Introduction to Data Management

SQL Subqueries

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

- HW 3 out soon
- Azure credits have been issued ($50)
  - Sent to @uw.edu emails
  - Post on Piazza if you have issues

Microsoft Azure

Accept your Azure lab assignment

You have a pending lab assignment. Please accept your assignment to get started with your course.

Accept lab assignment >

This email is generated from an unmonitored alias; please do not reply. If you have questions, please submit a request.
Recap – FWGHOS™

- Added grouping and grouped filtering to your SQL toolbox
  - GROUP BY
  - HAVING

- SQL “executed” in the following order
  1. FROM
  2. WHERE
  3. GROUP BY
  4. HAVING
  5. ORDER BY
  6. SELECT
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
Goals for Today

- Conclude our unit on SQL queries
  - After today you’ll have essentially all the building blocks of most all queries you can think of
- Use SQL queries to assist other SQL queries
Outline

- Witnessing via subquery
- Subquery mechanics
  - Set/bag operations
  - SELECT
  - FROM
  - WHERE/HAVING
- Decorrelation and unnesting along the way
- Notes about HW3
The Witnessing Problem Simplified

- Wanted to join respective maxima
  - GROUP BY technique was interesting
  - People have suggested that we can just compute the maxima first then join

<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>

Return the person (or people) with the highest salary for each job type
The Witnessing Problem Simplified

- Wanted to join respective maxima
  - GROUP BY technique was interesting
  - People have suggested that we can just compute the maxima first then join

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

Return the person (or people) with the highest salary for each job type.
The Witnessing Problem Simplified

- Wanted to join respective maxima
  - GROUP BY technique was interesting
  - People have suggested that we can just **compute the maxima first then join**

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

Return the person (or people) with the highest salary for each job type.
The Witnessing Problem Simplified

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

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)

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

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 Simplified

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
The Witnessing Problem Simplified

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

<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>

<table>
<thead>
<tr>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>
The Witnessing Problem Simplified

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

<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>

Solving a subproblem can make your life easy
# The Witnessing Problem Simplified

## 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>

## MaxPay

<table>
<thead>
<tr>
<th>Job</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA</td>
<td>60000</td>
</tr>
<tr>
<td>Prof</td>
<td>100000</td>
</tr>
</tbody>
</table>

## maxima

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Job</th>
<th>Salary</th>
<th>maxima</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Jack</td>
<td>TA</td>
<td>50000</td>
<td>60000</td>
</tr>
<tr>
<td>345</td>
<td>Allison</td>
<td>TA</td>
<td>60000</td>
<td>60000</td>
</tr>
<tr>
<td>567</td>
<td>Magda</td>
<td>Prof</td>
<td>90000</td>
<td>100000</td>
</tr>
<tr>
<td>789</td>
<td>Dan</td>
<td>Prof</td>
<td>100000</td>
<td>100000</td>
</tr>
</tbody>
</table>
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, but with duplicates
  - Instead of sets, called bags = duplicates allowed
  - **UNION (ALL)** → set union (bag union)
  - **INTERSECT (ALL)** → set intersection (bag intersection)
  - **EXCEPT (ALL)** → set difference (bag difference)

- **SQL Server Management Studio 2017**
  - INTERSECT ALL not supported
  - EXCEPT ALL not supported
Set Operations

- SQL set-like operators basically slap two queries together (not really a subquery...)

\[
(\text{SELECT } * \text{ FROM T1}) \\cup \\text{ UNION } \\cup \text{(SELECT } * \text{ FROM T2)}
\]
Subqueries in SELECT

- Must return a single value
- Uses:
  - Compute an associated value
Subqueries in SELECT

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

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

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

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

“Correlated” subquery!
Means outer table is referenced in the subquery.
Subqueries in SELECT

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

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

The Semantics of a correlated subquery are that the entire subquery is recomputed for each tuple.
Subqueries in SELECT

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>
Subqueries in SELECT

```sql
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
```

**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>

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

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>

55000
Subqueries in SELECT

```sql
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
```

### 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

```sql
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

```sql
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>

55000  55000  95000
Subqueries in SELECT

```sql
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>
Subqueries in SELECT

For each person find the average salary of their job

\[
\text{SELECT } P.\text{Name}, (\text{SELECT } \text{AVG}(P1.\text{Salary}) \text{ FROM Payroll AS P1} \text{ WHERE } P.\text{Job} = P1.\text{Job}) \text{ FROM Payroll AS P}
\]

Same (decorrelated and unnested)

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

Same? Discuss!

```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

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

0-count case not covered!

```
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

```
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
Subqueries in SELECT

For each person find the number of cars they drive

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

Still possible to decorrelate and unnest

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

- Equivalent to a WITH subquery
- Uses:
  - Solve subproblems that can be later joined/evaluated

```
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

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

Syntactic sugar
Subqueries in WHERE/HAVING

- **Uses:**
  - ANY $\rightarrow \exists$
  - ALL $\rightarrow \forall$
  - (NOT) IN $\rightarrow (\notin) \in$
  - (NOT) EXISTS $\rightarrow (\emptyset = \ldots) \emptyset \neq \ldots$
Subqueries in WHERE/HAVING

- **Uses:**
  - \( \text{ANY} \rightarrow \exists \)
  - \( \text{ALL} \rightarrow \forall \)
  - \( \text{(NOT) IN} \rightarrow (\emptyset) \in \)
  - \( \text{(NOT) EXISTS} \rightarrow (\emptyset = ...) \emptyset \neq ... \)

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

```sql
SELECT P.Name, P.Salary
FROM Payroll AS P
WHERE NOT EXISTS (SELECT *
                FROM Regist AS R
                WHERE P.UserID = R.UserID)
```
Subqueries in WHERE/HAVING

- Uses:
  - ANY $\rightarrow \exists$
  - ALL $\rightarrow \forall$
  - (NOT) IN $\rightarrow (\notin) \in$
  - (NOT) EXISTS $\rightarrow (\emptyset = \ldots) \emptyset \neq \ldots$

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

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

Decorrelated!
```
Recap

- Subquery in SELECT: return single value

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

"Correlated" subquery!
Means outer table is referenced in the subquery.

The Semantics of a correlated subquery are that the entire subquery is recomputed for each tuple
Subquery in FROM: return any size relation

Product (pname, price, cid)
Company (cid, cname, city)

Find all products whose price is > 20 and < 500

```
SELECT X.pname
FROM (SELECT *
    FROM Product AS Y
    WHERE price > 20) as X
WHERE X.price < 500
```

Try to unnest this query!

Side note: This is not a correlated subquery. X does not appear in the inner query.
Recap

- Usually best to avoid nested queries if trying for speed

- Be careful of semantics of nested queries
  - Correlated vs. decorrelated

- Think about edge cases
  - Zero matches
  - Null values
Subqueries in WHERE

- SELECT .......... WHERE EXISTS (sub);
- SELECT .......... WHERE NOT EXISTS (sub);
- SELECT .......... WHERE attribute IN (sub);
- SELECT .......... WHERE attribute NOT IN (sub);
- SELECT .......... WHERE value > ANY (sub);
- SELECT .......... WHERE value > ALL (sub);
Subqueries in WHERE

- **Existential quantifier:**
  - Indicates the existence of at least one element

- **Universal quantifiers:**
  - Indicated a property for all elements

- **Look to mathematics for more examples:**
  - [https://sites.math.washington.edu/~aloveles/Math300Summer2011/m300Quantifiers.pdf](https://sites.math.washington.edu/~aloveles/Math300Summer2011/m300Quantifiers.pdf)

\[ \forall x \in A, P(x) \quad \text{For all } x \text{ in } A, P(x) \text{ is true.} \]

\[ \exists x \in A, P(x) \quad \text{There exists some } x \text{ in } A \text{ such that } P(x) \text{ is true.} \]
3. Subqueries in WHERE

Product (pname, price, cid)
Company (cid, cname, city)

Find all companies that make some products with price < 200
Find all companies that make some products with price < 200

Slight rewording:
Return all companies such that there exists some product they make with price < 200
3. Subqueries in WHERE

Find all companies that make some products with price < 200

Using EXISTS:

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

Product (pname, price, cid)
Company (cid, cname, city)

Find all companies that make **some** products with price < 200

Using **IN**

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

Product (pname, price, cid)
Company (cid, cname, city)

Find all companies that make some products with price < 200

Using IN

```
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid IN (SELECT P.cid
                 FROM Product P
                 WHERE P.price < 200)
```

Existential quantifiers

Decorrelated!
3. Subqueries in WHERE

Find all companies that make some products with price < 200

Using **ANY**:

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

Product (pname, price, cid)
Company (cid, cname, city)

Find all companies that make some products with price < 200

Using ANY:

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

ANY not supported in sqlite
3. Subqueries in WHERE

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

Find all companies that make some products with price < 200

Now let’s unnest it:

```
SELECT DISTINCT C.cname
FROM Company C, Product P
WHERE C.cid = P.cid and P.price < 200
```
3. Subqueries in WHERE

Product (pname, price, cid)
Company (cid, cname, city)

Find all companies that make some products with price < 200

Existential quantifiers

Now let’s un nest it:

```
SELECT DISTINCT C.cname
FROM Company C, Product P
WHERE C.cid = P.cid and P.price < 200
```

Existential quantifiers are easy! 😊
3. Subqueries in WHERE

Product (pname, price, cid)
Company (cid, cname, city)

Find all companies s.t. all their products have price < 200

same as:

Find all companies that make only products with price < 200
3. Subqueries in WHERE

Product (pname, price, cid)
Company (cid, cname, city)

Find all companies s.t. all their products have price < 200

same as:

Find all companies that make only products with price < 200
3. Subqueries in WHERE

Product (pname, price, cid)
Company (cid, cname, city)

Find all companies s.t. all their products have price < 200

same as:

Find all companies that make only products with price < 200

Universal quantifiers are hard! 😞
3. Subqueries in \textit{WHERE}

Product (\textit{pname}, \textit{price}, \textit{cid})

Company (\textit{cid}, \textit{cname}, \textit{city})

Find all companies s.t. all their products have price < 200

Use the math property,

For all company products, price < 200

equivalent to:

There \textit{does not exist} some company product where price $\geq$ 200
3. Subqueries in WHERE

Product (pname, price, cid)
Company (cid, cname, city)

Find all companies s.t. all their products have price < 200

1. Find the other companies that make some product ≥ 200

```sql
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid IN (SELECT P.cid
FROM Product P
WHERE P.price >= 200)
```
3. Subqueries in WHERE

Product (pname, price, cid)
Company (cid, cname, city)

Find all companies s.t. all their products have price < 200

1. Find the other companies that make some product ≥ 200

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

2. Find all companies s.t. all their products have price < 200 (negate previous query)

```
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid NOT IN (SELECT P.cid
                    FROM Product P
                    WHERE P.price >= 200)
```
3. Subqueries in WHERE

Product (pname, price, cid)
Company (cid, cname, city)

Find all companies s.t. all their products have price < 200

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)
```
3. Subqueries in WHERE

Product (pname, price, cid)
Company (cid, cname, city)

Find all companies s.t. all their products have price < 200

Using \textit{ALL}:

\begin{verbatim}
SELECT DISTINCT C.cname
FROM Company C
WHERE 200 >= ALL (SELECT price
FROM Product P
WHERE P.cid = C.cid)
\end{verbatim}
3. Subqueries in WHERE

Product (pname, price, cid)
Company (cid, cname, city)

Find all companies s.t. all their products have price < 200

Using **ALL**: 

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

**Universal quantifiers**

**ALL not supported in sqlite**
Could we ever encode a universal quantifier with a SELECT-FROM-WHERE query with no subqueries or aggregates?
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 $I$.

In other words, adding more tuples to the input table never removes tuples from the output on the next query.
A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

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

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

Monotone

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

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

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

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

Is this query monotone? Yes!

```
SELECT  P.Name,  P.Car
FROM    Payroll AS P,  Regist AS R
WHERE   P.UserID = R.UserID
```
A **Monotonic** query is one that obeys the following rule where \( I \) and \( J \) are data instances and \( q \) is a query:

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

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

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

Is this query monotone? **Yes!**
A **Monotonic** query is one that obeys the following rule where I and J are data instances and q is a query:

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

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

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

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

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

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

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

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

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

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

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

Is this query monotone? No!

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

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.

\[
\text{SELECT} \quad \text{P.Job, COUNT(\ast)} \\
\text{FROM} \quad \text{Payroll AS P} \\
\text{GROUP BY} \quad \text{P.Job}
\]

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.

**SELECT**  
\[ P.Job, \ \text{COUNT}(*) \]  
**FROM**  
Payroll AS P  
**GROUP BY**  
P.Job

Is this query monotone? **No!**
Monotonicity

**Monotone**

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

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

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

Select

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

Is this query monotone? **No!**

Aggregates generally are sensitive to any new tuples

---

Subqueries
Monotone Queries

- **Theorem**: If Q is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it is monotone.
Monotone Queries

- **Theorem**: If Q is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it is monotone.

- **Proof**: We use the nested loop semantics: if we insert a tuple in a relation \( R_i \), this will not remove any tuples from the answer.

```
SELECT  a_1, a_2, ..., a_k
FROM     R_1 AS x_1, R_2 AS x_2, ..., R_n AS x_n
WHERE    Conditions
```

```
for x_1 in R_1 do
  for x_2 in R_2 do
    ...
    for x_n in R_n do
      if Conditions
        output (a_1, ..., a_k)
```
Monotonicity

- **Consequence:** If a query is not monotonic, then we cannot write it as a SELECT-FROM-WHERE query without nested subqueries or aggregates.

- Queries with universal quantifiers are not generally monotone
- You have to do something “complex” if you need to code a universal quantifier
Queries that must be nested

- Queries with universal quantifiers or with negation

- Queries that use aggregates in certain ways
  - `sum(..)` and `count(*)` are **NOT** monotone, because they do not satisfy set containment
  - `select count(*) from R` is **not** monotone!
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.
We are done with lectures on SQL queries!

Up next:
- Another query language
- Data modeling
- Ethics and Security