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)

![Diagram showing the process from SQL to RA tree to RDBMS output]
Recap - RA Operators

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

- RA
- Extended RA

- $\times$ Join
- $\times$ Cartesian Product
- $\sigma$ Selection
- $\pi$ Projection
- $\cup$ Union
- $\cap$ Intersection
- $-$ Difference
- $\gamma$ Grouping & Aggregation
- $\tau$ Sort
- $\delta$ Duplicate Elimination
Recap - RA Equivalencies

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

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

\[
\pi \quad \sigma \quad \tau \quad \gamma \\
\sigma \bowtie \times \cdots
\]

- Tables

- SELECT
- ORDER BY
- HAVING
- GROUP BY & aggregates
- FROM & WHERE
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>

\[
\Pi_{\text{Job}}(b \mid Payroll \ P)
\]
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>

\[
\Pi_{Job} (b \mid Payroll P)
\]

\[
\text{SELECT Job FROM Payroll}
\]
## 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>

\[
\Pi_{\text{Job}} (b \mid \text{Payroll } P)
\]

SELECT Job
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>

```
SELECT DISTINCT Job
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>

```
SELECT DISTINCT Job
FROM Payroll
```

```
\[ \Pi_{\text{Job}} \delta \beta \delta \beta \Pi_{\text{Job}} \]
```

```
\[ \beta \delta \beta \delta \beta \Pi_{\text{Job}} \]
```

```
\[ \text{Payroll } P \]
```

```
\[ \text{Payroll } P \]
```

```
Job
```

```
<table>
<thead>
<tr>
<th>TA</th>
</tr>
</thead>
<tbody>
<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>

**SELECT** $\text{DISTINCT \ Job}$

**FROM** Payroll

\[
\begin{align*}
\Pi_{\text{Job}} & \left( \delta_{b} \left( \Pi_{\text{Job}} \left( \text{Payroll} \right) \right) \right)
\end{align*}
\]
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

$\rho$  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

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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
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(*)
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(*)
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

$$\pi_{P.Name}$$

$$\tau_{cnt}$$

$$\sigma_{cnt>1}$$

$$\gamma_{P.UserID,P.Name,count(*)\rightarrow cnt}$$

$$\sigma_{P.Job='TA'}$$

$$\Delta_{P.UserID=R.UserID}$$

$$P$$ \rightarrow Payroll

$$R$$ \rightarrow Regist
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 Person (
    pid INT PRIMARY KEY, -- person ID
    name VARCHAR(100)));

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 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!

```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
What about subqueries?

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

- Witnessing with a subquery
- List the pid of people who write emails to themselves only shorter than 1000 characters and the number of emails they have sent to themselves

WITH SelfEmailMaxLength (  
  SELECT MAX(E.length) AS maxlength  
  FROM Email E, EmailTo T  
  WHERE E.eid = T.eid AND T.pidTo = E.pidFrom  
)  
SELECT E.pidFrom, SEML.maxlength  
FROM Email E, EmailTo T, SelfEmailMaxLength SEML  
WHERE E.eid = T.eid AND  
  T.pidTo = E.pidFrom AND  
  E.length = SEML.maxlength;
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
**Correlation** □ A table in the parent query is used in the subquery

```sql
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);
```

**Unnestled**

```
(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

```
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)

```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');
```

Write the uncorrelated version of 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
- This isn’t the end of RA!
- We will need RA again when we talk about database tuning