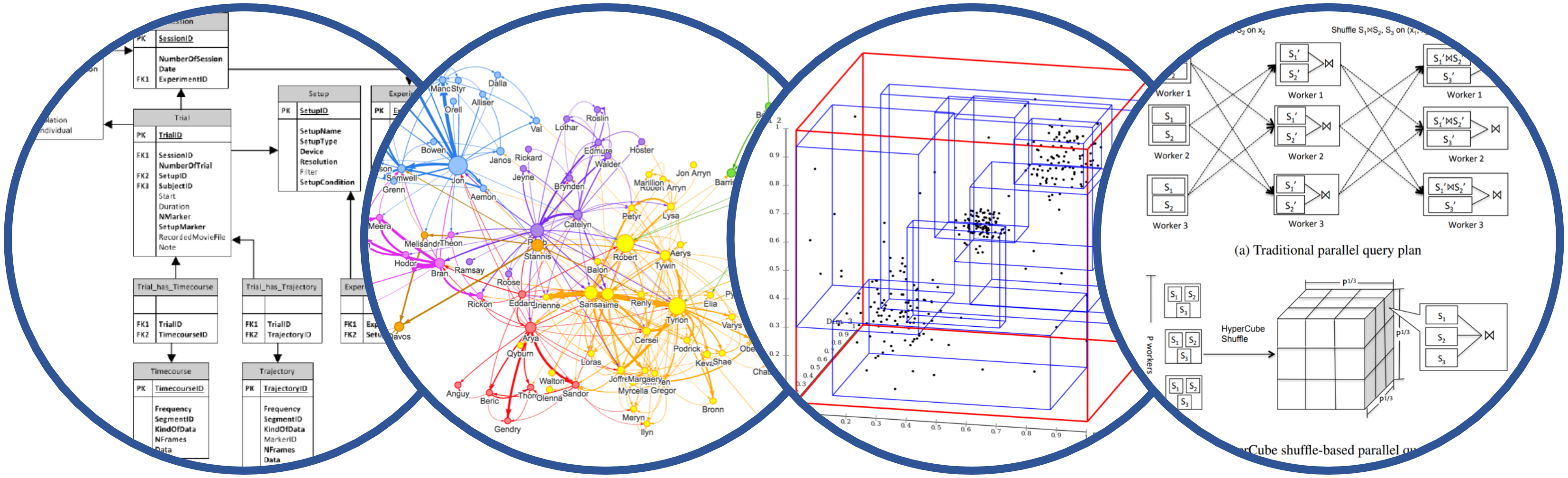


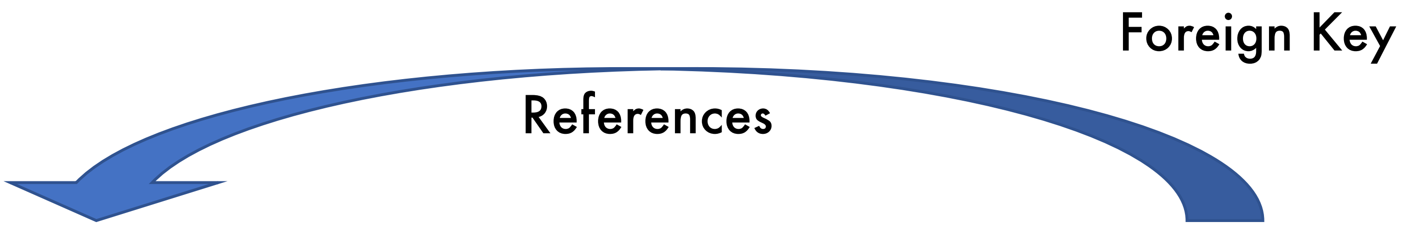
Introduction to Data Management

Aggregates

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

Recap – Keys and Foreign Keys

- Modeling multiple tables in the same database
  - Keys and foreign keys



Foreign Key

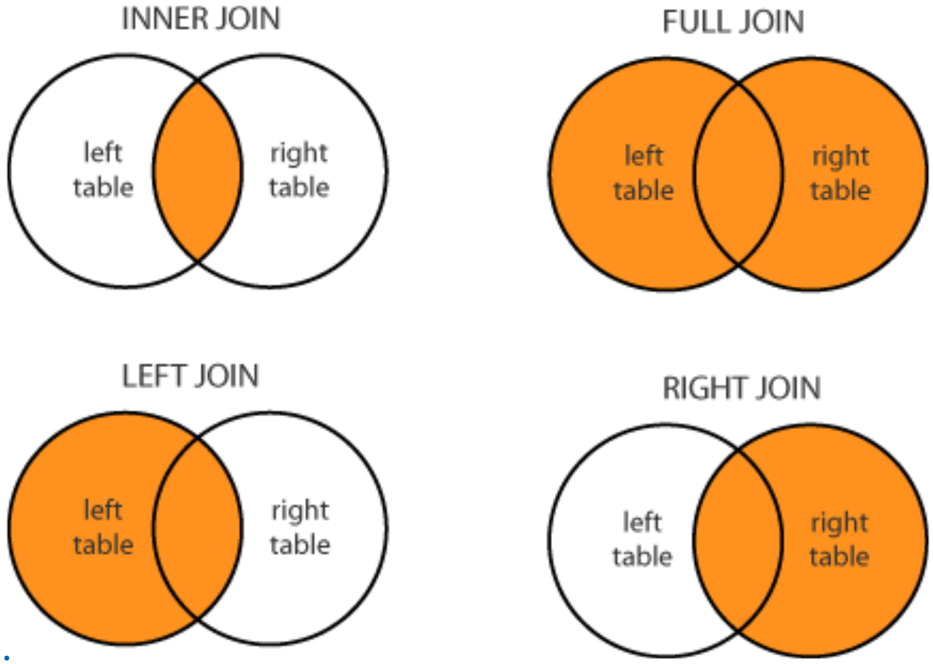
References

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

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
  - Self join pattern



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

Goals for Today

- Demo of SQL for past lecture
- Discussion of null values

- 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

3-Value Logic

- NULL values are neither TRUE nor FALSE

SQL 3-Valued Logic

Real data often has missing information

DBMSs often model missing information with NULL

SQL 3-Valued Logic

- FALSE = 0
- TRUE = 1
- UNKNOWN = 0.5

  [ex] price < 25 is UNKNOWN when price = NULL

SQL 3-Valued Logic

Formal definitions:

C1 AND C2	min(C1,C2)
C1 OR C2	max(C1,C2)
NOT C	1 – C

The rule for SELECT ... FROM ... WHERE <**C**> ... is the following:

if **C** = **TRUE**, then include the row in the output

if **C** = **FALSE** or **UNKNOWN**, then do not include it

SQL 3-Valued Logic

What is the output?

```
SELECT P.name
  FROM People AS P
 WHERE P.age >= 21
```

name	age
Bob	19
Amy	32
Joe	NULL
NULL	24

SQL 3-Valued Logic

What is the output?

```
SELECT P.name
  FROM People AS P
 WHERE P.age >= 21
```

name	age
Bob	19
Amy	32
Joe	NULL
NULL	24

SQL 3-Valued Logic

Why might NULL and 3-valued logic fail us?

SQL 3-Valued Logic

Why might NULL and 3-valued logic fail us?

```
SELECT P.name
  FROM People AS P
 WHERE P.age >= 21
       OR P.age < 21
```

name	age
Bob	19
Amy	32
Joe	NULL
NULL	24

SQL 3-Valued Logic

Why might NULL and 3-valued logic fail us?

```
SELECT P.name
  FROM People AS P
 WHERE P.age >= 21
       OR P.age < 21
```

Always true?

name	age
Bob	19
Amy	32
Joe	NULL
NULL	24

SQL 3-Valued Logic

Why might NULL and 3-valued logic fail us?

```
SELECT P.name
  FROM People AS P
 WHERE P.age >= 21
       OR P.age < 21
```

Nope.

name	age
Bob	19
Amy	32
Joe	NULL
NULL	24

SQL 3-Valued Logic

Another weird case

```
SELECT P.name
  FROM People AS P
 WHERE P.age = P.age
```

name	age
Bob	19
Amy	32
Joe	NULL
NULL	24

Aggregation functions

- New class of SQL queries:

Aggregates

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 anime?"

Actionable Results

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

Actionable Results

- We need summaries of data because we are often trying to **make decisions** and **succinctly convey information**
  - "How popular is this anime?" → 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 anime?" → 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 anime?" → 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 anime?" → 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 anime?" → 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 anime?" → 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 anime?” → 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 anime?" → 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 AnimeVideoViews …
  - 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 AnimeVideoViews …
  - 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"

What does "all the data" mean when there are things like joins?

Aggregation Semantics

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

```
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;
```

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

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

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

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;
```

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

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

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

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;
```

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

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

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

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

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

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

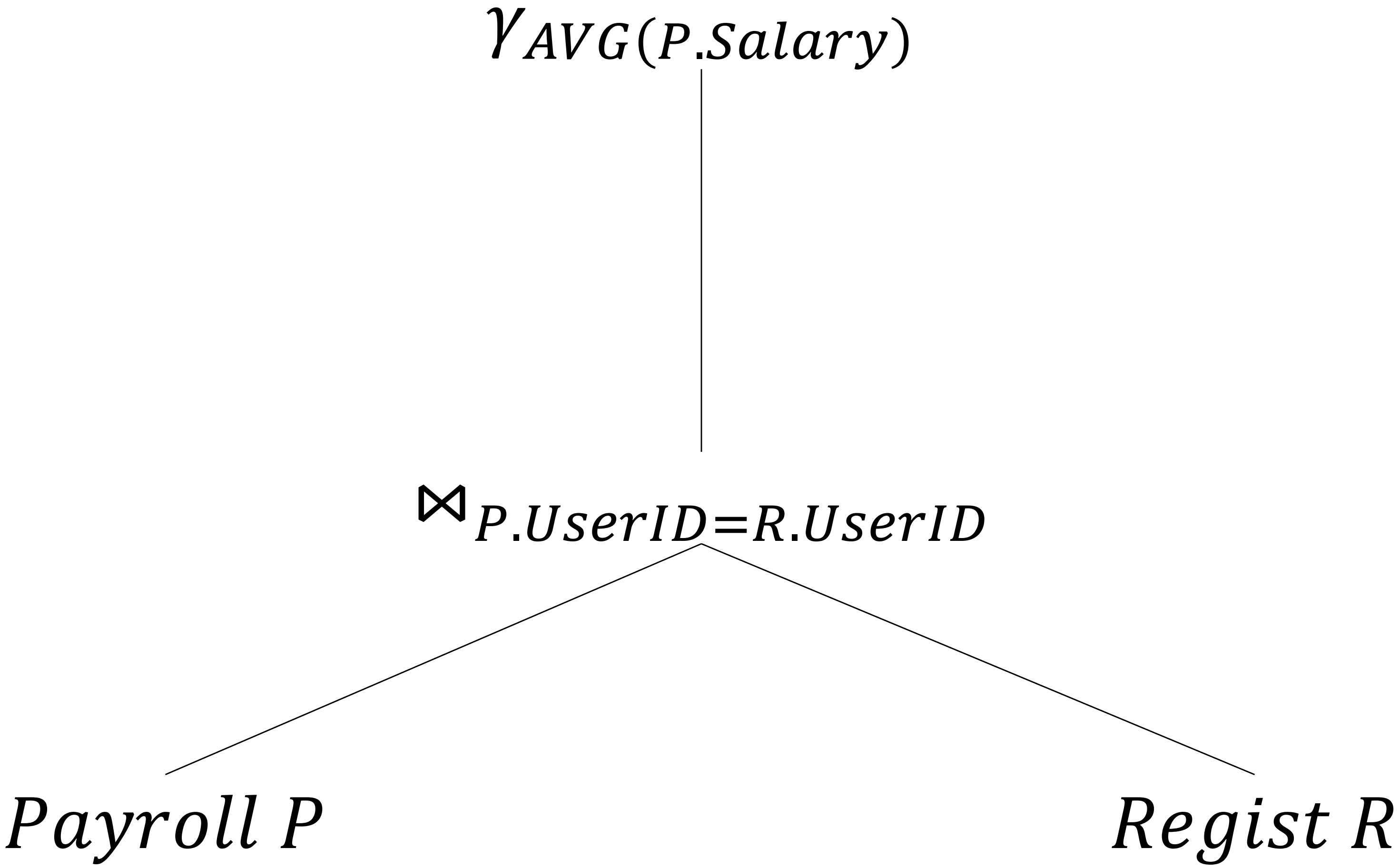
$\bowtie_{P.UserID=R.UserID}$

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;
```



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

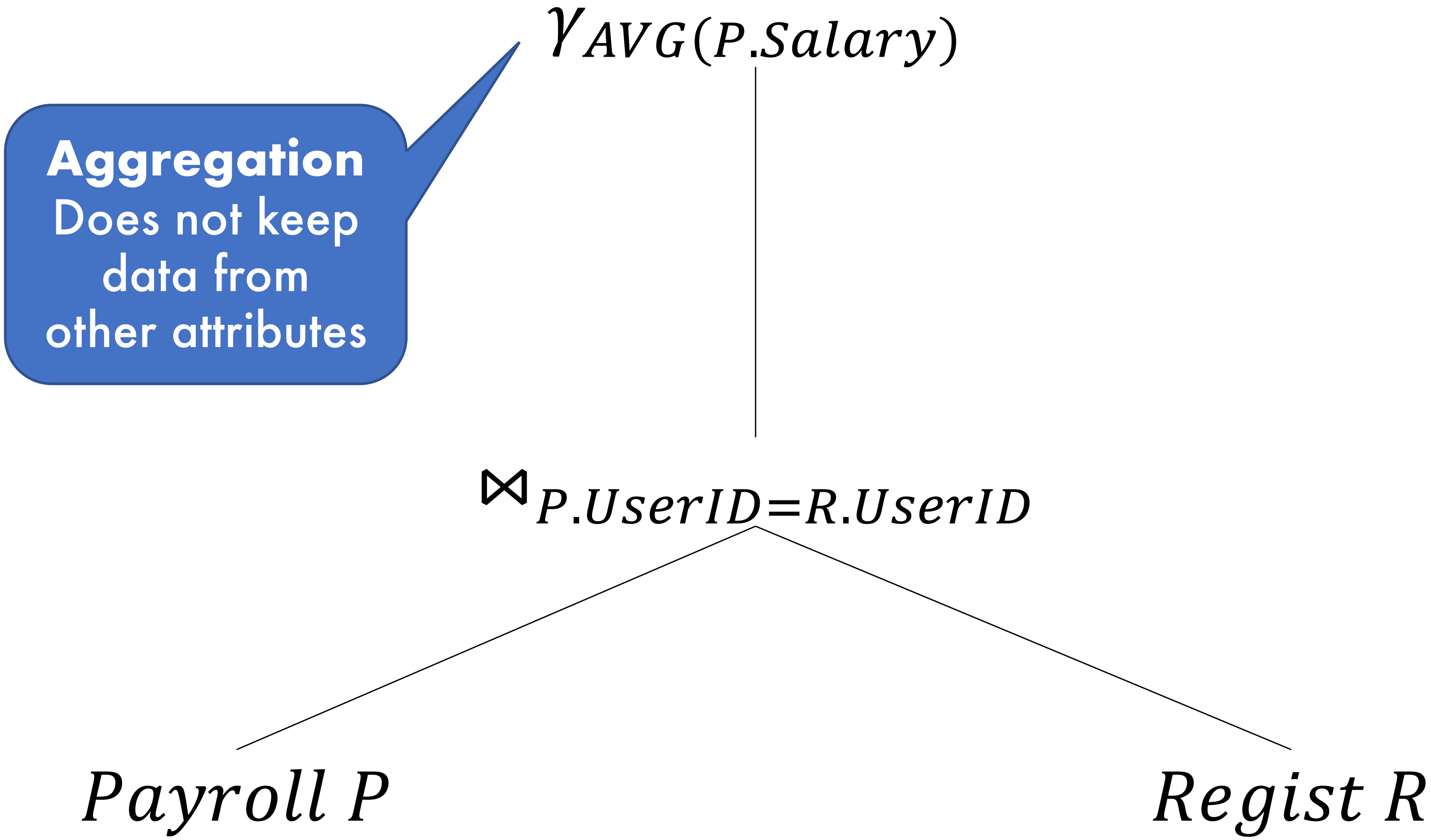
$\bowtie_{P.UserID=R.UserID}$

$Payroll\ P$

$Regist\ R$

Aggregation Semantics

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



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

Aggregation
Does not keep data from other attributes

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

$Payroll\ P$

$Regist\ R$

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

```
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

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

```
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

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

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

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

$$\sigma_{minSal>80000}$$

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

$$\sigma_{minSal>80000}$$

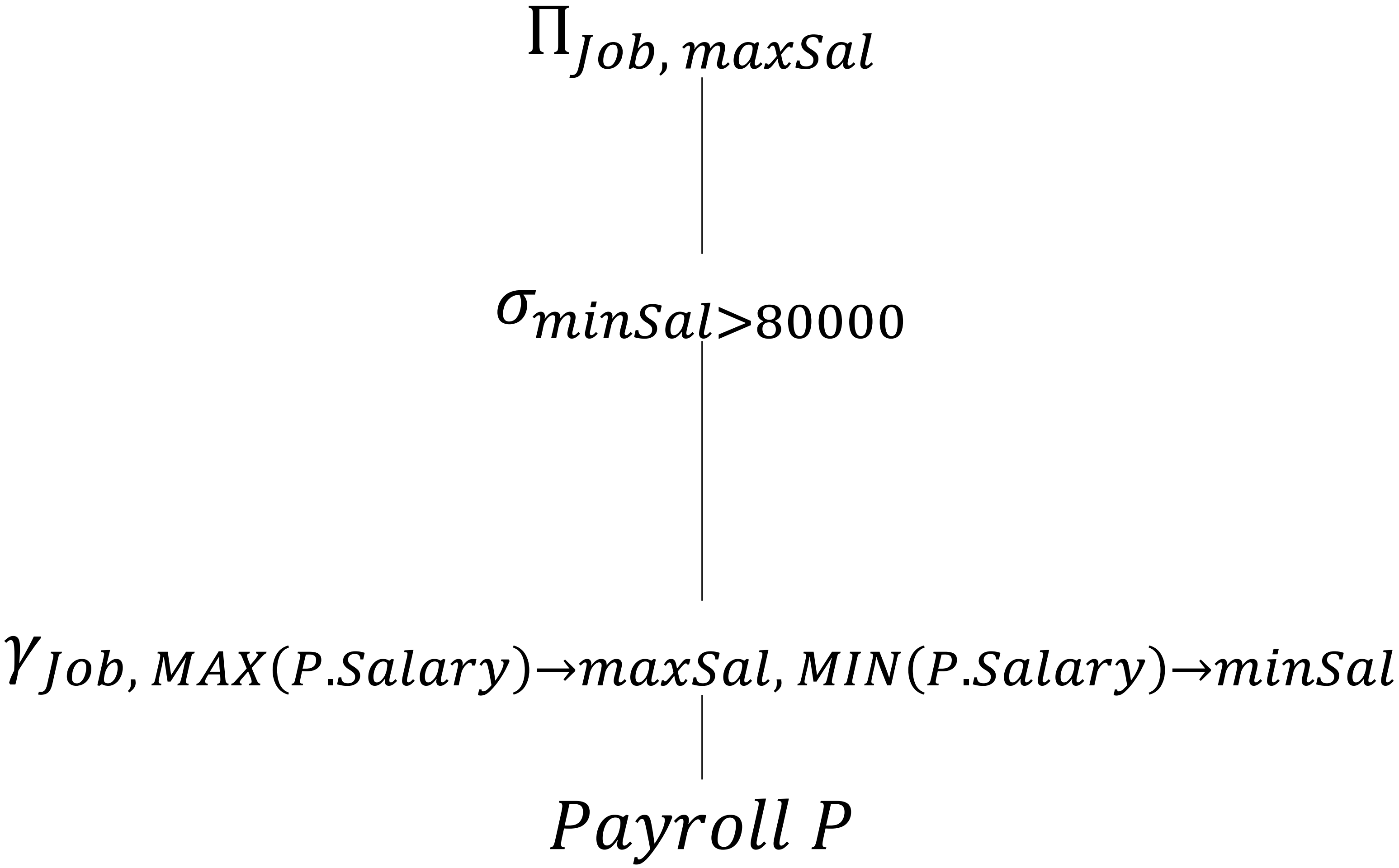
Job	maxSal	minSal
TA	60000	50000
Prof	100000	90000

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

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

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



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

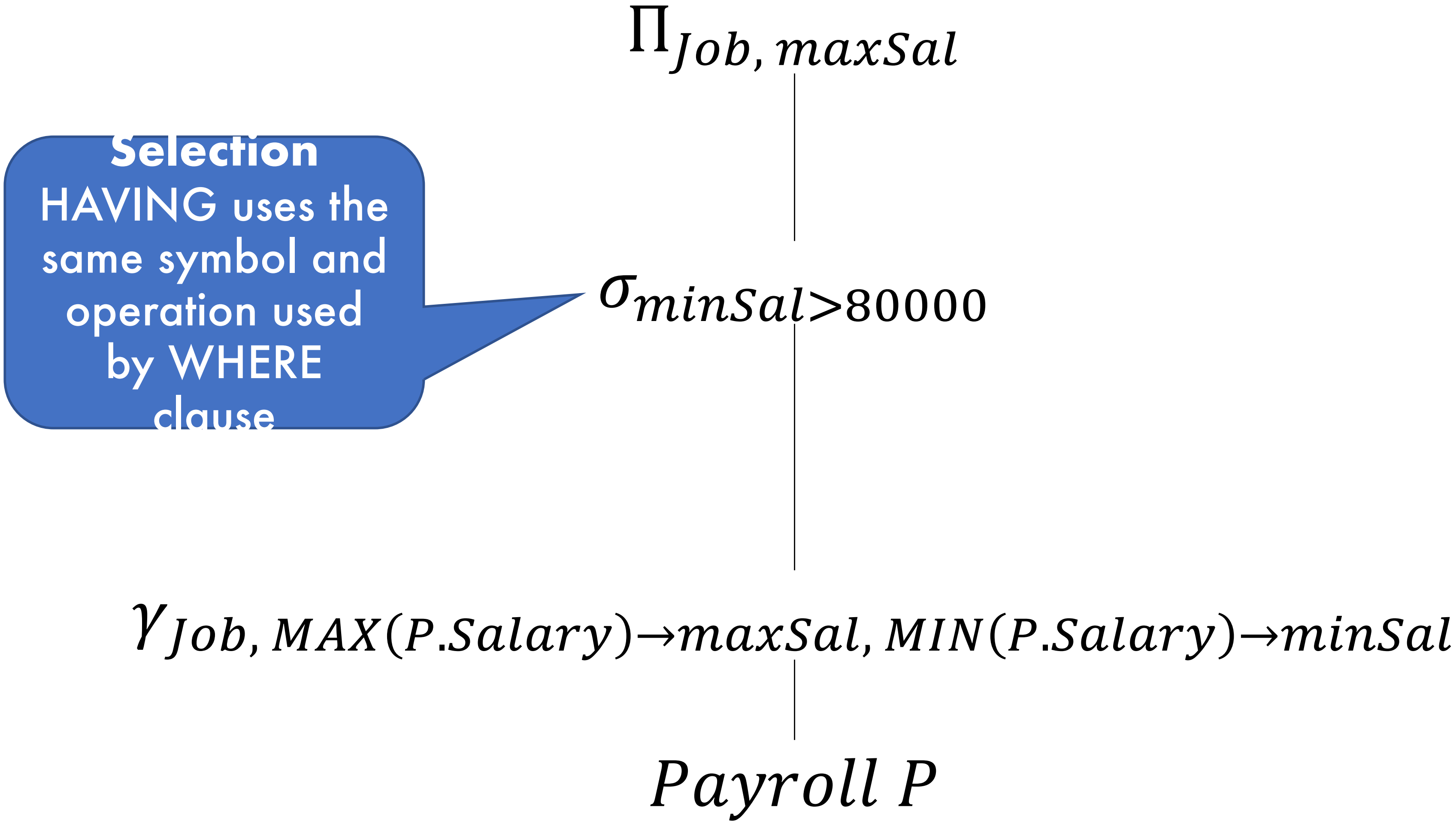
$$\sigma_{minSal > 80000}$$

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

$$Payroll\ P$$

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

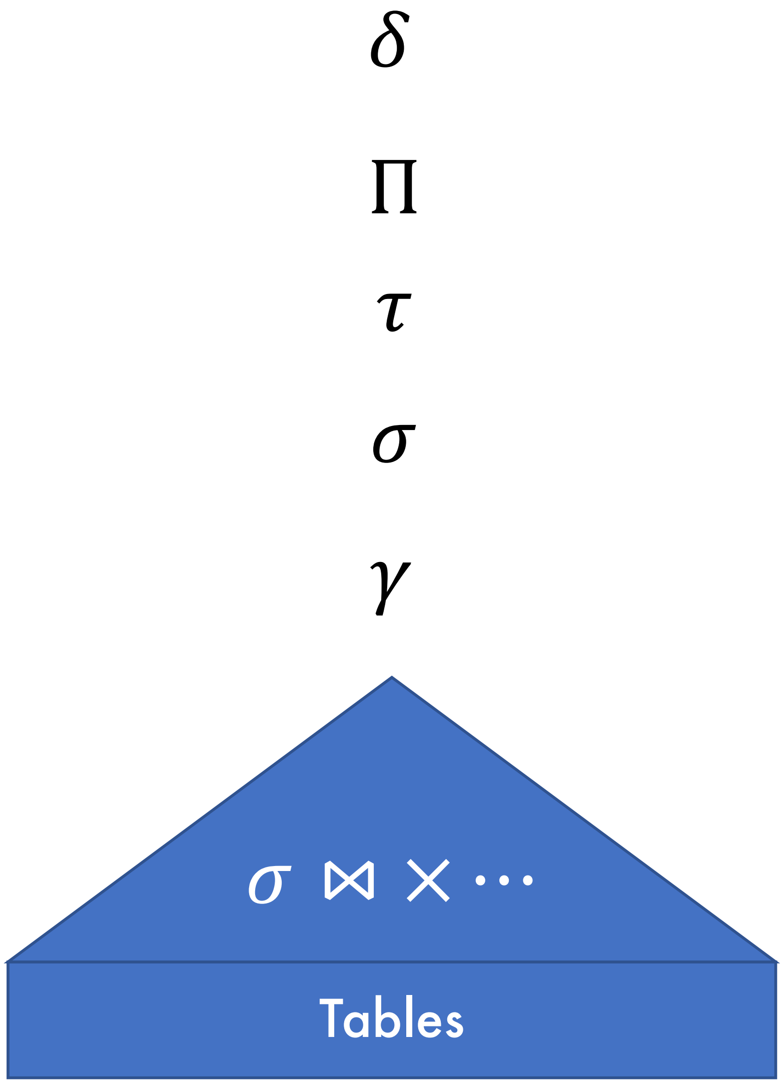
$$\sigma_{minSal > 80000}$$

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

$$Payroll\ P$$

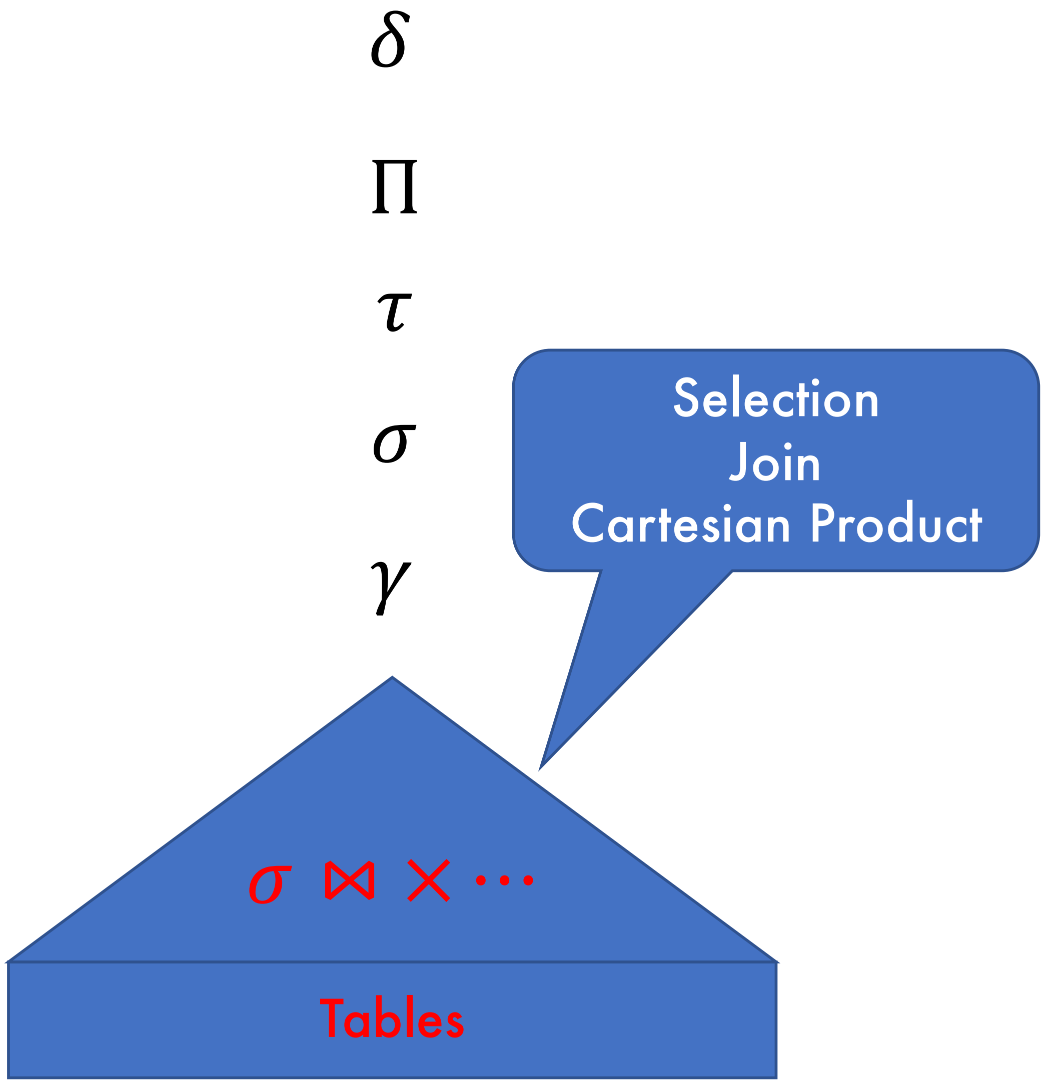
SQL and RA Vocab Summary

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



SQL and RA Vocab Summary

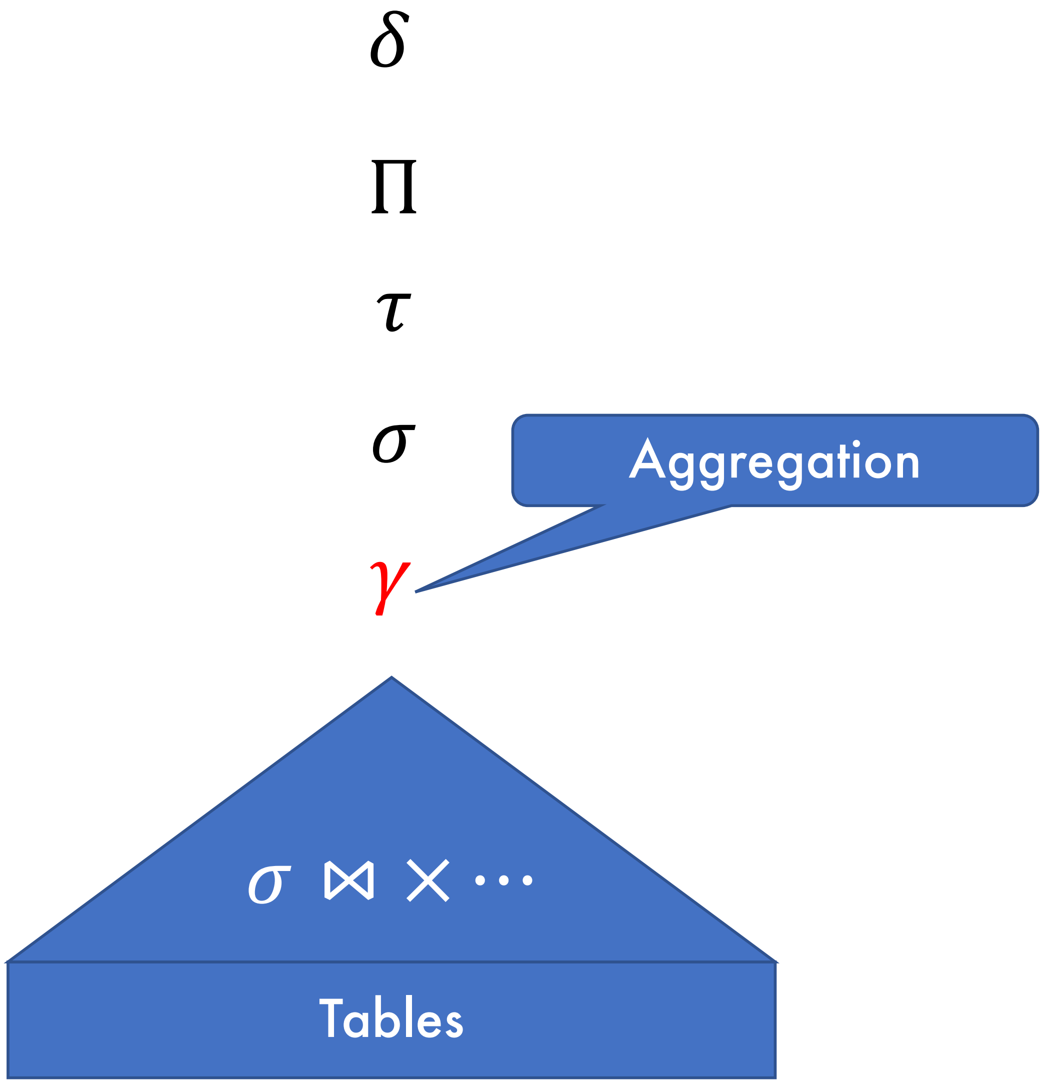
```
SELECT   …
  FROM   …
 WHERE   …
 GROUP BY …
HAVING   …
 ORDER BY …
```



Selection
Join
Cartesian Product

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

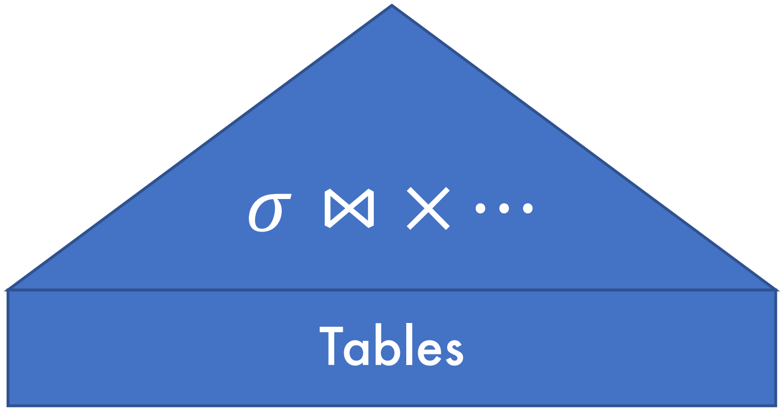
Π

τ

σ

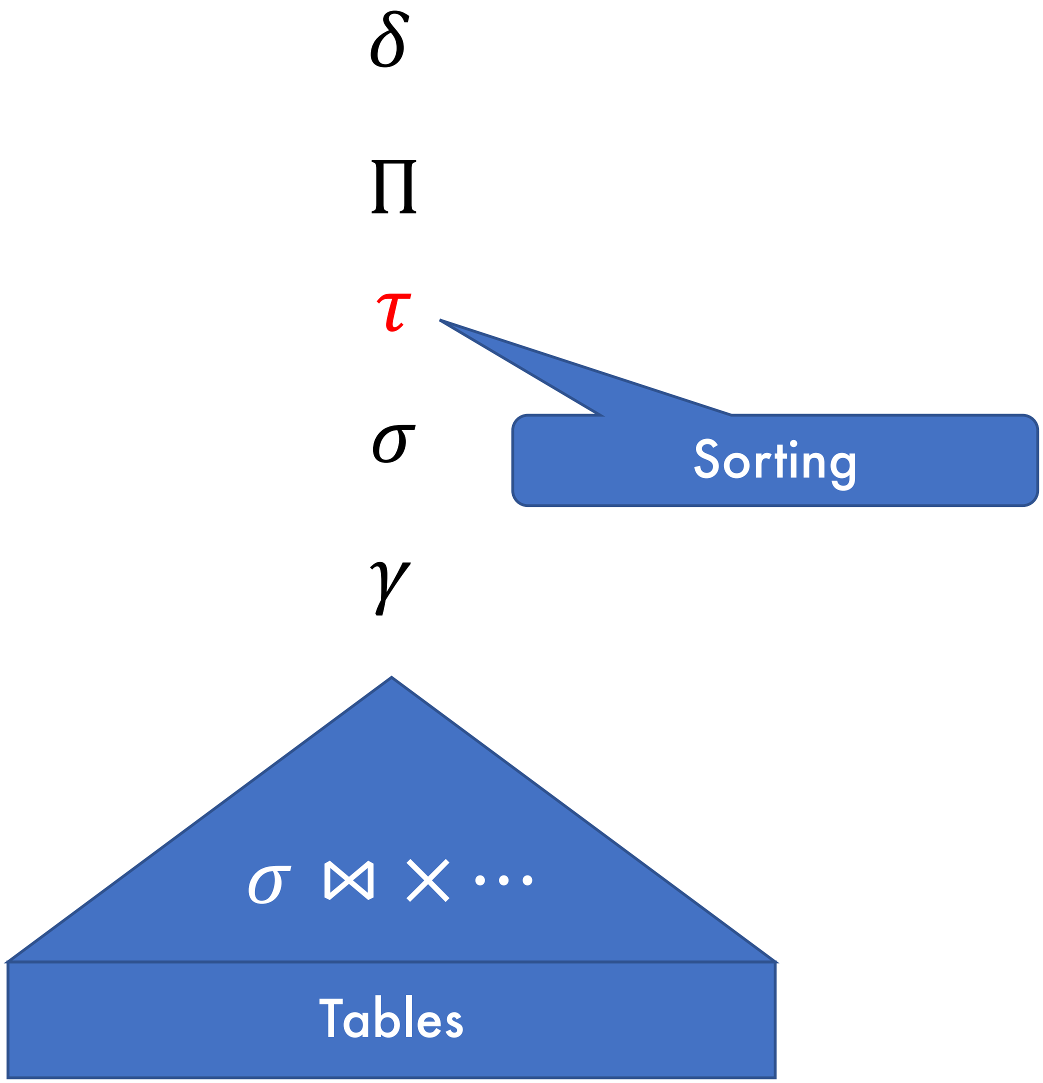
Selection

γ



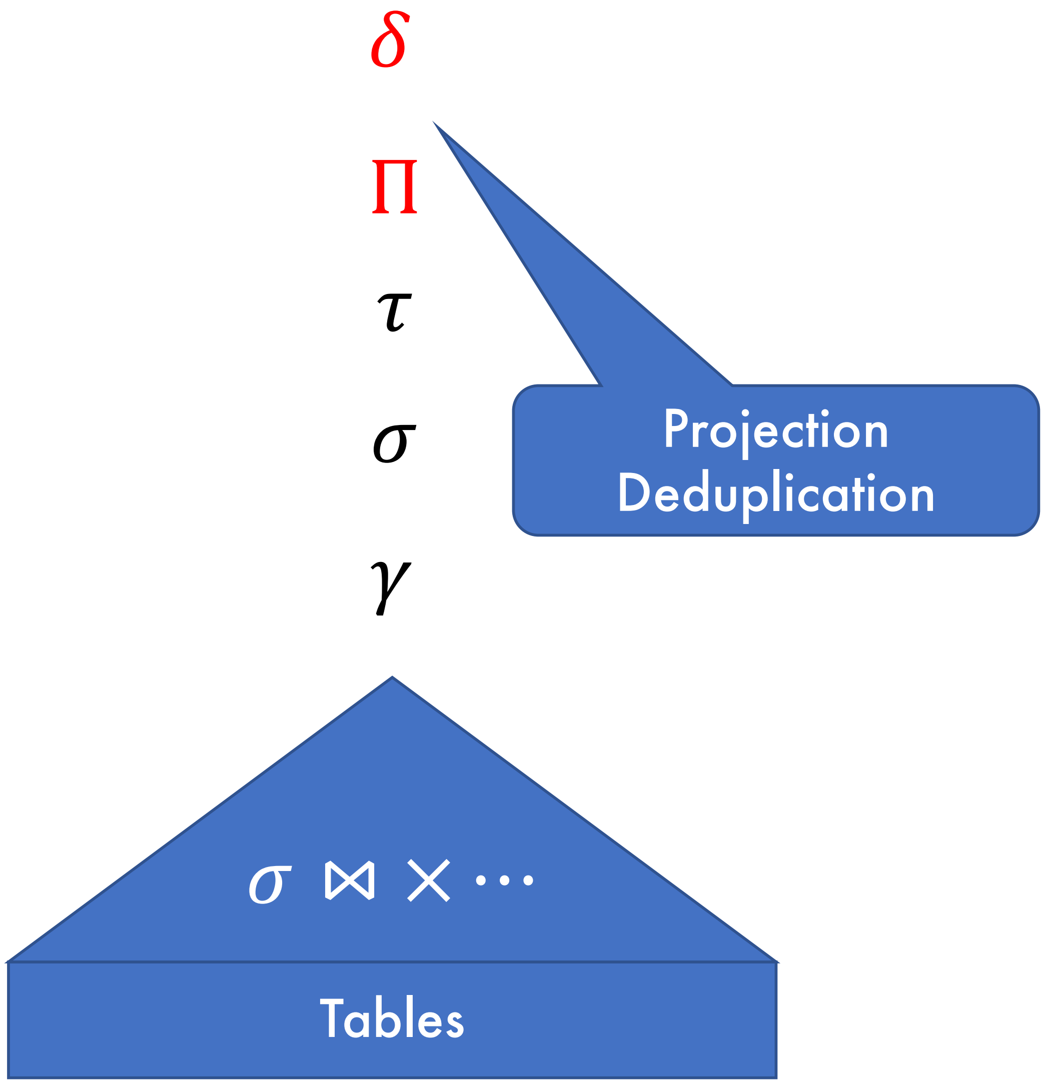
SQL and RA Vocab Summary

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



SQL and RA Vocab Summary

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



δ

Π

τ

σ

γ

Projection
Deduplication

$\sigma \bowtie \times \cdots$

Tables

SQL and RA Vocab Summary

FWGHOS™

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

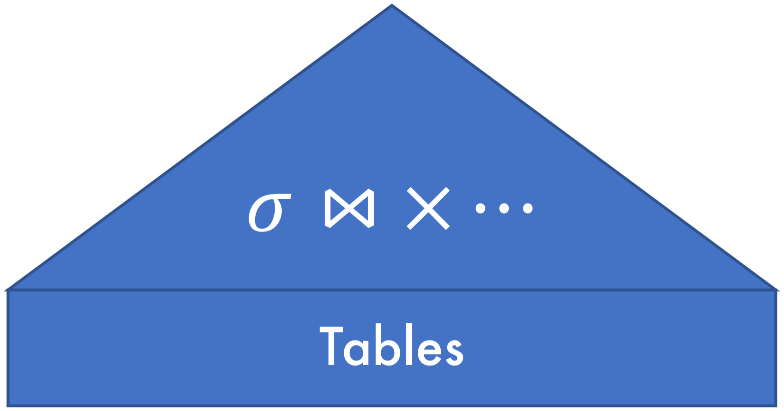
δ

Π

τ

σ

γ



The Witnessing Problem

- Also known as argmax/argmin
- Ex: Return the person with the highest salary for each job type

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

The Witnessing Problem

- Also known as argmax/argmin
- Ex: Return the person with the highest salary for each job type

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

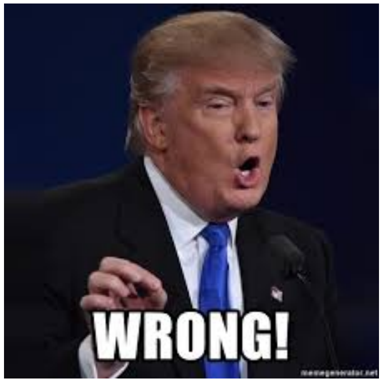
Easy right?

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

The Witnessing Problem

- Also known as argmax/argmin
- Ex: Return the person with the highest salary for each job type

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



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

The Witnessing Problem

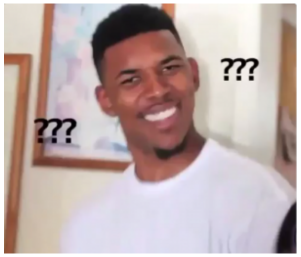
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

Name	MAX(Salary)
???	60000
???	100000

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

The Witnessing Problem

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



Name	MAX(Salary)
???	60000
???	100000

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

The Witnessing Problem

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

Failed to execute query. Error: Column 'Payroll.name' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.

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

The Witnessing Problem

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

Name	MAX(Salary)
	60000
???	100000

SELECT, HAVING, ORDER BY
Must use aggregate functions or attributes in GROUP BY

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

The Witnessing Problem

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

Return the person with the highest salary for each job type

How do we witness the maxima for a group?
Discuss!
Conceptual ideas are great

The Witnessing Problem

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

Return the person with the highest salary for each job type

Main idea: we need to join the respective maxima to each row

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

Main idea: we need to join the respective maxima to each row

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

Main idea: we need to join the respective maxima to each row

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

```
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)
```

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

Takeaways

- FWGHOS™
- Combining techniques (aggregates and joins) allows you to answer complex questions (e.g. witnessing queries)