Advanced SQL: Subqueries and Set Operators

Abdu Alawini

University of Illinois at Urbana-Champaign

CS411: Database Systems

January 30, 2021

Learning Objectives

After this lecture, you should be able to:

- Write advanced queries with **subqueries** in the **FROM**, **WHERE** and **SELECT** clauses
- Write advanced SQL sub-queries with Boolean Operators (ANY and ALL)

Subqueries

- A parenthesized SELECT-FROM-WHERE statement (*subquery*) can be used as a value in a number of places, including FROM and WHERE clauses.

```
SELECT *
FROM (SELECT * FROM Customer WHERE name LIKE 'A%') as temp
WHERE temp.phone LIKE '5%';
```

Subqueries that Return Scalar

- If a subquery is guaranteed to produce one tuple with one component, then the subquery can be used as a value.
  - "Single" tuple often guaranteed by key constraint.
  - A run-time error occurs if there is no tuple or more than one tuple.

Example

From Sells(cafe, drink, price), find the café shops that serve Latte for the same price Espresso Royal charges for Cappuccino.

Two queries would surely work:

1. Find the price Espresso Royal charges for Cappuccino.
2. Find the café shops that serve Latte at that price.

Query + Subquery Solution

Find the café shops that serve Latte at that price

```
SELECT cafe
FROM Sells
WHERE drink = 'Latte' AND
  price = (SELECT price
           FROM Sells
           WHERE cafe = 'Espresso Royal'
           AND drink = 'Cappuccino');
```

Find the price Espresso Royal charges for Cappuccino.

Outline

- Advanced SQL
  - ✓ Subqueries
  - Boolean Operators (IN, ANY, EXISTS, ALL)
  - Set operations

Boolean Operators: The IN Operator

- <tuple> IN <relation> is true if and only if the tuple is a member of the relation.
  - <tuple> NOT IN <relation> means the opposite.
- IN-expressions can appear in WHERE clauses.
- The <relation> is often a subquery.

Example

- From Drinks(name, manf) and Likes(customer, drink), find the name and manufacturer of each drink that Fred likes.

```
SELECT *
FROM Drinks
WHERE name IN (SELECT drink
               FROM Likes
               WHERE customer = 'Fred');
```

The set of drinks Fred likes

Boolean Operators: The Exists Operator

- EXISTS(<relation>) is true if and only if the <relation> is not empty.
- Being a boolean-valued operator, EXISTS can appear in WHERE clauses.

Example Query with EXISTS

Example: From Drinks(name, manf), and Sells (café, drink, price) find the name and manufacturer of drinks with price >= $4.99

Set of drinks (and their sales information) with the same name as b1 and price >= 4.99

```
SELECT *
FROM Drinks b1
WHERE EXISTS (SELECT *
              FROM Sells
              WHERE drink = b1.name AND price >= 4.99);
```

Read "Correlated Subqueries" Section 6.3.4.

Boolean Operators: The Operator ANY

- *x* = ANY(<relation>) is a Boolean condition meaning that *x* equals at least one tuple in the relation.
- Similarly, = can be replaced by any of the comparison operators.
- Example: *x* >= ANY(<relation>) means *x* is greater than or equal to at least one tuple in the relation.
  - Note tuples must have one component only.

Boolean Operators: The Operator ALL

- Similarly, *x* <> ALL(<relation>) is true if and only if for every tuple *t* in the relation, *x* is not equal to *t*.
  - That is, *x* is not a member of the relation.

- The <> can be replaced by any comparison operator.
- Example: *x* >= ALL(<relation>) means there is *x* is larger than every tuple in the relation.

Example

- From Sells(cafe, drink, price), find the drink(s) sold for the highest price.

```
SELECT drink
FROM Sells
WHERE price >= ALL(
    SELECT price
    FROM Sells);
```

price from the outer Sells must not be less than any price.

Outline

- Advanced SQL
  - ✓ Subqueries
  - ✓ Boolean Operators (IN, ANY, EXISTS, ALL)
  - Set operations

Union, Intersection, and Difference

- Union, intersection, and difference of relations are expressed by the following forms, each involving subqueries:
  - (subquery) UNION (subquery)
  - (subquery) INTERSECT (subquery)
  - (subquery) EXCEPT (subquery)

Example

From relations Likes(customer, drink), Sells(cafe, drink, price) and Frequents(customer, cafe), find the customers and drinks such that:

1. The customer likes the drink, and
2. The customer frequents at least one café shop that sells the drink.

How would we do this?

Solution

The customer frequents a café shop that sells the drink.

```
(SELECT * FROM Likes)
  INTERSECT
(SELECT customer, drink
 FROM Sells, Frequents
 WHERE Frequents.cafe = Sells.cafe
);
```

Bag Semantics for Set Operations in SQL

Although the SELECT-FROM-WHERE statement uses bag semantics, the default for union, intersection, and difference is set semantics.

- That is, duplicates are eliminated as the operation is applied.

Controlling Duplicate Elimination

- Force the result to be a set by SELECT DISTINCT . . .
- Force the result to be a bag (i.e., don't eliminate duplicates) by ALL, as in
  - . . . UNION ALL . . .