Product Queries

queries_products.sql

```sql
-- Add a product to the table with the name of "chair",
--  price of 44.00, and can_be_returned of false.

INSERT INTO products
  (name, price, can_be_returned)
VALUES
  ('chair', 44.00, 'f');

-- Add a product to the table with the name of "stool",
--  price of 25.99, and can_be_returned of true.

INSERT INTO products
  (name, price, can_be_returned)
VALUES
  ('stool', 25.99, 't');

-- Add a product to the table with the name of "table", price of
124.00,
--  and can_be_returned of false.

INSERT INTO products
  (name, price, can_be_returned)
VALUES
  ('table', 124.00, 'f');

-- Display all of the rows and columns in the table.

SELECT * FROM products;

-- Display all of the names of the products.

SELECT name FROM products;

-- Display all of the names and prices of the products.

SELECT name, price FROM products;

-- Add a new product - make up whatever you would like!

INSERT INTO products
```

```
  (name, price, can_be_returned)
VALUES
  ('hammock', 99.00, 't');

-- Display only the products that `can_be_returned`.

SELECT * FROM products WHERE can_be_returned;

-- Display only the products that have a price less than 44.00.

SELECT * FROM products WHERE price < 44.00;

-- Display only the products that have a price in between 22.50 and
99.99.

SELECT * FROM products WHERE price BETWEEN 22.50 AND 99.99;

-- There's a sale going on: Everything is $20 off! Update the database
accordingly.

UPDATE products SET price = price - 20;

-- Because of the sale, everything that costs less than $25 has sold
out.
  -- Remove all products whose price meets this criteria.

DELETE FROM products WHERE price < 25;

-- And now the sale is over. For the remaining products, increase
their price by $20.

UPDATE products SET price = price + 20;

-- There's been a change in company policy, and now all products are
returnable

UPDATE products SET can_be_returned = 't';
```

Playstore Queries

queries_playstore.sql

```sql
-- Query 0
SELECT * FROM analytics;

-- 1. Find the entire record for the app with an ID of `1880`.
SELECT * FROM analytics
  WHERE id = 1880;

-- 2. Find the ID and app name for all apps that were last updated on
August 01, 2018.
SELECT id, app_name FROM analytics
   WHERE last_updated = '2018-08-01';

-- 3. Count the number of apps in each category, e.g. "Family | 1972".
SELECT category, COUNT(*) FROM analytics
  GROUP BY category;

-- 4. Find the top 5 most-reviewed apps and the number of reviews for
each.
SELECT * FROM analytics
  ORDER BY reviews DESC
  LIMIT 5;

-- 5. Find the full record of the app that has the most reviews
--     with a rating greater than equal to 4.8.
SELECT * FROM analytics
  WHERE rating >= 4.8
  ORDER BY reviews DESC
  LIMIT 1;

-- 6. Find the average rating for each category ordered
--      by the highest rated to lowest rated.
SELECT category, AVG(rating) FROM analytics
  GROUP BY category
  ORDER BY avg DESC;

-- 7. Find the name, price, and rating of the most
--     expensive app with a rating that's less than 3
SELECT app_name, price, rating FROM analytics
  WHERE rating < 3
  ORDER BY price DESC
  LIMIT 1;

-- 8. Find all records with a min install not exceeding 50, that have
a rating.
--      Order your results by highest rated first.
```

```sql
SELECT * FROM analytics
  WHERE min_installs <= 50
    AND rating IS NOT NULL
  ORDER BY rating DESC;

-- 9. Find the names of all apps that are rated less than 3 with at
least 10000 reviews.
SELECT app_name FROM analytics
  WHERE rating < 3 AND reviews >= 10000;

-- 10. Find the top 10 most-reviewed apps that cost between 10 cents
and a dollar.
SELECT * FROM analytics
  WHERE price BETWEEN 0.1 and 1
  ORDER BY reviews DESC
  LIMIT 10;

-- 11. Find the most out of date app.
--  Hint: You don't need to do it this way, but it's possible to do
with a subquery:
--  http://www.postgresqltutorial.com/postgresql-max-function/
-- Option 1: with a subquery
SELECT * FROM analytics
  WHERE last_updated = (SELECT MIN(last_updated) FROM analytics);

-- Option 2: without a subquery
SELECT * FROM analytics
  ORDER BY last_updated LIMIT 1;

-- 12. Find the most expensive app (the query is very similar to #11).
-- Option 1: with a subquery
SELECT * FROM analytics
  WHERE price = (SELECT MAX(price) FROM analytics);

-- Option 2: without a subquery
SELECT * FROM analytics
  ORDER BY price DESC LIMIT 1;

-- 13. Count all the reviews in the Google Play Store.
SELECT SUM(reviews) AS "All the Reviews" FROM analytics;

-- 14. Find all the categories that have more than 300 apps in them.
SELECT category FROM analytics
  GROUP BY category
  HAVING COUNT(*) > 300;
```

```sql
-- 15. Find the app that has the highest proportion of reviews to
min_installs,
-- among apps that have been installed at least 100,000 times. Display
the name of the app
-- along with the number of reviews, the min_installs, and the
proportion.
SELECT app_name, reviews, min_installs,  min_installs / reviews AS
proportion
  FROM analytics
  WHERE min_installs >= 100000
  ORDER BY proportion DESC
  LIMIT 1;
```

Further Study

further_study.sql

```sql
-- FURTHER STUDY

-- FS1. Find the name and rating of the top rated apps in each
category,
-- among apps that have been installed at least 50,000 times.
SELECT app_name, rating, category FROM analytics
  WHERE (rating, category) in (
    SELECT MAX(rating), category FROM analytics
      WHERE min_installs >= 50000
      GROUP BY category
    )
  ORDER BY category;

-- FS2. Find all the apps that have a name similar to "facebook".
SELECT * FROM analytics
  WHERE app_name ILIKE '%facebook%';

-- FS3. Find all the apps that have more than 1 genre.
SELECT * FROM analytics
  WHERE  array_length(genres, 1) = 2;

-- FS4. Find all the apps that have education as one of their genres.
SELECT * FROM analytics

   WHERE genres @> '{"Education"}';
```