Patricia Hurlocker 8/30/2018

SQL RESULTS SAKILA DB

## Uploading Homework

\* To submit this homework using BootCampSpot:

\* Create a GitHub repository.

\* Upload your .sql file with the completed queries.

\* Submit a link to your GitHub repo through BootCampSpot.

|  |  |  |
| --- | --- | --- |
| 1a. | **Display the first and last names of all actors from the table `actor`.**  SQL  use sakila;  select actor.first\_name, actor.last\_name from actor  order by actor.last\_name; |  |
| 1b. | **Display the first and last name of each actor in a single column in upper case letters. Name the column `Actor Name`.**  SQL  select concat(actor.first\_name, ' ', actor.last\_name) as "ACTOR NAME"  from actor  order by actor.last\_name, actor.first\_name; |  |
| 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?**  SQL  select actor.actor\_id, actor.first\_name, actor.last\_name  from actor  where actor.first\_name like "JO%"  order by actor.last\_name, actor.first\_name; |  |
| 2b. | **Find all actors whose last name contain the letters `GEN`:**  SQL  select actor.first\_name, actor.last\_name  from actor  where actor.last\_name like "%GEN%"  order by actor.last\_name, actor.first\_name; |  |
| 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:**  SQL  select actor.first\_name, actor.last\_name  from actor  where actor.last\_name like "%LI%"  order by actor.last\_name, actor.first\_name; |  |
| 2d. | **Using `IN`, display the `country\_id` and `country` columns of the following countries: Afghanistan, Bangladesh, and China:**  SQL  select \* from country;  select country\_id, country  from country  WHERE country IN ("Afghanistan", "Bangladesh", "China"); |  |
| 3a. | **You want to keep a description of each actor. You don't think you will be performing queries on a description, so create a column in the table `actor` named `description` and use the data type `BLOB` (Make sure to research the type `BLOB`, as the difference between it and `VARCHAR` are significant).**  SQL – create new column as BLOB  select \* from actor;  ALTER TABLE actor  ADD COLUMN description blob;  select \* from actor; |  |
| 3b. | **Very quickly you realize that entering descriptions for each actor is too much effort. Delete the `description` column**.  SQL  ALTER TABLE actor  DROP COLUMN description; |  |
| 4a. | **List the last names of actors, as well as how many actors have that last name.**  **SQL**  select last\_name, COUNT(\*)  FROM actor  GROUP BY last\_name;  **Better SQL**  select last\_name, COUNT(\*)  FROM actor  GROUP BY last\_name  ORDER BY COUNT(\*) desc; |  |
| 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**.  SQL  SELECT last\_name, COUNT(\*)  FROM actor  GROUP BY last\_name  HAVING COUNT(\*) >2  ORDER BY COUNT(\*) desc; |  |
| 4c. | **The actor `HARPO WILLIAMS` was accidentally entered in the `actor` table as `GROUCHO WILLIAMS`. Write a query to fix the record.**  **First VERIFY ONLY ONE RECORD EXISTS:**  select \* from actor  where actor.first\_name ="GROUCHO" and actor.last\_name = "WILLIAMS";  Export the sakila.actor table using the WIZARD to save before update  **SQL to UPDATE**  UPDATE actor SET first\_name='HARPO'  WHERE actor\_id=172;  commit; |  |
| 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`.**  Well – I can restore the table from my export or I can perform an update again.  SELECT \* from actor  WHERE actor.first\_name in (“GROUCHO”, “HARPO”)  UPDATE actor  SET first\_name=”GROUCHO”  WHERE actor\_id=172;  COMMIT; |  |
| 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>  mysqldump sakila.address > dump.sql  **Restoring Selected Tables from a TTS Image Backup**  mysqlbackup --socket=/tmp/restoreserver.sock --datadir=/logs/restoreserverdata --backup-dir=/logs/backup \  --include-tables="^sales\." --exclude-tables="^sales\.hardware$" copy-back-and-apply-log  **Restoring and Renaming a Table from a TTS Backup**  # Using fully qualified table names:  mysqlbackup --socket=/tmp/restoreserver.sock --datadir=/logs/restoreserverdata \  --backup-dir=/BackupDirTemp --backup-image=/home/dbadmin/backups/tts-backup.mbi \  --include-tables="^sales\.cars" --rename="sales.cars to sales.autos" copy-back-and-apply-log  # It works the same if database names are omitted in the argument for --rename:  mysqlbackup --socket=/tmp/restoreserver.sock --datadir=/logs/restoreserverdata \  --backup-dir=/BackupDirTemp --backup-image=/home/dbadmin/backups/tts-backup.mbi \  --include-tables="^sales\.cars" --rename="cars to autos" copy-back-and-apply-log | The table ADDRESS should have been created using a script.  To recreate the table view the existing table and determine the PK and FK. Also determine the required fields, size, and the datatype. Check for TRIGGERS to update any other table, and any indicies.  **Table: address**   **Columns:**   |  |  | | --- | --- | | **address\_id** | smallint(5) UN AI PK | | address | varchar(50) | | address2 | varchar(50) | | district | varchar(20) | | **city\_id** | smallint(5) UN | | postal\_code | varchar(10) | | phone | varchar(20) | | **location** | geometry | | last\_update | timestamp |   **Related Tables:**   |  |  | | --- | --- | | Target | city (city\_id → city\_id) | | On Update | CASCADE | | On Delete | RESTRICT |   I’ve created a new table as ADDRESS\_ALT  CREATE TABLE address\_ALT (  address\_id INTEGER(5) AUTO\_INCREMENT NOT NULL, PRIMARY KEY (address\_id),  address VARCHAR(50) NOT NULL,  address2 VARCHAR(50),  district VARCHAR(20)  city\_id INTEGER(5) NOT NULL,  postal\_code VARCHAR(5) NOT NULL,  phone VARCHAR(20),  location BLOB,  last\_update TIMESTAMP(),  PRIMARY KEY (address\_id))  FOREIGN KEY (city\_id); |
| 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`:**  **First check the STAFF table to determine how many STAFF members are in the table**.  SELECT \* FROM staff;  **Review the other tables required to get the full address for the Staff.**  SELECT  staff.staff\_id,  staff.first\_name,  staff.last\_name,  address.address,  address.address2,  city.city,  address.postal\_code,  address.district,  country.country,  address.city\_id,  country.country\_id  FROM  staff  LEFT JOIN address  ON staff.address\_id = address.address\_id  JOIN city  on address.city\_id = city.city\_id  LEFT JOIN country  on city.country\_id = country.country\_id; |  |
| 6b. | **Use `JOIN` to display the total amount rung up by each staff member in August of 2005. Use tables `staff` and `payment`.**  UNDERSTAND: ENPOINTS WITH THE BETWEEN. BETTER USE FOR ACCURACY AND QUERY SPEED.  SELECT  SUM(payment.amount) AS 'SALES',  staff.staff\_id,  staff.first\_name,  staff.last\_name,  staff.store\_id  FROM  payment  JOIN  staff ON staff.staff\_id = payment.staff\_id  WHERE  payment.payment\_date BETWEEN '2005-08-01' AND '2005-08-31'  GROUP BY payment.staff\_id;  IF YOU WANT TO USE > = and <=  SELECT  SUM(payment.amount) AS 'SALES',  staff.staff\_id,  staff.first\_name,  staff.last\_name,  staff.store\_id  FROM  payment  JOIN staff  ON staff.staff\_id = payment.staff\_id  WHERE  (payment.payment\_date >= '2005-08-01' AND payment.payment\_date <='2005-08-31')  GROUP BY payment.staff\_id; | Result if you use endpoints 2005-07-31 AN 2005-09-01 (note: ENPOINTS ARE INCLUDED SO THE VALUE IS HIGHER). |
| 6c. | List each film and the number of actors who are listed for that film. Use tables `film\_actor` and `film`. Use inner join.  SELECT T1.film\_id, T1.title, COUNT(T2.actor\_id) AS actor\_cnt  FROM film AS T1  JOIN film\_actor AS T2  ON T1.film\_id = T2.film\_id  GROUP BY T2.film\_id;  IMPROVE CODE to show the movies with the highest actor\_cnt descending.  SELECT T1.film\_id, T1.title, COUNT(T2.actor\_id) AS actor\_cnt  FROM film AS T1  JOIN film\_actor AS T2  ON T1.film\_id = T2.film\_id  GROUP BY T2.film\_id  ORDER BY actor\_cnt desc; |  |
| 6d. | **How many copies of the film `Hunchback Impossible` exist in the inventory system?**  SELECT T1.film\_id, T1.title, COUNT(T2.inventory\_id) as inventory\_cnt  FROM film as T1  JOIN inventory as T2  ON T1.film\_id = T2.film\_id and T1.title like 'Hunchb%'  GROUP BY T2.film\_id; |  |
| 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**:  SELECT T1.first\_name, T1.last\_name, SUM(T2.amount) AS customer\_payments\_total  FROM customer as T1  JOIN payment as T2  ON T1.customer\_id = T2.customer\_id  GROUP BY T1.last\_name; |  |
| 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.**  Select film.film\_id, film.title, language.name  from film  join language on language.language\_id = film.language\_id  where ((film.title LIKE 'K%' or film.title LIKE 'K%') and language.name = 'English')  order by film.title; |  |
| 7b. | Use subqueries to display all actors who appear in the film `Alone Trip`  SELECT actor.first\_name, actor.last\_name  FROM actor  WHERE actor.actor\_id IN  (SELECT film\_actor.actor\_id FROM film\_actor  WHERE film\_actor.film\_id IN  (SELECT film.film\_id FROM film  WHERE film.title = "Alone Trip")); |  |
| 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.**  SELECT T1.first\_name, T1.last\_name, T1.email  FROM customer AS T1  JOIN address AS T2  ON T1.address\_id = T2.address\_id  JOIN city AS T3  ON T2.city\_id = T3.city\_id  JOIN country AS T4  ON T3.country\_id = T4.country\_id  WHERE T4.country = '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 family films.**  **Is a rating supposed to determine ‘Family’ such as like G, PG, or PG-13 a family rated film?**  SELECT T1.title  FROM film as T1  WHERE T1.film\_id IN  (SELECT T2.film\_id  FROM film\_category as T2  WHERE T2.category\_id IN  (SELECT T3.category\_id  FROM category as T3  WHERE T3.name = 'Family')); |  |
| 7e. | **Display the most frequently rented movies in descending order.**  SELECT  T1.title, COUNT(T3.rental\_id) AS movie\_rentals\_cnt  FROM  film AS T1  JOIN  inventory AS T2 ON T1.film\_id = T2.film\_id  JOIN  rental AS T3 ON T2.inventory\_id = T3.inventory\_id  GROUP BY T1.title  ORDER BY movie\_rentals\_cnt DESC; |  |
| 7f. | **Write a query to display how much business, in dollars, each store brought in.**  SELECT T1.store\_id, sum(T2.amount) as 'Total Sales per Store'  FROM payment T2  JOIN rental T3  ON (T3.rental\_id = T2.rental\_id)  JOIN inventory T4  ON (T4.inventory\_id = T3.inventory\_id)  JOIN store T1  ON (T1.store\_id = T4.store\_id)  GROUP BY T1.store\_id; |  |
| 7g. | **Write a query to display for each store its store ID, city, and country.**  select \* from store;  result is only 2 stores.  SELECT  store.store\_id,  address.address,  address.address2,  address.district,  address.city\_id,  address.postal\_code  FROM store  JOIN address on store.address\_id = address.address\_id; |  |
| 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.)  --T1 category  --T2 film\_category  --T3 inventory  --T4 rental  --T5 payment  SELECT  T1.name, SUM(T5.amount) AS grossrevenue  FROM  category AS T1  JOIN  film\_category AS T2 ON T1.category\_id = T2.category\_id  JOIN  inventory AS T3 ON T2.film\_id = T3.film\_id  JOIN  rental AS T4 ON T3.inventory\_id = T4.inventory\_id  JOIN  payment AS T5 ON T4.rental\_id = T5.rental\_id  GROUP BY T1.name  ORDER BY grossrevenue DESC  LIMIT 0 , 5; |  |
| 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.**  CREATE VIEW sales\_executive\_vw AS  SELECT  T1.name, SUM(T5.amount) AS grossrevenue  FROM  category AS T1  JOIN  film\_category AS T2 ON T1.category\_id = T2.category\_id  JOIN  inventory AS T3 ON T2.film\_id = T3.film\_id  JOIN  rental AS T4 ON T3.inventory\_id = T4.inventory\_id  JOIN  payment AS T5 ON T4.rental\_id = T5.rental\_id  GROUP BY T1.name  ORDER BY grossrevenue DESC  LIMIT 0 , 5; |  |
| 8b. | How would you display the view that you created in 8a?  SELECT \* FROM sakila.sales\_executive\_vw; |  |
| 8c. | You find that you no longer need the view `top\_five\_genres`. Write a query to delete it.  DROP VIEW sakila.sales\_executive\_vw; |  |