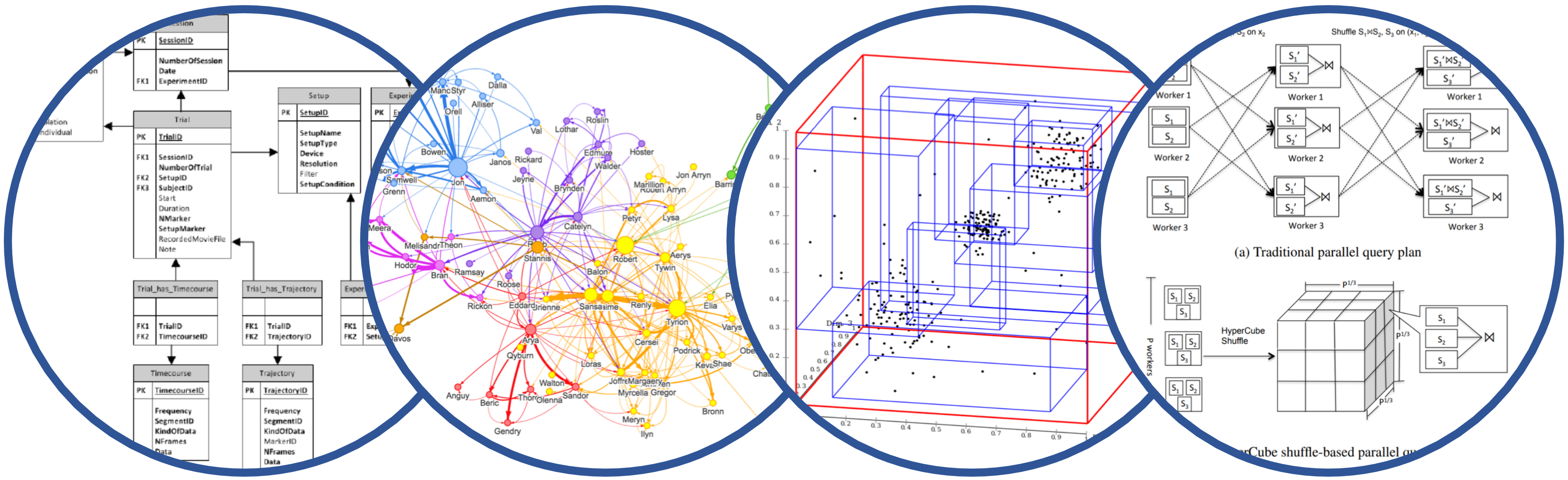


Introduction to Data Management

Relational Algebra

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

Recap

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

What am I aggregating over in a SELECT-FROM-WHERE query?

Answer:
The resulting tuples AFTER the join

There is an implicit "order of operations"

Order of operations

FROM → WHERE → ORDER BY → SELECT

"FWOS"

Order of operations

"FWOS"

SELECT

↑

ORDER BY

↑

WHERE

↑

FROM

Order of operations

"FWOS"

SELECT

↑

ORDER BY

↑

WHERE

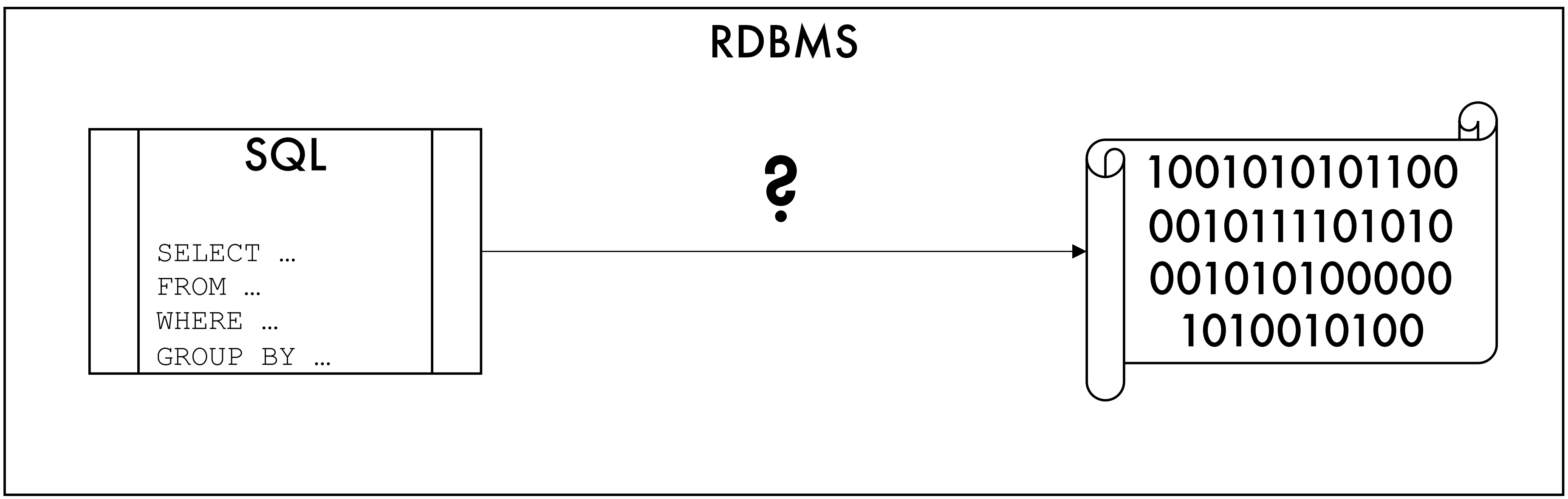
↑

FROM

This is the concept of a query plan in **Relational Algebra**

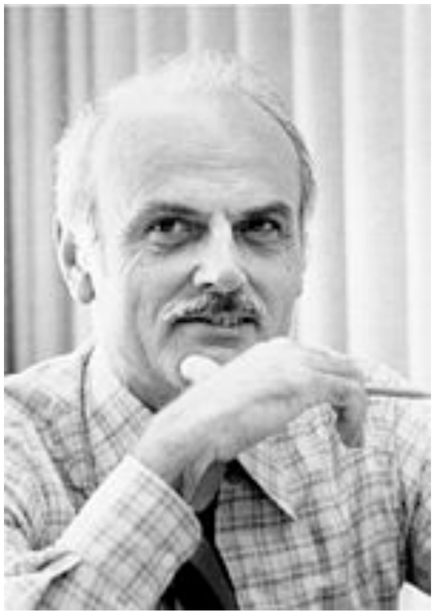
What's the Point of Relational Algebra?

- 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

Formalized and published by Ted Codd of IBM



Information Retrieval

A Relational Model of Data for Large Shared Data Banks

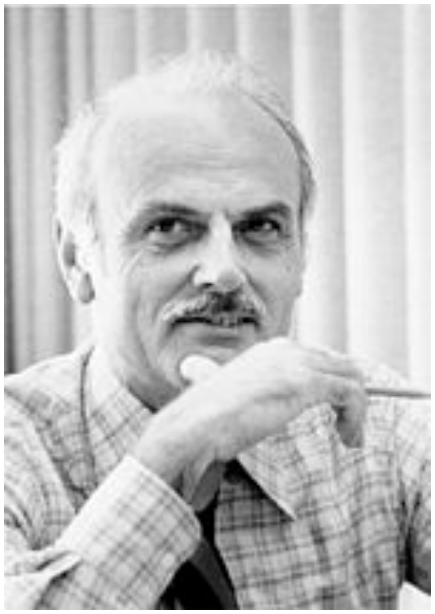
E. F. Codd
IBM Research Laboratory, San Jose, California

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.

Initially IBM didn't use his techniques…
10 years later he won the Turing award

History of RA

Formalized and published by Ted Codd of IBM



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

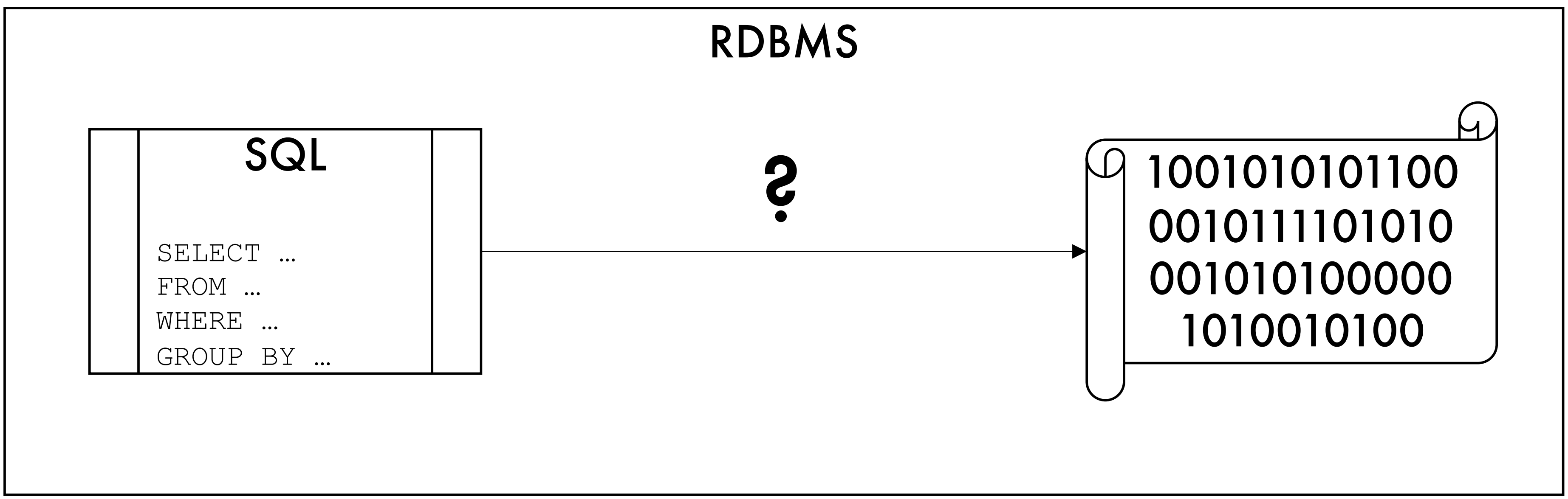
Physical Data Independence!

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.

Initially IBM didn't use his techniques…
10 years later he won the Turing award

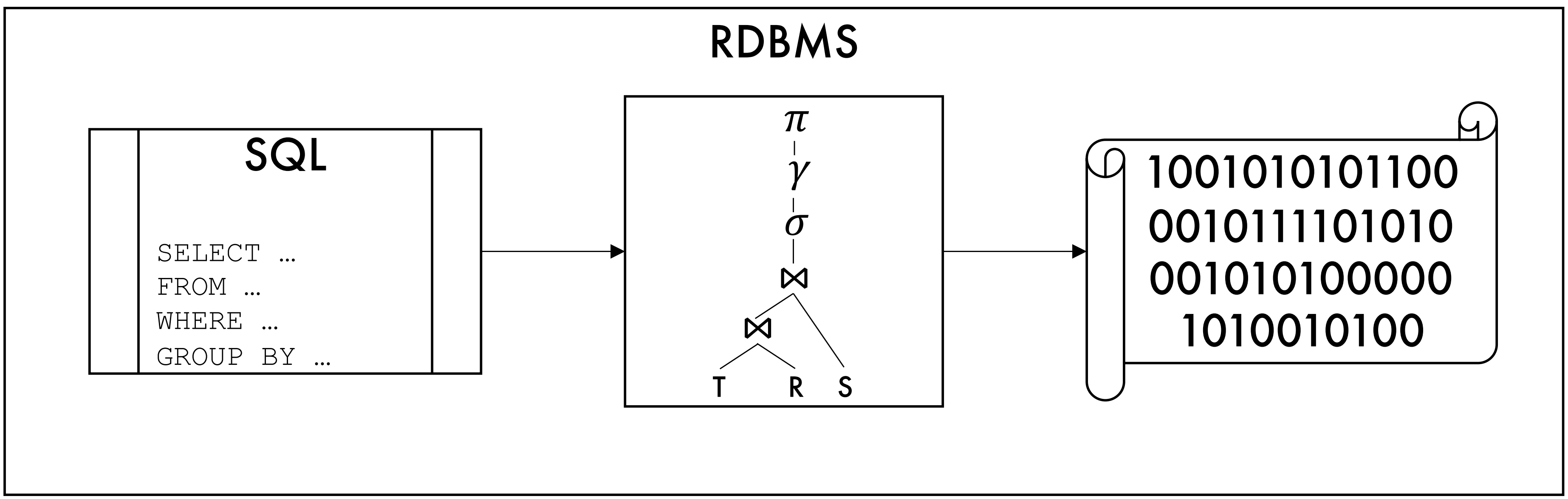
What's the Point of RA?

- 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)



RA Operators

- Read RA tree from bottom to top
  - Bottom → Data sources
  - Top → Query output
- Semantics
  - Every operator takes 1 or 2 relations as inputs
  - Every operator outputs a relation as an output

RA Operators

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

$\bowtie$	Inner 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

SQL

How does a computer understand abstract SQL text?

Database Internals

- 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';
```

Database Internals

- 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

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

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

$$Payroll\ P$$

Database Internals

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

For-each semantics

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

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

$$Payroll\ P$$

Database Internals

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

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

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

$$Payroll\ P$$

For-each semantics

```
for each row in P:
  if (row.Job == 'TA'):
    output (row.Name, row.UserID)
```

Database Internals

- 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 RA tree getting filtered and modified

RA Operators

- Get ready for some examples…

RA Operators

π Projection

- Unary operator
- Projection removes unspecified columns
- Happens in SQL "SELECT" clause

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

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
- Happens in SQL WHERE and HAVING clause

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

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
- Happens in JOIN clause (or explicit WHERE)

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

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

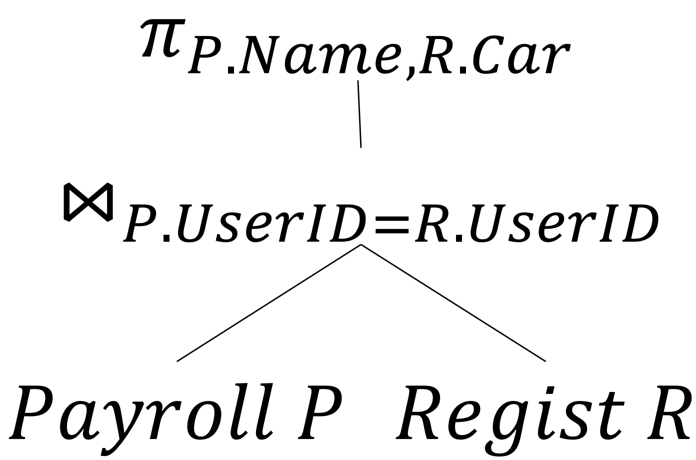
$$T(A,B) \times S(C,D) \rightarrow R(A,B,C,D)$$

RA Equivalencies

So far we haven't discussed equivalent RA trees.
But all joins can be parsed directly into a "join tree"

RA Equivalencies

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



RA Equivalencies

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

Join
Combine tuples on the provided predicate

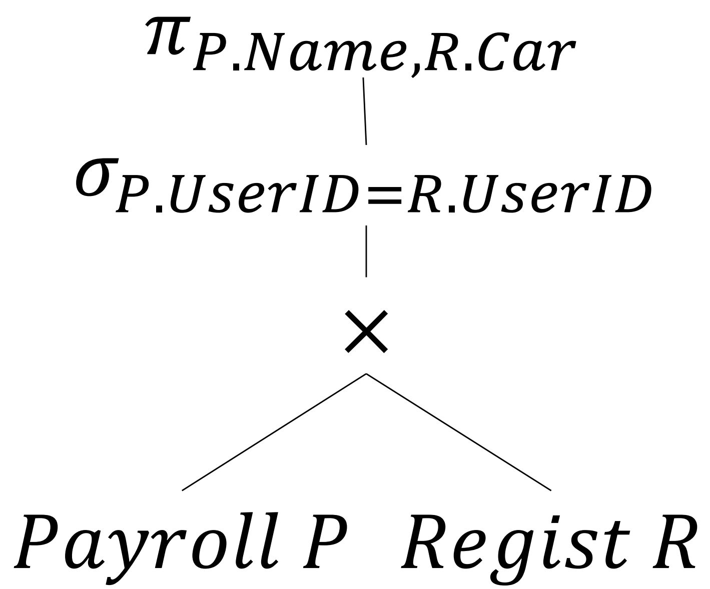
$$\pi_{P.Name, R.Car}$$

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

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

RA Equivalencies

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



RA Equivalencies

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

$$\pi_{P.Name,R.Car}$$

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

$$\times$$

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

Cross Product
Same intuition from set theory

More RA Operators

Select σ, Project π, Join $\bowtie$ are most common

We also have set operators

RA Operators

$\cup$ Union $\cap$ Intersection

- Binary operators
- Same semantics as in set theory (but over bags)

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

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

“Extended Relational Algebra”

Original relational algebra and data model were on sets

We clearly need operators based on bags:

“extended RA”

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

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

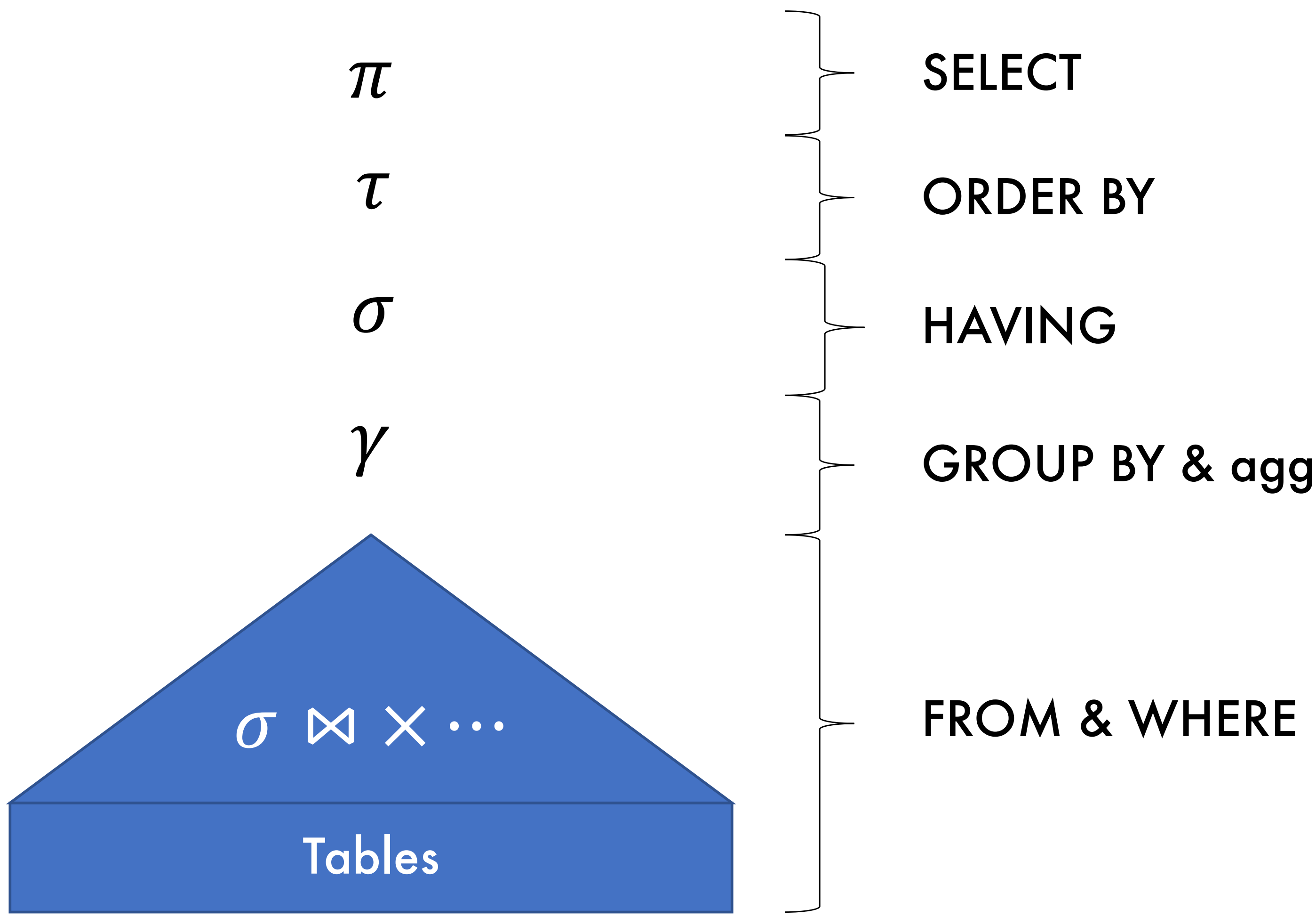
$$\delta\big(T(A,B,C)\big) \rightarrow R(A,B,C)$$

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

A	B	C
1	2	3
4	5	6

Basic SQL to RA Conversion

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



SQL and RA Vocab Summary

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

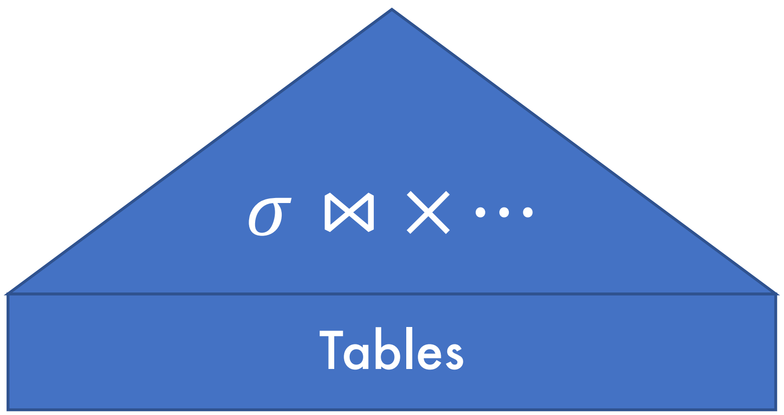
δ

π

τ

σ

γ



SQL and RA Vocab Summary

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

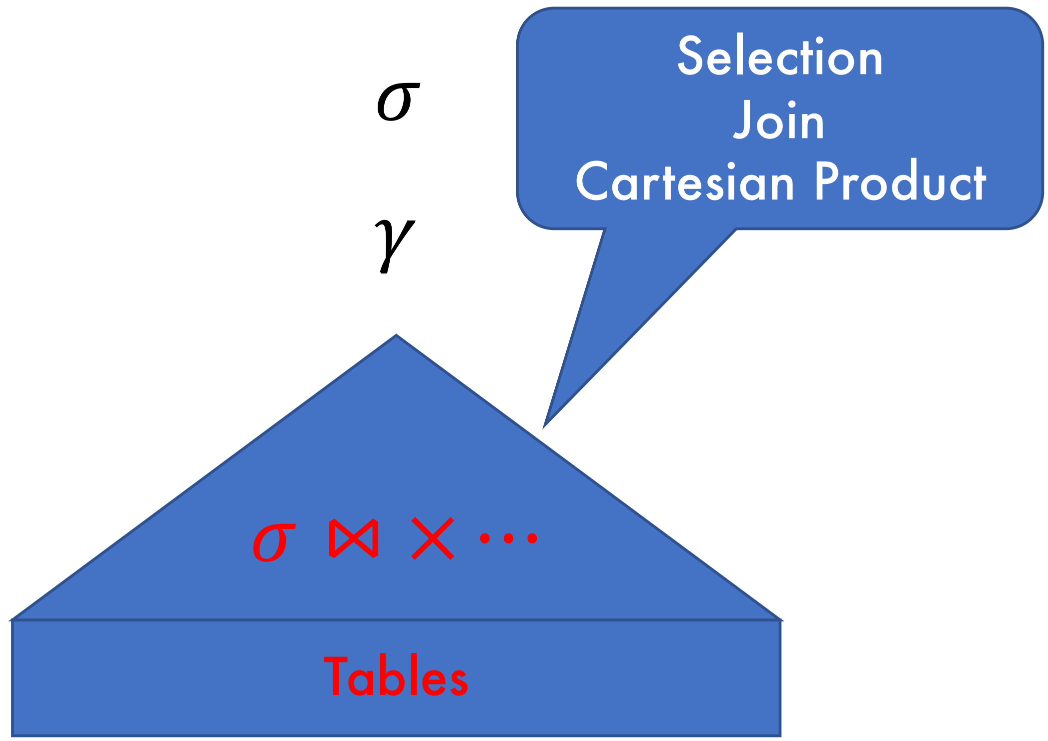
δ

π

τ

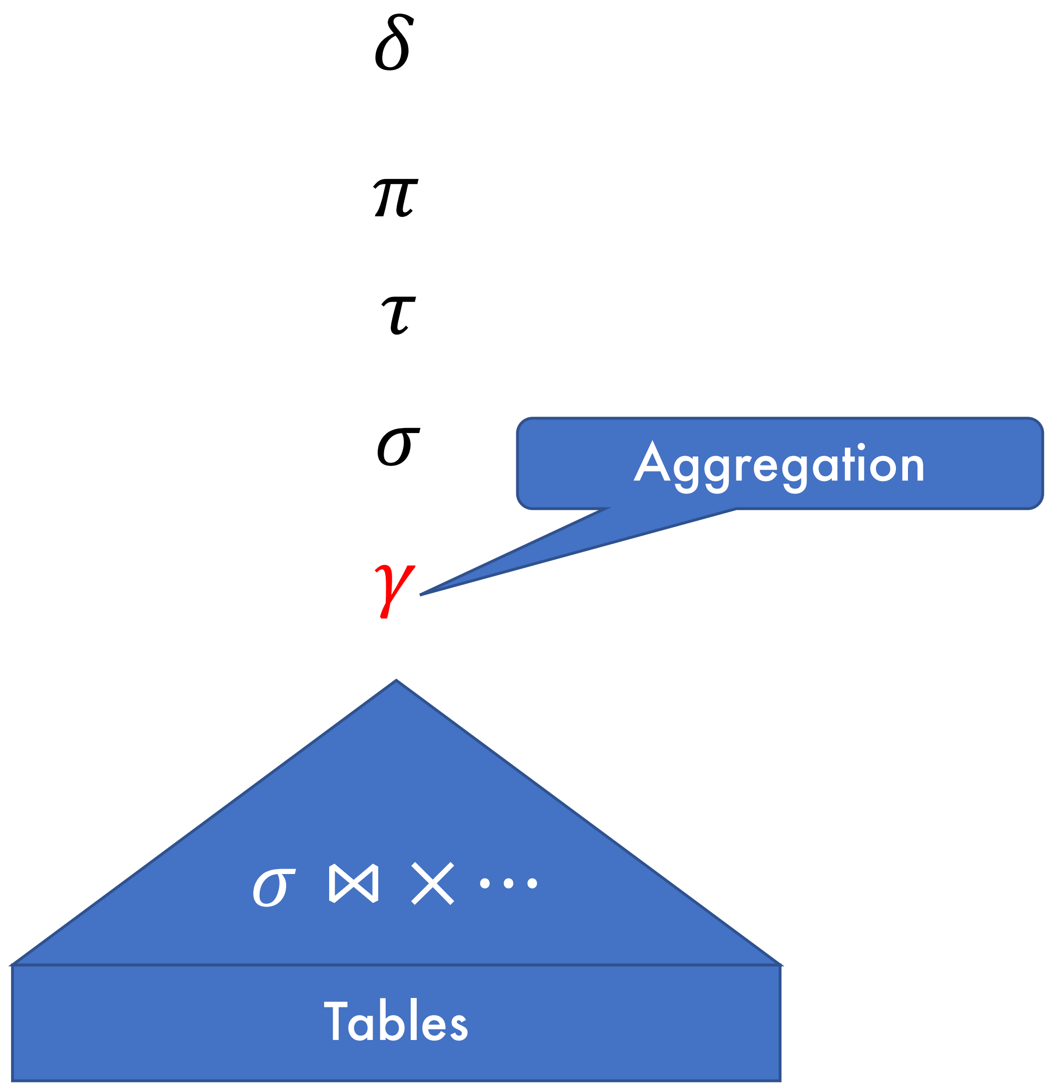
σ

γ



SQL and RA Vocab Summary

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



SQL and RA Vocab Summary

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

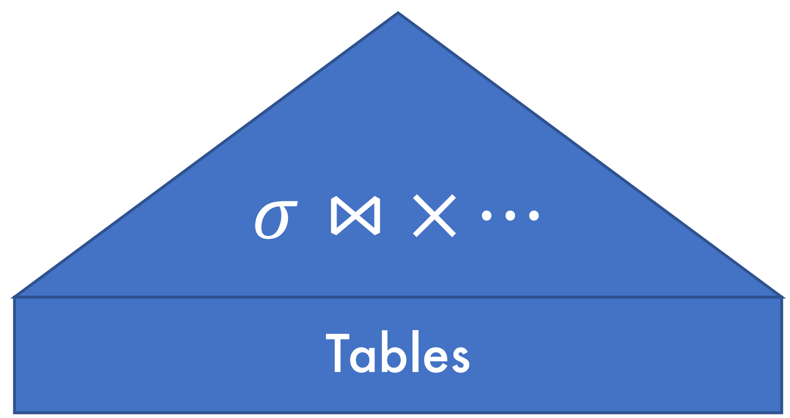
δ

π

τ

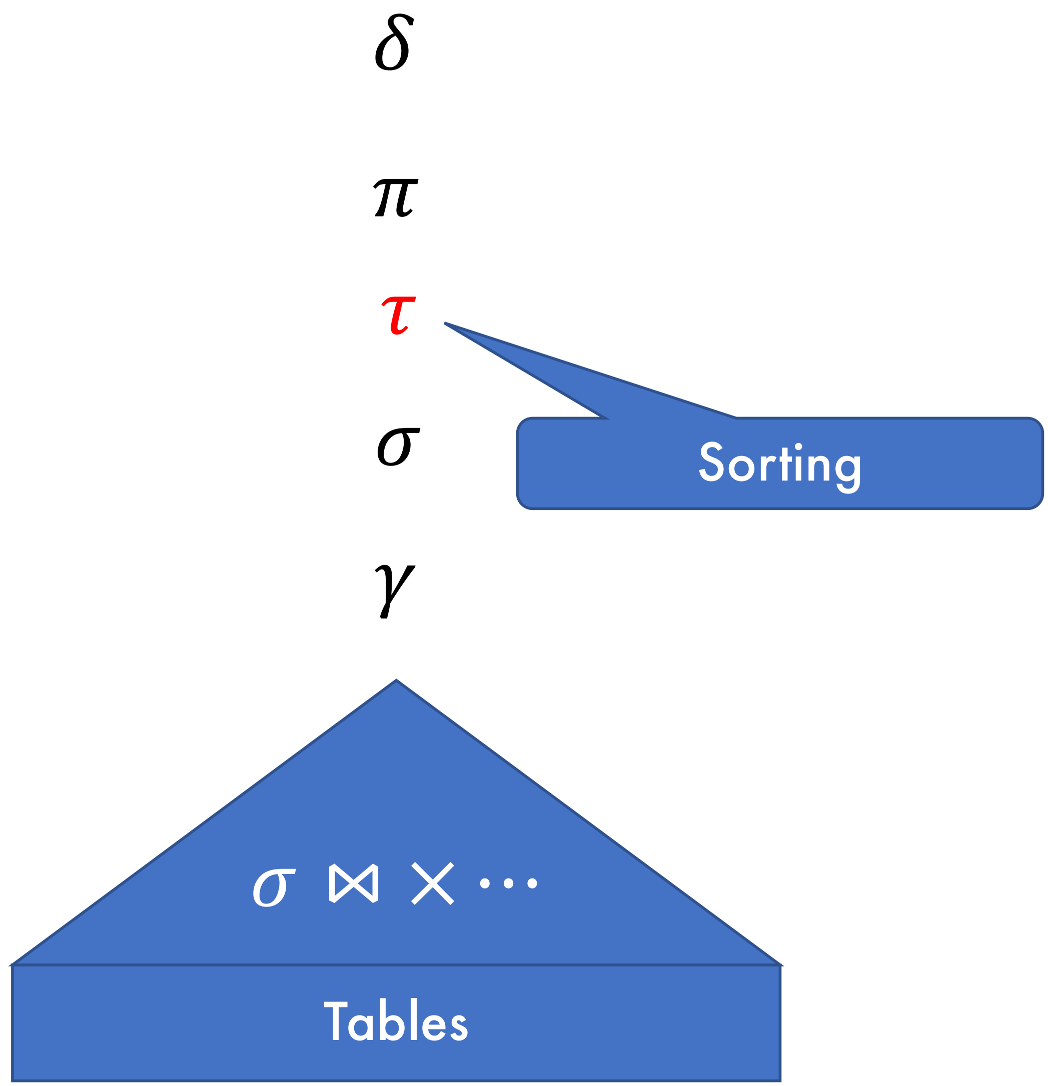
σ Selection

γ



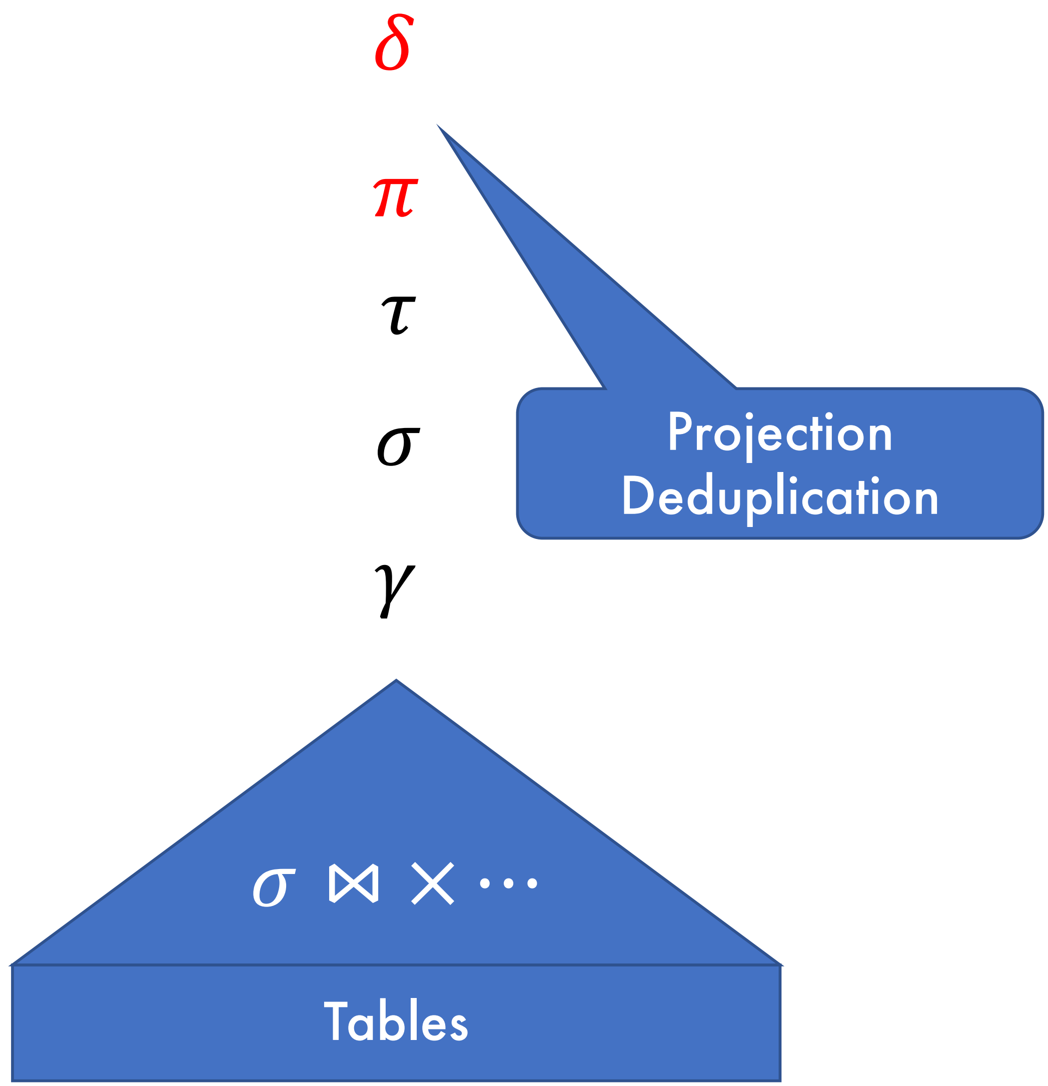
SQL and RA Vocab Summary

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



SQL and RA Vocab Summary

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



SQL and RA Vocab Summary

FWGHOS™

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

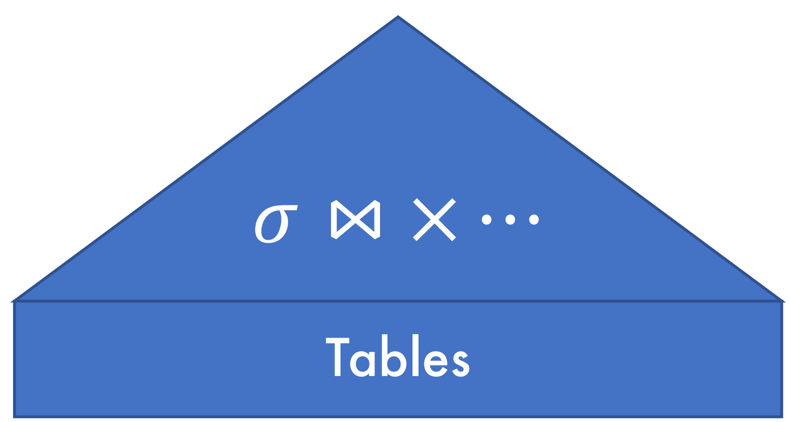
δ

π

τ

σ

γ



SQL and RA Vocab Summary

FWGHOS™

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

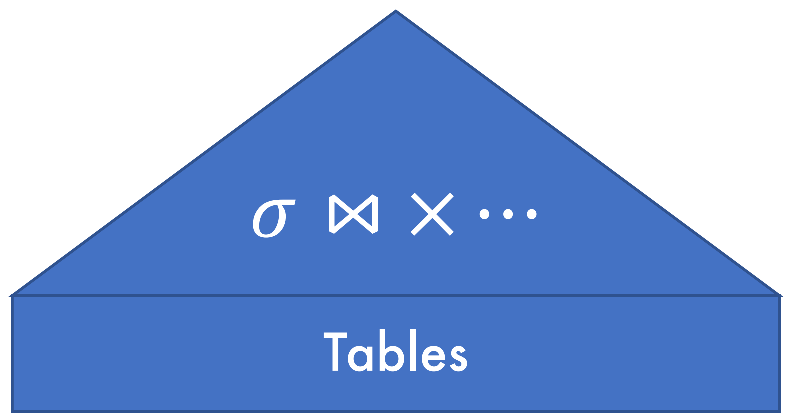
δ

π

τ

σ

γ



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(*)
```

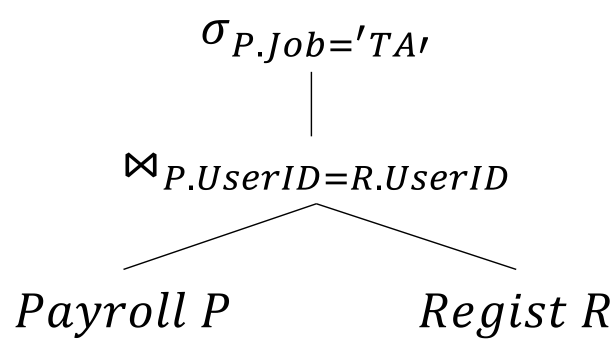
English to SQL to RA Example

```
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(*)
```

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

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

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

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

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(*)
```

$$\tau_{cnt}$$

$$\sigma_{cnt>1}$$

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

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

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

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

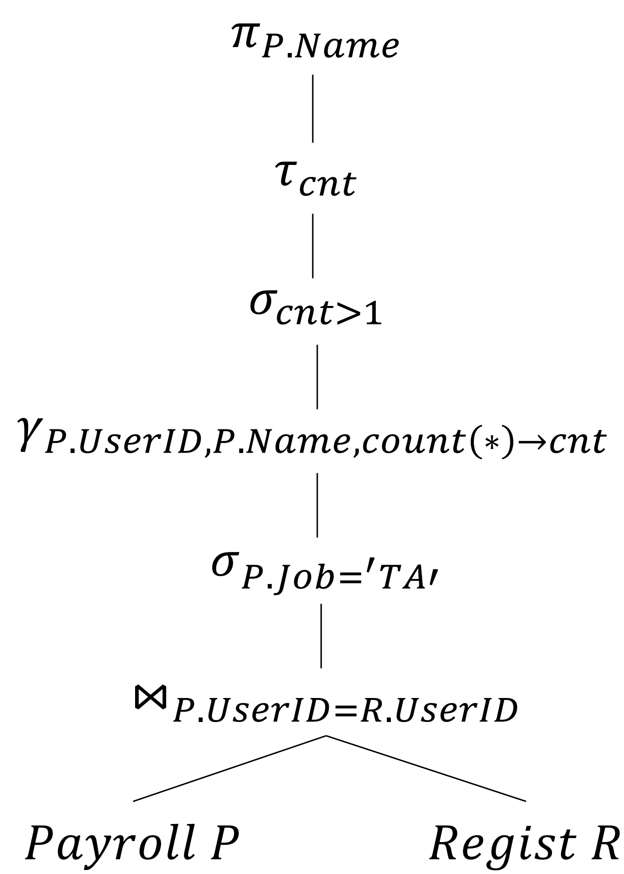
English to SQL to RA Example

```
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(*)
```



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(*)
```

δ

$\pi_{P.Name}$

τ_{cnt}

$\sigma_{cnt>1}$

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

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

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

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

Outlook

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

Summary of RA and SQL

- 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

RDBMS translate SQL → RA, then optimize RA