SQL_HomeWork

February 27, 2018

1 SQL Homework

```
In [1]: from sqlalchemy import create_engine
        import pandas as pd
        from warnings import filterwarnings
        import pymysql
        filterwarnings('ignore', category=pymysql.Warning)
        import os
        engine = create_engine('mysql+pymysql://root:kcmo1728@localhost/sakila')
```

1a. Display the first and last names of all actors from the table actor.

```
In [2]: actor = pd.read_sql_query('select first_name, last_name from actor', engine)
        actor.head()
```

```
Out[2]:   first_name     last_name
        0   PENELOPE       GUINESS
        1       NICK      WAHLBERG
        2         ED         CHASE
        3   JENNIFER         DAVIS
        4     JOHNNY  LOLLOBRIGIDA
```

1b. Display the first and last name of each actor in a single column in upper case letters. Name the column Actor Name.

```
In [5]: actor = pd.read_sql_query('select concat(ucase(first_name)," ", ucase(last_name)) as "A
        actor.head()
```

```
Out[5]:             Actor Name
        0     PENELOPE GUINESS
        1        NICK WAHLBERG
        2             ED CHASE
        3       JENNIFER DAVIS
        4  JOHNNY LOLLOBRIGIDA
```

2a. You need to find the ID number, first name, and last name of an actor, of whom you know only the first name, "Joe." What is one query would you use to obtain this information?

```
In [2]: actor = pd.read_sql_query('select actor_id, first_name, last_name from actor where firs
        actor.head()

Out[2]:    actor_id first_name last_name
        0         9        JOE     SWANK
```

2b. Find all actors whose last name contain the letters GEN:

```
In [19]: actor = pd.read_sql_query('select * from actor where last_name like "%%GEN%%"', engine
         actor.head()

Out[19]:    actor_id first_name  last_name         last_update
         0        14     VIVIEN     BERGEN 2006-02-15 04:34:33
         1        41      JODIE  DEGENERES 2006-02-15 04:34:33
         2       107       GINA  DEGENERES 2006-02-15 04:34:33
         3       166       NICK  DEGENERES 2006-02-15 04:34:33
```

2c. Find all actors whose last names contain the letters LI. This time, order the rows by last name and first name, in that order:

```
In [21]: actor = pd.read_sql_query('select last_name, first_name from actor \
                                    where last_name like "%%LI%%" \
                                    order by last_name, first_name', engine)
         actor.head()

Out[21]:   last_name first_name
         0   CHAPLIN       GREG
         1     JOLIE      WOODY
         2   OLIVIER     AUDREY
         3   OLIVIER       CUBA
         4  WILLIAMS    GROUCHO
```

2d. Using IN, display the country_id and country columns of the following countries: Afghanistan, Bangladesh, and China:

```
In [29]: actor = pd.read_sql_query('select * from country where country in ("Afghanistan", "Ba
         actor.head()

Out[29]:    country_id      country         last_update
         0           1  Afghanistan 2006-02-15 04:44:00
         1          12   Bangladesh 2006-02-15 04:44:00
         2          23        China 2006-02-15 04:44:00
```

3a. Add a middle_name column to the table actor. Position it between first_name and last_name. Hint: you will need to specify the data type.

```
In [3]: def RunSQL(sql_command):
            connection = pymysql.connect(host='localhost',
                                         user='root',
                                         password='kcmo1728',
```

```
                                db='sakila',
                                charset='utf8mb4',
                                cursorclass=pymysql.cursors.DictCursor)
        try:
            with connection.cursor() as cursor:
                commands = sql_command.split(';')
                for command in commands:
                    if command == '\n': continue
                    cursor.execute(command + ';')
                    connection.commit()
        except Exception as e:
            print(e)
        finally:
            connection.close()
```

```
In [35]: sql_query = """
          alter table actor
          add column middle_name varchar(30) after first_name;
         """
         RunSQL(sql_query)
```

3b. You realize that some of these actors have tremendously long last names. Change the data type of the middle_name column to blobs.

```
In [36]: sql_query = """
          alter table actor
          modify middle_name blob;
         """
         RunSQL(sql_query)
```

3c. Now delete the middle_name column.

```
In [38]: sql_query = """
          alter table actor
          drop middle_name;
         """
         RunSQL(sql_query)
```

4a. List the last names of actors, as well as how many actors have that last name.

```
In [42]: sql_query = """
         select last_name, count(last_name) AS 'Number of Actors'
         from actor
         group BY last_name;
         """
         RunSQL(sql_query)
         actor = pd.read_sql_query(sql_query, engine)
         actor.head()
```

```
Out[42]:   last_name  Number of Actors
         0    AKROYD                 3
         1     ALLEN                 3
         2   ASTAIRE                 1
         3    BACALL                 1
         4    BAILEY                 2
```

4b. List last names of actors and the number of actors who have that last name, but only for names that are shared by at least two actors

```
In [43]: sql_query = """
         select last_name, count(last_name) AS 'Number of Actors'
         from actor
         group by last_name
         having count(last_name) > 1;
         """
         RunSQL(sql_query)
         actor = pd.read_sql_query(sql_query, engine)
         actor.head()
```

```
Out[43]:   last_name  Number of Actors
         0    AKROYD                 3
         1     ALLEN                 3
         2    BAILEY                 2
         3    BENING                 2
         4     BERRY                 3
```

4c. Oh, no! The actor HARPO WILLIAMS was accidentally entered in the actor table as GROUCHO WILLIAMS, the name of Harpo's second cousin's husband's yoga teacher. Write a query to fix the record.

```
In [17]: sql_query = """
         update actor
         set first_name = 'HARPO'
         where first_name like '%GROUCHO%' and last_name = 'WILLIAMS';
         """
         RunSQL(sql_query)
```

```
In [18]: sql_query = """
         Select *
         From actor
         where first_name = 'HARPO' and last_name = 'WILLIAMS';
         """
         RunSQL(sql_query)
         actor = pd.read_sql_query(sql_query, engine)
         actor.head()
```

```
Out[18]:    actor_id first_name last_name         last_update
         0       172      HARPO  WILLIAMS 2018-02-27 06:03:15
```

4d. Perhaps we were too hasty in changing GROUCHO to HARPO. It turns out that GROUCHO was the correct name after all! In a single query, if the first name of the actor is currently HARPO, change it to GROUCHO. Otherwise, change the first name to MUCHO GROUCHO, as that is exactly what the actor will be with the grievous error. BE CAREFUL NOT TO CHANGE THE FIRST NAME OF EVERY ACTOR TO MUCHO GROUCHO, HOWEVER! (Hint: update the record using a unique identifier.)

```
In [19]: sql_query = """
         update actor
         set first_name =
             case
                 when first_name = "HARPO"
                     then "GROUCHO"
                 else "MUCHO GROUCHO"
             end
         where actor_id = 172;
         """
         RunSQL(sql_query)
```

```
In [20]: sql_query = """
         select * from actor
         where actor_id = 172;
         """
         RunSQL(sql_query)
         actor = pd.read_sql_query(sql_query, engine)
         actor.head()
```

```
Out[20]:    actor_id first_name last_name         last_update
         0       172    GROUCHO  WILLIAMS 2018-02-27 06:03:52
```

5a. You cannot locate the schema of the address table. Which query would you use to re-create it?

Hint: https://dev.mysql.com/doc/refman/5.7/en/show-create-table.html

```
In [27]: sql_query = """
         SHOW COLUMNS from sakila.address;
         """
         RunSQL(sql_query)
         actor = pd.read_sql_query(sql_query, engine)
         actor.head()
```

```
Out[27]:         Field                   Type Null  Key Default           Extra
         0  address_id  smallint(5) unsigned   NO  PRI    None  auto_increment
         1     address           varchar(50)   NO         None
         2    address2           varchar(50)  YES         None
         3    district           varchar(20)   NO         None
         4     city_id  smallint(5) unsigned   NO  MUL    None
```

```
In [ ]: sql_query = """
        SHOW COLUMNS from sakila.address;
```

```
SHOW CREATE TABLE sakila.address;

CREATE TABLE `address` (
  `address_id` smallint(5) unsigned NOT NULL AUTO_INCREMENT,
  `address` varchar(50) NOT NULL,
  `address2` varchar(50) DEFAULT NULL,
  `district` varchar(20) NOT NULL,
  `city_id` smallint(5) unsigned NOT NULL,
  `postal_code` varchar(10) DEFAULT NULL,
  `phone` varchar(20) NOT NULL,
  `location` geometry NOT NULL,
  `last_update` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAI
  PRIMARY KEY (`address_id`),
  KEY `idx_fk_city_id` (`city_id`),
  SPATIAL KEY `idx_location` (`location`),
  CONSTRAINT `fk_address_city` FOREIGN KEY (`city_id`) REFERENCES `city` (`city_id`) O
) ENGINE=InnoDB AUTO_INCREMENT=606 DEFAULT CHARSET=utf8

"""
RunSQL(sql_query)
actor = pd.read_sql_query(sql_query, engine)
actor.head()
```

6a. Use JOIN to display the first and last names, as well as the address, of each staff member. Use the tables staff and address:

```
In [28]: sql_query = """
         SELECT first_name, last_name, address from staff s
         INNER JOIN address a ON s.address_id = a.address_id;
         """
         RunSQL(sql_query)
         actor = pd.read_sql_query(sql_query, engine)
         actor.head()

Out[28]:   first_name last_name               address
         0       Mike   Hillyer     23 Workhaven Lane
         1        Jon  Stephens  1411 Lillydale Drive
```

6b. Use JOIN to display the total amount rung up by each staff member in August of 2005. Use tables staff and payment.

```
In [29]: sql_query = """
         SELECT s.staff_id, first_name, last_name, SUM(amount) as "Total Amount Rung Up"
         FROM staff s
         INNER JOIN payment p
         ON s.staff_id = p.staff_id
         GROUP BY s.staff_id;
```

```
"""
RunSQL(sql_query)
actor = pd.read_sql_query(sql_query, engine)
actor.head()
```

Out[29]:

	staff_id	first_name	last_name	Total Amount	Rung Up
0	1	Mike	Hillyer		33489.47
1	2	Jon	Stephens		33927.04

6c. List each film and the number of actors who are listed for that film. Use tables film_actor and film. Use inner join.

```
In [30]: sql_query = """
         Select f.title, COUNT(fa.actor_id) as "Number of Actors"
         FROM film f
         LEFT JOIN film_actor fa
         ON f.film_id = fa.film_id
         GROUP BY f.film_id;
         """
         RunSQL(sql_query)
         actor = pd.read_sql_query(sql_query, engine)
         actor.head()
```

Out[30]:

	title	Number of Actors
0	ACADEMY DINOSAUR	10
1	ACE GOLDFINGER	4
2	ADAPTATION HOLES	5
3	AFFAIR PREJUDICE	5
4	AFRICAN EGG	5

6d. How many copies of the film Hunchback Impossible exist in the inventory system?

```
In [31]: sql_query = """
         SELECT f.title, COUNT(i.inventory_id) as "Number in Inventory"
         FROM film f
         INNER JOIN inventory i
         ON f.film_id = i.film_id
         GROUP BY f.film_id
         HAVING title = "Hunchback Impossible";
         """
         RunSQL(sql_query)
         actor = pd.read_sql_query(sql_query, engine)
         actor.head()
```

Out[31]:

	title	Number in Inventory
0	HUNCHBACK IMPOSSIBLE	6

6e. Using the tables payment and customer and the JOIN command, list the total paid by each customer. List the customers alphabetically by last name:

```
In [32]: sql_query = """
         SELECT c.last_name, c.first_name, SUM(p.amount) as "Total Paid"
         FROM customer c
         INNER JOIN payment p
         ON c.customer_id = p.customer_id
         GROUP BY p.customer_id
         ORDER BY last_name, first_name;
         """
         RunSQL(sql_query)
         actor = pd.read_sql_query(sql_query, engine)
         actor.head()
```

Out[32]:

	last_name	first_name	Total Paid
0	ABNEY	RAFAEL	97.79
1	ADAM	NATHANIEL	133.72
2	ADAMS	KATHLEEN	92.73
3	ALEXANDER	DIANA	105.73
4	ALLARD	GORDON	160.68

7a. The music of Queen and Kris Kristofferson have seen an unlikely resurgence. As an unintended consequence, films starting with the letters K and Q have also soared in popularity. Use subqueries to display the titles of movies starting with the letters K and Q whose language is English.

```
In [37]: sql_query = """
         SELECT title FROM film
         WHERE language_id IN
             (SELECT language_id FROM language
             WHERE name = "English")
             AND (title LIKE "K%%") OR (title LIKE "Q%%");
         """
         RunSQL(sql_query)
         actor = pd.read_sql_query(sql_query, engine)
         actor.head()
```

Out[37]:

	title
0	KANE EXORCIST
1	KARATE MOON
2	KENTUCKIAN GIANT
3	KICK SAVANNAH
4	KILL BROTHERHOOD

7b. Use subqueries to display all actors who appear in the film Alone Trip.

```
In [38]: sql_query = """
         SELECT first_name, last_name FROM actor
         WHERE actor_id IN
             (SELECT actor_id FROM film_actor
             WHERE film_id IN
```

```
        (SELECT film_id FROM film
        WHERE title = "Alone Trip"));
"""
RunSQL(sql_query)
actor = pd.read_sql_query(sql_query, engine)
actor.head()
```

Out[38]:

	first_name	last_name
0	ED	CHASE
1	KARL	BERRY
2	UMA	WOOD
3	WOODY	JOLIE
4	SPENCER	DEPP

7c. You want to run an email marketing campaign in Canada, for which you will need the names and email addresses of all Canadian customers. Use joins to retrieve this information.

In [39]:
```
sql_query = """
SELECT c.first_name, c.last_name, c.email, co.country FROM customer c
LEFT JOIN address a
ON c.address_id = a.address_id
LEFT JOIN city ci
ON ci.city_id = a.city_id
LEFT JOIN country co
ON co.country_id = ci.country_id
WHERE country = "Canada";
"""
RunSQL(sql_query)
actor = pd.read_sql_query(sql_query, engine)
actor.head()
```

Out[39]:

	first_name	last_name	email	country
0	DERRICK	BOURQUE	DERRICK.BOURQUE@sakilacustomer.org	Canada
1	DARRELL	POWER	DARRELL.POWER@sakilacustomer.org	Canada
2	LORETTA	CARPENTER	LORETTA.CARPENTER@sakilacustomer.org	Canada
3	CURTIS	IRBY	CURTIS.IRBY@sakilacustomer.org	Canada
4	TROY	QUIGLEY	TROY.QUIGLEY@sakilacustomer.org	Canada

7d. Sales have been lagging among young families, and you wish to target all family movies for a promotion. Identify all movies categorized as famiy films.

In [40]:
```
sql_query = """
SELECT * from film
WHERE film_id IN
    (SELECT film_id FROM film_category
    WHERE category_id IN
        (SELECT category_id FROM category
        WHERE name = "Family"));
"""
```

```
RunSQL(sql_query)
actor = pd.read_sql_query(sql_query, engine)
actor.head()
```

Out[40]:

	film_id	title	description	release_year	language_id	original_language_id	rental_duration	rental_rate	length	replacement_cost	rating	special_features	last_update
0	5	AFRICAN EGG	A Fast-Paced Documentary of a Pastry Chef And ...	2006	1	None	6	2.99	130	22.99	G	Deleted Scenes	2006-02-15 05:03:42
1	31	APACHE DIVINE	A Awe-Inspiring Reflection of a Pastry Chef An...	2006	1	None	5	4.99	92	16.99	NC-17	Commentaries,Deleted Scenes,Behind the Scenes	2006-02-15 05:03:42
2	43	ATLANTIS CAUSE	A Thrilling Yarn of a Feminist And a Hunter wh...	2006	1	None	6	2.99	170	15.99	G	Behind the Scenes	2006-02-15 05:03:42
3	50	BAKED CLEOPATRA	A Stunning Drama of a Forensic Psychologist An...	2006	1	None	3	2.99	182	20.99	G	Commentaries,Behind the Scenes	2006-02-15 05:03:42
4	53	BANG KWAI	A Epic Drama of a Madman And a Cat who must Fa...	2006	1	None	5	2.99	87	25.99	NC-17	Commentaries,Deleted Scenes,Behind the Scenes	2006-02-15 05:03:42

7e. Display the most frequently rented movies in descending order.

```
In [41]: sql_query = """
         SELECT f.title , COUNT(r.rental_id) AS "Number of Rentals" FROM film f
         RIGHT JOIN inventory i
         ON f.film_id = i.film_id
         JOIN rental r
         ON r.inventory_id = i.inventory_id
         GROUP BY f.title
```

```
ORDER BY COUNT(r.rental_id) DESC;
"""
RunSQL(sql_query)
actor = pd.read_sql_query(sql_query, engine)
actor.head()
```

Out[41]:

	title	Number of Rentals
0	BUCKET BROTHERHOOD	34
1	ROCKETEER MOTHER	33
2	RIDGEMONT SUBMARINE	32
3	JUGGLER HARDLY	32
4	GRIT CLOCKWORK	32

7f. Write a query to display how much business, in dollars, each store brought in.

```
In [42]: sql_query = """
         SELECT s.store_id, sum(amount) as "Revenue" FROM store s
         RIGHT JOIN staff st
         ON s.store_id = st.store_id
         LEFT JOIN payment p
         ON st.staff_id = p.staff_id
         GROUP BY s.store_id;
         """
         RunSQL(sql_query)
         actor = pd.read_sql_query(sql_query, engine)
         actor.head()
```

Out[42]:

	store_id	Revenue
0	1	33489.47
1	2	33927.04

7g. Write a query to display for each store its store ID, city, and country.

```
In [43]: sql_query = """
         SELECT s.store_id, ci.city, co.country FROM store s
         JOIN address a
         ON s.address_id = a.address_id
         JOIN city ci
         ON a.city_id = ci.city_id
         JOIN country co
         ON ci.country_id = co.country_id;
         """
         RunSQL(sql_query)
         actor = pd.read_sql_query(sql_query, engine)
         actor.head()
```

Out[43]:

	store_id	city	country
0	1	Lethbridge	Canada
1	2	Woodridge	Australia

7h. List the top five genres in gross revenue in descending order. (Hint: you may need to use the following tables: category, film_category, inventory, payment, and rental.)

```
In [44]: sql_query = """
         SELECT c.name, sum(p.amount) as "Revenue per Category" FROM category c
         JOIN film_category fc
         ON c.category_id = fc.category_id
         JOIN inventory i
         ON fc.film_id = i.film_id
         JOIN rental r
         ON r.inventory_id = i.inventory_id
         JOIN payment p
         ON p.rental_id = r.rental_id
         GROUP BY name;
         """
         RunSQL(sql_query)
         actor = pd.read_sql_query(sql_query, engine)
         actor.head()
```

```
Out[44]:          name  Revenue per Category
         0      Action                4375.85
         1   Animation                4656.30
         2    Children                3655.55
         3    Classics                3639.59
         4      Comedy                4383.58
```

8a. In your new role as an executive, you would like to have an easy way of viewing the Top five genres by gross revenue. Use the solution from the problem above to create a view. If you haven't solved 7h, you can substitute another query to create a view.

```
In [58]: sql_query = """
         CREATE VIEW top_5_by_genre AS
         SELECT c.name, sum(p.amount) as "Revenue per Category" FROM category c
         JOIN film_category fc
         ON c.category_id = fc.category_id
         JOIN inventory i
         ON fc.film_id = i.film_id
         JOIN rental r
         ON r.inventory_id = i.inventory_id
         JOIN payment p
         ON p.rental_id = r.rental_id
         GROUP BY name
         ORDER BY SUM(p.amount) DESC
         LIMIT 5;
         """
         RunSQL(sql_query)
```

8b. How would you display the view that you created in 8a?

```
In [59]: sql_query = """
         SELECT * FROM top_5_by_genre;
         """
         RunSQL(sql_query)
         actor = pd.read_sql_query(sql_query, engine)
         actor.head()
```

```
Out[59]:         name  Revenue per Category
         0     Sports                5314.21
         1     Sci-Fi                4756.98
         2  Animation                4656.30
         3      Drama                4587.39
         4     Comedy                4383.58
```

8c. You find that you no longer need the view top_five_genres. Write a query to delete it.

```
In [60]: sql_query = """
         DROP VIEW top_5_by_genre;
         """
         RunSQL(sql_query)
```