In this exercise we're going to use the *OnlineCampus* database.

***OnlineCampus* is a fictional online open course platform that offers paid academic courses.**

The *OnlineCampus* database consists of a single table: *Courses* - which provides information about the various courses hosted by *OnlineCampus (1-10)*

1. **Write a query to display all of the information inside *courses* table**

```
-- Applies to MYSQL, MSSQL & POSTGRES
SELECT * FROM courses
```

2. **Write a query to display for each course - the *course_id*, *price_usd*, *num_students*, and *content_duration_minutes***

```
-- Applies to MYSQL, MSSQL & POSTGRES
SELECT course_id, price_usd, num_students, content_duration_minutes
FROM courses
```

3. **Write a query to display the :**
   a. *course_title*
   b. *num_students*
   c. *positive_reviews* + *negative_reviews* (name this calculation: `*total_reviews*` for example)
   d. A new column representing the ratio between the *total_reviews* & *num_students*: (*positive_reviews* + *negative_reviews*) / *num_students*

```
-- MYSQL
SELECT course_title,
       num_students,
       negative_reviews + negative_reviews as 'total_reviews',
       (negative_reviews + negative_reviews) / num_students AS 'reviews_ratio'
FROM courses
-- MSSQL
SELECT course_title,
       num_students,
       negative_reviews + negative_reviews as 'total_reviews',
       (negative_reviews + negative_reviews) / num_students AS 'reviews_ratio'
FROM courses
-- POSTGRES
SELECT course_title,
       num_students,
       negative_reviews + negative_reviews as "total_reviews",
       (negative_reviews + negative_reviews) / num_students AS "reviews_ratio"
FROM courses
```

4. **Write a query to display the *course_id*, *course_title*, *num_lessons*, *content_duration_minutes*, and a new column representing the average duration for each lecture (*content_duration_minutes* / *num_lessons*)**

```
-- MYSQL
SELECT course_id,
       course_title,
       num_lessons,
       content_duration_minutes,
       content_duration_minutes / num_lessons AS 'average_lecture_duration'
FROM courses
-- MSSQL
SELECT course_id,
       course_title,
```

```
       num_lessons,
       content_duration_minutes,
       content_duration_minutes / num_lessons AS 'average_lecture_duration'
FROM courses
-- POSTGRES
SELECT course_id,
       course_title,
       num_lessons,
       content_duration_minutes,
       content_duration_minutes / num_lessons AS "average_lecture_duration"
FROM courses
```

5. **Write a query to display the**
   a. course_id
   b. course_title
   c. num_students
   d. price_usd **(the column represents price in USD)**
   e. Total course revenues in USD (*num_students * price_usd*)

```
-- MYSQL
SELECT course_id,
       course_title,
       num_students,
       price_usd AS 'price_usd',
       num_students * price_usd  AS 'course_revenues_usd'
FROM courses
-- MSSQL
SELECT course_id,
       course_title,
       num_students,
       price_usd AS 'price_usd',
       num_students * price_usd  AS 'course_revenues_usd'
FROM courses
-- POSTGRES
SELECT course_id,
       course_title,
       num_students,
       price_usd AS "price_usd",
       num_students * price_usd  AS "course_revenues_usd"
FROM courses
```

6. **Write a query to display the *course_id*, and *course_subject* concatenated with *course_title***

```
-- MYSQL
SELECT course_id, CONCAT(course_subject, ' - ', course_title ) AS 'course_details'
FROM courses
-- MSSQL
SELECT course_id, course_subject + ' - '+ course_title AS 'course_details'
FROM courses
-- POSTGRES
SELECT course_id, course_subject || ' - ' || course_title AS "course_details"
FROM courses
```

7. **Write a query to display the *course_id*, *price_usd*, *content_duration_minutes*, and a new column representing the content duration in hours (*content_duration_minutes* / 60)**

```
-- MYSQL
SELECT course_id,
       price_usd,
       content_duration_minutes,
       content_duration_minutes / 60 AS 'content_duration_hours'
FROM courses
-- MSSQL
SELECT course_id,
```

```
       price_usd,
       content_duration_minutes,
       content_duration_minutes / 60 AS 'content_duration_hours'
FROM courses
-- POSTGRES
SELECT course_id,
       price_usd,
       content_duration_minutes,
       content_duration_minutes / 60 AS "content_duration_hours"
FROM courses
```

8. **Write a query to display the distinct values in *course_subject* column**

```
SELECT DISTINCT course_subject AS 'unique_list_of_subjects' FROM courses
-- MSSQL
SELECT DISTINCT course_subject AS 'unique_list_of_subjects' FROM courses
-- POSTGRES
SELECT DISTINCT course_subject AS "unique_list_of_subjects" FROM courses
```

9. **Write a query to display the unique values in *course_level* column**

```
-- MYSQL
SELECT DISTINCT course_level AS 'unique_list_of_levels' FROM courses
-- MSSQL
SELECT DISTINCT course_level AS 'unique_list_of_levels' FROM courses
-- POSTGRES
SELECT DISTINCT course_level AS "unique_list_of_levels" FROM courses
```

10. **Write a query to display the distinct combination of values in *course_subject* and *course_level* columns**

```
-- Applies to MYSQL, MSSQL & POSTGRES
SELECT DISTINCT course_subject, course_level
FROM courses
```

In this exercise we're going to use the *jobs_adverts* database.

The *jobs_adverts* database consists of a single table - *jobs*, which contains of a large number of rows, representing individual job ads.

Questions for this assignment (11 – 27)

11. ***[String Functions]***
    a. **Write a query to display the *job_id*, *job_title*, *salary_estimate_min*, *salary_estimate_max***
    b. **Repeat your last query, only this time display only the job adverts where *salary_estimate_max* is less than 105,000**

```
--  a.
-- Applies to MYSQL, MSSQL & POSTGRES
SELECT job_id, job_title, salary_estimate_min, salary_estimate_max
FROM jobs

-- b.
-- MYSQL
SELECT job_id, job_title, salary_estimate_min, salary_estimate_max
FROM jobs
WHERE REPLACE(salary_estimate_max,'K','') < 105
```

```
-- MSSQL
SELECT job_id, job_title, salary_estimate_min, salary_estimate_max
FROM jobs
WHERE REPLACE(salary_estimate_max,'K','') < 105
-- POSTGRES
SELECT job_id, job_title, salary_estimate_min, salary_estimate_max
FROM jobs
WHERE CAST(REPLACE(salary_estimate_max,'K','') AS INT) < 105
```

12. *[String Functions]*

Write a query to display the *company_name*, *company_rank*, *company_size_min*, and *company_size_max* for companies with more than 60 employees and less than 120 employees

```
-- MYSQL
SELECT DISTINCT company_name, company_rank, company_size_min, company_size_max
FROM jobs
WHERE REPLACE(company_size_min, ' Employees', '') > 60 AND
      REPLACE(company_size_max, ' Employees', '') < 120
-- MSSQL
SELECT DISTINCT company_name, company_rank, company_size_min, company_size_max
FROM jobs
WHERE REPLACE(company_size_min, ' Employees', '') > 60 AND
      REPLACE(company_size_max, ' Employees', '') < 120
-- POSTGRES
SELECT DISTINCT company_name, company_rank, company_size_min, company_size_max
FROM jobs
WHERE CAST(REPLACE(company_size_min, ' Employees', '') AS INT) > 60 AND
      CAST(REPLACE(company_size_max, ' Employees', '') AS INT) < 120
```

13. *[String Functions]*

Write a query to display the :

a. *job_id*

b. *job_title* in uppercase

c. *company_name* in lowercase

```
-- Applies to MYSQL, MSSQL & POSTGRES
SELECT job_id, UPPER(job_title) AS 'upper_jobtitle', LOWER(company_name) AS 'lower_companyname'
FROM jobs
```

14. *[String Functions]*

Write a query to display the:

a. *job_id*, *company_name*, *headquarters_of_company*

b. first letter of *company_name*

c. first letter of *headquarters_of_company*

```
-- MYSQL
SELECT job_id, company_name, headquarters_of_company,
       SUBSTRING(company_name,1,1)             AS 'company_name_first_letter',
       SUBSTRING(headquarters_of_company, 1,1) AS 'state_of_company_first_letter'
FROM jobs
-- MSSQL
SELECT job_id, company_name, headquarters_of_company,
       SUBSTRING(company_name,1,1)             AS 'company_name_first_letter',
       SUBSTRING(headquarters_of_company, 1,1) AS 'state_of_company_first_letter'
FROM jobs
-- POSTGRES
```

```
SELECT job_id, company_name, headquarters_of_company,
       SUBSTRING(company_name,1,1)             AS "company_name_first_letter",
       SUBSTRING(headquarters_of_company, 1,1) AS "state_of_company_first_letter"
FROM jobs
```

15. ***[String Functions]***

Write a query to display the :

a. *job_id*

b. *company_name*

c. *headquarters_of_company*

d. *company_code* - a new column containing a concatenation of: the first letter of *company_name* and the first letter of *headquarters_of_company*

For example : for Google located in Austin, the *company_code* would be: GA

```
-- MYSQL
SELECT job_id, company_name, headquarters_of_company,
       CONCAT(SUBSTRING(company_name,1,1), SUBSTRING(headquarters_of_company, 1,1))
'company_code'
FROM   jobs
-- MSSQL
SELECT job_id, company_name, headquarters_of_company,
       SUBSTRING(company_name,1,1) + SUBSTRING(headquarters_of_company, 1,1) AS 'company_code'
FROM   jobs
-- POSTGRES
SELECT job_id, company_name, headquarters_of_company,
       SUBSTRING(company_name,1,1) || SUBSTRING(headquarters_of_company, 1,1) AS "company_code"
FROM   jobs
```

16. ***[String Functions]***

Repeat your last query, only this time display the *company_code* in lowercase

```
-- MYSQL
SELECT job_id, company_name, headquarters_of_company,
       LOWER(CONCAT(SUBSTRING(company_name,1,1), SUBSTRING(headquarters_of_company, 1,1)))
'company_code'
FROM   jobs
-- MSSQL
SELECT job_id, company_name, headquarters_of_company,
       LOWER(SUBSTRING(company_name,1,1) + SUBSTRING(headquarters_of_company, 1,1)) AS
'company_code'
FROM   jobs
-- POSTGRES
SELECT job_id, company_name, headquarters_of_company,
       LOWER(SUBSTRING(company_name,1,1) || SUBSTRING(headquarters_of_company, 1,1)) AS
"company_code"
FROM   jobs
```

17. ***[String Functions]***

Write a query to display the job titles with length greater than 29

```
-- MYSQL
SELECT DISTINCT job_title
FROM jobs
```

```
WHERE LENGTH(job_title) > 29
-- MSSQL
SELECT DISTINCT job_title
FROM jobs
WHERE LEN(job_title) > 29
-- POSTGRES
SELECT DISTINCT job_title
FROM jobs
WHERE LENGTH(job_title) > 29
```

18. *[Numeric Functions]*

Write a query to display the:

a. *company_name*

b. *company_rank*

c. *company_market_value*

d. *company_market_value* rounded using ROUND function with precision of 2 digits

e. *company_market_value* rounded down using FLOOR

f. *company_market_value* rounded up using CEIL

```
-- MYSQL
SELECT company_name, company_rank, company_market_value,
       ROUND(company_market_value, 2) AS 'round_mv',
       FLOOR(company_market_value) AS 'floor_mv',
       CEIL(company_market_value) AS 'ceil_mv'
FROM   jobs
-- MSSQL
SELECT company_name, company_rank, company_market_value,
       ROUND(company_market_value, 2) AS 'round_mv',
       FLOOR(company_market_value) AS 'floor_mv',
       CEILING(company_market_value) AS 'ceil_mv'
FROM   jobs
-- POSTGRES
SELECT company_name, company_rank, company_market_value,
       ROUND(company_market_value, 2) AS "round_mv",
       FLOOR(company_market_value) AS "floor_mv",
       CEILING(company_market_value) AS "ceil_mv"
FROM   jobs
```

19. *[Date Functions]*

Write a query to display the *job_id*, *job_title*, *published_date*, and *removed_date* for all jobs that were published on 2016

```
-- MYSQL
SELECT job_id, job_title, published_date, removed_date
FROM jobs
WHERE YEAR(published_date) = 2016
-- MSSQL
SELECT job_id, job_title, published_date, removed_date
FROM jobs
WHERE YEAR(published_date) = 2016
-- POSTGRES
SELECT job_id, job_title, published_date, removed_date
FROM jobs
WHERE DATE_PART('year', published_date) = 2016
```

20. *[Date Functions]*

Which job adverts were posted during January 2017 ? Display columns you consider relevant

```
-- MYSQL
SELECT job_id, job_title, published_date, removed_date
FROM jobs
WHERE YEAR(published_date) = 2017 AND MONTH(published_date) = 1
-- MSSQL
SELECT job_id, job_title, published_date, removed_date
FROM jobs
WHERE YEAR(published_date) = 2017 AND MONTH(published_date) = 1
-- POSTGRES
SELECT job_id, job_title, published_date, removed_date
FROM jobs
WHERE DATE_PART('year', published_date) = 2017 AND DATE_PART('month', published_date) = 1
```

21.*[Date Functions]*

Which job adverts were removed after a single day ?

```
-- MYSQL OPTION 1
SELECT job_id, job_title, published_date, removed_date
FROM jobs
WHERE DATEDIFF(removed_date, published_date) = 1
-- MYSQL OPTION 2
SELECT job_id, job_title, published_date, removed_date
FROM jobs
WHERE TIMESTAMPDIFF(DAY, published_date, removed_date) = 1
-- MSSQL
SELECT job_id, job_title, published_date, removed_date
FROM jobs
WHERE DATEDIFF(DAY, published_date, removed_date) = 1
-- POSTGRES
SELECT job_id, job_title, published_date, removed_date
FROM jobs
WHERE DATE_PART('day', removed_date - published_date) = 1
```

22.*[Date Functions]*

Which job adverts were posted on the same day and month as the current date ?

For example, if today is February 11th 2021, which jobs were published on February 11th (regardless the year) ?

```
-- MYSQL
SELECT job_id, job_title, published_date, removed_date
FROM jobs
WHERE DAY(published_date) = DAY(CURDATE())
AND   MONTH(published_date) = MONTH(CURDATE())
-- MSSQL
SELECT job_id, job_title, published_date, removed_date
FROM jobs
WHERE DAY(published_date) = DAY(GETDATE())
AND   MONTH(published_date) = MONTH(GETDATE())
-- POSTGRES
SELECT job_id, job_title, published_date, removed_date
FROM jobs
WHERE DATE_PART('day', published_date) = DATE_PART('day',CURRENT_DATE)
AND   DATE_PART('month', published_date) = DATE_PART('month',CURRENT_DATE)
```

23.*[Date Functions]*

In a few job adverts, the value of *published_date* is greater than the *removed_date*, those rows represent invalid data.

Find those rows

```
-- MYSQL
SELECT job_id, job_title, published_date, removed_date
FROM jobs
WHERE published_date > removed_date
-- MSSQL
SELECT job_id, job_title, published_date, removed_date
FROM jobs
WHERE published_date > removed_date
-- POSTGRES
SELECT job_id, job_title, published_date, removed_date
FROM jobs
WHERE published_date > removed_date
```

24. *[NULL Related Functions]*

List the job adverts where at least one of the following conditions is met:

a. The row does not contain a value in *removed_date*

b. The row does not contain a value in *company_name*

c. The row does not contain a value in *headquarters_of_company*

```
-- Applies to MYSQL, MSSQL & POSTGRES
SELECT job_id, job_title, removed_date, company_name, headquarters_of_company
FROM jobs
WHERE removed_date IS NULL OR
      company_name IS NULL OR
      headquarters_of_company IS NULL
```

25. *[NULL Related Functions]*

Take your previous report and instead of the NULL values:

a. Display the current date instead of NULL values in *removed_date*

b. Display the company_state instead of NULL values in *headquarters_of_company*

c. Display 'Not Available' instead of NULL values in *company_name*

```
-- MYSQL
SELECT job_id, job_title,
       IFNULL(removed_date, CURDATE()) AS 'removed_date',
       IFNULL(headquarters_of_company, state_of_company) AS 'headquarters_of_company',
       IFNULL(company_name, 'Not Available') AS 'company_name'
FROM jobs
WHERE removed_date IS NULL OR
      company_name IS NULL OR
      headquarters_of_company IS NULL
-- MSSQL
SELECT job_id, job_title,
       ISNULL(removed_date, GETDATE()) AS 'removed_date',
       ISNULL(headquarters_of_company, state_of_company) AS 'headquarters_of_company',
       ISNULL(company_name, 'Not Available') AS 'company_name'
FROM jobs
WHERE removed_date IS NULL OR
      company_name IS NULL OR
      headquarters_of_company IS NULL
```

```
-- POSTGRES
SELECT job_id, job_title,
       COALESCE(removed_date, CURRENT_DATE) AS "removed_date",
       COALESCE(headquarters_of_company, state_of_company) AS "headquarters_of_company",
       COALESCE(company_name, 'Not Available') AS "company_name"
FROM jobs
WHERE removed_date IS NULL OR
      company_name IS NULL OR
      headquarters_of_company IS NULL
```

26. *[Case Statement]*

Write a query to display the *company_name*, *company_market_value*, and a new column : *company_market_value_rank*, based on the following logic:

a. For companies with *market_value* in the range of 0-300 provide the rank : 'low range'

b. For companies with *market_value* in the range of 301-600 provide the rank : 'mid range'

c. For companies with *market_value* in the range of 601-900 provide the rank : 'high range'

d. For any other range provide the rank : 'other range'

```
-- MYSQL
SELECT DISTINCT company_name, company_market_value,
                CASE WHEN company_market_value BETWEEN 0 AND 300 THEN   'low range'
                          WHEN company_market_value BETWEEN 301 AND 600 THEN 'mid range'
             WHEN company_market_value BETWEEN 601 AND 900 THEN 'high range'
                          ELSE 'other range'
                END AS 'company_market_value_rank'
FROM   jobs
-- MSSQL
SELECT DISTINCT company_name, company_market_value,
            CASE WHEN company_market_value BETWEEN 0 AND 300 THEN   'low range'
                   WHEN company_market_value BETWEEN 301 AND 600 THEN 'mid range'
             WHEN company_market_value BETWEEN 601 AND 900 THEN 'high range'
                   ELSE 'other range'
            END AS 'company_market_value_rank'
FROM   jobs
-- POSTGRES
SELECT DISTINCT company_name, company_market_value,
            CASE WHEN company_market_value BETWEEN 0 AND 300 THEN   'low range'
                   WHEN company_market_value BETWEEN 301 AND 600 THEN 'mid range'
             WHEN company_market_value BETWEEN 601 AND 900 THEN 'high range'
                   ELSE 'other range'
            END AS "company_market_value_rank"
FROM   jobs
```

27. *[Case Statement]*

Write a query to display
the *job_title*, *company_name*, *company_size_min*, *company_size_max*, and a new column - *company_size*, based on the following logic:

a. For companies with up to 60 employees, provide the value: 'Small Company'

b. For companies with up to 120 employees, provide the value: 'Medium Company'

c. For companies with up to 180 employees, provide the value: ' Large Company'

d. For any other range 'Unknown'

```
-- MYSQL
SELECT job_title, company_name, company_size_min, company_size_max,
           CASE WHEN REPLACE(company_size_max, ' Employees', '') <= 60 THEN 'Small Company'
                        WHEN REPLACE(company_size_max, ' Employees', '') <=  120 THEN 'Medium
Company'
            WHEN REPLACE(company_size_max, ' Employees', '') <=  180 THEN 'Large Company'
            ELSE 'Unknown'
                 END AS 'company_size'
FROM   jobs
-- MSSQL
SELECT job_title, company_name, company_size_min, company_size_max,
         CASE WHEN REPLACE(company_size_max, ' Employees', '') <= 60 THEN 'Small Company'
                  WHEN REPLACE(company_size_max, ' Employees', '') <=  120 THEN 'Medium
Company'
            WHEN REPLACE(company_size_max, ' Employees', '') <=  180 THEN 'Large Company'
            ELSE 'Unknown'
            END AS 'company_size'
FROM   jobs
-- POSTGRES
SELECT job_title, company_name, company_size_min, company_size_max,
         CASE WHEN CAST(REPLACE(company_size_max, ' Employees', '') AS INT) <= 60 THEN 'Small
Company'
                  WHEN CAST(REPLACE(company_size_max, ' Employees', '') AS INT) <=  120 THEN
'Medium Company'
            WHEN CAST(REPLACE(company_size_max, ' Employees', '') AS INT) <=  180 THEN 'Large
Company'
            ELSE 'Unknown'
            END AS "company_size"
FROM   jobs
```

In this exercise we're going to use The *Netflix* database.

***Netflix* database consists of tv shows and movies available on Netflix as of 2019.**

In this database you'll find two tables:

- ***movies* - lists all of the relevant information regarding Netflix's movies**
- ***series* - lists all of the relevant information regarding Netflix's series**

28. [movies] Write a query to display the *title*, *rating*, *country*, *release_year*, and *director*, for all movies directed by Michael Bay.
Sort the output by *release_year* (ascending)

```
-- Applies to MYSQL, MSSQL & POSTGRES
```

```
SELECT title, rating, country, release_year, director
FROM movies
WHERE director = 'Michael Bay'
ORDER BY release_year
```

29. **[movies] Write a query to display the *title*, *country*, *duration_in_minutes*, and *date_added*, for all movies that were added before March 2011. Sort the output by *duration_in_minutes* (ascending)**

```
-- Applies to MYSQL, MSSQL & POSTGRES
SELECT title, country, duration_in_minutes, date_added
FROM movies
WHERE date_added < '2011-03-01'
ORDER BY duration_in_minutes
```

30. **[movies] Write a query to display the *title*, *country*, *duration_in_minutes*, and *release_year*, for all movies that were released between 2014 and 2016. Sort the output by *duration_in_minutes* (descending)**

```
-- Applies to MYSQL, MSSQL & POSTGRES
SELECT title, country, duration_in_minutes, release_year
FROM movies
WHERE release_year BETWEEN 2014 AND 2016
ORDER BY duration_in_minutes DESC
```

31. **[movies] Write a query to display the *title*, *director*, *country*, and *duration_in_minutes*,**

for all movies with duration between 3-4 hours. Sort the output by *duration_in_minutes* (descending)

```
-- Applies to MYSQL, MSSQL & POSTGRES
SELECT title, director, country, duration_in_minutes
FROM movies
WHERE duration_in_minutes/60 BETWEEN 3 AND 4
```

32. **[series] Write a query to display the *title*, *director*, *rating*, *num_of_seasons* for all series with 10 to 14 seasons. Sort the output by *num_of_seasons* (descending)**

```
-- Applies to MYSQL, MSSQL & POSTGRES
SELECT title, director, rating, num_of_seasons
FROM series
WHERE num_of_seasons BETWEEN 10 AND 14
ORDER BY num_of_seasons DESC
```

33. **[series] Write a query to display the *title*, *director*, *rating*, *num_of_seasons* for all series**

a. containing value in the *director* column (*diretor* not equals null)

b. and having *num_of_seasons* > 7

```
-- Applies to MYSQL, MSSQL & POSTGRES
SELECT title, director, rating, num_of_seasons
FROM series
WHERE director IS NOT NULL AND num_of_seasons > 7
```

34. **[movies] Write a query to display the *title*, *director*, *cast*, *country* and *rating*, for all movies having Ryan Reynolds in their cast**

```
-- Applies to MYSQL, MSSQL & POSTGRES
SELECT title, director, cast, country, rating
FROM movies
WHERE cast LIKE '%Ryan Reynolds%'
```

35. **[movies] Write a query to display the *title*, *director*, *cast*, *country* and *rating*, for all movies having Ryan Reynolds and Nicolas Cage in their cast**

```
-- Applies to MYSQL, MSSQL & POSTGRES
SELECT title, director, cast, country, rating
FROM movies
WHERE cast LIKE '%Ryan Reynolds%' AND cast LIKE '%Nicolas Cage%'
```

36. **[movies] Write a query to display the *title*, *director*, *cast*, *country*, *duration_in_minutes*, and *rating***

    a. for all movies having PG (Parental Guidance) in their *rating*

    b. and movie duration is greater than 3 hours

```
-- Applies to MYSQL, MSSQL & POSTGRES
SELECT title, director, cast, country, duration_in_minutes, rating
FROM movies
WHERE rating LIKE '%PG%' and duration_in_minutes > 180
```

37. **[series] Write a query to display *title*, *director*, *cast*, *country*, and *release_year* for all series released in 2014, 2016, or 2018. Sort the output by title (ascending)**

```
-- Applies to MYSQL, MSSQL & POSTGRES
SELECT title, director, cast, country, release_year
FROM series
WHERE release_year IN (2014,2016,2018)
```

In this exercise we're going to use the *Spotify* database.

Spotify is a digital music streaming service that gives you access to millions of songs, podcasts and videos from artists all over the world.

The *Spotify* database consists of a single table - *tracks*, which contains audio statistics of the top 2000 tracks on *Spotify*. The table contains about 15 columns each describing the track and it's qualities.

Column Description
Please find below a description of each column:

- **track_id**: Track number
- **title**: Name of the Track
- **artist**: Name of the Artist
- **genre**: Genre of the track
- **year**: Release year of the track
- **danceability**: The higher the value, the easier it is to dance to this song (scale of 0-100).
- **duration**: The duration of the song (seconds).
- **popularity**: The higher the value the more popular the song is (scale of 0-100).

38. List the number of tracks made by each *artist*. Sort the output by the number of tracks (Descending)

```
--MYSQL
SELECT artist, COUNT(*) AS 'number_of_tracks'
FROM tracks
GROUP BY artist
ORDER BY COUNT(*) DESC
--MSSQL
SELECT artist, COUNT(*) AS 'number_of_tracks'
FROM tracks
GROUP BY artist
ORDER BY COUNT(*) DESC
--POSTGRES
SELECT artist, COUNT(*) AS "number_of_tracks"
FROM tracks
GROUP BY artist
ORDER BY COUNT(*) DESC
```

39. Display the average duration of tracks by *genre*. Sort the output by the average duration (Descending)

```
--MYSQL
SELECT genre, AVG(duration) AS 'average_duraion'
FROM tracks
GROUP BY genre
ORDER BY AVG(duration) DESC
--MSSQL
SELECT genre, AVG(duration) AS 'average_duraion'
FROM tracks
GROUP BY genre
ORDER BY AVG(duration) DESC
--POSTGRES
SELECT genre, AVG(duration) AS "average_duraion"
FROM tracks
GROUP BY genre
ORDER BY AVG(duration) DESC
```

40. Display the minimum, maximum, and average danceability of tracks made by Queen and The Beatles

```
--MYSQL
SELECT artist,
       MIN(danceability) AS 'min_danceability',
       MAX(danceability) AS 'max_danceability',
       AVG(danceability) AS 'avg_danceability'
FROM tracks
WHERE artist IN ('Queen', 'The Beatles')
GROUP BY artist
--MSSQL
SELECT artist,
       MIN(danceability) AS 'min_danceability',
       MAX(danceability) AS 'max_danceability',
       AVG(danceability) AS 'avg_danceability'
FROM tracks
WHERE artist IN ('Queen', 'The Beatles')
GROUP BY artist

--POSTGRES
SELECT artist,
       MIN(danceability) AS "min_danceability",
       MAX(danceability) AS "max_danceability",
       AVG(danceability) AS "avg_danceability"
FROM tracks
WHERE artist IN ('Queen', 'The Beatles')
```

```
GROUP BY artist
```

41. **Pop music consists of different genres, for example: Art Pop, Dance Pop, and Candy Pop are all Pop music genres.**
    How many pop music genres are listed in this dataset ?

```
--MYSQL
SELECT COUNT(DISTINCT genre) 'unique_pop_genres'
FROM tracks
WHERE genre LIKE '%pop%'
--MSSQL
SELECT COUNT(DISTINCT genre) 'unique_pop_genres'
FROM tracks
WHERE genre LIKE '%pop%'
--POSTGRES
SELECT COUNT(DISTINCT genre) "unique_pop_genres"
FROM tracks
WHERE genre LIKE '%pop%'
```

42. **Display the number of tracks, highest *popularity*, and lowest *popularity* each rock music *artist* has achieved.**
    Sort the output by the number of tracks (descending)

```
--MYSQL
SELECT artist,
       COUNT(*) AS 'number_of_tracks',
       MAX(popularity) AS 'highest_popularity',
       MIN(popularity) AS 'lowest_popularity'
FROM tracks
WHERE genre LIKE '%rock%'
GROUP BY artist
ORDER BY COUNT(*) DESC
--MSSQL
SELECT artist,
       COUNT(*) AS 'number_of_tracks',
       MAX(popularity) AS 'highest_popularity',
       MIN(popularity) AS 'lowest_popularity'
FROM tracks
WHERE genre LIKE '%rock%'
GROUP BY artist
ORDER BY COUNT(*) DESC
--POSTGRES
SELECT artist,
       COUNT(*) AS "number_of_tracks",
       MAX(popularity) AS "highest_popularity",
       MIN(popularity) AS "lowest_popularity"
FROM tracks
WHERE genre LIKE '%rock%'
GROUP BY artist
ORDER BY COUNT(*) DESC
```

43. **Tracks by genre**

    a. List the number of tracks by each genre, for tracks released during 2005-2010.

    b. Further restrict your result to display only genres with more than 10 tracks

```
--   a. List the number of tracks by each genre, for tracks released during 2005-2010.
--MYSQL
SELECT genre, COUNT(*) AS 'number_of_tracks'
FROM tracks
WHERE release_year BETWEEN 2005 AND 2010
GROUP BY genre
--MSSQL
SELECT genre, COUNT(*) AS 'number_of_tracks'
FROM tracks
WHERE release_year BETWEEN 2005 AND 2010
GROUP BY genre
--POSTGRES
SELECT genre, COUNT(*) AS "number_of_tracks"
FROM tracks
WHERE release_year BETWEEN 2005 AND 2010
GROUP BY genre

-- b. Further restrict your result to display only genres with more than 10 tracks
--MYSQL
SELECT genre, COUNT(*) AS 'number_of_tracks'
FROM tracks
WHERE release_year BETWEEN 2005 AND 2010
GROUP BY genre
HAVING COUNT(*) > 10
--MSSQL
SELECT genre, COUNT(*) AS 'number_of_tracks'
FROM tracks
WHERE release_year BETWEEN 2005 AND 2010
GROUP BY genre
HAVING COUNT(*) > 10
--POSTGRES
SELECT genre, COUNT(*) AS "number_of_tracks"
FROM tracks
WHERE release_year BETWEEN 2005 AND 2010
GROUP BY genre
HAVING COUNT(*) > 10
```

44. List the number of tracks released by Coldplay each year. Sort the output by *release_year* (ascending)

```
--MYSQL
SELECT artist, release_year, COUNT(*) AS 'number_of_tracks'
FROM tracks
WHERE artist = 'Coldplay'
GROUP BY artist, release_year
--MSSQL
SELECT artist, release_year, COUNT(*) AS 'number_of_tracks'
FROM tracks
WHERE artist = 'Coldplay'
GROUP BY artist, release_year
--POSTGRES
SELECT artist, release_year, COUNT(*) AS "number_of_tracks"
FROM tracks
WHERE artist = 'Coldplay'
GROUP BY artist, release_year
```

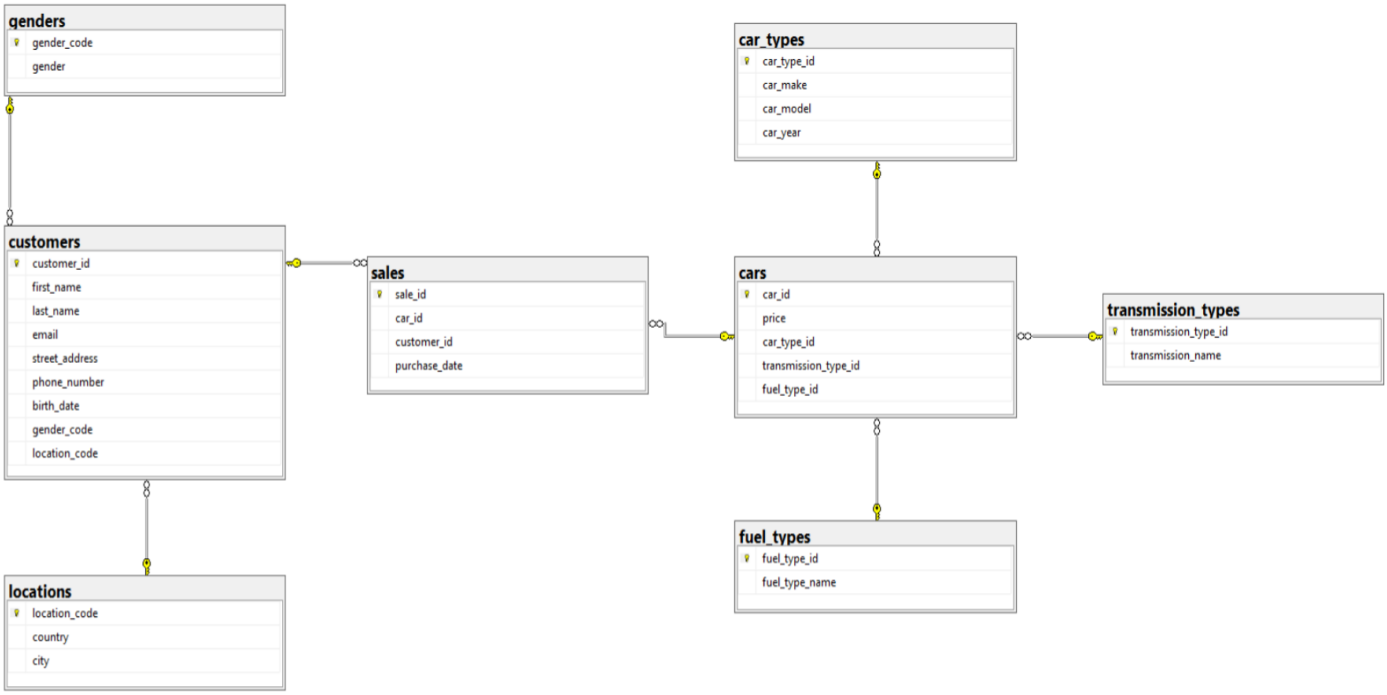
In this concluding exercise we're going to use the *CarsOnline* dataset

Have you found the car just right for you, but it's miles away? No worries, just contact us at *CarsOnline*, buy the car online and your car will be delivered right to your door.

***CarsOnline* is a fictional online platform, allowing customers to find the right car for them, buy it online, and get it right at their doorstep.**

The *CarsOnline* consists of the following tables:

- ***cars* - Provides details about the various cars (both sold and unsold)**
- ***car_types* - *Provides further details about the car type (i.e., Ford Mustang 1980)***
- **fuel_types - Provides further details about the car fuel type (i.e., Diesel, Electric etc.)**
- ***transmission_types* - Provides further details about the car gearbox type**
- ***customers* - Provides details about the company's customers who have bought at least one car**
- ***genders* - Provides further details about the gender of each customer**
- ***locations* - Provides further details about the location of each customer**
- ***sales* - Provides details about each car purchase**



45. [*cars* and *car_types*]

a. Write a query to display the *car_id*, *price*, *car_make*, *car_model*, and *car_year* for each car

b. restrict your query to display only cars made by bmw on 2019

c. Display the average price for each car model during this year

```
-- a. Write a query to display the car_id, price, car_make, car_model, and car_year for each
car
--Applies to MYSQL, MSSQL and POSTGRES
SELECT c.car_id, c.price, ct.car_make, ct.car_model, ct.car_year
FROM cars c JOIN car_types ct
ON   c.car_type_id = ct.car_type_id
```

```
-- b. restrict your query to display only cars made by bmw on 2019
--Applies to MYSQL, MSSQL and POSTGRES
SELECT c.car_id, c.price, ct.car_make, ct.car_model, ct.car_year
FROM cars c JOIN car_types ct
ON   c.car_type_id = ct.car_type_id
WHERE ct.car_make = 'bmw' AND ct.car_year BETWEEN 2015 AND 2019

-- c. Display the average price for each car model during this year
--Applies to MYSQL, MSSQL and POSTGRES
SELECT ct.car_model, AVG(c.price) AS 'average_price'
FROM cars c JOIN car_types ct
ON   c.car_type_id = ct.car_type_id
WHERE ct.car_make = 'bmw' AND ct.car_year = 2019
GROUP BY ct.car_model
```

46. [*cars* and *fuel_types*]

a. Write a query to display the *car_id*, *price* and *fuel_type_name* for each car

b. Display the number of cars by each *fuel_type_name*. Sort the output by the number of cars (descending)

```
-- a. Write a query to display the car_id, price and fuel_type_name for each car
--Applies to MYSQL, MSSQL and POSTGRES
SELECT c.car_id, c.price, ft.fuel_type_name
FROM cars c JOIN fuel_types ft
ON   c.fuel_type_id = ft.fuel_type_id

-- b. Display the number of cars by each fuel_type_name. Sort the output by the number of cars
(descending)
--MYSQL
SELECT ft.fuel_type_name, COUNT(*) AS 'number_of_cars'
FROM cars c JOIN fuel_types ft
ON   c.fuel_type_id = ft.fuel_type_id
GROUP BY ft.fuel_type_name
ORDER BY COUNT(*) DESC
--MSSQL
SELECT ft.fuel_type_name, COUNT(*) AS 'number_of_cars'
FROM cars c JOIN fuel_types ft
ON   c.fuel_type_id = ft.fuel_type_id
GROUP BY ft.fuel_type_name
ORDER BY COUNT(*) DESC
--POSTGRES
SELECT ft.fuel_type_name, COUNT(*) AS "number_of_cars"
FROM cars c JOIN fuel_types ft
ON   c.fuel_type_id = ft.fuel_type_id
GROUP BY ft.fuel_type_name
ORDER BY COUNT(*) DESC
```

47. [*cars* and *transmission_types*]

a. Write a query to display the *car_id*, *price* and *transmission_name* for each car

b. Display the average price for each *transmission_name*. Sort the output by the average price (descending)

```
-- a. Write a query to display the car_id, price and transmission_name for each car
--Applies to MYSQL, MSSQL and POSTGRES
SELECT c.car_id, c.price, tt.transmission_name
FROM cars c JOIN transmission_types tt
ON   c.transmission_type_id = tt.transmission_type_id

-- b. Display the average price for each transmission_name. Sort the output by the average
price (descending)
--MYSQL
SELECT tt.transmission_name, AVG(c.price) AS 'average_price'
FROM cars c JOIN transmission_types tt
ON   c.transmission_type_id = tt.transmission_type_id
GROUP BY tt.transmission_name
ORDER BY AVG(c.price) DESC
--MSSQL
SELECT tt.transmission_name, AVG(c.price) AS 'average_price'
FROM cars c JOIN transmission_types tt
ON   c.transmission_type_id = tt.transmission_type_id
GROUP BY tt.transmission_name
ORDER BY AVG(c.price) DESC
--POSTGRES
SELECT tt.transmission_name, AVG(c.price) AS "average_price"
FROM cars c JOIN transmission_types tt
ON   c.transmission_type_id = tt.transmission_type_id
GROUP BY tt.transmission_name
ORDER BY AVG(c.price) DESC
```

48. [*cars*, *car_types* and *fuel_types*]

Write a query to display the unique number of hybrid cars for each car_make. Sort the output by the number of cars (Descending)

```
--MYSQL
SELECT ct.car_make, COUNT(DISTINCT ct.car_model) AS 'number_of_hybrid_cars'
FROM cars c JOIN fuel_types ft
ON   c.fuel_type_id = ft.fuel_type_id
                        JOIN car_types ct
ON   c.car_type_id = ct.car_type_id
WHERE ft.fuel_type_name = 'Hybrid'
GROUP BY car_make
ORDER BY COUNT(DISTINCT car_model) DESC
--MSSQL
SELECT ct.car_make, COUNT(DISTINCT ct.car_model) AS 'number_of_hybrid_cars'
FROM cars c JOIN fuel_types ft
ON   c.fuel_type_id = ft.fuel_type_id
                        JOIN car_types ct
ON   c.car_type_id = ct.car_type_id
WHERE ft.fuel_type_name = 'Hybrid'
GROUP BY car_make
ORDER BY COUNT(DISTINCT car_model) DESC
--POSTGRES
SELECT ct.car_make, COUNT(DISTINCT ct.car_model) AS "number_of_hybrid_cars"
FROM cars c JOIN fuel_types ft
ON   c.fuel_type_id = ft.fuel_type_id
                        JOIN car_types ct
ON   c.car_type_id = ct.car_type_id
WHERE ft.fuel_type_name = 'Hybrid'
GROUP BY car_make
```

```
ORDER BY COUNT(DISTINCT car_model) DESC
```

49. [*cars*, *car_types* and *transmission_types*]

Write a query to display the number of manual-gearbox cars, by each car_year and car_make. Sort the output by the year (ascending)

```
--MYSQL
SELECT ct.car_year, ct.car_make, COUNT(*) AS 'number_of_cars'
FROM cars c JOIN transmission_types tt
ON   c.transmission_type_id = tt.transmission_type_id
                        JOIN car_types ct
ON   c.car_type_id = ct.car_type_id
WHERE tt.transmission_name = 'Manual'
GROUP BY ct.car_year, ct.car_make
ORDER BY ct.car_year
--MSSQL
SELECT ct.car_year, ct.car_make, COUNT(*) AS 'number_of_cars'
FROM cars c JOIN transmission_types tt
ON   c.transmission_type_id = tt.transmission_type_id
                        JOIN car_types ct
ON   c.car_type_id = ct.car_type_id
WHERE tt.transmission_name = 'Manual'
GROUP BY ct.car_year, ct.car_make
ORDER BY ct.car_year
--POSTGRES
SELECT ct.car_year, ct.car_make, COUNT(*) AS "number_of_cars"
FROM cars c JOIN transmission_types tt
ON   c.transmission_type_id = tt.transmission_type_id
                        JOIN car_types ct
ON   c.car_type_id = ct.car_type_id
WHERE tt.transmission_name = 'Manual'
GROUP BY ct.car_year, ct.car_make
ORDER BY ct.car_year
```

50. [*customers* and *genders*]

a. Write a query to display the *customer_id*, *first_name*, *last_name*, *birth_date*, and *gender*

b. Display the number of customers by each gender

c. Display the number of customers by each *gender* and age. Sort the output by the number of customers (descending)

d. Restrict your query to for customers above the age 59

```
--    a. Write a query to display the customer_id, first_name, last_name, birth_date, and
gender
--Applies to MYSQL, MSSQL and POSTGRES
SELECT c.customer_id, c.first_name, c.last_name, c.birth_date, g.gender
FROM customers c JOIN genders g
ON   c.gender_code = g.gender_code

--    b. Display the number of customers by each gender
```

```
--MYSQL
SELECT g.gender, COUNT(*) AS 'number_of_customers'
FROM customers c JOIN genders g
ON   c.gender_code = g.gender_code
GROUP BY g.gender
--MSSQL
SELECT g.gender, COUNT(*) AS 'number_of_customers'
FROM customers c JOIN genders g
ON   c.gender_code = g.gender_code
GROUP BY g.gender
--POSTGRES
SELECT g.gender, COUNT(*) AS "number_of_customers"
FROM customers c JOIN genders g
ON   c.gender_code = g.gender_code
GROUP BY g.gender

--    c. Display the number of customers by each gender and age. Sort the output by the number
of customers (descending)
--MYSQL
SELECT g.gender,
           TIMESTAMPDIFF(YEAR, c.birth_date, CURDATE()) AS 'age',
           COUNT(*) AS 'number_of_customers'
FROM customers c JOIN genders g
ON   c.gender_code = g.gender_code
GROUP BY g.gender, TIMESTAMPDIFF(YEAR, c.birth_date, CURDATE())
ORDER BY COUNT(*) DESC
--MSSQL
SELECT g.gender,
           DATEDIFF(YEAR, c.birth_date, GETDATE()) AS 'age',
       COUNT(*) AS 'number_of_customers'
FROM customers c JOIN genders g
ON   c.gender_code = g.gender_code
GROUP BY g.gender, DATEDIFF(YEAR, c.birth_date, GETDATE())
ORDER BY COUNT(*) DESC
--POSTGRES
SELECT g.gender,
       DATE_PART('year', CURRENT_DATE) - DATE_PART('year', c.birth_date) AS "age",
       COUNT(*) AS "number_of_customers"
FROM customers c JOIN genders g
ON   c.gender_code = g.gender_code
GROUP BY g.gender, DATE_PART('year', CURRENT_DATE) - DATE_PART('year', c.birth_date)
ORDER BY COUNT(*) DESC

-- d. Restrict your query to for customers above the age 59
--MYSQL
SELECT g.gender, TIMESTAMPDIFF(YEAR, c.birth_date, CURDATE()) AS 'age', COUNT(*) AS
'number_of_customers'
FROM customers c JOIN genders g
ON   c.gender_code = g.gender_code
WHERE TIMESTAMPDIFF(YEAR, c.birth_date, CURDATE()) > 59
GROUP BY g.gender, TIMESTAMPDIFF(YEAR, c.birth_date, CURDATE())
ORDER BY COUNT(*) DESC
--MSSQL
SELECT g.gender, DATEDIFF(YEAR, c.birth_date, GETDATE()) AS 'age', COUNT(*) AS
'number_of_customers'
FROM customers c JOIN genders g
ON   c.gender_code = g.gender_code
WHERE DATEDIFF(YEAR, c.birth_date, GETDATE()) > 59
GROUP BY g.gender, DATEDIFF(YEAR, c.birth_date, GETDATE())
ORDER BY COUNT(*) DESC
--POSTGRES
SELECT g.gender, DATE_PART('year', CURRENT_DATE) - DATE_PART('year', c.birth_date) AS "age",
       COUNT(*) AS "number_of_customers"
FROM customers c JOIN genders g
ON   c.gender_code = g.gender_code
```

```
WHERE DATE_PART('year', CURRENT_DATE) - DATE_PART('year', c.birth_date) > 59
GROUP BY g.gender, DATE_PART('year', CURRENT_DATE) - DATE_PART('year', c.birth_date)
ORDER BY COUNT(*) DESC
```

51. [*customers* and *locations*]

a. Write a query to display the number of customers living in Australia.

b. Write another query to display the number of customers with updated *phone_number* living in Australia (customers who has value in phone_number)

c. Write another query to display the number of australian customers with NULL value in their *phone_number*, break down the result for each city, sort it by the count (descending).

```
-- a. Write a query to display the number of customers living in Australia.
--MYSQL
SELECT COUNT(*) AS 'number_of_customers'
FROM customers c JOIN locations l
ON   c.location_code = l.location_code
WHERE l.country = 'Australia'
--MSSQL
SELECT COUNT(*) AS 'number_of_customers'
FROM customers c JOIN locations l
ON   c.location_code = l.location_code
WHERE l.country = 'Australia'
--POSTGRES
SELECT COUNT(*) AS "number_of_customers"
FROM customers c JOIN locations l
ON   c.location_code = l.location_code
WHERE l.country = 'Australia'

-- b. Write another query to display the number of customers with updated phone_number living
in Australia
--    (customers who has value in phone_number)
--MYSQL
SELECT COUNT(phone_number) AS 'number_of_customers'
FROM customers c JOIN locations l
ON   c.location_code = l.location_code
WHERE l.country = 'Australia'
--MSSQL
SELECT COUNT(phone_number) AS 'number_of_customers'
FROM customers c JOIN locations l
ON   c.location_code = l.location_code
WHERE l.country = 'Australia'
--POSTGRES
SELECT COUNT(phone_number) AS "number_of_customers"
FROM customers c JOIN locations l
ON   c.location_code = l.location_code
WHERE l.country = 'Australia'

-- c. Write another query to display the number of australian customers with NULL value in
their phone_number,
--    break down the result for each city, sort it by the count (descending).
--MYSQL
SELECT city, COUNT(*) AS 'number_of_customers'
FROM customers c JOIN locations l
ON   c.location_code = l.location_code
WHERE l.country = 'Australia' AND phone_number IS NULL
```

```
GROUP BY city
ORDER BY COUNT(*) DESC
--MSSQL
SELECT city, COUNT(*) AS 'number_of_customers'
FROM customers c JOIN locations l
ON   c.location_code = l.location_code
WHERE l.country = 'Australia' AND phone_number IS NULL
GROUP BY city
ORDER BY COUNT(*) DESC
--POSTGRES
SELECT city, COUNT(*) AS "number_of_customers"
FROM customers c JOIN locations l
ON   c.location_code = l.location_code
WHERE l.country = 'Australia' AND phone_number IS NULL
GROUP BY city
ORDER BY COUNT(*) DESC
```

52. [*sales* and *customers*]

Write a query to display the customer_id and full name of customers who bought more than 5 cars

```
--MYSQL
SELECT c.customer_id, CONCAT(c.first_name, ' ', c.last_name) AS 'full_name', COUNT(*) AS
'number_of_cars'
FROM sales s JOIN customers c
ON s.customer_id = c.customer_id
GROUP BY c.customer_id, CONCAT(c.first_name, ' ', c.last_name)
HAVING COUNT(*) > 5
--MSSQL
SELECT c.customer_id, CONCAT(c.first_name, ' ', c.last_name) AS 'full_name', COUNT(*) AS
'number_of_cars'
FROM sales s JOIN customers c
ON s.customer_id = c.customer_id
GROUP BY c.customer_id, CONCAT(c.first_name, ' ', c.last_name)
HAVING COUNT(*) > 5
--POSTGRES
SELECT c.customer_id, c.first_name || ' ' || c.last_name AS "full_name", COUNT(*) AS
"number_of_cars"
FROM sales s JOIN customers c
ON s.customer_id = c.customer_id
GROUP BY c.customer_id, c.first_name || ' ' || c.last_name
HAVING COUNT(*) > 5
```

53. [*sales* and *cars*]

Not every car on the *cars* table has been sold. Write a query to display the percent of sold cars.

```
--MYSQL
SELECT COUNT(s.customer_id) / COUNT(*) * 100 AS 'percent_of_sold_cars'
FROM sales s RIGHT OUTER JOIN cars c
```

```
ON s.car_id = c.car_id
--MSSQL
SELECT COUNT(s.customer_id) / CAST(COUNT(*) AS DECIMAL) * 100 AS 'percent_of_sold_cars'
FROM sales s RIGHT OUTER JOIN cars c
ON s.car_id = c.car_id
--POSTGRES
SELECT COUNT(s.customer_id) / CAST(COUNT(*) AS DECIMAL) * 100 AS "percent_of_sold_cars"
FROM sales s RIGHT OUTER JOIN cars c
ON s.car_id = c.car_id
```

54. [*sales*, *cars* and *car_types*]

On 2019 (of purchase date), What was the average price of sold cars made by Audi?

```
--MYSQL
SELECT AVG(cr.price) AS 'average_price'
FROM sales s JOIN cars cr
ON s.car_id = cr.car_id
                           JOIN car_types ct
ON  ct.car_type_id = cr.car_type_id
WHERE YEAR(s.purchase_date) = 2019 AND ct.car_make = 'Audi'
--MSSQL
SELECT AVG(cr.price) AS 'average_price'
FROM sales s JOIN cars cr
ON s.car_id = cr.car_id
                           JOIN car_types ct
ON  ct.car_type_id = cr.car_type_id
WHERE YEAR(s.purchase_date) = 2019 AND ct.car_make = 'Audi'
--POSTGRES
SELECT AVG(cr.price) AS "average_price"
FROM sales s JOIN cars cr
ON s.car_id = cr.car_id
                           JOIN car_types ct
ON  ct.car_type_id = cr.car_type_id
WHERE YEAR(s.purchase_date) = 2019 AND ct.car_make = 'Audi'
```

55. In this exercise we're going to use the *AppStore* database.

AppStore is a fictional online platform, allowing users to download different apps on their cell phone.

The *AppStore* database consists of a single table - *apps*, which contains information about the different available apps.
Write a query to display the *app_id*, *app_name*, *category* and *reviews* for apps with more reviews than *app_id* 64

```
SELECT app_id, app_name, category, reviews
FROM apps
WHERE reviews > (SELECT reviews FROM apps WHERE app_id = 64)
```

56. **Write a query to display the *app_name*, *category*, *size_in_mb*, and *rating*, for apps in the same category as Redhold (app_name)**

```
-- Applies to MYSQL, MSSQL and POSTGRES
SELECT app_name, category, size_in_mb, rating
FROM apps
WHERE category = (SELECT category FROM apps WHERE app_name = 'Redhold')
```

57. **Write a query to display the *app_name*, *category*, *app_version*, and *last_updated*, for apps which were *last_updated* before app_id 29**

```
-- Applies to MYSQL, MSSQL and POSTGRES
SELECT app_id, app_name, category, app_version, last_updated
FROM apps
WHERE last_updated < (SELECT last_updated FROM apps WHERE app_id = 29)
```

58. **Write a query to display the *app_name*, *category*, *app_version*, and *rating*, for apps with rating higher than the average**

```
-- Applies to MYSQL, MSSQL and POSTGRES
SELECT app_name, category, app_version, rating
FROM apps
WHERE rating > (SELECT AVG(rating) FROM apps)
```

59. **Write a query to display the categories having apps in the same *size_in_mb* as apps in the education category**

```
-- Applies to MYSQL, MSSQL and POSTGRES
SELECT DISTINCT category
FROM apps
WHERE size_in_mb IN (SELECT size_in_mb FROM apps WHERE category = 'Education')
AND category != 'Education'
```

60. **Minimum and Maximum Ratings**

    a. What is the min *rating* of apps in the Education *category* ? (subqueries are not needed to answer this one)

    b. What is the max *rating* of apps in the Education *category* ? (subqueries are not needed to answer this one)

    c. Write a query to display the *app_name* and *rating* for apps with *rating* in the range of Education min and max values

```
-- a. What is the min rating of apps in the Education category ? (subqueries are not needed to
answer this one)
-- Applies to MYSQL, MSSQL and POSTGRES
SELECT MIN(rating) FROM apps WHERE category = 'Education'

-- b. What is the max rating of apps in the Education category ? (subqueries are not needed to
answer this one)
```

```
-- Applies to MYSQL, MSSQL and POSTGRES
SELECT MAX(rating) FROM apps WHERE category = 'Education'

-- c. Write a query to display the app_name and rating for apps with rating in the range of
Education min and max values
-- Applies to MYSQL, MSSQL and POSTGRES
SELECT app_name, rating
FROM apps
WHERE rating >= (SELECT MIN(rating) FROM apps WHERE category = 'Education')
AND   rating <= (SELECT MAX(rating) FROM apps WHERE category = 'Education')
-- or
-- Applies to MYSQL, MSSQL and POSTGRES
SELECT app_name, rating
FROM apps
WHERE rating BETWEEN (SELECT MIN(rating) FROM apps WHERE category = 'Education')
             AND     (SELECT MAX(rating) FROM apps WHERE category = 'Education')
```

61. Write a query to display the *app_id*, *app_name*, *rating* and *reviews* for app with rating higher than *app_id* 131 and (number of) reviews higher than *app_id* 28

-- Applies to MYSQL, MSSQL and POSTGRES

SELECT app_id, app_name, rating and reviews FROM apps

WHERE rating > (SELECT rating FROM apps WHERE app_id = 131)

AND   reviews > (SELECT reviews FROM apps WHERE app_id = 28)