Announcements

- HW 2 due tomorrow
- Office hours adjusted
  - Shana has office hours Thursday at 3:30pm, Gates 150
  - TAs and I often available by appointment
- Midterm 2 weeks from today – will be added to calendar

Goals for Today

- Recap RA
- Use SQL queries to assist other SQL queries
- Subqueries give you 99% of the tools for queries you can think of

Recap RA Operators

- These are all the operators you will see in this class
  - We’ll profile these one at a time
- θ Inner Join
- γ Grouping & Aggregation
- τ Sort
- × Cartesian Product
- ∪ Union
- δ Duplicate Elimination
- σ Selection
- ∩ Intersection
- π Projection
- ∖ Difference

Simple Example

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

```
πJob
Payroll P
```

Simple Example

```
SELECT Job
FROM Payroll;
```

```
πJob
Serve P
```

```
SELECT Job
FROM Payroll;
```
Simple Example

**Payroll**

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

\[ \pi_{Job} \text{ FROM Payroll}; \]

Simple Example

**Payroll**

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

\[ \pi_{Job} \text{ FROM Payroll}; \]

Typical Plan for a Query

Answer

\[ \text{SELECT fields FROM R, S, \ldots WHERE condition} \]

\[ \text{SELECT PROJECT JOIN Query} \]

Aggregation Order

How is aggregation processed internally?

\[ \text{SELECT Job, MAX(Salary) FROM Payroll GROUP BY Job HAVING MIN(Salary) > 80000} \]
### Aggregation RA

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

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Job**  
**maxSal**  
**minSal**

<table>
<thead>
<tr>
<th>TA</th>
<th>$60000$</th>
<th>$50000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prof</td>
<td>$100000$</td>
<td>$90000$</td>
</tr>
</tbody>
</table>

**Job**  
**maxSal**  
**minSal**

<table>
<thead>
<tr>
<th>TA</th>
<th>$60000$</th>
<th>$50000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prof</td>
<td>$100000$</td>
<td>$90000$</td>
</tr>
</tbody>
</table>

### Aggregation RA

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

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Job**  
**maxSal**  
**minSal**

<table>
<thead>
<tr>
<th>TA</th>
<th>$60000$</th>
<th>$50000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prof</td>
<td>$100000$</td>
<td>$90000$</td>
</tr>
</tbody>
</table>

**Job**  
**maxSal**  
**minSal**

<table>
<thead>
<tr>
<th>TA</th>
<th>$60000$</th>
<th>$50000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prof</td>
<td>$100000$</td>
<td>$90000$</td>
</tr>
</tbody>
</table>

### Aggregation RA

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

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Job**  
**maxSal**  
**minSal**

<table>
<thead>
<tr>
<th>TA</th>
<th>$60000$</th>
<th>$50000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prof</td>
<td>$100000$</td>
<td>$90000$</td>
</tr>
</tbody>
</table>

**Job**  
**maxSal**  
**minSal**

<table>
<thead>
<tr>
<th>TA</th>
<th>$60000$</th>
<th>$50000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prof</td>
<td>$100000$</td>
<td>$90000$</td>
</tr>
</tbody>
</table>

### Aggregation RA

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

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Job**  
**maxSal**  
**minSal**

<table>
<thead>
<tr>
<th>TA</th>
<th>$60000$</th>
<th>$50000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prof</td>
<td>$100000$</td>
<td>$90000$</td>
</tr>
</tbody>
</table>

**Job**  
**maxSal**  
**minSal**

<table>
<thead>
<tr>
<th>TA</th>
<th>$60000$</th>
<th>$50000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prof</td>
<td>$100000$</td>
<td>$90000$</td>
</tr>
</tbody>
</table>
Typical Plan for an Aggregate Query

Recap – The Witnessing Problem

• A question pattern that asks for data associated with a maxima of some value
  • Observed how to do it with grouping
  • “Self join” on values you find the maxima for
  • GROUP BY to deduplicate one side of the join
  • HAVING to compare values with respective maxima

Outline

• Witnessing via subquery
• Subquery mechanics
  • Set/bag operations
  • SELECT
  • FROM
  • WHERE/HAVING
• Decorrelation and unnesting along the way
The Witnessing Problem Simplified

- Wanted to join respective maxima
- GROUP BY technique was interesting
- Last time people suggested that we can compute the maxima first then join

```
SELECT UserID, Name, Job, 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)
```

Return the person with the highest salary for each job type

---

The Witnessing Problem - Previously

```
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

---

RA and Subqueries

```
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 organizing for

---

RA and Subqueries
The Witnessing Problem Simplified

\[
\text{SELECT} \ P1.\text{Name}, \ \text{MAX}(P2.\text{Salary}) \\
\text{FROM} \ \text{Payroll AS P1, Payroll AS P2} \\
\text{WHERE} \ P1.\text{Job} = P2.\text{Job} \\
\text{GROUP BY} \ P2.\text{Job}, P1.\text{Salary}, P1.\text{Name} \\
\text{HAVING} \ P1.\text{Salary} = \text{MAX}(P2.\text{Salary})
\]

We can compute the same thing!

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

Useful intermediate result!

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

Payroll

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

Solving a subproblem can make your life easy

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

Payroll

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

Solving subproblem can make your life easy
The Punchline about Subqueries

- Subqueries can be interpreted as **single values**
  or as **whole relations**
  - A single value (a 1x1 relation) can be returned as part of a tuple
  - A relation can be:
    - Used as input for another query
    - Checked for containment of a value

Set Operations

- SQL mimics set theory in many ways
  - Bag = duplicates allowed
  - \texttt{UNION (ALL)} \to set union (bag union)
  - \texttt{INTERSECT (ALL)} \to set intersection (bag intersection)
  - \texttt{EXCEPT (ALL)} \to set difference (bag difference)

- SQL Server Management Studio 2017
  - \texttt{INTERSECT ALL} not supported
  - \texttt{EXCEPT ALL} not supported

Subqueries in SELECT

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

\begin{verbatim}
SELECT P.Name, (SELECT AVG(P1.Salary) FROM Payroll AS P1 WHERE P.Job = P1.Job) 
FROM Payroll AS P
\end{verbatim}
Subqueries in SELECT

- Must return a single value
- Uses:
  - Compute an associated value
  - Example: For each employee, return their name and average job salary.

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

- Must return a single value
- Uses:
  - Compute an associated value
  - Example: For each employee, return their name and average job salary.

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

- Must return a single value
- Uses:
  - Compute an associated value
  - Example: For each employee, return their name and average job salary.

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

Payroll P

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

Payroll P1

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>
Subqueries in SELECT

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

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

Subqueries in SELECT

For each person find the average salary of their job

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

```sql
```

For each person find the number of cars they drive

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```sql
SELECT P.Name, COUNT(R.Car) FROM Payroll AS P, Regist AS R WHERE P.UserID = R.UserID GROUP BY P.Name
```
Subqueries in SELECT

For each person find the number of cars they drive:

```sql
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:

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

For each person find the number of cars they drive:

```sql
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
```

Syntactic sugar:

```sql
SELECT P.Name, P.Salary FROM Payroll AS P, MaxPay AS MP WHERE P.Job = MP.Job AND P.Salary = MP.Salary
```

At the risk of repeating the example:

```sql
SELECT P.Name, P.Salary FROM Payroll AS P WHERE P.UserID NOT IN (SELECT UserID FROM Regist)
```

Announcements

- HW 3 out today
- Azure credits have been issued ($75)
  • Sent to @uw.edu emails
  • Enter your @uw.edu email in sign-in
  • Post on Piazza if you have issues

Subqueries in WHERE/HAVING

• Uses:
  • ANY \(\exists\)
  • ALL \(\forall\)
  • (NOT) IN \(\notin\)
  • (NOT) EXISTS \(\emptyset=\ldots\)

Find the name and salary of people who do not drive cars:

```sql
SELECT P.Name, P.Salary FROM Payroll AS P WHERE P.UserID NOT IN (SELECT UserID FROM Regist)
```

Decorrelated!
3. Subqueries in WHERE

- SELECT ........ WHERE EXISTS (subquery);
- SELECT ........ WHERE NOT EXISTS (subquery);
- SELECT ........ WHERE attribute IN (subquery);
- SELECT ........ WHERE attribute NOT IN (subquery);
- SELECT ........ WHERE constant > ANY (subquery);
- SELECT ........ WHERE constant > ALL (subquery);

Find all companies that make some products with price < 200

Using EXISTS: EXISTS (subquery) returns true if cardinality of subquery > 0

```
SELECT DISTINCT C.cname
FROM Company C
WHERE EXISTS (SELECT * FROM Product P
WHERE C.cid = P.cid AND P.price < 200)
```

Using ANY:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE 200 > ANY (SELECT price
FROM Product P
WHERE P.price < 200)
```
3. Subqueries in WHERE

Product \((\text{pname}, \text{price}, \text{cid})\)
Company \((\text{cid}, \text{name}, \text{city})\)

Find all companies that make some products with price < 200

**Existential quantifiers**

Using ANY: \(\text{const} \geq \text{ANY} (\text{sub})\) returns true if \(\text{const} \geq \text{value}\) for at least one value in sub

**Not supported in sqlite**

Now let's unnest it:

**Existential quantifiers**

Existential quantifiers are easy! 😊

Find all companies that make some products with price < 200

**Existential quantifiers**

Find all companies that make some products with price < 200

**Existential quantifiers**

Find all companies that make some products with price < 200

**Existential quantifiers**

Now let's unnest it:

**Universal quantifiers**
3. Subqueries in WHERE

Universal quantifiers are hard! 🙄

Universal quantifiers

Using EXISTS:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE NOT EXISTS(SELECT *
FROM Product P
WHERE P.cid = C.cid and P.price >= 200)
```

Using ALL:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE 200 >= ALL(SELECT price
FROM Product P
WHERE P.cid = C.cid)
```

Not supported in sqlite

Find all companies s.t. all their products have price < 200
Subqueries in WHERE/HAVING

- Uses:
  - \( \text{ANY} \rightarrow \exists \)
  - \( \text{ALL} \rightarrow \forall \)
  - \( \text{(NOT IN} \rightarrow \{ \exists \} \in \)
  - \( \text{(NOT EXISTS} \rightarrow \{ \exists \} \in \hat{

Find the name and salary of people who do not drive cars

\[
\text{SELECT P.Name, P.Salary}
\]
\[
\text{FROM Payroll AS P}
\]
\[
\text{WHERE NOT EXISTS (SELECT *}
\]
\[
\text{FROM Regist AS R}
\]
\[
\text{WHERE P.UserID = R.UserID)}
\]

Monotonicity

A \textbf{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

A \textbf{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

A \textbf{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 \( I \).

\[
\text{SELECT P.Name, P.Car}
\]
\[
\text{FROM Payroll AS P, Regist AS R}
\]
\[
\text{WHERE P.UserID = R.UserID}
\]

Is this query 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.

**Example 1:**

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

Is this query monotone? Yes!
```

**Example 2:**

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

Is this query monotone? Yes!
```

**Example 3:**

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

Is this query monotone? No!
```

**Example 4:**

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

Is this query monotone? No!
```

**Example 5:**

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

Is this query monotone? No!
```
A Monotonic query is one that obeys the following rule where I and J are data instances and q is a query:

\[ I \subseteq J \implies q(I) \subseteq q(J) \]

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

Is this query monotone? No!

Aggregates generally are sensitive to any new tuples.

All SELECT-FROM-WHERE queries (without aggregates) are monotone

Queries with universal quantifiers are not generally monotone

You have to do something “complex” if you need to code a universal quantifier

Takeaways

- SQL is able to mirror logic over sets more or less directly
- The internal interpretation of nested queries can be quite involved
  - But our DBMS is able to derive such interpretations automagically
- We can reason about expressive power of certain queries.

Next Unit

- We are done with lectures on SQL queries!
- Up next: Database Design
  - Data modeling
  - Ethics and Security