COMP 543: Tools & Models for Data Science

SQL 1

Chris Jermaine & Risa Myers

Rice University

SQL

- De-facto standard DB programming language
  - First proposed by IBM researchers in 1970's
  - Oracle first to offer commercial version in 1979
  - IBM soon after
- SQL is a H U G E language!!
  - Current standard runs to 100s of pages
  - Consists of a declarative DML
  - And an imperative DML
  - And a DDL
- We begin with the heart and soul of SQL: the declarative DML

Our First Query

- Given the following relations:

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

? Who goes to a cafe serving Cold Brew?

Our First Query

- Given the following relations:

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

- Who goes to a cafe serving Cold Brew?

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

? What happens without `DISTINCT`?

Our First Query

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

- Who goes to a cafe serving Cold Brew?

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

- What happens without `DISTINCT`?
- Closely related to RC! Same as:

$$\{f.\text{DRINKER} | \text{FREQUENTS}(f) \wedge \text{SERVES}(s) \\ \wedge f.\text{CAFE} = s.\text{CAFE} \wedge s.\text{COFFEE} = \text{'Cold Brew'}\}$$

Subqueries

- We can have a subquery in the `WHERE` clause
- It's linked with keywords
  - `EXISTS` / `NOT EXISTS`
  - <attribute> `IN` / <attribute> `NOT IN`
  - <attribute> <comparison operator> `ALL`
  - <attribute> <comparison operator> `SOME/ANY`

Subquery Example

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

? Who likes all of the coffees that Risa likes?

Subquery Example

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)
```

Subquery Example

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))
```

Subquery Example

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 (
    SELECT l3.COFFEE
    FROM LIKES l3
    WHERE l3.DRINKER = l.DRINKER))
```

Subquery Example

```
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 (
    SELECT l3.COFFEE
    FROM LIKES l3
    WHERE l3.DRINKER = l.DRINKER))
```

- Same as:

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

SOME predicate

- SOME is used like "`expression boolOp SOME (subquery)`"
- SOME returns TRUE if some item in the subquery can make the boolOp evaluate to true

SOME Example

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.

SOME Example

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.

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

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 r.DRINKER = 'Risa')
```

? What does this query return?

Some Closing Notes

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

Some Closing Notes on Style

- 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...

Views

- “Common” (non-materialized) views are just macros

? List the coffees that are not Risa’s favorite.

Views

- “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?

Materialized Views

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

Questions?