L10: Subqueries

Queries Within Queries

Dr Kieran T. Herley

Semester One, 2023-24

School of Computer Science & Information Technology
University College Cork

Summary

SQL's set operators. Subqueries.

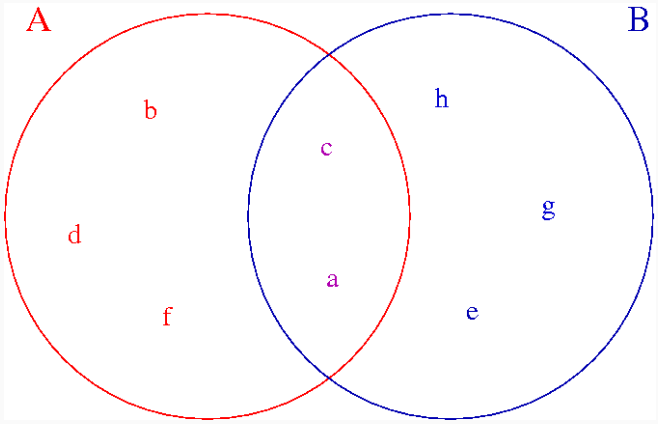
Subqueries

- Subqueries:
  - queries built out of simpler queries
  - often an alternative to join-based queries
- Today's DB:

```
movies(id, title, yr, score, votes, director)
actors(id, name)
castings(movieid, actorid)
```

Set Operations

Set Operators



$$A \cup B = \{a, b, c, d, e, f, g, h\}$$
$$A \cap B = \{a, c\}$$
$$A - B = \{b, d, f\}$$

- Recall:
  $A \cup B$ **(union)** contains all elements that belong either to set A or to set B (or both)
  $A \cap B$ **(intersection)** contains all elements that belong both to set A and to set B
  $A - B$ **(difference)** contains all elements that belong both to set A but not to set B
- Since relational theory based on set theoretic notion like sets, relations etc., it seems natural that SQL should support set operators . . .

SQLite set operators

- SQLite supports set operators to combine results of queries

  Q1 UNION Q2 All distinct rows from Q1 or Q2

  Q1 INTERSECT Q2 All rows common to Q1 and Q2

  Q1 EXCEPT Q2 All rows from Q1 not appearing in Q2

- Queries Q1 and Q2 need to have compatible column structure: same number, names and types

Subquery Intersection

- List films ids with both Humphrey Bogart and Katharine Hepburn
- Cunning Plan:
  - Separately find HB's films and those of KH
  - Find those in common (intersect)

Subquery Intersection

- List films ids with both Humphrey Bogart and Katharine Hepburn
- Cunning Plan:
  - Separately find HB's films and those of KH
  - Find those in common (intersect)
-

```
    SELECT movieid
    FROM actors JOIN castings
    ON actor.id = castings.actorid
    WHERE actor.name = 'Humphrey Bogart'
INTERSECT
    /* ids of films with KH */
```

- Other DB systems permit () around subqueries but not SQLite

Subquery Unions

- List ids of all actors whose name is 'Jack' or who have at least ten films to their credit
- UNION **is** supported by MySQL

Subquery Unions

- List ids of all actors whose name is 'Jack' or who have at least ten films to their credit
- UNION **is** supported by MySQL
-

```
  /* ids of actors named Jack */
UNION
  /* ids of actors with at least ten films */
```

- Two subqueries involve tables `actors` and `castings` respectively
- For UNION, subquery results must be "compatible" i.e. have same number of columns and types

Unions cont'd

- ```
  SELECT id
  FROM actors
  WHERE name LIKE 'Jack%'
UNION
  SELECT actorid AS id
  FROM castings
  GROUP BY actorid
  HAVING COUNT(*) >= 10
  ```

- SQL normally allows duplicates, but for `UNION` duplicates are suppressed by default; use `UNION ALL` if you really want them
  ```

Cautionary Example

List films that either released during the 1960s or have a score of at least 8.0

```
   SELECT title
   FROM movies
   WHERE yr BETWEEN 1960 AND 1969
UNION
   SELECT title
   FROM movies
   WHERE score >= 8.0
```

OK but . . .

Cautionary Example

List films that either released during the 1960s or have a score of at least 8.0

```
   SELECT title
   FROM movies
   WHERE yr BETWEEN 1960 AND 1969
UNION
   SELECT title
   FROM movies
   WHERE score >= 8.0
```

OK but . . .

```
SELECT title
FROM movies
WHERE
   yr BETWEEN 1960 AND 1969
   OR score >= 8.0;
```

Could re-express more easily as shown with no need for subqueries

Subqueries That Return Single Values

List The Film(s) With The Greatest Score

- If only we knew what the top score was . . .

List The Film(s) With The Greatest Score

- If only we knew what the top score was . . .
-

```
SELECT title, score
FROM movies
WHERE score =
(   SELECT MAX(score)
        FROM movies
);
```

- Inner subquery `(SELECT MAX(score) . . .)` returns the maximum score:

$$\frac{\text{MAX(score)}}{9.0}$$

  a single value (albeit "wrapped" in a 1×1 table)
- Outer (containing) query uses this *value* in its WHERE clause

But what about . . .

- Simpler query mysteriously produces right (?) answer

```
SELECT title, MAX(score)
FROM movies;
```

- Feature non-standard, not well documented and dangerous.
  Do not rely on this!

```
SELECT title, MAX(score), MIN(score)
FROM movies;
```

List The Ids Of All Actors Appearing In “The Godfather”

- Inner subquery extracts film’s id from `movies` table; outer query extracts associated actors from `castings`
-

```
SELECT actorid
FROM castings
WHERE movieid =
(   SELECT id
    FROM movies
    WHERE title = 'Godfather, The'
);
```

What Does This Do?

```
SELECT title, score
FROM movies
WHERE score >
(   SELECT score
    FROM movies
    WHERE title = 'Sound of Music, The'
);
```

Subqueries Returning One-Column Tables

Conditions Involving Relations

- Tables containing a single column are known as *unary relations*; such relations are effectively lists
- SQL provides some Boolean functions that operate on a unary relation (R):
  - `EXISTS R`: True if R is not empty
  - `s IN R`: True if s is one of the values in R (also `s NOT IN R`)

List The Names Of All Actors Appearing In "The Godfather"

- Idea:
  - Inner subquery extracts film's id from `movies` table
  - Outer containing query extracts associated actors from `castings`
-

```
SELECT name
FROM actors
WHERE id IN
(   /* Subquery to return unary relation of
       ids of actors in "Godfather, The" */
);
```

- Condition `id IN ( . . .)` returns True if the id value (from `actors`) is among those in relation returned by the subquery.

List The Names Of All Actors Appearing In “The Godfather”

```
/* Names of actors in "The Godfather" */
SELECT name
FROM actors
WHERE id IN
(   /* ids of actors in "The Godfather" */
    SELECT actorid FROM castings
    WHERE movieid =
    (   /* id of "The Godfather" */
        SELECT id FROM movies
        WHERE title = 'Godfather, The'
    )
);
```

What Does This Do?

```
SELECT *
FROM movies
WHERE director =
(   SELECT director
    FROM movies
    WHERE title =
    (   SELECT MAX(title)
        FROM movies
    )
);
```

Subqueries Returning Complete Tables

Subqueries Returning Multi-Column Tables

- Can use subqueries instead of table names in FROM clauses
-

```
SELECT . . .
FROM
(   /* subquery here */
)   AS subquery_name
WHERE . . .
```

- Note must give an alias to subquery in this case, so you can refer to it in the SELECT and WHERE clauses

What Is The Greatest Number Of Films Made By Any Actor?

- Strategy:
  - Subquery to count the number films per actor using GROUP BY and COUNT on `castings` table
  - Containing query uses MAX to determine largest count

What Is The Greatest Number Of Films Made By Any Actor?

- Strategy:
  - Subquery to count the number films per actor using GROUP BY and COUNT on `castings` table
  - Containing query uses MAX to determine largest count
-

```
SELECT MAX(num_films)
FROM
(   SELECT actorid, COUNT(movieid) AS 'num_films'
    FROM castings
    GROUP BY actorid
)   AS film_counts;
```

- Note subquery returns two-column, multi-row table (actors and the number of his/her films)

Find The Name of the Most Prolific Actor

```
SELECT name, COUNT(*)
FROM actors AS a
    JOIN castings AS c
        ON a.id = c.actorid
GROUP BY a.id
HAVING COUNT(*) =
(
    SELECT MAX(num_films)
    FROM
    (   SELECT actorid, COUNT(movieid) AS 'num_films'
        FROM castings
        GROUP BY actorid
    )   AS film_counts
);
```

A Complex Example

Find all films in which both Meryl Streep and Clint Eastwood both appeared

```
/* Name of film(s) with both Clint Eastwood and Meryl Streep */
SELECT title, yr FROM movies
WHERE id IN
(  /* ids of movies with Meryl Streep and Clint Eastwood */
   SELECT movieid FROM castings
   WHERE
      actorid =
      (  /* Meryl Streep's id */
         SELECT id FROM actors
         WHERE name = 'Meryl Streep'
      )
      AND
      movieid IN
      (  /* ids of movies with Clint Eastwood */
         SELECT movieid FROM castings
         WHERE actorid =
         (  /* Clint Eastwood's id */
            SELECT id FROM actors
            WHERE name = 'Clint Eastwood'
         )
      )
);
```

Complex Example Dissected

- Subquery to list all Clint Eastwood's films

```
(   /* ids of movies with Clint Eastwood */
    SELECT movieid FROM castings
    WHERE actorid =
    (   /* Clint Eastwood's id */
        SELECT id FROM actors
        WHERE name = 'Clint Eastwood'
    )
)
```

Complex Example Dissected cont'd

- Subquery to list all films with Meryl Streep and Clint Eastwood

```
(   /* ids of movies with both Meryl Streep and Clint Eastwood */
    SELECT movieid FROM castings
    WHERE
        actorid =
        (   /* Meryl Streep's id */
            SELECT id FROM actors
            WHERE name = 'Meryl Streep'
        )
        AND
        movieid IN
        (   /* Subquery that returns ids of movies with Clint Eastwood */
        )
);
```

Complex Example Dissected cont'd

- Subquery to list all films with Meryl Streep and Clint Eastwood

```
SELECT title, yr
FROM movies
WHERE id IN
(   /* Subquery that returns the ids of movies with both
       Meryl Streep and Clint Eastwood
    */
);
```

- Here each subquery is completed and the result passed to the containing query and so on; completion progresses from innermost outwards [1]

[1] Correlated subqueries (seen later) involve a more complex interplay between queries/subqueries

Notes and Acknowledgements

Some content here