BIG DATA MANAGING AND PROCESSING

Module Code: B8IT155

Name: Seán Carroll

Student Number: 20024157

Table of Contents

Question 1 - Structured Data – SQL

Link to Google Colab File -

https://colab.research.google.com/drive/1gOIDAbV0n7ZLefyiaKv85bJZ5dV8jJe-?usp=sharing

1. *How many tables does this database contain?*

The code below shows there are 23 tables in the database.

```
# Show the tables in the database
!mysql -D "sakila" -e "show tables"
```

```
+----------------------------+
| Tables_in_sakila           |
+----------------------------+
| actor                      |
| actor_info                 |
| address                    |
| category                   |
| city                       |
| country                    |
| customer                   |
| customer_list              |
| film                       |
| film_actor                 |
| film_category              |
| film_list                  |
| film_text                  |
| inventory                  |
| language                   |
| nicer_but_slower_film_list |
| payment                    |
| rental                     |
| sales_by_film_category     |
| sales_by_store             |
| staff                      |
| staff_list                 |
| store                      |
+----------------------------+
```

```
# Show how many tables are in the database.
# Name the return 'No. of Tables'
# Count from information_schema.tables where table_schema = 'sakila'
!mysql -e "Select count(*) as 'No. of Tables' from information_schema.tables where table_schema = 'sakila';"
```

```
+---------------+
| No. of Tables |
+---------------+
|            23 |
+---------------+
```

2. *How many films are listed in this database?*

The code below shows there are 1000 films in the database

```
# Return just the 'No. of Films' using the number of entries in film_id column
!mysql -D "sakila" -e "Select count(film_id) as 'No. of Films' from film;"
```

```
+---------------+
| No. of Films |
+---------------+
|         1000 |
+---------------+
```

3. *Display the first and last name of each actor in a single column in upper case letters. Name the column "Actor Name".*

The code below displays the information.

```
# Return the first and last names from the actor table
# Use Concat to return the entries together in the one column
# Upper displays the entries in upper case
!mysql -D "sakila" -e "Select upper(concat(first_name, ' ', last_name)) as 'Actor Name' from actor;"
```

```
+----------------------+
| Actor Name           |
+----------------------+
| PENELOPE GUINESS     |
| NICK WAHLBERG        |
| ED CHASE             |
| JENNIFER DAVIS       |
| JOHNNY LOLLOBRIGIDA  |
| BETTE NICHOLSON      |
| GRACE MOSTEL         |
| MATTHEW JOHANSSON    |
| JOE SWANK            |
| CHRISTIAN GABLE      |
| ZERO CAGE            |
| KARL BERRY           |
| UMA WOOD             |
| VIVIEN BERGEN        |
| CUBA OLIVIER         |
| FRED COSTNER         |
| HELEN VOIGHT         |
| DAN TORN             |
| BOB FAWCETT          |
| LUCILLE TRACY        |
| KIRSTEN PALTROW      |
| ELVIS MARX           |
| SANDRA KILMER        |
| CAMERON STREEP       |
| KEVIN BLOOM          |
| RIP CRAWFORD         |
```

4. *Find the ID number, first name, and last name of an actor/actors with the first name "John."*

The code below displays the information.

```
# Use 'actor' table
# Display the features asked
# IN ('John') returns all actors with the first name John
!mysql -D "sakila" -e "Select actor_id, first_name, last_name from actor where first_name IN('John');"
```

```
+----------+------------+-----------+
| actor_id | first_name | last_name |
+----------+------------+-----------+
|      192 | JOHN       | SUVARI    |
+----------+------------+-----------+
```

5. *Find all actors whose last names contain the letters OO. This time, order the rows by last name and first name, in that order.*

The code below displays the information.

```
# Select last name and first name in that order from actor table
# Order the results by last name
!mysql -D "sakila" -e "Select last_name, first_name from actor where last_name like '%OO%' order by last_name asc;"
```

```
+-------------+------------+
| last_name   | first_name |
+-------------+------------+
| BLOOM       | KEVIN      |
| GOODING     | EWAN       |
| GOODING     | GREGORY    |
| WITHERSPOON | ANGELA     |
| WOOD        | UMA        |
| WOOD        | FAY        |
+-------------+------------+
```

6. *Display the "country_id" and country columns of the following countries: Poland, Angola, and Zambia.*

The code below displays the information.

```
# Selct the country_id and country columns from the country table
# Use IN() function to display Poland, Angola, and Zambia
!mysql -D "sakila" -e "Select country_id, country from country where country IN('Poland', 'Angola', 'Zambia')"
```

```
+------------+---------+
| country_id | country |
+------------+---------+
|          4 | Angola  |
|         76 | Poland  |
|        109 | Zambia  |
+------------+---------+
```

7. *Display the title of all films in Mandarin. How many films are in Mandarin in the database?*

The code below shows that there are 0 films in Mandarin.

```
# Display the number of films in Mandarin from film table
# Left join language table using the language_id column in both tables
# Use IN() function to display
!mysql -D "sakila" -e "Select count(*) as 'No. of Films in Mandarin' from film join language on film.language_id = language.language_id \
where language.name IN('Mandarin');"
```

```
+------------------------+
| No. of Films in Mandarin |
+------------------------+
|                      0 |
+------------------------+
```

8. *You cannot locate the schema of the address table. Which query would you use to recreate it?*

The code below displays how the address table could be recreated.

```
# Describe the address table to find the details of each column
!mysql -D "sakila" -e "Describe address"
```

```
+-------------+-------------------+------+-----+-------------------+-----------------------------------------------+
| Field       | Type              | Null | Key | Default           | Extra                                         |
+-------------+-------------------+------+-----+-------------------+-----------------------------------------------+
| address_id  | smallint unsigned | NO   | PRI | NULL              | auto_increment                                |
| address     | varchar(50)       | NO   |     | NULL              |                                               |
| address2    | varchar(50)       | YES  |     | NULL              |                                               |
| district    | varchar(20)       | NO   |     | NULL              |                                               |
| city_id     | smallint unsigned | NO   | MUL | NULL              |                                               |
| postal_code | varchar(10)       | YES  |     | NULL              |                                               |
| phone       | varchar(20)       | NO   |     | NULL              |                                               |
| location    | geometry          | NO   | MUL | NULL              |                                               |
| last_update | timestamp         | NO   |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED on update CURRENT_TIMESTAMP |
+-------------+-------------------+------+-----+-------------------+-----------------------------------------------+
```

```
# Create the table calling it address
# Use the information in the described table above to create the address table
!mysql -D "sakila" -e "create table address \
    (address_id smallint unsigned NOT NULL auto_increment, \
    address varchar(50) NOT NULL, \
    address2 varchar(50), \
    district varchar(20) NOT NULL, \
    city_id smallint unsigned NOT NULL, \
    postal_code varchar(10), \
    phone varchar(20) NOT NULL, \
    location geometry, \
    last_update timestamp DEFAULT CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP, \
    PRIMARY KEY (address_id), \
    CONSTRAINT fk_address_city FOREIGN KEY (city_id) REFERENCES city (city_id));"
```

9. *Use JOIN to display the first and last names, as well as the address, of each staff member. Use the tables staff and address.*

The code below displays the information.

```
# Display first name, last name, and address from the staff table
# Inner join the address table using the address_id column in each table
!mysql -D "sakila" -e "select first_name, last_name, address from staff inner join address on staff.address_id = address.address_id;"
```

```
+------------+-----------+----------------------+
| first_name | last_name | address              |
+------------+-----------+----------------------+
| Mike       | Hillyer   | 23 Workhaven Lane    |
| Jon        | Stephens  | 1411 Lillydale Drive |
+------------+-----------+----------------------+
```

10. Use subqueries to display all actors who appear in the film 'Thief Pelican'.

The code below displays the information.

```
# Display the actor_id, first name, and last name of each actor from the actor table
# Use the IN() function to display the subquery where the actor_id is displayed from the film_actor table
# and where the film_id equals the subquery that displays films with the Thief Pelican film_id
!mysql -D "sakila" -e "select actor_id, first_name, last_name from actor \
where actor_id IN (select actor_id from film_actor where film_id = (select film_id from film where title = 'Thief Pelican'));"
```

```
+----------+------------+-----------+
| actor_id | first_name | last_name |
+----------+------------+-----------+
|       21 | KIRSTEN    | PALTROW   |
|       40 | JOHNNY     | CAGE      |
|       84 | JAMES      | PITT      |
|      108 | WARREN     | NOLTE     |
|      115 | HARRISON   | BALE      |
|      117 | RENEE      | TRACY     |
|      150 | JAYNE      | NOLTE     |
+----------+------------+-----------+
```

11. Identify all movies categorised as family films.

The code below displays the information.

```
# Display the title column from the film_list table
# Use the IN() function to display the 'Family' categorised titles
# Rename the displayed information as 'Family Films'
!mysql -D "sakila" -e "select title as 'Family Films' from film_list where category IN ('Family');"
```

```
| CHASING FIGHT          |
| CHISUM BEHAVIOR        |
| CHOCOLAT HARRY         |
| CONFUSED CANDLES       |
| CONVERSATION DOWNHILL  |
| DATE SPEED             |
| DINOSAUR SECRETARY     |
| DUMBO LUST             |
| EARRING INSTINCT       |
| EFFECT GLADIATOR       |
| FEUD FROGMEN           |
| FINDING ANACONDA       |
| GABLES METROPOLIS      |
| GANDHI KWAI            |
| GLADIATOR WESTWARD     |
| GREASE YOUTH           |
| HALF OUTFIELD          |
| HOCUS FRIDA            |
| HOMICIDE PEACH         |
| HOUSE DYNAMITE         |
| HUNTING MUSKETEERS     |
| INDIAN LOVE            |
| JASON TRAP             |
| JEDI BENEATH           |
| KILLER INNOCENT        |
| KING EVOLUTION         |
| LOLITA WORLD           |
| LOUISIANA HARRY        |
| MAGUIRE APACHE         |
| MANCHURIAN CURTAIN     |
```

12. Display the 10 most frequently rented movies in descending order.

The code below displays the information.

```
# Select the film_id and title columns from the film table and the rental_id column as a count from the rental table
# Join the inventory column using the film_id column, and the rental column using the inventory_id column
# Rename the count of the rental_id as '10 Most Frequently Rented Movies'
# Group the films by film_id and title, order them by the count, and limit to 10 to display the top 10 films
!mysql -D "sakila" -e "select film.film_id, film.title, count(rental.rental_id) as '10 Most Frequently Rented Movies' from film \
join inventory on film.film_id = inventory.film_id \
join rental on inventory.inventory_id = rental.inventory_id \
group by film.film_id, film.title order by count(rental.rental_id) desc limit 10;"
```

```
+---------+--------------------+---------------------------------+
| film_id | title              | 10 Most Frequently Rented Movies |
+---------+--------------------+---------------------------------+
|     103 | BUCKET BROTHERHOOD |                              34 |
|     738 | ROCKETEER MOTHER   |                              33 |
|     730 | RIDGEMONT SUBMARINE |                              32 |
|     382 | GRIT CLOCKWORK     |                              32 |
|     767 | SCALAWAG DUCK      |                              32 |
|     489 | JUGGLER HARDLY     |                              32 |
|     331 | FORWARD TEMPLE     |                              32 |
|     418 | HOBBIT ALIEN       |                              31 |
|     735 | ROBBERS JOON       |                              31 |
|    1000 | ZORRO ARK          |                              31 |
+---------+--------------------+---------------------------------+
```

13. List the top five genres in gross revenue in descending order.

The code below displays the information.

```
# Display the name from the category table and rename it 'Genre'
# Display the amount column from the payment table and rename it 'Gross Revenue'
# Join the film table to the film_category table using the category_id foreign key
# Join the film_category table to the category table using the category_id foreign key
# Join the inventory table to the film table using the film_id foreign key
# Join the rental table to the inventory table using the inventory_id foreign key
# Join the payment table to the rental table using the rental_id foreign key
# Group by name in the category table
# Order by 'Gross Revenue'
# Limit to the top 5
!mysql -D "sakila" -e "select category.name as 'Genre', sum(payment.amount) as 'Gross Revenue' from film \
join film_category on film.film_id = film_category.film_id \
join category on film_category.category_id = category.category_id \
join inventory on film.film_id = inventory.film_id \
join rental on inventory.inventory_id = rental.inventory_id \
join payment on rental.rental_id = payment.rental_id \
group by category.name \
order by sum(payment.amount) desc limit 5;"
```

```
+-----------+---------------+
| Genre     | Gross Revenue |
+-----------+---------------+
| Sports    |       5314.21 |
| Sci-Fi    |       4756.98 |
| Animation |       4656.30 |
| Drama     |       4587.39 |
| Comedy    |       4383.58 |
+-----------+---------------+
```

Question 2 - Semi-Structured Data - NoSQL – MongoDB

Code used to create the Members Table

```
[
  {
    "First_Name": "Carol",
    "Last_Name": "Smith",
    "Age": 50,
    "Favourite_Sports": ["Tennis", "Badminton"],
    "Address": {
      "City": "Dublin",
      "Street": "Abbey Rd.",
      "Number": 1
    }
  },
  {
    "First_Name": "Richard",
    "Last_Name": "Miller",
    "Age": 30.5,
    "Favourite_Sports": ["Tennis"],
    "Address": {
      "City": "Cork",
      "Street": "Temple St.",
      "Number": " "
    }
  },
  {
    "First_Name": "Thomas",
    "Last_Name": "Garcia",
    "Age": 55,
    "Favourite_Sports": ["Football"],
    "Address": {
      "City": "Galway",
      "Street": "Henry St.",
      "Number": 3
    }
  },
  {
    "First_Name": "Eugenia",
    "Last_Name": "Jones",
    "Age": 22,
    "Favourite_Sports": [
      "Handball",
      "Basketball"
    ],
    "Address": {
```

```
    "City": "Belfast",
    "Street": "Oakley Rd.",
    "Number": " "
   }
  },
  {
   "First_Name": "Andrew",
   "Last_Name": "Hernandez",
   "Age": 18,
   "Favourite_Sports": ["Basketball"],
   "Address": {
    "City": "Dublin",
    "Street": " ",
    "Number": 4
   }
  }
]
```


Questions

1. *Display all information about each person in the collection except for their address.*

Use the project option to show all of the information except for the address
Project: {First_Name: 1, Last_Name: 1, Age: 1, Favourite_Sports: 1}

2. *What age is Carol?*

Filter for the first name ‘Carol’
Show her age using the project option
Filter: {First_Name: 'Carol'}
Project: {Age: 1}

3. *Display the age of all records in the database in descending order. Also display their first name and last name.*

Use the project option to show the first name, last name and age
Sort by age descending
Project: {First_Name: 1, Last_Name: 1, Age: 1}
Sort: {Age: -1}

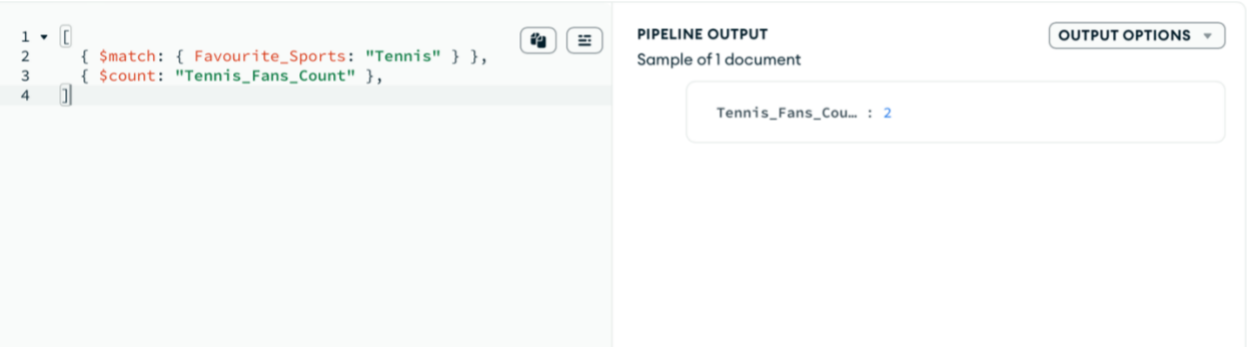
4. Display the name and surname of everyone who has tennis as their favourite sport.

Show people with tennis as their favourite sport
Use the project option to show the first name, last name, and favourite sport of each person
Filter: {Favourite_Sports: 'Tennis'}
Project: {First_Name: 1, Last_Name: 1, Favourite_Sports: 1}

5. Count the number of members who have tennis as their favourite sport.

Use aggregation option
$match function filters favourite sport for tennis
$count function counts the number of entries and renames is as “Tennis_Fans_Count”

Aggregation
[{$match: {Favourite_Sports: "Tennis"}},
{$count: "Tennis_Fans_Count"}]



6. *Display first name, last name, and age of all people older than 30 years old.*

Filter by age with $gt displaying entries greater than 30
Project option shows first name, last name, age, but no id
Filter: { Age: { $gt: 30 } }
Project: { First_Name: 1, Last_Name: 1, Age: 1, _id: 0 }

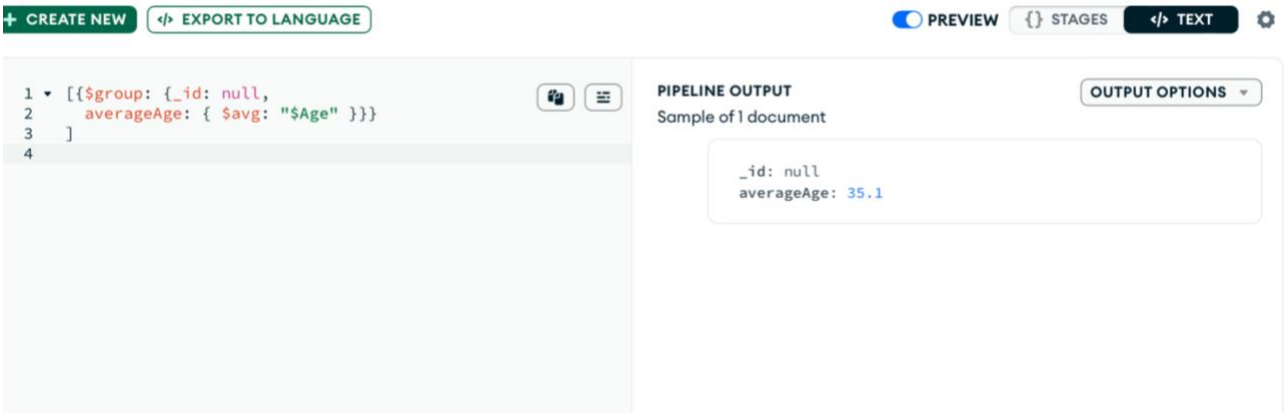
7. *Retrieve the name of all members whose first name contains the letter a.*

$regex function shows entries containing the letter ‘a’
Project option shows just the first name but not the id
Filter: { First_Name: { $regex: /a/i } }
Project: { First_Name: 1, _id: 0 }

8. Calculate the average age of everyone in the database.

Use aggregation option
$group adds the age of the group and the $avg averages the age
Name it averageAge

Aggregation
[{$group: {_id: null,
  averageAge: { $avg: "$Age" }}}
]



9. Calculate the average age of people who have tennis as their favourite sport.

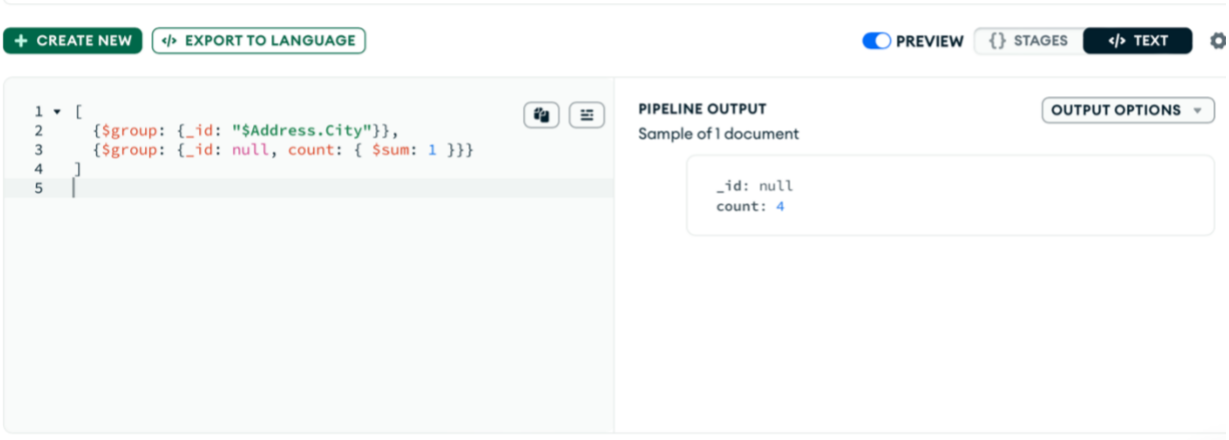
Use aggregation option
$match shows only people with tennis as their favourite sport
$group adds the age of the group and the $avg averages the age
Name it averageAge

Aggregation
[
  {$match: {Favourite_Sports: "Tennis"}},
  {$group: {_id: null, averageAge: { $avg: "$Age" }}}
]

10. How many unique cities are listed in this collection?

Use aggregation option
$group groups the addresses by city
It then displays the unique cities in the collection

Aggregation
```
[
  {$group: {_id: "$Address.City"}},
  {$group: {_id: null, count: { $sum: 1 }}}
]
```



Question 3 - Reflection on changes on the Big Data landscape

Chosen Question: 1

From reading "The Road to Composable Data Systems: Thoughts on the Last 15 Years and the Future" by Wes McKinney (2013) it was apparent that tools and methodologies used in data analytics have evolved drastically since the author started building data analysis tools. Throughout the blog post the author delves into his journey and offers intriguing insights into some of the challenges and innovations in the sector that have shaped data analytics. He stresses the importance of not suffering from "fragmentation" and for companies to collaborate to tackle potential challenges.

Initially it was clear that while python and more specifically the pandas community were popular, as pandas grew and expanded, so did its complexities. Pandas had foundational issues which led to the realization that it was limited in terms of performance and interoperability, particularly in terms of big data. It shows that as technology advances, each advancement brings both new possibilities and challenges.

As these new challenges became evident, there was a collective awakening with projects such as 'Apache Arrow,' 'RAPIDS,' 'Ibis,' 'dplyr,' and 'Substrait' developed to tackle these challenges. These projects addressed both immediate needs and laid the groundwork for future advancements. These advancements have come from harnessing cutting-edge technologies such as the integration of GPU acceleration into data analytics.

Looking ahead, the author wishes for a holistic approach to be taken to tackling these challenges, hoping for no more “fragmentation.” From each step forward and each challenge that is overcome, we are closer to understanding the true potential of these initiatives.

In conclusion, the road to composable data systems discussed by the author reflects a great testament to the ingenuity and innovation of the data science community, and I’m excited to see how the sector is shaped in the future because of these efforts.

References

McKinney, W. (2013) The Road to Composable Data Systems: Thoughts on the Last 15 Years and the Future [Online], Available at: https://wesmckinney.com/blog/looking-back-15-years/ (Accessed: March 20, 2024)