



# Recap

- In the first 2 lectures, you have seen almost all of the SQL that people use on a day-to-day basis
  - SELECT-FROM-WHERE
  - *“I’m a PhD student in databases and I don’t remember the last time I wrote a SQL query with a GROUP BY”*
    - Someone in the DB Lab
- In the 5 previous lectures, you have seen almost all of the SQL that people will ever touch
  - Analytical queries
  - CREATE TABLE ...

# 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

# 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

# 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

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

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

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 (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**


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 (or people) with the highest salary for each job type

# The Witnessing Problem Simplified

**MaxPay**

Job	Salary
TA	60000
Prof	100000



```
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
```

We can compute  
the same thing!

```
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
```

```
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

Useful intermediate result!

```
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
```

**Payroll**

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

**MaxPay**

Job	Salary
TA	60000
Prof	100000

# 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
```

Solving a subproblem  
can make your life easy

**Payroll**

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

**MaxPay**

Job	Salary
TA	60000
Prof	100000

# The Witnessing Problem Simplified

**Payroll**

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

**MaxPay**

Job	Salary
TA	60000
Prof	100000



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



# 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...)

```
(SELECT * FROM T1)  
UNION  
(SELECT * 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

```
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!

**Definition:** A subquery that references an attribute from the outer query  
(Payroll P is in the outer query, P1 is in the inner query)

# 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
```

**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**



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



# 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**



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


**Payroll P1**

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

# 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**



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

**Payroll P1**

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

# Subqueries in SELECT

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

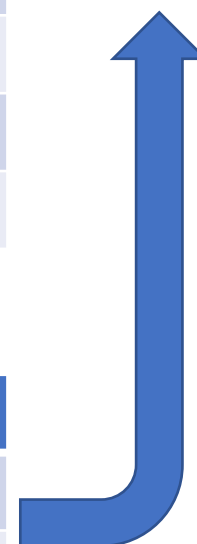
**Payroll P**

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

**Payroll P1**

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

55000



# 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**

UserID	Name	Job	Salary	
123	Jack	TA	50000	55000
→ 345	Allison	TA	60000	
567	Magda	Prof	90000	
789	Dan	Prof	100000	

# 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**

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

55000



**Payroll P1**

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

# Subqueries in SELECT

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

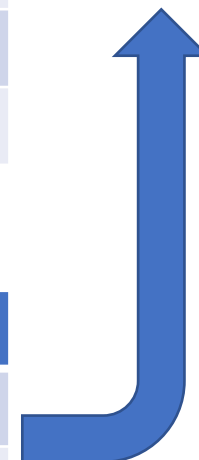
**Payroll P**

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

**Payroll P1**

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

55000



# 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**

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

**Payroll P1**

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

55000

55000



# 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**

UserID	Name	Job	Salary	
123	Jack	TA	50000	55000
345	Allison	TA	60000	55000
→ 567	Magda	Prof	90000	95000
789	Dan	Prof	100000	

**Payroll P1**

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



# 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**

UserID	Name	Job	Salary	
123	Jack	TA	50000	55000
345	Allison	TA	60000	55000
567	Magda	Prof	90000	95000
→ 789	Dan	Prof	100000	95000

**Payroll P1**

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

# Subqueries in SELECT

For each person find the average salary of their job

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



Same (decorrelated and unnested)

```
SELECT P1.Name, AVG(P2.Salary)
FROM Payroll AS P1, Payroll AS P2
WHERE P1.Job = P2.Job
GROUP BY P1.UserID, P1.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
```



Same? **Discuss!**

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

Name	Count
Jack	1
Allison	0
Magda	2
Dan	0

0-count

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

Name	Count
Jack	1
Magda	2

# 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.UserID, 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
```

Syntactic sugar

```
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
```



# 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

# Review of Subqueries in SELECT/FROM

## ■ SELECT

- The subquery must return a **single value**
- The subquery is **correlated** if it references a relation in the outer query.
  - Correlated subqueries run again for every tuple in the outer query

## ■ FROM

- Can write subquery in the style of:  
**FROM (<subquery>) AS alias**
- The normal semantics in the FROM clause apply! Be careful of **cross products** and don't forget join predicates.

# 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/HAVING

## ■ Uses:

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

Mathematical notation:

$\exists$  "There exists"

$\forall$  "For all"

$\in$  "Is contained in"

$\emptyset$  "The empty set"

# Subqueries in WHERE/HAVING

But first! Without subqueries:

Find the name and salary of people who **do** drive cars

# Subqueries in WHERE/HAVING

UserID	Car
123	Charger
567	Civic
567	Pinto

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

But first! Without subqueries:

Find the name and salary of people who **do** drive cars

# Subqueries in WHERE/HAVING

UserID	Car
123	Charger
567	Civic
567	Pinto

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

But first! Without subqueries:

Find the name and salary of people who **do** drive cars

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

# Subqueries in WHERE/HAVING

UserID	Car
123	Charger
567	Civic
567	Pinto

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

But first! Without subqueries:

Find the name and salary of people who **do** drive cars

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

Name	Salary
Jack	50000
Magda	90000
Magda	90000



# Subqueries in WHERE/HAVING

UserID	Car
123	Charger
567	Civic
567	Pinto

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

But first! Without subqueries:

Find the name and salary of people who **do** drive cars

```
SELECT DISTINCT P.Name, P.Salary  
FROM Payroll AS P, Regist AS R  
WHERE P.UserID = R.UserID
```

Name	Salary
Jack	50000
Magda	90000

# Subqueries in WHERE/HAVING

EXISTS (subquery) evaluates to **true** when subquery returns **at least one tuple**

- SELECT ..... WHERE EXISTS (sub);  $\emptyset \neq \{sub\}$

# Subqueries in WHERE/HAVING

EXISTS (subquery) evaluates to **true** when subquery returns **at least one tuple**

- SELECT ..... WHERE EXISTS (sub);
- SELECT ..... WHERE NOT EXISTS (sub);     $\emptyset = \{sub\}$

# Subqueries in WHERE/HAVING

- SELECT ..... WHERE EXISTS (sub);  $\emptyset \neq \{sub\}$
- SELECT ..... WHERE NOT EXISTS (sub);

Find the name and salary of people who **do** drive cars

```
SELECT P.Name, P.Salary
FROM Payroll AS P
WHERE EXISTS (SELECT *
                FROM Regist AS R
                WHERE P.UserID = R.UserID)
```

# Subqueries in WHERE/HAVING

- SELECT ..... WHERE EXISTS (sub);  $\emptyset \neq \{sub\}$
- SELECT ..... WHERE NOT EXISTS (sub);

Find the name and salary of people who **do** drive cars

```
SELECT P.Name, P.Salary
FROM Payroll AS P
WHERE EXISTS (SELECT *
                FROM Regist AS R
                WHERE P.UserID = R.UserID)
```

One single Boolean predicate  
that is true or false

# For-each semantics

When in doubt, go back to for-each semantics,  
starting with FROM clause in outer query!

**Payroll P**



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

**Regist R**

UserID	Car
123	Charger
567	Civic
567	Pinto

```
SELECT P.Name, P.Salary
FROM Payroll AS P
WHERE EXISTS (SELECT *
              FROM Regist AS R
              WHERE P.UserID = R.UserID)
```

# For-each semantics

```
SELECT P.Name, P.Salary
  FROM Payroll AS P
 WHERE EXISTS (SELECT *
               FROM Regist AS R
               WHERE P.UserID = R.UserID)
```

**Payroll P**



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

**Regist R**

UserID	Car
123	Charger
567	Civic
567	Pinto

**Output so far**

# For-each semantics

```
SELECT P.Name, P.Salary
  FROM Payroll AS P
 WHERE EXISTS (SELECT *
               FROM Regist AS R
               WHERE P.UserID = R.UserID)
```

**Payroll P**



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

**Regist R**

UserID	Car
123	Charger
567	Civic
567	Pinto

```
(SELECT *
 FROM Regist AS R
 WHERE 123 = R.UserID)
```

**Output so far**



# For-each semantics

```
SELECT P.Name, P.Salary
FROM Payroll AS P
WHERE EXISTS (SELECT *
              FROM Regist AS R
              WHERE P.UserID = R.UserID)
```

**Payroll P**



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

**Regist R**

UserID	Car
123	Charger
567	Civic
567	Pinto

```
(SELECT *
FROM Regist AS R
WHERE 123 = R.UserID)
```


returns one tuple, so EXISTS  
(subquery) is true for the Jack  
tuple

**Output so far**

# For-each semantics

```
SELECT P.Name, P.Salary
  FROM Payroll AS P
 WHERE EXISTS (SELECT *
               FROM Regist AS R
               WHERE P.UserID = R.UserID)
```

**Payroll P**



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

**Regist R**

UserID	Car
123	Charger
567	Civic
567	Pinto

```
(SELECT *
 FROM Regist AS R
 WHERE 123 = R.UserID)
```

returns one tuple, so EXISTS  
(subquery) is true for the Jack  
tuple

**Output so far**

Name	Salary
Jack	50000

# For-each semantics

```
SELECT P.Name, P.Salary
FROM Payroll AS P
WHERE EXISTS (SELECT *
              FROM Regist AS R
              WHERE P.UserID = R.UserID)
```

**Payroll P**

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



**Regist R**

UserID	Car
123	Charger
567	Civic
567	Pinto

```
(SELECT *
FROM Regist AS R
WHERE 345 = R.UserID)
```

returns nothing, so EXISTS  
(subquery) is false for Allison

**Output so far**

Name	Salary
Jack	50000

# For-each semantics

```
SELECT P.Name, P.Salary
  FROM Payroll AS P
 WHERE EXISTS (SELECT *
               FROM Regist AS R
               WHERE P.UserID = R.UserID)
```

**Payroll P**

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

**Regist R**

UserID	Car
123	Charger
567	Civic
567	Pinto

(SELECT \*  
FROM Regist AS R  
WHERE 567 = R.UserID)  
returns two tuples, so EXISTS  
(subquery) is true for the Magda  
tuple

**Output so far**

Name	Salary
Jack	50000
Magda	90000

# For-each semantics

```
SELECT P.Name, P.Salary
FROM Payroll AS P
WHERE EXISTS (SELECT *
              FROM Regist AS R
              WHERE P.UserID = R.UserID)
```

**Payroll P**

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



**Regist R**

UserID	Car
123	Charger
567	Civic
567	Pinto

```
(SELECT *
FROM Regist AS R
WHERE 789 = R.UserID)
```

returns nothing, so EXISTS  
(subquery) is false for Dan

**Output so far**

Name	Salary
Jack	50000
Magda	90000

# Subqueries in WHERE/HAVING

- SELECT ..... WHERE EXISTS (sub);  $\emptyset \neq \{sub\}$
- SELECT ..... WHERE NOT EXISTS (sub);

Find the name and salary of people who **do** drive cars

```
SELECT P.Name, P.Salary
FROM Payroll AS P
WHERE EXISTS (SELECT *
                FROM Regist AS R
                WHERE P.UserID = R.UserID)
```

# Subqueries in WHERE/HAVING

NOT EXISTS (subquery) evaluates to **true** when subquery returns **no tuples**

- SELECT ..... WHERE EXISTS (sub);
- SELECT ..... WHERE NOT EXISTS (sub);  $\emptyset = \{sub\}$

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

```
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

NOT EXISTS (subquery) evaluates to **true** when subquery returns **no tuples**

- SELECT ..... WHERE EXISTS (sub);
- SELECT ..... WHERE NOT EXISTS (sub);  $\emptyset = \{sub\}$

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

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


Person P s.t.  
 $\emptyset = \{\text{cars P drives}\}$



# For-each semantics

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

**Payroll P**



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

**Regist R**

UserID	Car
123	Charger
567	Civic
567	Pinto

```
(SELECT *
 FROM Regist AS R
 WHERE 345 = R.UserID)
```

**Output**

Name	Salary
------	--------

# For-each semantics

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

**Payroll P**



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

**Regist R**

UserID	Car
123	Charger
567	Civic
567	Pinto

```
(SELECT *
 FROM Regist AS R
 WHERE 345 = R.UserID)
```

returns nothing, so NOT EXISTS (subquery) is true for the Allison tuple

**Output**

Name	Salary
Allison	60000

# For-each semantics

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

**Payroll P**

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

**Regist R**

UserID	Car
123	Charger
567	Civic
567	Pinto

**Output**

Name	Salary
Allison	60000
Dan	100000

# Subqueries in WHERE/HAVING

NOT EXISTS (subquery) evaluates to **true** when subquery returns **no tuples**

- SELECT ..... WHERE EXISTS (sub);
- SELECT ..... WHERE NOT EXISTS (sub);  $\emptyset = \{sub\}$

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

```
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

NOT EXISTS (subquery) evaluates to **true** when subquery returns **no tuples**

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

Exercise:

“Find the names of people who are the **ONLY** employee with their job in the Payroll table.”

# Subqueries in WHERE/HAVING

NOT EXISTS (subquery) evaluates to **true** when subquery returns **no tuples**

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

Exercise:

“Find the names of people who are the **ONLY** employee with their job in the Payroll table.”

May be easier if we re-write:

“Find the names of people where there does not exist another person with the same job.”

# Subqueries in WHERE/HAVING

NOT EXISTS (subquery) evaluates to **true** when subquery returns **no tuples**

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

Exercise:

“Find the names of people where there does not exist another person with the same job.”

# Subqueries in WHERE/HAVING

NOT EXISTS (subquery) evaluates to **true** when subquery returns **no tuples**

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

Exercise:

“Find the names of people where there does not exist another person with the same job.”

```
SELECT P.Name
FROM Payroll AS P
WHERE NOT EXISTS (SELECT *
                    FROM Payroll AS P1
                    WHERE P.Job = P1.Job AND
                           P.UserID != P1.UserID )
```



# Subqueries in WHERE/HAVING

- SELECT ..... WHERE EXISTS (sub);
- SELECT ..... WHERE NOT EXISTS (sub);
- SELECT ..... WHERE attribute IN (sub);  $e \in \{sub\}$
- SELECT ..... WHERE attribute NOT IN (sub);  $e \notin \{sub\}$

# Subqueries in WHERE/HAVING

**attribute** IN (subquery) evaluates to **true** when value of attribute matches some result in (subquery)

- SELECT ..... WHERE EXISTS (sub);
- SELECT ..... WHERE NOT EXISTS (sub);
- SELECT ..... WHERE **attribute** IN (sub);  $e \in \{sub\}$
- SELECT ..... WHERE **attribute** NOT IN (sub);  $e \notin \{sub\}$

Find the name and salary of people who **do** drive cars

```
SELECT P.Name, P.Salary
FROM Payroll AS P
WHERE P.UserID IN (SELECT UserID
                    FROM Regist)
```

# Subqueries in WHERE/HAVING

**attribute** NOT IN (subquery) evaluates to **true** when value of attribute matches nothing in (subquery)

- SELECT ..... WHERE EXISTS (sub);
- SELECT ..... WHERE NOT EXISTS (sub);
- SELECT ..... WHERE **attribute** IN (sub);  $e \in \{sub\}$
- SELECT ..... WHERE **attribute** NOT IN (sub);  $e \notin \{sub\}$

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)
```

# Subqueries in WHERE/HAVING

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

Our NOT EXISTS method:

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

Our NOT IN method:

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

# Subqueries in WHERE/HAVING

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

Our NOT EXISTS method:

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

Our NOT IN method:

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

Decorrelated! Our EXISTS version was a correlated subquery

# For-each semantics

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

**Payroll P**



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

**Regist R**

UserID	Car
123	Charger
567	Civic
567	Pinto

```
(SELECT UserID
FROM Regist)
```

returns

UserID
123
567
567

**Output so far**

# For-each semantics

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

**Payroll P**

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



**Regist R**

UserID	Car
123	Charger
567	Civic
567	Pinto

(SELECT UserID  
FROM Regist)

returns

UserID
123
567
567

**Output so far**

Name	Salary
Allison	60000

# For-each semantics

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

**Payroll P**

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

**Regist R**

UserID	Car
123	Charger
567	Civic
567	Pinto

(SELECT UserID  
FROM Regist)

returns

UserID
123
567
567

**Output so far**

Name	Salary
Allison	60000
Dan	100000



# Subqueries in WHERE/HAVING

- SELECT ..... WHERE EXISTS (sub);  $\emptyset \neq \{sub\}$
- SELECT ..... WHERE NOT EXISTS (sub);  $\emptyset = \{sub\}$
- SELECT ..... WHERE attribute IN (sub);  $e \in \{sub\}$
- SELECT ..... WHERE attribute NOT IN (sub);  $e \notin \{sub\}$

Now, on to quantifier logic!

# Subqueries in WHERE/HAVING

- 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);  $\exists e. v > e$
- SELECT ..... WHERE value > ALL (sub);  $\forall e. v > e$

# Subqueries in WHERE/HAVING

Find the name and salary of people who drive a car made before 2017

# Subqueries in WHERE/HAVING

Find the name and salary of people who drive a car made before 2017

*Person P s. t.*  
 $\exists c. year(c) < 2017$

# Subqueries in WHERE/HAVING

Find the name and salary of people who drive a car made before 2017

*Person P s. t.*  
 $\exists c. \text{year}(c) < 2017$

```
SELECT P.Name, P.Salary  
  FROM Payroll AS P  
 WHERE 2017 > ANY (SELECT R.Year  
                      FROM Regist AS R  
                      WHERE P.UserID = R.UserID)
```

True when 2017 > at  
least one of subquery

# Subqueries in WHERE/HAVING

Find the name and salary of people who **only** drive cars made before 2017

*Person P s.t.*  
 $\forall c. \text{year}(c) < 2017$

```
SELECT P.Name, P.Salary
FROM Payroll AS P
WHERE 2017 > ALL (SELECT R.Year
                     FROM Regist AS R
                     WHERE P.UserID = R.UserID)
```

True when 2017 > every  
single result of subquery

# Subquery Takeaways

- Lots of different ways to get the same answer
  - You have an arsenal of tools now
- Side note: Depending on the DBMS/optimizer subqueries could be slow
  - Use them when needed, or when subquery is the more clear solution (like argmax/argmin)

# Encoding Universal Quantifiers

- SQL (and all relational query languages) naturally support the existential quantifier  $\exists$  with joins.
- Could we ever encode a universal quantifier with a SELECT-FROM-WHERE query with no subqueries or aggregates?
- “Do I need to do something complex?”



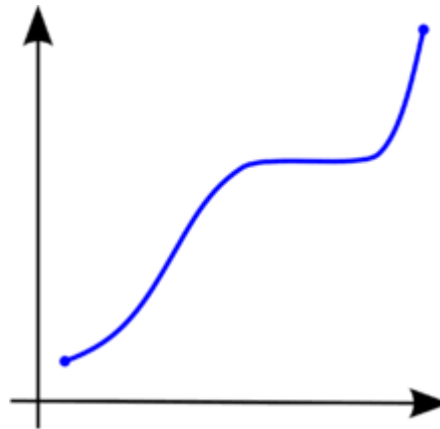
# Monotonicity

## Monotone

A **Monotonic** query is one that obeys the following rule where  $I$  and  $J$  are data instances and  $q()$  are the results of running the query  $q$  on those instances:

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

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



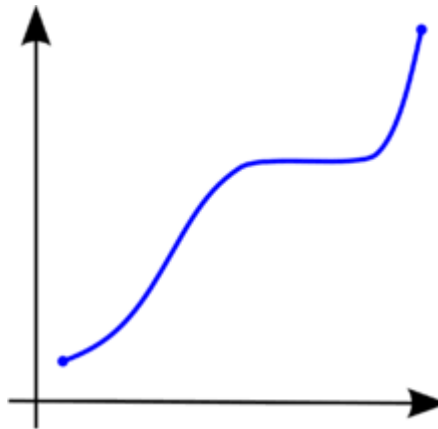
# Monotonicity

## Monotone

A **Monotonic** query is one that obeys the following rule where  $I$  and  $J$  are data instances and  $q()$  are the results of running the query  $q$  on those instances:

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

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



# Monotonicity

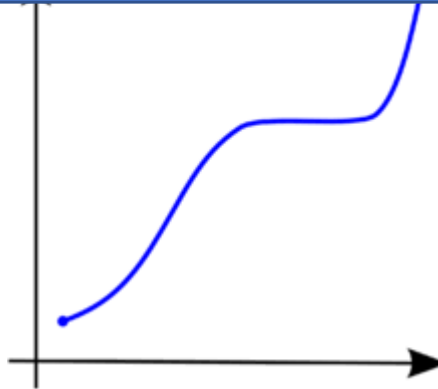
## Monotone

A **Monotonic** query is one that obeys the following rule where  $I$  and  $J$  are data instances and  $q()$  are the results of running the query  $q$  on those instances:

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

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.



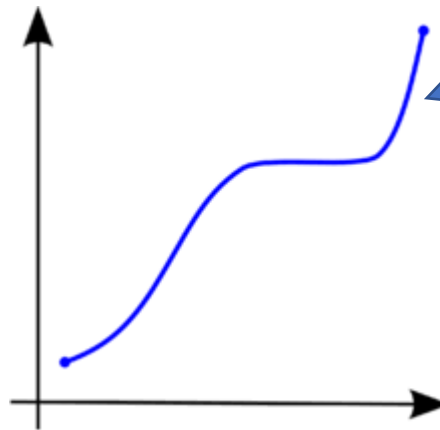
# Monotonicity

## Monotone

A **Monotonic** query is one that obeys the following rule where  $I$  and  $J$  are data instances and  $q()$  are the results of running the query  $q$  on those instances:

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

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



Monotone queries can be similar to monotonically increasing functions when considering cardinalities of results

# Monotonicity

## Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q() are the results of running the query q on those instances:

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

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?

# Monotonicity

## Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q() are the results of running the query q on those instances:

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

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!**

# Monotonicity

## Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q() are the results of running the query q on those instances:

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

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
```

I can't add tuples to Payroll or Regist that would "remove" a previous result

Is this query monotone? **Yes!**

# Monotonicity

## Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q() are the results of running the query q on those instances:

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

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?



# Monotonicity

## Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q() are the results of running the query q on those instances:

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

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!**

# Monotonicity

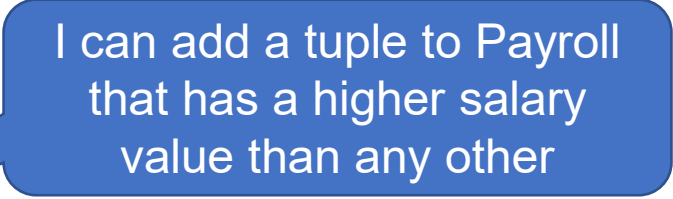
## Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q() are the results of running the query q on those instances:

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

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!**

# Monotonicity

## Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q() are the results of running the query q on those instances:

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

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

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

Is this query monotone?

# Monotonicity

## Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q() are the results of running the query q on those instances:

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

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

```
SELECT P.Job, 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() are the results of running the query q on those instances:

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

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

The value of this aggregate changing is like removing the old tuple

```
SELECT P.Job, 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() are the results of running the query q on those instances:

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

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

Don't need the SQL to tell whether a query is monotone,

e.g.

Return all people who don't own a car

# Monotonicity

## Monotone

A **Monotonic** query is one that obeys the following rule where I and J are data instances and q() are the results of running the query q on those instances:

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

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

Don't need the SQL to tell whether a query is monotone,

e.g.

Return all people who don't own a car

If we add a new car for someone, removed from output

# 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 a1, a2, ..., ak  
FROM   R1 AS x1, R2 AS x2, ..., Rn AS xn  
WHERE  Conditions
```

```
for x1 in R1 do  
  for x2 in R2 do  
    ...  
    for xn in Rn do  
      if Conditions  
        output (a1, ..., ak)
```

# 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

# Takeaways

- SQL is able to mirror logic over sets more or less directly
- The internal interpretation of nested queries can be quite involved
  - But often the DBMS will derive such interpretations automatically and pick the fastest method

# Next Unit

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