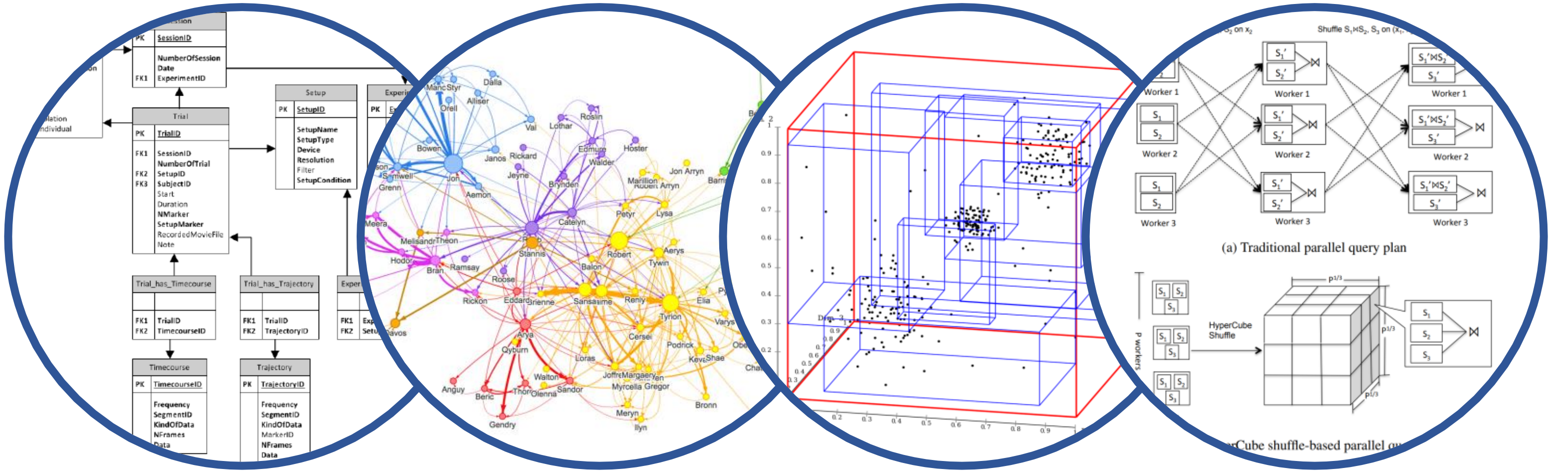


Introduction to Data Management

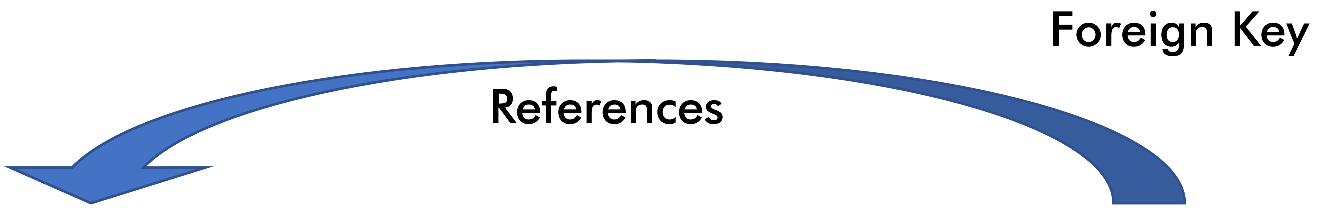
Feeling *aggregated*?

Shana Hutchison

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

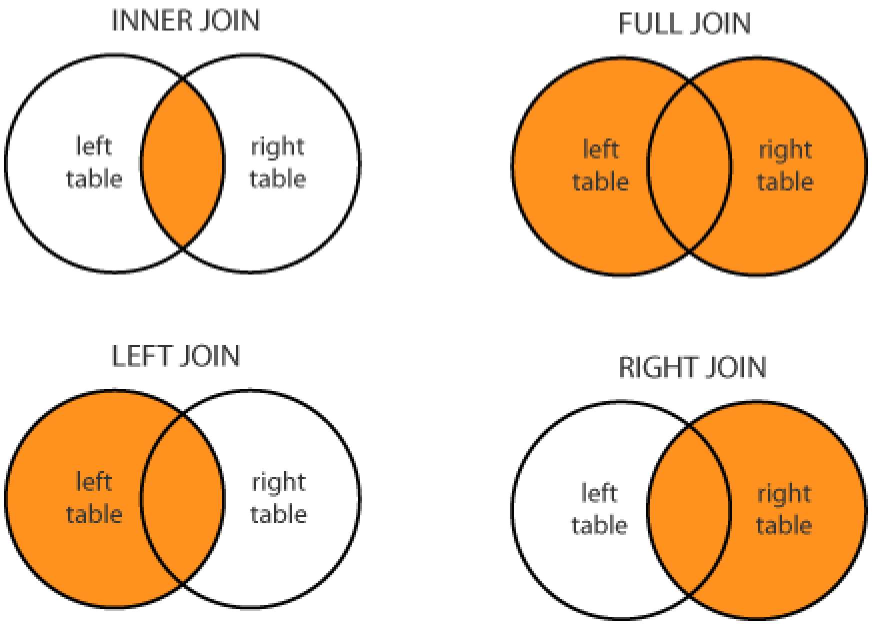
References

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 combine data across tables
  - Nested-loop semantics
  - Filtered Cartesian product semantics
  - Inner join (most common)
  - Outer joins preserve non-matching tuples
  - Self join pattern



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

Actionable Results

- Summaries of data help **make decisions** and **succinctly convey information**
  - "How popular is this anime?" → COUNT
  - "Do I spend too much on tea?" → SUM
  - "Am I being ripped off by this dealer?" → AVG
  - "Who raised the most money for charity?" → MAX
  - "What is the cheapest Greek yogurt?" → MIN

Actionable Results

- Summaries of data help **make decisions** and **succinctly convey information**
  - SELECT **COUNT**(*) FROM AnimeVideoViews …
  - SELECT **SUM**(cost) FROM TeaReceipts …
  - SELECT **AVG**(price) FROM CarDealers …
  - SELECT **MAX** (score) FROM Donations …
  - SELECT **MIN**(price) FROM YogurtStores …

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

Watch out for NULLs!

Actionable Results

- Summaries of data help **make decisions** and **succinctly convey information**
  - SELECT **COUNT**(*) FROM AnimeVideoViews …
  - SELECT SUM(cost) FROM TeaReceipts …
  - SELECT AVG price) FROM CarDealers …
  - SELECT MA core) FROM Donations …
  - SELECT MI rice) FROM YogurtStores …

COUNT(*) → # of rows regardless of NULL

Watch out for NULLs!

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;
```

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

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

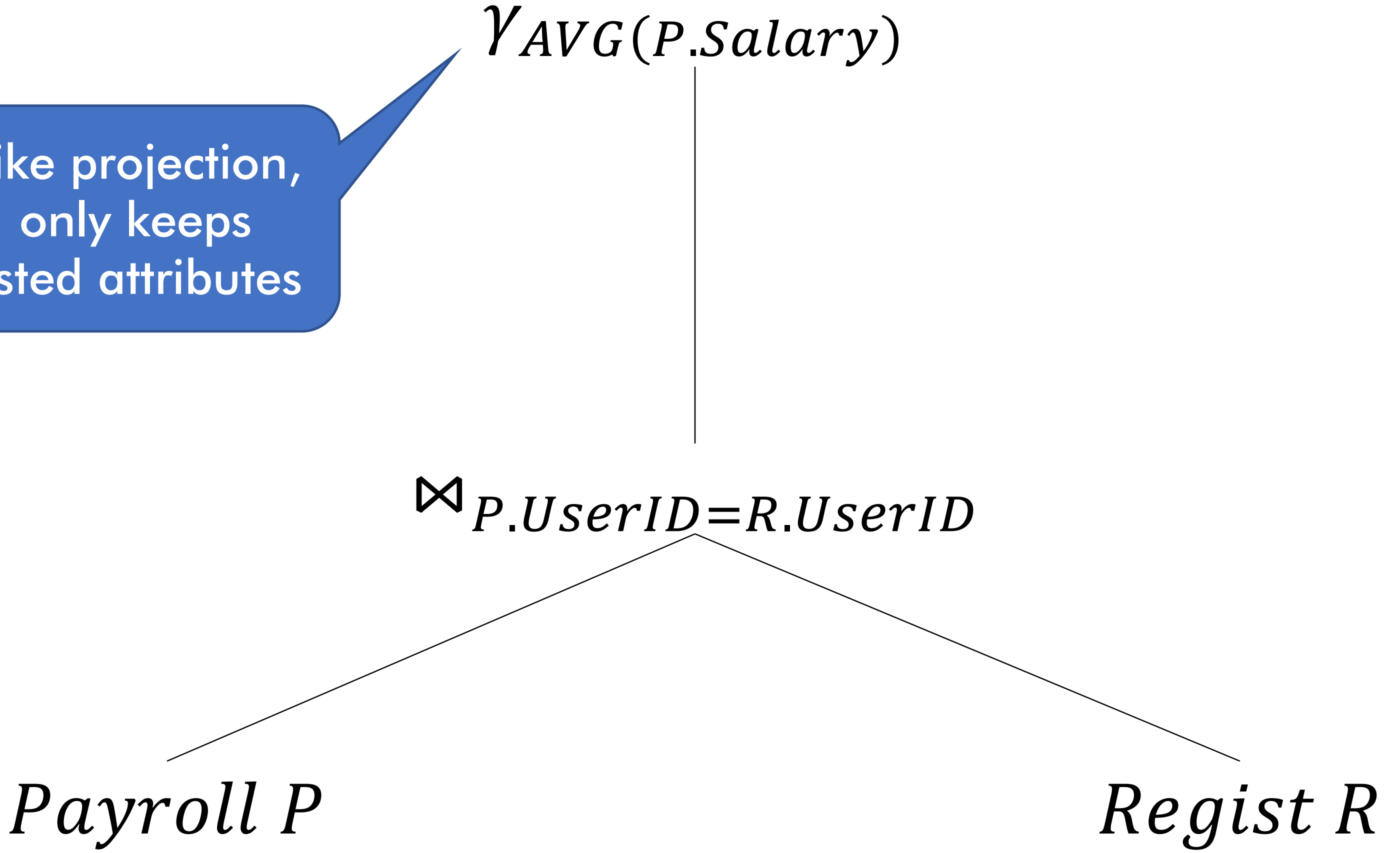
UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000

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;
```

1-arg **Aggregation** op (Greek “gamma”)
Compute aggregates,
grouping by non-aggregates



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

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

$Payroll\ P$

$Regist\ R$

Like projection, only keeps listed attributes

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

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
424	Jack	TA	55000

Name	Salary
Jack	55000
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
```

Example RA syntax for creating aliases maxSal, minSal

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

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

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

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

$$\sigma_{minSal > 80000}$$

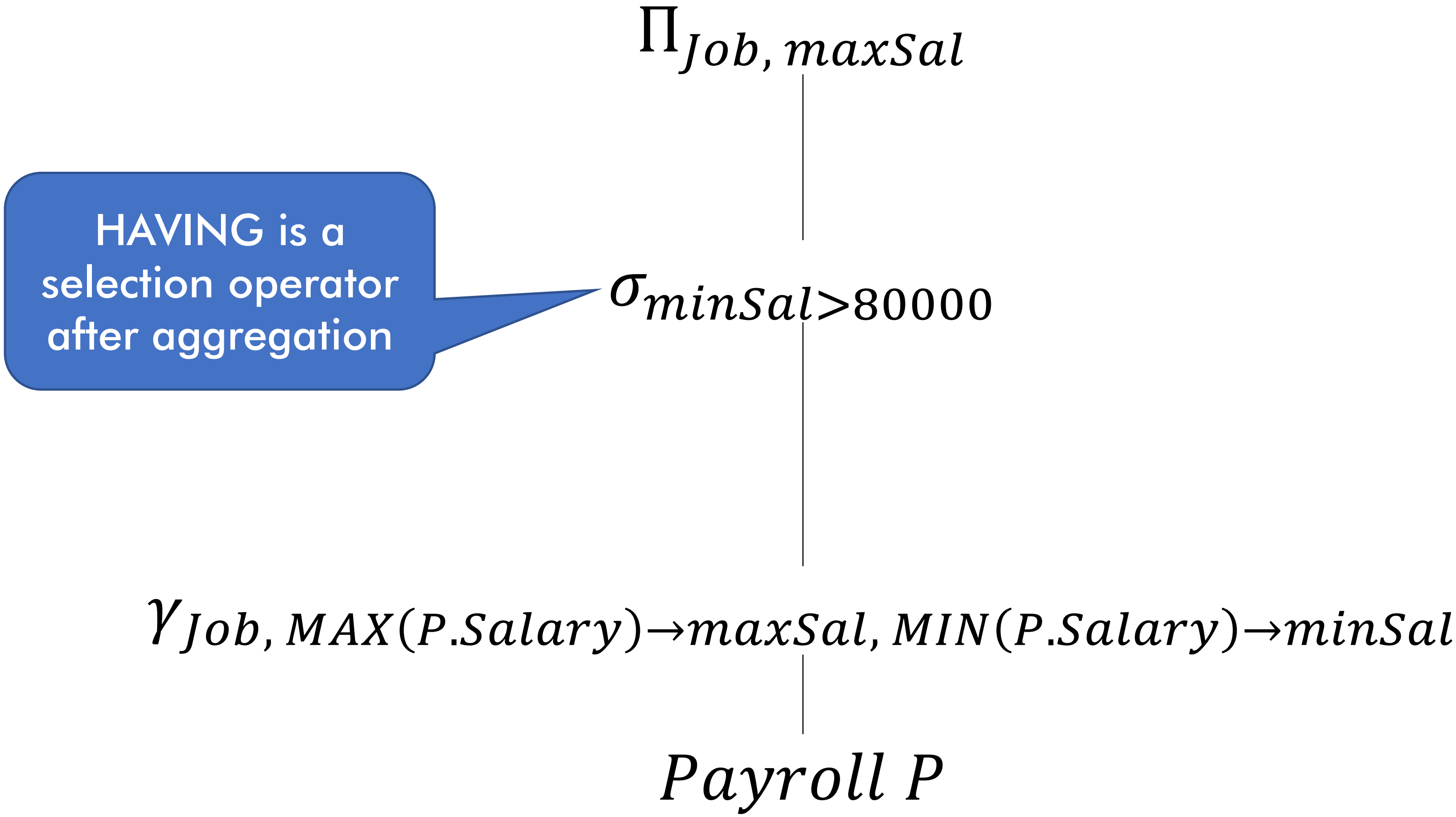
Job	maxSal	minSal
TA	60000	50000
Prof	100000	90000

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

UserID	Name	Job	Salary
…	…	…	…

Aggregation RA

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



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

$$\sigma_{minSal > 80000}$$

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

$$Payroll\ P$$

HAVING is a selection operator after aggregation

Agg: Nested-Loop Semantics

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

Job	MAX(Salary)	MIN(Salary)

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

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

$$\sigma_{minSal > 80000}$$

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

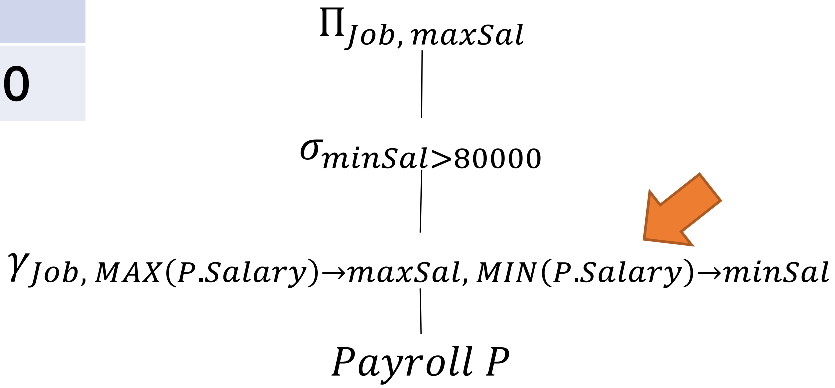
$$Payroll\ P$$

```
for each row in Payroll:
 insert into a dictionary D
  row.Job => MAX(row.Salary), MIN(row.Salary)
for each row in D:
 if (row.MIN(Salary) > 80000)
  output (row.Job, row.MAX(Salary))
```

Agg: Nested-Loop Semantics

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

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



$\Pi_{Job,\,maxSal}$

$\sigma_{minSal>80000}$

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

$Payroll\ P$

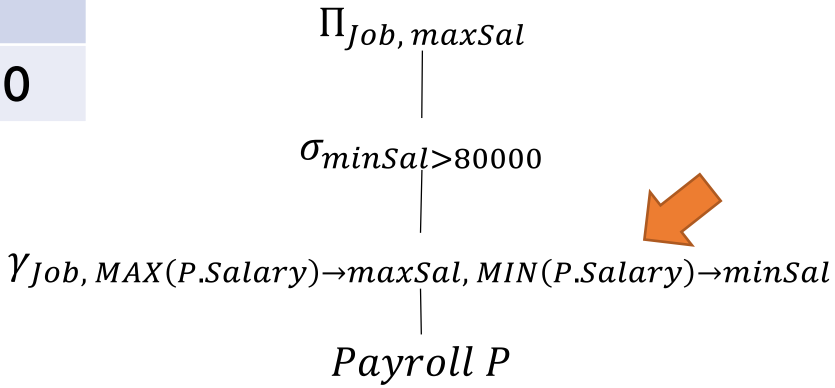
Job	MAX(Salary)	MIN(Salary)
TA	50000	50000

```
for each row in Payroll:
 insert into a dictionary D
  row.Job => MAX(row.Salary), MIN(row.Salary)
for each row in D:
 if (row.MIN(Salary) > 80000)
  output (row.Job, row.MAX(Salary))
```

Agg: Nested-Loop Semantics

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

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



$\Pi_{Job,\, maxSal}$

$\sigma_{minSal > 80000}$

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

$Payroll\ P$

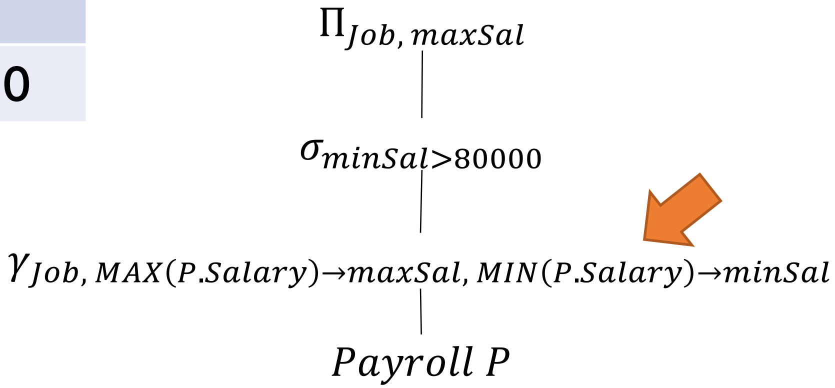
Job	MAX(Salary)	MIN(Salary)
TA	50000	50000

```
for each row in Payroll:
 insert into a dictionary D
  row.Job => MAX(row.Salary), MIN(row.Salary)
for each row in D:
 if (row.MIN(Salary) > 80000)
  output (row.Job, row.MAX(Salary))
```

Agg: Nested-Loop Semantics

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

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



$\Pi_{Job,\, maxSal}$

$\sigma_{minSal>80000}$

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

$Payroll\ P$

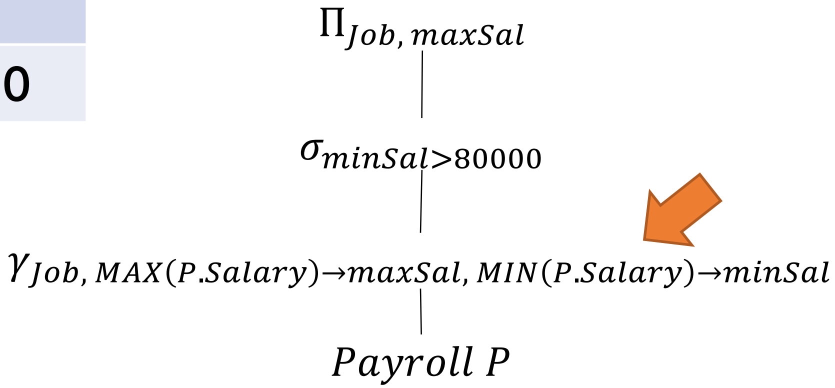
Job	MAX(Salary)	MIN(Salary)
TA	60000	50000

```
for each row in Payroll:
 insert into a dictionary D
  row.Job => MAX(row.Salary), MIN(row.Salary)
for each row in D:
 if (row.MIN(Salary) > 80000)
  output (row.Job, row.MAX(Salary))
```

Agg: Nested-Loop Semantics

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

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



$\Pi_{Job,\, maxSal}$

$\sigma_{minSal > 80000}$

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

$Payroll\ P$

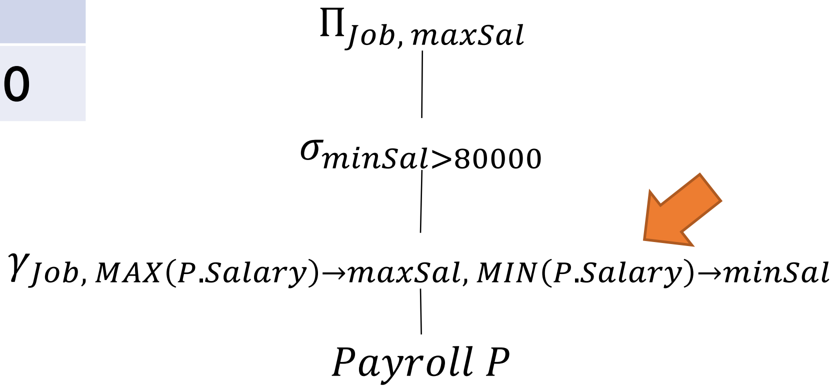
Job	MAX(Salary)	MIN(Salary)
TA	60000	50000

```
for each row in Payroll:
 insert into a dictionary D
  row.Job => MAX(row.Salary), MIN(row.Salary)
for each row in D:
 if (row.MIN(Salary) > 80000)
  output (row.Job, row.MAX(Salary))
```

Agg: Nested-Loop Semantics

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

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



$\Pi_{Job,\, maxSal}$

$\sigma_{minSal>80000}$

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

$Payroll\ P$

Job	MAX(Salary)	MIN(Salary)
TA	60000	50000
Prof	90000	90000

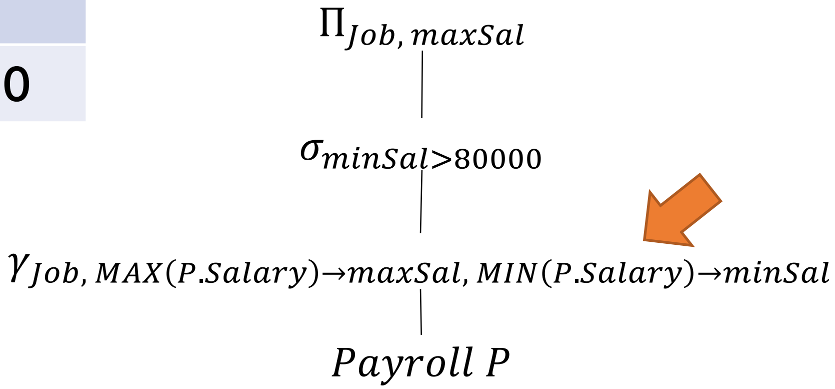
```
for each row in Payroll:
 insert into a dictionary D
  row.Job => MAX(row.Salary), MIN(row.Salary)
for each row in D:
 if (row.MIN(Salary) > 80000)
  output (row.Job, row.MAX(Salary))
```

Agg: Nested-Loop Semantics

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

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

Job	MAX (Salary)	MIN(Salary)
TA	60000	50000
Prof	90000	90000



$\Pi_{Job,\,maxSal}$
$\sigma_{minSal>80000}$
$\gamma_{Job,\,MAX(P.Salary)\rightarrow maxSal,\,MIN(P.Salary)\rightarrow minSal}$
$Payroll\ P$

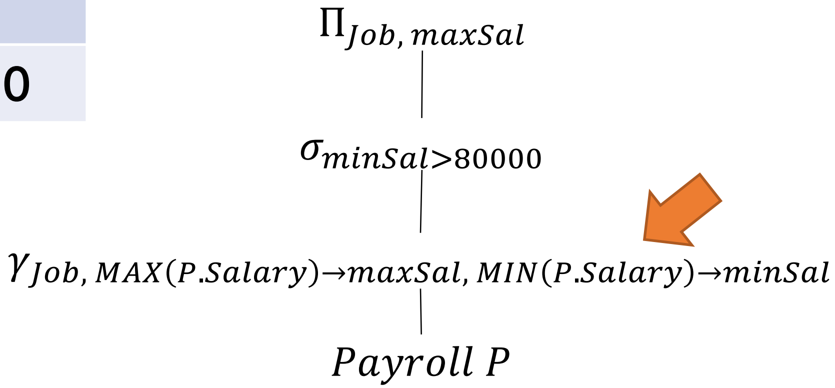
```
for each row in Payroll:
 insert into a dictionary D
  row.Job => MAX(row.Salary), MIN(row.Salary)
for each row in D:
 if (row.MIN(Salary) > 80000)
  output (row.Job, row.MAX(Salary))
```

Agg: Nested-Loop Semantics

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

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

Job	MAX(Salary)	MIN(Salary)
TA	60000	50000
Prof	**100000**	90000



$\Pi_{Job,\, maxSal}$

$\sigma_{minSal > 80000}$

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

$Payroll\ P$

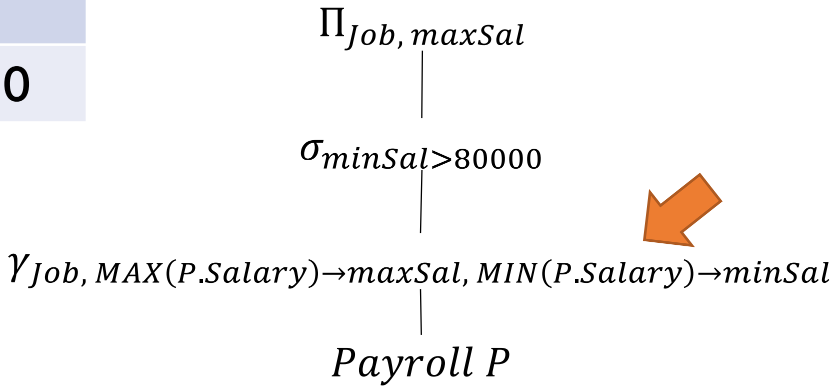
```
for each row in Payroll:
 insert into a dictionary D
  row.Job => MAX(row.Salary), MIN(row.Salary)
for each row in D:
 if (row.MIN(Salary) > 80000)
  output (row.Job, row.MAX(Salary))
```

Agg: Nested-Loop Semantics

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

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

Job	MAX(Salary)	MIN(Salary)
TA	60000	50000
Prof	100000	90000



$\Pi_{Job,\, maxSal}$

$\sigma_{minSal > 80000}$

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

$Payroll\ P$

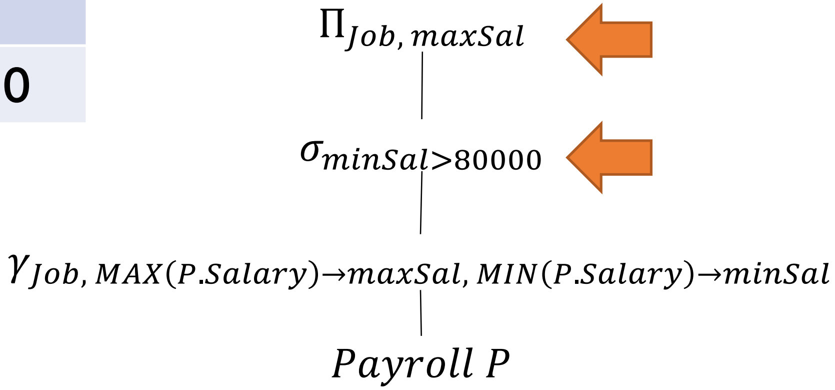
```
for each row in Payroll:
 insert into a dictionary D
  row.Job => MAX(row.Salary), MIN(row.Salary)
for each row in D:
 if (row.MIN(Salary) > 80000)
  output (row.Job, row.MAX(Salary))
```

Agg: Nested-Loop Semantics

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

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

Job	MAX(Salary)	MIN(Salary)
TA	60000	50000
Prof	100000	90000



$\Pi_{Job,\, maxSal}$

$\sigma_{minSal > 80000}$

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

$Payroll\ P$

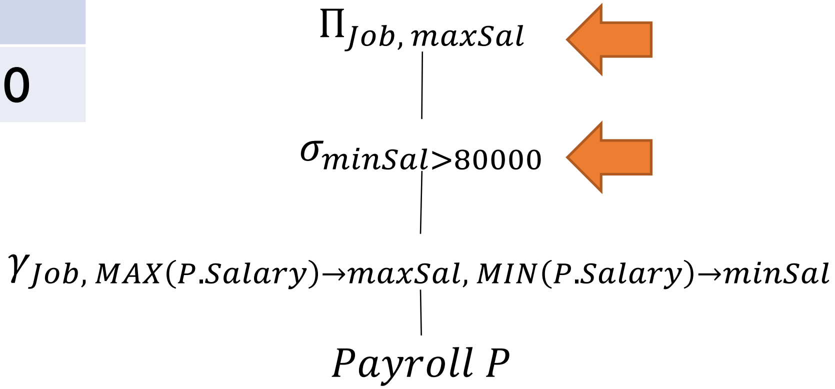
Job	MAX(Salary)

```
for each row in Payroll:
 insert into a dictionary D
  row.Job => MAX(row.Salary), MIN(row.Salary)
for each row in D:
 if (row.MIN(Salary) > 80000)
  output (row.Job, row.MAX(Salary))
```

Agg: Nested-Loop Semantics

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

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



$\Pi_{Job,\, maxSal}$

$\sigma_{minSal>80000}$

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

$Payroll\ P$

Job	MAX(Salary)	MIN(Salary)
TA	60000	50000
Prof	100000	90000

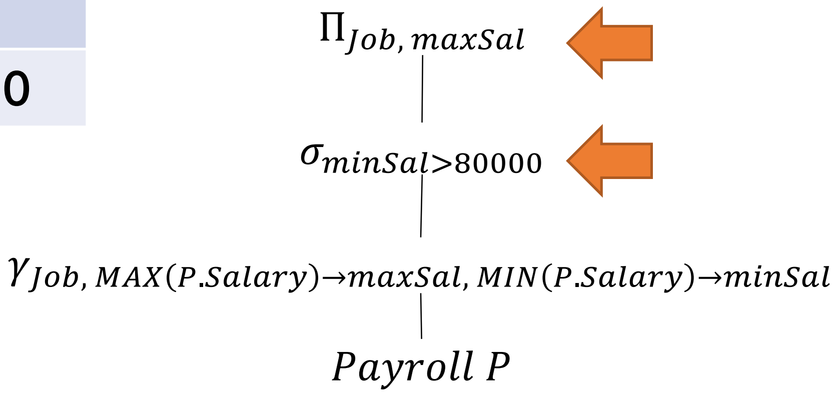
Job	MAX(Salary)

```
for each row in Payroll:
 insert into a dictionary D
  row.Job => MAX(row.Salary), MIN(row.Salary)
for each row in D:
 if (row.MIN(Salary) > 80000)
  output (row.Job, row.MAX(Salary))
```

Agg: Nested-Loop Semantics

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

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


$\Pi_{Job,\, maxSal}$

$\sigma_{minSal>80000}$

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

$Payroll\ P$

Job	MAX(Salary)	MIN(Salary)
TA	60000	50000
Prof	100000	90000

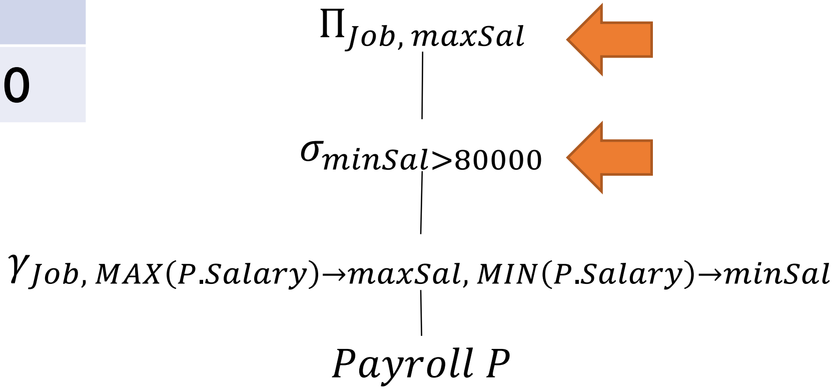
Job	MAX(Salary)
Prof	100000

```
for each row in Payroll:
 insert into a dictionary D
  row.Job => MAX(row.Salary), MIN(row.Salary)
for each row in D:
 if (row.MIN(Salary) > 80000)
  output (row.Job, row.MAX(Salary))
```

Agg: Nested-Loop Semantics

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

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



$\Pi_{Job,\, maxSal}$

$\sigma_{minSal > 80000}$

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

$Payroll\ P$

Job	MAX(Salary)	MIN(Salary)
TA	60000	50000
Prof	100000	90000

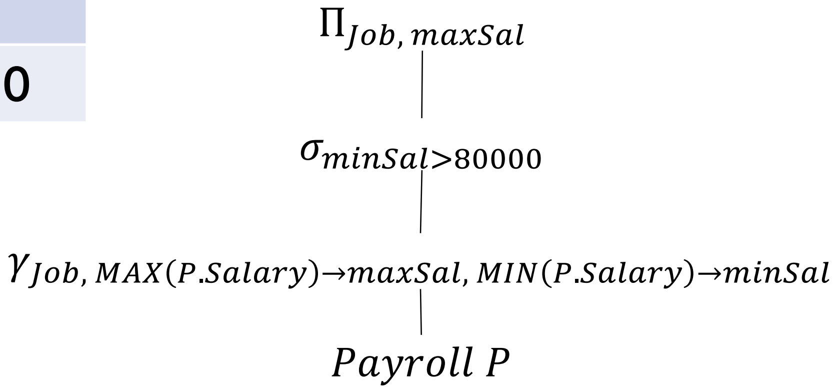
Job	MAX(Salary)
Prof	100000

```
for each row in Payroll:
 insert into a dictionary D
  row.Job => MAX(row.Salary), MIN(row.Salary)
for each row in D:
 if (row.MIN(Salary) > 80000)
  output (row.Job, row.MAX(Salary))
```

Agg: Nested-Loop Semantics

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

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



$\Pi_{Job,\, maxSal}$

$\sigma_{minSal > 80000}$

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

$Payroll\ P$

Job	MAX(Salary)	MIN(Salary)
TA	60000	50000
Prof	100000	90000

Job	MAX(Salary)
Prof	100000

```
for each row in Payroll:
 insert into a dictionary D
  row.Job => MAX(row.Salary), MIN(row.Salary)
for each row in D:
 if (row.MIN(Salary) > 80000)
  output (row.Job, row.MAX(Salary))
```

SQL and RA Vocab Summary

```
SELECT (DISTINCT) …
   FROM …
  WHERE …
  GROUP BY …
 HAVING …
  ORDER BY …
```

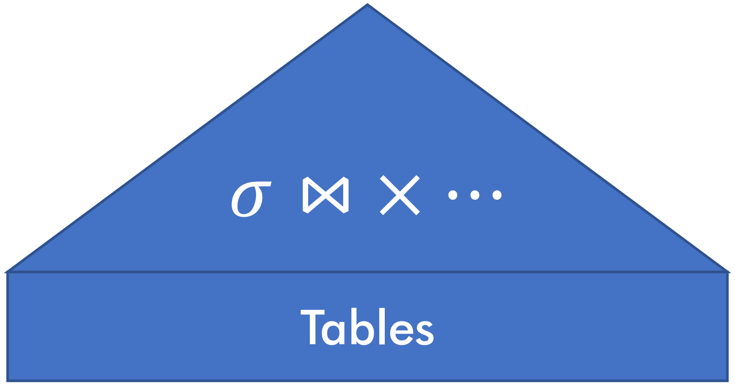
δ

Π

τ

σ

γ



SQL and RA Vocab Summary

```
SELECT (DISTINCT) …
   FROM …
  WHERE …
  GROUP BY …
 HAVING …
  ORDER BY …
```

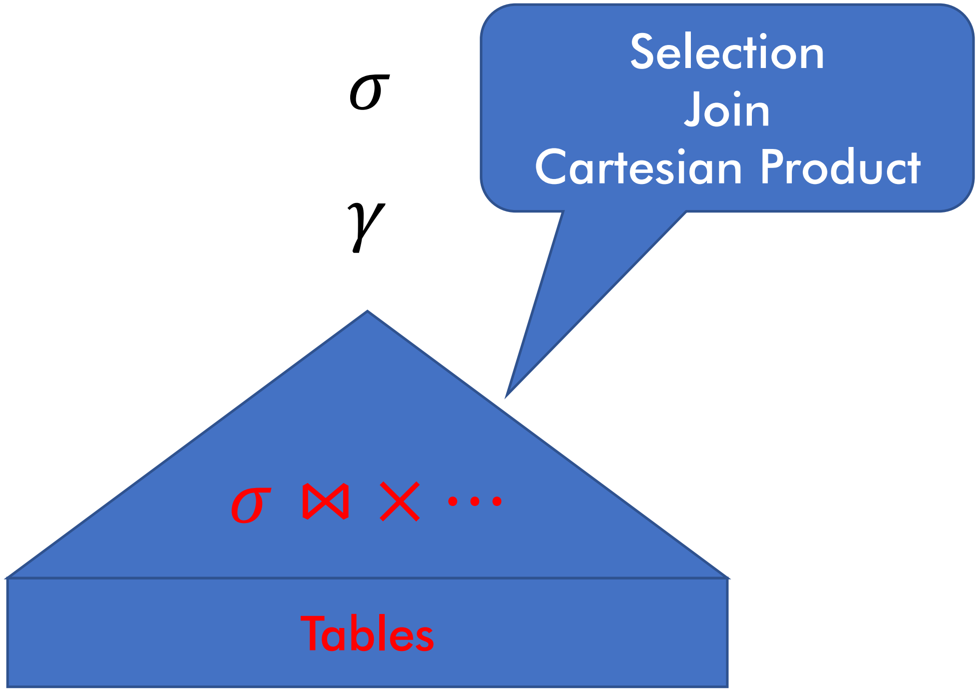
δ

Π

τ

σ

γ



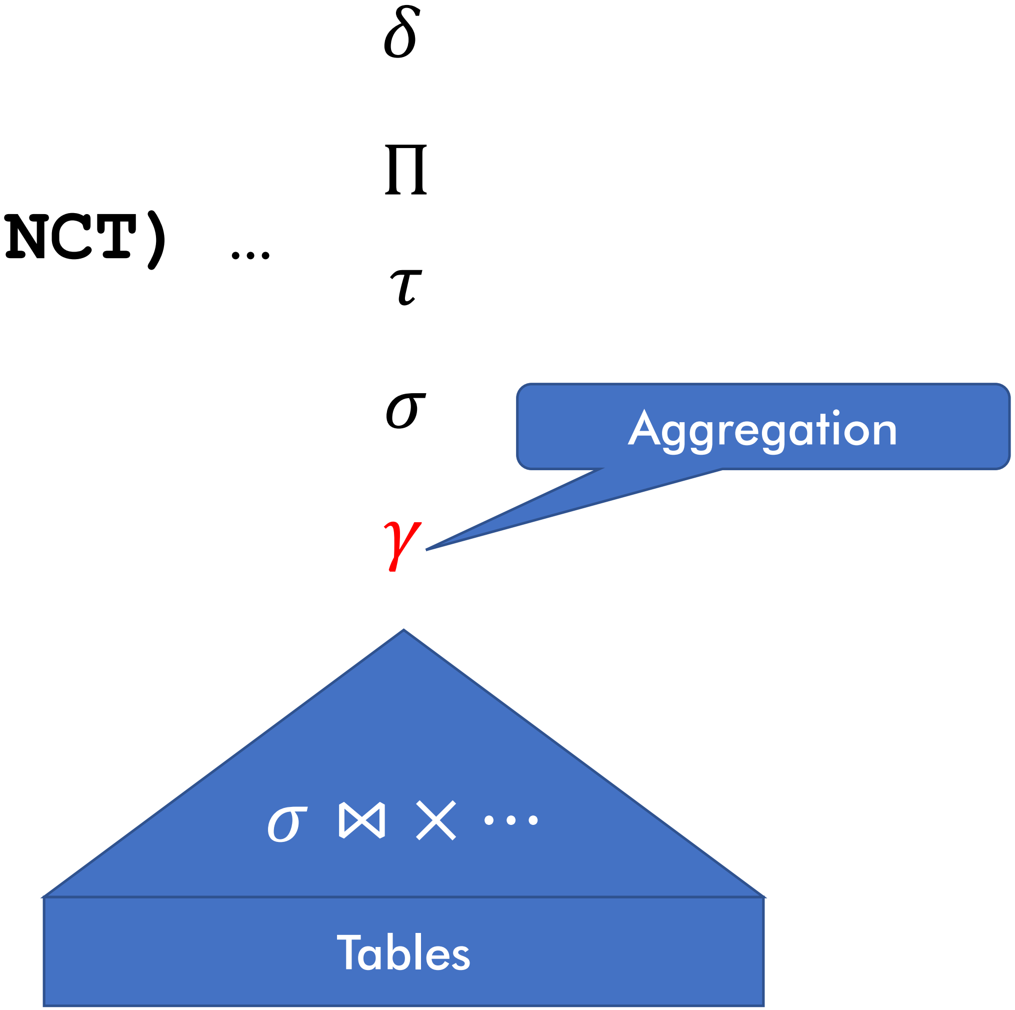
Selection
Join
Cartesian Product

$\sigma \bowtie \times \cdots$

Tables

SQL and RA Vocab Summary

```
SELECT (DISTINCT) …
  FROM …
 WHERE …
 GROUP BY …
HAVING …
 ORDER BY …
```



δ

Π

τ

σ

γ

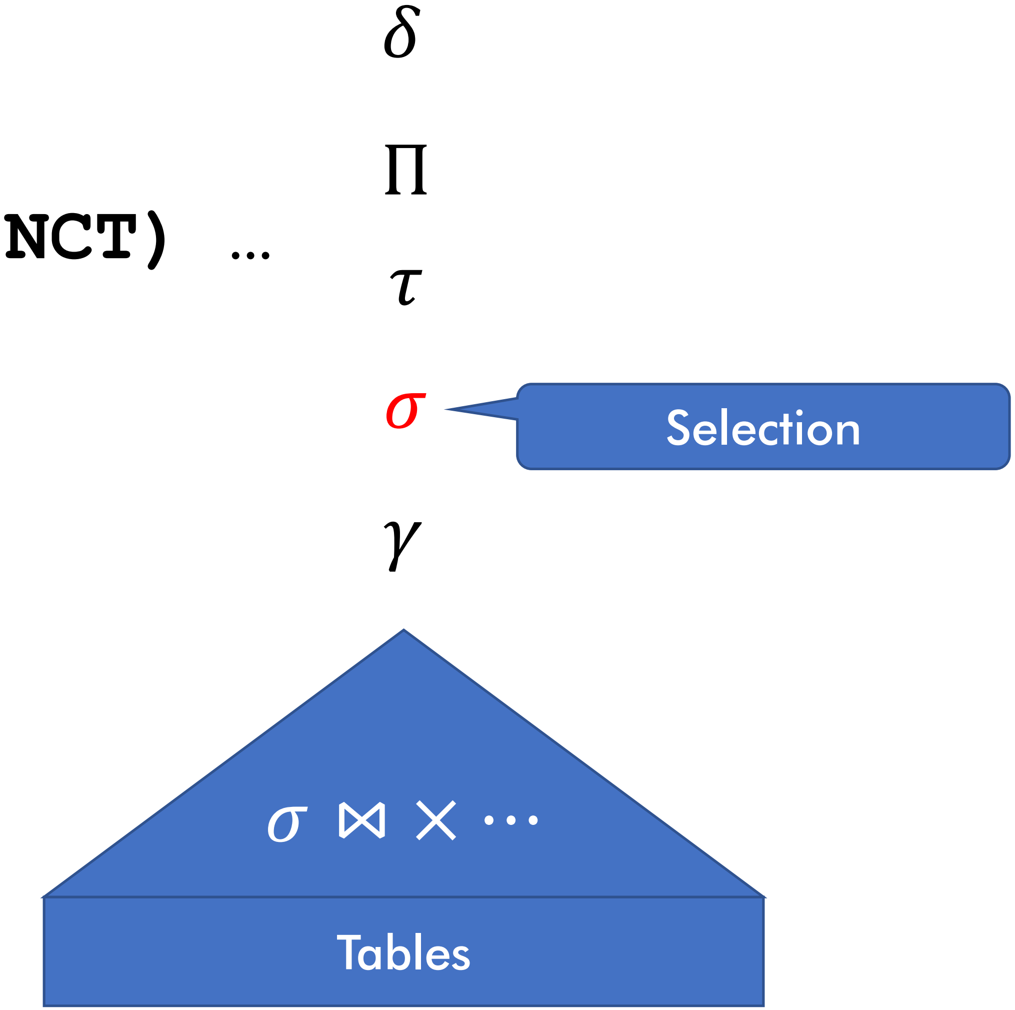
Aggregation

$\sigma \bowtie \times \cdots$

Tables

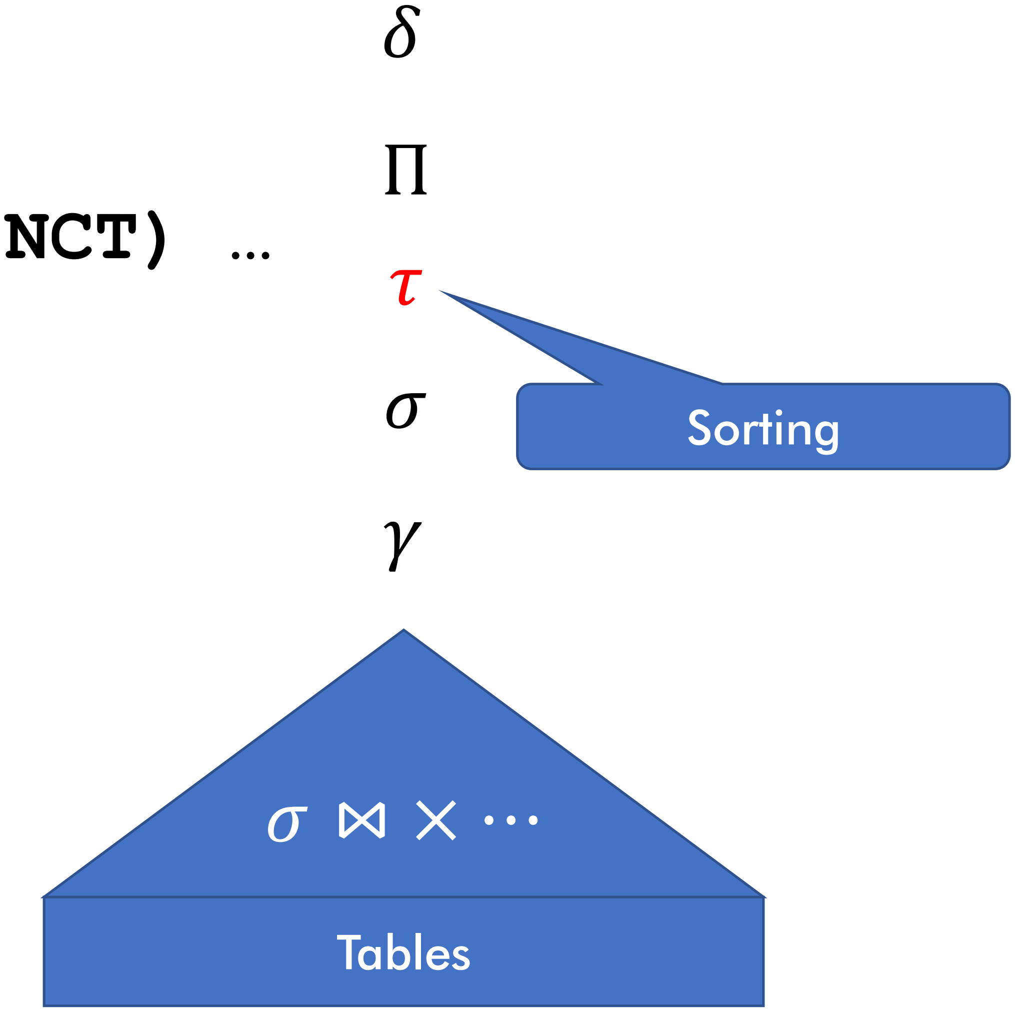
SQL and RA Vocab Summary

```
SELECT (DISTINCT) …
  FROM …
 WHERE …
 GROUP BY …
HAVING …
 ORDER BY …
```



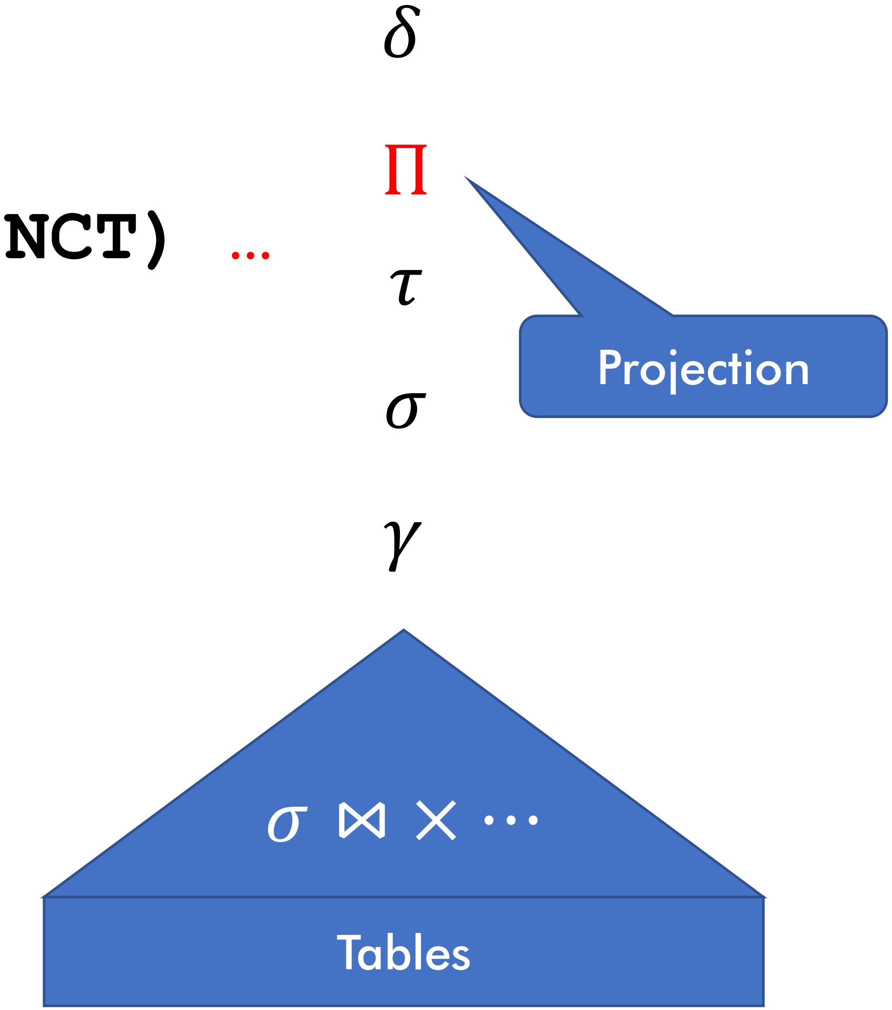
SQL and RA Vocab Summary

```
SELECT (DISTINCT) …
   FROM …
  WHERE …
  GROUP BY …
HAVING …
  ORDER BY …
```



SQL and RA Vocab Summary

```
SELECT (DISTINCT) …
  FROM …
 WHERE …
 GROUP BY …
HAVING …
 ORDER BY …
```



δ

Π

τ

σ

γ

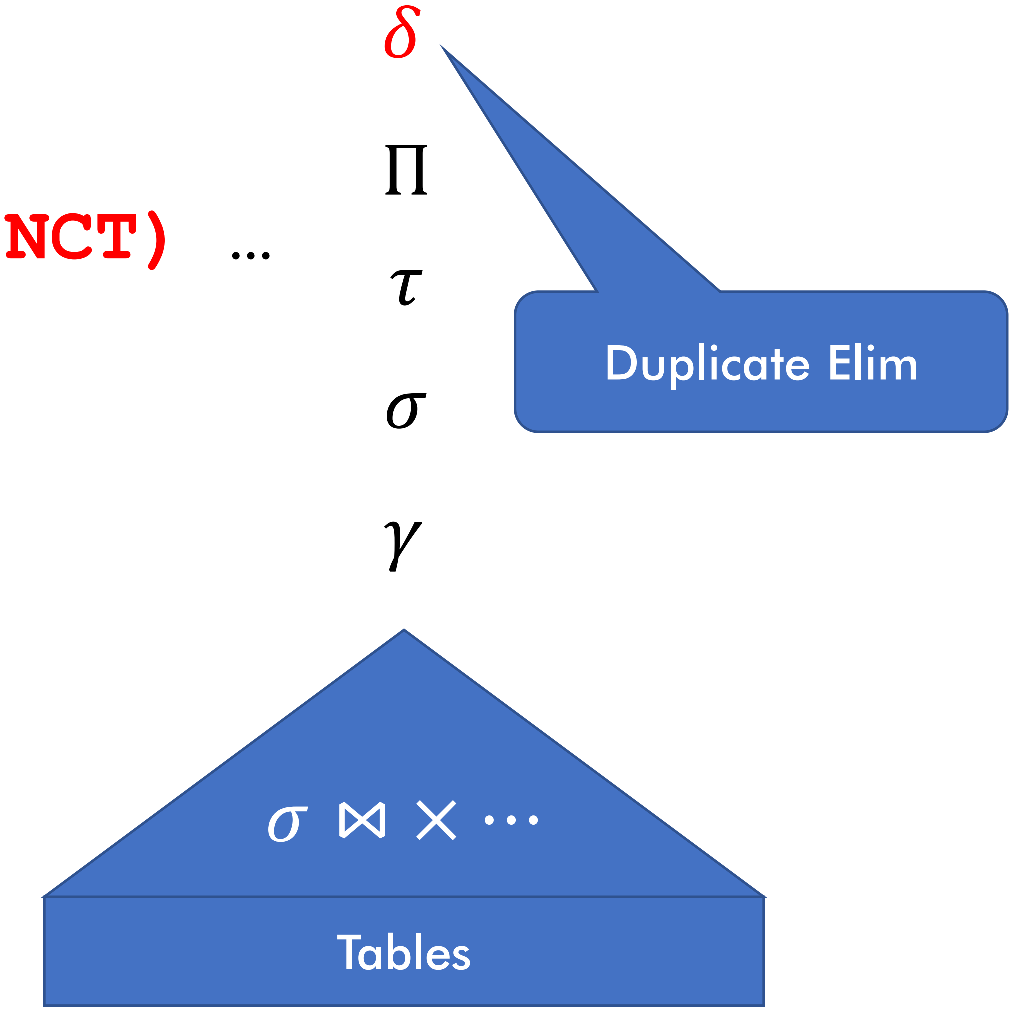
Projection

$\sigma \bowtie \times \cdots$

Tables

SQL and RA Vocab Summary

```
SELECT (DISTINCT) …
  FROM …
 WHERE …
 GROUP BY …
HAVING …
 ORDER BY …
```



SQL and RA Vocab Summary

FWGHOS™

```
SELECT (DISTINCT) …
  FROM …
 WHERE …
 GROUP BY …
HAVING …
 ORDER BY …
```

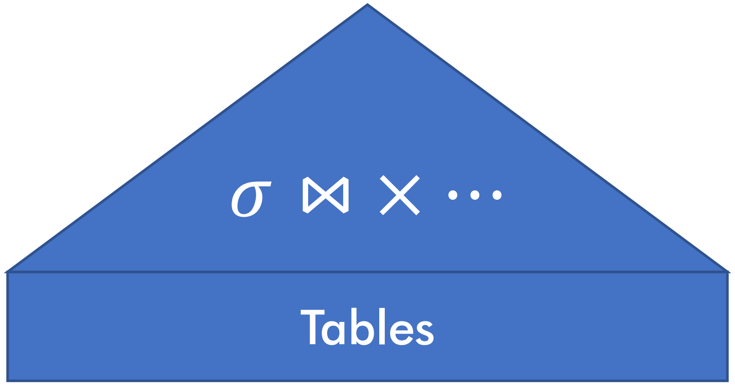
δ

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
242	**Gibbs**	**TA**	**60000**

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

The Witnessing Problem

- Also known as argmax/
- Ex: Return th ghest salary for each j

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.

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

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

SELECT, HAVING, ORDER BY can only use GROUP BY attributes or aggregates

The Witnessing Problem

- Also known as argmax/
- Ex: Return th ghest salary for each j

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.

serID	Name		
123	Jack		
345	Allison	TA	60000
567	Magda	Prof	90000
789	Dan	Prof	100000
242	Gibbs	TA	60000

This is a problem even for 'Prof' where it *seems* like Dan is the clear answer. SQL Standard says it *could be ambiguous* in general, so it is never allowed

SQLite allows it… returns a random name

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

SELECT, HAVING, ORDER BY can only use GROUP BY attributes or aggregates

The Witnessing Problem

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

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	maxima
123	Jack	TA	50000	60000
345	Allison	TA	60000	60000
567	Magda	Prof	90000	100000
789	Dan	Prof	100000	100000
242	Gibbs	TA	60000	60000

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
242	Gibbs	TA	60000	60000

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
242	Gibbs	TA	60000	60000

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

Result of the Join

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
123	Jack	TA	50000	242	Gibbs	TA	60000
345	Allison	TA	60000	123	Jack	TA	50000
345	Allison	TA	60000	345	Allison	TA	60000
345	Allison	TA	60000	242	Gibbs	TA	60000
242	Gibbs	TA	60000	123	Jack	TA	50000
242	Gibbs	TA	60000	345	Allison	TA	60000
242	Gibbs	TA	60000	242	Gibbs	TA	60000
567	Magda	Prof					
567	Magda	Prof					
789	Dan	Prof					
789	Dan	Prof					

```
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

Grouping

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
123	Jack	TA	50000	242	Gibbs	TA	60000
345	Allison	TA	60000	123	Jack	TA	50000
345	Allison	TA	60000	345	Allison	TA	60000
345	Allison	TA	60000	242	Gibbs	TA	60000
242	Gibbs	TA	60000	123	Jack	TA	50000
242	Gibbs	TA	60000	345	Allison	TA	60000
242	Gibbs	TA	60000	242	Gibbs	TA	60000
567	Magda	Prof					
567	Magda	Prof					
789	Dan	Prof					
789	Dan	Prof					

```
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

Max within groups

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	
123	Jack	TA	50000	242	Gibbs	TA	60000
345	Allison	TA	60000	123	Jack	TA	50000
345	Allison	TA	60000	345	Allison	TA	
345	Allison	TA	60000	242	Gibbs	TA	60000
242	Gibbs	TA	60000	123	Jack	TA	0000
242	Gibbs	TA	60000	345	Allison	TA	
242	Gibbs	TA	60000	242	Gibbs	TA	60000
567	Magda						
567	Magda						100
789	Dan						
789	Dan						1000

MAX(Salary)

60000

60000

60000

100000

100000

```
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

Cut down to groups

Name	Salary	Job	MAX(Salary)
Jack	50000	TA	60000
Allison	60000	TA	60000
Gibbs	60000	TA	60000
Magda	90000	Prof	100000
Dan	100000	Prof	100000

```
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)
```

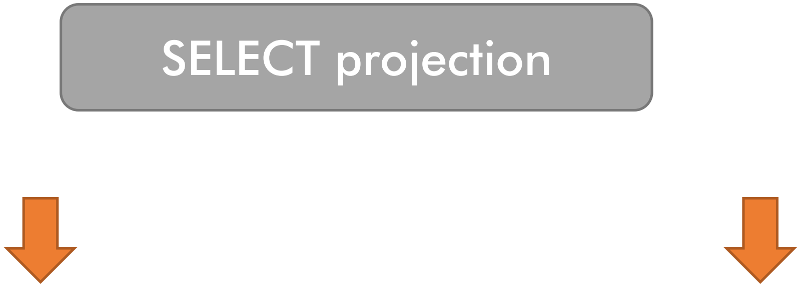
The Witnessing Problem

HAVING filter

Name	Salary	Job	MAX(Salary)
Jack	50000	TA	60000
Allison	60000	TA	60000
Gibbs	60000	TA	60000
Magda	90000	Prof	100000
Dan	100000	Prof	100000

```
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 projection

Name	Salary	Job	MAX(Salary)
Allison	60000	TA	60000
Gibbs	60000	TA	60000
Dan	100000	Prof	100000

```
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

Output

Name	MAX(Salary)
Allison	60000
Gibbs	60000
Dan	100000

Warning: what if Allison and Gibbs had the same name?

➔ Group by and select **UserID** to distinguish

```
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)
```

Takeaways

- FWGHOS™
- Combining techniques (aggregates and joins) allows you to answer complex questions (e.g. witnessing queries)
- Next week: simplifying with subqueries