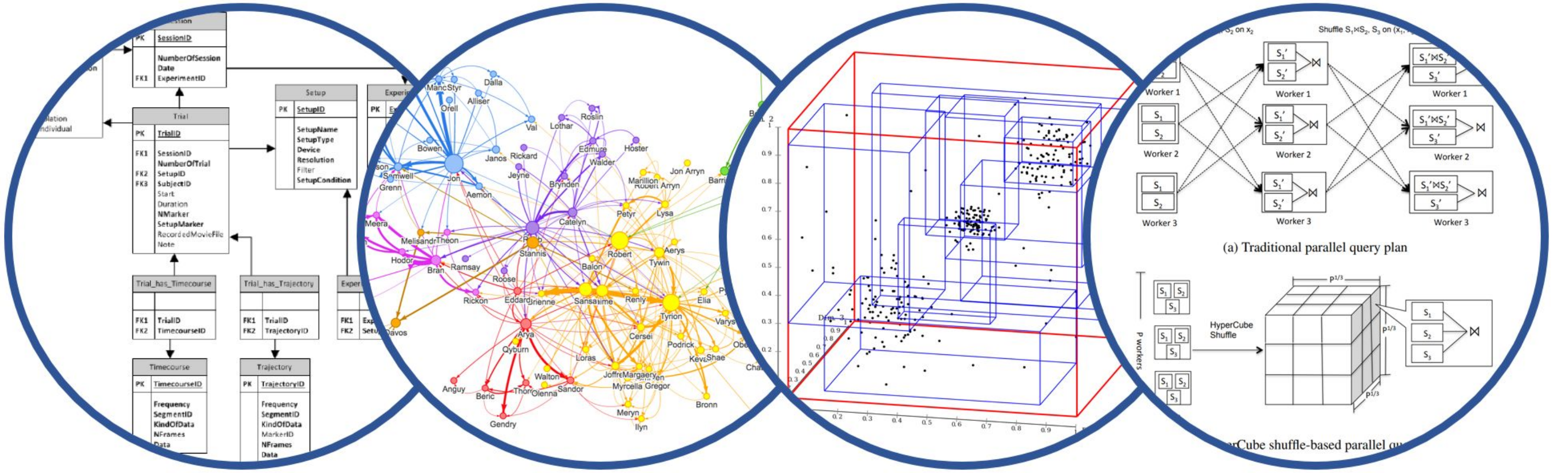


# Introduction to Data Management

# Relational Algebra

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

# The Witnessing Problem

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

Return the person with the highest salary for each job type

# The Witnessing Problem

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

Return the person with the highest salary for each job type

```
SELECT P1.Name, MAX(P2.Salary)
  FROM Payroll AS P1, Payroll AS P2
 WHERE P1.Job = P2.Job
 GROUP BY P2.Job, P1.Salary, P1.Name
HAVING P1.Salary = MAX(P2.Salary)
```

# The Witnessing Problem

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

Return the person with the highest salary for each job type

the maxima

```
SELECT P1.Name, MAX(P2.Salary)
  FROM Payroll AS P1, Payroll AS P2
 WHERE P1.Job = P2.Job
 GROUP BY P2.Job, P1.Salary, P1.Name
HAVING P1.Salary = MAX(P2.Salary)
```

# The Witnessing Problem

```
SELECT P1.Name, MAX(P2.Salary)
  FROM Payroll AS P1, Payroll AS P2
 WHERE P1.Job = P2.Job
 GROUP BY P2.Job, P1.Salary, P1.Name
HAVING P1.Salary = MAX(P2.Salary)
```

Join on “original” grouping attributes

| P1 | | | | P2 | | | |
|---|---|---|---|---|---|---|---|
| **UserID** | **Name** | **Job** | **Salary** | **UserID** | **Name** | **Job** | **Salary** |
| 123 | Jack | TA | 50000 | 123 | Jack | TA | 50000 |
| 123 | Jack | TA | 50000 | 345 | Allison | TA | 60000 |
| 345 | Allison | TA | 60000 | 345 | Allison | TA | 60000 |
| 345 | Allison | TA | 60000 | 123 | Jack | TA | 50000 |
| 567 | Magda | Prof | 90000 | 567 | Magda | Prof | 90000 |
| 567 | Magda | Prof | 90000 | 789 | Dan | Prof | 100000 |
| 789 | Dan | Prof | 100000 | 789 | Dan | Prof | 100000 |
| 789 | Dan | Prof | 100000 | 567 | Magda | Prof | 90000 |

# The Witnessing Problem

```
SELECT P1.Name, MAX(P2.Salary)
  FROM Payroll AS P1, Payroll AS P2
 WHERE P1.Job = P2.Job
 GROUP BY P2.Job, P1.Salary, P1.Name
HAVING P1.Salary = MAX(P2.Salary)
```

Group on additional attributes that you are argmax-ing for

| P1 | | | | P2 | | | |
|---|---|---|---|---|---|---|---|
| UserID | Name | Job | Salary | UserID | Name | Job | Salary |
| 123 | Jack | TA | 50000 | 123 | Jack | TA | 50000 |
| 123 | Jack | TA | 50000 | 345 | Allison | TA | 60000 |
| 345 | Allison | TA | 60000 | 345 | Allison | TA | 60000 |
| 345 | Allison | TA | 60000 | 123 | Jack | TA | 50000 |
| 567 | Magda | Prof | 90000 | 567 | Magda | Prof | 90000 |
| 567 | Magda | Prof | 90000 | 789 | Dan | Prof | 100000 |
| 789 | Dan | Prof | 100000 | 789 | Dan | Prof | 100000 |
| 789 | Dan | Prof | 100000 | 567 | Magda | Prof | 90000 |

# The Witnessing Problem

```
SELECT P1.Name, MAX(P2.Salary)
  FROM Payroll AS P1, Payroll AS P2
 WHERE P1.Job = P2.Job
 GROUP BY P2.Job, P1.Salary, P1.Name
HAVING P1.Salary = MAX(P2.Salary)
```

Group on additional attributes that you are argmax-ing for

| P1 | | | | P2 | | | |
|---|---|---|---|---|---|---|---|
| UserID | Name | Job | Salary | UserID | Name | Job | Salary |
| 123 | Jack | TA | 50000 | 123 | Jack | TA | 50000 |
| 123 | Jack | TA | 50000 | 345 | Allison | TA | 60000 |
| 345 | Allison | TA | 60000 | 345 | Allison | TA | 60000 |
| 345 | Allison | TA | 60000 | 123 | Jack | TA | 50000 |
| 567 | Magda | Prof | 90000 | 567 | Magda | Prof | 90000 |
| 567 | Magda | Prof | 90000 | 789 | Dan | Prof | 100000 |
| 789 | Dan | Prof | 100000 | 789 | Dan | Prof | 100000 |
| 789 | Dan | Prof | 100000 | 567 | Magda | Prof | 90000 |

# The Witnessing Problem

```
SELECT P1.Name, MAX(P2.Salary)
  FROM Payroll AS P1, Payroll AS P2
 WHERE P1.Job = P2.Job
 GROUP BY P2.Job, P1.Salary, P1.Name
HAVING P1.Salary = MAX(P2.Salary)
```

Group on additional attributes that you are argmax-ing for

| P1 | | | | P2 | | | |
|---|---|---|---|---|---|---|---|
| UserID | Name | Job | Salary | UserID | Name | Job | Salary |
| 123 | Jack | TA | 50000 | 123 | Jack | TA | 50000 |
| 123 | Jack | TA | 50000 | 345 | Allison | TA | 60000 |
| 345 | Allison | TA | 60000 | 345 | Allison | TA | 60000 |
| 345 | Allison | TA | 60000 | 123 | Jack | TA | 50000 |
| 567 | Magda | Prof | 90000 | 567 | Magda | Prof | 90000 |
| 567 | Magda | Prof | 90000 | 789 | Dan | Prof | 100000 |
| 789 | Dan | Prof | 100000 | 789 | Dan | Prof | 100000 |
| 789 | Dan | Prof | 100000 | 567 | Magda | Prof | 90000 |

# The Witnessing Problem

```
SELECT P1.Name, MAX(P2.Salary)
  FROM Payroll AS P1, Payroll AS P2
 WHERE P1.Job = P2.Job
 GROUP BY P2.Job, P1.Salary, P1.Name
HAVING P1.Salary = MAX(P2.Salary)
```

| P1 | | | | P2 | | | |
|---|---|---|---|---|---|---|---|
| UserID | Name | Job | Salary | UserID | Name | Job | Salary |
| ~~123~~ | ~~Jack~~ | ~~TA~~ | ~~50000~~ | ~~123~~ | ~~Jack~~ | ~~TA~~ | ~~50000~~ |
| ~~123~~ | ~~Jack~~ | ~~TA~~ | ~~50000~~ | ~~345~~ | ~~Allison~~ | ~~TA~~ | ~~60000~~ |
| 345 | Allison | TA | 60000 | 345 | Allison | TA | 60000 |
| 345 | Allison | TA | 60000 | 123 | Jack | TA | 50000 |
| 567 | Magda | Prof | 90000 | 567 | Magda | Prof | 90000 |
| 567 | Magda | Prof | 90000 | 789 | Dan | Prof | 100000 |
| 789 | Dan | Prof | 100000 | 789 | Dan | Prof | 100000 |
| 789 | Dan | Prof | 100000 | 567 | Magda | Prof | 90000 |

# The Witnessing Problem

```
SELECT P1.Name, MAX(P2.Salary)
  FROM Payroll AS P1, Payroll AS P2
 WHERE P1.Job = P2.Job
 GROUP BY P2.Job, P1.Salary, P1.Name
HAVING P1.Salary = MAX(P2.Salary)
```

| Name | MAX(Salary) |
|---|---|
| Allison | 60000 |
| Dan | 100000 |

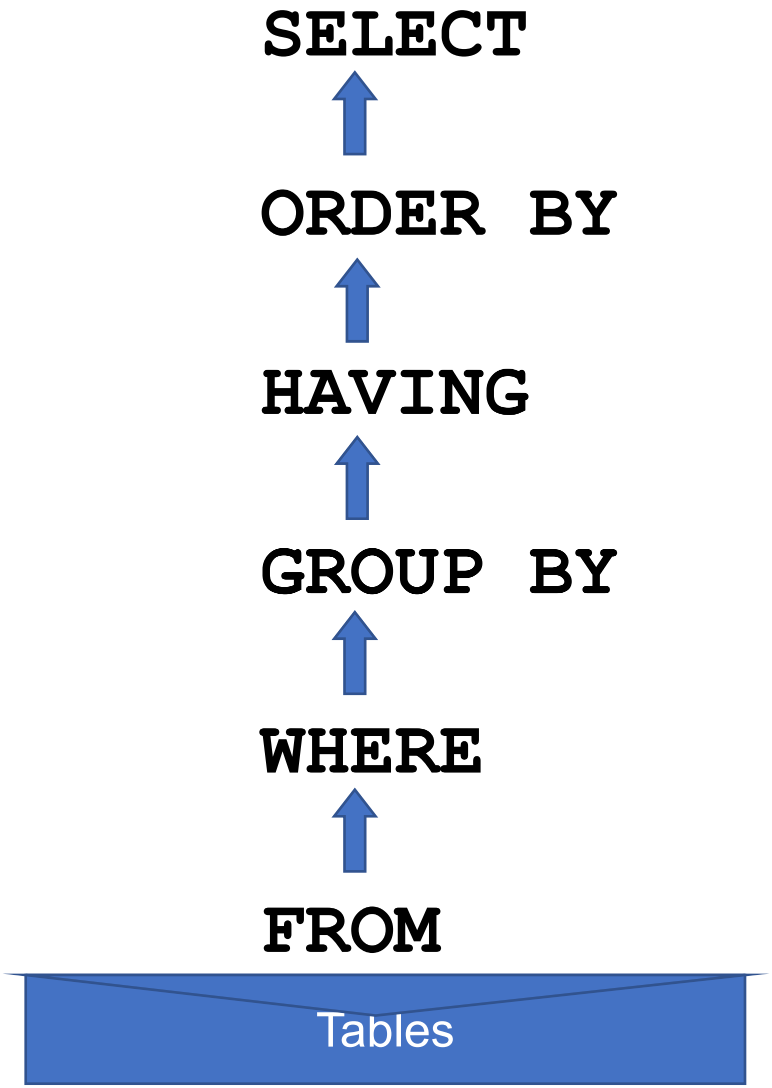
# Outline

- 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 …
```



SELECT

ORDER BY

HAVING

GROUP BY

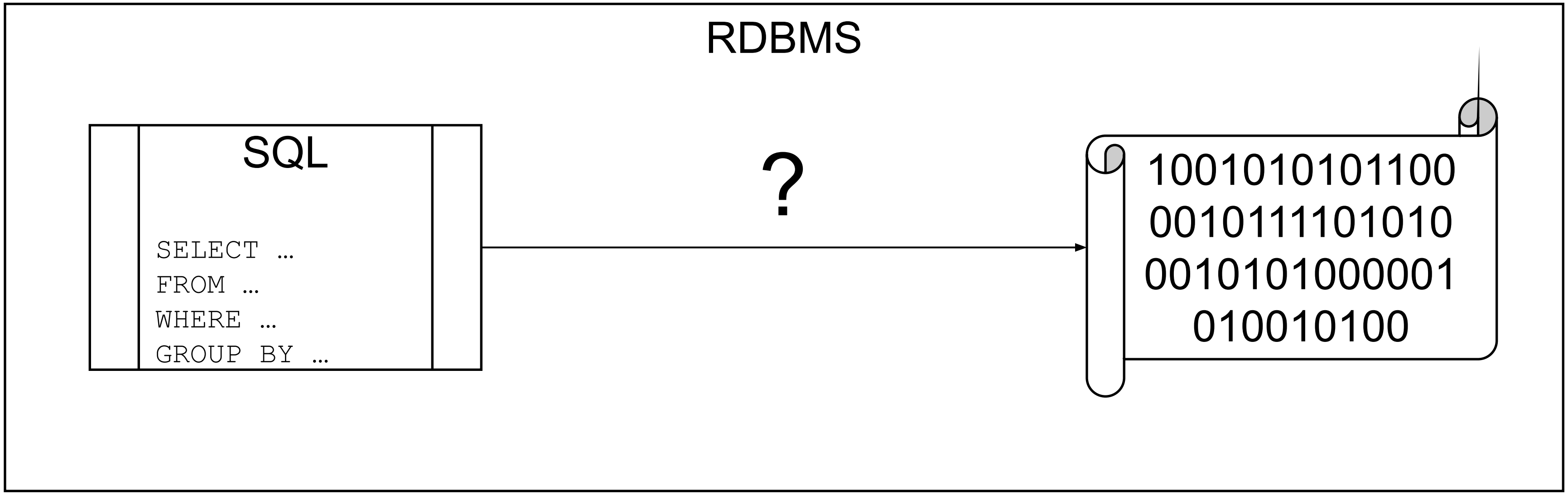
WHERE

FROM

Tables

# What’s the Point of RA?

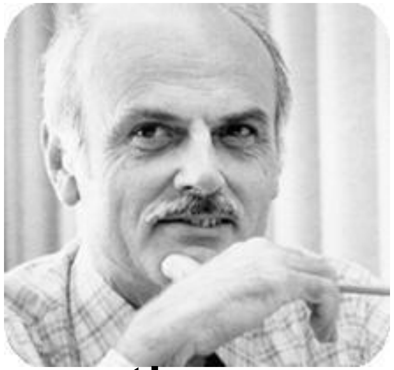
- SQL is a **Declarative Language**
  - “What to get” rather than “how to get it”
  - Easier to write a SQL query than write a whole Java program that will probably perform worse
- But computers are imperative/procedural
  - Computers only understand the “how”



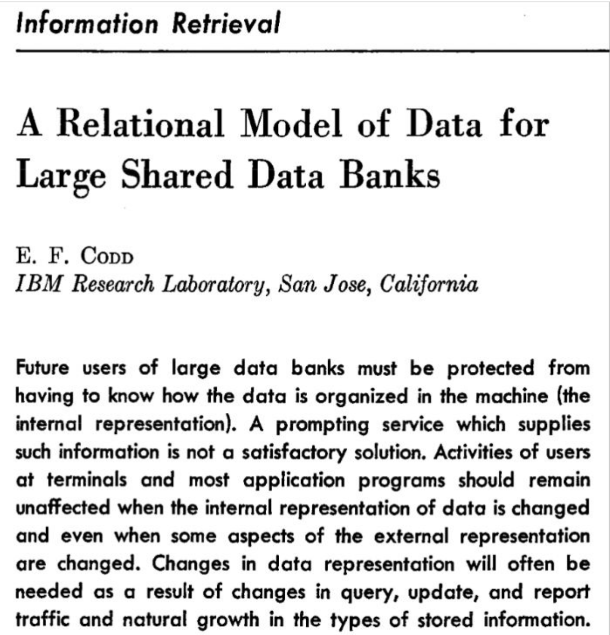
```
SELECT …
FROM …
WHERE …
GROUP BY …
```

# 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



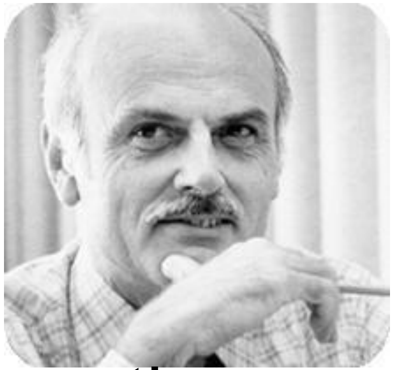
*Information Retrieval*

A Relational Model of Data for Large Shared Data Banks

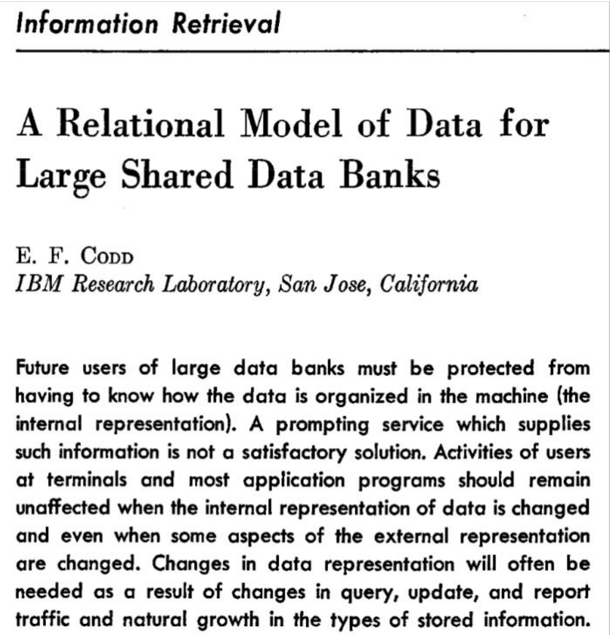
E. F. Codd
*IBM Research Laboratory, San Jose, California*

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**



*Information Retrieval*

A Relational Model of Data for Large Shared Data Banks

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

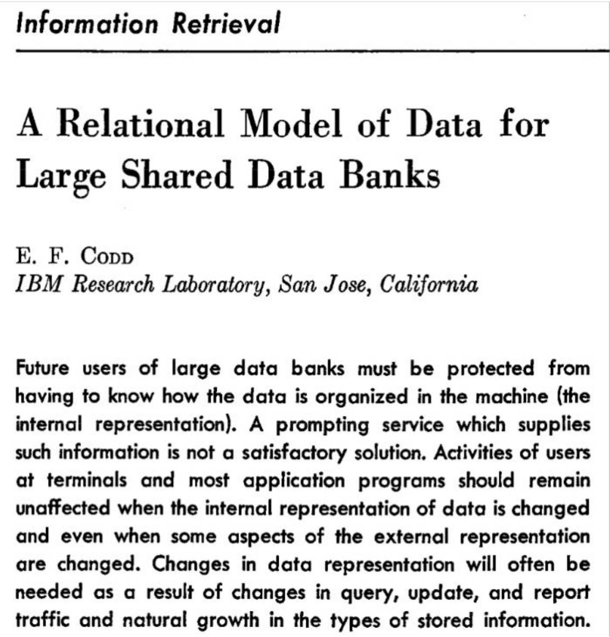
Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain 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.

- IBM initially ignored his techniques
- 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*

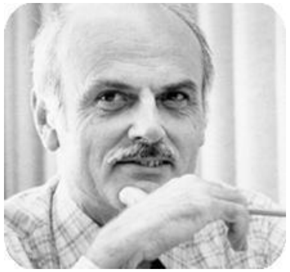
A Relational Model of Data for Large Shared Data Banks

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*



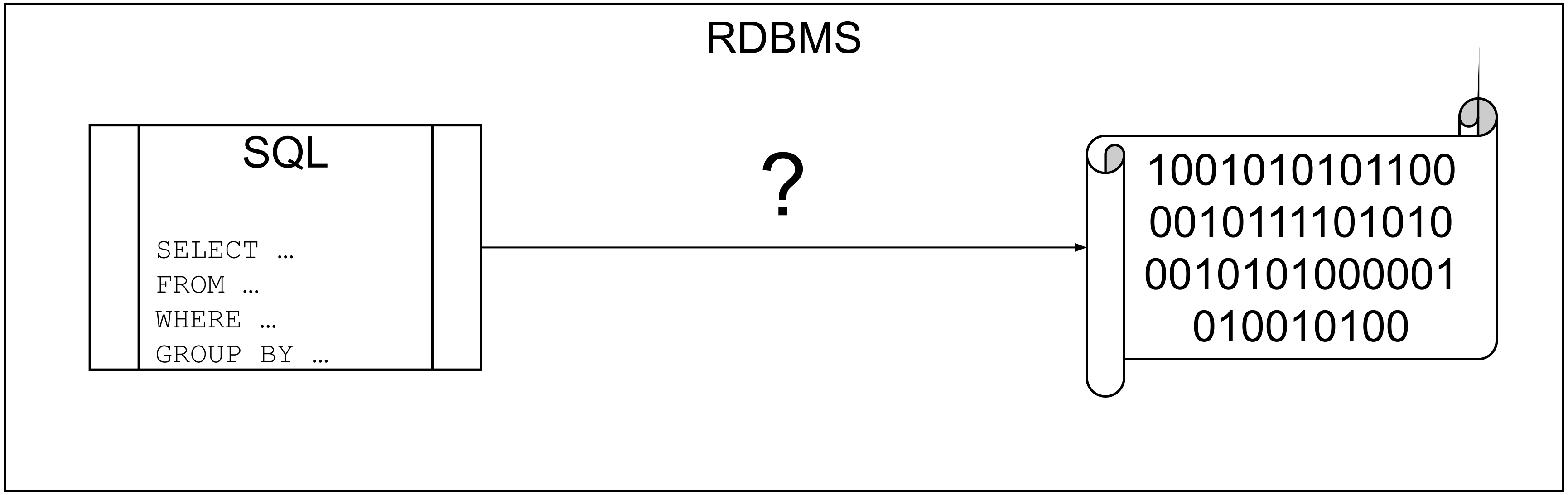
Ted Codd, 1981
*Relational model*

Jim Gray, 1998
*Transaction processing*

Michael Stonebraker, 2014
*INGRES and Postgres*

# What's the Point of RA?

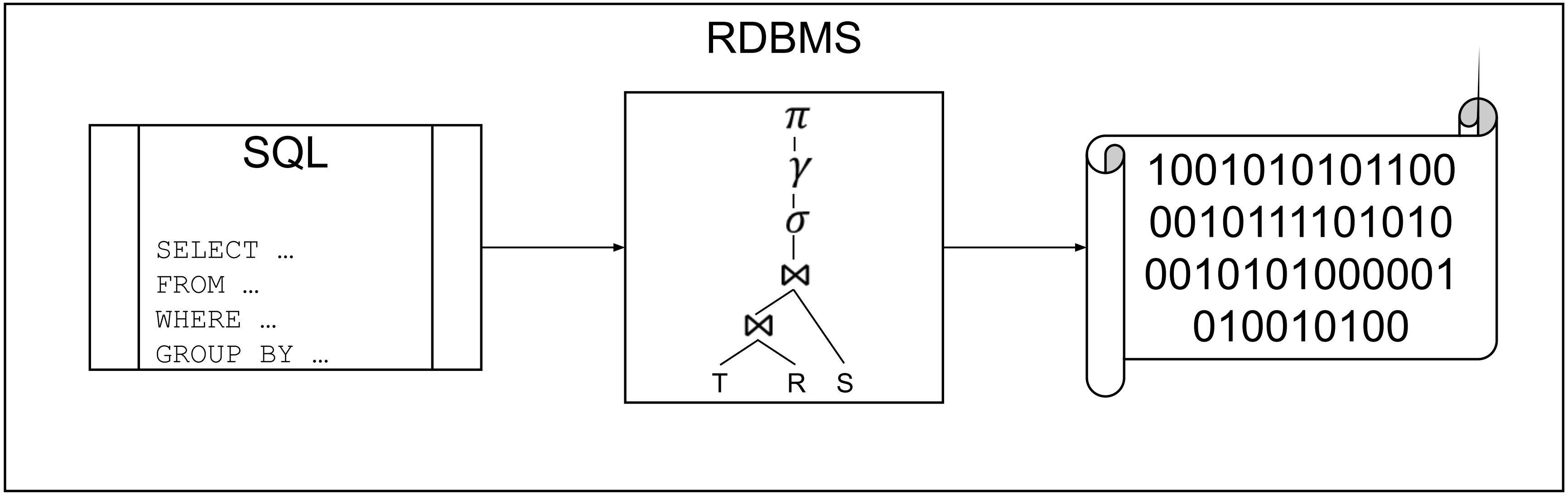
- We need a language that reads more like **instructions** but still captures the fundamental operations of a query



```
SELECT …
FROM …
WHERE …
GROUP BY …
```

# 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)



RDBMS

SQL

```
SELECT …
FROM …
WHERE …
GROUP BY …
```

1001010101100
0010111101010
0010101000001
010010100

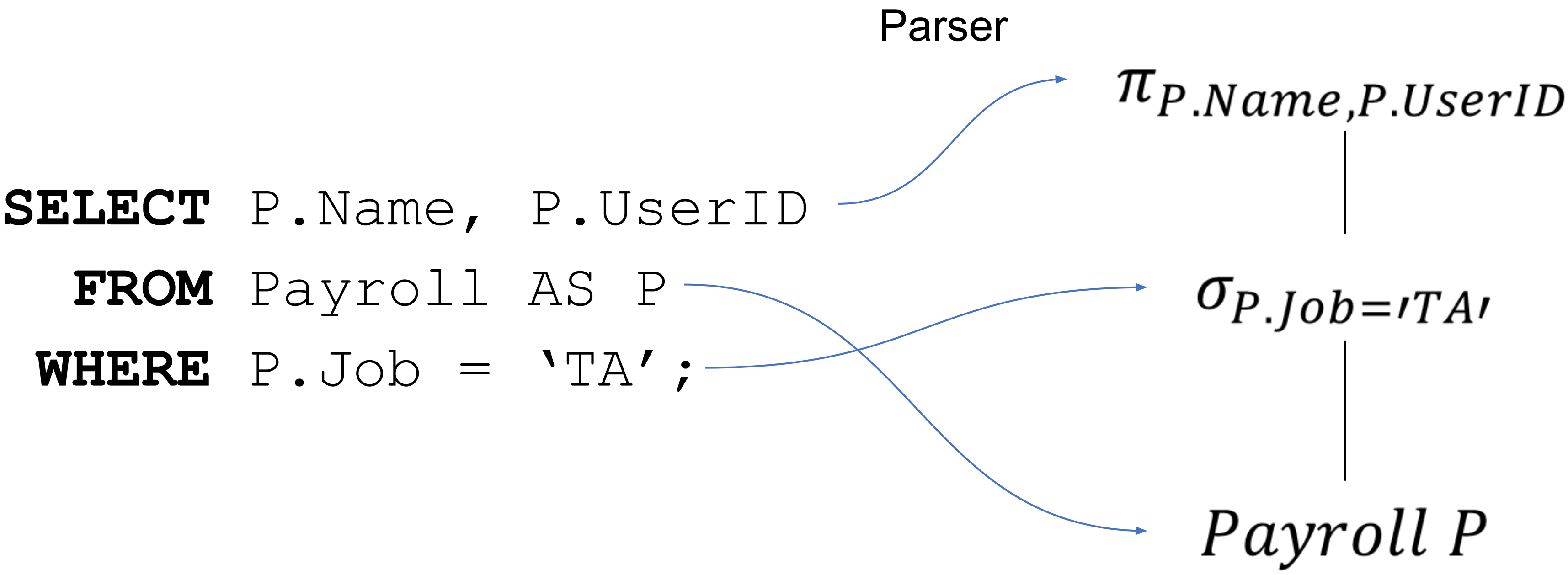
# Flashback to our first query

- 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)



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

Parser

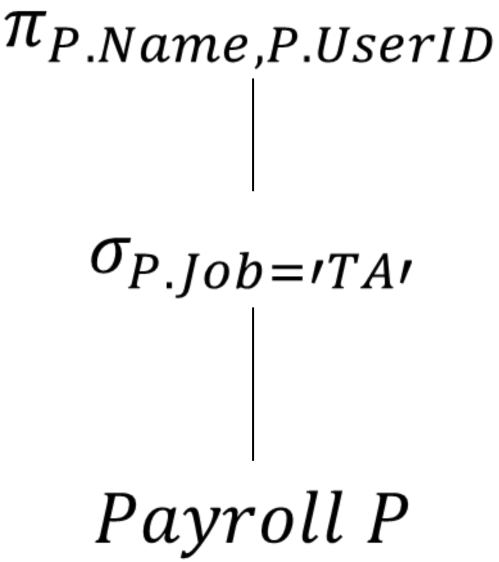
$\pi_{P.Name,P.UserID}$

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

$Payroll\ P$

# Flashback to our first query

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



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

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

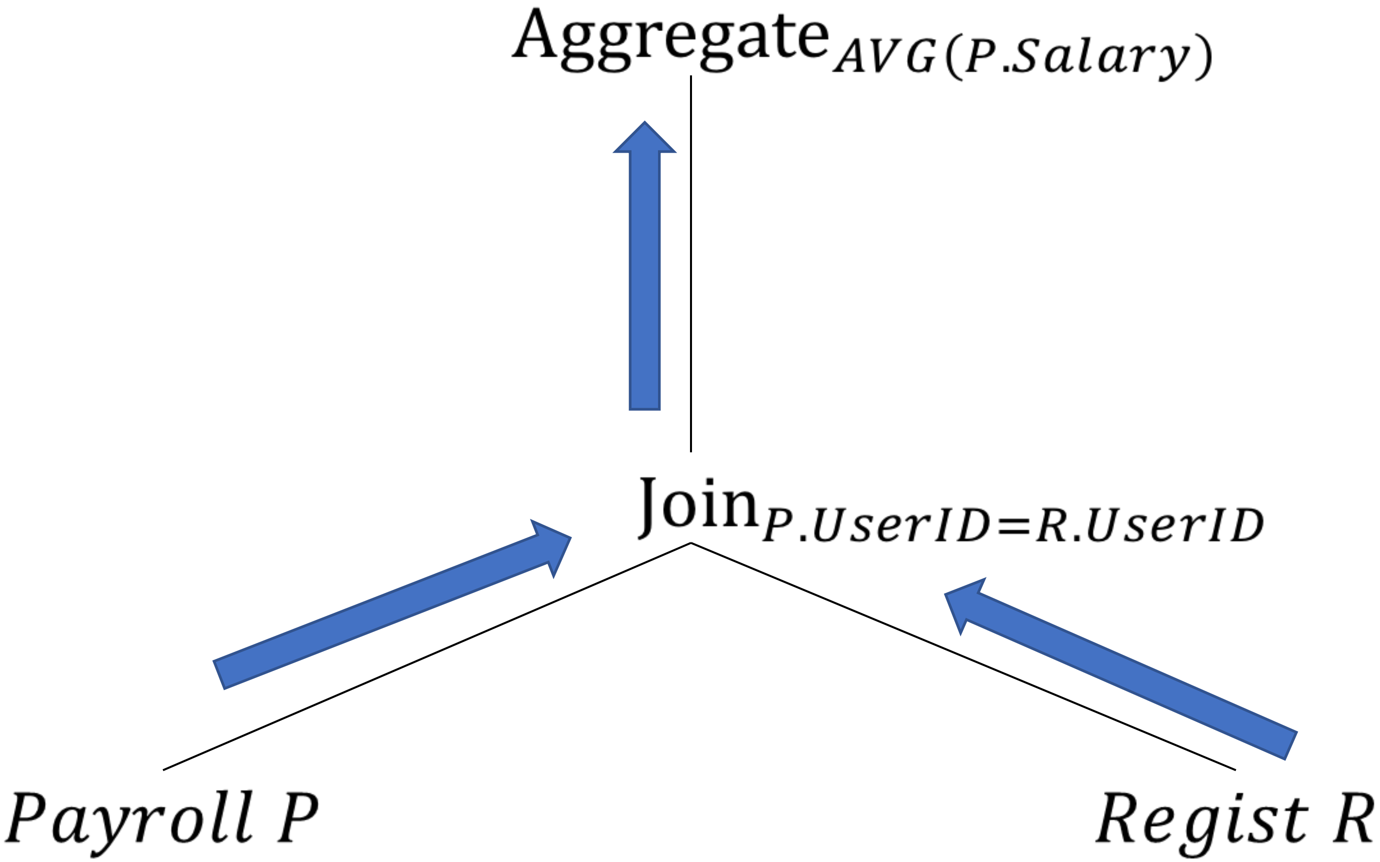
$Payroll\ P$

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 |



$\text{Aggregate}_{AVG(P.Salary)}$

$\text{Join}_{P.UserID=R.UserID}$

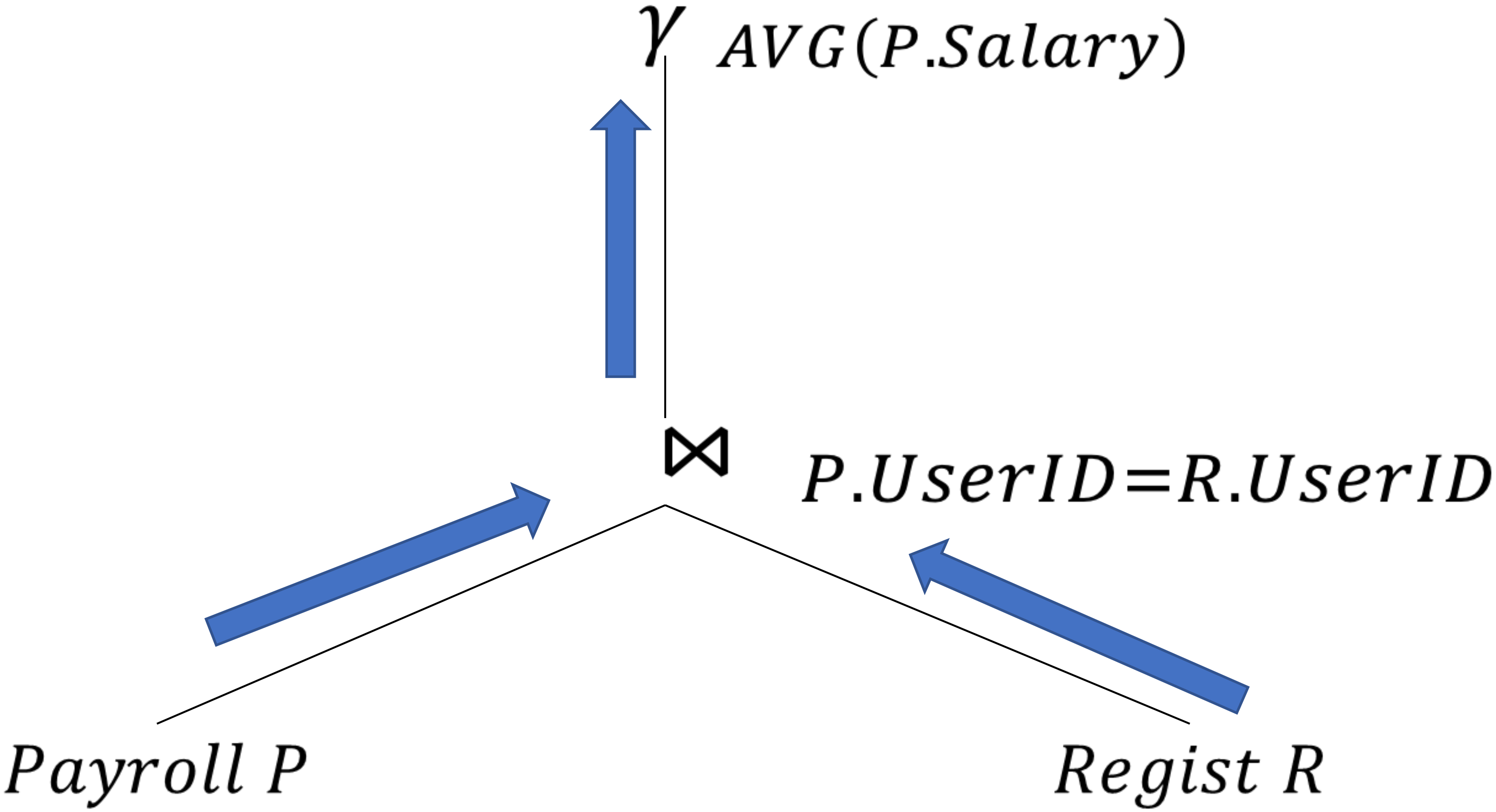
$Payroll\ P$

$Regist\ R$

# Another example from before…

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

| AVG(P.Salary) |
|---|
| 76666 |



$\gamma_{AVG(P.Salary)}$

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

$Payroll\ P$

$Regist\ R$

# RA Operators

- Symbols are mostly Greek letters like $\pi$
  - $\sigma$ (sigma)
  - $\gamma$ (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 | $\gamma$ | Grouping & Aggregation | $\tau$ | Sort |
| $\times$ | Cartesian Product | $\cup$ | Union | $\delta$ | Duplicate Elimination |
| $\sigma$ | Selection | $\cap$ | Intersection | | |
| $\pi$ | Projection | $-$ | Difference | | |

# RA Operators

- For the curious…

| | | | |
|---|---|---|---|
| ⟖ | Right Outer Join | | |
| ⟕ | Left Outer Join | $\rho$ | Rename |
| ⟗ | Full Outer Join | | |

# RA Operators

- Get ready for some definitions…

# RA Operators

$\pi$ Projection ⟷ **`SELECT`** …

- 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

$\sigma$ Selection ⟷ **WHERE** …
**HAVING** …

- Unary operator
- Selection filters tuples from the input

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

| 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

$\gamma$ Grouping & Aggregation ⟷ **GROUP BY** ***(+Aggregates)***

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

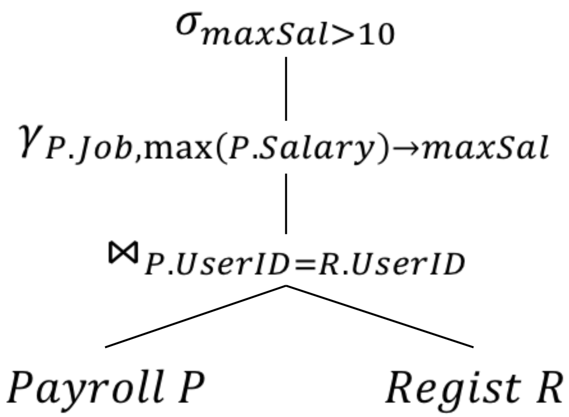
$$\gamma_{T.A,\max(T.B)\rightarrow mB}\big(T(A,B,C)\big) \rightarrow R(A,mB)$$

| 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

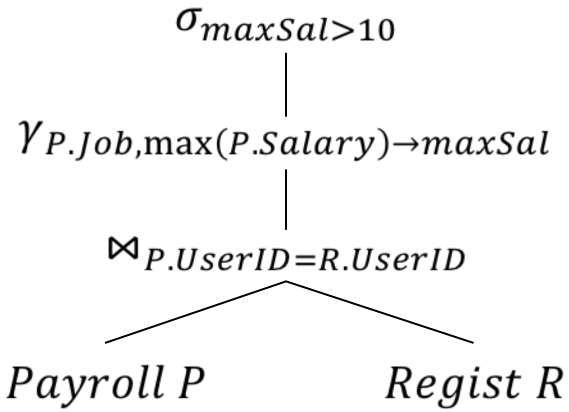
$$
\begin{aligned}
&\sigma_{maxSal>10} \\
&\quad(\gamma_{P.Job,\max(P.Salary)\to maxSal} \\
&\qquad((Payroll\ P) \bowtie_{P.UserID=R.UserID} \\
&\qquad\quad( \qquad\qquad\qquad (Regist\ R)))))
\end{aligned}
$$


$\sigma_{maxSal>10}$
$\gamma_{P.Job,\max(P.Salary)\to maxSal}$
$\bowtie_{P.UserID=R.UserID}$
$Payroll\ P$
$Regist\ R$

# Sometimes RA can be written in-line

The tree style is easier and preferred!

$$
\begin{aligned}
&\sigma_{maxSal>10} \\
&\quad (\gamma_{P.Job,\max(P.Salary)\rightarrow maxSal} \\
&\qquad ((Payroll\ P) \bowtie_{P.UserID=R.UserID} \\
&\qquad\quad ( \qquad\qquad\qquad (Regist\ R)))))
\end{aligned}
$$


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

# RA Operators

$\tau$ 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

$\delta$ Duplicate Elimination

- Unary operator
- Deduplicates tuples
- Could get the same effect by grouping on all attributes (if you haven’t used a group by yet)

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

| 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)

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

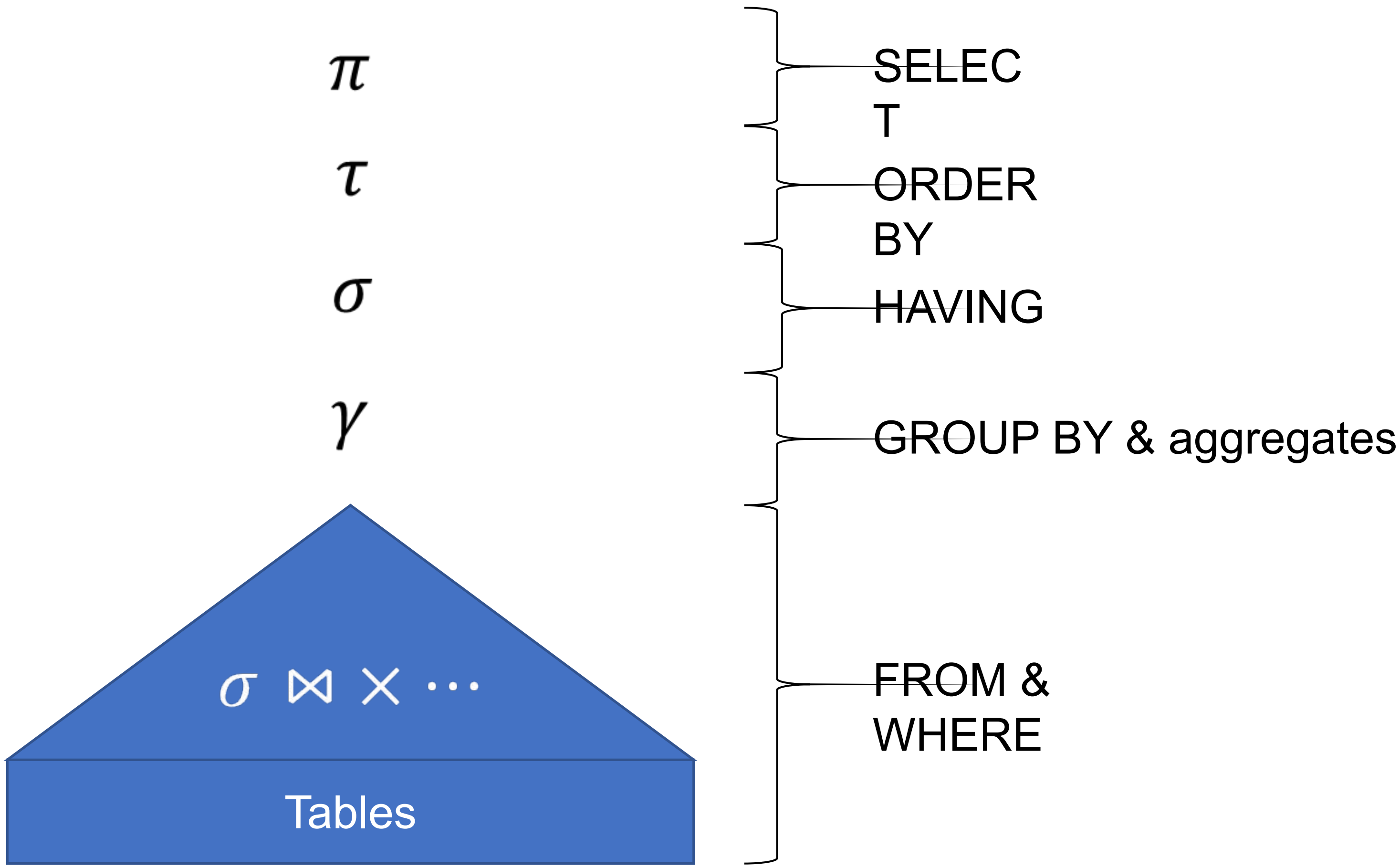
| A | B |
|---|---|
| 1 | 2 |
| 3 | 4 |

| 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(*)
```

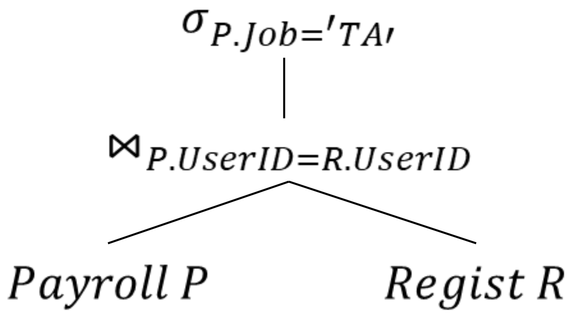
# English to SQL to RA Example

```
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(*)
```



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

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

$Payroll\ P$  $Regist\ R$

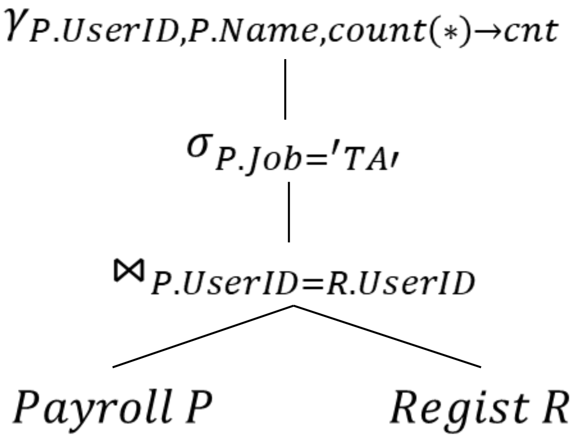
# English to SQL to RA Example

```
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(*)
```



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

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

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

$Payroll\ P$

$Regist\ R$

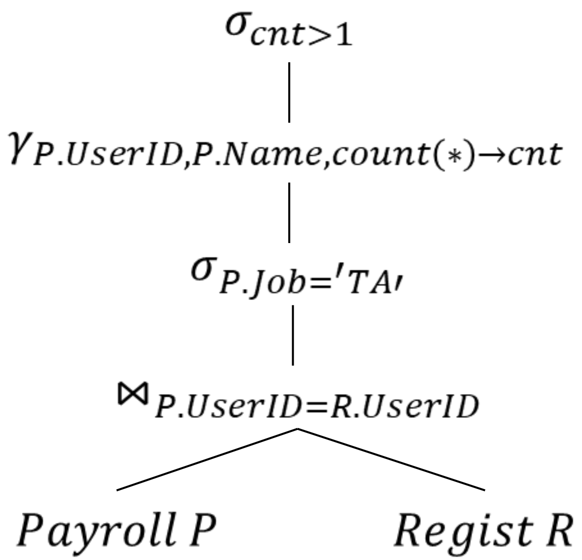
# English to SQL to RA Example

```
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(*)
```



$\sigma_{cnt>1}$

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

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

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

$Payroll\ P$

$Regist\ R$

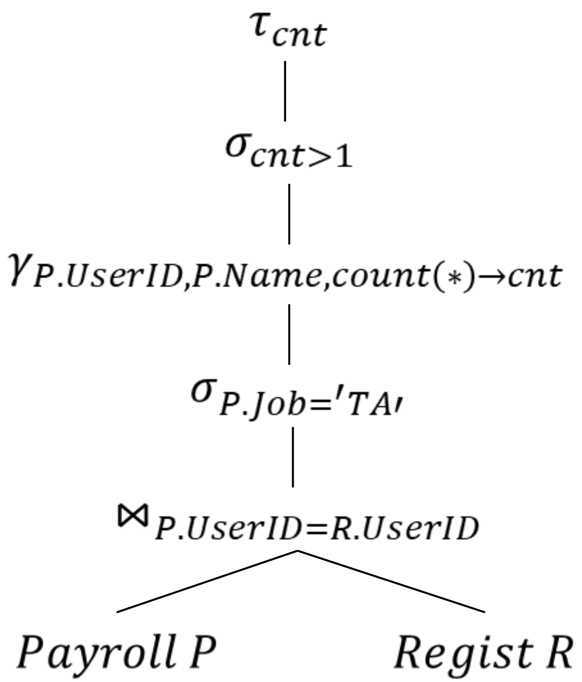
# English to SQL to RA Example

```
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(*)
```



$\tau_{cnt}$

$\sigma_{cnt>1}$

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

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

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

$Payroll\ P$ $\quad$ $Regist\ R$

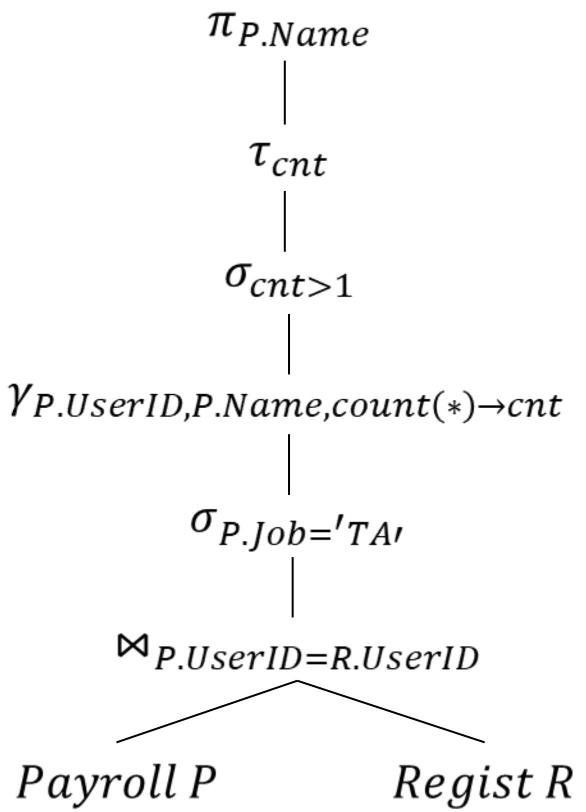
# English to SQL to RA Example

```
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(*)
```



$\pi_{P.Name}$

$\tau_{cnt}$

$\sigma_{cnt>1}$

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

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

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

$Payroll\ P$ $\quad$ $Regist\ R$

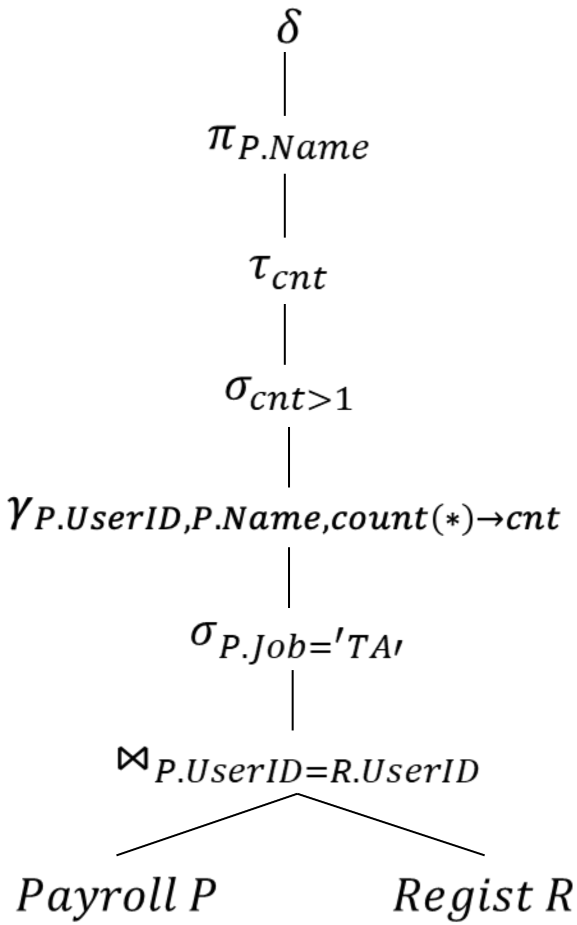
# English to SQL to RA Example

```
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(*)
```



$\delta$

$\pi_{P.Name}$

$\tau_{cnt}$

$\sigma_{cnt>1}$

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

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

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

$Payroll\ P$ $\quad$ $Regist\ R$

# How about Subqueries?

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
```

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = ‘WA’
  and Q.sno not in
  (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

# How about Subqueries?

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
```

Remove subquery

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = ‘WA’
  and Q.sno not in
  (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = ‘WA’)
    EXCEPT
 (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

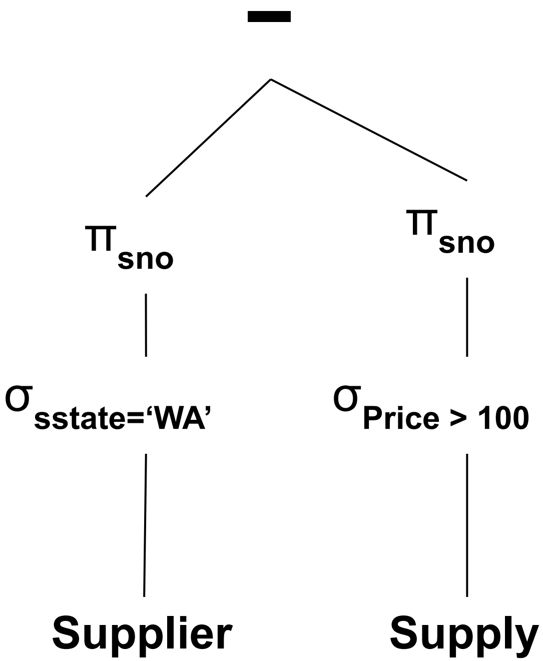
EXCEPT = set difference

# How about Subqueries?

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
```

```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = ‘WA’)
    EXCEPT
 (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

Now we have operator



# 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