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

- Join
- Cartesian Product
- Selection
- Projection
- Union
- Intersection
- Difference
- Grouping & Aggregation
- Sort
- Duplicate Elimination

RA

Extended RA
Recap – RA Equivalencies

\[
\begin{align*}
\text{SELECT} & \quad \text{P.Name, R.Car} \\
\text{FROM} & \quad \text{Payroll AS P, Regist AS R} \\
\text{WHERE} & \quad \text{P.UserID = R.UserID};
\end{align*}
\]
Recap – Basic SQL to RA Conversion

- The general plan structure for a “flat” SQL query

```plaintext
π  τ  σ  γ

SELECT
ORDER BY
HAVING
GROUP BY & aggregates
FROM & WHERE
```

Tables
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

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

\[
\Pi_{\text{Job}}(b \mid \text{Payroll } P)
\]
**Simple RA Example**

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

\[
\Pi_{Job}^{b} (Payroll P) = \text{SELECT} \text{Job} \text{FROM Payroll}
\]
Simple RA Example

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

\[
\Pi_{Job} \left( \sigma_{b} Payroll P \right)
\]

\[
\text{SELECT} \quad \text{Job} \quad \text{FROM} \quad \text{Payroll}
\]

Job
- TA
- TA
- Prof
- Prof
### Simple RA Example

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

**SELECT** DISTINCT Job **FROM** Payroll

<table>
<thead>
<tr>
<th>Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA</td>
</tr>
<tr>
<td>Prof</td>
</tr>
</tbody>
</table>
### Simple RA Example

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

#### SQL Query

```
SELECT DISTINCT Job
FROM Payroll
```

#### Relational Algebra

```
\[ \Pi_{Job} (\delta \theta (\approx \bowtie \text{Payroll P})) OR (\Pi_{Job} (\delta \theta (\approx \bowtie \text{Payroll P}))) \]
```

Where:
- \( \Pi_{Job} \) is the projection operation on the Job attribute.
- \( \delta \theta \) is the selection operation based on some condition \( \theta \).
- \( \bowtie \) is the join operation on Payroll P.
- \( \approx \) is the equality condition.

**Job**
- TA
- Prof
Simple RA Example

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

```
SELECT DISTINCT Job
FROM Payroll
```
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(*)
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(*)

\[
\forall_{P.UserID,P.Name,count(*)\rightarrow cnt}  \\
\sigma_{P.Job='TA'}  \\
\exists_{P.UserID=R.UserID}
\]

\[
\text{Payroll } P \quad \text{Regist } R
\]
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(*)
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(*)
```

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));
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(*)
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(*)
```
-- 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.

```sql
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!

```sql
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

```
Y

E1.pidFrom,
E1.length,
MAX(E2.length) -> max

X

∧
E1.eid = T1.eid AND
T1.pidTo = E1.pidFrom

∧
E2.eid = T2.eid AND
T2.pidTo = E2.pidFrom

∧
Email E1

∧
EmailTo T1

∧
Email E2

∧
EmailTo T2
```
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:

```
\[ \pi_{E1.pidFrom, \text{max}} \]
\[ \sigma_{E1.length = \text{max}} \]
\[ \gamma_{E1.pidFrom, E1.length, \text{MAX}(E2.length) \rightarrow \text{max}} \]
\[ \chi \]
\[ \nabla_{E1.eid = T1.eid \text{ AND } T1.pidTo = E1.pidFrom} \]
\[ \nabla_{E2.eid = T2.eid \text{ AND } T2.pidTo = E2.pidFrom} \]
\[ \text{Email E1} \quad \text{EmailTo T1} \quad \text{Email E2} \quad \text{EmailTo T2} \]
```
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
SQL set-like operators basically slap two queries together (not really a subquery...)

\[
\text{(SELECT * FROM T1) UNION (SELECT * FROM T2)}
\]

**IS THIS A SUBQUERY?**
Onto Set RA

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

We also have set operators
RA Operators

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

\[ T(A, B) \cup S(A, B) \rightarrow R(A, B) \]
RA Operators

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

\[ T(A, B) − S(A, B) \rightarrow R(A, B) \]
RA Operators

- Binary operator (but direction matters)
- Reads as \((\text{left input}) - (\text{right input})\)
- Used in SQL “DIFFERENCE”
  - Also useful when rewriting SQL to RA

\[
T(A, B) - S(A, B) \rightarrow R(A, B)
\]
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

\[ T(A, B) - S(A, B) \rightarrow R(A, B) \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

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

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

- We won’t look at outer joins in RA
RA Operators

\[ \rho \]

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

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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
What about subqueries?

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

![Diagram showing the connection of subqueries within a tree structure with operations like π, τ, σ, and γ.]
Decorrelation and Unnesting

- The hardest type of SQL-to-RA conversions are ones that involve 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
**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  
PRIMAR 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

\[
\begin{align*}
\text{SELECT} & \quad S.\text{sid} \\
\text{FROM} & \quad \text{Supplier S} \\
\text{WHERE} & \quad S.\text{state} = \text{'WA'} \quad \text{AND} \\
& \quad \text{NOT EXISTS} \quad (\text{SELECT} \quad * \\
& \quad \text{FROM} \quad \text{Inventory I} \\
& \quad \text{WHERE} \quad I.\text{sid} = S.\text{sid} \quad \text{AND} \\
& \quad I.\text{price} > 100); \\
\end{align*}
\]

Decorrelated

\[
\begin{align*}
\text{SELECT} & \quad S.\text{sid} \\
\text{FROM} & \quad \text{Supplier S} \\
\text{WHERE} & \quad S.\text{state} = \text{'WA'} \quad \text{AND} \\
& \quad S.\text{sid} \quad \text{NOT IN} \quad (\text{SELECT} \quad I.\text{sid} \\
& \quad \text{FROM} \quad \text{Inventory I} \\
& \quad \text{WHERE} \quad I.\text{price} > 100); \\
\end{align*}
\]

Correlated

\[
\begin{align*}
\text{SELECT} & \quad S.\text{sid} \\
\text{FROM} & \quad \text{Supplier S} \\
\text{WHERE} & \quad S.\text{state} = \text{'WA'} \quad \text{AND} \\
& \quad \text{NOT EXISTS} \quad (\text{SELECT} \quad * \\
& \quad \text{FROM} \quad \text{Inventory I} \\
& \quad \text{WHERE} \quad I.\text{sid} = S.\text{sid} \quad \text{AND} \\
& \quad I.\text{price} > 100); \\
\end{align*}
\]

Decorrelated

\[
\begin{align*}
\text{SELECT} & \quad S.\text{sid} \\
\text{FROM} & \quad \text{Supplier S} \\
\text{WHERE} & \quad S.\text{state} = \text{'WA'} \quad \text{AND} \\
& \quad S.\text{sid} \quad \text{NOT IN} \quad (\text{SELECT} \quad I.\text{sid} \\
& \quad \text{FROM} \quad \text{Inventory I} \\
& \quad \text{WHERE} \quad I.\text{price} > 100); \\
\end{align*}
\]
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);
```

Unnesting

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

```sql
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 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');
```

Same as:

```
(SELECT E1.eid FROM Email E1) -
(SELECT E2.eid FROM EmailTo E2, Person P
WHERE E2.pidTo = P.pid
AND P.name != 'Alice');
```
Your Turn! (again)

```sql
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)

```sql
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)

\[
\text{SELECT E1.eid}
\text{FROM Email E1}
\text{WHERE E1.eid NOT IN (SELECT E2.eid}
\text{FROM EmailTo E2, Person P}
\text{WHERE E2.pidTo = P.pid AND P.name \neq 'Alice');}
\]

Draw the RA tree for the query
Your Turn! (again)

Your task is to translate the following relational algebra query into a relational algebra tree,

\[
\begin{align*}
\text{SELECT} & \quad \text{E1.eid} \\
\text{FROM} & \quad \text{Email E1} \\
\text{WHERE} & \quad \text{E1.eid NOT IN (SELECT E2.eid} \\
& \quad \quad \text{FROM EmailTo E2, Person P} \\
& \quad \quad \quad \text{WHERE E2.pidTo = P.pid AND} \\
& \quad \quad \quad \quad \text{P.name != 'Alice');}
\end{align*}
\]

Draw the RA tree for the query
Takeaways

▪ SQL to RA conversions aren’t always straightforward
  • Decorrelating, unnesting, and using set operations can help
This isn’t the end of RA!
We will need RA again when we talk about database tuning