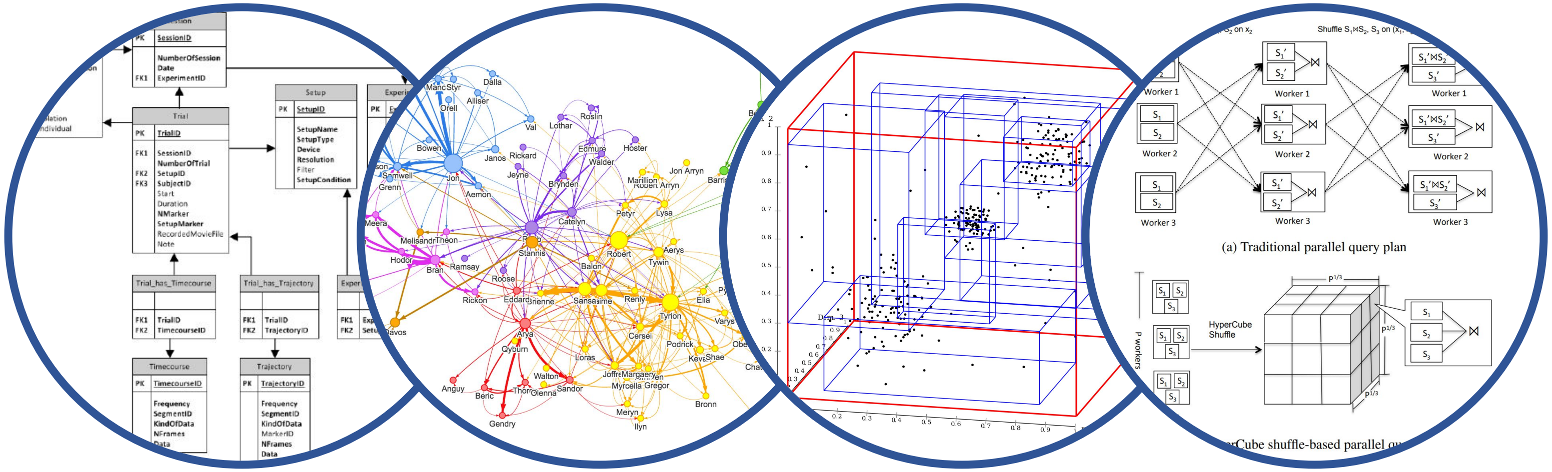


Introduction to Data Management

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

Announcements

- Holiday Monday, no class/OH/minimal Piazza
- HW2 due Tuesday
- HW3 will be released then
  - Will announce on Piazza
  - Accept your Azure credits! Email from invites@microsoft.com
  - TAs will walk through Azure set up next section

Recap – Nested Queries

A subquery is a SQL query nested inside a larger query

A subquery may occur in:

- A SELECT clause
- A FROM clause
- A WHERE or HAVING clause

Rule of thumb:

Avoid nested queries when possible…

…but sometimes it's impossible

Recap - Subqueries in SELECT

- Must return a single value
- Uses:
  - Compute an associated value

```
SELECT P.Name, (SELECT AVG(P1.Salary)
                FROM Payroll AS P1
                WHERE P.Job = P1.Job)
  FROM Payroll AS P
```

Correlated subquery! Semantics are that the entire subquery is recomputed for each tuple

Recap - Unnesting

For each person find the number of cars they drive

```
SELECT P.Name, (SELECT COUNT(R.Car)
                  FROM Regist AS R
                 WHERE P.UserID =
                       R.UserID)
  FROM Payroll AS P
```

Still possible to decorrelate and unnest

```
SELECT P.Name, COUNT(R.Car)
  FROM Payroll AS P LEFT OUTER JOIN
       Regist AS R ON P.UserID = R.UserID
 GROUP BY P.Name
```

Recap - Subqueries in FROM

- Uses:
  - Solve subproblems that can be later joined/evaluated

```
SELECT P.Name, P.Salary
  FROM Payroll AS P,
       (SELECT P1.Job AS Job,
               MAX(P1.Salary) AS Salary
         FROM Payroll AS P1
         GROUP BY P1.Job) AS Pmax
 WHERE P.Job = Pmax.Job AND
       P.Salary = Pmax.Salary
```

Recap - Subqueries in WHERE/HAVING

- Can return a relation
- Uses:
  - Use with an existential or universal quantifier
    - (NOT) EXISTS, (NOT) IN, ANY, ALL

Ex: Find all people who drive some car made before 2017.

```
SELECT P.Name
  FROM Payroll AS P
 WHERE EXISTS (SELECT *
               FROM Regist R
               WHERE R.UserID = P.UserID
                     AND R.Year < 2017)
```

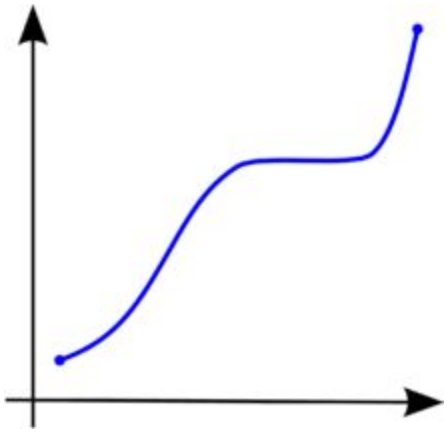
Recap - Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.



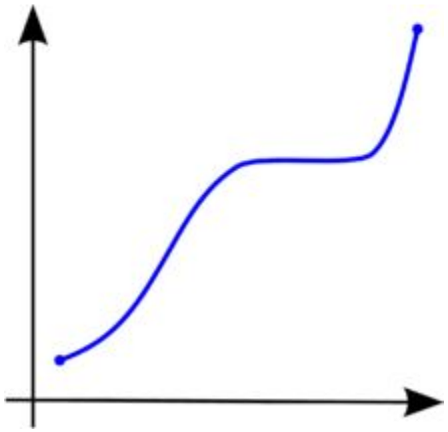
Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.



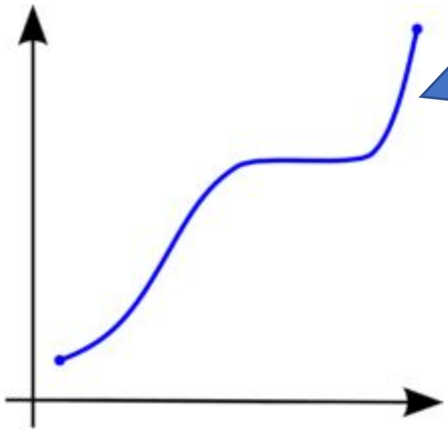
Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.



Monotone queries can be similar to monotonically increasing functions when considering cardinalities of results

Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.

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

Is this query monotone?

Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.

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

Is this query monotone? Yes!

Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.

I can't add tuples to Payroll or Regist that would "remove" a previous result

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

Is this query monotone? Yes!

Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.

```
SELECT P.Name
  FROM Payroll AS P
 WHERE P.Salary >= ALL (SELECT Salary
                          FROM Payroll)
```

Is this query monotone?

Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.

```
SELECT P.Name
  FROM Payroll AS P
 WHERE P.Salary >= ALL (SELECT Salary
                          FROM Payroll)
```

Is this query monotone? No!

Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.

```
SELECT P.Name
  FROM Payroll AS P
 WHERE P.Salary >= ALL (SELECT Salary
                          FROM Payroll)
```

I can add a tuple to Payroll that has a higher salary value than any other

Is this query monotone? No!

Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.

```
SELECT P.Job, COUNT(*)
  FROM Payroll AS P
 GROUP BY P.Job
```

Is this query monotone?

Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over that superset must contain at least the query results of I.

```
SELECT P.Job, COUNT(*)
  FROM Payroll AS P
 GROUP BY P.Job
```

Is this query monotone? No!

Monotonicity

Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

$$I \subseteq J \rightarrow q(I) \subseteq q(J)$$

That is for any superset of I, the query over tho
at least the query results of I.

Aggregates generally are sensitive to any new tuples since the aggregate value will change

```
SELECT P.Job, COUNT(*)
  FROM Payroll AS P
 GROUP BY P.Job
```

Is this query monotone? No!

Monotonicity

Theorem:

If Q is a SELECT-FROM-WHERE query that does not have subqueries or aggregates, then it is monotone.

Monotonicity

Theorem:

If Q is a SELECT-FROM-WHERE query that does not have subqueries or aggregates, then it is monotone.

Proof:

We use nested loop semantics. If we insert a tuple in relation R, this will not remove any tuples from the answer.

```
SELECT a1, a2, …, ak
FROM R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE Conditions
```

```
for x1 in R1 do
 for x2 in R2 do
  …
   for xn in Rn do
    if Conditions
     output (a1,…,ak)
```

Monotonicity

Theorem:

The query “Find all people who drive only cars older than 2017” is not monotone.

Proof:

We use example. For user 123 who previously only drove a car made in 2009, we add another car made in 2018. Now user 123 does not appear in the results. Thus, the query is not monotone.

Monotonicity

Theorem:

The query “Find all p… opl… ho drive only cars older than …017” is … ton…

Proof:

We use e… …o previously only … …e add another … …ow user 123 does not ap… …s. Thus, the query is not monoto…

If a query is not monotonic, then we can’t write it as a SELECT-FROM-WHERE query without subqueries

Queries That Cannot Be S-F-W

- Queries with universal quantifiers or negation

```
SELECT P.Name
  FROM Payroll AS P
 WHERE P.UserID NOT IN (SELECT R.UserID
                          FROM Regist AS R
                        WHERE R.Year < 2017)
```

```
SELECT P.Name
  FROM Payroll AS P
 WHERE P.Salary >= ALL (SELECT Salary
                          FROM Payroll)
```

Goals for Today

- We've completed SQL! Now we know how to write a query for any question the relational model can answer.
- Next we'll dive into another language for working with relational data - Relational Algebra.

Outline

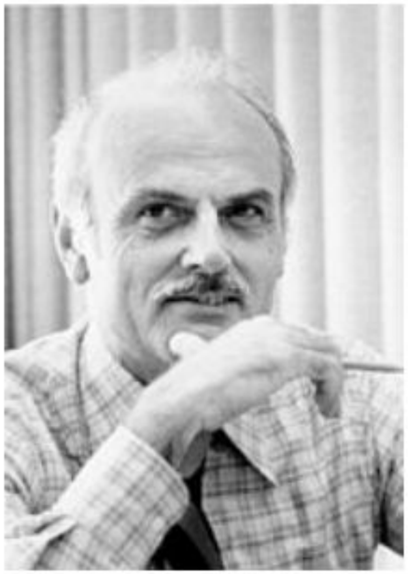
- What is Relational Algebra (RA)?
- Introduce RA operators
- See equivalent SQL and RA queries

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

Formalized and published by Ted Codd of IBM



Information Retrieval

A Relational Model of Data for Large Shared Data Banks

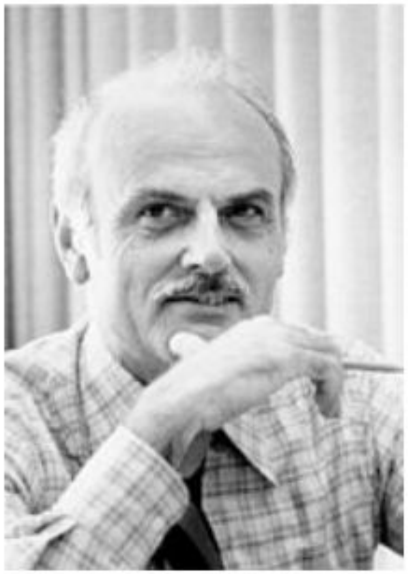
E. F. Codd
IBM Research Laboratory, San Jose, California

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 approach…
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

Physical data independence!

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.

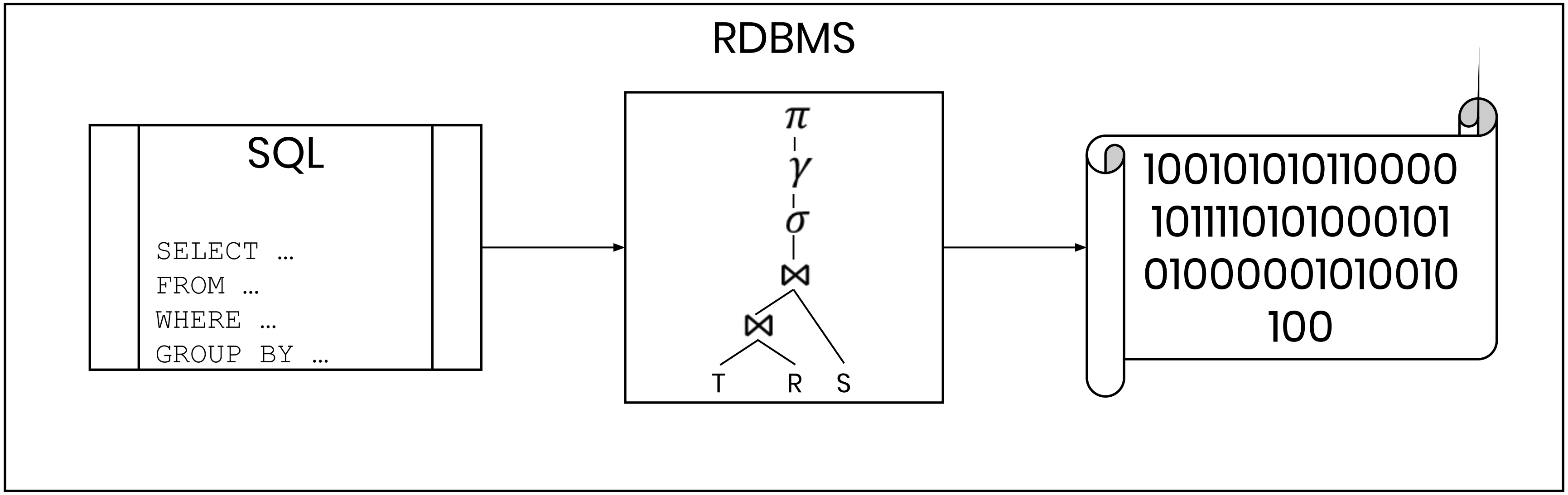
Initially IBM didn't use his approach…
10 years later he won the Turing Award

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

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

RA Operators

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

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

RA

Extended RA

RA Operators

- For the curious…

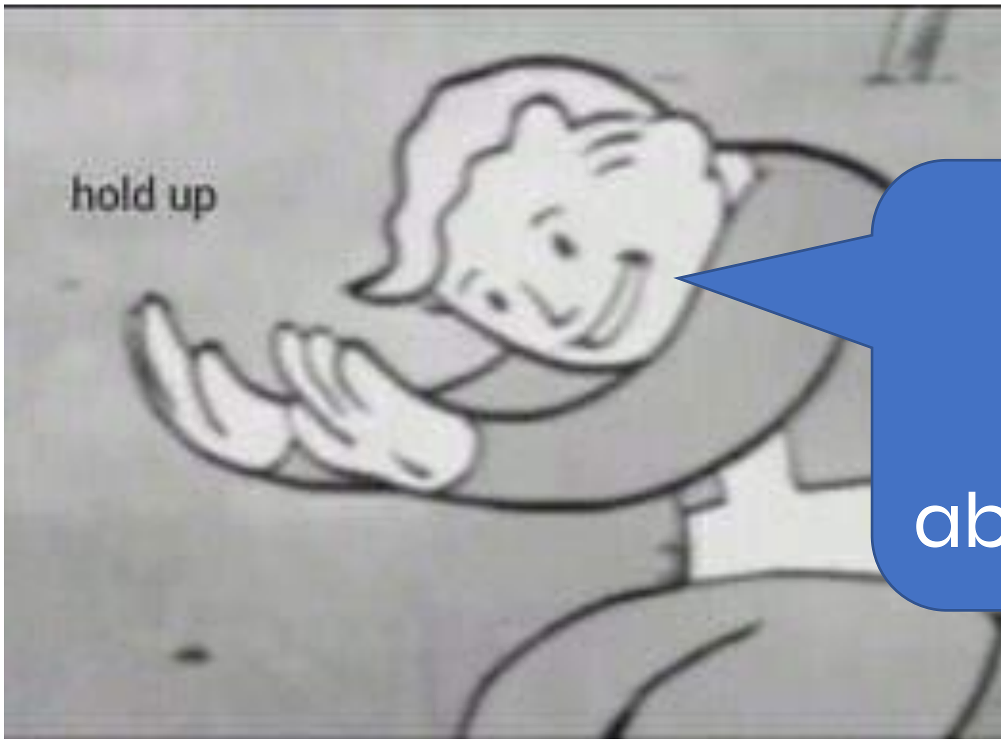
⟖ Right Outer Join

⟕ 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

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

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

$$Payroll\ P$$

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

For-each semantics

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

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

$$Payroll\ P$$

```
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).

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

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

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 the 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 returns tuples from the input which satisfy the condition (filtering)
- Happens in the SQL "WHERE" or "HAVING" clauses

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

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

A	B	C
1	2	3
4	5	6

RA Operators

σ Selection

- Unary operator
- Selection returns tuples from the input which satisfy the condition (filtering)
- Happens in the SQL "WHERE" or "HAVING" clauses

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

Can use =, <, <=, >, >=, <>
Combine with AND, OR, NOT

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 the SQL "JOIN" clause (or implicit joins using 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

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

A	B
1	2
3	4

C	D
2	3
5	6

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

RA Operators

$\times$ Cartesian Product

- Binary operator
- Same ser antics in t theory
- Indiscrimina elations

Rare in practice –
this is mainly used to
express joins.
Think our nested loop
semantics.

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

A	B
1	2
3	4

C	
2	3
5	6

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

RA Equivalencies

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

RA Equivalencies

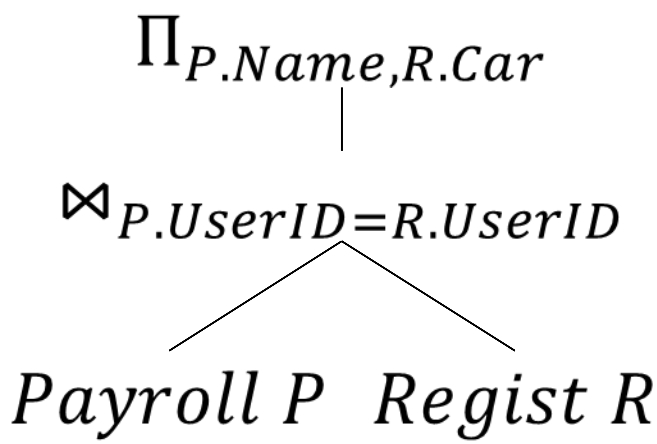
```
SELECT P.Name, R.Car
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID;
```

$$\Pi_{P.Name,R.Car}(\bowtie_{P.UserID=R.UserID}(Payroll\ P\ ,\ Regist\ R\ ))$$

RA Equivalencies

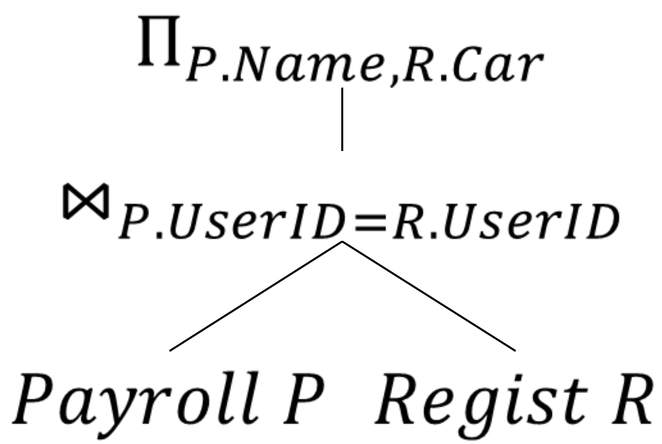
```sql
SELECT P.Name, R.Car
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID;
```

$$\Pi_{P.Name,R.Car}(\bowtie_{P.UserID=R.UserID}(Payroll\ P,\ Regist\ R))$$



RA Equivalencies

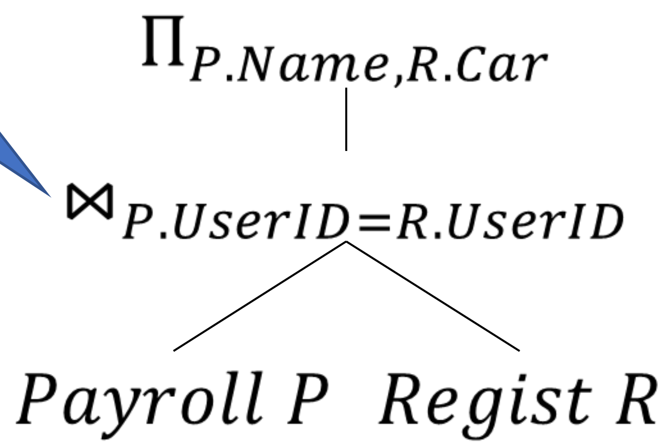
```sql
SELECT P.Name, R.Car
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID;
```



RA Equivalencies

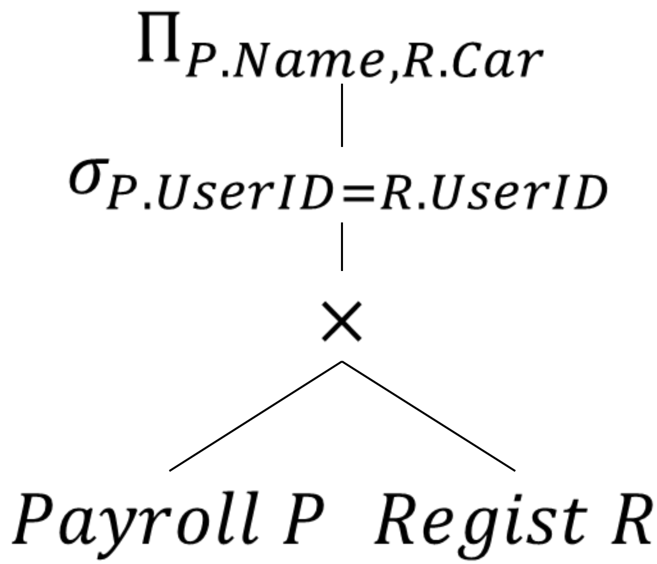
```
SELECT P.Name, R.Car
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID;
```

Join
Combine tuples on the provided predicate



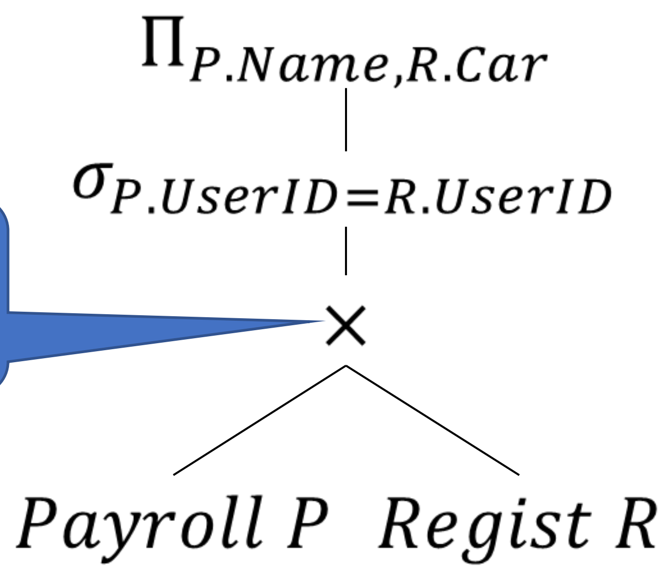
RA Equivalencies

```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;
```



Cross Product
Same intuition from set theory

Your Turn!

```
-- 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!

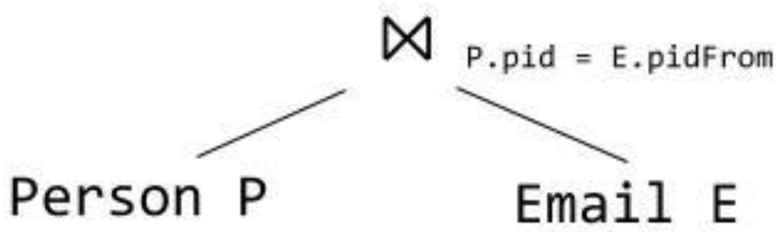
- A warm up
- Find the length of all emails Alice sent.

```
SELECT E.length
  FROM Person P, Email E
 WHERE P.pid = E.pidFrom AND
       P.name = ‘Alice’;
```

Your Turn!

- A warm up
- Find the length of all emails Alice sent.

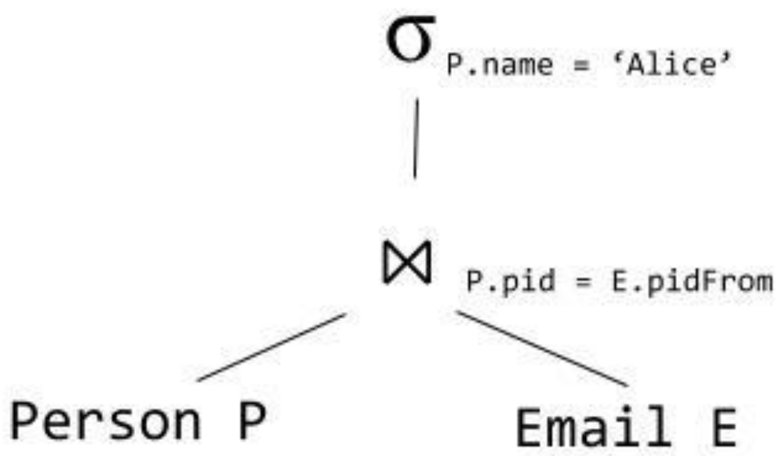
```
SELECT E.length
  FROM Person P, Email E
 WHERE P.pid = E.pidFrom AND
       P.name = 'Alice';
```



Your Turn!

- A warm up
- Find the length of all emails Alice sent.

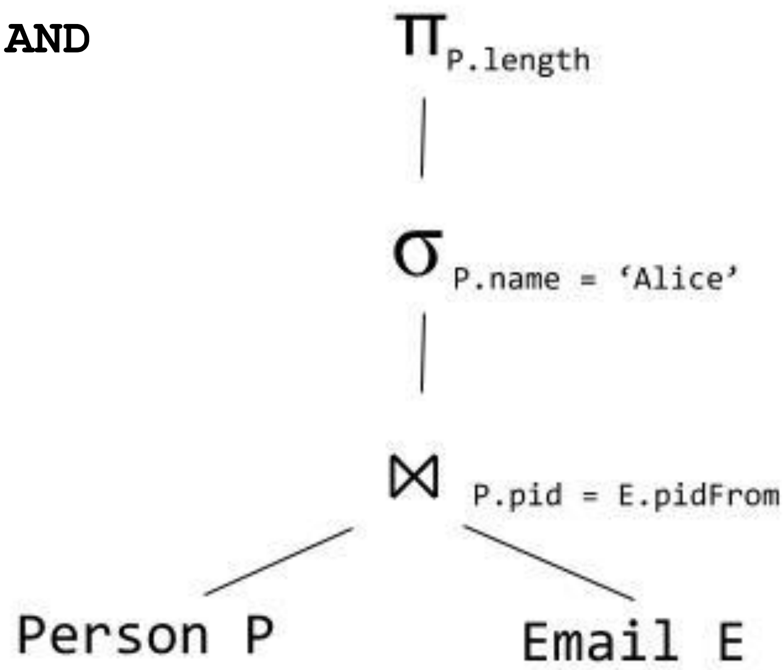
```
SELECT E.length
  FROM Person P, Email E
 WHERE P.pid = E.pidFrom AND
       P.name = 'Alice';
```



Your Turn!

- A warm up
- Find the length of all emails Alice sent.

```
SELECT E.length
  FROM Person P, Email E
 WHERE P.pid = E.pidFrom AND
       P.name = 'Alice';
```



Onto extended RA

Original relational algebra only worked with sets

We clearly need operators working with real-life relations: bags, ordering, grouping…

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

γ Grouping & Aggregation

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

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

attribute to group by

aggregate to compute

only these attributes will be passed "up" the tree

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
- Happens in SQL “ORDER BY” clause
- Assume default ascending order like in SQL

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

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
- Happens with SQL "DISTINCT" keyword
- 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

SQL and RA Vocab Summary

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

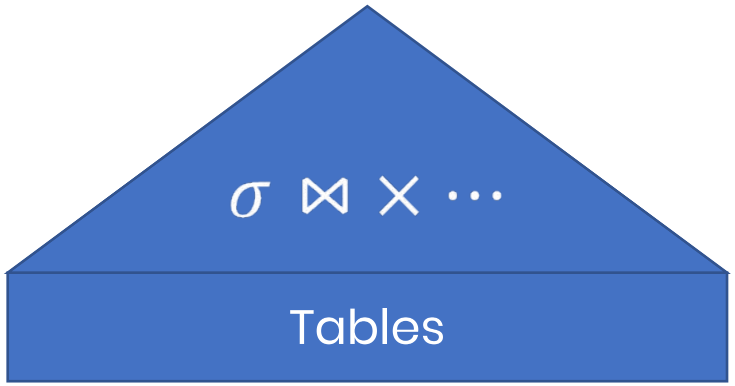
δ

Π

τ

σ

γ



SQL and RA Vocab Summary

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

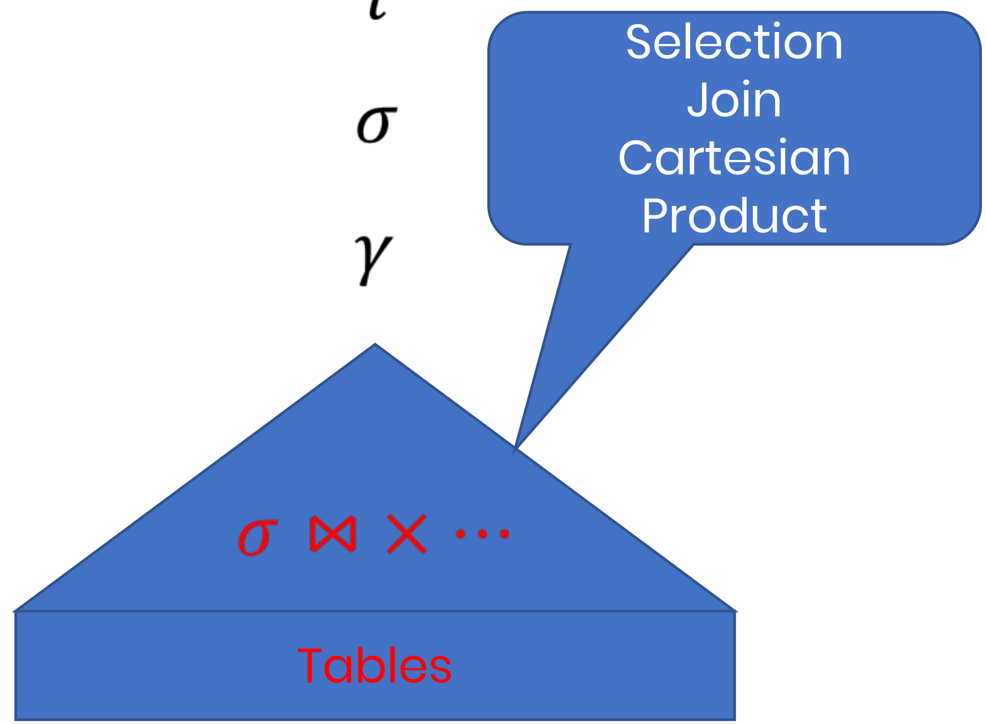
δ

Π

τ

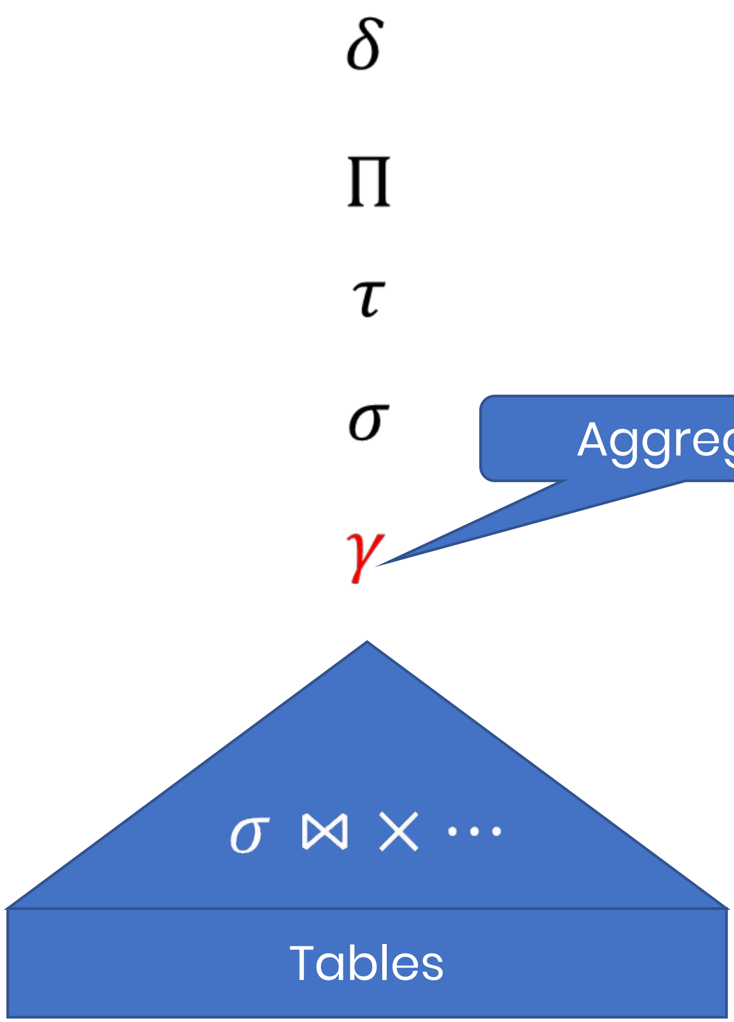
σ

γ



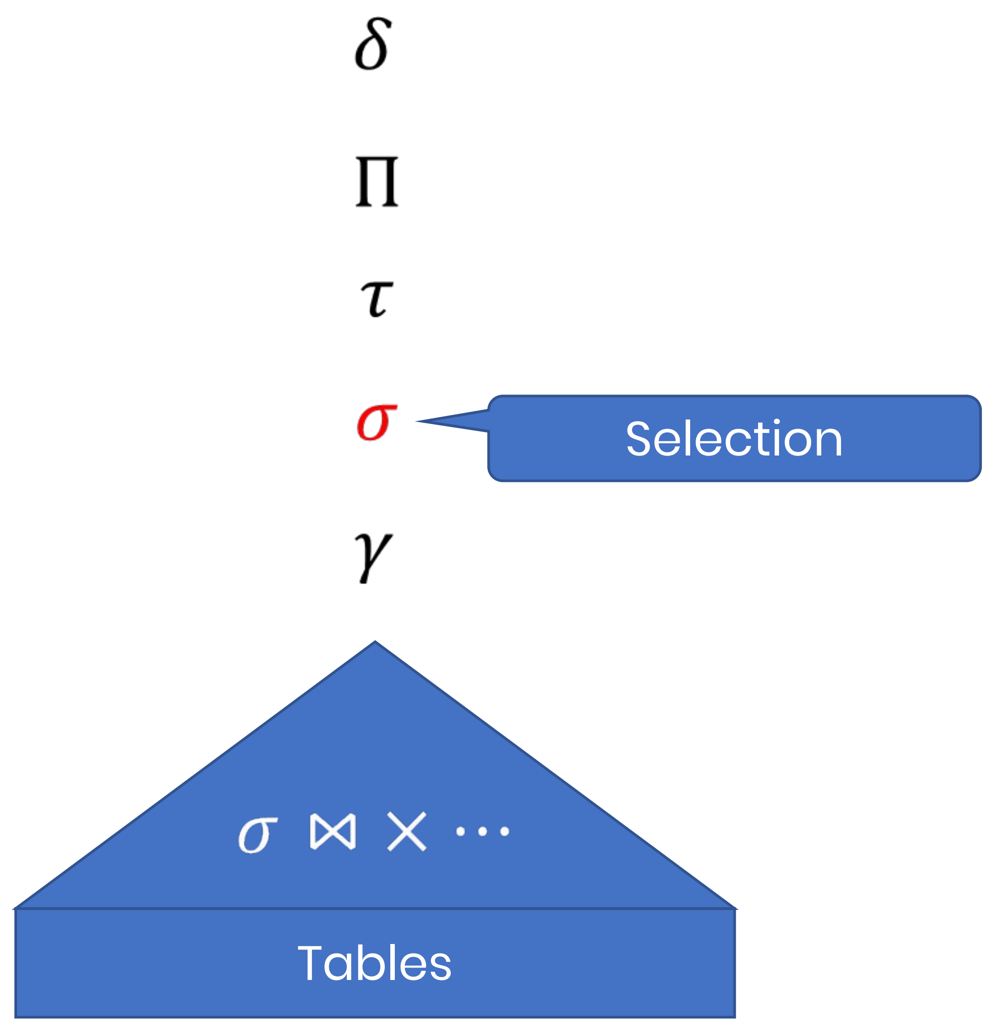
SQL and RA Vocab Summary

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



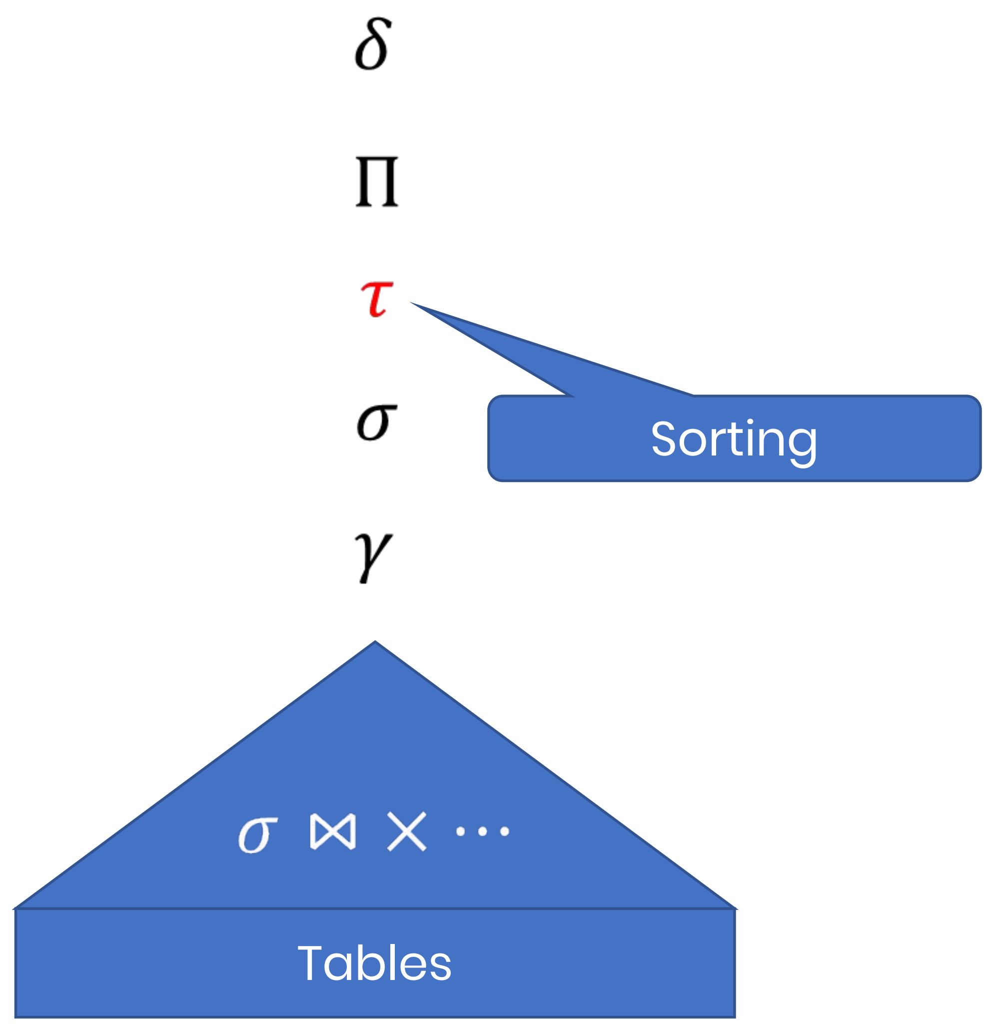
SQL and RA Vocab Summary

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



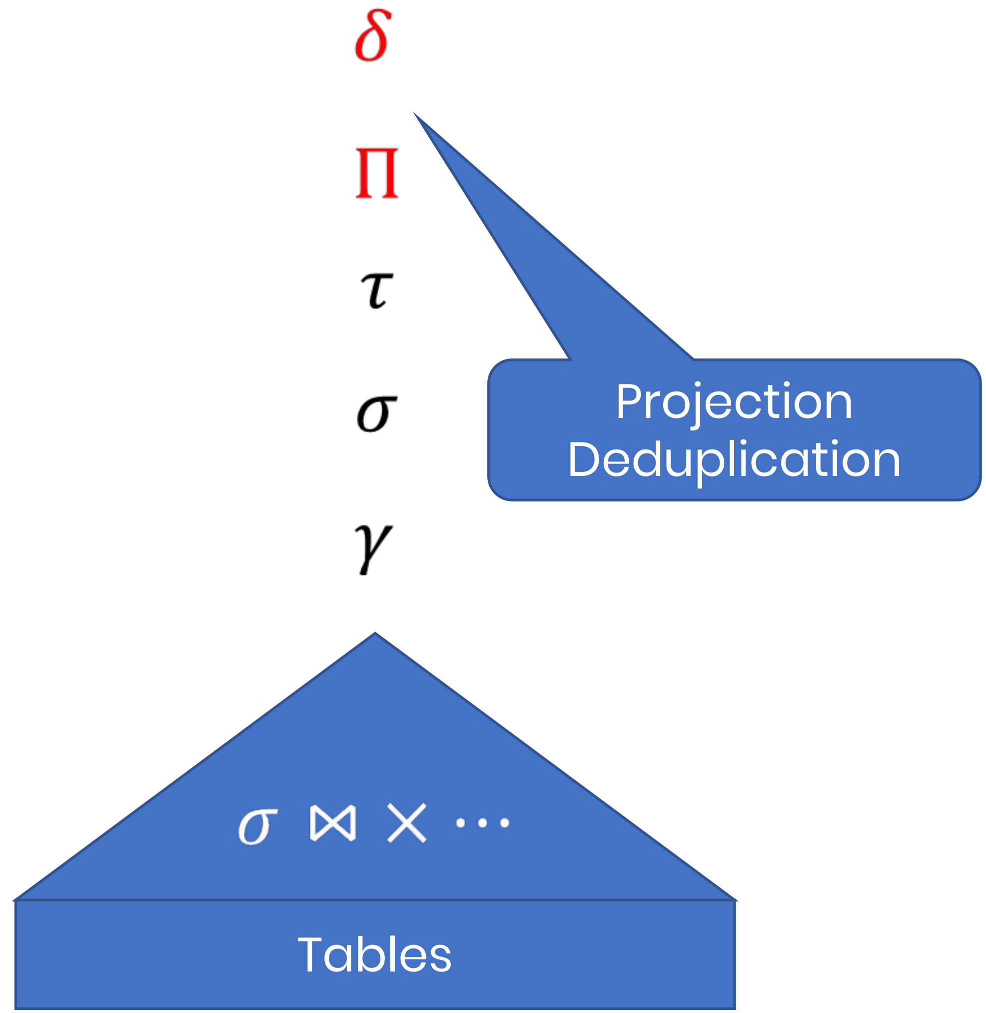
SQL and RA Vocab Summary

```
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 …
```

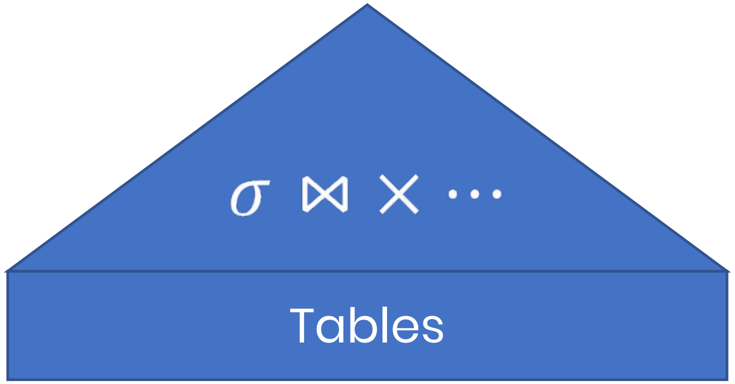
δ

Π

τ

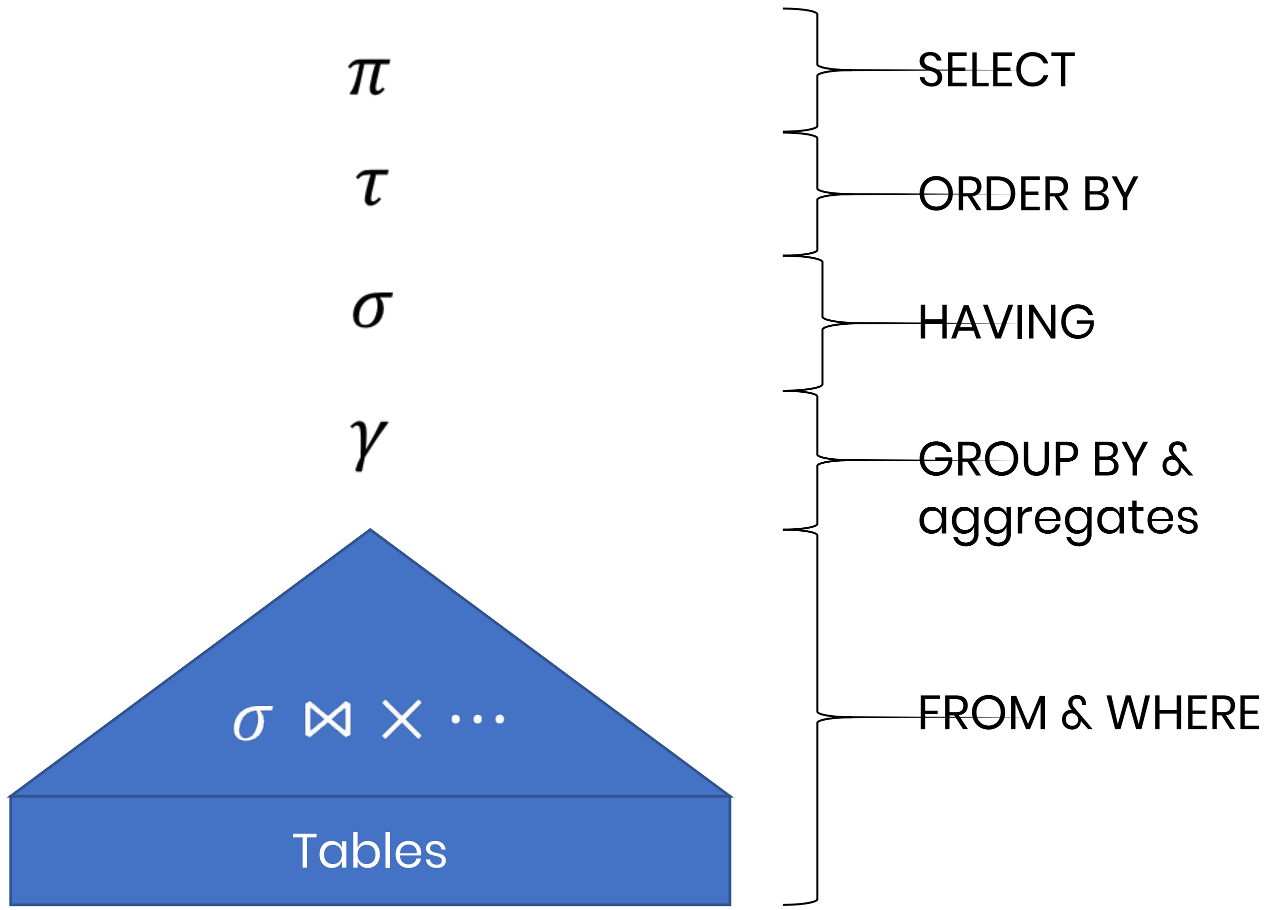
σ

γ



Basic SQL to RA Conversion

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



Aggregation RA

How is aggregation processed internally?

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```

Aggregation RA

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```

$$\gamma_{Job,\, MAX(P.Salary)\rightarrow maxSal,\, MIN(P.Salary)\rightarrow minSal}$$

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```

Job	maxSal	minSal
TA	60000	50000
Prof	100000	90000

$$\gamma_{Job,\, MAX(P.Salary) \to maxSal,\, MIN(P.Salary) \to minSal}$$

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```

$$\sigma_{minSal>80000}$$

Job	maxSal	minSal
TA	60000	50000
Prof	100000	90000

$$\gamma_{Job,\, MAX(P.Salary)\rightarrow maxSal,\, MIN(P.Salary)\rightarrow minSal}$$

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```

Job	maxSal	minSal
Prof	100000	90000

$$\sigma_{minSal>80000}$$

Job	maxSal	minSal
TA	60000	50000
Prof	100000	90000

$$\gamma_{Job,\, MAX(P.Salary)\rightarrow maxSal,\, MIN(P.Salary)\rightarrow minSal}$$

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```

$$\Pi_{Job,\, maxSal}$$

Job	maxSal	minSal
Prof	100000	90000

$$\sigma_{minSal > 80000}$$

Job	maxSal	minSal
TA	60000	50000
Prof	100000	90000

$$\gamma_{Job,\, MAX(P.Salary) \rightarrow maxSal,\, MIN(P.Salary) \rightarrow minSal}$$

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```

Job	maxSal
Prof	100000

$$\Pi_{Job,\, maxSal}$$

Job	maxSal	minSal
Prof	100000	90000

$$\sigma_{minSal > 80000}$$

Job	maxSal	minSal
TA	60000	50000
Prof	100000	90000

$$\gamma_{Job,\, MAX(P.Salary) \rightarrow maxSal,\, MIN(P.Salary) \rightarrow minSal}$$

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```

$$\Pi_{Job,\, maxSal}$$

$$\sigma_{minSal > 80000}$$

$$\gamma_{Job,\, MAX(P.Salary) \rightarrow maxSal,\, MIN(P.Salary) \rightarrow minSal}$$

$$Payroll\ P$$

Aggregation RA

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
HAVING MIN(Salary) > 80000
```

$$\Pi_{Job,\, maxSal}$$

Selection
HAVING uses the same symbol and operation used by WHERE clause

$$\sigma_{minSal > 80000}$$

$$\gamma_{Job,\, MAX(P.Salary) \rightarrow maxSal,\, MIN(P.Salary) \rightarrow minSal}$$

$$Payroll\ P$$

Your Turn!

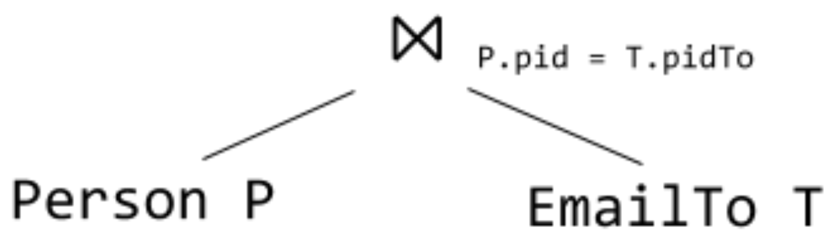
- An extended problem
- Find the number of emails that each person has received.

```
SELECT P.name, COUNT(*)
  FROM Person P, EmailTo T
 WHERE P.pid = T.pidTo
 GROUP BY P.pid, P.name;
```

Your Turn!

- An extended problem
- Find the number of emails that each person has received.

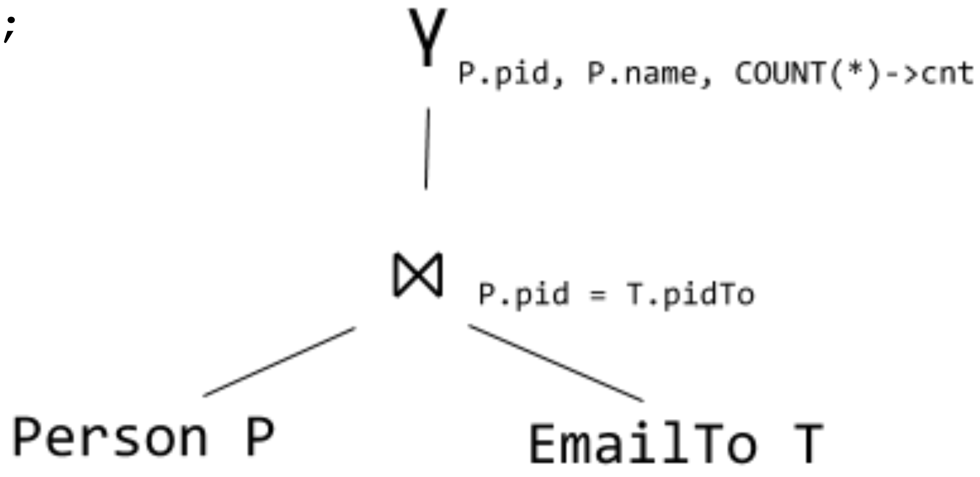
```
SELECT P.name, COUNT(*)
  FROM Person P, EmailTo T
 WHERE P.pid = T.pidTo
 GROUP BY P.pid, P.name;
```



Your Turn!

- An extended problem
- Find the number of emails that each person has received.

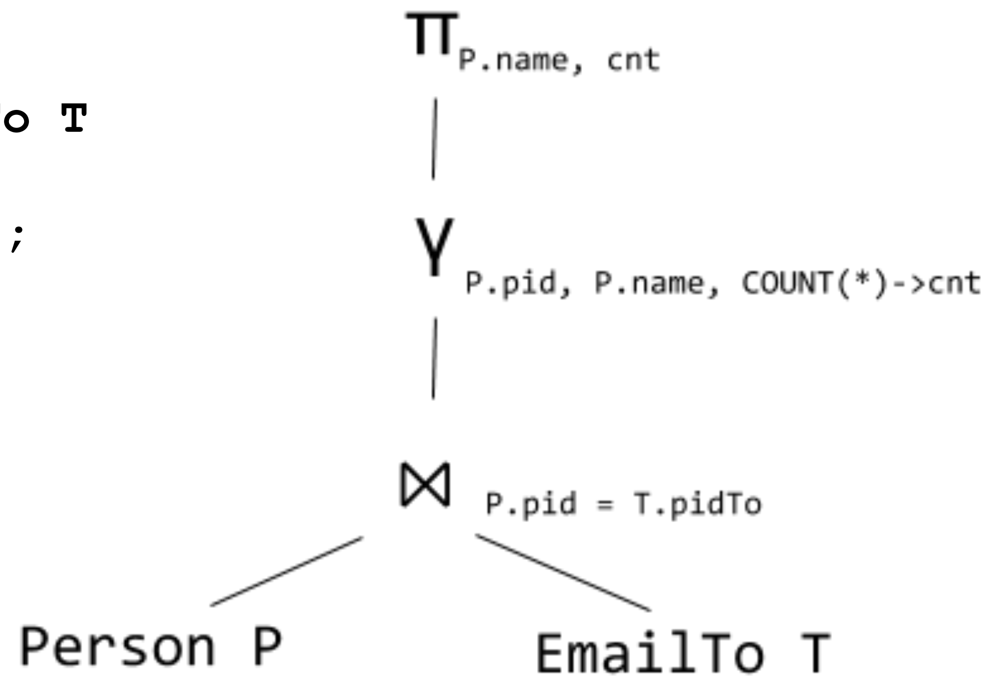
```
SELECT P.name, COUNT(*)
  FROM Person P, EmailTo T
 WHERE P.pid = T.pidTo
 GROUP BY P.pid, P.name;
```



Your Turn!

- An extended problem
- Find the number of emails that each person has received.

```
SELECT P.name, COUNT(*)
  FROM Person P, EmailTo T
 WHERE P.pid = T.pidTo
 GROUP BY P.pid, P.name;
```



Takeaways

- Relational Algebra has operators that can express everything we can express in SQL
- We can convert SQL to equivalent RA trees