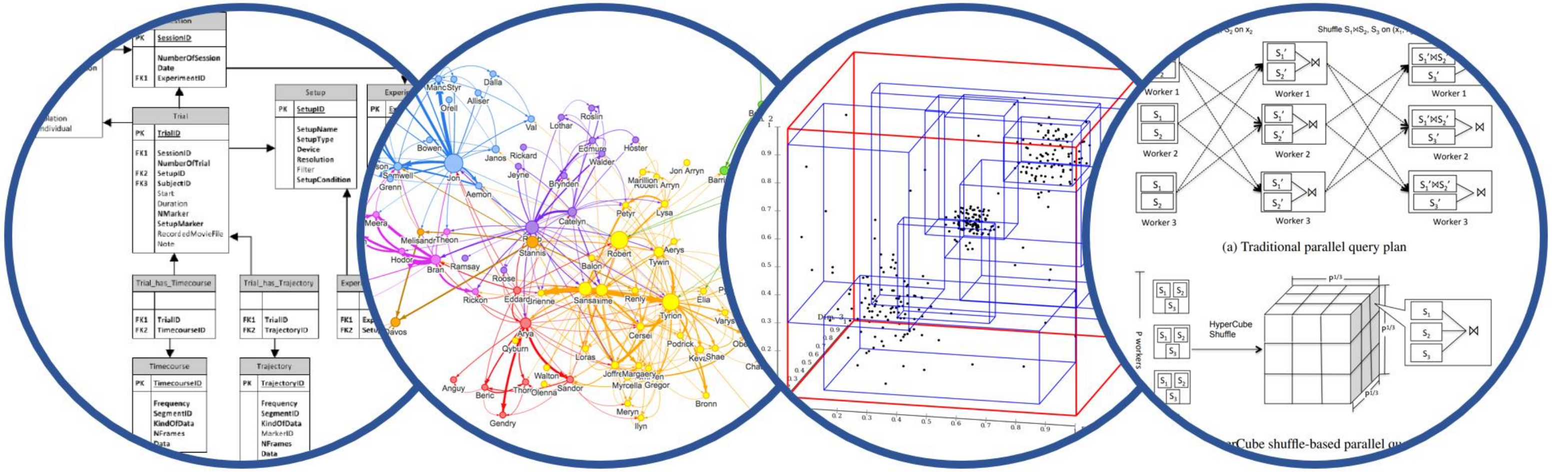


Introduction to Data Management

SQL Subqueries

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

Announcements

- Homework 3 is out
  - Important to get the setup working ASAP
  - Create server and log in to online query editor
  - Look for an email from Microsoft Azure:

    "Action required: Accept your Azure lab assignment"

    (could be in spam)

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

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!
Means outer table is referenced in the subquery.

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

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

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

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

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

55000

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	55000
345	Allison	TA	60000	55000
567	Magda	Prof	90000	95000
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

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
```

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

Subqueries in WHERE/HAVING

- Uses:
  - ANY → ∃
  - ALL → ∀
  - (NOT) IN → (∉) ∈
  - (NOT) EXISTS → (∅ = …) ∅ ≠ …

Subqueries in WHERE/HAVING

- Uses:
  - ANY → ∃
  - ALL → ∀
  - (NOT) IN → (∉) ∈
  - (NOT) EXISTS → (∅ = …) ∅ ≠ …

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

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

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!

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

$\forall x \in A, P(x)$ For all x in A, P(x) is true.

$\exists x \in A, P(x)$ There exists some x in A such that P(x) is true.

3. Subqueries in WHERE

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

Find all companies that make some products with 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

Slight rewording:
Return all companies such that **there exists some product** they make with price < 200

3. Subqueries in WHERE

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

Find all companies that make <u>some</u> products with price < 200

Existential quantifiers

Slight rewording:
Return all companies such that **there exists some product** they make 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 <u>some</u> products with price < 200

Existential quantifiers

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

Existential quantifiers

Using IN

Decorrelated!

```
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

Existential quantifiers

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

Existential quantifiers

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

Find all companies that make <u>some</u> products with price < 200

Existential quantifiers

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 unnest 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. <u>all</u> their products have price < 200

same as:

Find all companies that make <u>only</u> 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

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

Universal quantifiers are hard! ☹

3. Subqueries in WHERE

```
Product (pname,  price, cid)
Company (cid, cname, 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 does not exist some company product where 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)
```

3. Subqueries in WHERE

Product (<u>pname</u>, price, cid)
Company (<u>cid</u>, cname, city)

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

1. Find *the other* companies that make <u>some</u> 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. <u>all</u> 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

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)
```

3. Subqueries in WHERE

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

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

Universal quantifiers

Using ALL:

```
SELECT DISTINCT  C.cname
FROM  Company C
WHERE 200 >= ALL (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 s.t. all their products have price < 200

Universal quantifiers

Using ALL:

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

ALL not supported in sqlite

Encoding Universal Quantifiers

- Could we ever encode a universal quantifier with a SELECT-FROM-WHERE query with no subqueries or aggregates?

- To answer, need to define a property over queries

Monotonicity

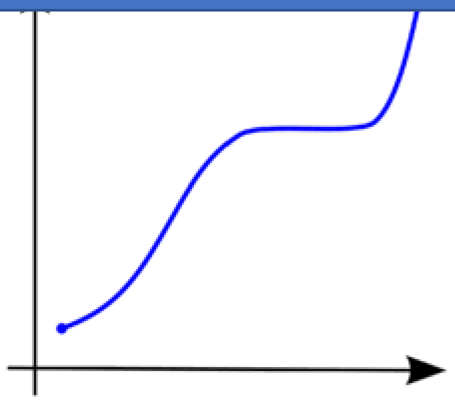
Monotone

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.

In other words, adding more tuples to the input table never removes tuples from the output on the next query.



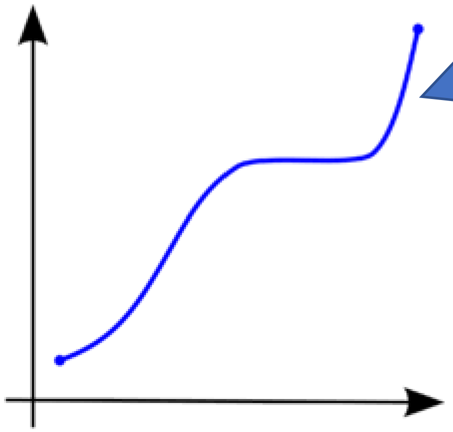
Monotonicity

Monotone

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



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 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?

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? Yes!

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.

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

```
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 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?

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
  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 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.

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

```
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 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, 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 is a query:

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

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 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.

Aggregates generally are sensitive to any new tuples

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

Is this query monotone? No!

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

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.

Next Unit

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