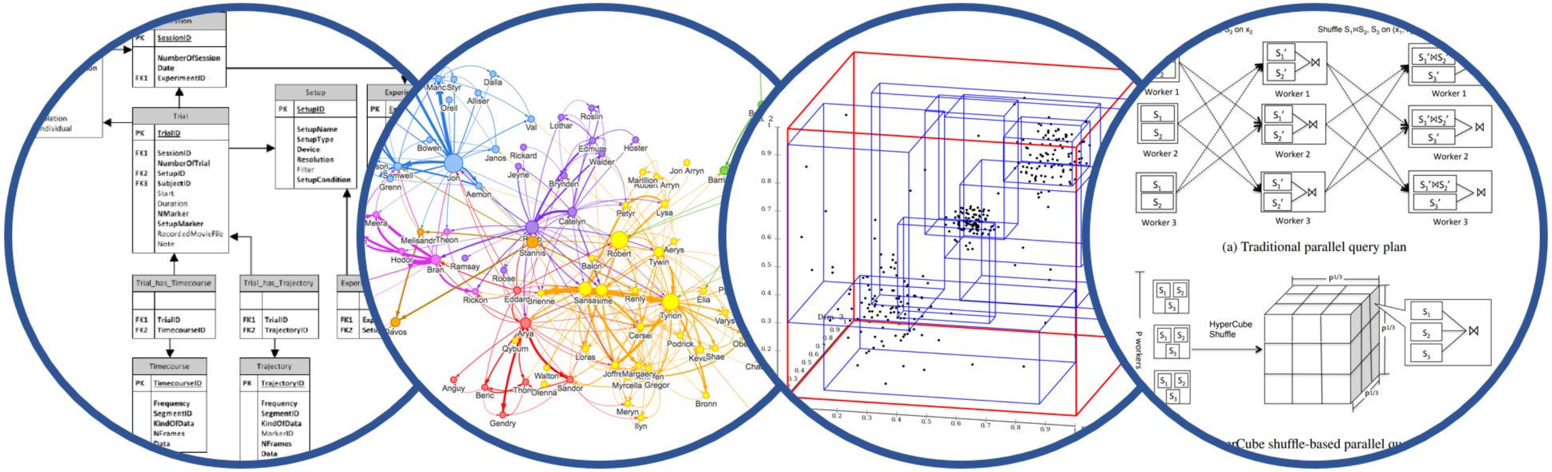


Introduction to Data Management

Relational Algebra

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

Announcements

- HW 2 extended to Thursday

- HW 3 out Thursday morning – go to section to learn how to set up
  - Using Microsoft Azure – cloud service

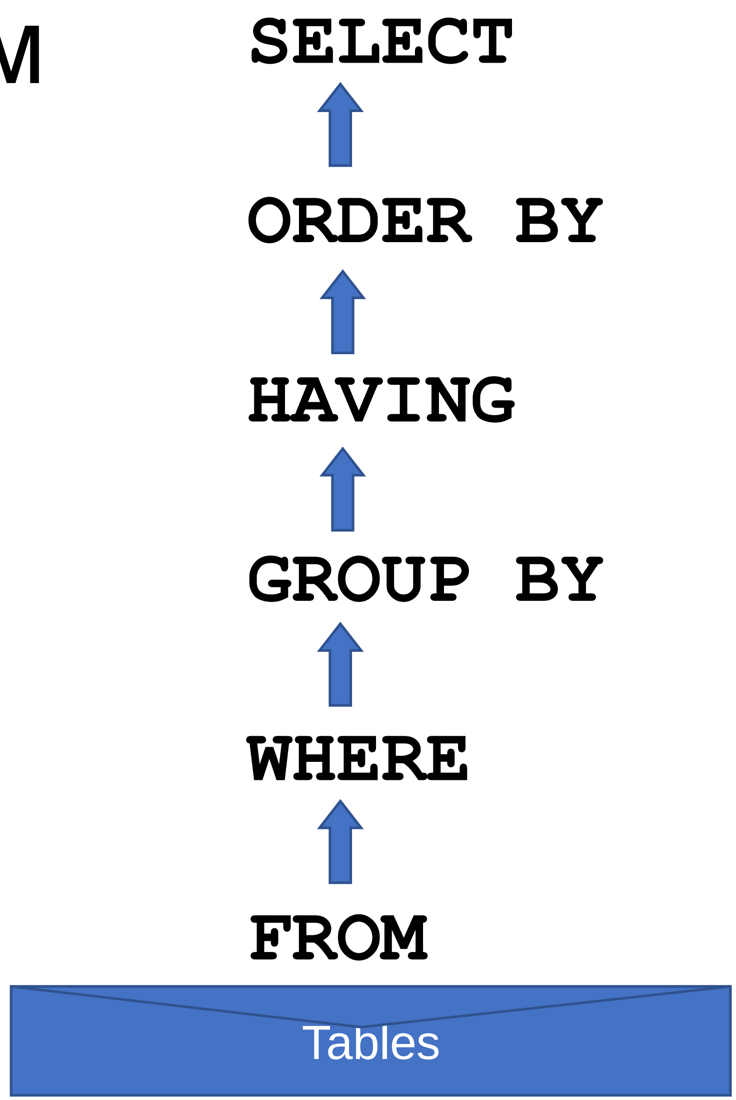
Outline

- Introduce relational algebra
- Look at some example RA from previous lectures
- Translating SQL ⟷ RA

Starting point

FWGHOS™

```
SELECT ...
  FROM ...
 WHERE ...
 GROUP BY ...
HAVING ...
 ORDER BY ...
```

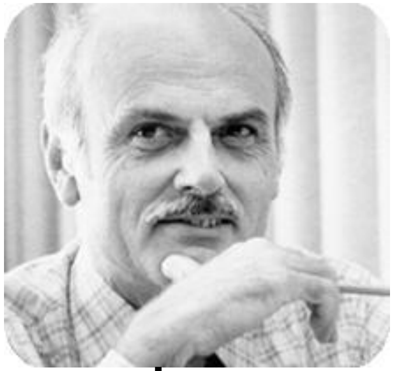


What's the Point of RA?

- SQL is a **Declarative Language**
  - "What to get" rather than "how to get it"
  - Easier to write a SQL query than write a whole Java program that will probably perform worse
- But computers are imperative/procedural
  - Computers only understand the "how"

History of RA

- Invented/Formalized by Ted Codd while working for IBM



- He realized we need a way to describe imperative programming on tables **without knowing physical details**
- IBM initially ignored his techniques

Information Retrieval

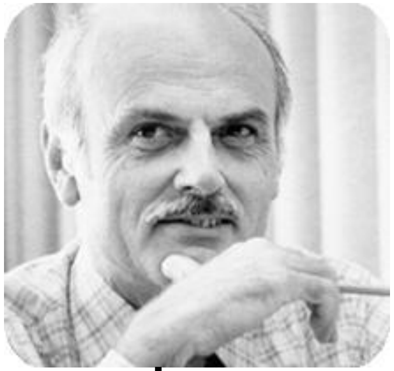
A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

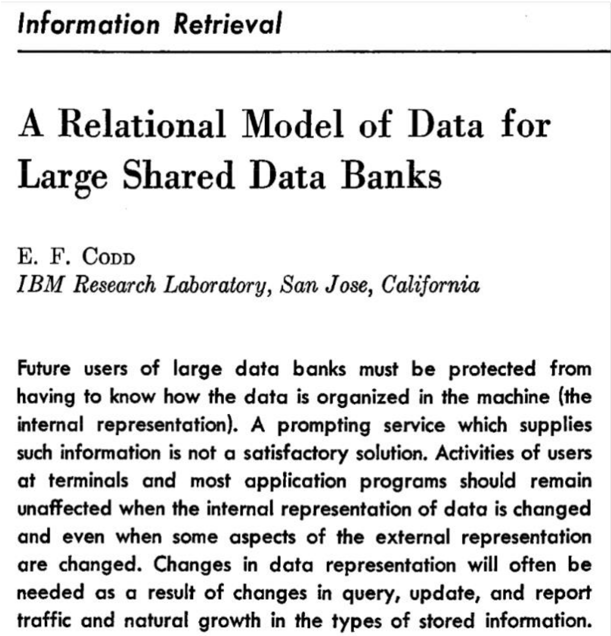
History of RA

- Invented/Formalized by Ted Codd while working for IBM



- He realized we need a way to describe imperative programming on tables **without knowing physical details**

- IBM initially ignored his techniques



Information Retrieval

A Relational Model of Data for Large Shared Data Banks

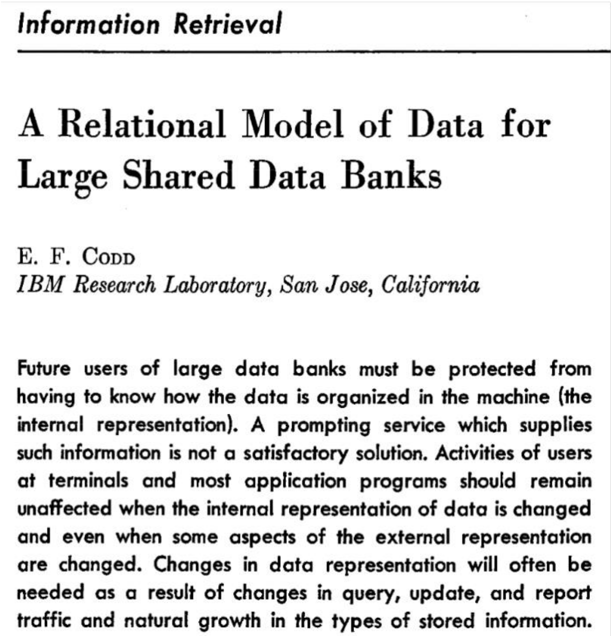
E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

- 10 years later he won the Turing Award

History of RA

- Invented/Formalized by Ted Codd while working for IBM
- He realized we need a way to describe imperative programming on tables **without knowing physical details**
- IBM initially ignored his techniques
- 10 years later he won the Turing Award



Information Retrieval

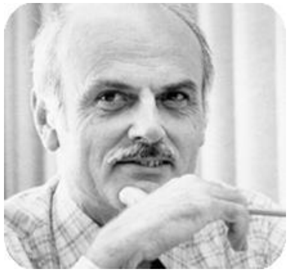
A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Turing Awards in Data Management

Charles Bachman, 1973
IDS and CODASYL



Ted Codd, 1981
Relational model

Jim Gray, 1998
Transaction processing

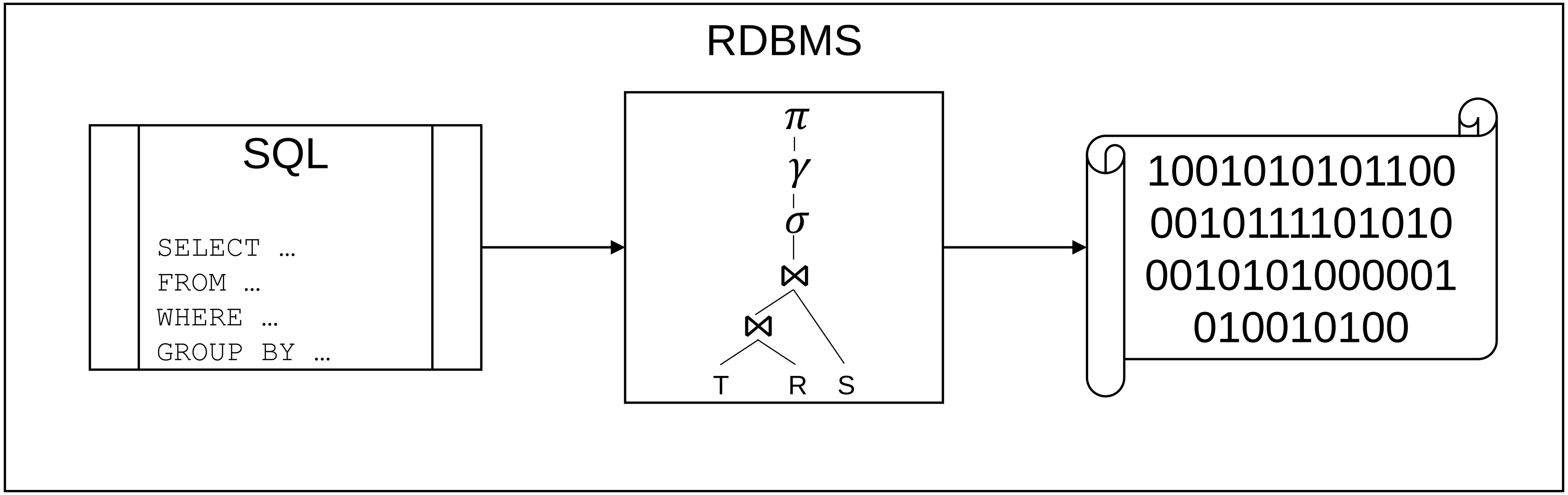
Michael Stonebraker, 2014
INGRES and Postgres

What’s the Point of RA?

- We need a language that reads more like **instructions** but still captures the fundamental operations of a query

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)



Flashback to our first query

- Code has to boil down to instructions at some point
- Relational Database Management Systems (RDBMSs) use **Relational Algebra** (RA)

```
SELECT P.Name, P.UserID
  FROM Payroll AS P
 WHERE P.Job = 'TA';
```

Flashback to our first query

- Code has to boil down to instructions at some point
- Relational Database Management Systems (RDBMSs) use **Relational Algebra** (RA)

Parser

```
SELECT P.Name, P.UserID
  FROM Payroll AS P
 WHERE P.Job = 'TA';
```

$$\pi_{P.Name, P.UserID}$$

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

$$Payroll\ P$$

Flashback to our first query

- Code has to boil down to instructions at some point
- Relational Database Management Systems (RDBMSs) use **Relational Algebra** (RA).

$$\pi_{P.Name, P.UserID}$$

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

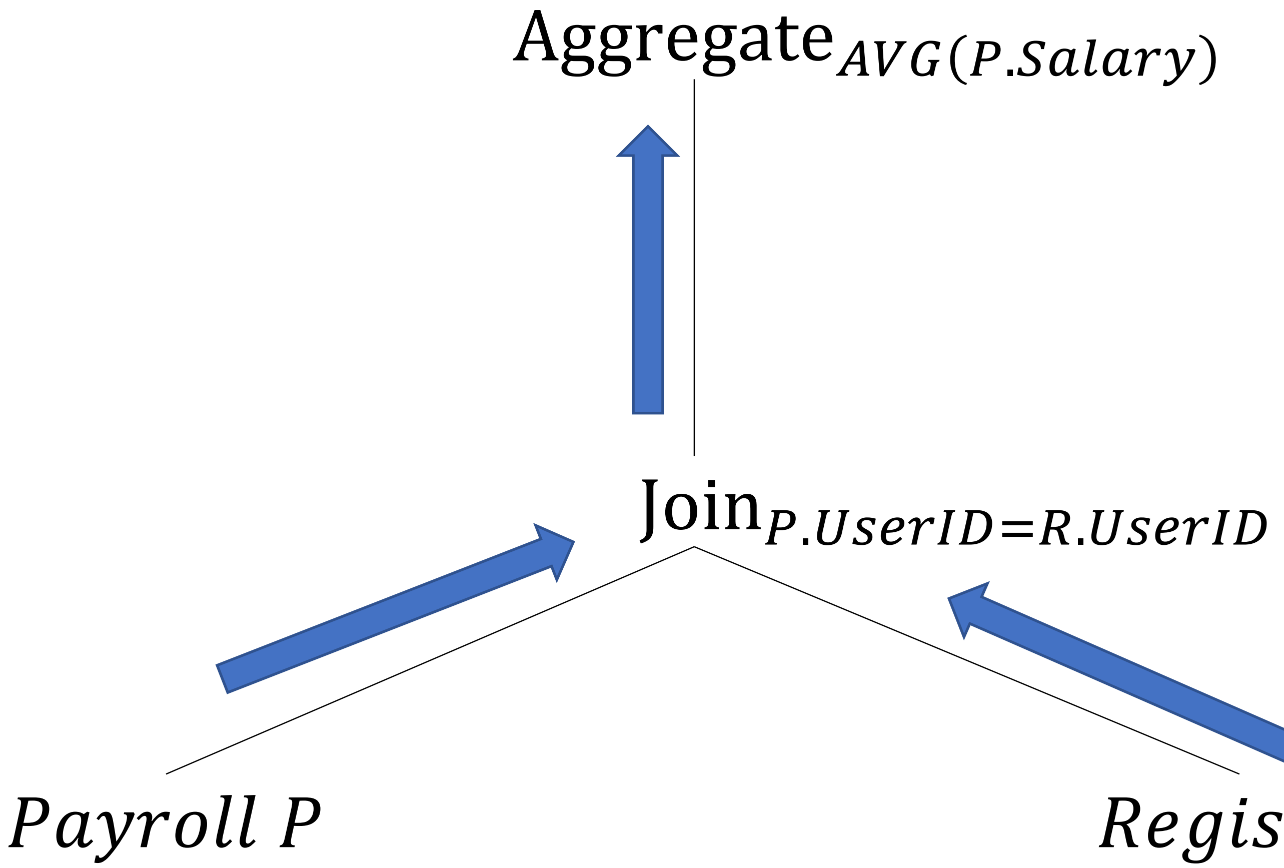
$$Payroll\ P$$

Tuples "flow up" the tree, getting modified along the way.

Another example from before…

```
SELECT AVG(P.Salary)
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID;
```

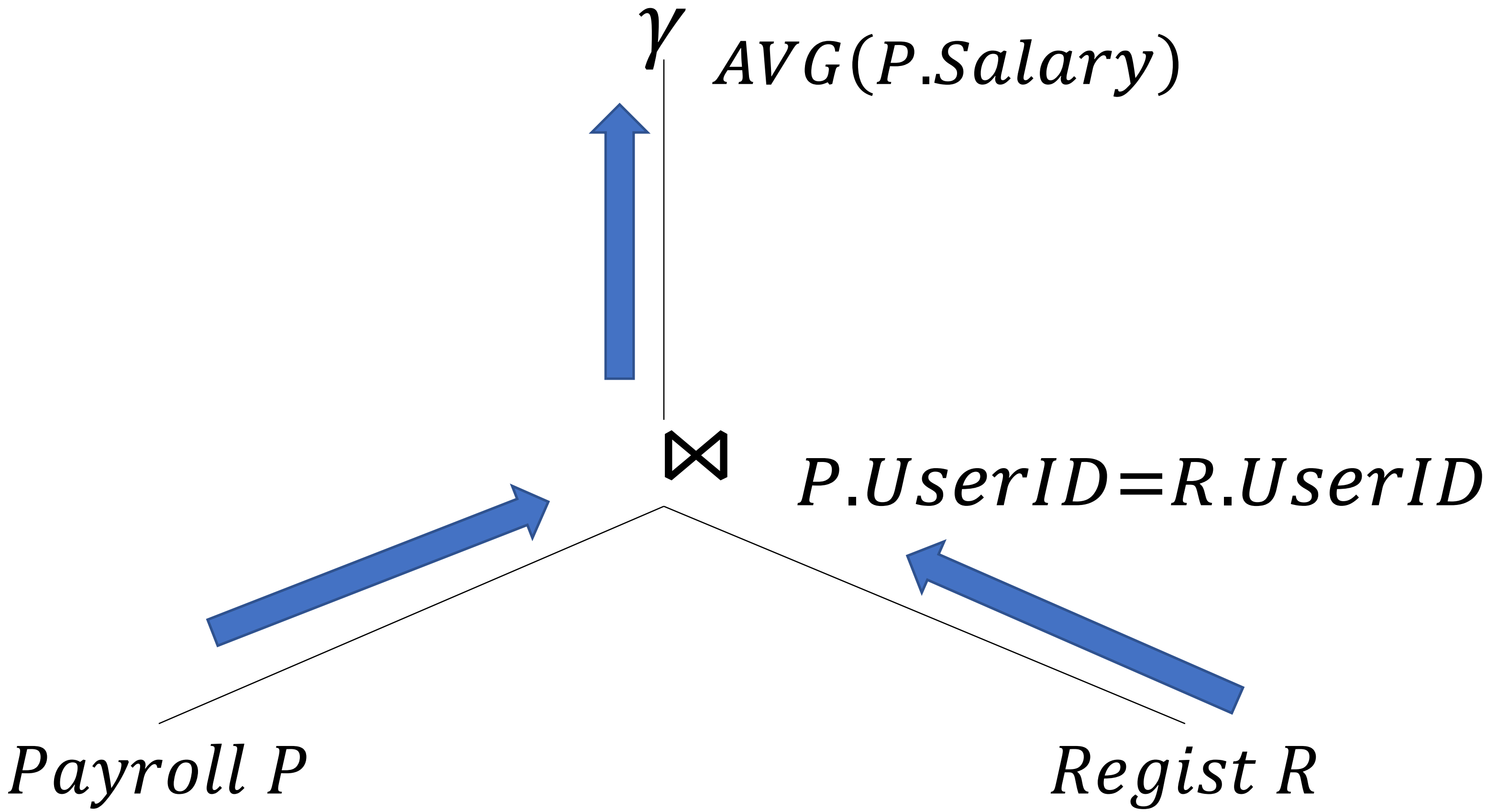
AVG(P.Salary)
76666



Another example from before…

```
SELECT AVG(P.Salary)
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID;
```

AVG(P.Salary)
76666



RA Operators

- Symbols are mostly Greek letters like π
  - σ (sigma)
  - γ (gamma)

  You don't have to know their Greek names, but this reference may be helpful:

  https://www.rapidtables.com/math/symbols/greek_alphabet.html
- Read RA tree from bottom to top
  - Bottom → Data sources
  - Top → Query output
- Semantics
  - Every operator takes 1 (unary) or 2 (binary) relations as inputs
  - Every operator outputs a relation

RA Operators

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

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

RA Operators

- For the curious…

$⟖$ Right Outer Join

$⟕$ Left Outer Join

$⟗$ Full Outer Join

ρ Rename

RA Operators

- Get ready for some math…

RA Operators

π Projection

- Unary operator
- Projection removes unspecified columns

$$\pi_{A,B}\big(T(A,B,C)\big) \rightarrow S(A,B)$$

A	B	C
1	2	3
4	5	6
7	8	9

A	B
1	2
4	5
7	8

RA Operators

σ Selection

- Unary operator
- Selection filters tuples from the input

$$\sigma_{T.A<6}\left(T(A,B,C)\right) \rightarrow S(A,B,C)$$

A	B	C
1	2	3
4	5	6
7	8	9

A	B	C
1	2	3
4	5	6

RA Operators

$\bowtie$ Join

- Binary operator
- Joins inputs relations on the specified condition

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

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

RA Operators

$\times$ Cartesian Product

- Binary operator
- Same semantics as in set theory
- Indiscriminate join of input relations

$$T(A,B) \times S(C,D) \rightarrow R(A,B,C,D)$$

RA Operators

γ Grouping & Aggregation

- Unary operator
- Specifies grouped attributes and then aggregates
- ONLY operation that can compute aggregates

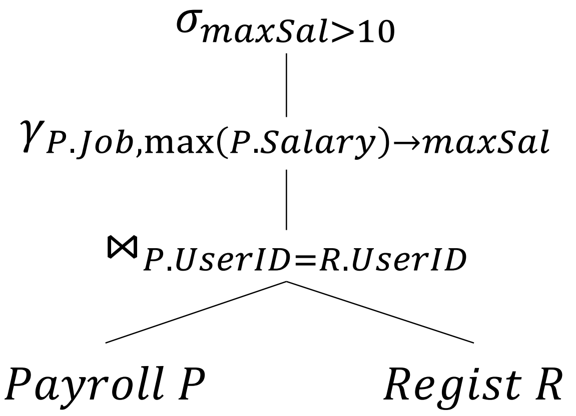
$$\gamma_{T.A,\max(T.B)\rightarrow mB}\big(T(A,B,C)\big) \rightarrow R(A,mB)$$

A	B	C
1	2	3
1	5	6
7	8	9

A	mB
1	5
7	8

Sometimes RA can be written in-line

$$\sigma_{maxSal>10}\left(\gamma_{P.Job,\max(P.Salary)\rightarrow maxSal}\left(\left(Payroll\ P\right) \bowtie_{P.UserID=R.UserID} \left(\sigma_{R.Car='Pinto'}\left(Regist\ R\right)\right)\right)\right)$$



RA Operators

τ Sort

- Unary operator
- Orders the input by any of the columns
- Assume default ascending order like in SQL

$$\tau_{T.A,T.B}\big(T(A,B,C)\big) \rightarrow R(A,B,C)$$

A	B	C
7	8	9
1	5	6
1	2	3

A	B	C
1	2	3
1	5	6
7	8	9

RA Operators

δ Duplicate Elimination

- Unary operator
- Deduplicates tuples
- Technically useless because it’s the same as grouping on all attributes

$$\delta\big(T(A,B,C)\big) \rightarrow R(A,B,C)$$

A	B	C
1	2	3
1	2	3
4	5	6

A	B	C
1	2	3
4	5	6

RA Operators

∪ Union ∩ Intersection

- Binary operators
- Same semantics as in set theory (but over bags)
- Input tables must have # columns and type

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

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)

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

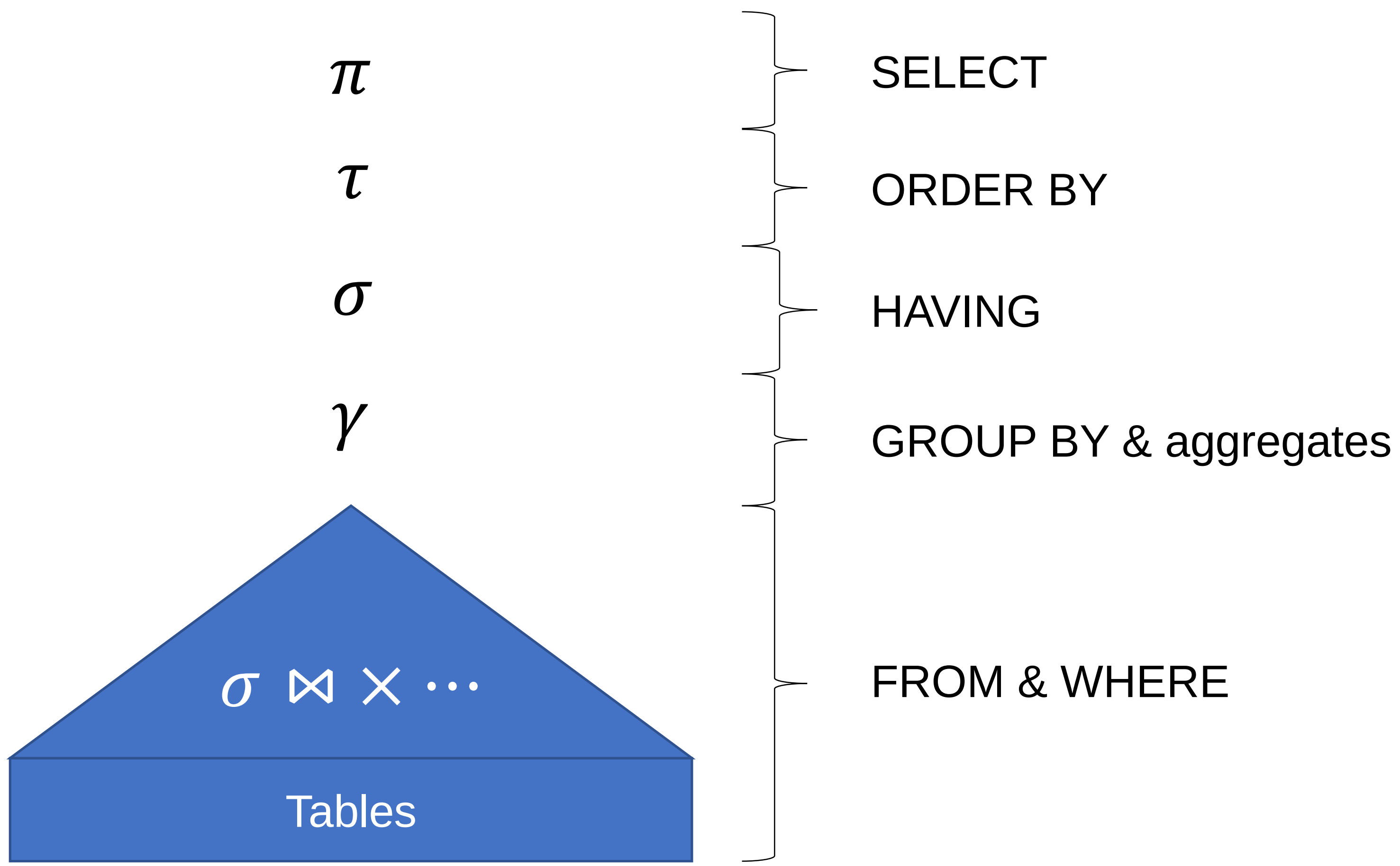
A	B
1	2
3	4

A	B
1	2
5	6

A	B
3	4

Basic SQL to RA Conversion

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



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(*)
```

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

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

$$Payroll\ P \qquad Regist\ R$$

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(*)
```

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

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

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

$$Payroll\ P \qquad Regist\ R$$

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(*)
```

$$\sigma_{cnt>1}$$

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

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

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

$$Payroll\ P \qquad Regist\ R$$

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(*)
```

$$\tau_{cnt}$$

$$\sigma_{cnt>1}$$

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

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

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

$$Payroll\ P \qquad Regist\ R$$

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(*)
```

$$\pi_{P.Name}$$

$$\tau_{cnt}$$

$$\sigma_{cnt>1}$$

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

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

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

$$Payroll\ P \qquad Regist\ R$$

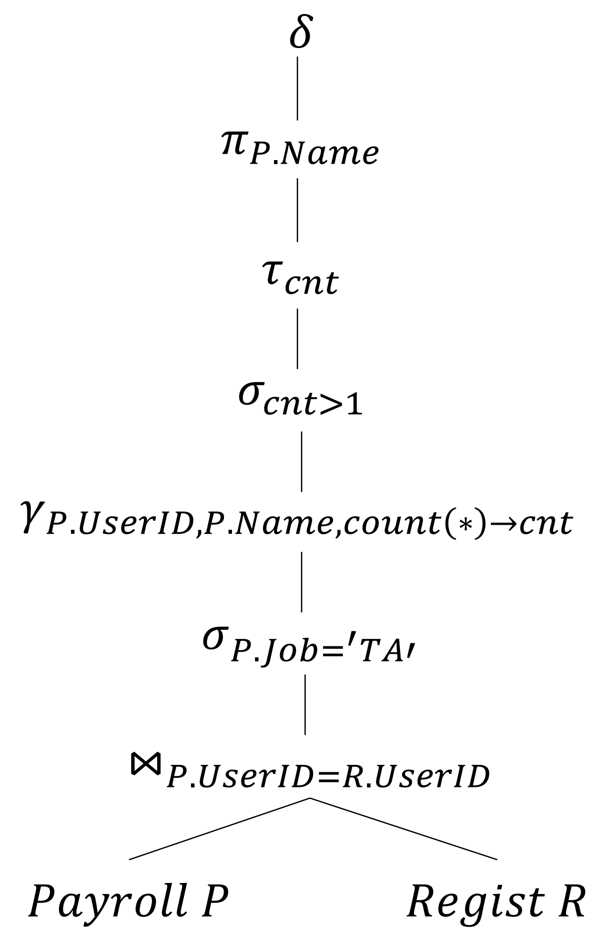
English to SQL to RA Example

```
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(*)
```



Summary of RA

- SQL = a declarative language where we say ***what*** data we want to retrieve

- RA = an algebra where we say ***how*** we want to retrieve the data

- RDMS translates SQL to RA then optimizes for performance