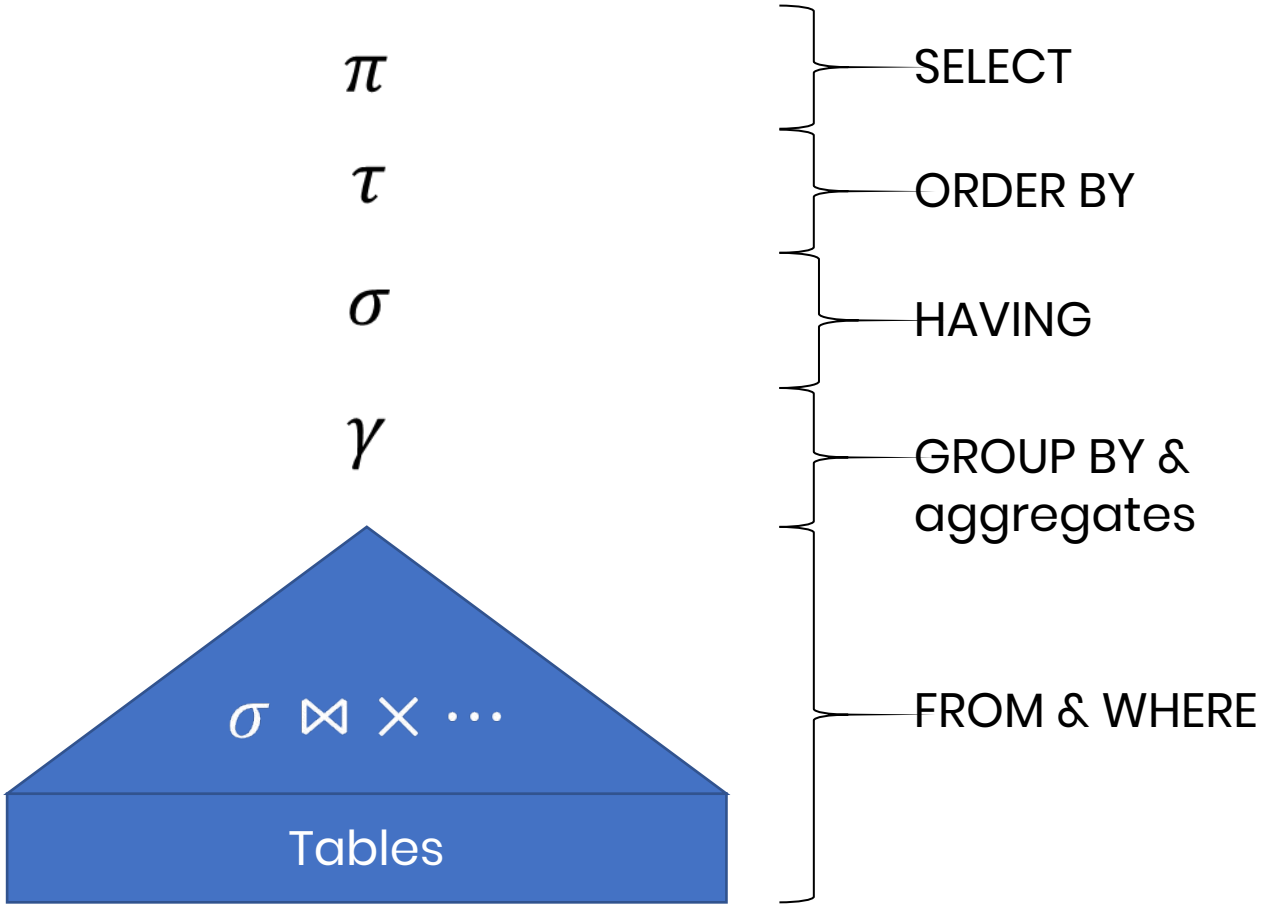
Recap - RA Equivalencies

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



Recap – Basic SQL to RA Conversion

- The general plan structure for a "flat" SQL query



Goals for Today

- We've learned RA operators and basics.
- Next we'll learn about trickier RA conversions.

Outline

- Practice SQL to RA conversion
- See how RA represents subqueries

Simple RA Example

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

$$\Pi_{Jo}$$

$$b \mid$$

$$Payroll\ P$$

Simple RA Example

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

$$\Pi_{Jo}$$

$$b \mid$$

$$Payroll\ P$$

```
SELECT Job
  FROM Payroll
```

Simple RA Example

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

$$\Pi_{Jo}$$

$$b \mid$$

$$Payroll\ P$$

```
SELECT Job
  FROM Payroll
```

Job
TA
TA
Prof
Prof

Simple RA Example

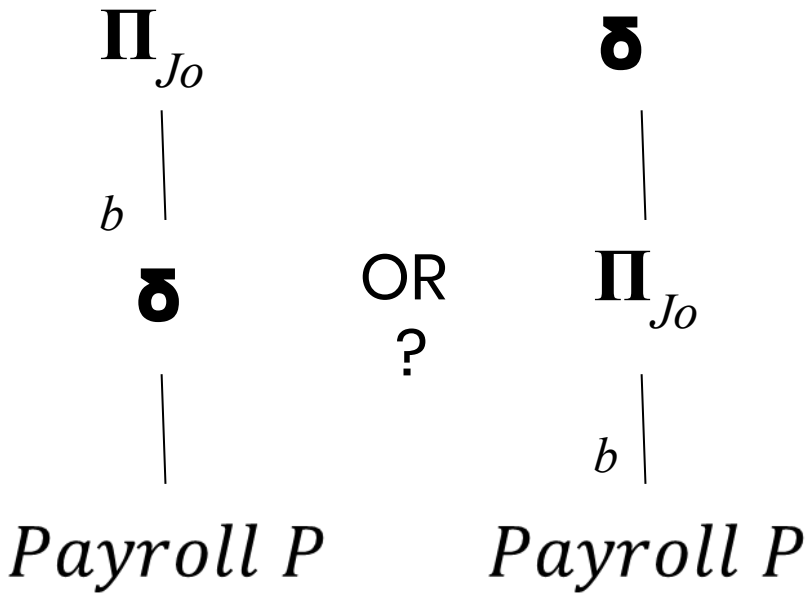
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

```
SELECT DISTINCT Job
FROM Payroll
```

Job
TA
Prof

Simple RA Example

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

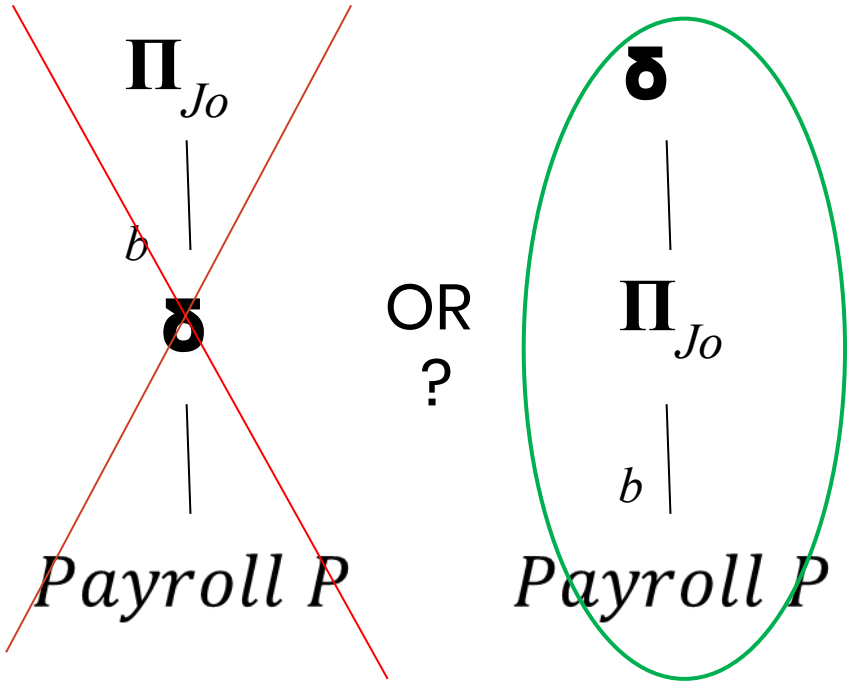



```
SELECT DISTINCT Job
  FROM Payroll
```

Job
TA
Prof

Simple RA Example

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




```
SELECT DISTINCT Job
  FROM Payroll
```

Job
TA
Prof

English to SQL to RA Example

```
CREATE TABLE Payroll (
  UserID INT PRIMARY KEY,
  Name   VARCHAR(100),
  Job    VARCHAR(100),
  Salary INT);
```

```
CREATE TABLE Regist (
  UserID INT REFERENCES Payroll,
  Car    VARCHAR(100));
```

Name all the TAs that drive multiple cars ordered by the number of cars they drive

```
SELECT DISTINCT P.Name
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID AND
       P.Job = 'TA'
 GROUP BY P.UserID, P.Name
HAVING COUNT(*) > 1
 ORDER BY COUNT(*)
```

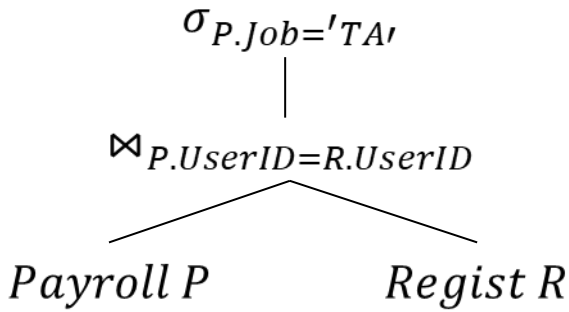
English to SQL to RA Example

```
CREATE TABLE Payroll (
  UserID INT PRIMARY KEY,
  Name   VARCHAR(100),
  Job    VARCHAR(100),
  Salary INT);
```

```
CREATE TABLE Regist (
  UserID INT REFERENCES Payroll,
  Car    VARCHAR(100));
```

Name all the TAs that drive multiple cars ordered by the number of cars they drive

```
SELECT DISTINCT P.Name
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID AND
       P.Job = 'TA'
 GROUP BY P.UserID, P.Name
HAVING COUNT(*) > 1
 ORDER BY COUNT(*)
```



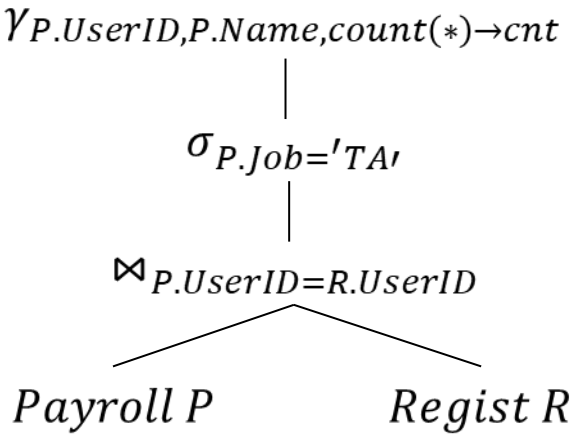
English to SQL to RA Example

```
CREATE TABLE Payroll (
  UserID INT PRIMARY KEY,
  Name   VARCHAR(100),
  Job    VARCHAR(100),
  Salary INT);
```

```
CREATE TABLE Regist (
  UserID INT REFERENCES Payroll,
  Car    VARCHAR(100));
```

Name all the TAs that drive multiple cars ordered by the number of cars they drive

```
SELECT DISTINCT P.Name
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID AND
       P.Job = 'TA'
 GROUP BY P.UserID, P.Name
HAVING COUNT(*) > 1
 ORDER BY COUNT(*)
```



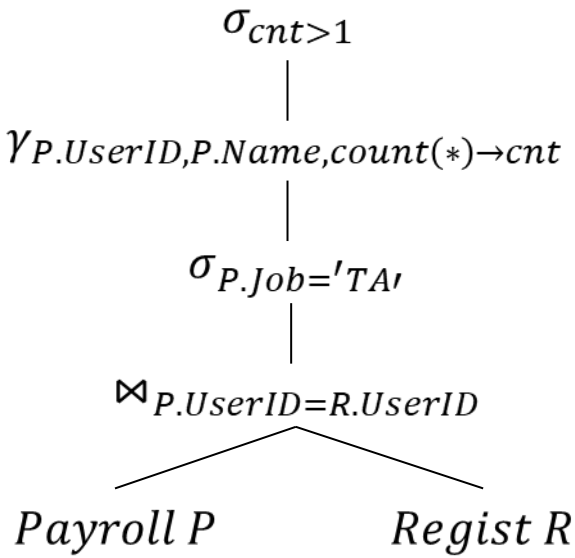
English to SQL to RA Example

```
CREATE TABLE Payroll (
  UserID INT PRIMARY KEY,
  Name   VARCHAR(100),
  Job    VARCHAR(100),
  Salary INT);
```

```
CREATE TABLE Regist (
  UserID INT REFERENCES Payroll,
  Car    VARCHAR(100));
```

Name all the TAs that drive multiple cars ordered by the number of cars they drive

```
SELECT DISTINCT P.Name
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID AND
       P.Job = 'TA'
 GROUP BY P.UserID, P.Name
HAVING COUNT(*) > 1
 ORDER BY COUNT(*)
```



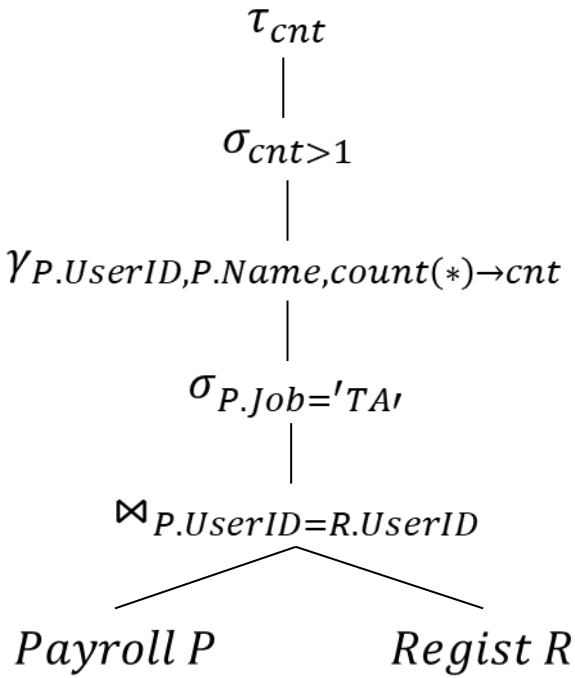
English to SQL to RA Example

```
CREATE TABLE Payroll (
  UserID INT PRIMARY KEY,
  Name   VARCHAR(100),
  Job    VARCHAR(100),
  Salary INT);
```

```
CREATE TABLE Regist (
  UserID INT REFERENCES Payroll,
  Car    VARCHAR(100));
```

Name all the TAs that drive multiple cars ordered by the number of cars they drive

```
SELECT DISTINCT P.Name
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID AND
       P.Job = 'TA'
 GROUP BY P.UserID, P.Name
HAVING COUNT(*) > 1
 ORDER BY COUNT(*)
```



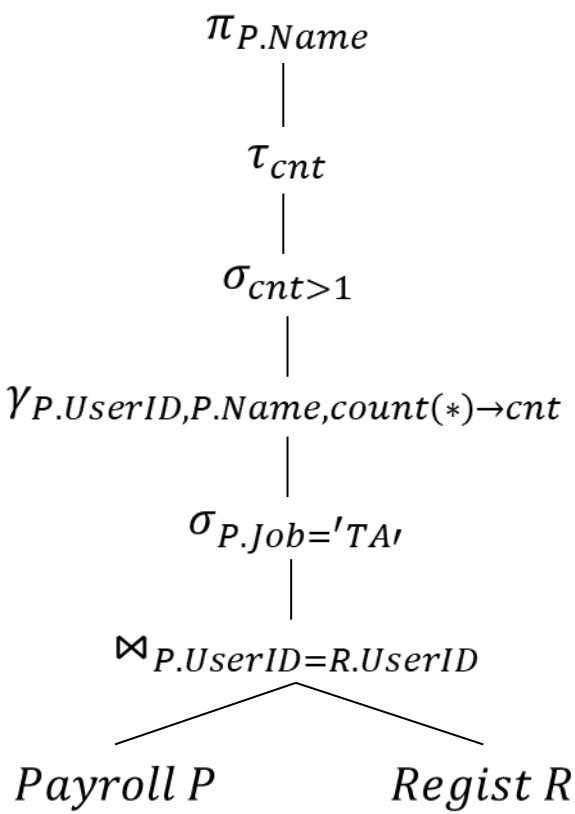
English to SQL to RA Example

```
CREATE TABLE Payroll (
  UserID INT PRIMARY KEY,
  Name   VARCHAR(100),
  Job    VARCHAR(100),
  Salary INT);
```

```
CREATE TABLE Regist (
  UserID INT REFERENCES Payroll,
  Car    VARCHAR(100));
```

Name all the TAs that drive multiple cars ordered by the number of cars they drive

```
SELECT DISTINCT P.Name
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID AND
       P.Job = 'TA'
 GROUP BY P.UserID, P.Name
HAVING COUNT(*) > 1
 ORDER BY COUNT(*)
```



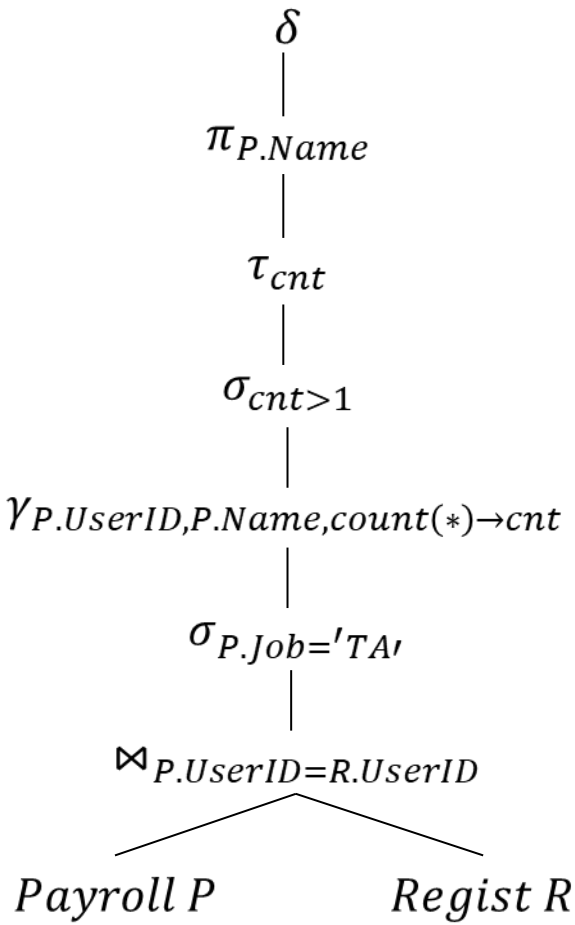
English to SQL to RA Example

```
CREATE TABLE Payroll (
  UserID INT PRIMARY KEY,
  Name   VARCHAR(100),
  Job    VARCHAR(100),
  Salary INT);
```

```
CREATE TABLE Regist (
  UserID INT REFERENCES Payroll,
  Car    VARCHAR(100));
```

Name all the TAs that drive multiple cars ordered by the number of cars they drive

```
SELECT DISTINCT P.Name
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID AND
       P.Job = 'TA'
 GROUP BY P.UserID, P.Name
HAVING COUNT(*) > 1
 ORDER BY COUNT(*)
```



Your Turn!

```
-- Adapted from 12WI Final
CREATE TABLE Person (
   pid  INT PRIMARY KEY, -- person ID
   name VARCHAR(100));   -- person name
CREATE TABLE Email (
   eid     INT PRIMARY KEY,        -- email ID
   pidFrom INT REFERENCES Person, -- email sender
   length  INT);                   -- email char length
CREATE TABLE EmailTo (
   eid   INT REFERENCES Email,   -- email ID
   pidTo INT REFERENCES Person,  -- email recipient
   PRIMARY KEY (eid, pidTo));
```

Your Turn!

- Witnessing with a self-join
- List the pid of the people who wrote the longest emails to themselves and the length of the emails.

```
SELECT E1.pidFrom, MAX(E2.length)
  FROM Email E1, EmailTo T1, Email E2, EmailTo T2
 WHERE E1.eid = T1.eid AND
       T1.pidTo = E1.pidFrom AND
       E2.eid = T2.eid AND
       T2.pidTo = E2.pidFrom
 GROUP BY E1.pidFrom, E1.length
HAVING E1.length = MAX(E2.length);
```

Your Turn!

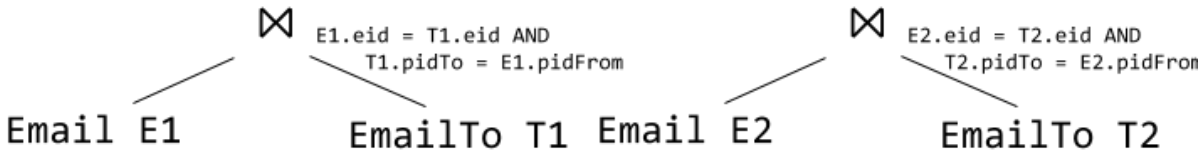
```
SELECT E1.pidFrom, MAX(E2.length)
  FROM Email E1, EmailTo T1,
       Email E2, EmailTo T2
 WHERE E1.eid = T1.eid AND
       T1.pidTo = E1.pidFrom AND
       E2.eid = T2.eid AND
       T2.pidTo = E2.pidFrom
 GROUP BY E1.pidFrom, E1.length
HAVING E1.length = MAX(E2.length);
```

Draw the RA tree for the query

Your Turn!

```
SELECT E1.pidFrom, MAX(E2.length)
  FROM Email E1, EmailTo T1,
       Email E2, EmailTo T2
 WHERE E1.eid = T1.eid AND
       T1.pidTo = E1.pidFrom AND
       E2.eid = T2.eid AND
       T2.pidTo = E2.pidFrom
 GROUP BY E1.pidFrom, E1.length
HAVING E1.length = MAX(E2.length);
```

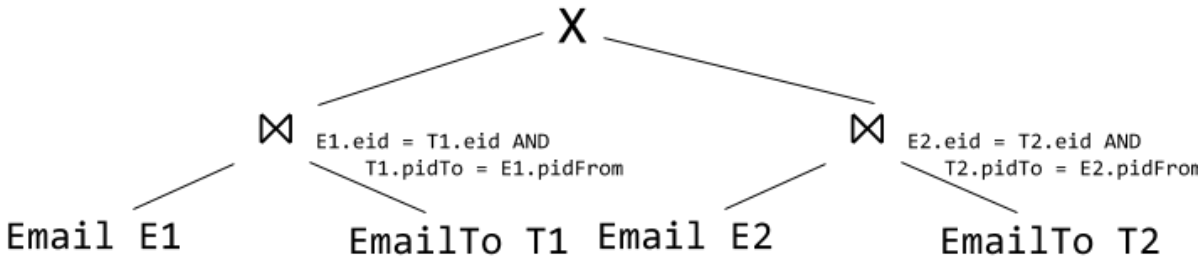
Draw the RA tree for the query



Your Turn!

```
SELECT E1.pidFrom, MAX(E2.length)
  FROM Email E1, EmailTo T1,
       Email E2, EmailTo T2
 WHERE E1.eid = T1.eid AND
       T1.pidTo = E1.pidFrom AND
       E2.eid = T2.eid AND
       T2.pidTo = E2.pidFrom
 GROUP BY E1.pidFrom, E1.length
HAVING E1.length = MAX(E2.length);
```

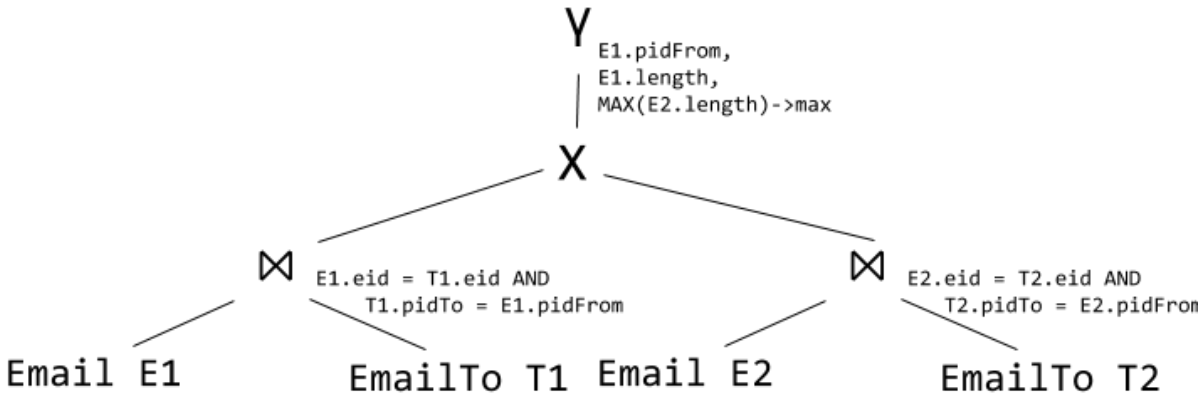
Draw the RA tree for the query



Your Turn!

```
SELECT E1.pidFrom, MAX(E2.length)
  FROM Email E1, EmailTo T1,
       Email E2, EmailTo T2
 WHERE E1.eid = T1.eid AND
       T1.pidTo = E1.pidFrom AND
       E2.eid = T2.eid AND
       T2.pidTo = E2.pidFrom
 GROUP BY E1.pidFrom, E1.length
HAVING E1.length = MAX(E2.length);
```

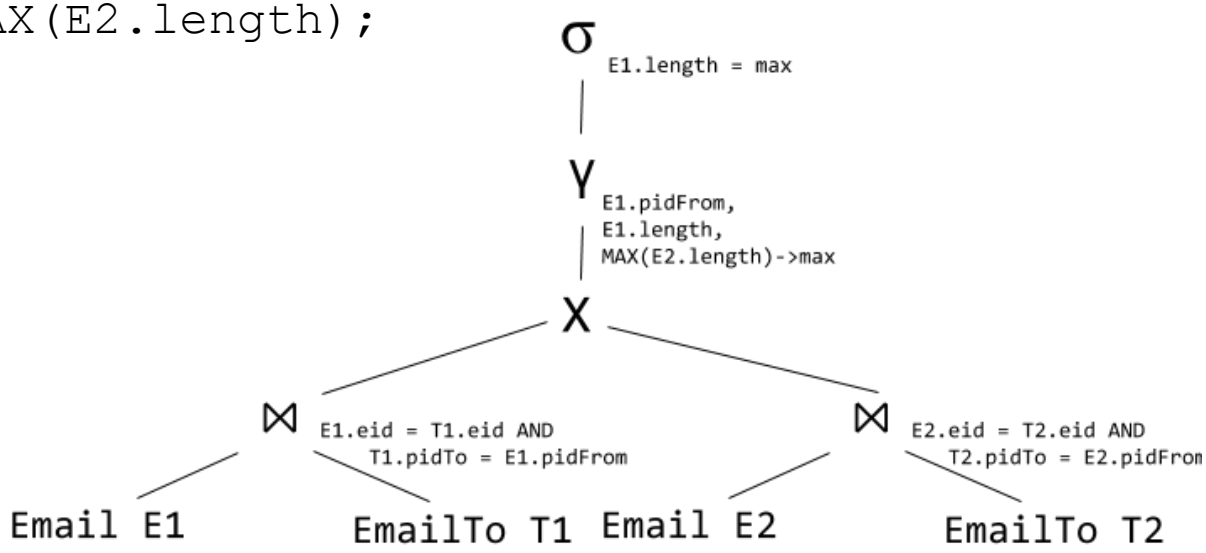
Draw the RA tree for the query



Your Turn!

```
SELECT E1.pidFrom, MAX(E2.length)
  FROM Email E1, EmailTo T1,
       Email E2, EmailTo T2
 WHERE E1.eid = T1.eid AND
       T1.pidTo = E1.pidFrom AND
       E2.eid = T2.eid AND
       T2.pidTo = E2.pidFrom
 GROUP BY E1.pidFrom, E1.length
HAVING E1.length = MAX(E2.length);
```

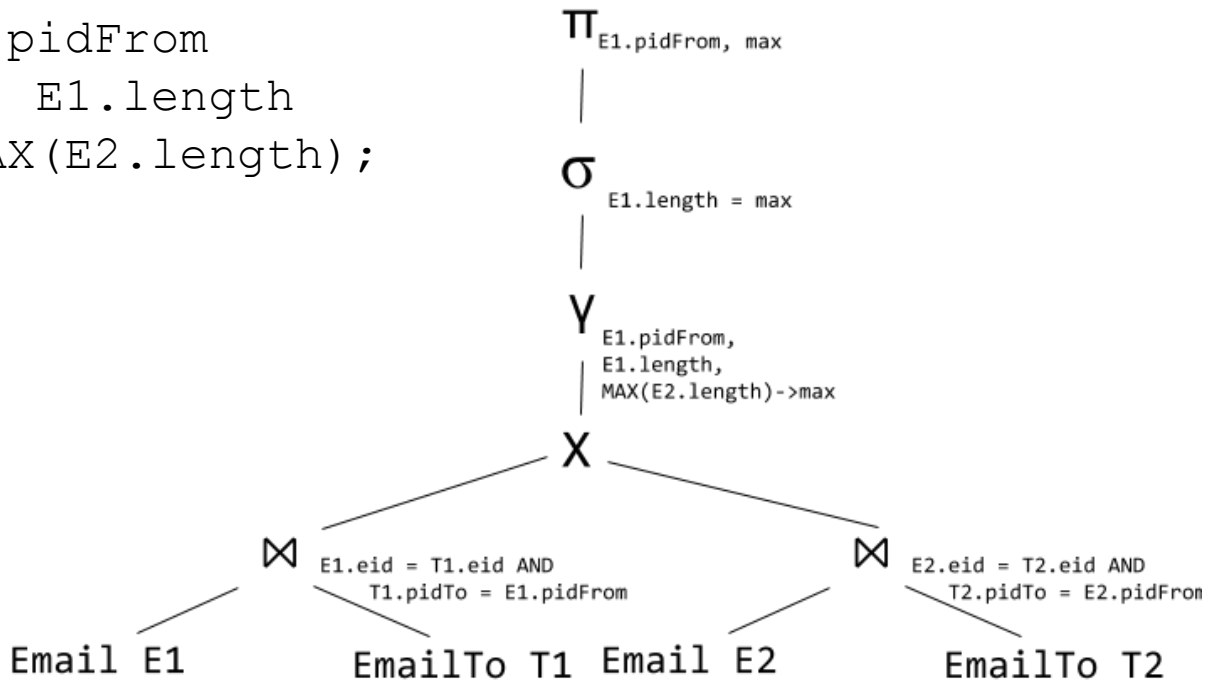
Draw the RA tree for the query



Your Turn!

```
SELECT E1.pidFrom, MAX(E2.length)
  FROM Email E1, EmailTo T1,
       Email E2, EmailTo T2
 WHERE E1.eid = T1.eid AND
       T1.pidTo = E1.pidFrom AND
       E2.eid = T2.eid AND
       T2.pidTo = E2.pidFrom
 GROUP BY E1.pidFrom, E1.length
HAVING E1.length = MAX(E2.length);
```

Draw the RA tree for the query



Bonus: SQL 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

Bonus: SQL Set Operations

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

Onto Set RA

Select $\boldsymbol{\sigma}$, Project $\boldsymbol{\pi}$, Join $\bowtie$ are the most common operators

We also have set operators

RA Operators

$\cup$ Union $\cap$ Intersection

- Binary operators
- Same semantics as in set theory (but over bags)
- Used in SQL "UNION" and "INTERSECTION"
  - Also useful when rewriting SQL to RA

$$T(A,B) \cup S(A,B) \rightarrow R(A,B)$$

A	B
1	2
3	4

A	B
1	2
5	6

A	B
1	2
3	4
1	2
5	6

RA Operators

— Difference

- Binary operator (but direction matters)
- Reads as (left input) – (right input)
- Used in SQL "DIFFERENCE"
  - Also useful when rewriting SQL to RA

$$T(A,B) - S(A,B) \rightarrow R(A,B)$$

A	B
1	2
3	4

A	B
1	2
5	6

A	B
3	4

RA Operators

— Difference

- Binary operator (but direction matters)
- Reads as (left input) – (right input)
- Used in SQL "DIFFERENCE"
  - Also useful when rewriting SQL to RA

$$T(A,B) - S(A,B) \rightarrow R(A,B)$$

A	B
1	2
1	2
3	4

A	B
1	2
5	6

A	B
?	?

RA Operators

— Difference

- Binary operator (but direction matters)
- Reads as (left input) – (right input)
- Used in SQL "DIFFERENCE"
  - Also useful when rewriting SQL to RA

$$T(A,B) - S(A,B) \rightarrow R(A,B)$$

A	B
1	2
1	2
3	4

A	B
1	2
5	6

A	B
1	2
3	4

Bag semantics!

Some Simplifications

- The book discusses a variety of joins that sometimes remove redundant attributes – we'll stick with theta joins.
  - Always specify the join condition
  - All attributes from both tables will be in the output relation

$$T(A,B) \bowtie_{T.B=S.C} S(C,D) \rightarrow R(A,B,C,D)$$

A	B
1	2
3	4
5	6

C	D
2	3
5	6
6	7

A	B	C	D
1	2	2	3
5	6	6	7

- We won't look at outer joins in RA

RA Operators

ρ Rename

- Unary operator
- Operates on the schema, not the instance
- Renames the attributes
  - Useful to ensure relations used in set operations have the same schema

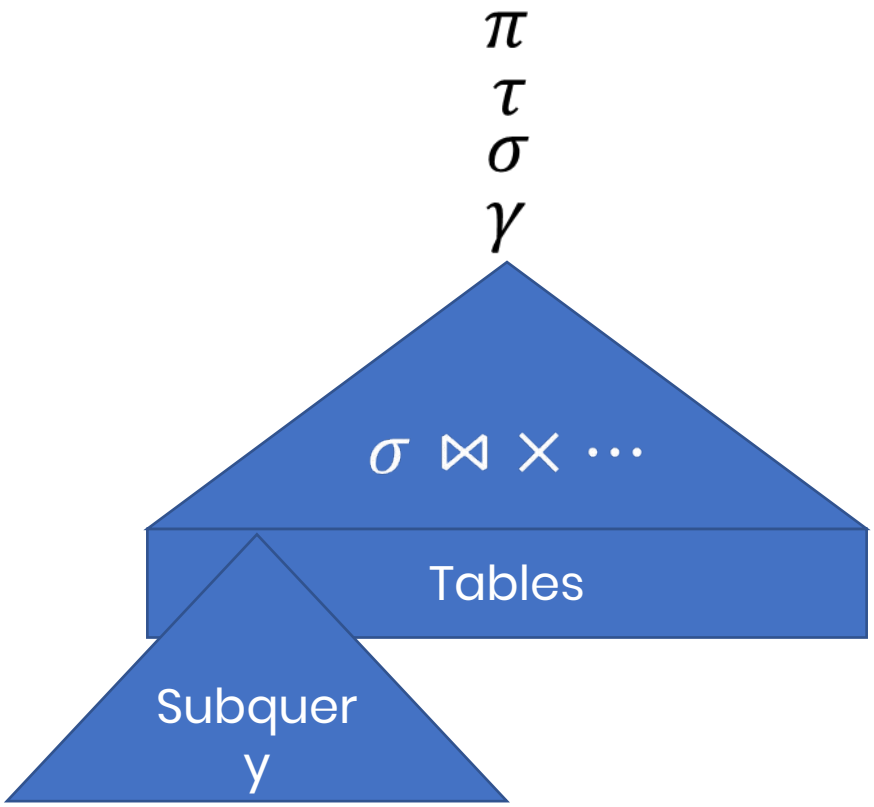
$$\rho_{C,D}(T(A,B)) \rightarrow T(C,D)$$

A	B
1	2
3	4
5	6

C	D
1	2
3	4
5	6

What about subqueries?

- FROM/WITH subquery is pretty mechanical too
- Connect the subquery tree like it was a real table



Decorrelation and Unnesting

- The hardest type of SQL-to-RA conversions are ones that involves correlated WHERE subqueries
- The precise algorithms for arbitrary SQL-to-RA conversion are beyond the scope of this class
  - A nice document for the curious
  - A cool research paper by Thomas Neumann and Alfons Kemper (2015) for the masochistic

Decorrelation and Unnesting

- **Correlation** □ A table in the parent query is used in the subquery

```
CREATE TABLE Supplier (
  sid   INT PRIMARY KEY
  state VARCHAR(100));
```

```
CREATE TABLE Inventory (
  sid    INT
  partNo INT
  price  INT
  PRIMARY KEY (sid, partNo));
```

```
SELECT S.sid
  FROM Supplier S
 WHERE S.state = 'WA' AND
       NOT EXISTS (SELECT *
                     FROM Inventory I
                    WHERE I.sid = S.sid AND
                          I.price > 100);
```

Decorrelation and Unnesting

Correlated

```
SELECT S.sid
  FROM Supplier S
 WHERE S.state = 'WA' AND
       NOT EXISTS (SELECT *
                     FROM Inventory I
                    WHERE I.sid = S.sid AND
                          I.price > 100);
```

Decorrelated

```
SELECT S.sid
  FROM Supplier S
 WHERE S.state = 'WA' AND
       S.sid NOT IN (SELECT I.sid
                       FROM Inventory I
                      WHERE I.price > 100);
```

Decorrelation and Unnesting

Nested

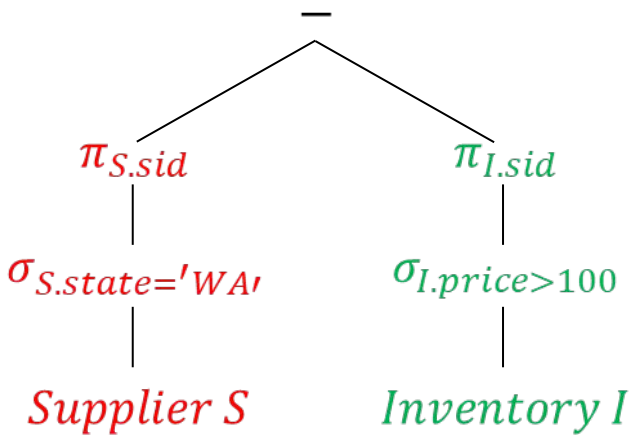
```
SELECT S.sid
  FROM Supplier S
 WHERE S.state = 'WA' AND
       S.sid NOT IN (SELECT I.sid
                       FROM Inventory I
                      WHERE I.price > 100);
```

Unnested

```
(SELECT S.sid
   FROM Supplier S
  WHERE S.state = 'WA')
EXCEPT
(SELECT I.sid
   FROM Inventory I
  WHERE I.price > 100)
```

Decorrelation and Unnesting

```
(SELECT S.sid
   FROM Supplier S
  WHERE S.state = 'WA')
EXCEPT
(SELECT I.sid
   FROM Inventory I
  WHERE I.price > 100)
```



Your Turn! (again)

- Find all emails where all of the recipients are named Alice.
- We can start from a correlated subquery

```
SELECT E1.eid
  FROM Email E1
 WHERE NOT EXISTS (SELECT *
                     FROM EmailTo E2, Person P
                    WHERE E1.eid = E2.eid AND
                          E2.pidTo = P.pid AND
                          P.name != 'Alice');
```

Your Turn! (again)

```
SELECT E1.eid
  FROM Email E1
 WHERE NOT EXISTS (SELECT *
                     FROM EmailTo E2, Person P
                    WHERE E1.eid = E2.eid AND
                          E2.pidTo = P.pid AND
                          P.name != 'Alice');
```

Write the uncorrelated version of the query

```
SELECT E1.eid
  FROM Email E1
 WHERE E1.eid NOT IN (SELECT E2.eid
                      FROM EmailTo E2, Person P
                      WHERE E2.pidTo = P.pid AND
                            P.name != 'Alice');
```

Your Turn! (again)

```
SELECT E1.eid
  FROM Email E1
 WHERE NOT EXISTS (SELECT *
                     FROM EmailTo E2, Person P
                    WHERE E1.eid = E2.eid AND
                          E2.pidTo = P.pid AND
                          P.name != 'Alice');
```

Write the uncorrelated version of the que

```
SELECT E1
  FROM E
 WHERE E1.e
                                    P
                  .p    P.pi    D
           name !   Alice');
```

Same as:
(SELECT E1.eid FROM Email E1)
–
(subquery)

Your Turn! (again)

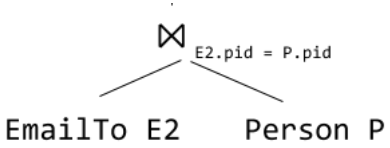
```
SELECT E1.eid
  FROM Email E1
 WHERE E1.eid NOT IN (SELECT E2.eid
                        FROM EmailTo E2, Person P
                       WHERE E2.pidTo = P.pid AND
                             P.name != 'Alice');
```

Draw the RA tree for the query

Your Turn! (again)

```
SELECT E1.eid
  FROM Email E1
 WHERE E1.eid NOT IN (SELECT E2.eid
                        FROM EmailTo E2, Person P
                       WHERE E2.pidTo = P.pid AND
                             P.name != 'Alice');
```

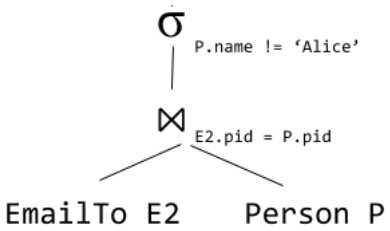
Draw the RA tree for the query



Your Turn! (again)

```
SELECT E1.eid
  FROM Email E1
 WHERE E1.eid NOT IN (SELECT E2.eid
                        FROM EmailTo E2, Person P
                       WHERE E2.pidTo = P.pid AND
                             P.name != 'Alice');
```

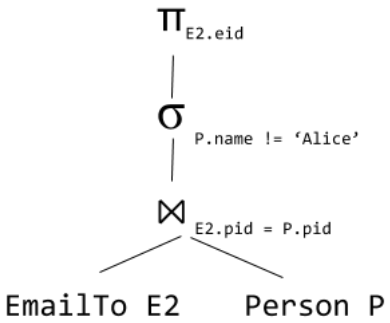
Draw the RA tree for the query



Your Turn! (again)

```
SELECT E1.eid
  FROM Email E1
 WHERE E1.eid NOT IN (SELECT E2.eid
                        FROM EmailTo E2, Person P
                       WHERE E2.pidTo = P.pid AND
                             P.name != 'Alice');
```

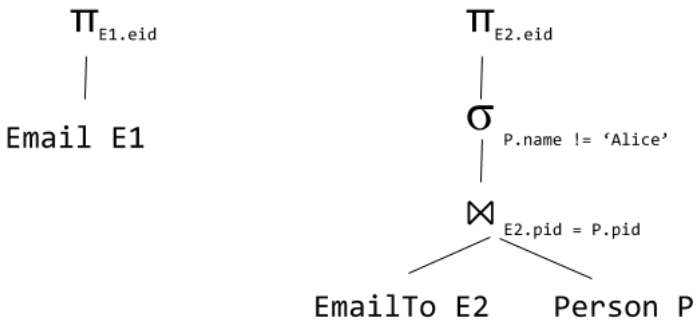
Draw the RA tree for the query



Your Turn! (again)

```
SELECT E1.eid
  FROM Email E1
 WHERE E1.eid NOT IN (SELECT E2.eid
                        FROM EmailTo E2, Person P
                       WHERE E2.pidTo = P.pid AND
                             P.name != 'Alice');
```

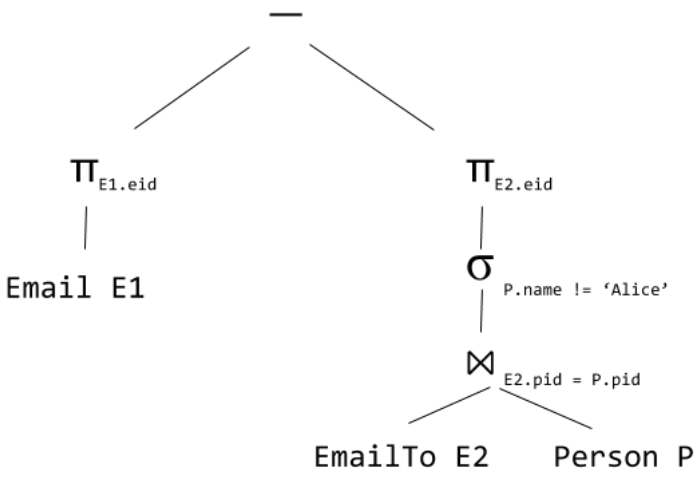
Draw the RA tree for the query



Your Turn! (again)

```
SELECT E1.eid
  FROM Email E1
 WHERE E1.eid NOT IN (SELECT E2.eid
                        FROM EmailTo E2, Person P
                       WHERE E2.pidTo = P.pid AND
                             P.name != 'Alice');
```

Draw the RA tree for the query



Takeaways

- SQL to RA conversions aren't always straightforward
  - Decorrelating, unnesting, and using set operations can help

Outlook

- This isn't the end of RA!
- We will need RA again when we talk about database tuning