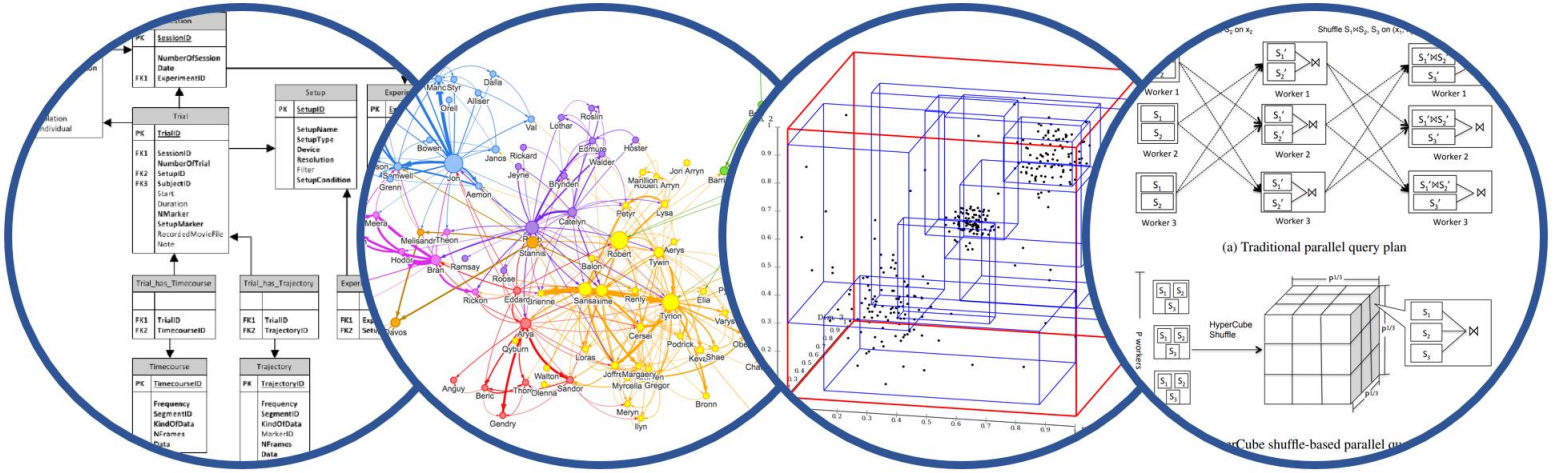


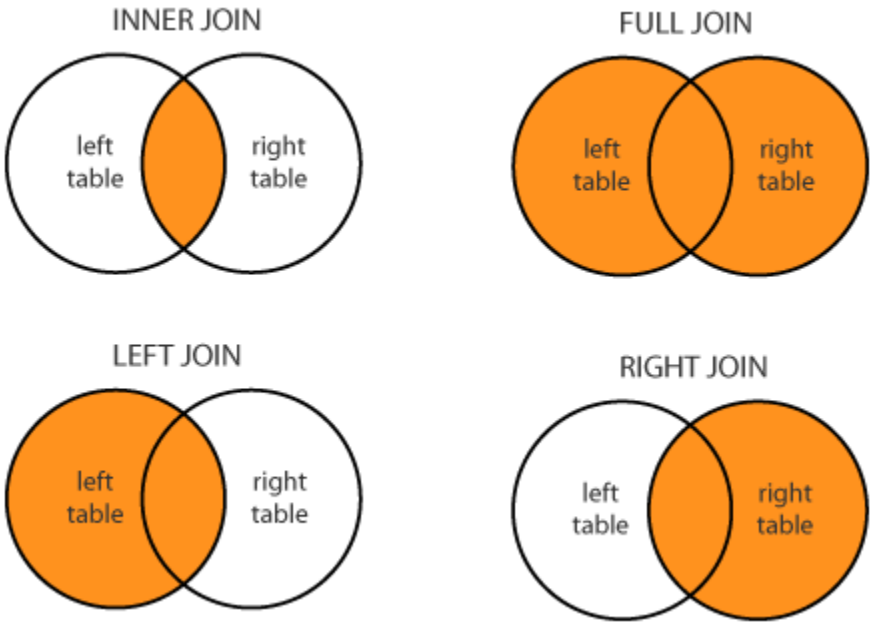
Introduction to Data Management

Aggregates and Grouping

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

Recap – Joins

- Join to combine data from different tables
  - Nested-loop semantics
  - Filtered cross product semantics
  - Inner join (the most common)
  - Outer joins can preserve information



https://www.dofactory.com/sql/join

Recap – Inner Joins

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Explicit

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

Implicit

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

Recap – Inner Joins

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

- What if we have no join predicate?

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

for each row1 in Payroll:
   for each row2 in Regist:
      output (row1.Name, row2.Car)
```

- Output every possible pair: ”Cross product”

Outer Joins

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Name	Car
Jack	Charger
Allison	NULL
Magda	Civic
Magda	Pinto
Dan	NULL

NULL is a value placeholder. Depending on context, it may mean unknown, not applicable, etc.

Outer Joins

- LEFT OUTER JOIN
  - All rows in left table are preserved
- RIGHT OUTER JOIN
  - All rows in right table are preserved
- FULL OUTER JOIN
  - All rows are preserved

Self Joins

Find all people who drive a Civic and Pinto

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

UserID	Car
123	Charger
567	Civic
567	Pinto

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

Self Joins

Find all people who drive a Civic and Pinto

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

UserID	Car
123	Charger
567	Civic
567	Pinto

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

Will this work?

Self Joins

Find all people who drive a Civic and Pinto

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

UserID	Car
123	Charger
567	Civic
567	Pinto

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

Will this work?

Nope, empty set is returned

Self Joins

Find all people who drive a Civic and Pinto

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

UserID	Car
123	Charger
567	Civic
567	Pinto
789	Pinto

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

Discuss with the people around you how you would solve this.

Self Joins

Find all people who drive a Civic and Pinto

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

UserID	Car
123	Charger
567	Civic
567	Pinto

```
SELECT P.Name, R1.Car
  FROM Payroll AS P, Regist AS R1, Regist AS R2
 WHERE P.UserID = R1.UserID AND
       P.UserID = R2.UserID AND
       R1.Car = 'Civic' AND
       R2.Car = 'Pinto';
```

Self Joins

Find all people who drive a Civic and Pinto

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

UserID	Car
123	Charger
567	Civic
567	Pinto

All pairs of cars a person can drive

```
SELECT P.Name, R1.Car
  FROM Payroll AS P, Regist AS R1, Regist AS R2
 WHERE P.UserID = R1.UserID AND
       P.UserID = R2.UserID AND
       R1.Car = 'Civic' AND
       R2.Car = 'Pinto';
```

Goals for Today

- We have started to build our SQL toolbox
  - Not just reading and filtering data anymore
  - Starting to answer complex questions
- Today we want to effectively summarize results

Aggregation functions

- New class of SQL queries

Aggregates

Outline

- Aggregation functions
- GROUP BY and HAVING clauses in SQL
- The witnessing problem

Actionable Results

- We need summaries of data because we are often trying to **make decisions** and **succinctly convey information**

Actionable Results

- We need summaries of data because we are often trying to **make decisions** and **succinctly convey information**
  - “How popular is this tv-show?”

Actionable Results

- We need summaries of data because we are often trying to **make decisions** and **succinctly convey information**
  - “How popular is this tv-show?” → COUNT

Actionable Results

- We need summaries of data because we are often trying to **make decisions** and **succinctly convey information**
  - “How popular is this tv-show?” → COUNT
  - “Do I spend too much on coffee?”

Actionable Results

- We need summaries of data because we are often trying to **make decisions** and **succinctly convey information**
  - “How popular is this tv-show?” → COUNT
  - “Do I spend too much on coffee?” → SUM

Actionable Results

- We need summaries of data because we are often trying to **make decisions** and **succinctly convey information**
  - “How popular is this tv-show?” → COUNT
  - “Do I spend too much on coffee?” → SUM
  - “Am I being ripped off by this dealer?”

Actionable Results

- We need summaries of data because we are often trying to **make decisions** and **succinctly convey information**
  - “How popular is this tv-show?” → COUNT
  - “Do I spend too much on coffee?” → SUM
  - “Am I being ripped off by this dealer?” → AVG

Actionable Results

- We need summaries of data because we are often trying to **make decisions** and **succinctly convey information**
  - “How popular is this tv-show?” → COUNT
  - “Do I spend too much on coffee?” → SUM
  - “Am I being ripped off by this dealer?” → AVG
  - “Who got the highest grade in the class?”

Actionable Results

- We need summaries of data because we are often trying to **make decisions** and **succinctly convey information**
  - “How popular is this tv-show?” → COUNT
  - “Do I spend too much on coffee?” → SUM
  - “Am I being ripped off by this dealer?” → AVG
  - “Who got the highest grade in the class?” → MAX

Actionable Results

- We need summaries of data because we are often trying to **make decisions** and **succinctly convey information**
  - “How popular is this tv-show?” → COUNT
  - “Do I spend too much on coffee?” → SUM
  - “Am I being ripped off by this dealer?” → AVG
  - “Who got the highest grade in the class?” → MAX
  - “What’s the cheapest food on the Ave?”

Actionable Results

- We need summaries of data because we are often trying to **make decisions** and **succinctly convey information**
  - “How popular is this tv-show?” → COUNT
  - “Do I spend too much on coffee?” → SUM
  - “Am I being ripped off by this dealer?” → AVG
  - “Who got the highest grade in the class?” → MAX
  - “What’s the cheapest food on the Ave?” → MIN

Actionable Results

- We need summaries of data because we are often trying to **make decisions** and **succinctly convey information**
  - COUNT
  - SUM
  - AVG
  - MAX
  - MIN

Actionable Results

- We need summaries of data because we are often trying to **make decisions** and **succinctly convey information**
  - COUNT
  - SUM
  - AVG
  - MAX
  - MIN

Very common attributes found in DBMS

Actionable Results

- We need summaries of data because we are often trying to **make decisions** and **succinctly convey information**
  - SELECT **COUNT**(*) FROM StreamingViews …
  - SELECT **SUM**(cost) FROM CoffeeReceipts …
  - SELECT **AVG**(price) FROM CarDealers …
  - SELECT **MAX**(score) FROM StudentGrades …
  - SELECT **MIN**(price) FROM AveLunchPrices …

AGG(attr) → computes **AGG** over non-NULL values
AGG(DISTINCT attr) is also possible

Actionable Results

- We need summaries of data because we are often trying to **make decisions** and **succinctly convey information**
  - SELECT **COUNT**(*) FROM StreamingViews …
  - SELECT **SUM**(cost) FROM CoffeeReceipts …
  - SELECT **AVG**(price) FROM CarDealers …
  - SELECT **MAX**(score) FROM StudentGrades …
  - SELECT **MIN**(price) FROM AveLunchPrices …

COUNT(*) → # of rows regardless of NULL

Aggregation Semantics

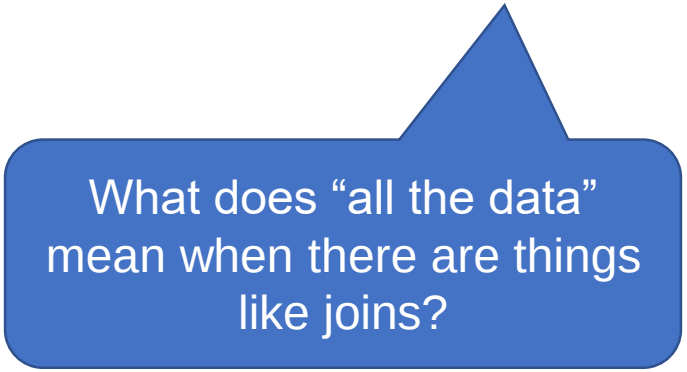
What am I aggregating over in a SELECT-FROM-WHERE query?

Intuitively: “all the data”

Aggregation Semantics

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

Intuitively: “all the data”



Aggregation Semantics

Will this query get me the correct calculation for average salary of all people who own cars?

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

Payroll

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

Regist

UserID	Car
123	Charger
567	Civic
567	Pinto

Aggregation Semantics

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

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

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Aggregation Semantics

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

P.UserID	P.Name	P.Job	P.Salary	R.UserID	R.Car
123	Jack	TA	50000	123	Charger
567	Magda	Prof	90000	567	Civic
567	Magda	Prof	90000	567	Pinto

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

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Aggregation Semantics

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

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

P.UserID	P.Name	P.Job	P.Salary	R.UserID	R.Car
123	Jack	TA	50000	123	Charger
567	Magda	Prof	90000	567	Civic
567	Magda	Prof	90000	567	Pinto

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

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Aggregation Semantics

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

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

P.UserID	P.Name	P.Job	P.Salary	R.UserID	R.Car
123	Jack	TA	50000	123	Charger
567	Magda	Prof	90000	567	Civic
567	Magda	Prof	90000	567	Pinto

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

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Aggregation Semantics

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

AVG(P.Salary)
76666

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

P.UserID	P.Name	P.Job	P.Salary	R.UserID	R.Car
123	Jack	TA	50000	123	Charger
567	Magda	Prof	90000	567	Civic
567	Magda	Prof	90000	567	Pinto

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

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Aggregation Semantics

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

AVG(P.Salary)
76666

90000 was counted twice…

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

P.UserID	P.Name	P.Job	P.Salary	R.UserID	R.Car
123	Jack	TA	50000	123	Charger
567	Magda	Prof	90000	567	Civic
567	Magda	Prof	90000	567	Pinto

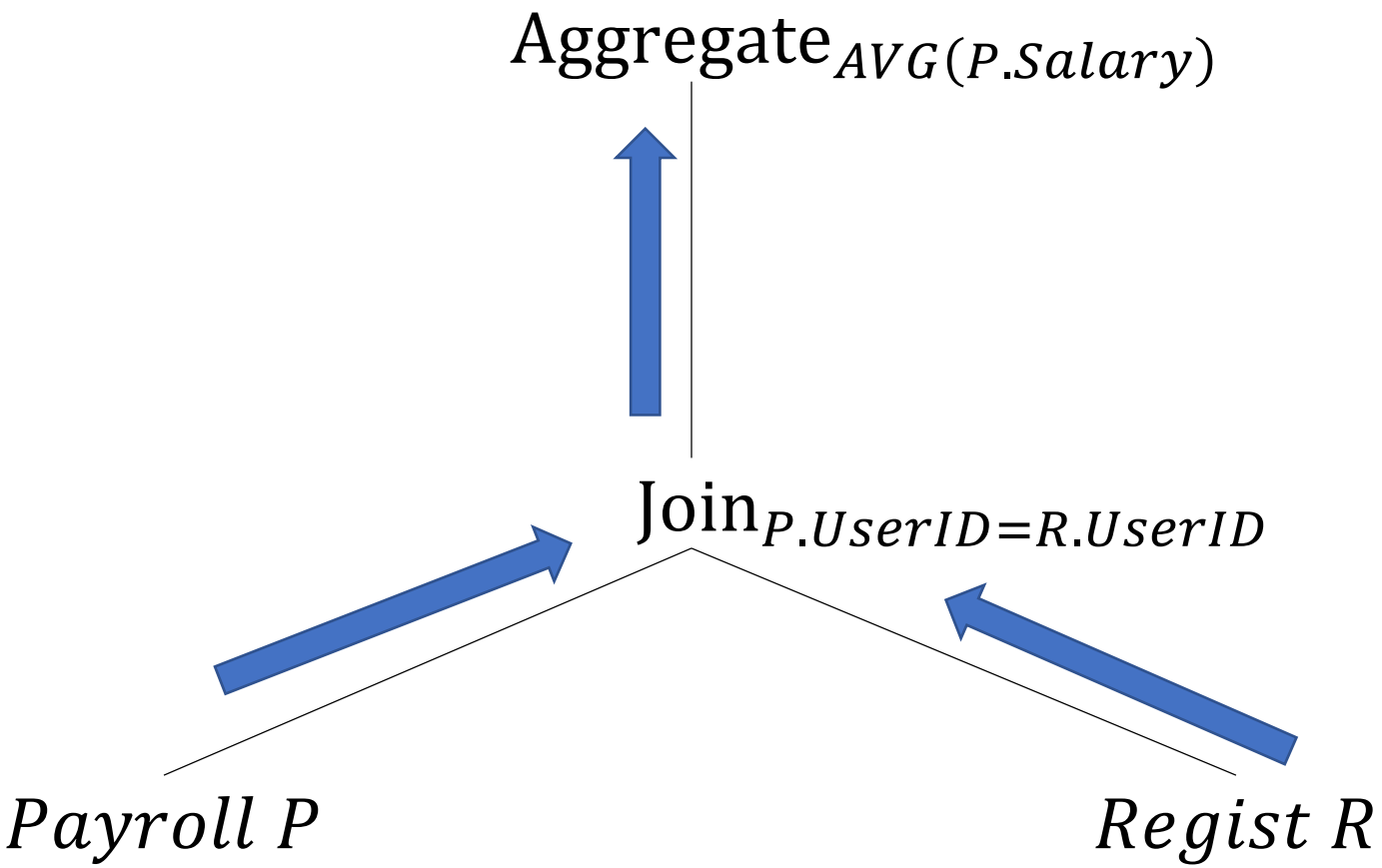
$$\text{Join}_{P.UserID=R.UserID}$$

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

UserID	Car
123	Charger
567	Civic
567	Pinto

Aggregation Semantics

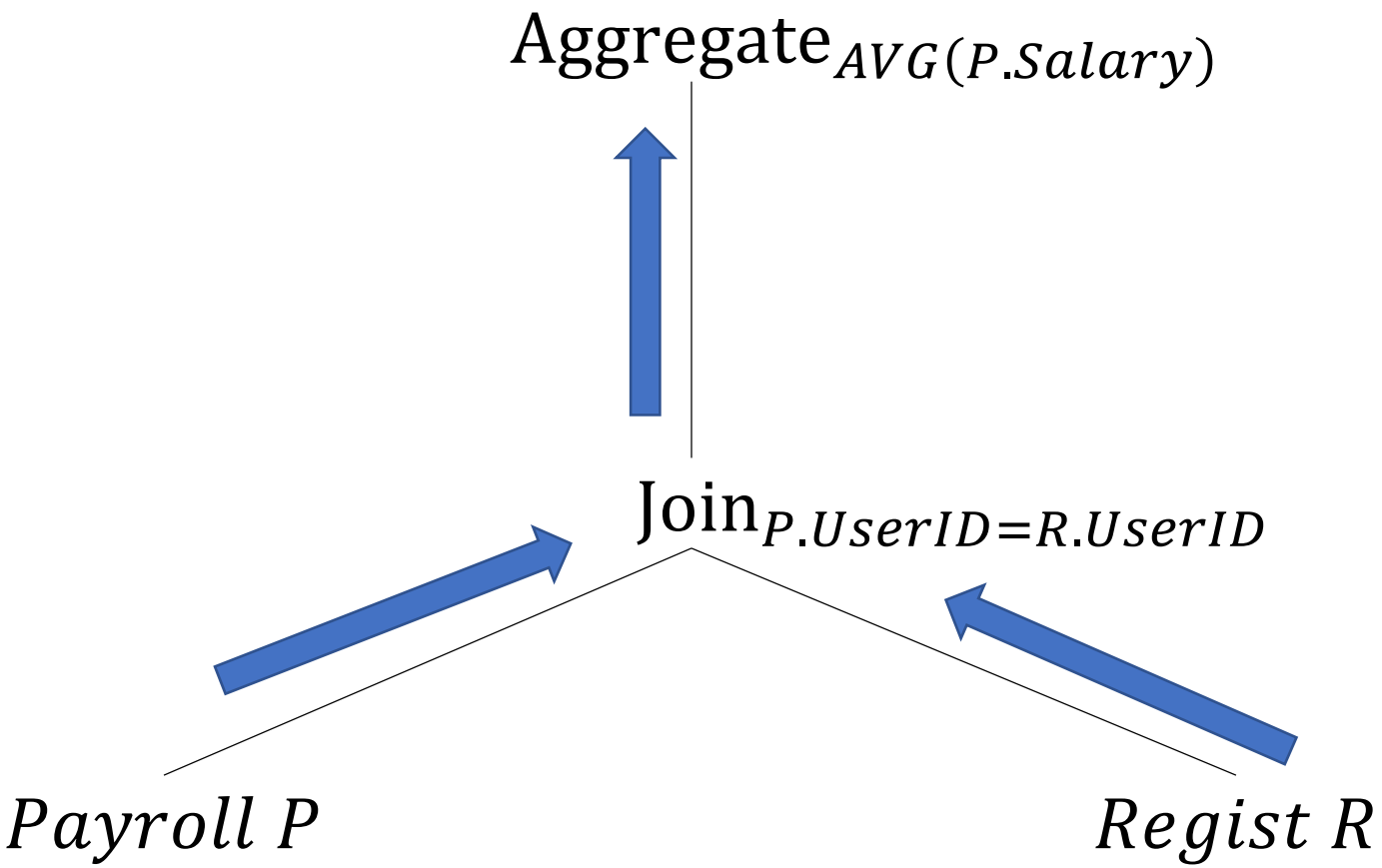
```
SELECT AVG(P.Salary)
  FROM Payroll AS P, Regist AS R
 WHERE P.UserID = R.UserID;
```



Aggregation Semantics

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

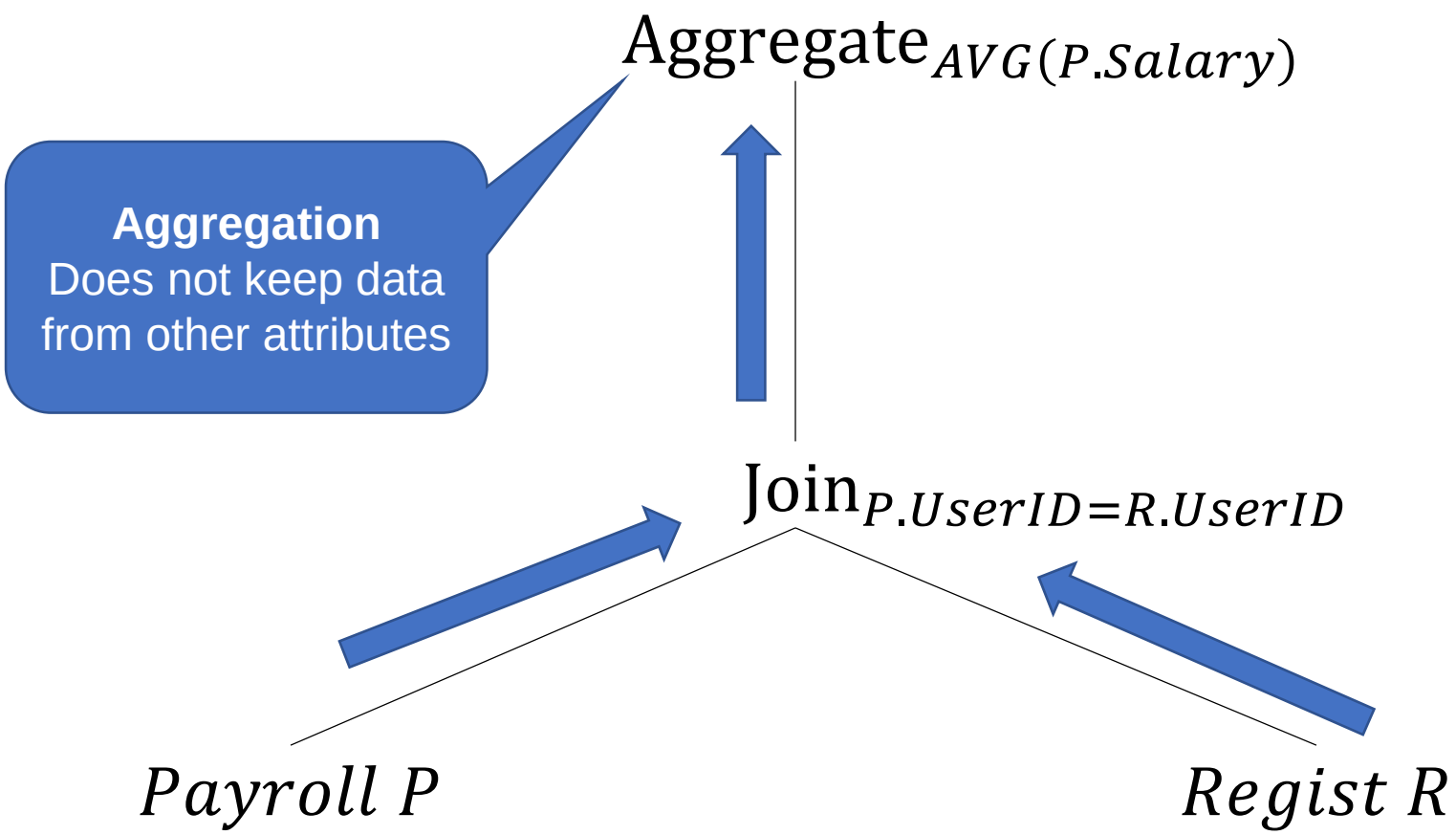
AVG(P.Salary)
76666



Aggregation Semantics

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

AVG(P.Salary)
76666



Grouping

- SQL allows you to specify what groups your query operates over
  - Sometimes a “whole-table” aggregation is too coarse-grained
  - We can partition our data based on **matching attribute values**

Grouping

- SQL allows you to specify what groups your query operates over
  - Sometimes a “whole-table” aggregation is too coarse-grained
  - We can partition our data based on **matching attribute values**

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

```
...
GROUP BY Job
...
```

Grouping

- SQL allows you to specify what groups your query operates over
  - Sometimes a “whole-table” aggregation is too coarse-grained
  - We can partition our data based on **matching attribute values**

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

```
...
GROUP BY Job
...
```

Grouping Example

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

Grouping Example

```
SELECT Job, MAX(Salary)
  FROM Payroll
 GROUP BY Job
```

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

Job	MAX(Salary)
TA	60000
Prof	100000

Grouping on Multiple Attributes

```
SELECT Name, MAX(Salary)
  FROM Payroll
 GROUP BY Job, Name
```

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

Name	Salary
Jack	50000
Allison	60000
Magda	90000
Dan	100000

Filtering Groups with HAVING

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

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

Filtering Groups with HAVING

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

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

Job	MAX(Salary)
Prof	100000

Aggregation RA

How is aggregation processed internally?

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

Aggregation RA

How is aggregation processed internally?

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

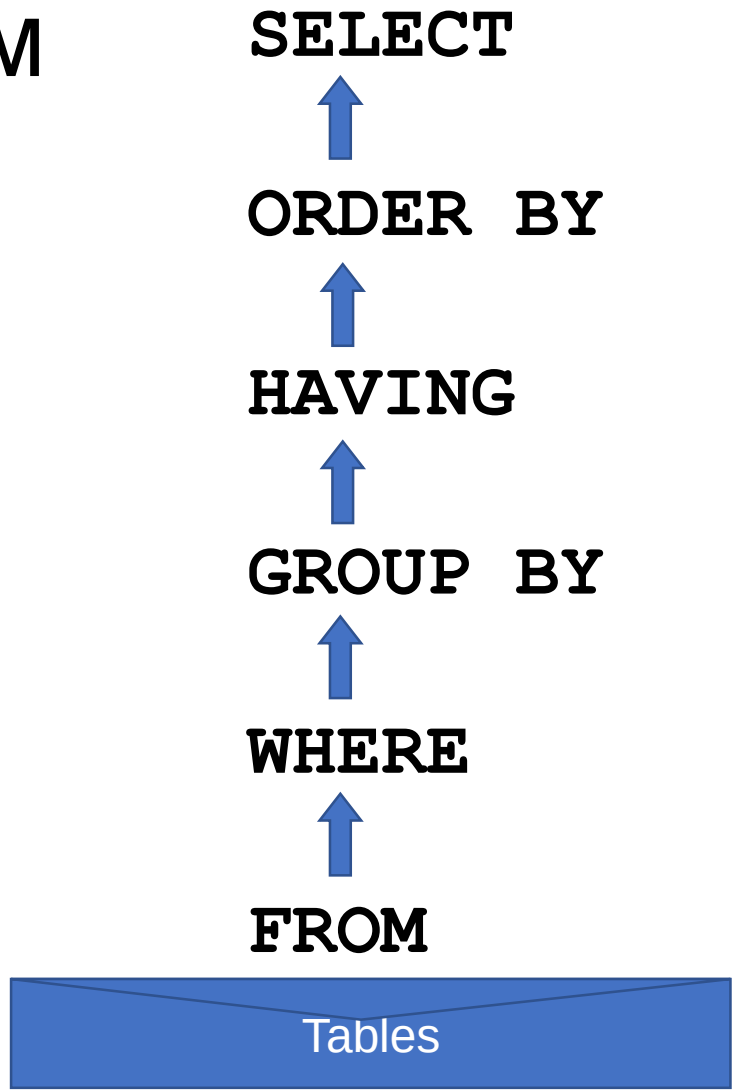
Our first preview of Relational Algebra:

“Having” applies **after** grouping

SQL and RA Vocab Summary

FWGHOS™

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



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
```

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

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

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

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

$$\text{Having}_{minSal>80000}$$

Job	maxSal	minSal
TA	60000	50000
Prof	100000	90000

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

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

$$\text{Having}_{minSal>80000}$$

Job	maxSal	minSal
TA	60000	50000
Prof	100000	90000

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

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

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

$$Job, maxSal$$

Job	maxSal	minSal
Prof	100000	90000

$$\text{Having}_{minSal>80000}$$

Job	maxSal	minSal
TA	60000	50000
Prof	100000	90000

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

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

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

Job	maxSal
Prof	100000

$$\text{Select}_{Job,\, maxSal}$$

Job	maxSal	minSal
Prof	100000	90000

$$\text{Having}_{minSal>80000}$$

Job	maxSal	minSal
TA	60000	50000
Prof	100000	90000

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

UserID	Name	Job	Salary
…	…	…	…

Preview: Relational Algebra

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

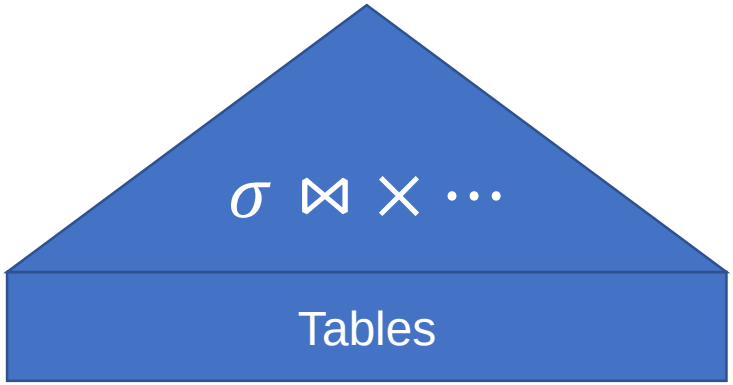
δ

Π

τ

σ

γ



SQL and RA Vocab Summary

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

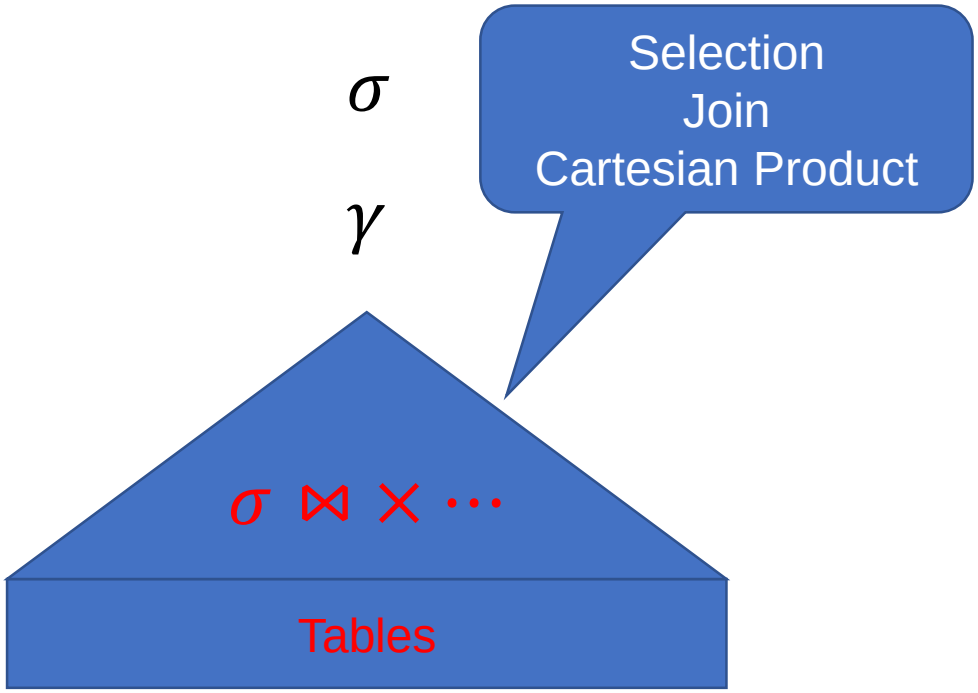
δ

Π

τ

σ

γ



SQL and RA Vocab Summary

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

δ

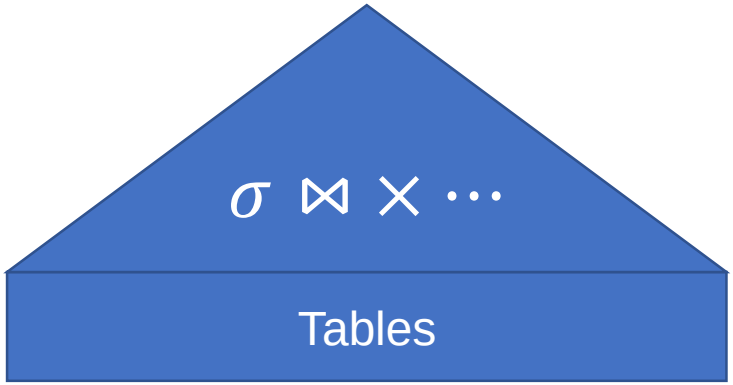
Π

τ

σ

γ

Aggregation



SQL and RA Vocab Summary

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

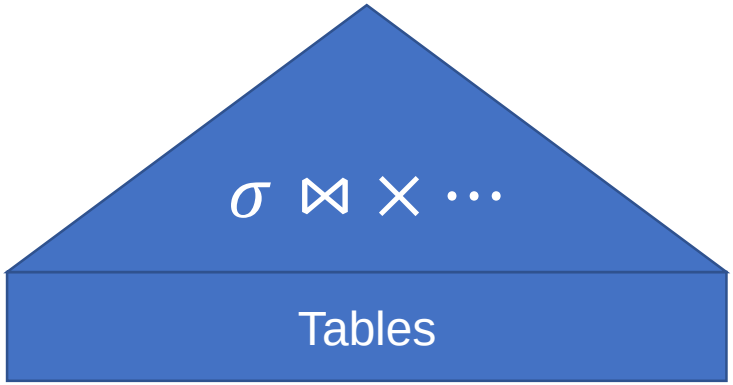
δ

Π

τ

σ Selection

γ



SQL and RA Vocab Summary

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

δ

Π

τ

σ

γ

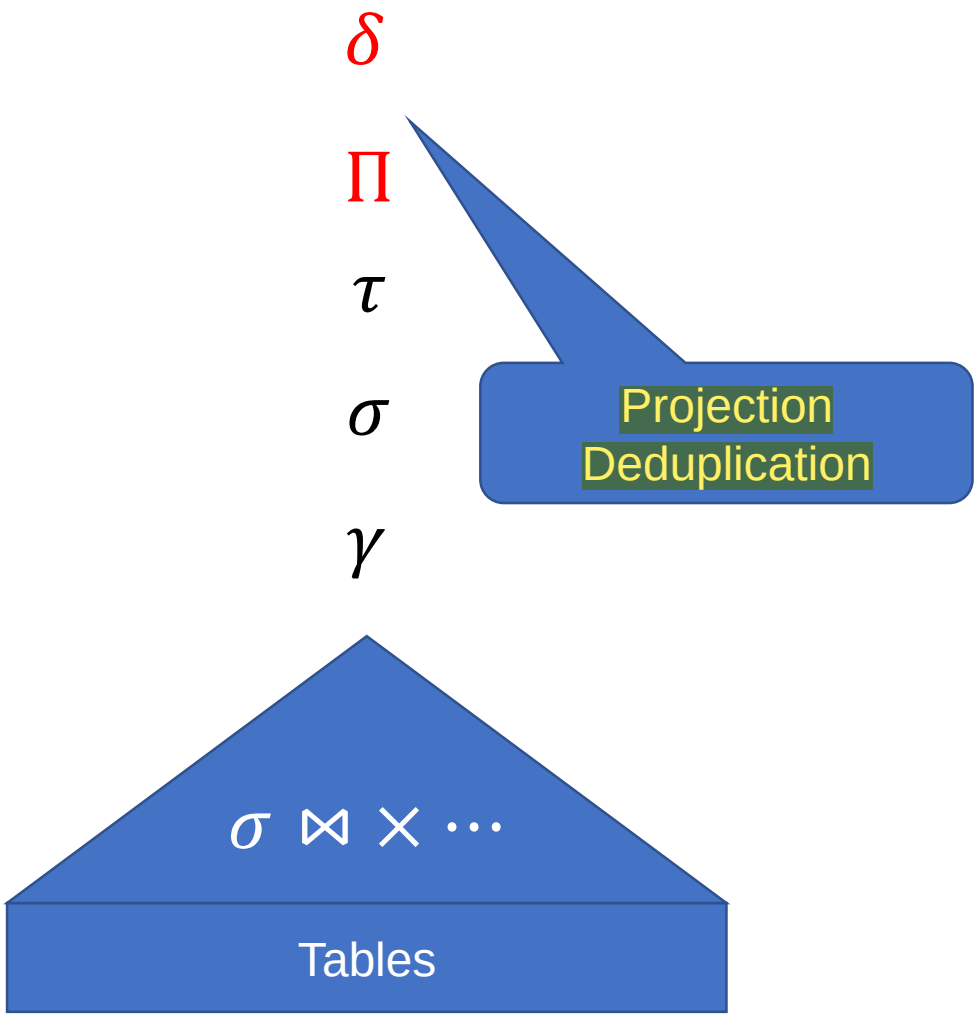
Sorting

$\sigma \bowtie \times \cdots$

Tables

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