

# Tools & Models for Data Science

## SQL Set Operations and Subqueries

Risa Myers & Chris Jermaine

Rice University



# Set Operations in SQL

- Results are unordered multisets/bag
- It could be useful to perform operations on these
  - Union
  - Intersection
  - Difference
- Different RDBMs provide different levels of support

- UNION- eliminates duplicates
- UNION ALL- does NOT eliminate duplicates
- Uses the column names from the first result set
- Data types must match
- Number of attributes must match

# UNION and UNION ALL Example

STUDENT(NETID, FIRSTNAME, LASTNAME)

FACULTY(NETID, FIRSTNAME, LASTNAME)

```
SELECT lastName, firstName, 'student'
```

```
FROM Student
```

```
UNION
```

```
SELECT lastName, firstName, 'faculty'
```

```
FROM Faculty;
```

# Intersection and Difference

- Intersection - Implemented via `INNER JOIN`
- Difference - Implemented via `EXCEPT`

# SELECT-FROM-WHERE

```
SELECT <attribute list>  
FROM <tables>  
WHERE <conditions>
```

```
SELECT *  
FROM FREQUENTS f  
WHERE f.drinker = 'Risa'
```

DRINKER	CAFE
Risa	Double Trouble
Risa	Java Lava

- 1 <attribute> = <value>
- 2 <attribute> BETWEEN [value1] AND [value2]
- 3 <attribute> IN ([value1], [value2], ...)
- 4 <attribute> LIKE 'SST%'
- 5 <attribute> LIKE 'SST\_'
- 6 <attribute> IS NULL and [attribute] IS NOT NULL
- 7 Logical combinations with AND and OR
- 8 Mathematical functions <>, !=, >, <, ...
- 9 Subqueries ...

- We can have a subquery in the `WHERE` clause
- It's linked with keywords
  - `EXISTS / NOT EXISTS`
    - If the subquery returns at least one tuple, the `EXISTS` clause evaluates to `TRUE`



- We can have a subquery in the `WHERE` clause
- It's linked with keywords
  - `<operand> IN / <operand> NOT IN`
- ? How does `IN` work? (How else could the expression be written?)

- We can have a subquery in the `WHERE` clause
- It's linked with keywords
  - `<operand> IN / <operand> NOT IN`
- How does `IN` work? (How else could the expression be written?)
- Logical OR of the operand and each value returned by the subquery

- We can have a subquery in the `WHERE` clause
- It's linked with keywords
  - `<operand> <comparison operator> ALL`
  - `<operand> <comparison operator> SOME/ANY`
- ? What is meant by an operand, in this context?

- We can have a subquery in the `WHERE` clause
- It's linked with keywords
  - `<operand> <comparison operator> ALL`
  - `<operand> <comparison operator> SOME/ANY`
- What is meant by an operand, in this context?
- An operand could be an attribute, a function or even a constant

# Subqueries - How do They Work?

- Basically, we iterate over the tuples in the outer query and evaluate the inner query for each outer tuple
- Some can be evaluated once and the result is used in the outer query
  - Ex: a subquery that returns the number of CAFES that are frequented
- Some require the subquery to be evaluated for every value assignment in the outer query (correlated subquery)
  - Ex: a subquery that returns the number of CAFES that each DRINKER frequents

# Subquery Example 1 `IN`

LIKES (DRINKER, COFFEE)

- Who likes 'Cold Brew' and 'Espresso'?
- Both subqueries return the same result
- Many (all?) subqueries can be written as JOINS, people tend to find it easier to reason about one way or the other

```
SELECT DISTINCT 1.DRINKER
FROM LIKES 1
WHERE 1.COFFEE = 'Cold_Brew'
      AND 1.DRINKER IN (
          SELECT 12.DRINKER
          FROM LIKES 12
          WHERE 12.COFFEE = 'Espresso')
```

```
SELECT DISTINCT 11.DRINKER
FROM LIKES 11, LIKES 12
WHERE 11.DRINKER = 12.DRINKER
      AND 11.COFFEE = 'Cold_Brew'
      AND 12.COFFEE = 'Espresso'
```

## Subquery Example 2 EXISTS

LIKES (DRINKER, COFFEE)

? Who goes to a cafe that serves 'Cold Brew'?

```
SELECT DISTINCT f.DRINKER
FROM FREQUENTS f, SERVES s
WHERE f.CAFE = s.CAFE
        AND s.COFFEE = 'Cold_Brew'
```

```
SELECT DISTINCT f.DRINKER
FROM FREQUENTS f
WHERE EXISTS (
        -- Your code here
    )
```

# Subquery Example 2 EXISTS

LIKES (DRINKER, COFFEE)

- Who goes to a cafe that serves 'Cold Brew'?

```
SELECT DISTINCT f.DRINKER
FROM FREQUENTS f, SERVES s
WHERE f.CAFE = s.CAFE
      AND s.COFFEE = 'Cold_Brew'
```

```
SELECT DISTINCT f.DRINKER
FROM FREQUENTS f
WHERE EXISTS (
    SELECT s.CAFE
    FROM SERVES s
    WHERE s.COFFEE = 'Cold_Brew'
      AND f.CAFE = s.CAFE)
```



## Subquery Example 3

LIKES (DRINKER, COFFEE)

FREQUENTS (DRINKER, CAFE)

SERVES (CAFE, COFFEE)

? Who likes all of the coffees that Risa likes?

## Subquery Example 3: Relational Calculus

LIKES (DRINKER, COFFEE)  
FREQUENTS (DRINKER, CAFE)  
SERVES (CAFE, COFFEE)

? Who likes all of the coffees that Risa likes?

- There doesn't exist a coffee Risa likes that is not also liked by these drinkers
- Every coffee Risa likes is liked by these drinkers BUT they might like other coffees as well
- Same as:

$$\{l.DRINKER | \text{LIKES}(l) \wedge \neg \exists(l_2)(\text{LIKES}(l_2) \wedge l_2.DRINKER = \text{'Risa'} \\ \wedge \neg \exists(l_3)(\text{LIKES}(l_3) \wedge l_3.DRINKER = l.DRINKER \\ \wedge l_3.COFFEE = l_2.COFFEE))\}$$

# Subquery Example 3 Component

LIKES (DRINKER, COFFEE)  
FREQUENTS (DRINKER, CAFE)  
SERVES (CAFE, COFFEE)

? Coffees that Risa likes

# Subquery Example 3 Component

LIKES (DRINKER, COFFEE)  
FREQUENTS (DRINKER, CAFE)  
SERVES (CAFE, COFFEE)

## ■ Coffees that Risa likes

```
SELECT 12.COFFEE  
FROM LIKES 12  
WHERE 12.DRINKER = 'Risa'
```

## Subquery Example 3

LIKES (DRINKER, COFFEE)  
FREQUENTS (DRINKER, CAFE)  
SERVES (CAFE, COFFEE)

? Who likes all of the coffees that Risa likes?

```
SELECT DISTINCT l.DRINKER  
FROM LIKES l  
WHERE NOT EXISTS (a coffee Risa likes that is not also liked by l.DRINKER)
```

$$\{l.DRINKER \mid \text{LIKES}(l) \wedge \neg \exists (l_2)(\text{LIKES}(l_2) \wedge l_2.DRINKER = \text{'Risa'} \\ \wedge \neg \exists (l_3)(\text{LIKES}(l_3) \wedge l_3.DRINKER = l.DRINKER \\ \wedge l_3.COFFEE = l_2.COFFEE))\}$$

## Subquery Example 3

LIKES (DRINKER, COFFEE)  
FREQUENTS (DRINKER, CAFE)  
SERVES (CAFE, COFFEE)

? Who likes all of the coffees that Risa likes?

```
SELECT DISTINCT l.DRINKER
FROM LIKES l
WHERE NOT EXISTS (
  SELECT l2.COFFEE
  FROM LIKES l2
  WHERE l2.DRINKER = 'Risa' AND l2.COFFEE NOT IN (
    the set of coffees liked by l.DRINKER ) )
```

■ There doesn't exist a coffee that Risa likes where that coffee is not ...

$$\{l.DRINKER | LIKES(l) \wedge \neg \exists (l_2)(LIKES(l_2) \wedge l_2.DRINKER = 'Risa' \\ \wedge \neg \exists (l_3)(LIKES(l_3) \wedge l_3.DRINKER = l.DRINKER \\ \wedge l_3.COFFEE = l_2.COFFEE))\}$$

## Subquery Example 3 Final

LIKES (DRINKER, COFFEE)  
FREQUENTS (DRINKER, CAFE)  
SERVES (CAFE, COFFEE)

? Who likes all of the coffees that Risa likes?

■ Still need: A coffee Risa likes that is not also liked by I.DRINKER

```
SELECT DISTINCT 1.DRINKER
FROM LIKES 1
WHERE NOT EXISTS (
    SELECT 12.COFFEE
    FROM LIKES 12
    WHERE 12.DRINKER = 'Risa'
    AND that coffee is also not liked by 1.DRINKER)
```

## Subquery Example 3 Final

LIKES (DRINKER, COFFEE)  
FREQUENTS (DRINKER, CAFE)  
SERVES (CAFE, COFFEE)

? Who likes all of the coffees that Risa likes?

■ Still need: A coffee Risa likes that is not also liked by I.DRINKER

```
SELECT DISTINCT 1.DRINKER
FROM LIKES 1
WHERE NOT EXISTS (
    SELECT 12.COFFEE
    FROM LIKES 12
    WHERE 12.DRINKER = 'Risa'
    AND 12.COFFEE NOT IN (coffees liked by 1.DRINKER)
```



## Subquery Example 3 Final

LIKES (DRINKER, COFFEE)  
FREQUENTS (DRINKER, CAFE)  
SERVES (CAFE, COFFEE)

? Who likes all of the coffees that Risa likes?

■ Still need: A coffee Risa likes that is not also liked by I.DRINKER

```
SELECT DISTINCT 1.DRINKER
FROM LIKES 1
WHERE NOT EXISTS (
  SELECT 12.COFFEE
  FROM LIKES 12
  WHERE 12.DRINKER = 'Risa'
  AND 12.COFFEE NOT IN (
    SELECT 13.COFFEE
    FROM LIKES 13
    WHERE 13.DRINKER = 1.DRINKER))
```

# Subquery Example 3 vs. Relational Calculus

```
SELECT DISTINCT l.DRINKER
FROM LIKES l
WHERE NOT EXISTS (
  SELECT 12.COFFEE
  FROM LIKES 12
  WHERE 12.DRINKER = 'Risa' AND 12.COFFEE NOT IN (
    SELECT 13.COFFEE
    FROM LIKES 13
    WHERE 13.DRINKER = l.DRINKER))
```

■  $l.DRINKER = 13.DRINKER$

■  $12.DRINKER = \text{'Risa'}$

■ Same as:

$$\{l.DRINKER | \text{LIKES}(l) \wedge \neg \exists (l_2)(\text{LIKES}(l_2) \wedge l_2.DRINKER = \text{'Risa'})} \\ \wedge \neg \exists (l_3)(\text{LIKES}(l_3) \wedge l_3.DRINKER = l.DRINKER \\ \wedge l_3.COFFEE = l_2.COFFEE))\}$$

- SOME/ANY is used like “`expression boolOp {SOME, ANY }(subquery)`”
- SOME/ANY returns TRUE if there is at least 1 item in the subquery can make the boolOp evaluate to true

Given the relation:

RATES (DRINKER, COFFEE, SCORE)

- Ratings go from low to high, with increasing values indicating higher levels of liking the coffee.
- ? Of the coffees Risa has rated, list the coffees that are not Risa's favorite.
- ? What does it mean, in terms of RATES, when we say favorite?

Given the relation:

RATES (DRINKER, COFFEE, SCORE)

- Of the coffees Risa has rated, list the coffees that are not Risa's favorite.

```
SELECT r.COFFEE
FROM RATES r
WHERE r.DRINKER = 'Risa' AND r.SCORE < SOME (
    SELECT r2.SCORE
    FROM RATES r2
    WHERE r2.DRINKER = 'Risa' )
```

## SOME/ANY Example Unpacked

Given the relation:

RATES (DRINKER, COFFEE, SCORE)

- Of the coffees Risa has rated, list the coffees that are not Risa's favorite.

```
SELECT r.COFFEE
FROM RATES r
WHERE r.DRINKER = 'Risa' AND r.SCORE < SOME (
    SELECT r2.SCORE
    FROM RATES r2
    WHERE r2.DRINKER = 'Risa' )
```

- The subquery returns the multiset of all the scores that Risa has given to coffees
- The  $r.SCORE < SOME$  clause evaluates to TRUE if the multiset is not empty

# ALL predicate

- ALL is used like “expression boolOp ALL (subquery)”
- Similar to SOME
- BoolOp must evaluate to true for **everything** in the subquery

RATES (DRINKER, COFFEE, SCORE)

```
SELECT DISTINCT r.DRINKER
FROM RATES r
WHERE r.SCORE < ALL (
  SELECT r2.SCORE
  FROM RATES r2
  WHERE r2.DRINKER = 'Risa')
```

? What does this query return?



RATES (DRINKER, COFFEE, SCORE)

```
SELECT DISTINCT r.DRINKER
FROM RATES r
WHERE r.SCORE < ALL (
  SELECT r2.SCORE
  FROM RATES r2
  WHERE r2.DRINKER = 'Risa')
```

- What does this query return?
- Drinkers who rated a coffee lower than all of Risa's scores

# Subqueries in FROM Clause

## FREQUENTS (DRINKER, CAFE)

- Can have a subquery in FROM clause
- Treated as a temporary table
- MUST be assigned an alias
- ? Who goes to a cafe that serves 'Cold Brew'?

### Old way

```
SELECT DISTINCT f.DRINKER  
FROM FREQUENTS f, SERVES s  
WHERE f.CAFE = s.CAFE  
        AND s.COFFEE = 'Cold_Brew'
```

### New way

```
SELECT DISTINCT f.DRINKER  
FROM FREQUENTS f,  
        (SELECT s.CAFE FROM SERVES s  
         WHERE s.COFFEE = 'Cold_Brew') s2  
WHERE f.CAFE = s2.CAFE
```

## FREQUENTS (DRINKER, CAFE)

- Note: The code is a lot cleaner with a view!

```
CREATE VIEW CB_COFFEE AS  
SELECT s.CAFE FROM SERVES s  
      WHERE s.COFFEE = 'Cold_Brew'
```

```
SELECT DISTINCT f.DRINKER  
FROM FREQUENTS f, CB_COFFEE c  
WHERE f.CAFE = c.CAFE
```

- “Common” (non-materialized) views are just macros
- Query definition
- Unexecuted query
- Virtual table
- Can be used in place of a table
- Convenient way to simplify a query
- Is executed when called (results are NOT stored)
- ? List the coffees that are not Risa's favorite.

- “Common” (non-materialized) views are just macros
- List the coffees that are not Risa's favorite.

```
CREATE VIEW RISA_COFFEES AS  
SELECT *  
FROM RATES r  
WHERE r.DRINKER = 'Risa'
```

```
SELECT r.COFFEE  
FROM RISA_COFFEES r  
WHERE r.SCORE < SOME (  
    SELECT r2.SCORE  
    FROM RISA_COFFEES r2)
```

# Materialized Views

? What's different?

- What's different?
  - Query is run when the view is created
  - Results are stored until the view is `REFRESHED`

- Declarative SQL code tends to be very short
- Good: because effort & bugs  $\propto$  code length
- Bad: because it can be difficult to understand!



- Hence, style is important. Some suggestions
  - Always alias tuple variables and relations
  - Always indent carefully
  - Only one major keyword per line (`SELECT`, `FROM`, etc.)
  - Pick a capitalization scheme and religiously stick to it
  - Make frequent use of views...

- Explicitly list all attributes
- Specify the relation attribute for each attribute
- Left outer join / Right outer join - CHOOSE 1

# True/False Questions

- 1 Every SQL query must contain a WHERE clause
- 2 By default, a VIEW stores the data retrieved from the query
- 3 Subqueries isolate parts of queries
- 4 Using VIEWS makes it harder to figure out what's going on in a query
- 5 Subqueries can only return a single value
- 6 Subqueries must reference an attribute from the outer query
- 7 Subqueries can only appear in the WHERE clause

- ? How can we use what we learned today?
- ? What do we know now that we didn't know before?