Advanced SQL Querying
Advanced Database Management

Suryateja Chalapati
MS in Business Analytics and Information Systems
University of South Florida

September 9, 2020

Contents

1 Group 1 Queries **1**
1.1 Query 1 . . . 1
1.2 Query 2 . . . 1
1.3 Query 3 . . . 1
1.4 Query 4 . . . 2
1.5 Query 5 . . . 3
1.6 Query 6 . . . 3
1.7 Query 7 . . . 4
1.8 Query 8 . . . 4

2 Group 2 Queries **5**
2.1 Query 10 . . . 5
2.2 Query 11 . . . 5
2.3 Query 12 . . . 6
2.4 Query 13 . . . 7
2.5 Query 14 . . . 7
2.6 Query 15 . . . 8
2.7 Query 16 . . . 9
2.8 Query 18 . . . 9

3 Group 3 Queries **10**
3.1 Query 19 . . . 10
3.2 Query 20 . . . 10
3.3 Query 21 . . . 11

4 Interesting Queries **12**
4.1 Query 1 . . . 12
4.2 Query 2 . . . 13
4.3 Query 3 . . . 14

1 Group 1 Queries

1.1 Query 1

Display each beer's name and style name. A beer should be displayed regardless of whether a style name exists or not.

```sql
SELECT
    d.beer_name,
    t.style_name
FROM
    db2slate.beerdb_beers d
    LEFT JOIN db2slate.beerdb_styles t
        ON d.style_id = t.style_id;
```

1.2 Query 2

Display each beer's name, category name, color example, and style name, for all beers that have values for category name, color example, and style name.

```sql
SELECT
    t1.beer_id,
    t2.category_name,
    t3.examples,
    t4.style_name
FROM
    db2slate.beerdb_beers t1
    JOIN db2slate.beerdb_categories t2
        ON t1.cat_id = t2.category_id
    JOIN db2slate.beerdb_colors t3
        ON t1.srm = t3.lovibond_srm
    JOIN db2slate.beerdb_styles t4
        ON t1.style_id = t4.style_id;
```

1.3 Query 3

Display each brewer's name along with the minimum, maximum, and average alcohol by volume (ABV) of its beers. Exclude any beers with an ABV of zero. Show the brewers with the highest average ABV first.

```sql
SELECT
    br.name,
    ROUND(MIN(abv), 1) AS MIN_abv,
    ROUND(MAX(abv), 1) AS MAX_abv,
    ROUND(AVG(abv), 1) AS avg_abv
FROM
    beerdb.breweries br
    INNER JOIN beerdb.beers be
        ON br.brewery_id = be.brewery_id
WHERE
    abv > 0
GROUP BY
    br.name
ORDER BY
    AVG(abv) DESC;
```

1.4 Query 4

Find which cities would be good for hosting microbrewery tours. A city must have at least 10 breweries to be considered. Display the city's name as well as how many breweries are in the city. Show cities with the most breweries first.

```sql
SELECT
    *
FROM
    (
    SELECT
        city,
        COUNT(*) as NumofBrew
    FROM
        db2slate.beerdb_breweries
    WHERE
        city is NOT NULL
    GROUP BY
        city
    ORDER BY
        2 DESC
    ) T
WHERE
    T.numofbrew > 9;
```

1.5 Query 5

Display all beer names that (1) belong to a category with a name containing “Lager” somewhere in the name and (2) have an alcohol by volume (ABV) of eight or greater. Show the beer names in alphabetical order.

```
SELECT
    b.beer_name,
    b.ABV,
    c.category_name
FROM
    db2slate.beerdb_beers b
    JOIN db2slate.beerdb_categories c
        ON b.cat_id = c.category_id
WHERE
    category_name LIKE '%Lager%' AND ABV >= 8
ORDER BY
    b.beer_name ASC;
```

1.6 Query 6

Display the name of all movies that have an IMDB rating of at least 8.0, with more than 100,000 IMDB votes, and were released from 2007 to 2013. Show the movies with the highest IMDB ratings first.

```
SELECT
    film_title,
    imdb_rating,
    imdb_votes,
    film_year
FROM
    relmdb.movies
WHERE
```

```
    imdb_rating >= 8.0 and imdb_votes >= 100000
    AND film_year >= 2007 AND film_year <= 2013
ORDER BY
    imdb_rating DESC;
```

1.7 Query 7

Display each movie's title and total gross, where total gross is USA gross and worldwide gross combined. Exclude any movies that do not have values for either USA gross or worldwide gross. Show the highest grossing movies first.

```
SELECT
    Film_Title,
    USA_GROSS,
    WORLDWIDE_GROSS,
    (USA_GROSS + WORLDWIDE_GROSS) AS TOTAL_GROSS
FROM
    relmdb.movies
WHERE
    (USA_GROSS IS NOT NULL)
    AND (WORLDWIDE_GROSS IS NOT NULL)
ORDER BY
    TOTAL_GROSS DESC;
```

1.8 Query 8

Display the titles of any movies where Tom Hanks or Tim Allen were cast members. Each movie title should be shown only once.

```
SELECT
    DISTINCT f.film_title
FROM
    relmdb.MOVIES f
    JOIN relmdb.casts c
        ON f.film_id = c.film_id
WHERE
    cast_member IN ('Tom Hanks','Tim Allen');
```

2 Group 2 Queries

2.1 Query 10

Label the strength of a beer based on its ABV. For each beer display the beer's name, ABV, and a textual label describing the strength of the beer. The label should be "Very High" for an ABV more than 10, "High" for an ABV of 6 to 10, "Average" for an ABV of 3 to 6, and "Low" for an ABV less than 3. Show the records by beer name.

```
SELECT
    CASE
        WHEN ABV > 10 THEN 'VERY HIGH'
        WHEN ABV BETWEEN 6 AND 10 THEN 'HIGH'
        WHEN ABV BETWEEN 3 AND 6 THEN 'AVERAGE'
        ELSE 'LOW'
    END AS Strength,
    beer_name,
    ABV
FROM
    db2slate.beerdb_beers
ORDER BY
    beer_name DESC;
```

2.2 Query 11

Find all breweries that specialize in a particular beer style. A brewer is considered specialized if they produce at least 10 beers from the same style. Show the brewer's name, style name, and how many beers the brewer makes of that style. Display the records by style name first and then by breweries with the most beers within that style.

```
SELECT
    t2.name as Brewery_Name,
    t3.style_name,
    COUNT (*) Total_count
FROM
    db2slate.beerdb_beers t1
    JOIN db2slate.beerdb_breweries t2
```

```
        ON t1.brewery_id = t2.brewery_id
    JOIN db2slate.beerdb_styles t3
        ON t1.style_id = t3.style_id
GROUP BY
    t3.style_name, t2.name
HAVING
    COUNT (*) >= 10
ORDER BY
    Total_count desc;
```

2.3 Query 12

Display each brewer's name and how many beers they have associated with their brewery. Only include brewers that are located outside the United States and have more than the average number of beers from all breweries (excluding itself when calculating the average). Show the brewers with the most beers first. If there is a tie in number of beers, then sort by the brewers' names.

```
SELECT
    name,
    count(beer_name)
FROM
    db2slate.beerdb_beers be
    INNER JOIN db2slate.beerdb_breweries br
        ON br.brewery_id = be.brewery_id
WHERE
    country NOT LIKE 'United States'
GROUP BY
    name
HAVING
    COUNT(beer_name) > (
        SELECT AVG(COUNT(beer_name))
        FROM
            db2slate.beerdb_beers b2
        WHERE
            b2.brewery_id <> br.brewery_id
        GROUP BY
            b2.brewery_id
            )
ORDER BY
    COUNT(beer_name) desc;
```

2.4 Query 13

For each movie display its movie title, year, and how many cast members were a part of the movie. Exclude movies with five or fewer cast members. Display movies with the most cast members first, followed by movie year and title.

```
SELECT
    COUNT(c.film_id) cast_number,
    m.film_year,
    m.film_title
FROM
    relmdb.movies m
    INNER JOIN relmdb.casts c
    ON c.film_id = m.film_id
WHERE
    c.film_id IS NOT NULL
GROUP BY
    c.film_id, m.film_title, m.film_year
HAVING
    COUNT(c.film_id) > 5
ORDER BY
    COUNT(c.film_id) DESC, m.film_year DESC;
```

2.5 Query 14

For each genre display the total number of films, average fan rating, and average USA gross. A genre should only be shown if it has at least five films. Any film without a USA gross should be excluded. A film should be included regardless of whether any fans have rated the film. Show the results by genre. (Hint: use the TRIM function to only show a single record from the same genre.)

```
SELECT
    G.GENRE_NAME,
    COUNT(FILM_ID) FILM_COUNT,
    ROUND(AVG(F.IMDB_RATING),1) AVG_FAN_RATING,
    ROUND(AVG(M.USA_GROSS),1) AVG_USA_GROSS
FROM
    RELMDB.MOVIES M
    JOIN RELMDB.GENRES G
        USING (FILM_ID)
    JOIN RELMDB.FAN_RATINGS F
        USING (FILM_ID)
WHERE
    M.USA_GROSS IS NOT NULL
GROUP BY
    G.GENRE_NAME
HAVING
    COUNT(G.GENRE_NAME) > 5
ORDER BY
    G.GENRE_NAME ASC;
```

2.6 Query 15

Find the average budget for all films from a director with at least one movie in the top 25 IMDB ranked films. Show the director with the highest average budget first.

```
SELECT
    D.DIRECTOR,
    ROUND(AVG(BUDGET),1) AVG_BUDGET
FROM
    RELMDB.DIRECTORS D
    JOIN RELMDB.MOVIES M
        ON D.FILM_ID = M.FILM_ID
WHERE
    IMDB_RANK <= 25 AND BUDGET IS NOT NULL
GROUP BY
    D.DIRECTOR
ORDER BY
    AVG(BUDGET) DESC;
```

2.7 Query 16

Find all duplicate fans. A fan is considered duplicate if they have the same first name, last name, city, state, zip, and birth date.

```
SELECT
    F.FNAME,
    F.LNAME,
    F.CITY,
    F.STATE,
    F.ZIP,
    F.BIRTH_DAY,
    COUNT(*) AS DUPLICATE
FROM
    RELMDB.FANS F
GROUP BY
    F.FNAME, F.LNAME, F.CITY, F.STATE, F.ZIP, F.BIRTH_DAY
HAVING
    COUNT(*) > 1;
```

2.8 Query 18

The movies database has two tables that contain data on fans (FANS_OLD and FANS). Due to a bug in our application, fans may have been entered into the old fans table rather then the new table. Find all fans that exist in the old fans table but not the new table. Use only the first and last name when comparing fans between the two tables.

```
SELECT
    *
FROM
    RELMDB.FANS_OLD
WHERE NOT EXISTS
    (SELECT
        *
    FROM
        RELMDB.FANS
    WHERE
        FANS.FNAME = FANS_OLD.FNAME AND
        FANS.LNAME = FANS_OLD.LNAME);
```

3 Group 3 Queries

3.1 Query 19

Assign breweries to groups based on the number of beers they brew. Display the brewery ID, name, number of beers they brew, and group number for each brewery. The group number should range from 1 to 4, with group 1 representing the top 25% of breweries, group 3 the next 25% and group 4 for the last 25%. Breweries with the most beers should be shown first. In the case of a tie, show breweries by brewery ID (lowest to highest).

```
SELECT
    b.brewery_id ,
    br.name,
    COUNT(b.brewery_id)  no_of_beer,
    NTILE(4) OVER(ORDER BY COUNT(br.brewery_id) DESC) rank_amount
FROM
    beerdb.beers b
    INNER JOIN beerdb.breweries br
    ON b.brewery_id = br.brewery_id
WHERE
    b.brewery_id IS NOT NULL
GROUP BY
    b.brewery_id,br.name, br.brewery_id
ORDER BY
    rank_amount,no_of_beer DESC, b.brewery_id ;
```

3.2 Query 20

Rank beers in descending order by their alcohol by volume (ABV) content. Only consider beers with an ABV greater than zero. Display the rank number, beer name, and ABV for all beers ranked 1-10. Do not leave any gaps in the ranking sequence when there are ties (e.g., 1, 2, 2, 2, 3, 4, 4, 5). (Hint: derived tables may help with this query).

```
SELECT
    ROW_NUMBER()
    OVER (ORDER BY ABV DESC) AS RANKING,
    B.BEER_NAME,
```

```sql
    B.ABV
FROM
    DB2SLATE.beerdb_beers B
WHERE
    ABV > 0
FETCH FIRST 10 ROWS ONLY;
```

3.3 Query 21

Display the film title, film year and worldwide gross for all movies directed by Christopher Nolan that have a worldwide gross greater than zero. In addition, each row should contain the cumulative worldwide gross (current row's worldwide gross plus the sum of all previous rows' worldwide gross). Records should be sorted in ascending order by film year.

```sql
SELECT
    movie_title,
    release_year,
    imdb_top250_rank,
    worldwide_gross,
    SUM(worldwide_gross) over (ORDER BY release_year
    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
    AS Cummulative_gross
FROM
    rmdb.movies m
    INNER JOIN rmdb.movie_directors md
        ON m.movie_guid = md.movie_guid
    INNER JOIN rmdb.persons USING(person_guid)
WHERE
    worldwide_gross IS NOT NULL AND
    person_name LIKE 'Alfred Hitchcock'
ORDER BY
    release_year;
```

4 Interesting Queries

4.1 Query 1

Display the genre, MPAA ratings, film count, total fan votes, average fan rating along with the total Budget and total Worldwide gross. Show the budget and worldwide gross in descending order. Only show entries with average fan ratings above 7 and film count greater than 200. Show genre entries with at least 100 films and entries without budget and worldwide gross must be excluded.

```sql
SELECT
    *
FROM (
    SELECT
        G.GENRE_NAME, M.MPAA_RATING,
        COUNT(FILM_ID) FILM_COUNT,
        SUM(IMDB_VOTES) FAN_VOTES,
        ROUND(AVG(F.IMDB_RATING),1) AVG_FAN_RATING,
        SUM(M.BUDGET) BUDGET,
        SUM(M.WORLDWIDE_GROSS) WORLDWIDE_GROSS
    FROM
        RELMDB.MOVIES M
        JOIN RELMDB.GENRES G
            USING (FILM_ID)
        JOIN RELMDB.FAN_RATINGS F
            USING (FILM_ID)
    WHERE
        BUDGET IS NOT NULL AND WORLDWIDE_GROSS IS NOT NULL
    GROUP BY
        GENRE_NAME, MPAA_RATING
    HAVING
        COUNT(G.GENRE_NAME) > 100
    ORDER BY
        GENRE_NAME ASC) T
WHERE
    AVG_FAN_RATING >= 7.0 AND FILM_COUNT >= 200
ORDER BY
    WORLDWIDE_GROSS DESC, BUDGET DESC;
```

4.2 Query 2

Display all the combinations of the category name, style name with beer count and average ABV. Only show entries with ABV above 7 and for united states. Include at least 20 beers for each category name. Then, label the strength of a beer count based on its ABV. The label should be ”Very High” for an ABV more than 10, ”High” for an ABV of 6 to 10, ”Average” for an ABV of 3 to 6, and ”Low” for an ABV less than 3. Show the beer count in descending order.

```sql
SELECT
    CATEGORY_NAME, STYLE_NAME, BEER_COUNT, AVG_ABV,
    CASE
        WHEN S.AVG_ABV > 10 THEN 'VERY HIGH'
        WHEN S.AVG_ABV BETWEEN 6 AND 10 THEN 'HIGH'
        WHEN S.AVG_ABV BETWEEN 3 AND 6 THEN 'AVERAGE'
        ELSE 'LOW'
    END AS STRENGTH
FROM (
    SELECT
        C.CATEGORY_NAME, S.STYLE_NAME,
        COUNT(BEER_NAME) BEER_COUNT,
        ROUND(AVG(BE.ABV),1) AVG_ABV
    FROM
        BEERDB.BEERS BE
        JOIN BEERDB.CATEGORIES C
            ON BE.CAT_ID = C.CATEGORY_ID
        JOIN BEERDB.BREWERIES B
            USING (BREWERY_ID)
        JOIN BEERDB.STYLES S
            USING (STYLE_ID)
    WHERE
        ABV > 0 AND COUNTRY IN ('United States')
        AND WEBSITE IS NOT NULL
    GROUP BY
        CATEGORY_NAME, STYLE_NAME
    HAVING
        COUNT(BE.BEER_NAME) > 20
    ORDER BY
        CATEGORY_NAME) S
WHERE
```

```
    BEER_COUNT > 40
ORDER BY
    BEER_COUNT DESC;
```

4.3 Query 3

Select Duplicate Person_Name who are not linked with any movies.

```
SELECT
    *
FROM
    rmdb.persons
WHERE
    person_guid IN
    (
        (
        SELECT
            person_guid
        FROM
            RMDB.persons
        WHERE
            person_name IN
            (SELECT
                person_name
            FROM
                RMDB.persons
            GROUP BY
                person_name
            HAVING
                count(person_name) > 1)
            )
            MINUS
            (
            SELECT
                person_guid
            FROM
                rmdb.movie_actors
```

```
                  )
                MINUS
                    (
                    SELECT person_guid
                    FROM
                        RMDB.movie_directors
                    )
                    MINUS
                    (
                    SELECT
                        person_guid
                    FROM
                        RMDB.movie_writers
                    )
                )
ORDER BY
    person_name;
```

Observation: There are 170 person_guid which are for duplicate person_name and are not linked with any movie. We have 204 duplicate person_name, this leaves us with 34 duplicate people with more than 1 person_guid linked with movies. Foe example this can be due to the fact that 1 guid can be linked as director and the other can be linked as actor.