UNIVERSITY OF PORTSMOUTH SCHOOL OF COMPUTING

Module Name:

M32363 Database Principle (DBPRIN)

Lab Assignment (Individual)

Date:

2 Aug 2023

Intake: _

PT-UOP BScDSA-11__________

UOP ID Number: UP2200918

Contents

LAB 1 (16/07/2023) - Database Design

Lab Description

Lab 1 involves understanding and applying concepts related to the Entity-Relationship (ER) model, cardinality, and relationship types. Covers topics such as developing an ERD based on a scenario, identifying entity attributes and primary keys, determining cardinality and relationship symbols, and mapping ERD to table structures. The lab also includes SQL queries to retrieve specific information from the database based on the given ERD and table structure.

Q1. Which of the following steps occurs first in the process of building an ERD based on a scenario?

a. Develop the initial ERD.

b. Create a detailed narrative of the organisation's description of operations.

c. Identify the attributes and primary keys that adequately describe the entities.

d. Identify the business rules based on the description of operations.

Q2. A student can attend 5 modules. Different lecturers can offer the same module. The relationship of students → lecturers is a _____________ relationship.

a. Many-to-many (M:M)

b. One-to-many (1:M)

c. One-to-one (1:1)

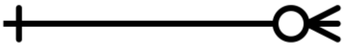
d. Many-to-one (M:1)

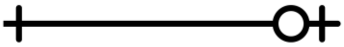
e. Many-to-zero OR one (M:0-1)

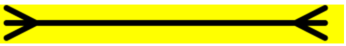
f. One-to-one OR many (1:1-M)

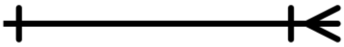
g. Many-to-zero

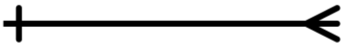
Q3. What would be the cardinality symbol for the relationship identified in Q2?

a. 

b. 

c. 

d. 

e. 

Q4. Which of the following statements best describes the function of an entity relation model?

a. An ER model is concerned primarily with the view of the attributes that will be used in physical implementation

b. An ER model is concerned primarily with a physical implementation of the data and secondly with the logical view

c. An ER model provides a view of the logic of the data and not the physical implementation

d. An ER model is entirely concerned with modelling the physical implementation

Q5. Consider Figure 1 representing instances of a relationship between EMPLOYEE and the DEPARTMENT that the employees work in.

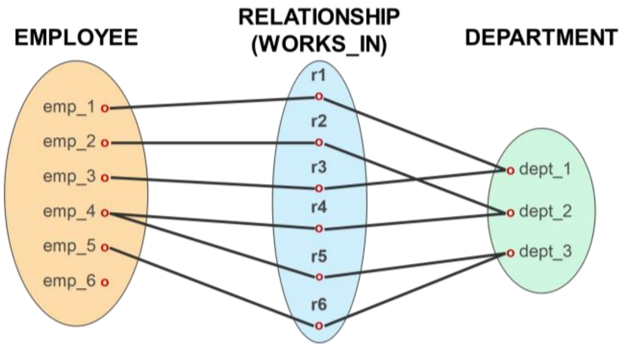


Figure.1

Which of the following relationships are solved and represented by the above figure?

a. **1:1** relationship between EMPLOYEE and DEPARTMENT and total participation from EMPLOYEE and total participation from DEPARTMENT

b. **1:M** relationship between EMPLOYEE and DEPARTMENT and partial participation from EMPLOYEE and partial participation from DEPARTMENT

c. **M:M** relationship between EMPLOYEE and DEPARTMENT and partial participation from EMPLOYEE and total participation from DEPARTMENT

d. **M:1** relationship between EMPLOYEE and DEPARTMENT and total participation from EMPLOYEE and total participation from DEPARTMENT

Q6. Suppose we map the following ERD (Figure 2) to the relations T1(A, B) and T2(B, C). The CREATE table statement for T2 is defined as the following:

CREATE TABLE T2(B SERIAL PRIMARY KEY, C INT).

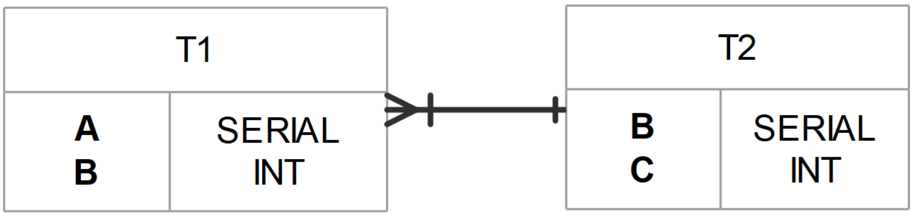
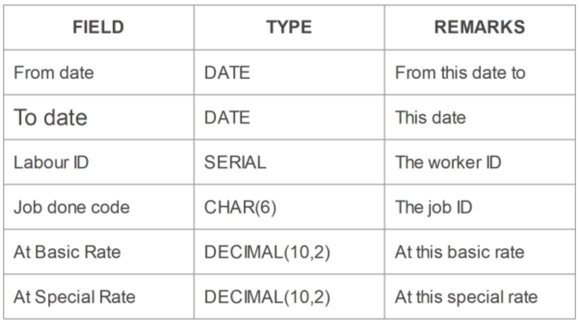


Figure. 2

Which one of the following create table statements would be correct for T1, enforcing the FK and relationship?

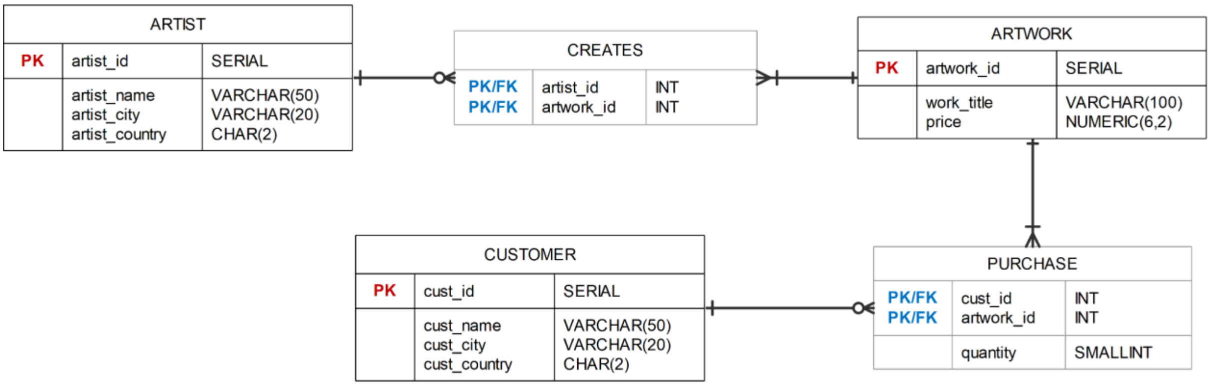
a. CREATE TABLE T1(A SERIAL PRIMARY KEY UNIQUE, B INT, FOREIGN KEY NOT NULL);
b. CREATE TABLE T1(A SERIAL PRIMARY KEY, B INT NOT NULL REFERENCES T2(B));
c. CREATE TABLE T1(A SERIAL UNIQUE, B REFERENCES T2(B));
d. CREATE TABLE T1(A SERIAL, B INT);

Q7. In the manufacturing industry labourers are given different jobs on different days and each job has its own monthly basic and monthly special rates as wages to be paid to labourers. A worker is not given more than one type of job on a day. A database designer is given the job to design database for above situation and the designer designs a draft of one of the tables as :



FIELD	TYPE	REMARKS
From date	DATE	From this date to
To date	DATE	This date
Labour ID	SERIAL	The worker ID
Job done code	CHAR(6)	The job ID
At Basic Rate	DECIMAL(10,2)	At this basic rate
At Special Rate	DECIMAL(10,2)	At this special rate

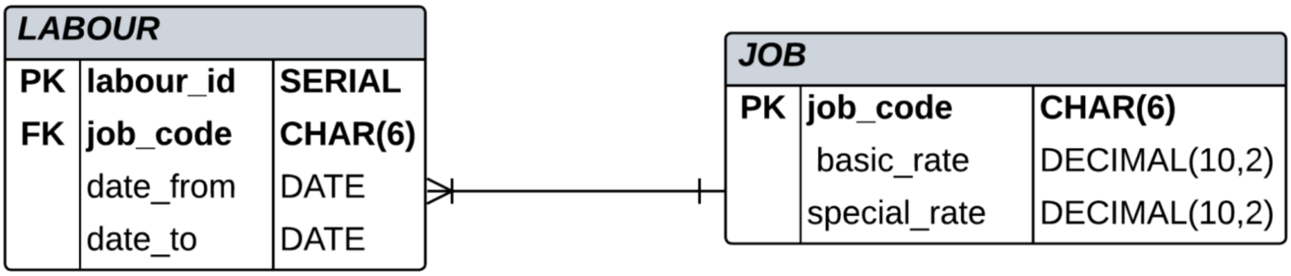
Fig. 3



Draw an ERD for the above situation that represents the relationship between **job** and **labour** (place each attribute to the correct table along with data type/size).

Fig. 4

A:



Create a new database in your VM machine named **lab1** and paste the provided SQL code [dbprin_lab1] into the database. Analysing the provided ERD (Figure 4) and the code, provide answers for the following questions.

Note: *You should format ALL your queries for the optimal output using appropriate column names and CONCAT where available (e.g. instead of "cus_name" it should be "Customer Name").*

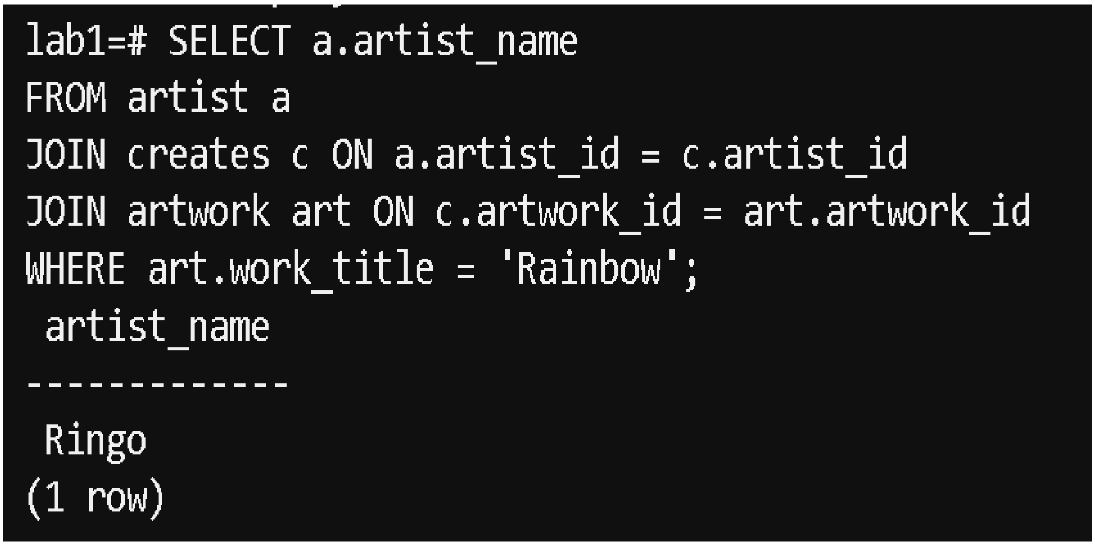
Q8. Find the name of the artist who has created the Artwork titled ‘*Rainbow*’.

A:

[EDITABLE CODE]

```
SELECT a.artist_name
FROM artist a
JOIN creates c ON a.artist_id = c.artist_id
JOIN artwork art ON c.artwork_id = art.artwork_id
WHERE art.work_title = 'Rainbow';
```

[OUTPUT SCREENSHOT]



```
lab1=# SELECT a.artist_name
FROM artist a
JOIN creates c ON a.artist_id = c.artist_id
JOIN artwork art ON c.artwork_id = art.artwork_id
WHERE art.work_title = 'Rainbow';
 artist_name
-------------
 Ringo
(1 row)
```

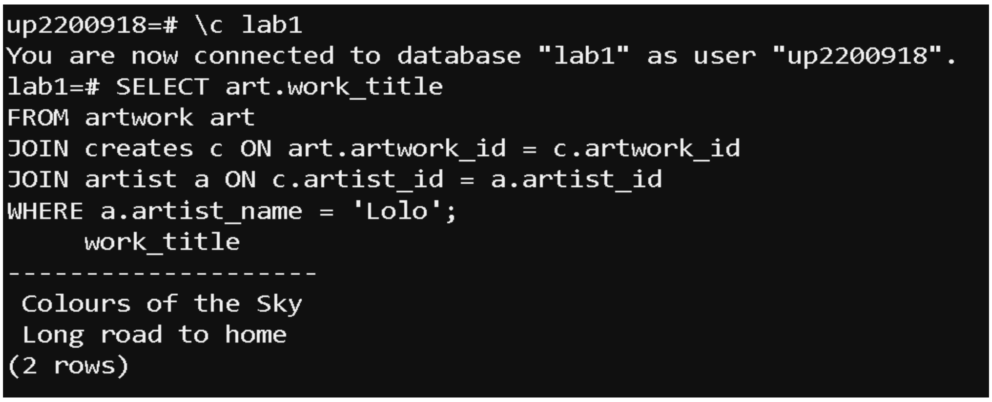
Q9. Find the titles of the Artworks created by ‘Lolo’.

A:

[EDITABLE CODE]

```
SELECT art.work_title
FROM artwork art
JOIN creates c ON art.artwork_id = c.artwork_id
JOIN artist a ON c.artist_id = a.artist_id
WHERE a.artist_name = 'Lolo';
```

[OUTPUT SCREENSHOT]



```
up2200918=# \c lab1
You are now connected to database "lab1" as user "up2200918".
lab1=# SELECT art.work_title
FROM artwork art
JOIN creates c ON art.artwork_id = c.artwork_id
JOIN artist a ON c.artist_id = a.artist_id
WHERE a.artist_name = 'Lolo';
     work_title
--------------------
 Colours of the Sky
 Long road to home
(2 rows)
```

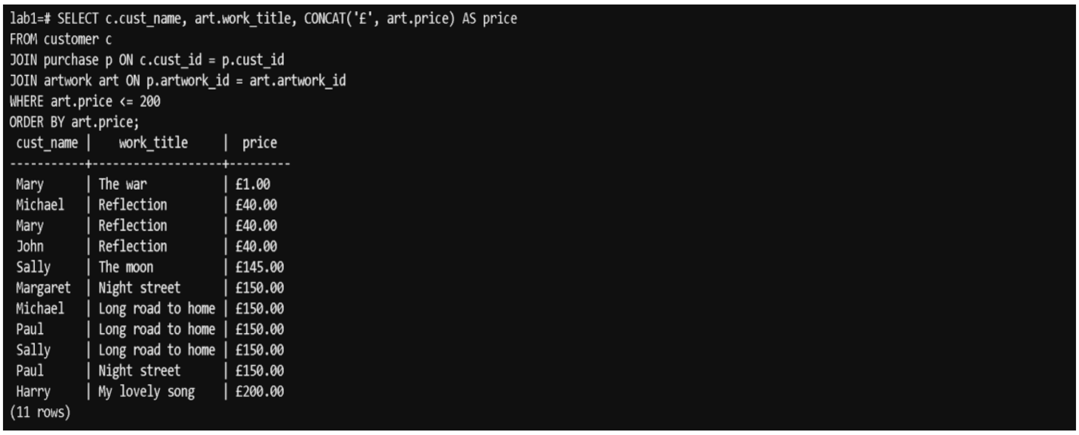
Q10. Find the names of customers who have not bought an artwork priced more than £ 200. List the customer’s name, artwork title and the price. Add the £ sign to the output and order on price from lowest to highest.

A:

[EDITABLE CODE]

```
SELECT c.cust_name, art.work_title, CONCAT('£', art.price) AS price
FROM customer c
JOIN purchase p ON c.cust_id = p.cust_id
JOIN artwork art ON p.artwork_id = art.artwork_id
WHERE art.price <= 200
ORDER BY art.price;
```

[OUTPUT SCREENSHOT]



```
lab1=# SELECT c.cust_name, art.work_title, CONCAT('£', art.price) AS price
FROM customer c
JOIN purchase p ON c.cust_id = p.cust_id
JOIN artwork art ON p.artwork_id = art.artwork_id
WHERE art.price <= 200
ORDER BY art.price;
 cust_name |    work_title     |  price
-----------+-------------------+---------
 Mary      | The war           | £1.00
 Michael   | Reflection        | £40.00
 Mary      | Reflection        | £40.00
 John      | Reflection        | £40.00
 Sally     | The moon          | £145.00
 Margaret  | Night street      | £150.00
 Michael   | Long road to home | £150.00
 Paul      | Long road to home | £150.00
 Sally     | Long road to home | £150.00
 Paul      | Night street      | £150.00
 Harry     | My lovely song    | £200.00
(11 rows)
```

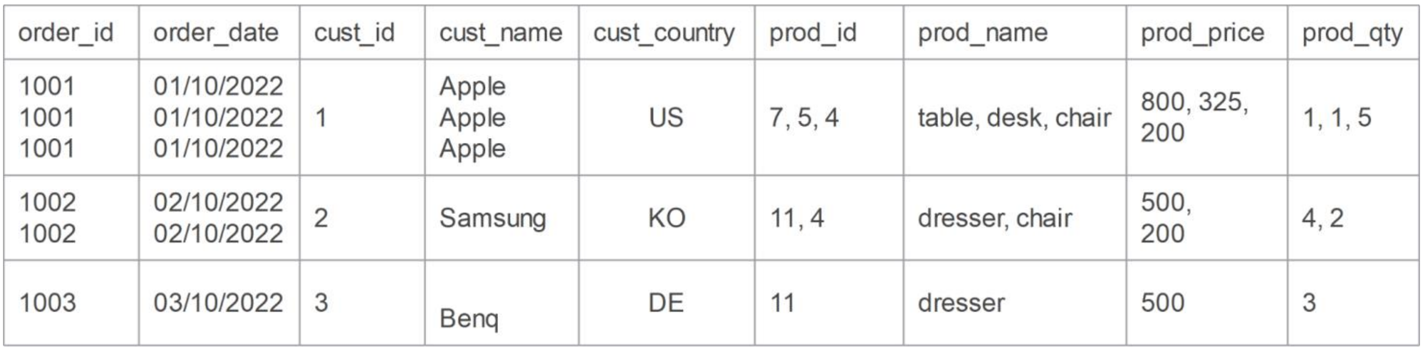
Conclusion/Reflection

Lab 1 provided practical experience in database design using the ER model. I learned how to identify entities, attributes, and relationships in a scenario and represent them in an ERD. The lab also improved my skills in SQL query writing and database mapping. Overall, it gave me a strong foundation in database modeling and design.

LAB 2 (16/07/2023) - Normalization

Lab Description

Lab 2 involves normalizing tables to achieve 1NF and 3NF, creating an Entity-Relationship Diagram (ERD) based on the normalized tables, and working with data types and sizes. It also includes setting up a new database, creating tables with appropriate primary keys, foreign keys, constraints, and data types, as well as inserting sample data into the tables. The lab focuses on database normalisation principles and schema design.



order_id	order_date	cust_id	cust_name	cust_country	prod_id	prod_name	prod_price	prod_qty
1001 1001 1001	01/10/2022 01/10/2022 01/10/2022	1	Apple Apple Apple	US	7, 5, 4	table, desk, chair	800, 325, 200	1, 1, 5
1002 1002	02/10/2022 02/10/2022	2	Samsung	KO	11, 4	dresser, chair	500, 200	4, 2
1003	03/10/2022	3	Benq	DE	11	dresser	500	3

Fig. 1

Q1. You have the following table (Fig.1) with data. Normalise the table in 1NF (show 1NF) and create the ERD for 3NF, with associated data type. You can use Visio to create diagram

Hint: *You should have 4 (or 5) tables for 3NF*

A:

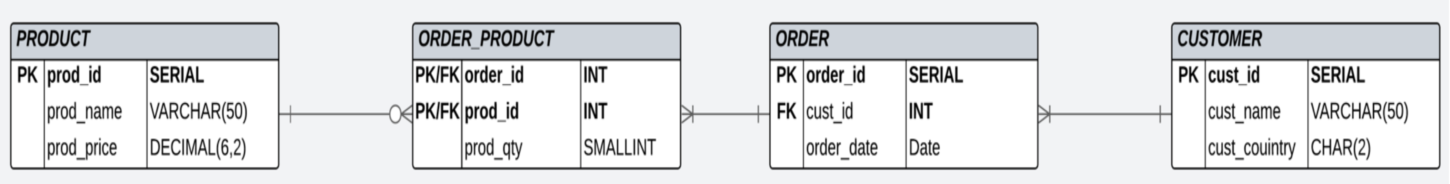
order_id	order_date	cust_id	cust_name	cust_country	prod_id	prod_name	prod_price	prod_qty
1001	01/10/2022	1	Apple	US	7	table	800	1
1001	01/10/2022	1	Apple	US	5	desk	325	1
1001	01/10/2022	1	Apple	US	4	chair	200	5
1002	02/10/2022	2	Samsung	KO	11	dresser	500	4
1002	02/10/2022	2	Samsung	KO	4	chair	200	2
1003	03/10/2022	3	Benq	DE	11	dresser	500	3

PRODUCT (prod_id, prod_name, prod_price)

CUSTOMER (cust_id, cust_name, cust_country)

ORDER (order_id, cust_id*, order_date)

ORDER_PRODUCT (order_id*, prod_id*, prod_qty)



PRODUCT		
PK	prod_id	SERIAL
	prod_name	VARCHAR(50)
	prod_price	DECIMAL(6,2)

ORDER_PRODUCT		
PK/FK	order_id	INT
PK/FK	prod_id	INT
	prod_qty	SMALLINT

ORDER		
PK	order_id	SERIAL
FK	cust_id	INT
	order_date	Date

CUSTOMER		
PK	cust_id	SERIAL
	cust_name	VARCHAR(50)
	cust_couintry	CHAR(2)

Q2. You are given the following table in 1NF. Normalise data in 3NF and create the ERD with all data type and size. One Employee works 1 or M projects and on each project a different time is allocated

dept_id	dep_name	emp_id	emp_name	project_id	project_name	budgeted_time
10	Finance	1	Harry	100	Alpha	4.5
10	Finance	5	Dewey	105	Beta	3
10	Finance	11	Louie	103	Gamma	7
20	R&D	2	Jack	107	Delta	8
20	R&D	4	Jill	102	Echo	9

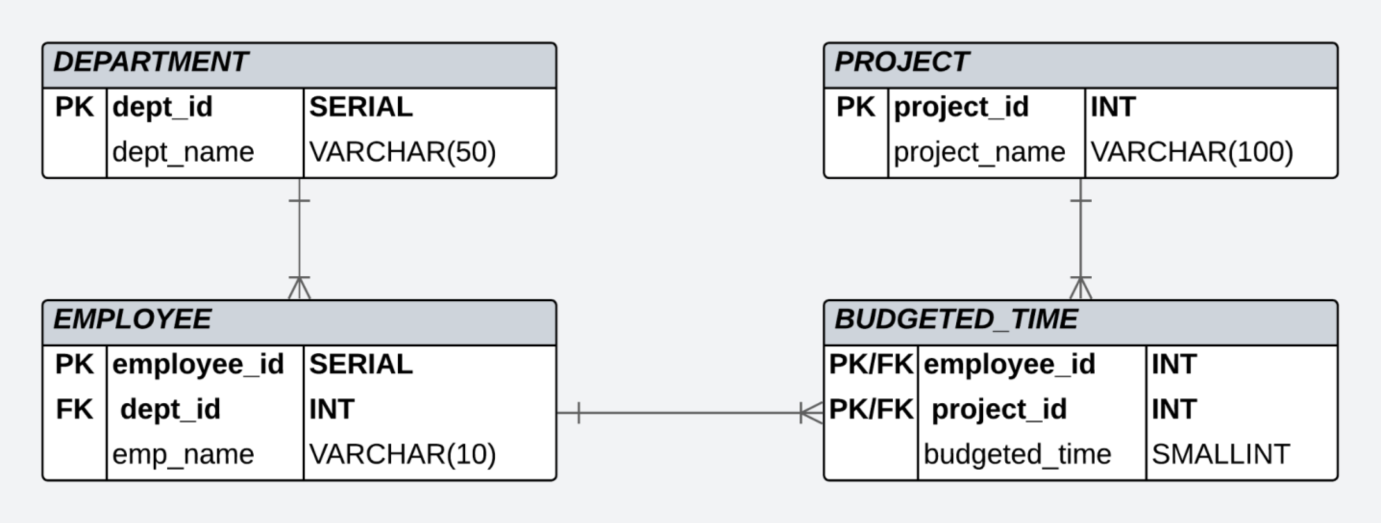
A:

DEPARTMENT(dept_id, dept_name)

EMPLOYEE(emp_id, dept_id, emp_name)

PROTECT(project_id, project_name)

BUDGET_TIME(emp_id*, project_id*, budgeted_time)



DEPARTMENT		
PK	**dept_id**	**SERIAL**
	dept_name	VARCHAR(50)

PROJECT		
PK	**project_id**	**INT**
	project_name	VARCHAR(100)

EMPLOYEE		
PK	**employee_id**	**SERIAL**
FK	**dept_id**	**INT**
	emp_name	VARCHAR(10)

BUDGETED_TIME		
PK/FK	**employee_id**	**INT**
PK/FK	**project_id**	**INT**
	budgeted_time	SMALLINT

Q3. **University of Portsmouth keeps the following details about a student and the various modules the student studied (not accurate):**

up_number (student registration number to university)

stu_name (student name)

stu_addr (student address)

tut_id (tutor id)

tut_name (tutor name)

course_id - (course code)

course_name (course name)

module_id (module code)

module_name (module name)

module_results (module exam result)

Which of the following is one of the steps for normalizing **the relation Student to 1NF** as a student can have only one COURSE and one TUTOR but multiple modules?

Hint: you can input a quick sample output in a spreadsheet (like Q1) and remove the repeating group.

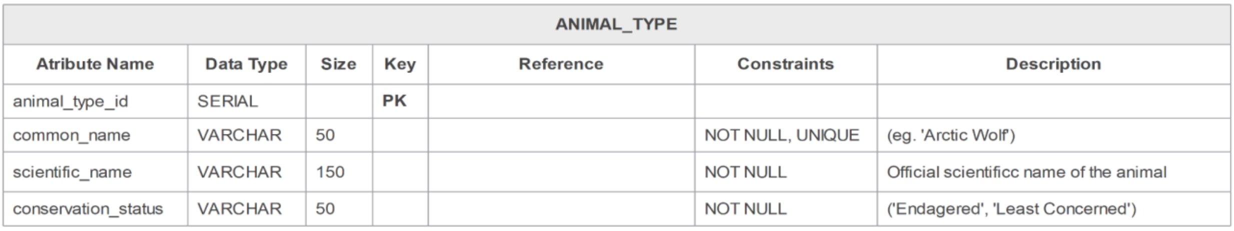
a. **STUDENT**(up_number, stu_name, stu_addr, tutor_id, tutor_name, course_id, course_name)
   MODULE(module_id, module_name, module_results)
b. **STUDENT**(up_number, stu_name, stu_addr, tutor_id, tutor_name, course_id, course_name, module_code, module_results)
   MODULE (module_id, module_name, course_id)
c. **STUDENT**(stu_name, stu_addr, tutor_id, tutor_name, course_id, module_id, module_name, module_results)
   COURSE(up_number, course_id, course_name)
d. **STUDENT**(student_id**,** stu_name, stu_addr)
   MODULE(tutor_id, module_id, module_name, module_results)
   COURSE(course_id, course_name, tutor_id, tutor_name)

Q4. Based on the previous data sample what would be a 3NF normalisation? Write just the table name and the attributes as above.

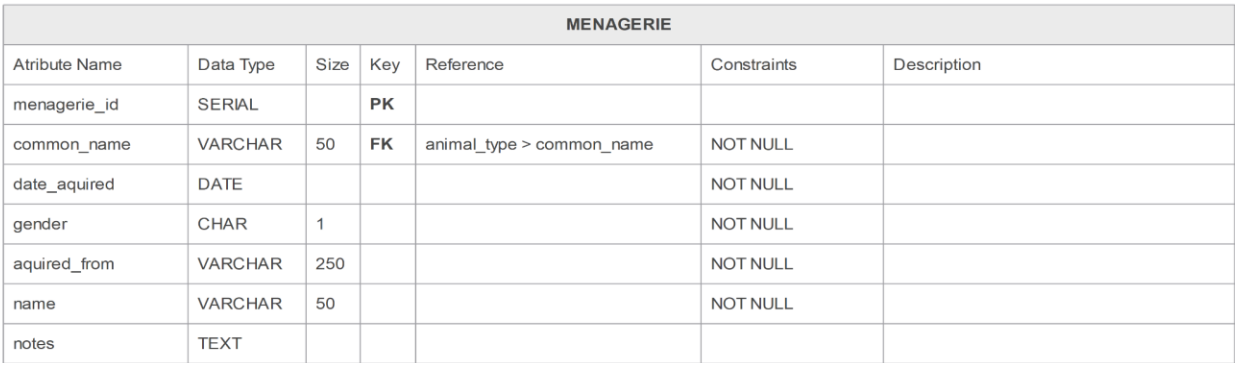
A:

STUDENT(stu_id, stu_name, stu_addr)
MODULE(module_id, module_name, module_results)
COURSE(course_id, course_name)
TUTOR(tut_id, tut_name)

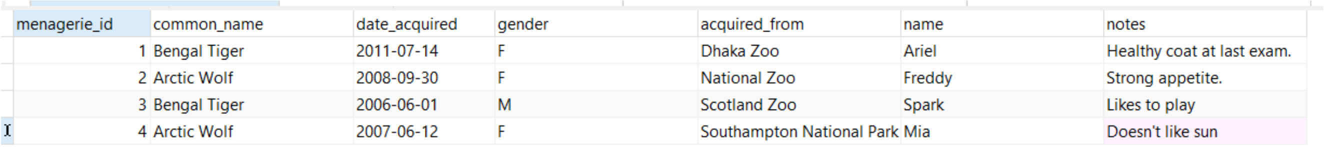
Q5. Open your VM and create a new database (lab2). You are given the following data dictionary.



ANIMAL_TYPE						
Atribute Name	**Data Type**	**Size**	**Key**	**Reference**	**Constraints**	**Description**
animal_type_id	SERIAL		**PK**			
common_name	VARCHAR	50			NOT NULL, UNIQUE	(eg. 'Arctic Wolf')
scientific_name	VARCHAR	150			NOT NULL	Official scientificc name of the animal
conservation_status	VARCHAR	50			NOT NULL	('Endagered', 'Least Concerned')



MENAGERIE						
Atribute Name	Data Type	Size	Key	Reference	Constraints	Description
menagerie_id	SERIAL		**PK**			
common_name	VARCHAR	50	**FK**	animal_type > common_name	NOT NULL	
date_aquired	DATE				NOT NULL	
gender	CHAR	1			NOT NULL	
aquired_from	VARCHAR	250			NOT NULL	
name	VARCHAR	50			NOT NULL	
notes	TEXT					



menagerie_id	common_name	date_acquired	gender	acquired_from	name	notes
1	Bengal Tiger	2011-07-14	F	Dhaka Zoo	Ariel	Healthy coat at last exam.
2	Arctic Wolf	2008-09-30	F	National Zoo	Freddy	Strong appetite.
3	Bengal Tiger	2006-06-01	M	Scotland Zoo	Spark	Likes to play
4	Arctic Wolf	2007-06-12	F	Southampton National Park	Mia	Doesn't like sun

and the data sample output:

- Write the CREATE statements for tables ANIMAL_TYPE and MENAGERIE, including PKs, FKs, constraints, data type and size.

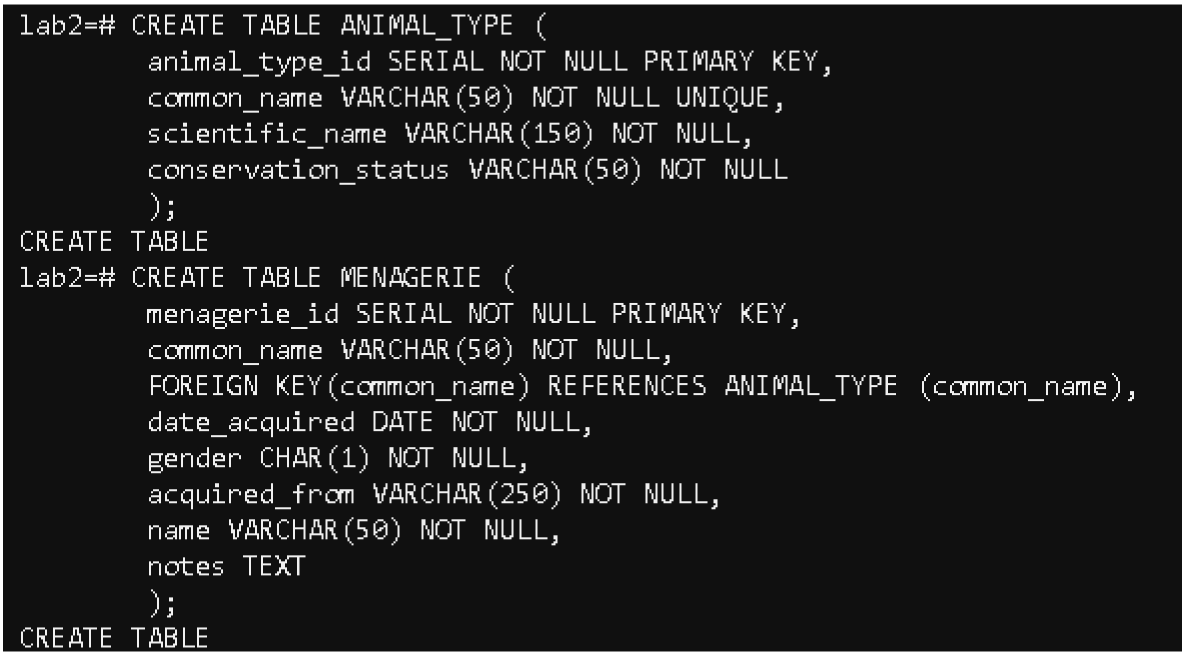
A:

[EDITABLE CODE]

```
CREATE TABLE ANIMAL_TYPE (
    animal_type_id SERIAL NOT NULL PRIMARY KEY,
    common_name VARCHAR(50) NOT NULL UNIQUE,
    scientific_name VARCHAR(150) NOT NULL,
    conservation_status VARCHAR(50) NOT NULL
    );

CREATE TABLE MENAGERIE (
    menagerie_id SERIAL NOT NULL PRIMARY KEY,
    common_name VARCHAR(50) NOT NULL,
    FOREIGN KEY(common_name) REFERENCES ANIMAL_TYPE (common_name),
    date_acquired DATE NOT NULL,
    gender CHAR(1) NOT NULL,
    acquired_from VARCHAR(250) NOT NULL,
    name VARCHAR(50) NOT NULL,
    notes TEXT
    );
```

[OUTPUT SCREENSHOT]



```
lab2=# CREATE TABLE ANIMAL_TYPE (
        animal_type_id SERIAL NOT NULL PRIMARY KEY,
        common_name VARCHAR(50) NOT NULL UNIQUE,
        scientific_name VARCHAR(150) NOT NULL,
        conservation_status VARCHAR(50) NOT NULL
        );
CREATE TABLE
lab2=# CREATE TABLE MENAGERIE (
        menagerie_id SERIAL NOT NULL PRIMARY KEY,
        common_name VARCHAR(50) NOT NULL,
        FOREIGN KEY(common_name) REFERENCES ANIMAL_TYPE (common_name),
        date_acquired DATE NOT NULL,
        gender CHAR(1) NOT NULL,
        acquired_from VARCHAR(250) NOT NULL,
        name VARCHAR(50) NOT NULL,
        notes TEXT
        );
CREATE TABLE
```

- Write 6 INSERT STATEMENTS for the following:
    - Common Name: ‘Bengal Tiger’, ‘Arctic Wolf’
    - Scientific Name: ‘Panthera tigris tigris’, ‘Canis lupus arctos’
    - Conservation Status: ‘Endangered’, ‘Least Concern’
    - Acquired Date: ‘14/07/2011’, ‘30/09/2008’, ‘01/06/2006’, ‘12/06/2007’
    - Gender: ‘M’, ‘F’
    - Acquired From: ‘Dhaka Zoo’, ‘National Zoo’, ‘Scotland Zoo’, ‘Southampton National Park’
    - Name: ‘Ariel’, ‘Freddy’, ‘Spark’, ‘Mia’
    - Notes: ‘Healthy coat at last exam’, ‘Strong appetite’, ‘Likes to play’, ‘Doesn't like sun’

The output of both tables should be exactly like in provided examples for ANIMAL_TYPE and MENAGERIE.

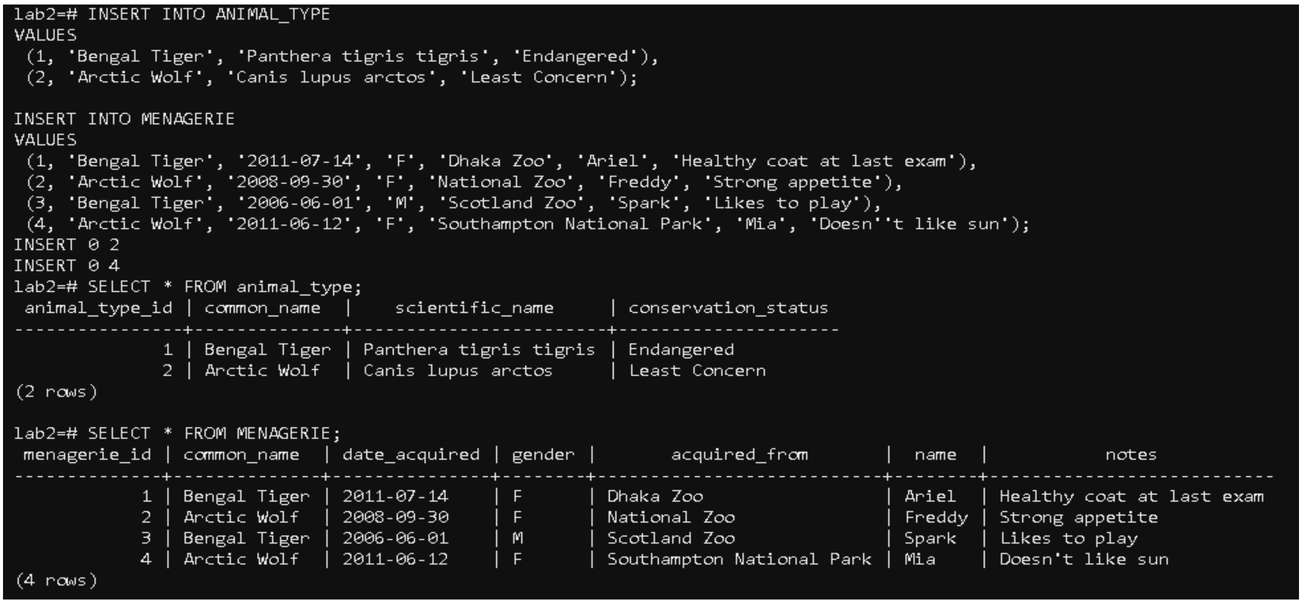
A:

[EDITABLE CODE]

```
INSERT INTO ANIMAL_TYPE
VALUES
 (1, 'Bengal Tiger', 'Panthera tigris tigris', 'Endangered'),
 (2, 'Arctic Wolf', 'Canis lupus arctos', 'Least Concern');

INSERT INTO MENAGERIE
VALUES
 (1, 'Bengal Tiger', '2011-07-14', 'F', 'Dhaka Zoo', 'Ariel', 'Healthy coat at last exam'),
 (2, 'Arctic Wolf', '2008-09-30', 'F', 'National Zoo', 'Freddy', 'Strong appetite'),
 (3, 'Bengal Tiger', '2006-06-01', 'M', 'Scotland Zoo', 'Spark', 'Likes to play'),
 (4, 'Arctic Wolf', '2011-06-12', 'F', 'Southampton National Park', 'Mia', 'Doesn''t like sun');
```

[OUTPUT SCREENSHOT]



```
lab2=# INSERT INTO ANIMAL_TYPE
VALUES
 (1, 'Bengal Tiger', 'Panthera tigris tigris', 'Endangered'),
 (2, 'Arctic Wolf', 'Canis lupus arctos', 'Least Concern');

INSERT INTO MENAGERIE
VALUES
 (1, 'Bengal Tiger', '2011-07-14', 'F', 'Dhaka Zoo', 'Ariel', 'Healthy coat at last exam'),
 (2, 'Arctic Wolf', '2008-09-30', 'F', 'National Zoo', 'Freddy', 'Strong appetite'),
 (3, 'Bengal Tiger', '2006-06-01', 'M', 'Scotland Zoo', 'Spark', 'Likes to play'),
 (4, 'Arctic Wolf', '2011-06-12', 'F', 'Southampton National Park', 'Mia', 'Doesn''t like sun');
INSERT 0 2
INSERT 0 4
lab2=# SELECT * FROM animal_type;
 animal_type_id | common_name  |    scientific_name     | conservation_status
----------------+--------------+------------------------+---------------------
              1 | Bengal Tiger | Panthera tigris tigris | Endangered
              2 | Arctic Wolf  | Canis lupus arctos     | Least Concern
(2 rows)

lab2=# SELECT * FROM MENAGERIE;
 menagerie_id | common_name  | date_acquired | gender |       acquired_from       |  name  |           notes
--------------+--------------+---------------+--------+---------------------------+--------+---------------------------
            1 | Bengal Tiger | 2011-07-14    | F      | Dhaka Zoo                 | Ariel  | Healthy coat at last exam
            2 | Arctic Wolf  | 2008-09-30    | F      | National Zoo              | Freddy | Strong appetite
            3 | Bengal Tiger | 2006-06-01    | M      | Scotland Zoo              | Spark  | Likes to play
            4 | Arctic Wolf  | 2011-06-12    | F      | Southampton National Park | Mia    | Doesn't like sun
(4 rows)
```

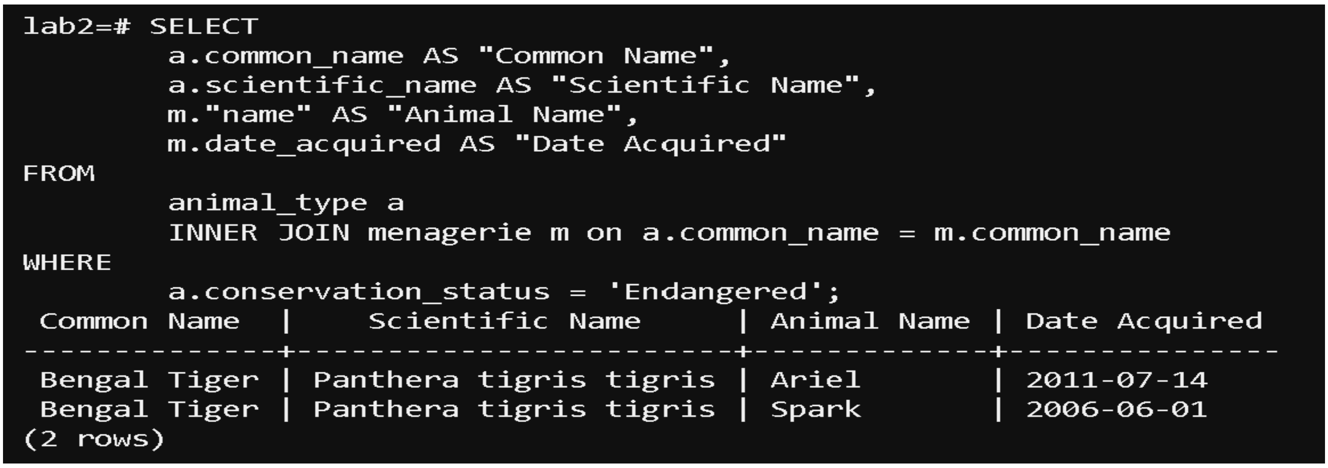
Q6. Based on the previous database you have created, list all the animals that are endangered, along with common name, scientific name, animal name and date acquired.

A:

[EDITABLE CODE]

```
SELECT
    a.common_name AS "Common Name",
    a.scientific_name AS "Scientific Name",
    m."name" AS "Animal Name",
    m.date_acquired AS "Date Acquired"
FROM animal_type a
INNER JOIN menagerie m ON a.common_name = m.common_name
WHERE
    a.conservation_status = 'Endangered';
```

[OUTPUT SCREENSHOT]



```
lab2=# SELECT
        a.common_name AS "Common Name",
        a.scientific_name AS "Scientific Name",
        m."name" AS "Animal Name",
        m.date_acquired AS "Date Acquired"
FROM
        animal_type a
        INNER JOIN menagerie m on a.common_name = m.common_name
WHERE
        a.conservation_status = 'Endangered';
 Common Name  |    Scientific Name     | Animal Name | Date Acquired
--------------+------------------------+-------------+---------------
 Bengal Tiger | Panthera tigris tigris | Ariel       | 2011-07-14
 Bengal Tiger | Panthera tigris tigris | Spark       | 2006-06-01
(2 rows)
```

Conclusion/Reflection

Lab 2 provided hands-on experience in database normalization and schema design. I learned how to eliminate data redundancy and organize data efficiently. Creating an ERD helped me visualize relationships between entities. Setting up the database and writing SQL queries improved my practical skills in database implementation.

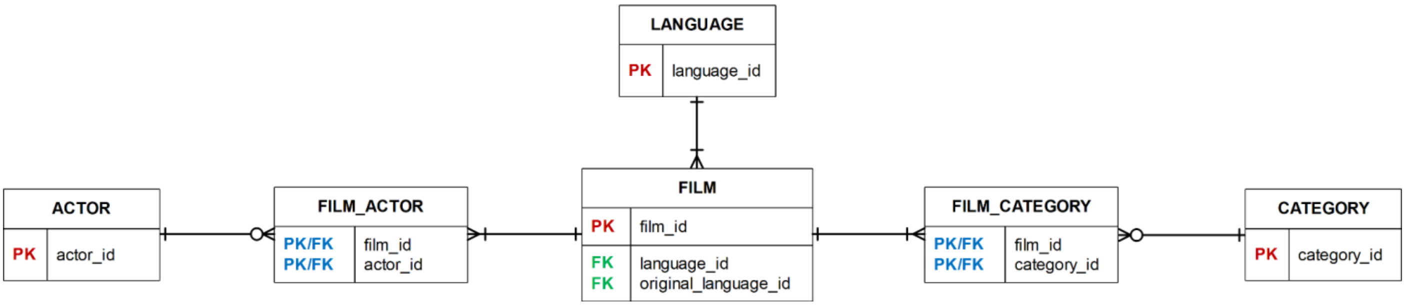
LAB 3 (26/07/2023) - Transactions

Lab Description

Lab 3 focused on database transactions and normalization. Involved building a database based on an ERD and performing various operations using SQL queries such as SELECT, INSERT, UPDATE, and DELETE. The lab also included the creation of a Transaction Analysis Matrix to analyze transactional behavior and its impact on the database.

Q1. You are given the following ERD without any additional attributes. Build up your database using your VM and **write 4 separate queries** that will contain a **SELECT**, **INSERT, UPDATE** and **DELETE** that will query the database. Create the ***Transaction Analysis Matrix*** for those transactions.

Do not spend time making the matrix “pretty” but accurate. You can use a spreadsheet or a table inserted in a document. Examples of the matrix are in the lecture presentation. Don't forget to add your work to the logbook.



(Query example - You might assume that some attributes are there (e.g. actor_name; actor_lname etc)

List the name of all movies where the actor's id (or surname) is X.

```
SELECT
 actor_name, film_name
FROM
 table_name
 JOIN table_name ON table1.attr = table2.attr
 JOIN table_name ON table3.attr = table4.attr
```

```
WHERE
 table.attr = 'X';
```

Same for an new INSERT and for an UPDATE

A:

[EDITABLE CODE]

```
-- SELECT
SELECT f.film_id, f.title
FROM film f
JOIN film_actor fa ON f.film_id = fa.film_id
JOIN actor a ON fa.actor_id = a.actor_id
WHERE a.last_name = 'X';
-- INSERT
INSERT INTO actor (actor_id, first_name, last_name)
VALUES (1, 'John', 'Doe');
-- UPDATE
UPDATE actor
SET last_name = 'Smith'
WHERE actor_id = 1;
-- DELETE
DELETE FROM actor
WHERE actor_id = 1;
```

Transaction Analysis Matrix:

```
          | actor             | film_actor  | film
---------------------------------------------------
SELECT    | READ (a)          | READ (fa)   | READ (f)
---------------------------------------------------
INSERT    | WRITE (a)         | --          | --
---------------------------------------------------
UPDATE    | READ/WRITE (a)    | --          | --
---------------------------------------------------
DELETE    | READ/WRITE (a)    | --          | --
```

Q2. Define in your own understanding (do not copy/paste definitions) of these terms: atomicity, consistency, isolation, durability, schedule, blind write, dirty read, unrepeatable read, serializable schedule, recoverable schedule, avoids-cascading-aborts schedule.

Note: *Some of the answers were in lecture, some in the book.*

A:

- **Atomicity**: Ensures transactions are all-or-nothing.
- **Consistency**: Keeps data valid and according to predefined rules.
- **Isolation**: Makes each transaction operate independently of others.
- **Durability**: Makes committed transactions survive system failures.
- **Schedule**: Defines the order of transaction execution.
- **Blind write**: An update operation that doesn't require reading the data first.
- **Dirty read**: Reading data from an uncommitted transaction.
- **Unrepeatable read:** Getting different values from reading the same data due to concurrent changes.
- **Serializable schedule:** Makes concurrent transactions equivalent to sequential ones.
- **Recoverable schedule:** Ensures database recovery after transaction failures.
- **Avoids-cascading-aborts schedule:** Prevents cascading rollbacks to preserve data consistency.

Q3. Consider the situation below, in which a number of account records have the following values:

A1 = £40; A2 = £50; A3 = £30

To transfer £10 from A3 to A1 while concurrently calculating the total funds in the three accounts, the following sequence of events may occur. Show the value of each data item in the last column, and discuss the reason for an incorrect summary value.

Note: Look at the lecture example and/or chapter 22 in Connolly and Begg.

Time	**Transaction1**	**Transaction2**	**SUM**
t_1	sum = 0		
t_2	read(**A1**)		
t_3	sum = sum + A1		
t4		read(**A3**)	
t5		A3 = A3 - 10	
t6		write(A3)	
t7		read(A1)	
t8		A1 = A1 + 10	
t9		write(A1)	
t10		commit;	
t11	read(**A3**)		
t12	sum = sum + A3		
t13	read(**A2**)		
t14	sum = sum + A2		

A:

Time	Transaction1	Transaction2	A1	A2	A3	SUM
t_1	sum = 0		40	50	30	0
t_2	read(A1)		40	50	30	0
t_3	sum = sum + A1		40	50	30	40
t4		read(A3)	40	50	30	40
t5		A3 = A3 - 10	40	50	30	40
t6		write(A3)	40	50	20	40
t7		read(A1)	40	50	20	40
t8		A1 = A1 + 10	40	50	20	50
t9		write(A1)	50	50	20	50
t10		commit;	50	50	20	50
t11	read(A3)		50	50	20	50
t12	sum = sum + A3		50	50	20	60
t13	read(A2)		50	50	20	60
t14	sum = sum + A2		50	50	20	110
t15	commit;					

Conclusion/Reflection

In Lab 3, I gained practical experience in handling transactions and implementing normalization techniques. I learned about transaction properties and how to ensure data consistency and integrity. The lab deepened my understanding of concurrency control and the importance of efficient database design. Overall, it enhanced my skills in database management and query execution.

LAB 4 (19/07/2023) - Relational Algebra

Lab Description

Lab 4 focused on SQL query formulation, its translation into relational algebra expressions, query tree drawings, and size estimations of query results, thereby enhancing understanding of database operations and data manipulation.

Q1. You are given the following query:

```
SELECT first_name, last_name,   hire_date
FROM Employees
WHERE
        Title = 'Sales Representative';
```

Write the correspondent RA with ***π*** and ***σ*** and draw the tree diagram to show this relation

Relational Algebra Symbols:

σ SELECT (or RESTRICT) - Sigma

π PROJECT - Pi

∪ UNION

∩ INTERSECTION

- DIFFERENCE

x PRODUCT (Cartesian Product)

⋈ ⟕ ⟖ JOINs - Theta

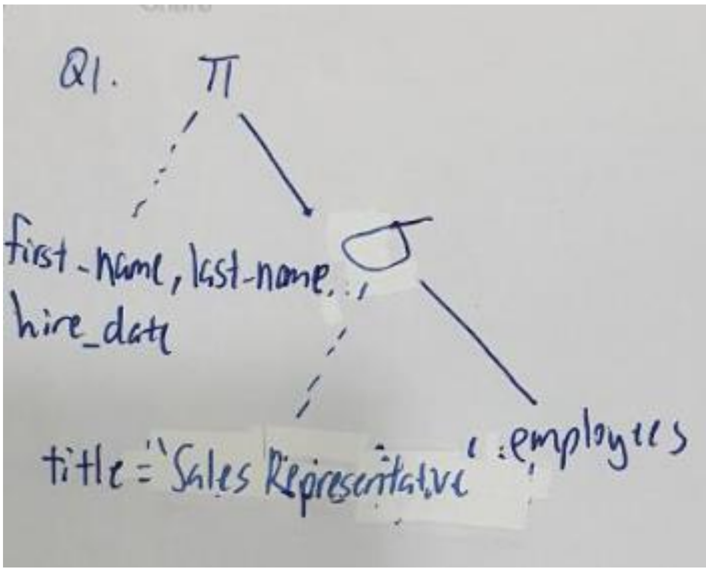
τ ORDER (SORT) - Tau

A:

Relational Algebra Expression:

π(first_name, last_name, hire_date)(***σ***Title='Sales Representative'(Employees))

Tree Diagram:


Q1.
π
first_name, last_name, hire_date
σ
title='Sales Representative'
employees

Q2. Let’s expand the above query and we are adding a second WHERE clause filter. Write the new RA and the tree diagram.

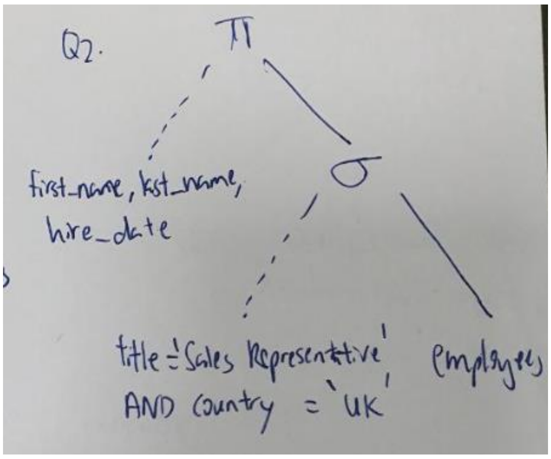
```
Select first_name, last_name,      hire_date
From Employees
Where
        Title = 'Sales Representative'
        and Country = 'UK';
```

A:

Relational Algebra Expression:

π(first_name, last_name, hire_date)(***σ***Title='Sales Representative' and Country='UK'(Employees))

Tree Diagram:


Q2.
π
first_name, last_name, hire_date
σ
title='Sales Representative' AND Country = 'UK'
employees

Q3. The following query has a conditional WHERE clause and ordering. What would be the RA for this?

```
SELECT order_id, order_date, shipper
FROM orders INNER JOIN shippers ON shippers . shipper_id = orders . ship_via
WHERE order_id < 10300
ORDER BY order_id
```

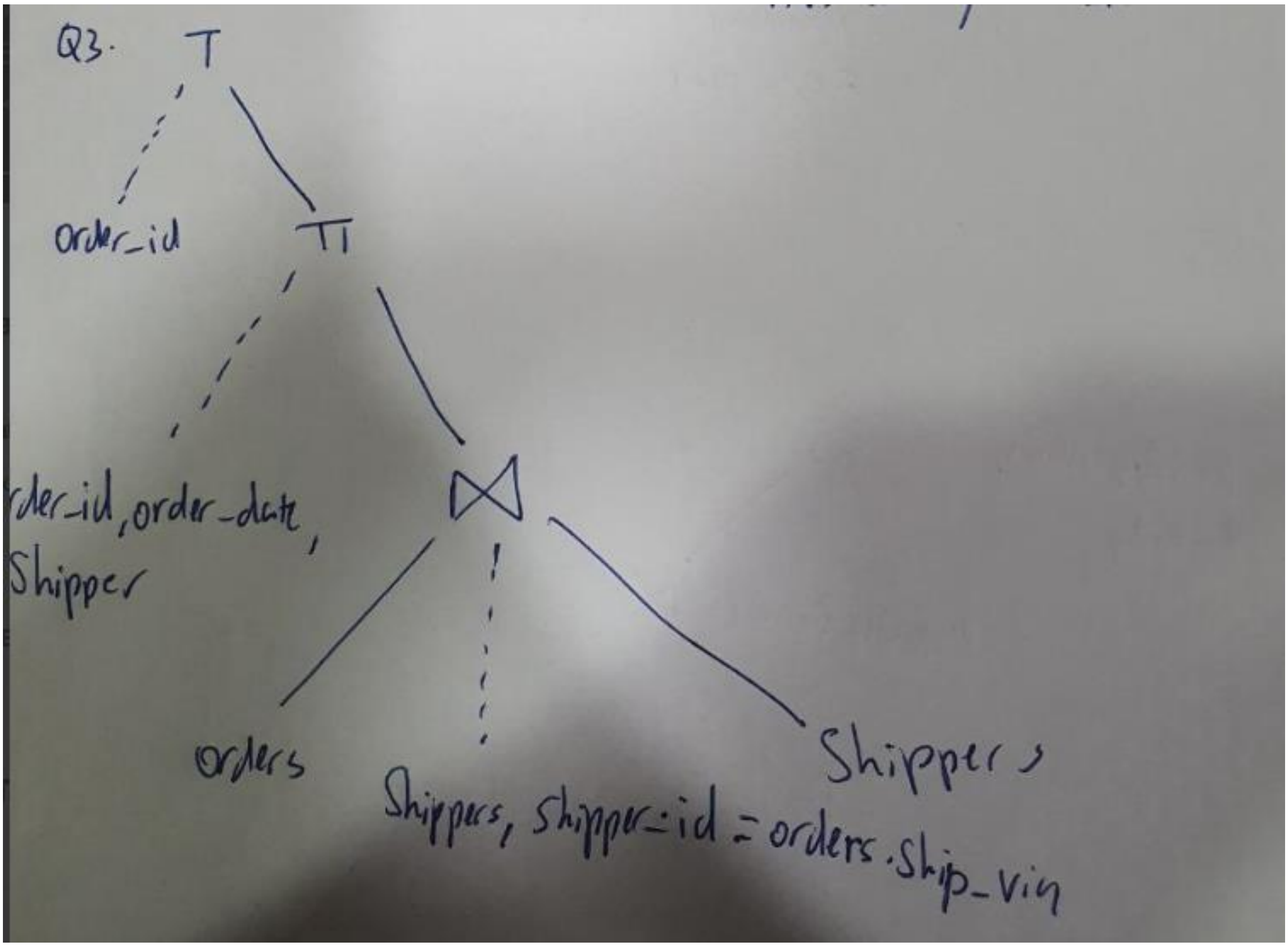
A:

Relational Algebra Expression:

т (order_id)

π(order_id, order_date, shipper)

(***σ***order_vvid < 10300 (orders ⋈ shippers.shipper_id = orders.ship_via shippers))



Q4. Consider a database with the following schema:

USER (name, age, gender) PK - name

FREQUENTS (name, pizzeria) PK name,pizzeria

EATS (name, pizza) PK - name, pizza

SERVES (pizzeria, pizza, price) PK - pizzeria, pizza

Write the **query, relational algebra expressions and query tree** for the following output.

“*Find the names of all females who eat either mushroom or pepperoni pizza (or both).***”**

Hint: *You can quickly draw the ERD using pen and paper for the above schema for an easier visualisation.*

A:

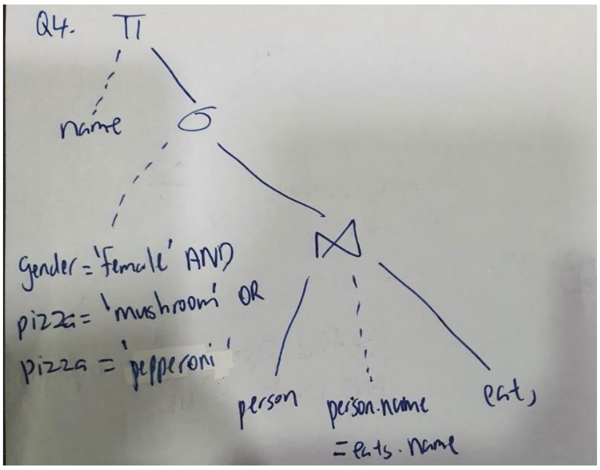
Query:

```
SELECT name
FROM user u
INNER JOIN eats e ON u.name = e.name
WHERE gender = 'female' AND
pizza = 'mushroom' OR pizza = 'pepperoni'
```

Relational Algebra Expressions:

π(name) (***σ*** gender=’female’ & pizza=’mushroom’ | pizza =’pepperoni’)(USER ⋈ EATS)

Query Tree:



Q5. Consider a database with the following schema:

PASSENGER (**p_id,** p_name, p_gender, p_city)

AGENCY (**a_id**, a_name, a_city)

FLIGHT (**f_id,** f_date, time, src, dest)

BOOKING (**p_id*, a_id*, f_id*, f_date**)

Write the relational algebra expression and query tree for the following request:

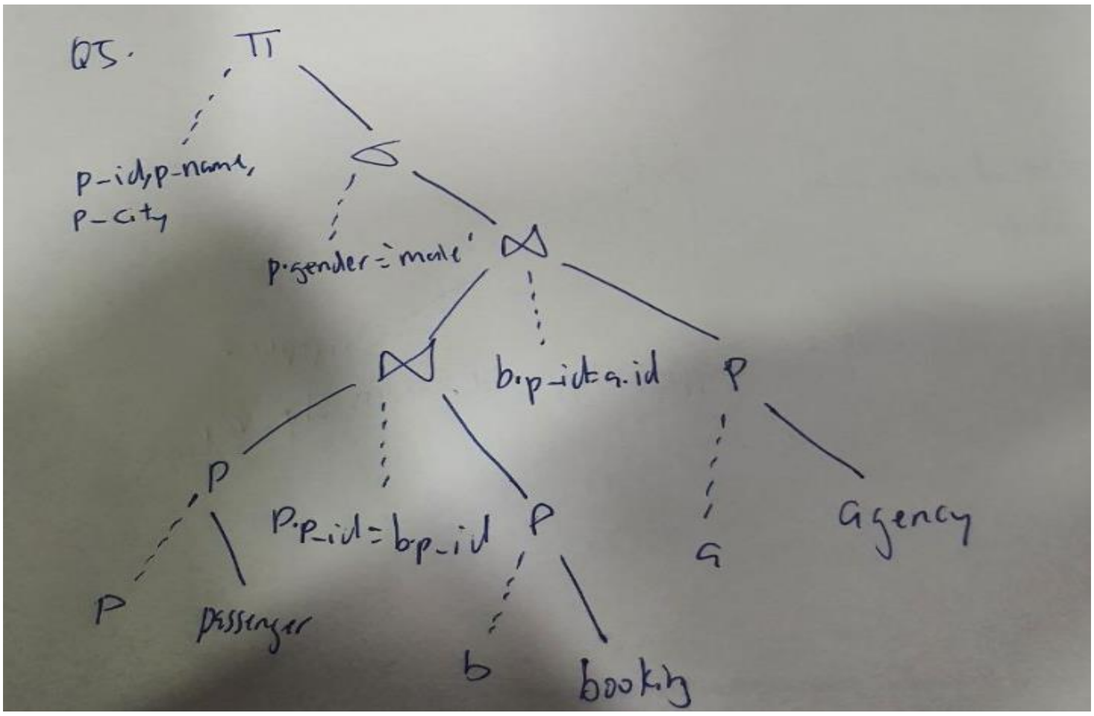
"*List details of all male passengers who are booked through operator*"

A:

Relational Algebra Expressions:

π(p_id, p_name, p_city)(***σ***p_gender='male'(PASSENGER ⋈ BOOKING ⋈ AGENCY))

Query Tree:



Q6. Relational algebra is a _______________________ Data Manipulation Language (DML).

a. **Declarative**

b. **Object-oriented**

c. **Procedural**

d. **Static**

e. **None of the above**

Q7. Assuming that we have two relations R_1 and R_2 where R_1 has 10 tuples (records) and R_2 5, how many records will there be in the result of the operation R_1 x R_2 (cartesian product)?

a. 10 Records

b. 5 Records

c. 15 Records

d. 50 Records

e. None

Q8. For relation *Student(up, name, school)*, suppose school can be one of {soc, smp, sces, smde, seee, dlw, icg}.

If the number of tuples in relation to Student is 1000, then what is the size estimate of the query

$\sigma_{\text{school = '}\textit{physics}\text{'}}$**(Student)?**

A:

0 tuples because Physics is not one of the schools. Hence, no students belong to that school.

Conclusion/Reflection

In this lab, I honed my skills in writing SQL queries and their equivalent relational algebra expressions, enhancing my database operation capabilities. Additionally, it provided insights into query optimization and size estimation. This practical application of theoretical concepts has improved my data retrieval and problem-solving skills in database management.

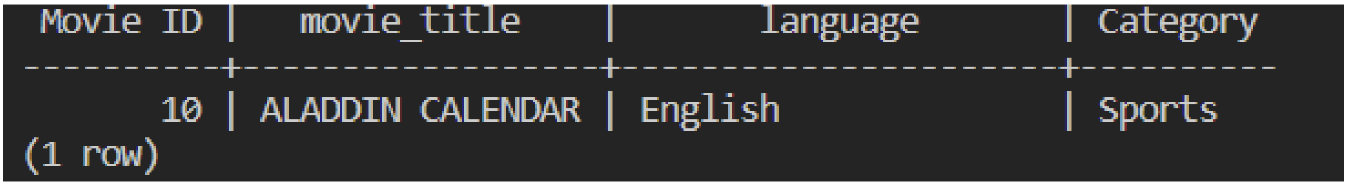
LAB 5 (20/07/2023) - SQL, Query Optimization

Lab Description

Lab 5 focused on practical SQL concepts, with the installation of a new database, Movie Rental, serving as a primary resource. This lab involved executing SQL queries on this database, performing operations like counting views, querying movie titles, and generating reports. Furthermore, we dove into the concept of query optimization, considering the performance of different queries. Lastly, hands-on tasks such as identifying and correcting SQL syntax errors were also included.

Q1. The below SQL statement has five syntax errors. Can you identify them? What would be the correct statement to return *exactly the same result* as in Fig1? Write the correct SQL statement.

```
SELECT
  movie_id AS "Movie ID",
  movie_title AS Movie_Title
  movie_lang AS 'LANGUAGE',
  cat_name AS "Category"
FROM
  movie
  INNER JOIN category LINK movie.movie_id = category.movie_id
WHERE
  movie_title = ALADDIN CALENDAR
```



```
 Movie ID |    movie_title     |         language          | Category
----------+--------------------+---------------------------+----------
       10 | ALADDIN CALENDAR | English                   | Sports
(1 row)
```

Fig1.

A:

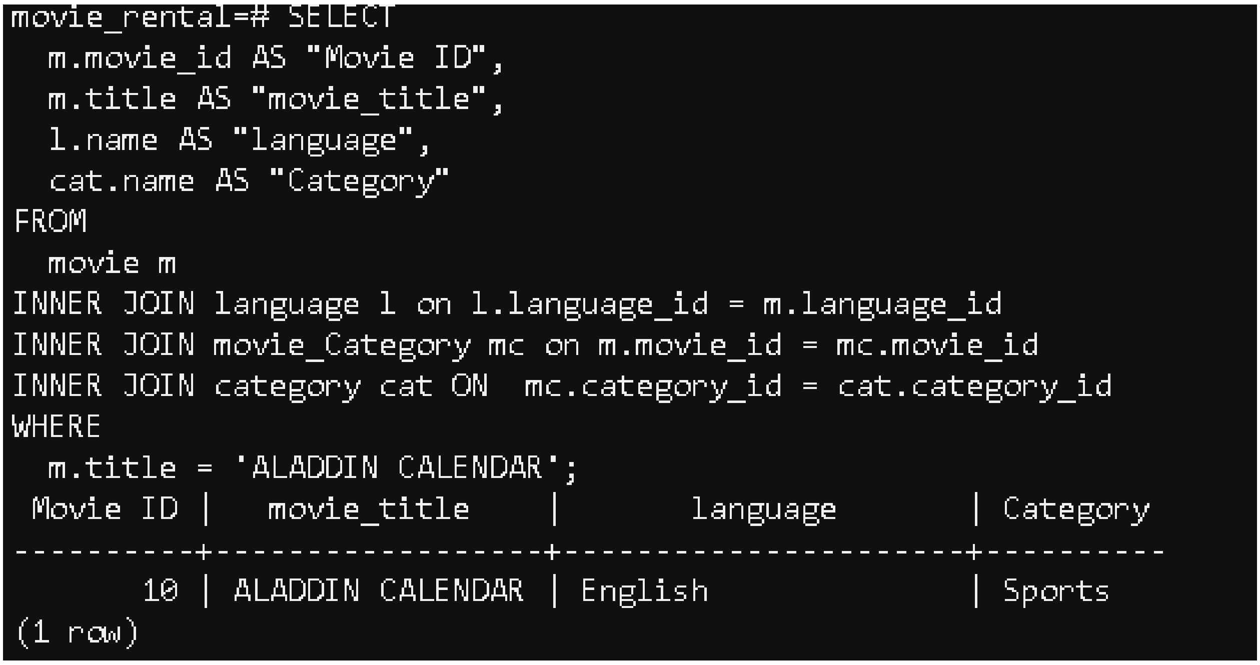
Syntax errors:

1. **Missing commas between column aliases:**
   SQL requires commas to separate each column in the SELECT clause. In your query, you missed a comma between movie_title AS Movie_Title and movie_lang AS 'LANGUAGE'.
2. **Incorrect use of 'LINK' instead of 'ON' in the INNER JOIN syntax:**
   The correct keyword to use when specifying the join condition is 'ON', not 'LINK'. So, your query should have INNER JOIN category ON movie.movie_id = category.movie_id instead of INNER JOIN category LINK movie.movie_id = category.movie_id.
3. **Incorrect column name movie_lang**:
   There is no movie_lang column in the movie table, this will result in a syntax error. You need to replace movie_lang with the correct column name (name) from your language table that stores the language information.
4. **Missing single quotes for the string in the WHERE clause**:
   In SQL, string values need to be enclosed in single quotes. So, you should have movie_title = 'ALADDIN CALENDAR' instead of movie_title = ALADDIN CALENDAR.
5. **Missing semicolon at the end of the statement**:
   SQL statements should end with a semicolon. So, you should have **movie_title = 'ALADDIN CALENDAR';** instead of **movie_title = ALADDIN CALENDAR**.

[EDITABLE CODE]

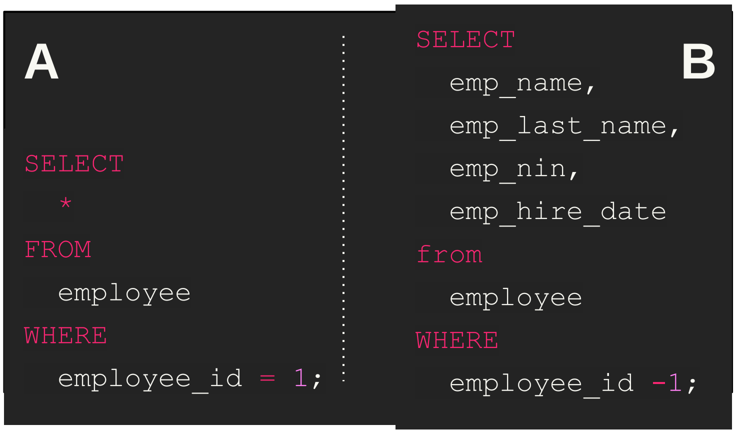
```
SELECT
  m.movie_id AS "Movie ID",
  m.title AS "movie_title",
  l.name AS "language",
  cat.name AS "Category"
FROM
  movie m
INNER JOIN language l on l.language_id = m.language_id
INNER JOIN movie_Category mc on m.movie_id = mc.movie_id
INNER JOIN category cat ON  mc.category_id = cat.category_id
WHERE
  m.title = 'ALADDIN CALENDAR';
```

[OUTPUT SCREENSHOT]



```
movie_rental=# SELECT
  m.movie_id AS "Movie ID",
  m.title AS "movie_title",
  l.name AS "language",
  cat.name AS "Category"
FROM
  movie m
INNER JOIN language l on l.language_id = m.language_id
INNER JOIN movie_Category mc on m.movie_id = mc.movie_id
INNER JOIN category cat ON  mc.category_id = cat.category_id
WHERE
  m.title = 'ALADDIN CALENDAR';
 Movie ID |   movie_title    |       language        | Category
----------+------------------+-----------------------+----------
       10 | ALADDIN CALENDAR | English               | Sports
(1 row)
```

Q2. You are given the following two queries. The table employees have numerous attributes and your department needs the name, NIN and hire date of an employee. Which one will have a better performance and why?



```
SELECT
  *
FROM
  employee
WHERE
  employee_id = 1;
```

```
SELECT
  emp_name,
  emp_last_name,
  emp_nin,
  emp_hire_date
from
  employee
WHERE
  employee_id -1;
```

A:

Query B performs better because it only retrieves specific columns (name, last_name, nim, hire_date), while Query A retrieves all columns with SELECT *. Fetching less data results in faster performance.

From this point forward we will install and use a new database (Movie Rental). This database will be your main DB for the rest of the module hence it is mandatory to install it for this and future labs.

Please follow these instructions on how to install the database. The full ERD (or you can find it on Moodle) and an Interactive Data Dictionary.

DO NOT COPY/PASTE THE SQL STATEMENT AS WILL TAKE OVER 2 HOURS TO INSTALL IT (... yes I have counted). USE SERVER SIDE COMMAND LINE(s) TO INSTALL IN 20-30 SECONDS.

Q3. Using DB specific commands, How many ***VIEWS*** are inside of the movie_rental database?

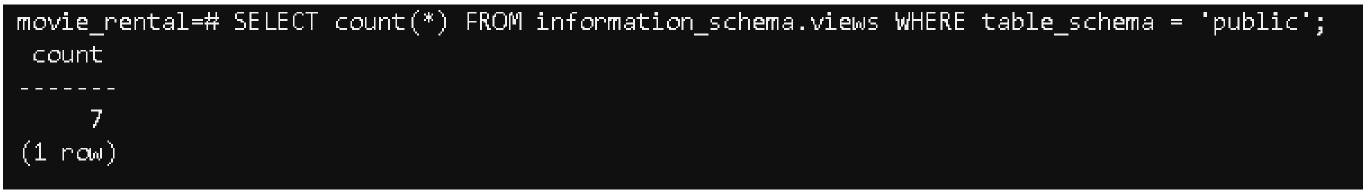
A:

7 VIEWS

[EDITABLE CODE]

```
SELECT count(*) FROM information_schema.views WHERE table_schema = 'public';
```

[OUTPUT SCREENSHOT]



```
movie_rental=# SELECT count(*) FROM information_schema.views WHERE table_schema = 'public';
 count
-------
     7
(1 row)
```

Q4. Using the provided ERD and Data Dictionary, create a query that will return all the movie titles that are in the ***Documentary*** category. Create 2 versions of the same query and in one use category as string (‘**documentary’)** and in the other use category ID (you will need to check what is the category ID for **Documentary**). Use the **EXPLAIN ANALYZE** command and see which query performs better? Why? Note: More information about analysing queries you can find here.

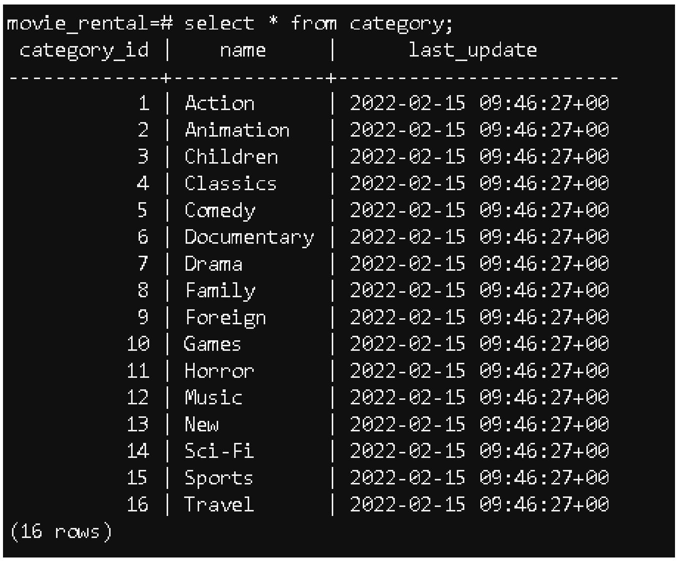
A:

[EDITABLE CODE]

```
 --Version 1: Using category as a string ('documentary'):
EXPLAIN ANALYZE
SELECT m.title as "Movie Title", cat.name as "Category"
FROM movie m
INNER JOIN movie_Category mc on m.movie_id = mc.movie_id
INNER JOIN category cat ON  mc.category_id = cat.category_id
WHERE cat.name = 'Documentary';
--Version 2: Using category ID ('Documentary' has category_id = 6):
EXPLAIN ANALYZE
```

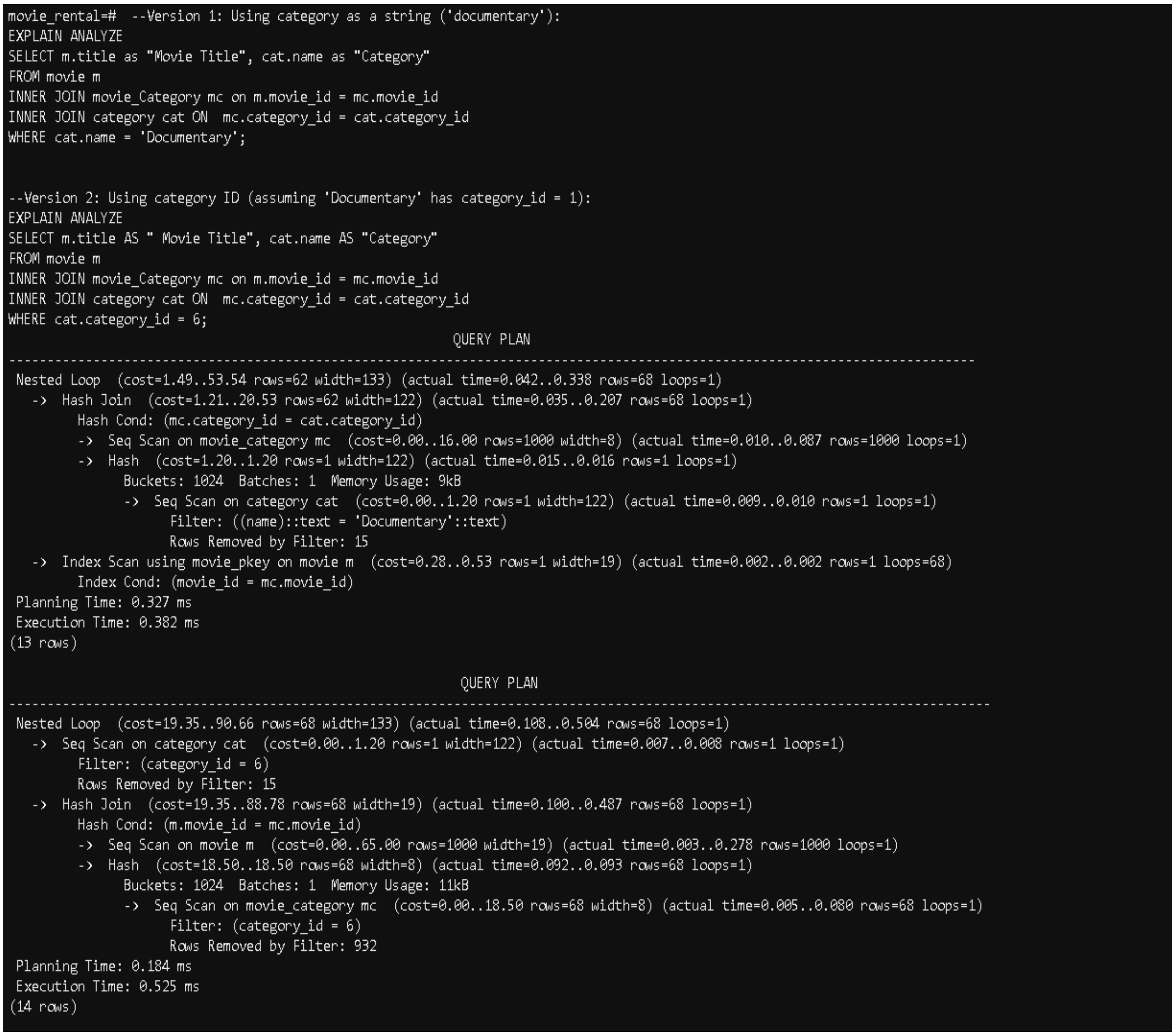
```
SELECT m.title AS " Movie Title", cat.name AS "Category"
FROM movie m
INNER JOIN movie_Category mc on m.movie_id = mc.movie_id
INNER JOIN category cat ON  mc.category_id = cat.category_id
WHERE cat.category_id = 6;
```

[OUTPUT SCREENSHOT]



```
movie_rental=# select * from category;
 category_id |    name     |      last_update
-------------+-------------+------------------------
           1 | Action      | 2022-02-15 09:46:27+00
           2 | Animation   | 2022-02-15 09:46:27+00
           3 | Children    | 2022-02-15 09:46:27+00
           4 | Classics    | 2022-02-15 09:46:27+00
           5 | Comedy      | 2022-02-15 09:46:27+00
           6 | Documentary | 2022-02-15 09:46:27+00
           7 | Drama       | 2022-02-15 09:46:27+00
           8 | Family      | 2022-02-15 09:46:27+00
           9 | Foreign     | 2022-02-15 09:46:27+00
          10 | Games       | 2022-02-15 09:46:27+00
          11 | Horror      | 2022-02-15 09:46:27+00
          12 | Music       | 2022-02-15 09:46:27+00
          13 | New         | 2022-02-15 09:46:27+00
          14 | Sci-Fi      | 2022-02-15 09:46:27+00
          15 | Sports      | 2022-02-15 09:46:27+00
          16 | Travel      | 2022-02-15 09:46:27+00
(16 rows)
```

(category_id = 6 (Documentary))



```
movie_rental=#  --Version 1: Using category as a string ('documentary'):
EXPLAIN ANALYZE
SELECT m.title as "Movie Title", cat.name as "Category"
FROM movie m
INNER JOIN movie_Category mc on m.movie_id = mc.movie_id
INNER JOIN category cat ON  mc.category_id = cat.category_id
WHERE cat.name = 'Documentary';


--Version 2: Using category ID (assuming 'Documentary' has category_id = 1):
EXPLAIN ANALYZE
SELECT m.title AS " Movie Title", cat.name AS "Category"
FROM movie m
INNER JOIN movie_Category mc on m.movie_id = mc.movie_id
INNER JOIN category cat ON  mc.category_id = cat.category_id
WHERE cat.category_id = 6;
                                                        QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=1.49..53.54 rows=62 width=133) (actual time=0.042..0.338 rows=68 loops=1)
   ->  Hash Join  (cost=1.21..20.53 rows=62 width=122) (actual time=0.035..0.207 rows=68 loops=1)
         Hash Cond: (mc.category_id = cat.category_id)
         ->  Seq Scan on movie_category mc  (cost=0.00..16.00 rows=1000 width=8) (actual time=0.010..0.087 rows=1000 loops=1)
         ->  Hash  (cost=1.20..1.20 rows=1 width=122) (actual time=0.015..0.016 rows=1 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on category cat  (cost=0.00..1.20 rows=1 width=122) (actual time=0.009..0.010 rows=1 loops=1)
                     Filter: ((name)::text = 'Documentary'::text)
                     Rows Removed by Filter: 15
   ->  Index Scan using movie_pkey on movie m  (cost=0.28..0.53 rows=1 width=19) (actual time=0.002..0.002 rows=1 loops=68)
         Index Cond: (movie_id = mc.movie_id)
 Planning Time: 0.327 ms
 Execution Time: 0.382 ms
(13 rows)

                                                         QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=19.35..90.66 rows=68 width=133) (actual time=0.108..0.504 rows=68 loops=1)
   ->  Seq Scan on category cat  (cost=0.00..1.20 rows=1 width=122) (actual time=0.007..0.008 rows=1 loops=1)
         Filter: (category_id = 6)
         Rows Removed by Filter: 15
   ->  Hash Join  (cost=19.35..88.78 rows=68 width=19) (actual time=0.100..0.487 rows=68 loops=1)
         Hash Cond: (m.movie_id = mc.movie_id)
         ->  Seq Scan on movie m  (cost=0.00..65.00 rows=1000 width=19) (actual time=0.003..0.278 rows=1000 loops=1)
         ->  Hash  (cost=18.50..18.50 rows=68 width=8) (actual time=0.092..0.093 rows=68 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 11kB
               ->  Seq Scan on movie_category mc  (cost=0.00..18.50 rows=68 width=8) (actual time=0.005..0.080 rows=68 loops=1)
                     Filter: (category_id = 6)
                     Rows Removed by Filter: 932
 Planning Time: 0.184 ms
 Execution Time: 0.525 ms
(14 rows)
```

Version 1 perform better and the reasons on why as follows:

1. **Indexing**: If an index is present on the name column in the category table, the database can quickly look up rows with 'Documentary' without scanning the whole table.

2. **Data Distribution**: If there are very few categories with the name 'Documentary' compared to categories with the ID '6', the first query could be faster.

3. **Query Planning and Optimization**: PostgreSQL's query planner may have chosen different strategies for executing the two queries based on table statistics, leading to different performance characteristics.

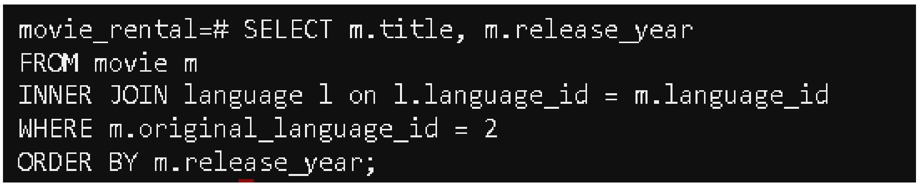
Q5. Using your SQL knowledge create a query that will list all movies that have original language Italian along with the release year. The movies should be listed by release year in chronological order.

A:

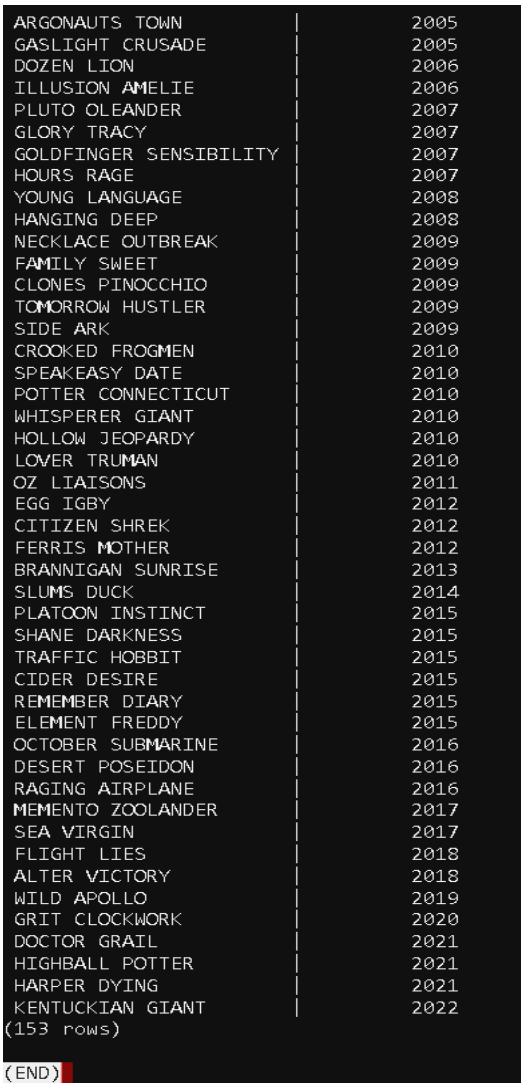
[EDITABLE CODE]

```
SELECT m.title, m.release_year
FROM movie m
INNER JOIN language l on l.language_id = m.language_id
WHERE m.original_language_id = 2
ORDER BY m.release_year;
```

[OUTPUT SCREENSHOT]



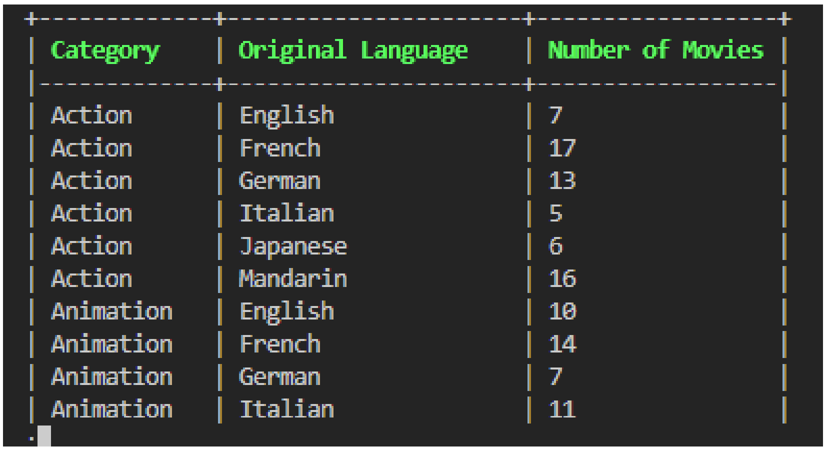
```
movie_rental=# SELECT m.title, m.release_year
FROM movie m
INNER JOIN language l on l.language_id = m.language_id
WHERE m.original_language_id = 2
ORDER BY m.release_year;
```



```
 ARGONAUTS TOWN                |          2005
 GASLIGHT CRUSADE              |          2005
 DOZEN LION                    |          2006
 ILLUSION AMELIE               |          2006
 PLUTO OLEANDER                |          2007
 GLORY TRACY                   |          2007
 GOLDFINGER SENSIBILITY        |          2007
 HOURS RAGE                    |          2007
 YOUNG LANGUAGE                |          2008
 HANGING DEEP                  |          2008
 NECKLACE OUTBREAK             |          2009
 FAMILY SWEET                  |          2009
 CLONES PINOCCHIO              |          2009
 TOMORROW HUSTLER              |          2009
 SIDE ARK                      |          2009
 CROOKED FROGMEN               |          2010
 SPEAKEASY DATE                |          2010
 POTTER CONNECTICUT            |          2010
 WHISPERER GIANT               |          2010
 HOLLOW JEOPARDY               |          2010
 LOVER TRUMAN                  |          2010
 OZ LIAISONS                   |          2011
 EGG IGBY                      |          2012
 CITIZEN SHREK                 |          2012
 FERRIS MOTHER                 |          2012
 BRANNIGAN SUNRISE             |          2013
 SLUMS DUCK                    |          2014
 PLATOON INSTINCT              |          2015
 SHANE DARKNESS                |          2015
 TRAFFIC HOBBIT                |          2015
 CIDER DESIRE                  |          2015
 REMEMBER DIARY                |          2015
 ELEMENT FREDDY                |          2015
 OCTOBER SUBMARINE             |          2016
 DESERT POSEIDON               |          2016
 RAGING AIRPLANE               |          2016
 MEMENTO ZOOLANDER             |          2017
 SEA VIRGIN                    |          2017
 FLIGHT LIES                   |          2018
 ALTER VICTORY                 |          2018
 WILD APOLLO                   |          2019
 GRIT CLOCKWORK                |          2020
 DOCTOR GRAIL                  |          2021
 HIGHBALL POTTER               |          2021
 HARPER DYING                  |          2021
 KENTUCKIAN GIANT              |          2022
(153 rows)

(END)
```

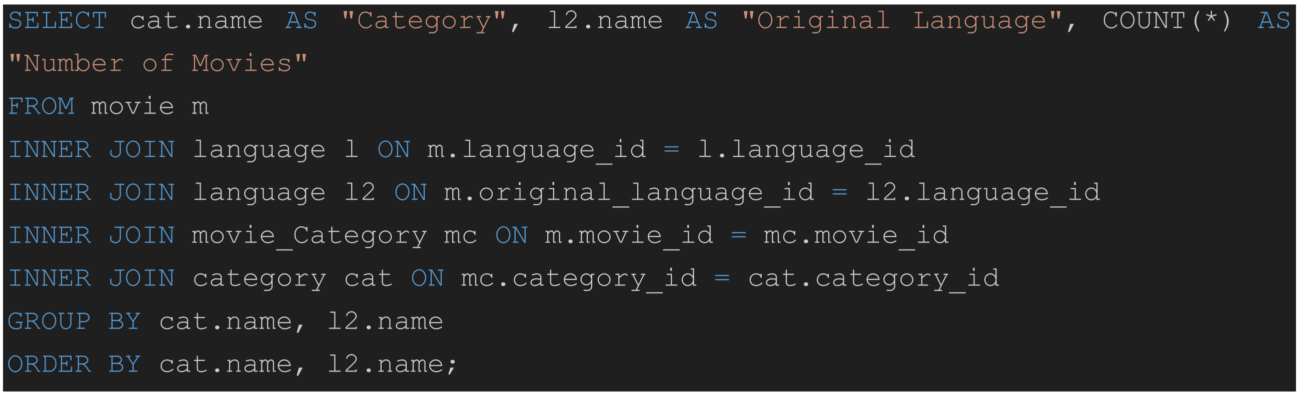
Q6. List how many movies are in each category per original language. The output should be similar to the one below.



```
+-------------+----------------------+------------------+
| Category    | Original Language    | Number of Movies |
|-------------+----------------------+------------------|
| Action      | English              | 7                |
| Action      | French               | 17               |
| Action      | German               | 13               |
| Action      | Italian              | 5                |
| Action      | Japanese             | 6                |
| Action      | Mandarin             | 16               |
| Animation   | English              | 10               |
| Animation   | French               | 14               |
| Animation   | German               | 7                |
| Animation   | Italian              | 11               |
:
```

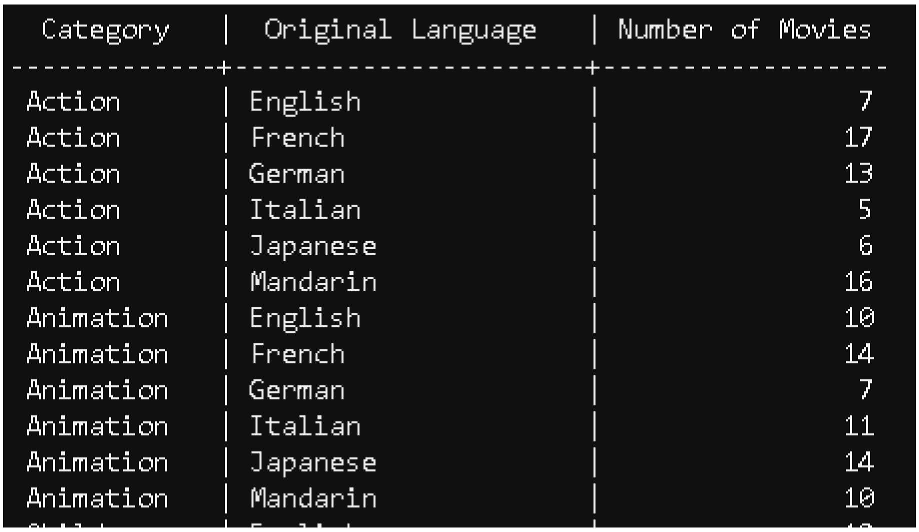
A:

[EDITABLE SCREENSHOT]



```
SELECT cat.name AS "Category", l2.name AS "Original Language", COUNT(*) AS
"Number of Movies"
FROM movie m
INNER JOIN language l ON m.language_id = l.language_id
INNER JOIN language l2 ON m.original_language_id = l2.language_id
INNER JOIN movie_Category mc ON m.movie_id = mc.movie_id
INNER JOIN category cat ON mc.category_id = cat.category_id
GROUP BY cat.name, l2.name
ORDER BY cat.name, l2.name;
```

[OUTPUT SCREENSHOT]



```
  Category   |  Original Language   | Number of Movies
-------------+----------------------+------------------
 Action      | English              |                7
 Action      | French               |               17
 Action      | German               |               13
 Action      | Italian              |                5
 Action      | Japanese             |                6
 Action      | Mandarin             |               16
 Animation   | English              |               10
 Animation   | French               |               14
 Animation   | German               |                7
 Animation   | Italian              |               11
 Animation   | Japanese             |               14
 Animation   | Mandarin             |               10
```

Conclusion/Reflection

Through Lab 5, I deepened my understanding of SQL and its real-world applications. Working directly with the Movie Rental database, I executed and analyzed queries, which sharpened my comprehension of SQL syntax and its nuances. I was able to distinguish between the performance of different queries based on how they interact with the database's indexes, and the concept of sargability became clearer. In addition, by identifying and correcting SQL syntax errors, I gained more confidence in formulating my own SQL queries. Overall, Lab 5 provided valuable insights into how databases are managed and manipulated in real-world scenarios.

LAB 6 (21/07/2023) - Optimization

Lab Description

In Lab 6, advanced SQL abilities through the "movie_rental" database, focusing on data manipulation, query optimization, and index application for enhanced query performance. Key tasks included updating specific records, creating and dropping indexes, and creating a VIEW to simplify data retrieval.

Q1. Connect to the movie_rental database (that you have installed in LAB5) and change the phone number from the current one to *02392844444* for address with id 100. Do not alter any other data/attribute.

A:

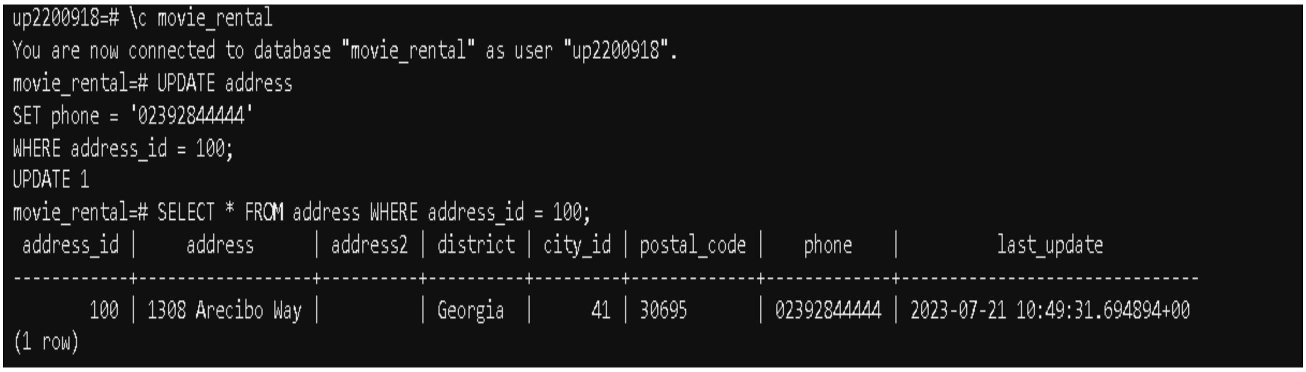
[EDITABLE CODE]

```
-- Connect to ‘movie_rental’ database
\c movie_rental

-- Change address with id   100
UPDATE address
SET phone = '02392844444'
WHERE address_id = 100;

-- View updated address with id 100
SELECT * FROM address WHERE address_id = 100;
```

[OUTPUT SCREENSHOT]



```
up2200918=# \c movie_rental
You are now connected to database "movie_rental" as user "up2200918".
movie_rental=# UPDATE address
SET phone = '02392844444'
WHERE address_id = 100;
UPDATE 1
movie_rental=# SELECT * FROM address WHERE address_id = 100;
 address_id |     address      | address2 | district | city_id | postal_code |    phone    |          last_update
------------+------------------+----------+----------+---------+-------------+-------------+-------------------------------
        100 | 1308 Arecibo Way |          | Georgia  |      41 | 30695       | 02392844444 | 2023-07-21 10:49:31.694894+00
(1 row)
```

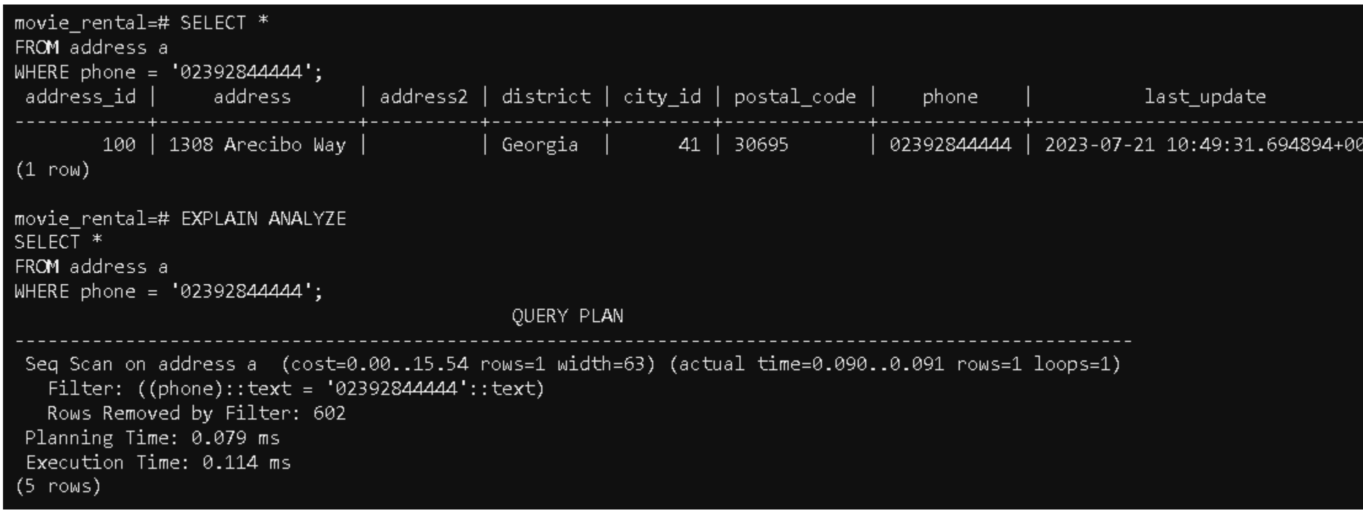
Q2. Create a query that will return all the details for the record where the phone number is *02392844444* *and use EXPLAIN. Take a screenshot of the output.*

A:

[EDITABLE CODE]

```
EXPLAIN ANALYZE
SELECT *
FROM address a
WHERE phone = '02392844444';
```

[OUTPUT SCREENSHOT]



```
movie_rental=# SELECT *
FROM address a
WHERE phone = '02392844444';
 address_id |      address       | address2 | district | city_id | postal_code |    phone     |           last_update
------------+--------------------+----------+----------+---------+-------------+--------------+--------------------------------
        100 | 1308 Arecibo Way |          | Georgia  |      41 | 30695       | 02392844444 | 2023-07-21 10:49:31.694894+00
(1 row)

movie_rental=# EXPLAIN ANALYZE
SELECT *
FROM address a
WHERE phone = '02392844444';
                                              QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Seq Scan on address a  (cost=0.00..15.54 rows=1 width=63) (actual time=0.090..0.091 rows=1 loops=1)
   Filter: ((phone)::text = '02392844444'::text)
   Rows Removed by Filter: 602
 Planning Time: 0.079 ms
 Execution Time: 0.114 ms
(5 rows)
```

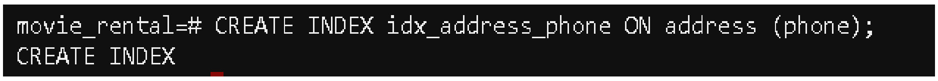
Q3. Apply an INDEX on the phone attribute.

A:

[EDITABLE CODE]

```
CREATE INDEX idx_address_phone ON address (phone);
```

[OUTPUT SCREENSHOT]



```
movie_rental=# CREATE INDEX idx_address_phone ON address (phone);
CREATE INDEX
```

Q4. Use EXPLAIN again. What are the differences? Which one performs better and why?

A:

- Without Index: In this case, the query may involve a "Sequential Scan" of the table, which means the database engine will go through all rows one by one to find the matching phone number. This could be less efficient, especially for large tables, as it requires reading the entire table.

- With Index: After applying the index, the query should utilize an "Index Scan." The index allows the database engine to quickly locate the rows with the specified phone number, resulting in a much faster execution time, particularly for tables with a large number of records.

- The **query with the index should perform significantly better** than the query without the index, especially when dealing with large datasets. The index provides an efficient way to narrow down the search and directly access the required rows, leading to improved performance in searching for the phone number '02392844444'.

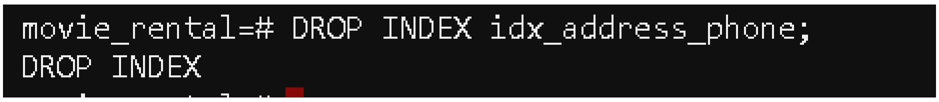
Q5. Delete the INDEX you have created in Q3.

A:

[EDITABLE CODE]

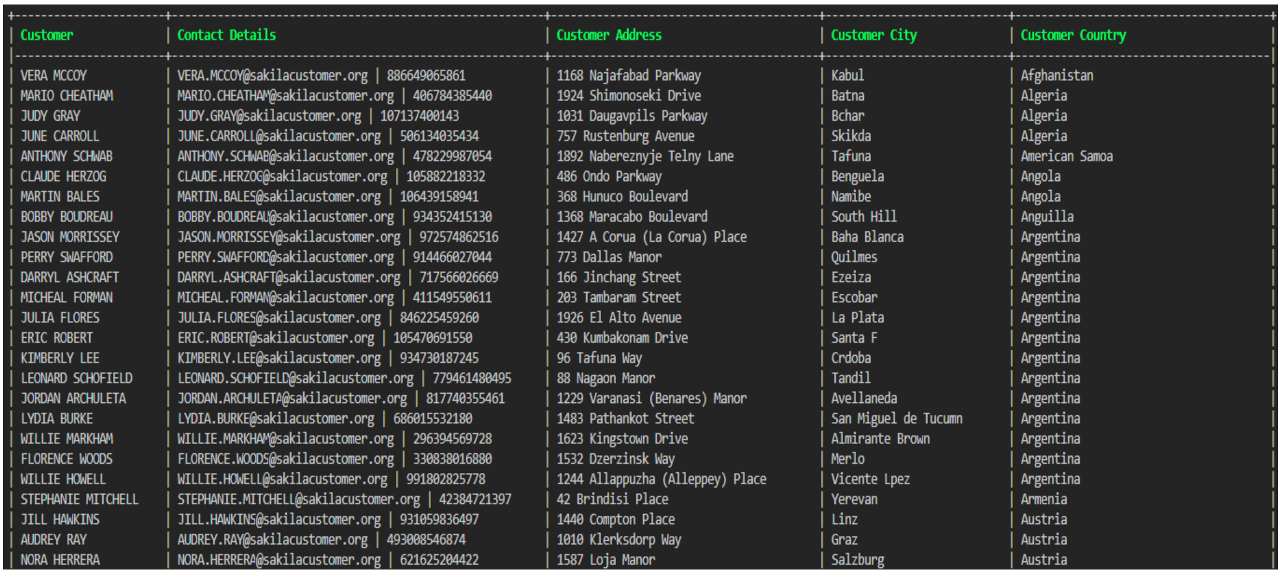
```
DROP INDEX idx_address_phone;
```

[OUTPUT SCREENSHOT]



```
movie_rental=# DROP INDEX idx_address_phone;
DROP INDEX
```

Q6. Create a VIEW (*customer_details*) that will hold the customers details exactly as below.



```
+--------------------+------------------------------------------------+----------------------------------+-------------------------+-------------------------------+
| Customer           | Contact Details                                | Customer Address                 | Customer City           | Customer Country              |
|--------------------+------------------------------------------------+----------------------------------+-------------------------+-------------------------------|
| VERA MCCOY         | VERA.MCCOY@sakilacustomer.org | 886649065861   | 1168 Najafabad Parkway           | Kabul                   | Afghanistan                   |
| MARIO CHEATHAM     | MARIO.CHEATHAM@sakilacustomer.org | 406784385440 | 1924 Shimonoseki Drive          | Batna                   | Algeria                       |
| JUDY GRAY          | JUDY.GRAY@sakilacustomer.org | 107137400143    | 1031 Daugavpils Parkway          | Bchar                   | Algeria                       |
| JUNE CARROLL       | JUNE.CARROLL@sakilacustomer.org | 506134035434 | 757 Rustenburg Avenue            | Skikda                  | Algeria                       |
| ANTHONY SCHWAB     | ANTHONY.SCHWAB@sakilacustomer.org | 478229987054 | 1892 Nabereznyje Telny Lane     | Tafuna                  | American Samoa                |
| CLAUDE HERZOG      | CLAUDE.HERZOG@sakilacustomer.org | 105882218332 | 486 Ondo Parkway                | Benguela                | Angola                        |
| MARTIN BALES       | MARTIN.BALES@sakilacustomer.org | 106439158941 | 368 Hunuco Boulevard             | Namibe                  | Angola                        |
| BOBBY BOUDREAU     | BOBBY.BOUDREAU@sakilacustomer.org | 934352415130 | 1368 Maracabo Boulevard         | South Hill              | Anguilla                      |
| JASON MORRISSEY    | JASON.MORRISSEY@sakilacustomer.org | 972574862516 | 1427 A Corua (La Corua) Place  | Baha Blanca             | Argentina                     |
| PERRY SWAFFORD     | PERRY.SWAFFORD@sakilacustomer.org | 914466027044 | 773 Dallas Manor                | Quilmes                 | Argentina                     |
| DARRYL ASHCRAFT    | DARRYL.ASHCRAFT@sakilacustomer.org | 717566026669 | 166 Jinchang Street            | Ezeiza                  | Argentina                     |
| MICHEAL FORMAN     | MICHEAL.FORMAN@sakilacustomer.org | 411549550611 | 203 Tambaram Street             | Escobar                 | Argentina                     |
| JULIA FLORES       | JULIA.FLORES@sakilacustomer.org | 846225459260 | 1926 El Alto Avenue              | La Plata                | Argentina                     |
| ERIC ROBERT        | ERIC.ROBERT@sakilacustomer.org | 105470691550  | 430 Kumbakonam Drive             | Santa F                 | Argentina                     |
| KIMBERLY LEE       | KIMBERLY.LEE@sakilacustomer.org | 934730187245 | 96 Tafuna Way                    | Crdoba                  | Argentina                     |
| LEONARD SCHOFIELD  | LEONARD.SCHOFIELD@sakilacustomer.org | 779461480495 | 88 Nagaon Manor              | Tandil                  | Argentina                     |
| JORDAN ARCHULETA   | JORDAN.ARCHULETA@sakilacustomer.org | 817740355461 | 1229 Varanasi (Benares) Manor | Avellaneda              | Argentina                     |
| LYDIA BURKE        | LYDIA.BURKE@sakilacustomer.org | 686015532180  | 1483 Pathankot Street            | San Miguel de Tucumn    | Argentina                     |
| WILLIE MARKHAM     | WILLIE.MARKHAM@sakilacustomer.org | 296394569728 | 1623 Kingstown Drive            | Almirante Brown         | Argentina                     |
| FLORENCE WOODS     | FLORENCE.WOODS@sakilacustomer.org | 330838016880 | 1532 Dzerzinsk Way              | Merlo                   | Argentina                     |
| WILLIE HOWELL      | WILLIE.HOWELL@sakilacustomer.org | 991802825778 | 1244 Allappuzha (Alleppey) Place | Vicente Lpez            | Argentina                     |
| STEPHANIE MITCHELL | STEPHANIE.MITCHELL@sakilacustomer.org | 42384721397 | 42 Brindisi Place            | Yerevan                 | Armenia                       |
| JILL HAWKINS       | JILL.HAWKINS@sakilacustomer.org | 931059836497 | 1440 Compton Place               | Linz                    | Austria                       |
| AUDREY RAY         | AUDREY.RAY@sakilacustomer.org | 493008546874   | 1010 Klerksdorp Way              | Graz                    | Austria                       |
| NORA HERRERA       | NORA.HERRERA@sakilacustomer.org | 621625204422 | 1587 Loja Manor                  | Salzburg                | Austria                       |
```

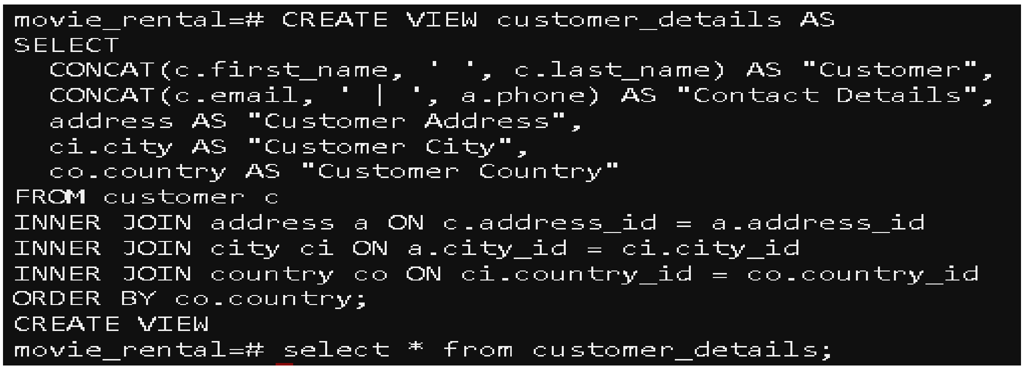
A:

[EDITABLE CODE]

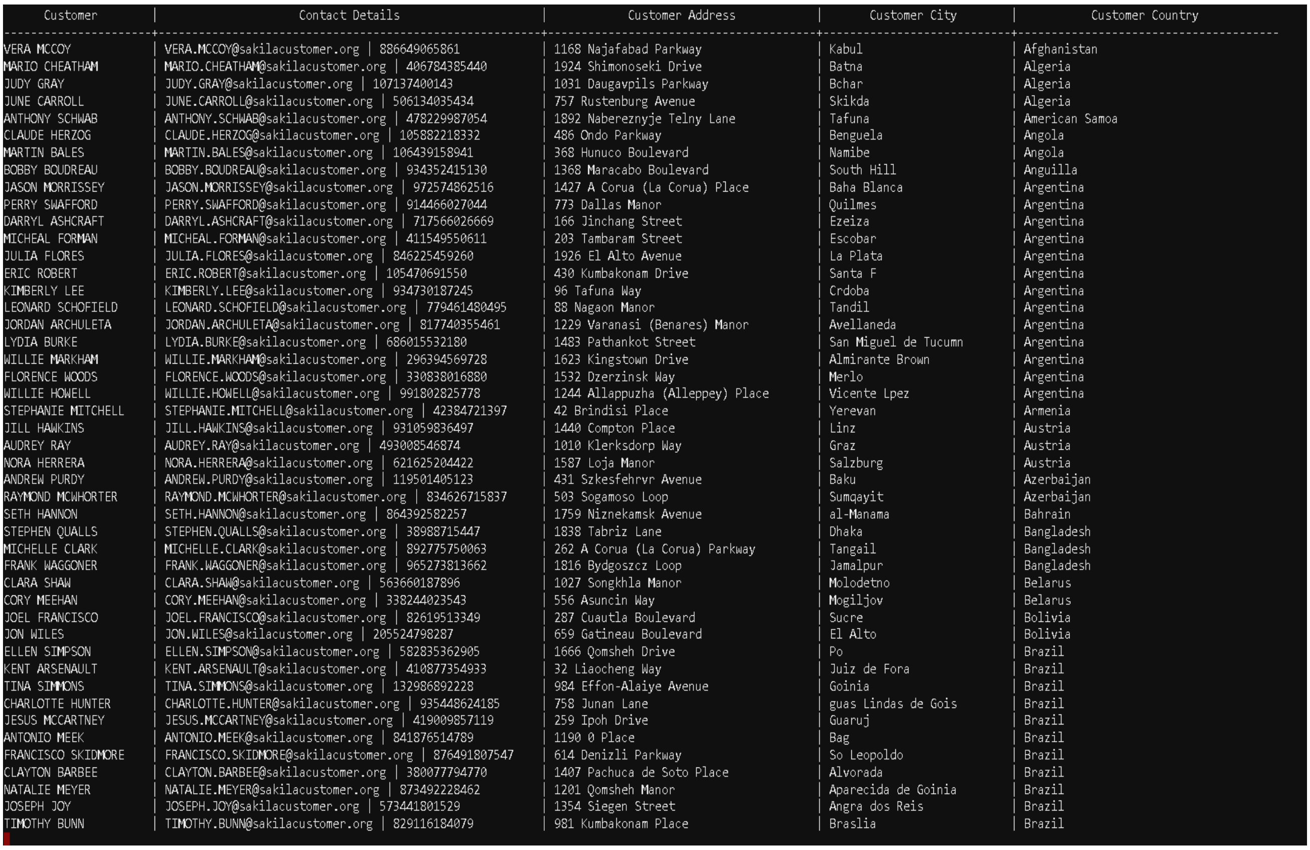
```
CREATE VIEW customer_details AS
```

```
SELECT
  CONCAT(c.first_name, ' ', c.last_name) AS "Customer",
  CONCAT(c.email, ' | ', a.phone) AS "Contact Details",
  address AS "Customer Address",
  ci.city AS "Customer City",
  co.country AS "Customer Country"
FROM customer c
INNER JOIN address a ON c.address_id = a.address_id
INNER JOIN city ci ON a.city_id = ci.city_id
INNER JOIN country co ON ci.country_id = co.country_id
ORDER BY co.country;
```

[OUTPUT SCREENSHOT]



```
movie_rental=# CREATE VIEW customer_details AS
SELECT
   CONCAT(c.first_name, ' ', c.last_name) AS "Customer",
   CONCAT(c.email, ' | ', a.phone) AS "Contact Details",
   address AS "Customer Address",
   ci.city AS "Customer City",
   co.country AS "Customer Country"
FROM customer c
INNER JOIN address a ON c.address_id = a.address_id
INNER JOIN city ci ON a.city_id = ci.city_id
INNER JOIN country co ON ci.country_id = co.country_id
ORDER BY co.country;
CREATE VIEW
movie_rental=# select * from customer_details;
```



```
      Customer       |                 Contact Details                  |         Customer Address          |     Customer City      |     Customer Country
---------------------+--------------------------------------------------+-----------------------------------+------------------------+----------------------
VERA MCCOY           | VERA.MCCOY@sakilacustomer.org | 886649065861     | 1168 Najafabad Parkway            | Kabul                  | Afghanistan
MARIO CHEATHAM       | MARIO.CHEATHAM@sakilacustomer.org | 406784385440 | 1924 Shimonoseki Drive            | Batna                  | Algeria
JUDY GRAY            | JUDY.GRAY@sakilacustomer.org | 107137400143      | 1031 Daugavpils Parkway           | Bchar                  | Algeria
JUNE CARROLL         | JUNE.CARROLL@sakilacustomer.org | 506134035434   | 757 Rustenburg Avenue             | Skikda                 | Algeria
ANTHONY SCHWAB       | ANTHONY.SCHWAB@sakilacustomer.org | 478229987054 | 1892 Nabereznyje Telny Lane       | Tafuna                 | American Samoa
CLAUDE HERZOG        | CLAUDE.HERZOG@sakilacustomer.org | 105882218332  | 486 Ondo Parkway                  | Benguela               | Angola
MARTIN BALES         | MARTIN.BALES@sakilacustomer.org | 106439158941   | 368 Hunuco Boulevard              | Namibe                 | Angola
BOBBY BOUDREAU       | BOBBY.BOUDREAU@sakilacustomer.org | 934352415130 | 1368 Maracabo Boulevard           | South Hill             | Anguilla
JASON MORRISSEY      | JASON.MORRISSEY@sakilacustomer.org | 972574862516 | 1427 A Corua (La Corua) Place    | Baha Blanca            | Argentina
PERRY SWAFFORD       | PERRY.SWAFFORD@sakilacustomer.org | 914466027044 | 773 Dallas Manor                  | Quilmes                | Argentina
DARRYL ASHCRAFT      | DARRYL.ASHCRAFT@sakilacustomer.org | 717566026669 | 166 Jinchang Street              | Ezeiza                 | Argentina
MICHEAL FORMAN       | MICHEAL.FORMAN@sakilacustomer.org | 411549550611 | 203 Tambaram Street               | Escobar                | Argentina
JULIA FLORES         | JULIA.FLORES@sakilacustomer.org | 846225459260   | 1926 El Alto Avenue               | La Plata               | Argentina
ERIC ROBERT          | ERIC.ROBERT@sakilacustomer.org | 105470691550    | 430 Kumbakonam Drive              | Santa F                | Argentina
KIMBERLY LEE         | KIMBERLY.LEE@sakilacustomer.org | 934730187245   | 96 Tafuna Way                     | Crdoba                 | Argentina
LEONARD SCHOFIELD    | LEONARD.SCHOFIELD@sakilacustomer.org | 779461480495 | 88 Nagaon Manor                | Tandil                 | Argentina
JORDAN ARCHULETA     | JORDAN.ARCHULETA@sakilacustomer.org | 817740355461 | 1229 Varanasi (Benares) Manor   | Avellaneda             | Argentina
LYDIA BURKE          | LYDIA.BURKE@sakilacustomer.org | 686015532180    | 1483 Pathankot Street             | San Miguel de Tucumn   | Argentina
WILLIE MARKHAM       | WILLIE.MARKHAM@sakilacustomer.org | 296394569728 | 1623 Kingstown Drive              | Almirante Brown        | Argentina
FLORENCE WOODS       | FLORENCE.WOODS@sakilacustomer.org | 330838016880 | 1532 Dzerzinsk Way                | Merlo                  | Argentina
WILLIE HOWELL        | WILLIE.HOWELL@sakilacustomer.org | 991802825778  | 1244 Allappuzha (Alleppey) Place  | Vicente Lpez           | Argentina
STEPHANIE MITCHELL   | STEPHANIE.MITCHELL@sakilacustomer.org | 42384721397 | 42 Brindisi Place              | Yerevan                | Armenia
JILL HAWKINS         | JILL.HAWKINS@sakilacustomer.org | 931059836497   | 1440 Compton Place                | Linz                   | Austria
AUDREY RAY           | AUDREY.RAY@sakilacustomer.org | 493008546874     | 1010 Klerksdorp Way               | Graz                   | Austria
NORA HERRERA         | NORA.HERRERA@sakilacustomer.org | 621625204422   | 1587 Loja Manor                   | Salzburg               | Austria
ANDREW PURDY         | ANDREW.PURDY@sakilacustomer.org | 119501405123   | 431 Szkesfehrvr Avenue            | Baku                   | Azerbaijan
RAYMOND MCWHORTER    | RAYMOND.MCWHORTER@sakilacustomer.org | 834626715837 | 503 Sogamoso Loop              | Sumqayit               | Azerbaijan
SETH HANNON          | SETH.HANNON@sakilacustomer.org | 864392582257    | 1759 Niznekamsk Avenue            | al-Manama              | Bahrain
STEPHEN QUALLS       | STEPHEN.QUALLS@sakilacustomer.org | 38988715447  | 1838 Tabriz Lane                  | Dhaka                  | Bangladesh
MICHELLE CLARK       | MICHELLE.CLARK@sakilacustomer.org | 892775750063 | 262 A Corua (La Corua) Parkway    | Tangail                | Bangladesh
FRANK WAGGONER       | FRANK.WAGGONER@sakilacustomer.org | 965273813662 | 1816 Bydgoszcz Loop               | Jamalpur               | Bangladesh
CLARA SHAW           | CLARA.SHAW@sakilacustomer.org | 563660187896     | 1027 Songkhla Manor               | Molodetno              | Belarus
CORY MEEHAN          | CORY.MEEHAN@sakilacustomer.org | 338244023543    | 556 Asuncin Way                   | Mogiljov               | Belarus
JOEL FRANCISCO       | JOEL.FRANCISCO@sakilacustomer.org | 82619513349  | 287 Cuautla Boulevard             | Sucre                  | Bolivia
JON WILES            | JON.WILES@sakilacustomer.org | 205524798287      | 659 Gatineau Boulevard            | El Alto                | Bolivia
ELLEN SIMPSON        | ELLEN.SIMPSON@sakilacustomer.org | 582835362905  | 1666 Qomsheh Drive                | Po                     | Brazil
KENT ARSENAULT       | KENT.ARSENAULT@sakilacustomer.org | 410877354933 | 32 Liaocheng Way                  | Juiz de Fora           | Brazil
TINA SIMMONS         | TINA.SIMMONS@sakilacustomer.org | 132986892228   | 984 Effon-Alaiye Avenue           | Goinia                 | Brazil
CHARLOTTE HUNTER     | CHARLOTTE.HUNTER@sakilacustomer.org | 935448624185 | 758 Junan Lane                  | guas Lindas de Gois    | Brazil
JESUS MCCARTNEY      | JESUS.MCCARTNEY@sakilacustomer.org | 419009857119 | 259 Ipoh Drive                   | Guaruj                 | Brazil
ANTONIO MEEK         | ANTONIO.MEEK@sakilacustomer.org | 841876514789   | 1190 0 Place                      | Bag                    | Brazil
FRANCISCO SKIDMORE   | FRANCISCO.SKIDMORE@sakilacustomer.org | 876491807547 | 614 Denizli Parkway           | So Leopoldo            | Brazil
CLAYTON BARBEE       | CLAYTON.BARBEE@sakilacustomer.org | 380077794770 | 1407 Pachuca de Soto Place        | Alvorada               | Brazil
NATALIE MEYER        | NATALIE.MEYER@sakilacustomer.org | 873492228462  | 1201 Qomsheh Manor                | Aparecida de Goinia    | Brazil
JOSEPH JOY           | JOSEPH.JOY@sakilacustomer.org | 573441801529     | 1354 Siegen Street                | Angra dos Reis         | Brazil
TIMOTHY BUNN         | TIMOTHY.BUNN@sakilacustomer.org | 829116184079   | 981 Kumbakonam Place              | Braslia                | Brazil
```

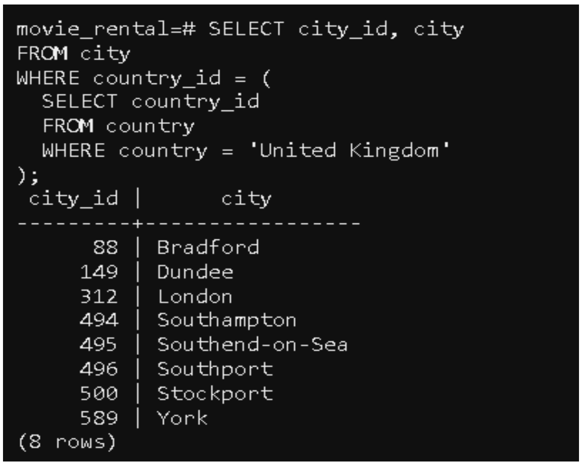
Q7. **Using a subquery list all cities from the United Kingdom along with the city ID.**

A:

[EDITABLE CODE]

```
SELECT city_id, city
FROM city
WHERE country_id = (
  SELECT country_id
  FROM country
  WHERE country = 'United Kingdom'
);
```

[OUTPUT SCREENSHOT]



```
movie_rental=# SELECT city_id, city
FROM city
WHERE country_id = (
  SELECT country_id
  FROM country
  WHERE country = 'United Kingdom'
);
 city_id |       city
---------+-----------------
      88 | Bradford
     149 | Dundee
     312 | London
     494 | Southampton
     495 | Southend-on-Sea
     496 | Southport
     500 | Stockport
     589 | York
(8 rows)
```

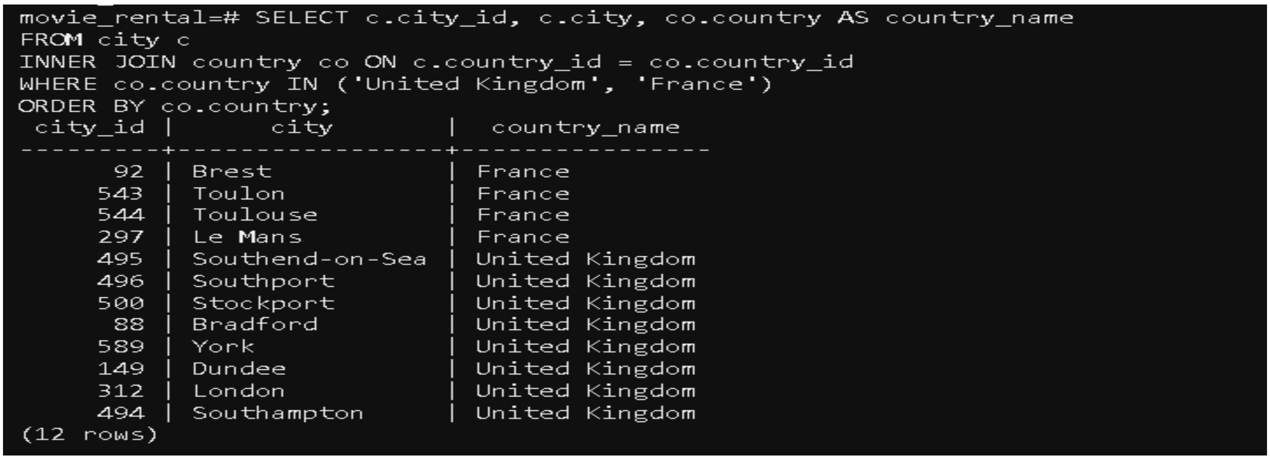
Q8. **As an extension of the previous query, list the cities from the United Kingdom and France using the IN operator, this time showing the country name as well. Group them by country.**

A:

[EDITABLE CODE]

```
SELECT c.city_id, c.city, co.country AS country_name
FROM city c
INNER JOIN country co ON c.country_id = co.country_id
WHERE co.country IN ('United Kingdom', 'France')
ORDER BY co.country;
```

[OUTPUT SCREENSHOT]



```
movie_rental=# SELECT c.city_id, c.city, co.country AS country_name
FROM city c
INNER JOIN country co ON c.country_id = co.country_id
WHERE co.country IN ('United Kingdom', 'France')
ORDER BY co.country;
 city_id |       city        |   country_name
---------+-------------------+----------------
      92 | Brest             | France
     543 | Toulon            | France
     544 | Toulouse          | France
     297 | Le Mans           | France
     495 | Southend-on-Sea   | United Kingdom
     496 | Southport         | United Kingdom
     500 | Stockport         | United Kingdom
      88 | Bradford          | United Kingdom
     589 | York              | United Kingdom
     149 | Dundee            | United Kingdom
     312 | London            | United Kingdom
     494 | Southampton       | United Kingdom
(12 rows)
```

Conclusion/Reflection

The practical application of SQL concepts deepened my understanding, particularly in the areas of data manipulation, query optimization, and index usage. I gained valuable insights into the effective use of indexes for performance improvement, especially in large datasets. Moreover, the creation and use of a VIEW simplified data access, reinforcing the utility of these structures in SQL. Additionally, the use of subqueries and the IN operator to filter data from multiple countries further expanded my SQL capabilities.

LAB 8 (22/07/2023) -Data Manipulation and Query Output Formatting

Lab Description

In Lab 8, expanded proficiency in SQL through the "movie_rental" database, focusing on complex querying across multiple tables and data manipulation. The lab tasks involved pattern matching using ILIKE/LIKE, wildcard characters, counting unique values, and sorting data.

Additionally, addressed scenarios that involved handling data across multiple rows, such as identifying the most frequent occurrence of a value.

For this week we will use the movie_rental database. Make sure that you have the database installed and you are connected to it. You can use this ERD and this interactive Data Dictionary.

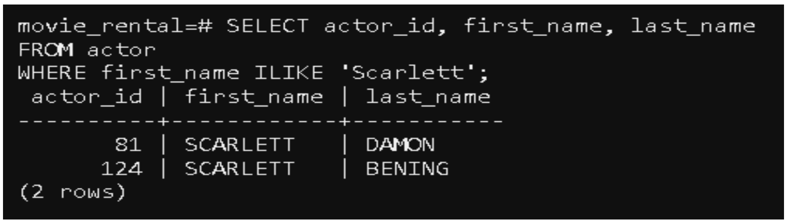
Q1. List **id, first name and last name** of all actors that have the first name as **Scarlett**. The query should ignore the name capitalisation. Hint: *Look for the **ILIKE**/**LIKE** keywords or wildcard characters.*

A:

[EDITABLE CODE]

```
SELECT actor_id, first_name, last_name
FROM actor
WHERE first_name ILIKE 'Scarlett';
```

[OUTPUT SCREENSHOT]



```
movie_rental=# SELECT actor_id, first_name, last_name
FROM actor
WHERE first_name ILIKE 'Scarlett';
 actor_id | first_name | last_name
----------+------------+-----------
       81 | SCARLETT   | DAMON
      124 | SCARLETT   | BENING
(2 rows)
```

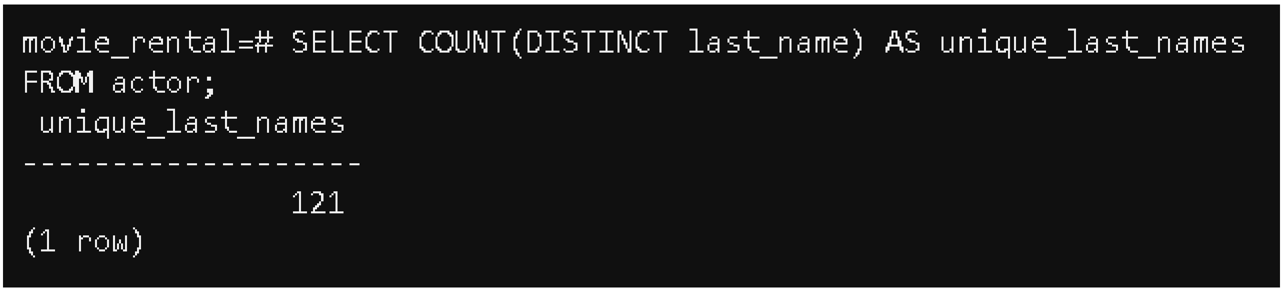
Q2. How many ***unique last name***s are in actors' names? (e.g. If they are 3 Smith's will be counted only once).

A:

[EDITABLE CODE]

```
SELECT COUNT(DISTINCT last_name) AS unique_last_names
FROM actor;
```

[OUTPUT SCREENSHOT]



```
movie_rental=# SELECT COUNT(DISTINCT last_name) AS unique_last_names
FROM actor;
 unique_last_names
-------------------
               121
(1 row)
```

Q3. Following from the previous query, we know how many unique last names are in the database. However, how many are truly unique?

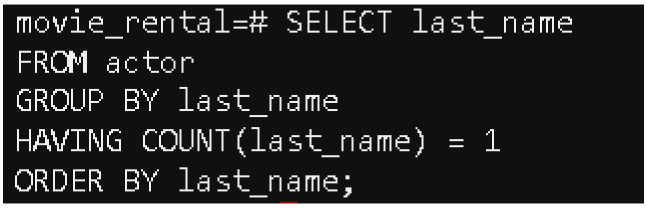
List in alphabetical order the last names that are repeated only once (e.g. if a last name is more than once in the database it will not be considered).

A:

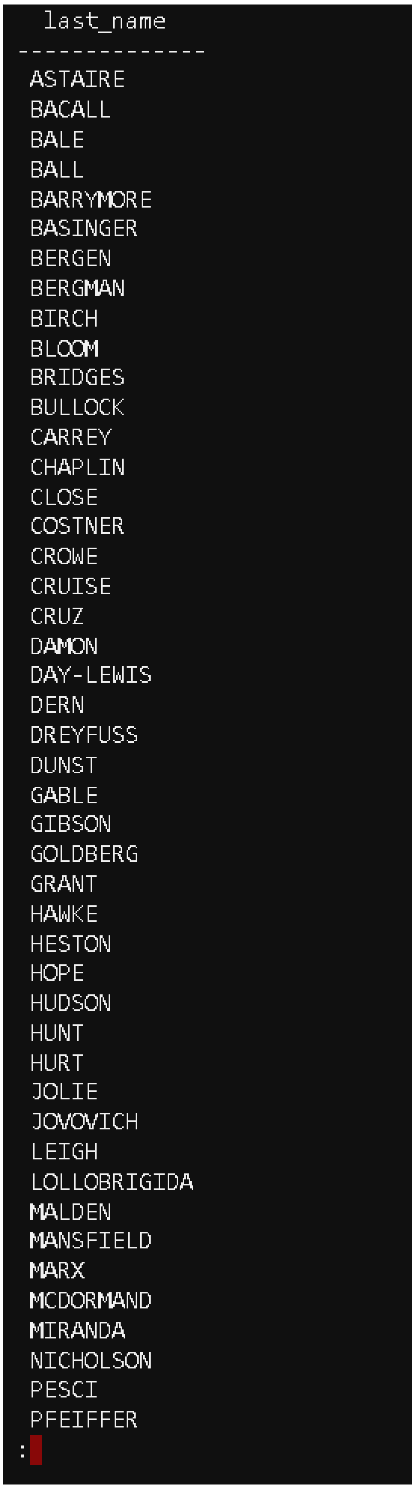
[EDITABLE CODE]

```
SELECT last_name
FROM actor
GROUP BY last_name
HAVING COUNT(last_name) = 1
ORDER BY last_name;
```

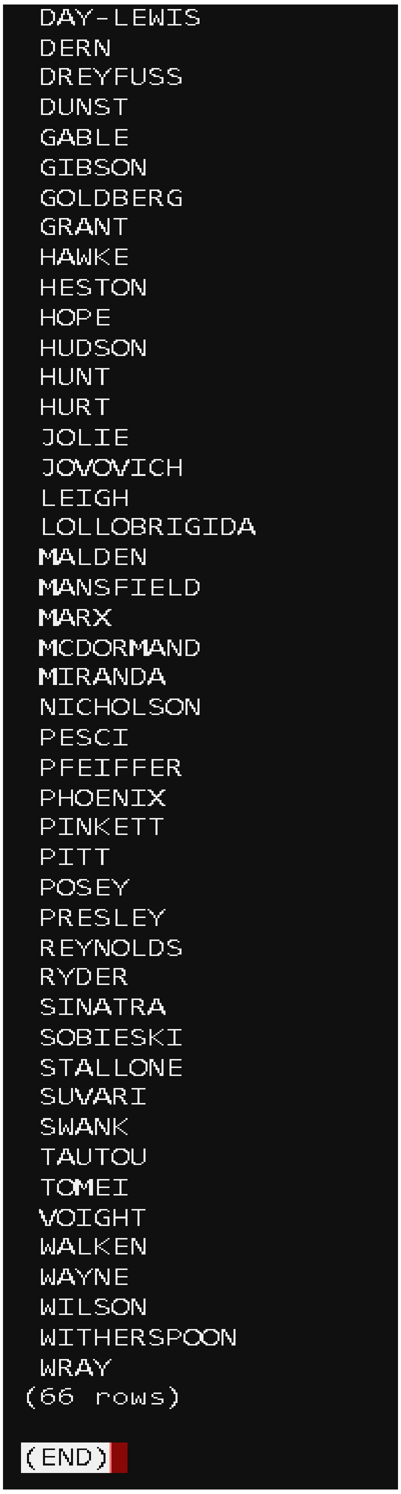
[OUTPUT SCREENSHOT]



```
movie_rental=# SELECT last_name
FROM actor
GROUP BY last_name
HAVING COUNT(last_name) = 1
ORDER BY last_name;
```



```
  last_name
--------------
 ASTAIRE
 BACALL
 BALE
 BALL
 BARRYMORE
 BASINGER
 BERGEN
 BERGMAN
 BIRCH
 BLOOM
 BRIDGES
 BULLOCK
 CARREY
 CHAPLIN
 CLOSE
 COSTNER
 CROWE
 CRUISE
 CRUZ
 DAMON
 DAY-LEWIS
 DERN
 DREYFUSS
 DUNST
 GABLE
 GIBSON
 GOLDBERG
 GRANT
 HAWKE
 HESTON
 HOPE
 HUDSON
 HUNT
 HURT
 JOLIE
 JOVOVICH
 LEIGH
 LOLLOBRIGIDA
 MALDEN
 MANSFIELD
 MARX
 MCDORMAND
 MIRANDA
 NICHOLSON
 PESCI
 PFEIFFER
:
```



```
 DAY-LEWIS
 DERN
 DREYFUSS
 DUNST
 GABLE
 GIBSON
 GOLDBERG
 GRANT
 HAWKE
 HESTON
 HOPE
 HUDSON
 HUNT
 HURT
 JOLIE
 JOVOVICH
 LEIGH
 LOLLOBRIGIDA
 MALDEN
 MANSFIELD
 MARX
 MCDORMAND
 MIRANDA
 NICHOLSON
 PESCI
 PFEIFFER
 PHOENIX
 PINKETT
 PITT
 POSEY
 PRESLEY
 REYNOLDS
 RYDER
 SINATRA
 SOBIESKI
 STALLONE
 SUVARI
 SWANK
 TAUTOU
 TOMEI
 VOIGHT
 WALKEN
 WAYNE
 WILSON
 WITHERSPOON
 WRAY
(66 rows)

(END)
```

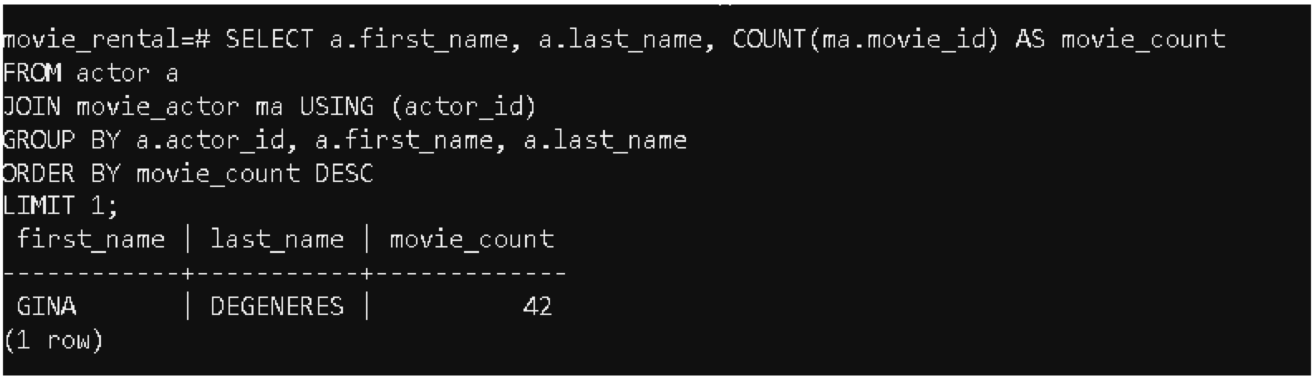
Q4. List the **first name and last name** of the actor that appear in most movies and the **number of movies**. Hint: Look for ***USING***.

A:

[EDITABLE CODE]

```
        SELECT a.first_name, a.last_name, COUNT(ma.movie_id) AS movie_count
        FROM actor a
        JOIN movie_actor ma USING (actor_id)
        GROUP BY a.actor_id, a.first_name, a.last_name
        ORDER BY movie_count DESC
        LIMIT 1;
```

[OUTPUT SCREENSHOT]



```
movie_rental=# SELECT a.first_name, a.last_name, COUNT(ma.movie_id) AS movie_count
FROM actor a
JOIN movie_actor ma USING (actor_id)
GROUP BY a.actor_id, a.first_name, a.last_name
ORDER BY movie_count DESC
LIMIT 1;
 first_name | last_name | movie_count
------------+-----------+-------------
 GINA       | DEGENERES |          42
(1 row)
```

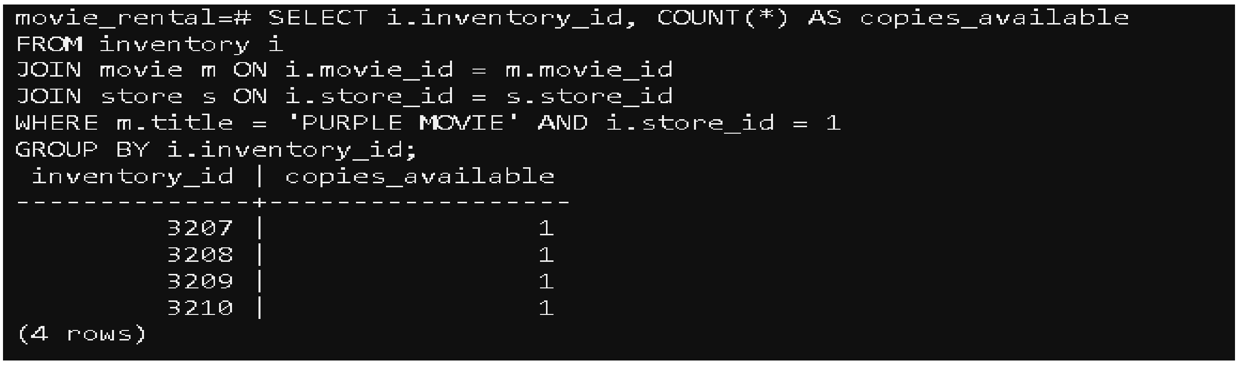
Q5. Each store has several copies of each movie. List the available **store 1 inventory** for the movie named ‘**Purple Movie**’. How many copies are available and what are the inventory IDs?

A:

[EDITABLE CODE]

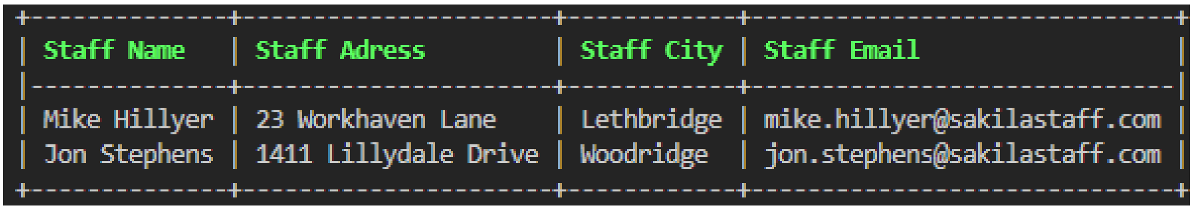
```
SELECT i.inventory_id, COUNT(*) AS copies_available
FROM inventory i
JOIN movie m ON i.movie_id = m.movie_id
JOIN store s ON i.store_id = s.store_id
WHERE m.title = 'PURPLE MOVIE' AND i.store_id = 1
GROUP BY i.inventory_id;
```

[OUTPUT SCREENSHOT]



```
movie_rental=# SELECT i.inventory_id, COUNT(*) AS copies_available
FROM inventory i
JOIN movie m ON i.movie_id = m.movie_id
JOIN store s ON i.store_id = s.store_id
WHERE m.title = 'PURPLE MOVIE' AND i.store_id = 1
GROUP BY i.inventory_id;
 inventory_id | copies_available
--------------+------------------
         3207 |                1
         3208 |                1
         3209 |                1
         3210 |                1
(4 rows)
```

Q6. List staff name and last name, along with their home address, city and email address. The data should be presented in a nice format (exactly as below).



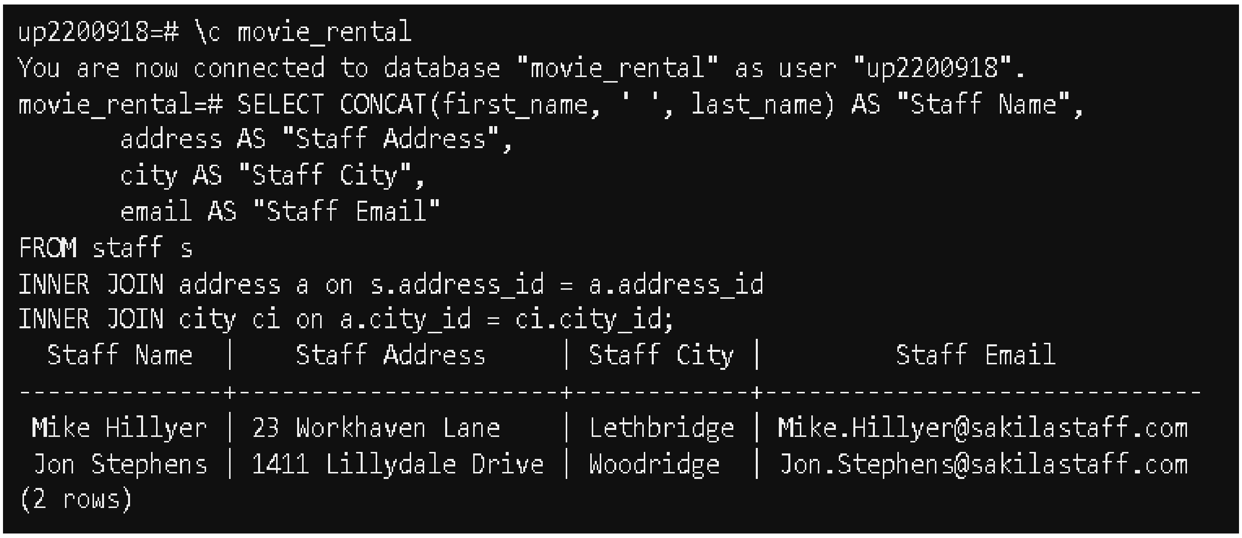
```
+-------------+---------------------+------------+------------------------------+
| Staff Name  | Staff Adress        | Staff City | Staff Email                  |
|-------------+---------------------+------------+------------------------------|
| Mike Hillyer| 23 Workhaven Lane   | Lethbridge | mike.hillyer@sakilastaff.com |
| Jon Stephens| 1411 Lillydale Drive| Woodridge  | jon.stephens@sakilastaff.com |
+-------------+---------------------+------------+------------------------------+
```

A:

[EDITABLE CODE]

```
SELECT CONCAT(first_name, ' ', last_name) AS "Staff Name",
       address AS "Staff Address",
       city AS "Staff City",
       email AS "Staff Email"
FROM staff s
INNER JOIN address a on s.address_id = a.address_id
INNER JOIN city ci on a.city_id = ci.city_id;
```

[OUTPUT SCREENSHOT]



```
up2200918=# \c movie_rental
You are now connected to database "movie_rental" as user "up2200918".
movie_rental=# SELECT CONCAT(first_name, ' ', last_name) AS "Staff Name",
       address AS "Staff Address",
       city AS "Staff City",
       email AS "Staff Email"
FROM staff s
INNER JOIN address a on s.address_id = a.address_id
INNER JOIN city ci on a.city_id = ci.city_id;
  Staff Name  |    Staff Address     | Staff City |         Staff Email
--------------+----------------------+------------+------------------------------
 Mike Hillyer | 23 Workhaven Lane    | Lethbridge | Mike.Hillyer@sakilastaff.com
 Jon Stephens | 1411 Lillydale Drive | Woodridge  | Jon.Stephens@sakilastaff.com
(2 rows)
```

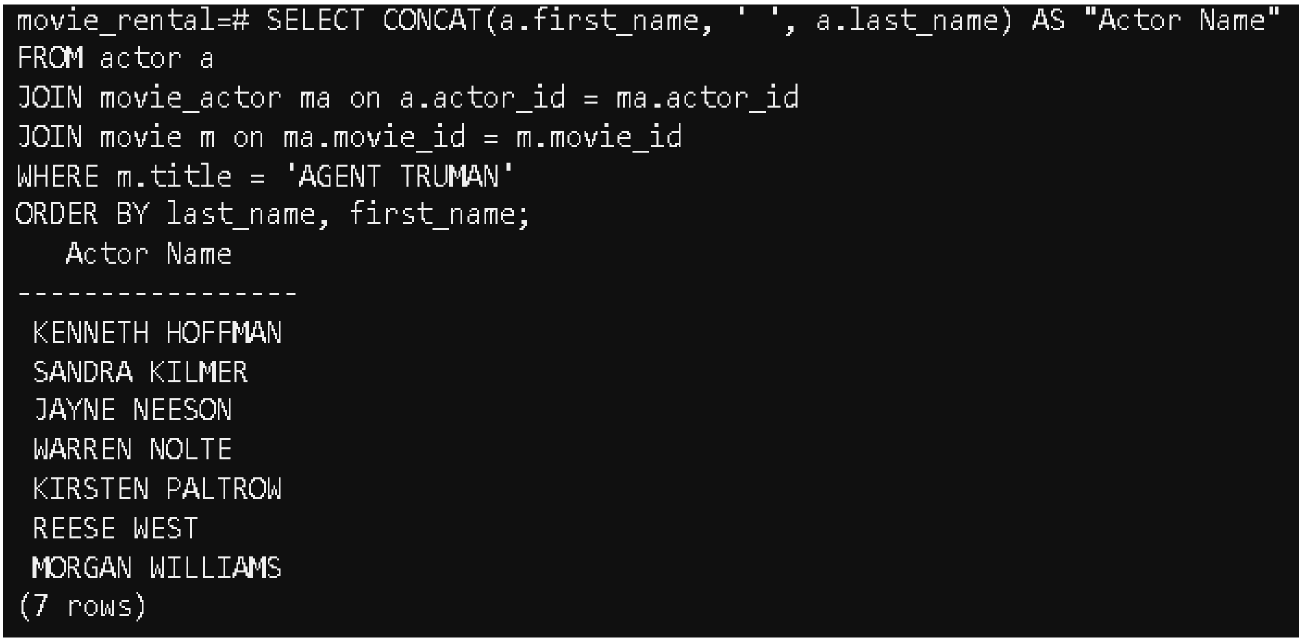
Q7. List the ***name and the last name*** in alphabetical order (on surname) of all actors that have acted in the movie named '**Agent Truman**'. The names should appear in a single column just the name of the actors, not the movie.

A:

[EDITABLE CODE]

```
SELECT CONCAT(a.first_name, ' ', a.last_name) AS "Actor Name"
FROM actor a
JOIN movie_actor ma on a.actor_id = ma.actor_id
JOIN movie m on ma.movie_id = m.movie_id
WHERE m.title = 'AGENT TRUMAN'
ORDER BY last_name, first_name;
```

[OUTPUT SCREENSHOT]



```
movie_rental=# SELECT CONCAT(a.first_name, ' ', a.last_name) AS "Actor Name"
FROM actor a
JOIN movie_actor ma on a.actor_id = ma.actor_id
JOIN movie m on ma.movie_id = m.movie_id
WHERE m.title = 'AGENT TRUMAN'
ORDER BY last_name, first_name;
   Actor Name
-----------------
 KENNETH HOFFMAN
 SANDRA KILMER
 JAYNE NEESON
 WARREN NOLTE
 KIRSTEN PALTROW
 REESE WEST
 MORGAN WILLIAMS
(7 rows)
```

Conclusion/Reflection

Lab 8 was an engaging learning experience that allowed us to apply and enhance our SQL skills in a practical context. I got to explore advanced SQL concepts, such as pattern matching, counting unique instances, and ordering data. I also gained valuable insights into how to tackle complex queries, handle data distributed across multiple rows, and present data in a user-friendly format.

This lab further solidified my SQL abilities, enabling me to analyze complex data patterns and optimize database querying. This knowledge will be beneficial for handling database-related tasks and projects in the future.

LAB 9 (25/07/2023) - Advanced SQL

Lab Description

In Lab 9, we continued to explore the "movie_rental" database with a specific focus on database functions and stored procedures. We learned how to list all the functions available in the database, call a function with parameters to retrieve specific data, and create and utilize a stored procedure for data insertion. The lab also involved tasks such as modifying a column data type, ensuring uniqueness, and performing calculations to derive insights from the data.

For this week we will use the movie_rental database. Make sure that you have the database installed and you are connected to it. You can use this ERD and Data Dictionary.

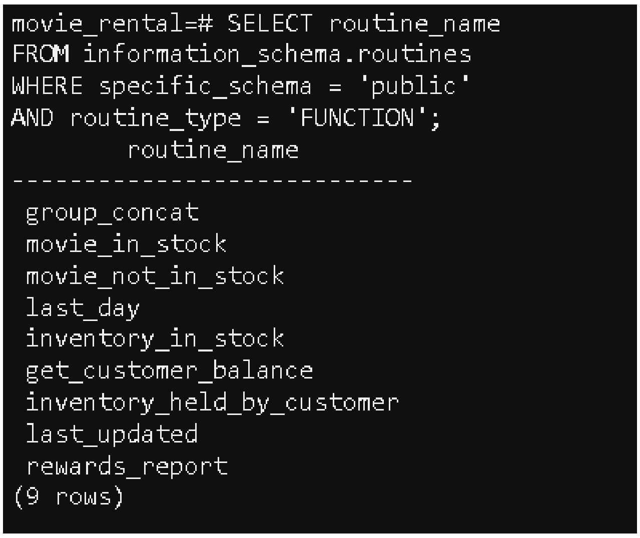
Q1. List all the functions available in the movie_rental database.

A:

[EDITABLE CODE]

```
SELECT routine_name
FROM information_schema.routines
WHERE specific_schema = 'public'
AND routine_type = 'FUNCTION';
```

[OUTPUT SCREENSHOT]

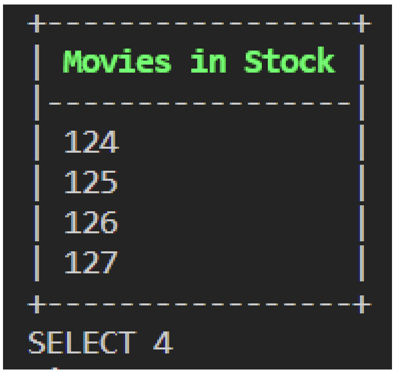


```
movie_rental=# SELECT routine_name
FROM information_schema.routines
WHERE specific_schema = 'public'
AND routine_type = 'FUNCTION';
        routine_name
----------------------------
 group_concat
 movie_in_stock
 movie_not_in_stock
 last_day
 inventory_in_stock
 get_customer_balance
 inventory_held_by_customer
 last_updated
 rewards_report
(9 rows)
```

Q2. In the database is a function (***movie_in_stock***), with two INT parameters (***p_movie_id*** and ***p_store_id***), that will return all available copies of a given movie in a particular store. Using the named function, find the number of copies of the movie Named “***Angels Life***” in store 1.

Hint: Look for named notation syntax format to call a function parameter.

The output should look like below.



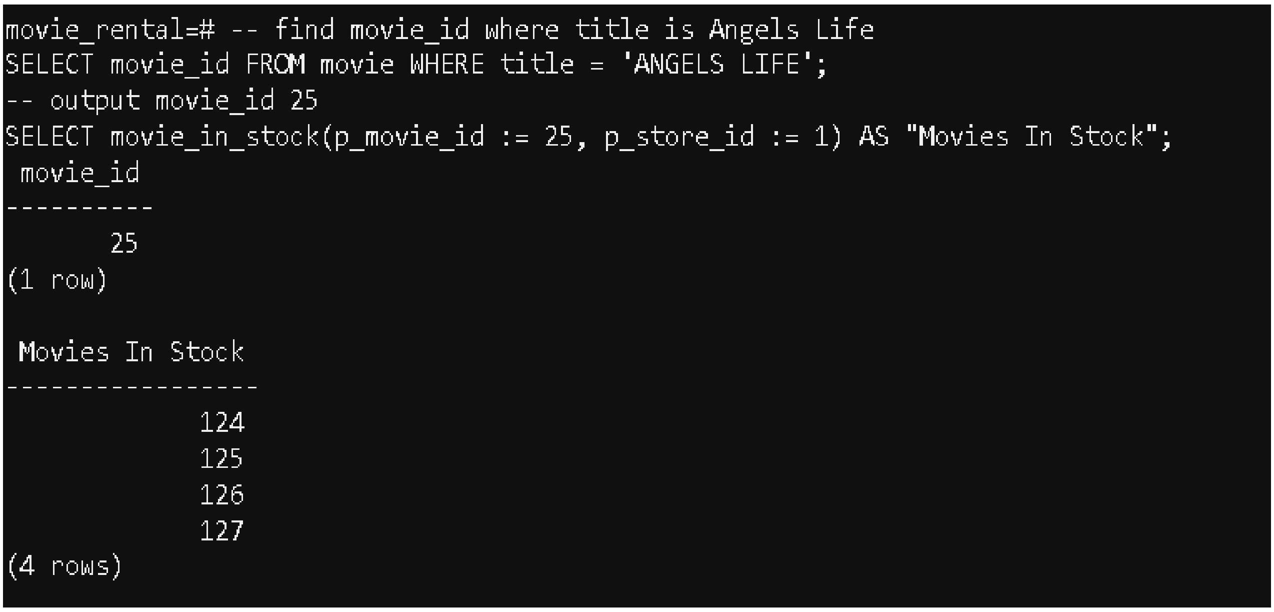
```
+-----------------+
| Movies in Stock |
|-----------------|
| 124             |
| 125             |
| 126             |
| 127             |
+-----------------+
SELECT 4
```

A:

[EDITABLE CODE]

```
-- find movie_id where title is Angels Life
SELECT movie_id FROM movie WHERE title = 'ANGELS LIFE';
-- output movie_id 25
SELECT movie_in_stock(p_movie_id := 25, p_store_id := 1) AS "Movies In
Stock";
```

[OUTPUT SCREENSHOT]



```
movie_rental=# -- find movie_id where title is Angels Life
SELECT movie_id FROM movie WHERE title = 'ANGELS LIFE';
-- output movie_id 25
SELECT movie_in_stock(p_movie_id := 25, p_store_id := 1) AS "Movies In Stock";
 movie_id
----------
       25
(1 row)

 Movies In Stock
-----------------
             124
             125
             126
             127
(4 rows)
```

Q3. Create a stored procedure (sp_add_new_actor) that will automatically insert a new actor.Insert your own name through stored procedure **CALL** e.g. CALL sp_add_new_actor('Val', 'Adamescu');

You don’t need to allocate any ID as is automatically added by SERIAL PK and the last_update is automatically updated by another function. The procedure should take only two parameters (First Name & Last Name).

A:

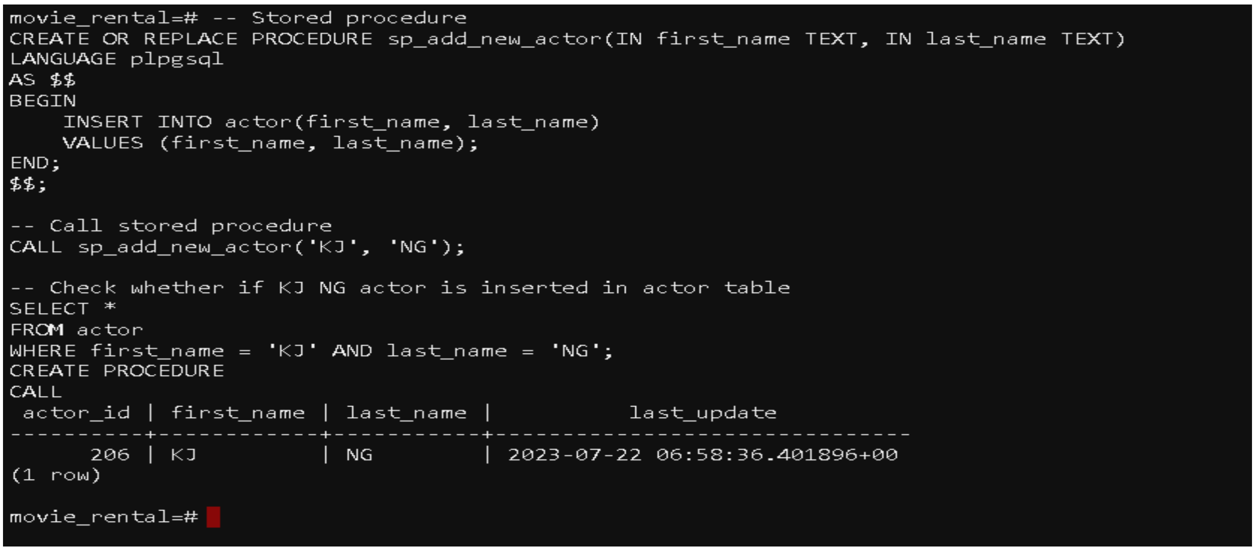
[EDITABLE CODE]

```
-- Stored procedure
CREATE OR REPLACE PROCEDURE sp_add_new_actor(IN first_name TEXT, IN
last_name TEXT)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO actor(first_name, last_name)
    VALUES (first_name, last_name);
END;
$$;

-- Call stored procedure
CALL sp_add_new_actor('KJ', 'NG');

-- Check whether if KJ NG actor is inserted in actor table
SELECT *
FROM actor
WHERE first_name = 'KJ' AND last_name = 'NG';
```

[OUTPUT SCREENSHOT]



```
movie_rental=# -- Stored procedure
CREATE OR REPLACE PROCEDURE sp_add_new_actor(IN first_name TEXT, IN last_name TEXT)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO actor(first_name, last_name)
    VALUES (first_name, last_name);
END;
$$;

-- Call stored procedure
CALL sp_add_new_actor('KJ', 'NG');

-- Check whether if KJ NG actor is inserted in actor table
SELECT *
FROM actor
WHERE first_name = 'KJ' AND last_name = 'NG';
CREATE PROCEDURE
CALL
 actor_id | first_name | last_name |          last_update
----------+------------+-----------+-------------------------------
      206 | KJ         | NG        | 2023-07-22 06:58:36.401896+00
(1 row)

movie_rental=#
```

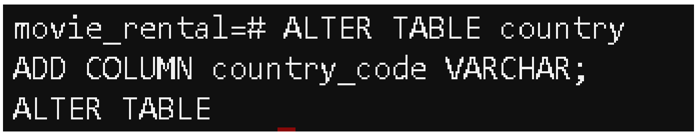
Q4. Create a new column in the country table named country_code as VARCHAR.

A:

[EDITABLE CODE]

```
ALTER TABLE country
ADD COLUMN country_code VARCHAR;
```

[OUTPUT SCREENSHOT]



```
movie_rental=# ALTER TABLE country
ADD COLUMN country_code VARCHAR;
ALTER TABLE
```

Q5. oops, my bad.. CHAR(2) data type will be more efficient. Modify the country_code from VARCHAR to CHAR(2) and to have only unique values for the column country_code. Use **\d country** and take a screenshot of the table details.

A:

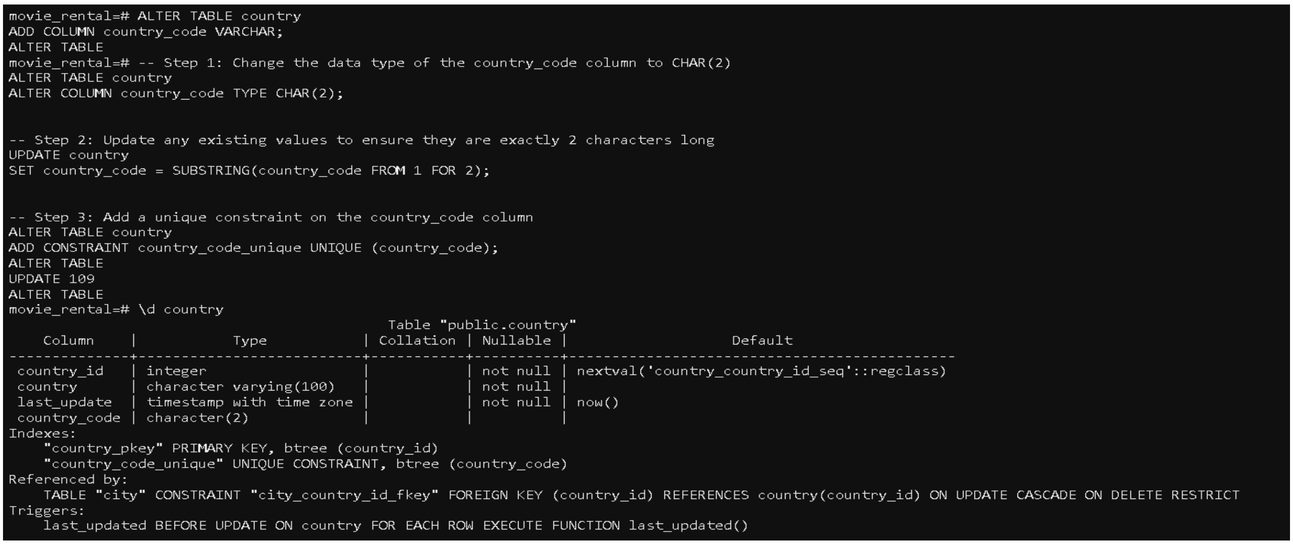
[EDITABLE CODE]

```
-- Step 1: Change the data type of the country_code column to CHAR(2)
ALTER TABLE country
ALTER COLUMN country_code TYPE CHAR(2);

-- Step 2: Update any existing values to ensure they are exactly 2
characters long
UPDATE country
SET country_code = SUBSTRING(country_code FROM 1 FOR 2);

-- Step 3: Add a unique constraint on the country_code column
ALTER TABLE country
ADD CONSTRAINT country_code_unique UNIQUE (country_code);
```

[OUTPUT SCREENSHOT]



```
movie_rental=# ALTER TABLE country
ADD COLUMN country_code VARCHAR;
ALTER TABLE
movie_rental=# -- Step 1: Change the data type of the country_code column to CHAR(2)
ALTER TABLE country
ALTER COLUMN country_code TYPE CHAR(2);


-- Step 2: Update any existing values to ensure they are exactly 2 characters long
UPDATE country
SET country_code = SUBSTRING(country_code FROM 1 FOR 2);


-- Step 3: Add a unique constraint on the country_code column
ALTER TABLE country
ADD CONSTRAINT country_code_unique UNIQUE (country_code);
ALTER TABLE
UPDATE 109
ALTER TABLE
movie_rental=# \d country
                                          Table "public.country"
    Column    |           Type           | Collation | Nullable |                   Default
--------------+--------------------------+-----------+----------+---------------------------------------------
 country_id   | integer                  |           | not null | nextval('country_country_id_seq'::regclass)
 country      | character varying(100)   |           | not null |
 last_update  | timestamp with time zone |           | not null | now()
 country_code | character(2)             |           |          |
Indexes:
    "country_pkey" PRIMARY KEY, btree (country_id)
    "country_code_unique" UNIQUE CONSTRAINT, btree (country_code)
Referenced by:
    TABLE "city" CONSTRAINT "city_country_id_fkey" FOREIGN KEY (country_id) REFERENCES country(country_id) ON UPDATE CASCADE ON DELETE RESTRICT
Triggers:
    last_updated BEFORE UPDATE ON country FOR EACH ROW EXECUTE FUNCTION last_updated()
```

Q6. Now if everything is ready, insert the country code as **UK** for the United Kingdom and create the output as country id, country name and country code.

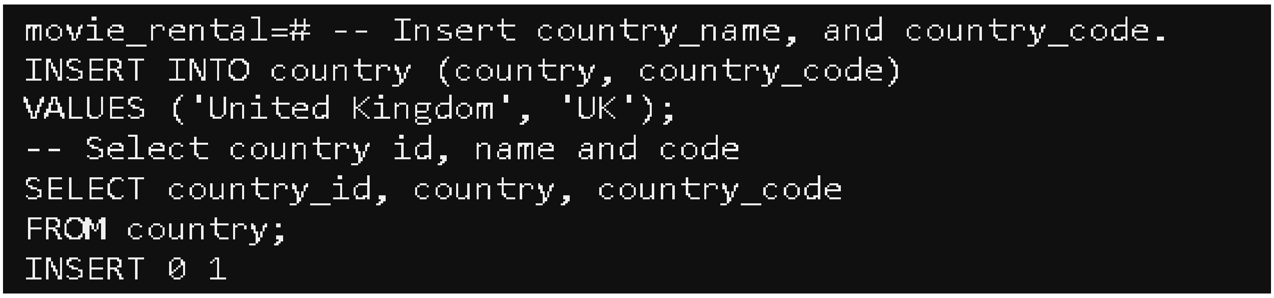
A:

[EDITABLE CODE]

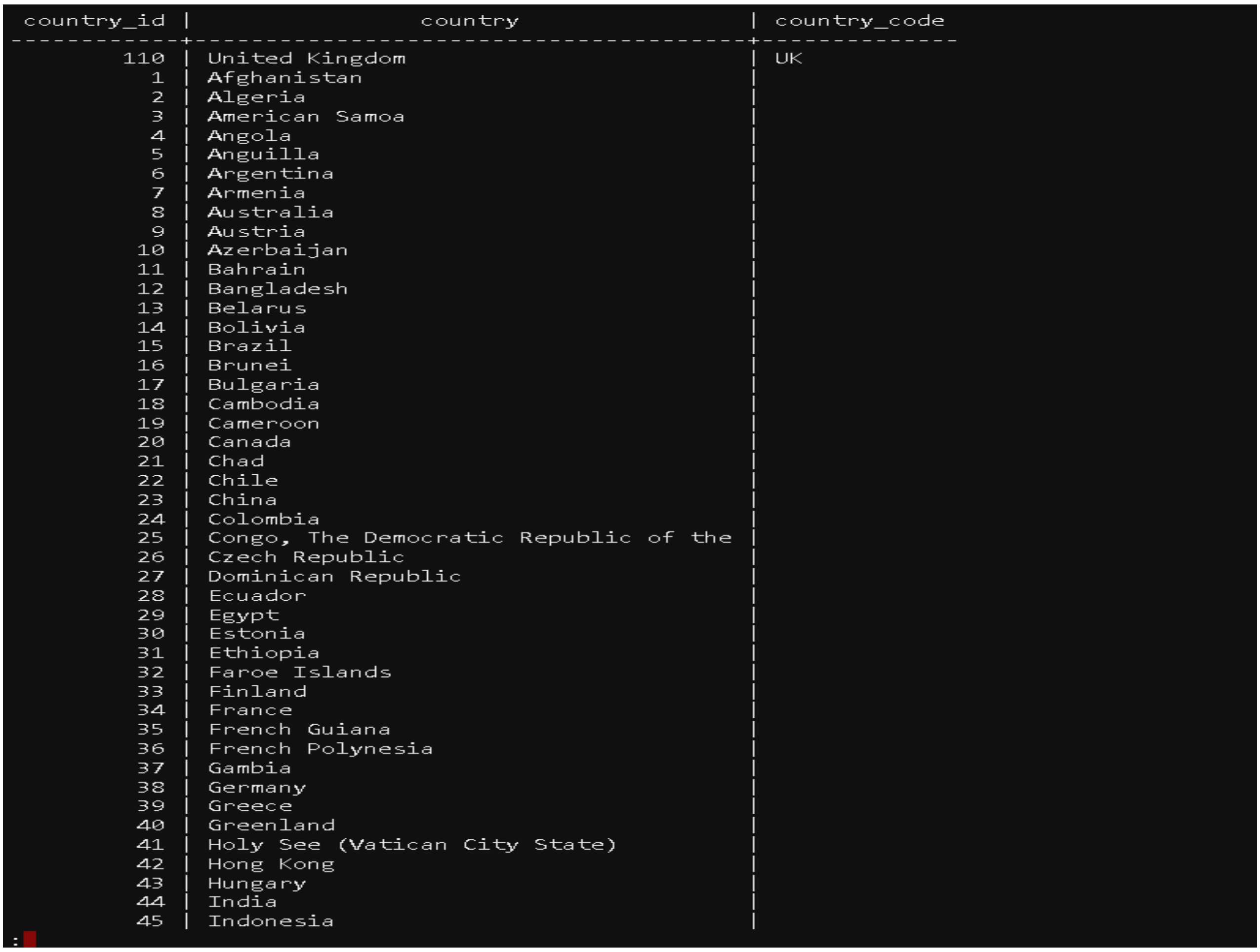
```
-- Insert country_name, and country_code.
INSERT INTO country (country, country_code)
VALUES ('United Kingdom', 'UK');

-- Select country id, name and code
SELECT country_id, country, country_code
FROM country;
```

[OUTPUT SCREENSHOT]



```
movie_rental=# -- Insert country_name, and country_code.
INSERT INTO country (country, country_code)
VALUES ('United Kingdom', 'UK');
-- Select country id, name and code
SELECT country_id, country, country_code
FROM country;
INSERT 0 1
```



```
 country_id |                country                 | country_code
------------+----------------------------------------+--------------
        110 | United Kingdom                         | UK
          1 | Afghanistan                            |
          2 | Algeria                                |
          3 | American Samoa                         |
          4 | Angola                                 |
          5 | Anguilla                               |
          6 | Argentina                              |
          7 | Armenia                                |
          8 | Australia                              |
          9 | Austria                                |
         10 | Azerbaijan                             |
         11 | Bahrain                                |
         12 | Bangladesh                             |
         13 | Belarus                                |
         14 | Bolivia                                |
         15 | Brazil                                 |
         16 | Brunei                                 |
         17 | Bulgaria                               |
         18 | Cambodia                               |
         19 | Cameroon                               |
         20 | Canada                                 |
         21 | Chad                                   |
         22 | Chile                                  |
         23 | China                                  |
         24 | Colombia                               |
         25 | Congo, The Democratic Republic of the  |
         26 | Czech Republic                         |
         27 | Dominican Republic                     |
         28 | Ecuador                                |
         29 | Egypt                                  |
         30 | Estonia                                |
         31 | Ethiopia                               |
         32 | Faroe Islands                          |
         33 | Finland                                |
         34 | France                                 |
         35 | French Guiana                          |
         36 | French Polynesia                       |
         37 | Gambia                                 |
         38 | Germany                                |
         39 | Greece                                 |
         40 | Greenland                              |
         41 | Holy See (Vatican City State)          |
         42 | Hong Kong                              |
         43 | Hungary                                |
         44 | India                                  |
         45 | Indonesia                              |
:
```

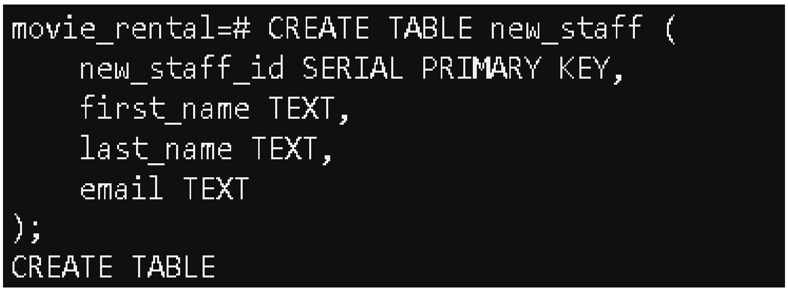
Q7. We are planning some migration of our data but we don't want to transfer everything. Create a new table (new_staff) and copy only id, first & last name and email address.

A:

[EDITABLE CODE]

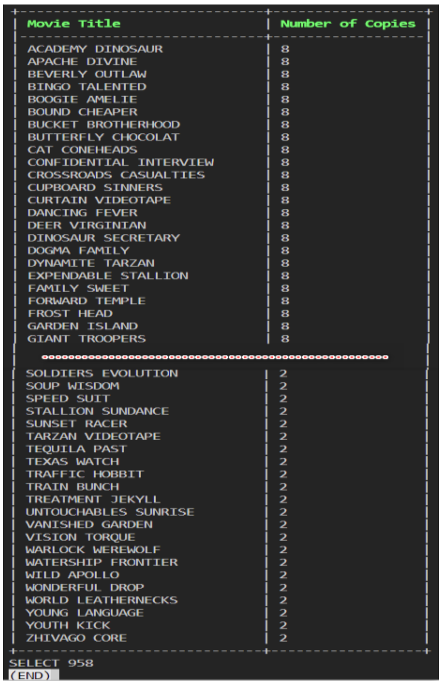
```
CREATE TABLE new_staff (
    new_staff_id SERIAL PRIMARY KEY,
    first_name TEXT,
    last_name TEXT,
    email TEXT
);
```

[OUTPUT SCREENSHOT]



```
movie_rental=# CREATE TABLE new_staff (
    new_staff_id SERIAL PRIMARY KEY,
    first_name TEXT,
    last_name TEXT,
    email TEXT
);
CREATE TABLE
```

Q8. We have so many copies of each movie for rental but how many? The output of the movies in inventory must be exactly as below. Column naming, text formatting, number of copies starting from the movies with most copies until the ones with less copies.



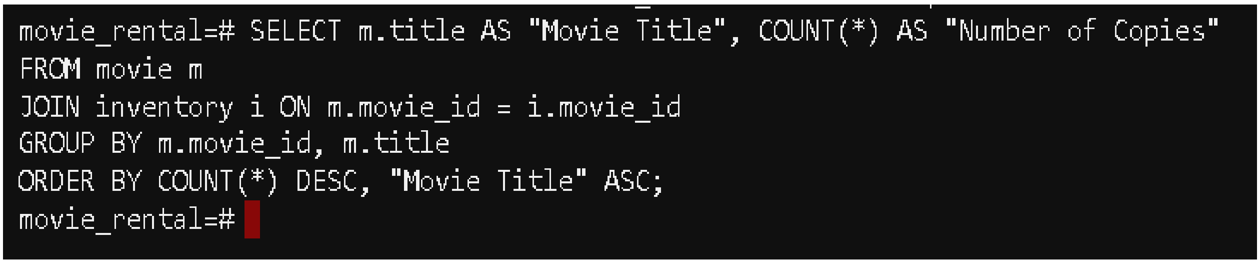
```
+--------------------------------+---------------------+
| Movie Title                    | Number of Copies    |
|--------------------------------+---------------------|
| ACADEMY DINOSAUR               | 8                   |
| APACHE DIVINE                  | 8                   |
| BEVERLY OUTLAW                 | 8                   |
| BINGO TALENTED                 | 8                   |
| BOOGIE AMELIE                  | 8                   |
| BOUND CHEAPER                  | 8                   |
| BUCKET BROTHERHOOD             | 8                   |
| BUTTERFLY CHOCOLAT             | 8                   |
| CAT CONEHEADS                  | 8                   |
| CONFIDENTIAL INTERVIEW         | 8                   |
| CROSSROADS CASUALTIES          | 8                   |
| CUPBOARD SINNERS               | 8                   |
| CURTAIN VIDEOTAPE              | 8                   |
| DANCING FEVER                  | 8                   |
| DEER VIRGINIAN                 | 8                   |
| DINOSAUR SECRETARY             | 8                   |
| DOGMA FAMILY                   | 8                   |
| DYNAMITE TARZAN                | 8                   |
| EXPENDABLE STALLION            | 8                   |
| FAMILY SWEET                   | 8                   |
| FORWARD TEMPLE                 | 8                   |
| FROST HEAD                     | 8                   |
| GARDEN ISLAND                  | 8                   |
| GIANT TROOPERS                 | 8                   |
|                                                      |
| SOLDIERS EVOLUTION             | 2                   |
| SOUP WISDOM                    | 2                   |
| SPEED SUIT                     | 2                   |
| STALLION SUNDANCE              | 2                   |
| SUNSET RACER                   | 2                   |
| TARZAN VIDEOTAPE               | 2                   |
| TEQUILA PAST                   | 2                   |
| TEXAS WATCH                    | 2                   |
| TRAFFIC HOBBIT                 | 2                   |
| TRAIN BUNCH                    | 2                   |
| TREATMENT JEKYLL               | 2                   |
| UNTOUCHABLES SUNRISE           | 2                   |
| VANISHED GARDEN                | 2                   |
| VISION TORQUE                  | 2                   |
| WARLOCK WEREWOLF               | 2                   |
| WATERSHIP FRONTIER             | 2                   |
| WILD APOLLO                    | 2                   |
| WONDERFUL DROP                 | 2                   |
| WORLD LEATHERNECKS             | 2                   |
| YOUNG LANGUAGE                 | 2                   |
| YOUTH KICK                     | 2                   |
| ZHIVAGO CORE                   | 2                   |
+--------------------------------+---------------------+
SELECT 958
(END)
```

A:

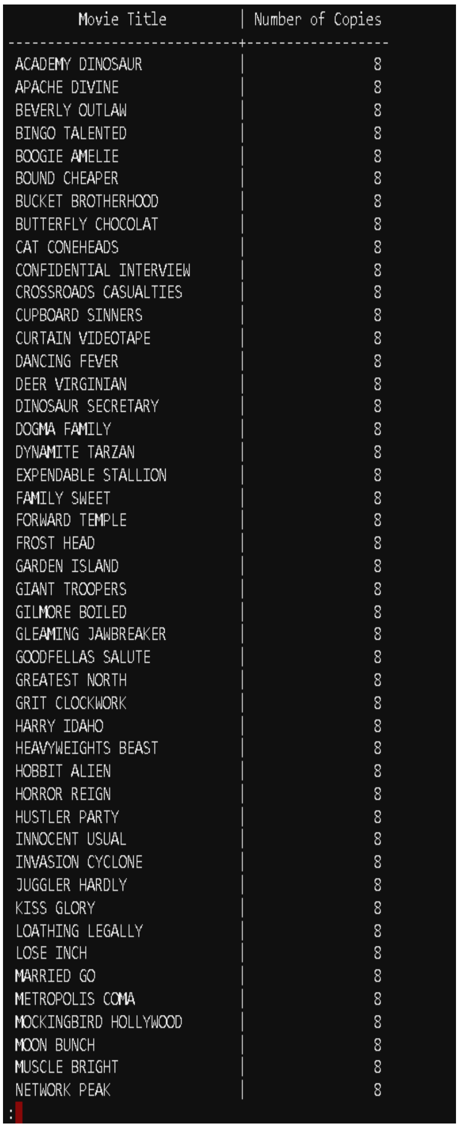
[EDITABLE CODE]

```sql
SELECT m.title AS "Movie Title",
       COUNT(*) AS "Number of Copies"
FROM movie m
JOIN inventory i ON m.movie_id = i.movie_id
GROUP BY m.movie_id, m.title
ORDER BY COUNT(*) DESC, "Movie Title" ASC;
```

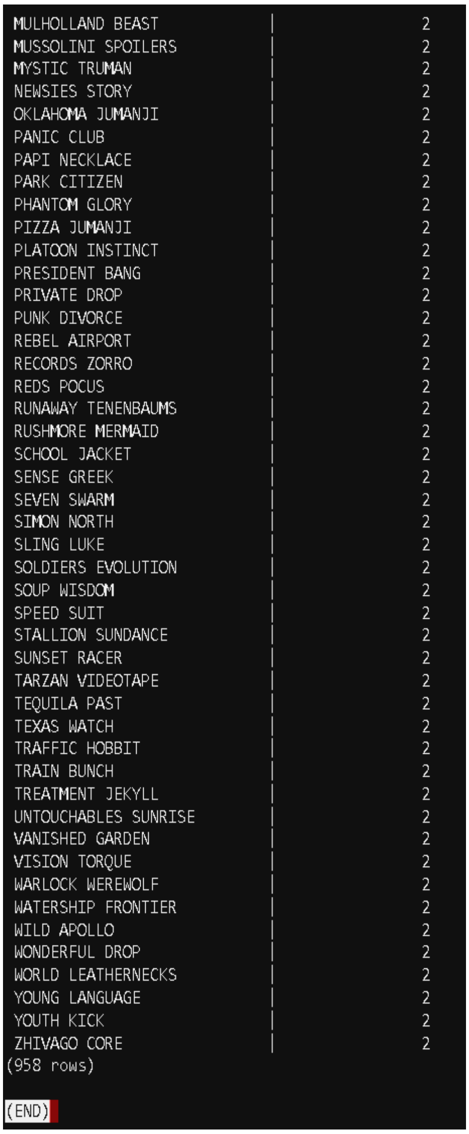
[OUTPUT SCREENSHOT]



```
movie_rental=# SELECT m.title AS "Movie Title", COUNT(*) AS "Number of Copies"
FROM movie m
JOIN inventory i ON m.movie_id = i.movie_id
GROUP BY m.movie_id, m.title
ORDER BY COUNT(*) DESC, "Movie Title" ASC;
movie_rental=#
```



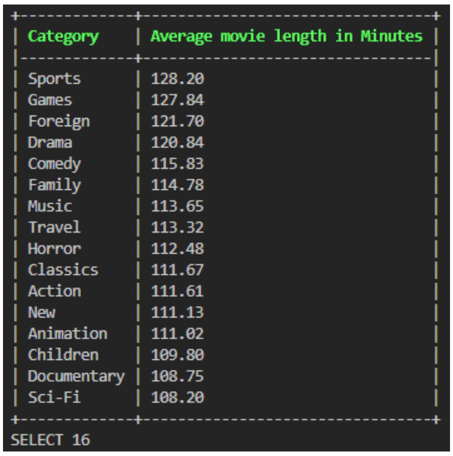
```
        Movie Title         | Number of Copies
----------------------------+------------------
 ACADEMY DINOSAUR           |                8
 APACHE DIVINE              |                8
 BEVERLY OUTLAW             |                8
 BINGO TALENTED             |                8
 BOOGIE AMELIE              |                8
 BOUND CHEAPER              |                8
 BUCKET BROTHERHOOD         |                8
 BUTTERFLY CHOCOLAT         |                8
 CAT CONEHEADS              |                8
 CONFIDENTIAL INTERVIEW     |                8
 CROSSROADS CASUALTIES      |                8
 CUPBOARD SINNERS           |                8
 CURTAIN VIDEOTAPE          |                8
 DANCING FEVER              |                8
 DEER VIRGINIAN             |                8
 DINOSAUR SECRETARY         |                8
 DOGMA FAMILY               |                8
 DYNAMITE TARZAN            |                8
 EXPENDABLE STALLION        |                8
 FAMILY SWEET               |                8
 FORWARD TEMPLE             |                8
 FROST HEAD                 |                8
 GARDEN ISLAND              |                8
 GIANT TROOPERS             |                8
 GILMORE BOILED             |                8
 GLEAMING JAWBREAKER        |                8
 GOODFELLAS SALUTE          |                8
 GREATEST NORTH             |                8
 GRIT CLOCKWORK             |                8
 HARRY IDAHO                |                8
 HEAVYWEIGHTS BEAST         |                8
 HOBBIT ALIEN               |                8
 HORROR REIGN               |                8
 HUSTLER PARTY              |                8
 INNOCENT USUAL             |                8
 INVASION CYCLONE           |                8
 JUGGLER HARDLY             |                8
 KISS GLORY                 |                8
 LOATHING LEGALLY           |                8
 LOSE INCH                  |                8
 MARRIED GO                 |                8
 METROPOLIS COMA            |                8
 MOCKINGBIRD HOLLYWOOD      |                8
 MOON BUNCH                 |                8
 MUSCLE BRIGHT              |                8
 NETWORK PEAK               |                8
:
```



```
 MULHOLLAND BEAST           |                2
 MUSSOLINI SPOILERS         |                2
 MYSTIC TRUMAN              |                2
 NEWSIES STORY              |                2
 OKLAHOMA JUMANJI           |                2
 PANIC CLUB                 |                2
 PAPI NECKLACE              |                2
 PARK CITIZEN               |                2
 PHANTOM GLORY              |                2
 PIZZA JUMANJI              |                2
 PLATOON INSTINCT           |                2
 PRESIDENT BANG             |                2
 PRIVATE DROP               |                2
 PUNK DIVORCE               |                2
 REBEL AIRPORT              |                2
 RECORDS ZORRO              |                2
 REDS POCUS                 |                2
 RUNAWAY TENENBAUMS         |                2
 RUSHMORE MERMAID           |                2
 SCHOOL JACKET              |                2
 SENSE GREEK                |                2
 SEVEN SWARM                |                2
 SIMON NORTH                |                2
 SLING LUKE                 |                2
 SOLDIERS EVOLUTION         |                2
 SOUP WISDOM                |                2
 SPEED SUIT                 |                2
 STALLION SUNDANCE          |                2
 SUNSET RACER               |                2
 TARZAN VIDEOTAPE           |                2
 TEQUILA PAST               |                2
 TEXAS WATCH                |                2
 TRAFFIC HOBBIT             |                2
 TRAIN BUNCH                |                2
 TREATMENT JEKYLL           |                2
 UNTOUCHABLES SUNRISE       |                2
 VANISHED GARDEN            |                2
 VISION TORQUE              |                2
 WARLOCK WEREWOLF           |                2
 WATERSHIP FRONTIER         |                2
 WILD APOLLO                |                2
 WONDERFUL DROP             |                2
 WORLD LEATHERNECKS         |                2
 YOUNG LANGUAGE             |                2
 YOUTH KICK                 |                2
 ZHIVAGO CORE               |                2
(958 rows)

(END)
```

Q9. What is the average movie length per category? Round it to the nearest two decimal places in descending order as below.



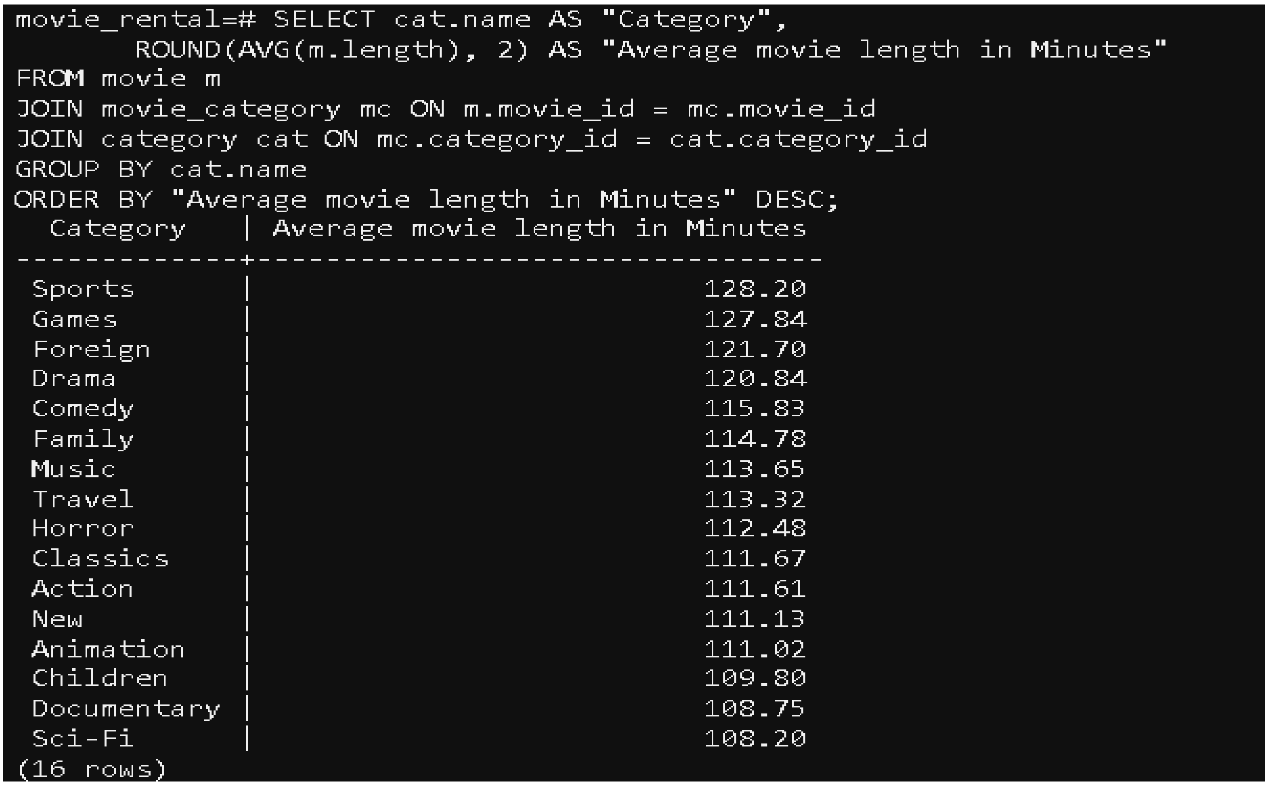
```
+-------------+--------------------------------+
| Category    | Average movie length in Minutes |
|-------------+--------------------------------|
| Sports      | 128.20                          |
| Games       | 127.84                          |
| Foreign     | 121.70                          |
| Drama       | 120.84                          |
| Comedy      | 115.83                          |
| Family      | 114.78                          |
| Music       | 113.65                          |
| Travel      | 113.32                          |
| Horror      | 112.48                          |
| Classics    | 111.67                          |
| Action      | 111.61                          |
| New         | 111.13                          |
| Animation   | 111.02                          |
| Children    | 109.80                          |
| Documentary | 108.75                          |
| Sci-Fi      | 108.20                          |
+-------------+--------------------------------+
SELECT 16
```

A:

[EDITABLE CODE]

```
SELECT cat.name AS "Category",
       ROUND(AVG(m.length), 2) AS "Average movie length in Minutes"
FROM movie m
JOIN movie_category mc ON m.movie_id = mc.movie_id
JOIN category cat ON mc.category_id = cat.category_id
GROUP BY cat.name
ORDER BY "Average movie length in Minutes" DESC;
```

[OUTPUT SCREENSHOT]



```
movie_rental=# SELECT cat.name AS "Category",
       ROUND(AVG(m.length), 2) AS "Average movie length in Minutes"
FROM movie m
JOIN movie_category mc ON m.movie_id = mc.movie_id
JOIN category cat ON mc.category_id = cat.category_id
GROUP BY cat.name
ORDER BY "Average movie length in Minutes" DESC;
  Category   | Average movie length in Minutes
-------------+---------------------------------
 Sports      |                          128.20
 Games       |                          127.84
 Foreign     |                          121.70
 Drama       |                          120.84
 Comedy      |                          115.83
 Family      |                          114.78
 Music       |                          113.65
 Travel      |                          113.32
 Horror      |                          112.48
 Classics    |                          111.67
 Action      |                          111.61
 New         |                          111.13
 Animation   |                          111.02
 Children    |                          109.80
 Documentary |                          108.75
 Sci-Fi      |                          108.20
(16 rows)
```

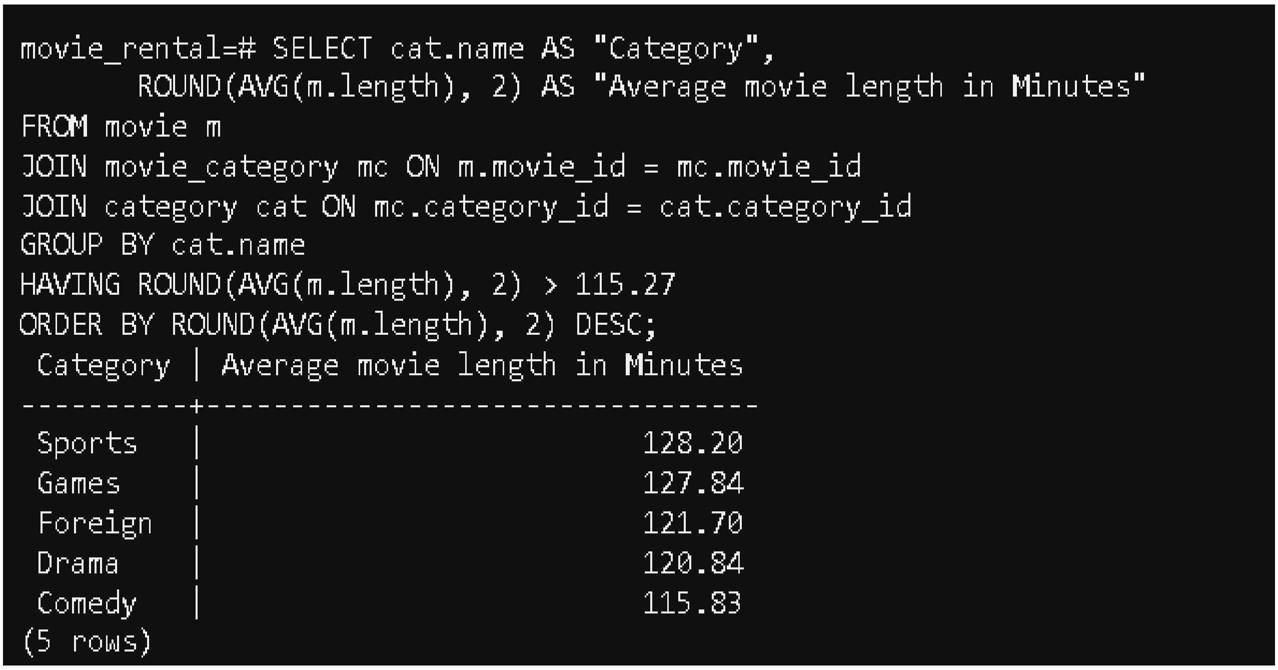
Q10. We know that the average length of all movies is 115.27. Which categories have movies above average? Do not use LIMIT but select only categories that are above the average in descending order and rounded to nearest two decimal places. *Hint: This query can be achieved in multiple ways. You can use HAVING, USING, and subqueries.*

A:

[EDITABLE CODE]

```
SELECT cat.name AS "Category",
       ROUND(AVG(m.length), 2) AS "Average movie length in Minutes"
FROM movie m
JOIN movie_category mc ON m.movie_id = mc.movie_id
JOIN category cat ON mc.category_id = cat.category_id
GROUP BY cat.name
HAVING ROUND(AVG(m.length), 2) > 115.27
ORDER BY ROUND(AVG(m.length), 2) DESC;
```

[OUTPUT SCREENSHOT]



```
movie_rental=# SELECT cat.name AS "Category",
       ROUND(AVG(m.length), 2) AS "Average movie length in Minutes"
FROM movie m
JOIN movie_category mc ON m.movie_id = mc.movie_id
JOIN category cat ON mc.category_id = cat.category_id
GROUP BY cat.name
HAVING ROUND(AVG(m.length), 2) > 115.27
ORDER BY ROUND(AVG(m.length), 2) DESC;
 Category | Average movie length in Minutes
----------+---------------------------------
 Sports   |                          128.20
 Games    |                          127.84
 Foreign  |                          121.70
 Drama    |                          120.84
 Comedy   |                          115.83
(5 rows)
```

Conclusion/Reflection

Lab 9 provided an excellent opportunity to dive deeper into the practical aspects of SQL, particularly functions and stored procedures. I learned how to use existing functions to retrieve specific data and how to create a stored procedure to automate common database operations, such as inserting a new actor.

The lab improved my understanding of how to effectively use these powerful SQL features to simplify database tasks and improve code reusability. Moreover, working on modifying column data types and ensuring uniqueness reinforced my understanding of database design principles and the importance of data integrity.

Overall, Lab 9 was a challenging but enlightening experience, significantly enhancing my SQL skills and practical knowledge of databases. This knowledge will be invaluable in future work involving database manipulation and data analysis.

LAB 10 (25/07/2023) - Monitoring, Tuning, Backup & Recovery

Lab Description

Lab 10 was an intensive hands-on exercise that dove into the nitty-gritty of database administration through Linux-specific commands on the "movie_rental" database. The lab guided us through the creation of a backup for the database, followed by its deletion and subsequent restoration from the backup. Alongside this, we also executed various SQL queries to extract actionable business insights like identifying countries with the most rented movies, customers with a penchant for Sci-Fi movies, the most revenue-generating movie, and more.

For this week we will use the movie_rental database. Make sure that you have the database installed and you are connected to it. You can use this ERD and Data Dictionary.

Note: All Q1 - Q3 should be done through CLI at the server level (**$**) and not through ***psql****,* including CREATE DATABASE command.

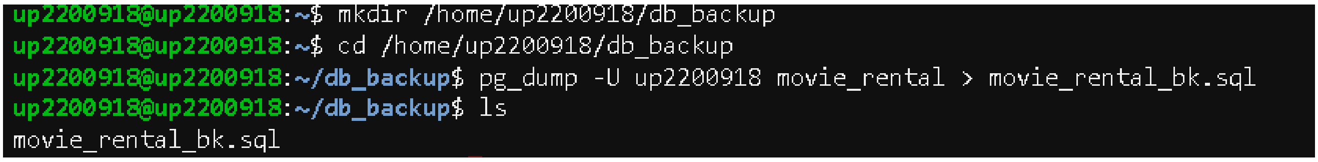
Q1. Using linux specific commands (CLI) ***create a new folder and a SQL backup file of the database movie_rental,*** on your server as ***upxxxxx/db_backup/movie_rental_bk.sql .*** Remember, in order to use Linux specific commands, you need to be at the server level and **not** database (**$**).

A:

[EDITABLE CODE]

```
mkdir /home/up2200918/db_backup
cd /home/up2200918/db_backup
pg_dump -U up2200918 movie_rental > movie_rental_bk.sql
```

[OUTPUT SCREENSHOT]



```
up2200918@up2200918:~$ mkdir /home/up2200918/db_backup
up2200918@up2200918:~$ cd /home/up2200918/db_backup
up2200918@up2200918:~/db_backup$ pg_dump -U up2200918 movie_rental > movie_rental_bk.sql
up2200918@up2200918:~/db_backup$ ls
movie_rental_bk.sql
```

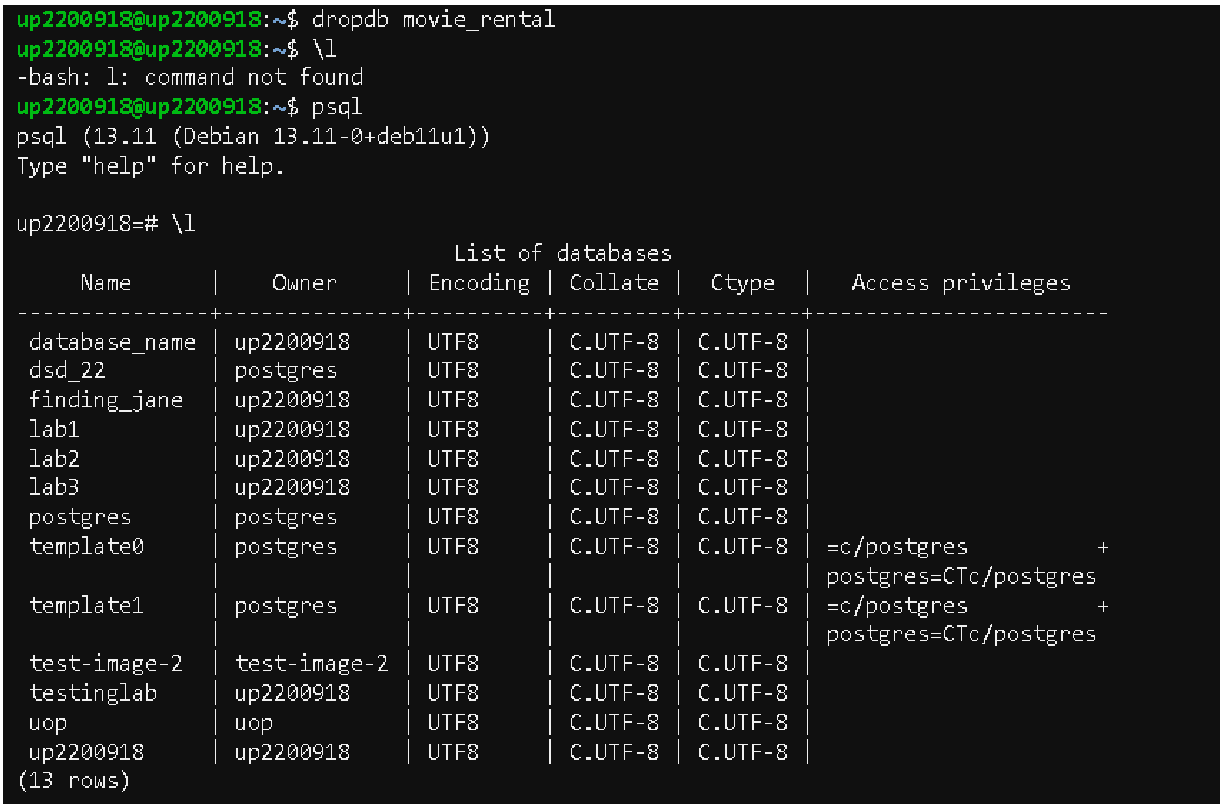
Q2. Using linux specific commands (CLI) ***delete database movie_rental***.

A:

[EDITABLE CODE]

```
dropdb movie_rental
```

[OUTPUT SCREENSHOT]



```
up2200918@up2200918:~$ dropdb movie_rental
up2200918@up2200918:~$ \l
-bash: l: command not found
up2200918@up2200918:~$ psql
psql (13.11 (Debian 13.11-0+deb11u1))
Type "help" for help.

up2200918=# \l
                                  List of databases
     Name      |    Owner     | Encoding | Collate |  Ctype  |   Access privileges
---------------+--------------+----------+---------+---------+-----------------------
 database_name | up2200918    | UTF8     | C.UTF-8 | C.UTF-8 |
 dsd_22        | postgres     | UTF8     | C.UTF-8 | C.UTF-8 |
 finding_jane  | up2200918    | UTF8     | C.UTF-8 | C.UTF-8 |
 lab1          | up2200918    | UTF8     | C.UTF-8 | C.UTF-8 |
 lab2          | up2200918    | UTF8     | C.UTF-8 | C.UTF-8 |
 lab3          | up2200918    | UTF8     | C.UTF-8 | C.UTF-8 |
 postgres      | postgres     | UTF8     | C.UTF-8 | C.UTF-8 |
 template0     | postgres     | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres          +
               |              |          |         |         | postgres=CTc/postgres
 template1     | postgres     | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres          +
               |              |          |         |         | postgres=CTc/postgres
 test-image-2  | test-image-2 | UTF8     | C.UTF-8 | C.UTF-8 |
 testinglab    | up2200918    | UTF8     | C.UTF-8 | C.UTF-8 |
 uop           | uop          | UTF8     | C.UTF-8 | C.UTF-8 |
 up2200918     | up2200918    | UTF8     | C.UTF-8 | C.UTF-8 |
(13 rows)
```

Q3. Using linux commands, ***restore your database from the backup***. If you are encountering issues or for any reason you cannot restore the database from backup, make another clean installation of the database and start from Q1.

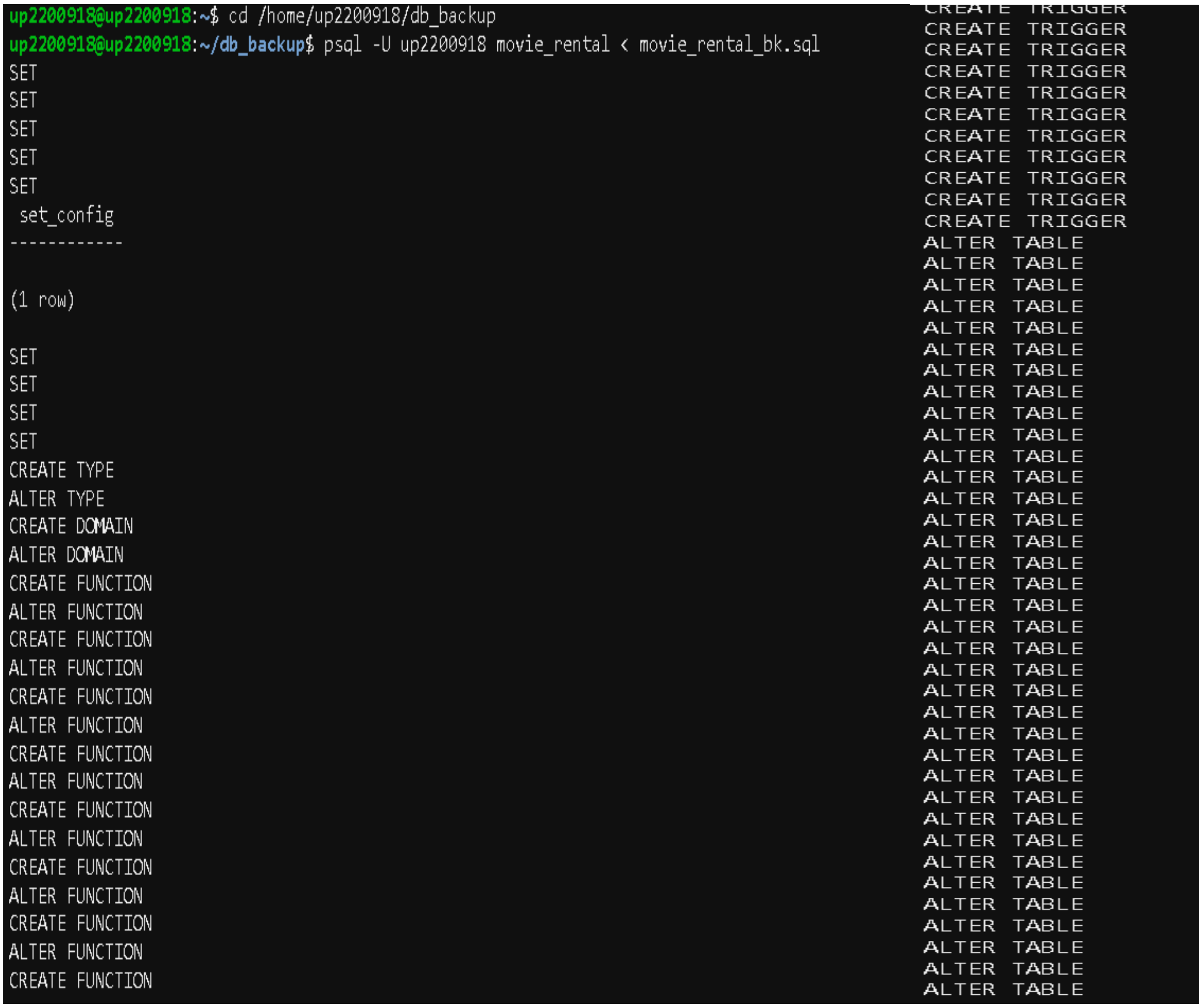
A:

[EDITABLE CODE]

```
-- Create a new clean database
createdb movie_rental
-- Move to backup folder
```

```
cd /home/up2200918/db_backup
-- Restore the database from the backup
psql -U up2200918 movie_rental < movie_rental_bk.sql
```

[OUTPUT SCREENSHOT]



```
up2200918@up2200918:~$ cd /home/up2200918/db_backup
up2200918@up2200918:~/db_backup$ psql -U up2200918 movie_rental < movie_rental_bk.sql
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
CREATE TYPE
ALTER TYPE
CREATE DOMAIN
ALTER DOMAIN
CREATE FUNCTION
ALTER FUNCTION
CREATE FUNCTION
ALTER FUNCTION
CREATE FUNCTION
ALTER FUNCTION
CREATE FUNCTION
ALTER FUNCTION
CREATE FUNCTION
ALTER FUNCTION
CREATE FUNCTION
ALTER FUNCTION
CREATE FUNCTION
ALTER FUNCTION
CREATE FUNCTION
```

```
CREATE TRIGGER
CREATE TRIGGER
CREATE TRIGGER
CREATE TRIGGER
CREATE TRIGGER
CREATE TRIGGER
CREATE TRIGGER
CREATE TRIGGER
CREATE TRIGGER
CREATE TRIGGER
CREATE TRIGGER
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
```

Q4. Now that the DB is restored, let's see which country has the most rented movies, so the manager will be able to create some marketing strategies. The manager would like to focus on the top 5.

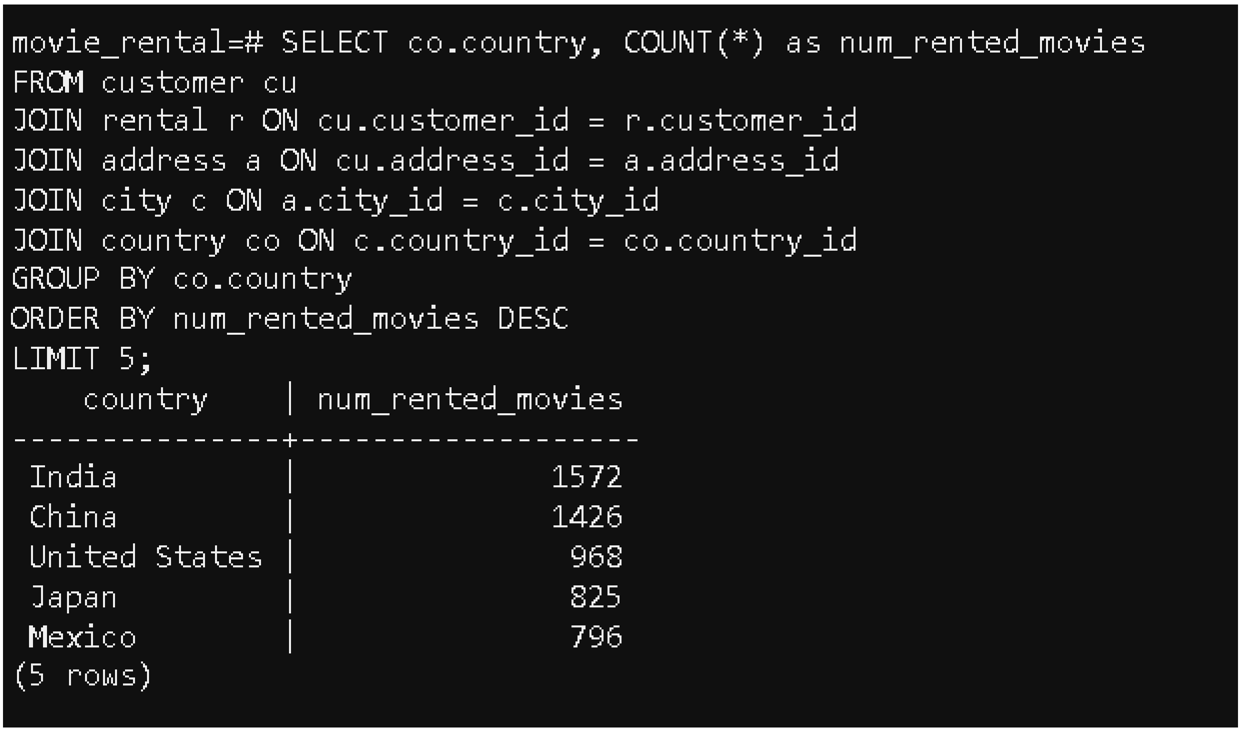
A:

[EDITABLE CODE]

```
SELECT co.country, COUNT(*) as num_rented_movies
FROM customer cu
JOIN rental r ON cu.customer_id = r.customer_id
JOIN address a ON cu.address_id = a.address_id
JOIN city c ON a.city_id = c.city_id
JOIN country co ON c.country_id = co.country_id
GROUP BY co.country
ORDER BY num_rented_movies DESC
```

```
LIMIT 5;
```

[OUTPUT SCREENSHOT]



```
movie_rental=# SELECT co.country, COUNT(*) as num_rented_movies
FROM customer cu
JOIN rental r ON cu.customer_id = r.customer_id
JOIN address a ON cu.address_id = a.address_id
JOIN city c ON a.city_id = c.city_id
JOIN country co ON c.country_id = co.country_id
GROUP BY co.country
ORDER BY num_rented_movies DESC
LIMIT 5;
    country     | num_rented_movies
---------------+-------------------
 India         |              1572
 China         |              1426
 United States |               968
 Japan         |               825
 Mexico        |               796
(5 rows)
```

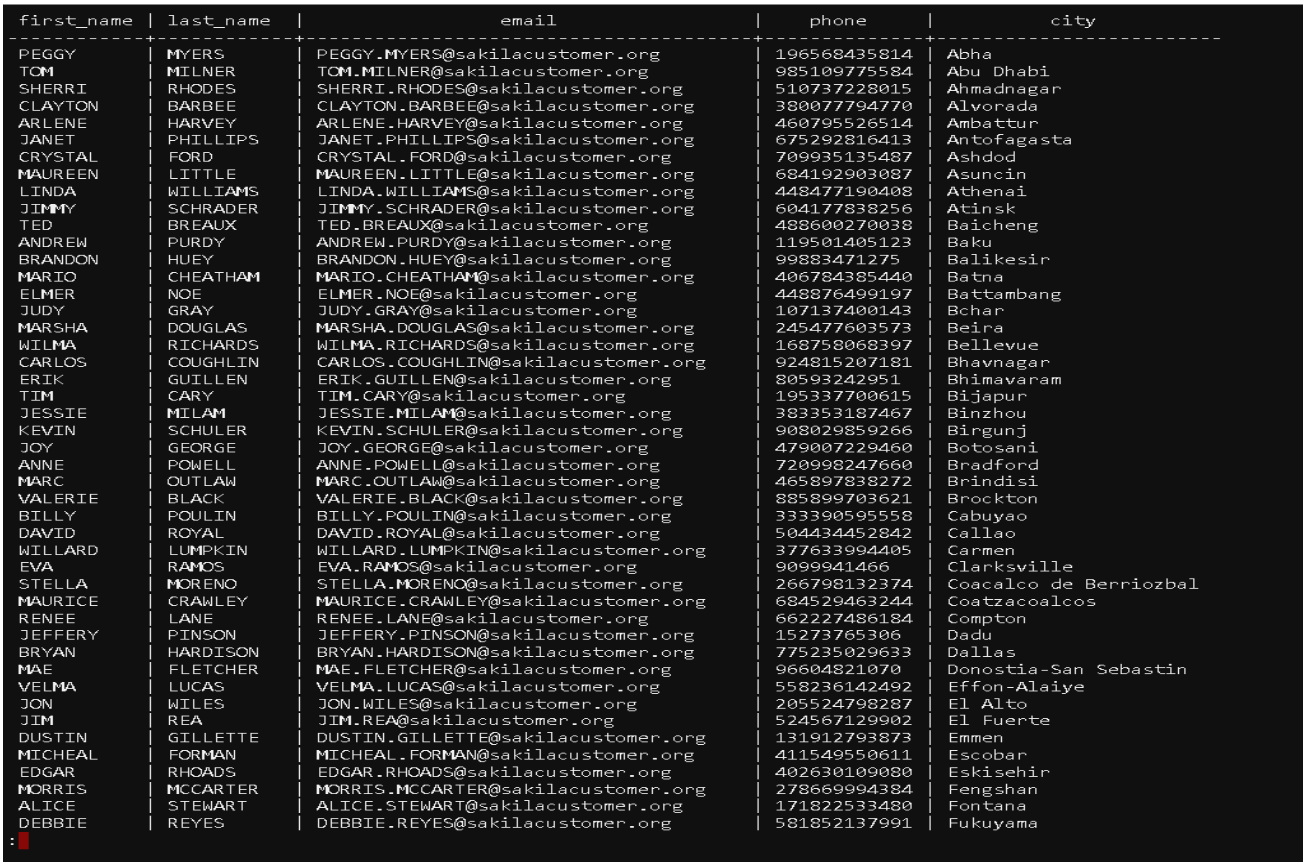
Q5. The manager is a huge Sci-Fi genre fan and would like to find which **customers have rented more than 2 movies** from this genre. Also he would like to run some statistics over the city the customer is from and their contact details (phone and email). Order by city and customer name.

A:

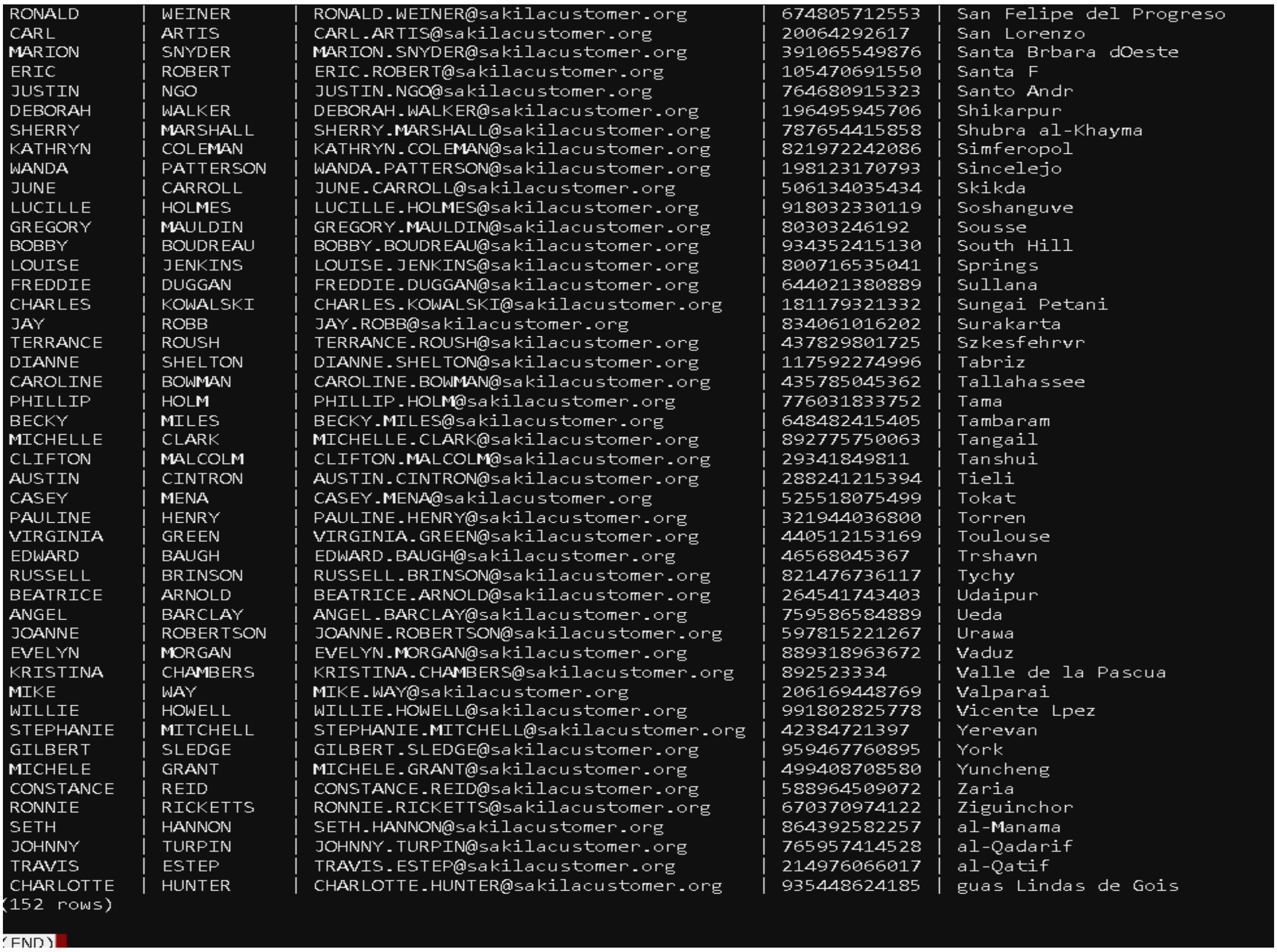
[EDITABLE CODE]

```
SELECT c.first_name, c.last_name, c.email, a.phone, ct.city
FROM customer c
JOIN rental r ON c.customer_id = r.customer_id
JOIN inventory i ON r.inventory_id = i.inventory_id
JOIN movie m ON i.movie_id = m.movie_id
JOIN movie_category mc ON m.movie_id = mc.movie_id
JOIN category cat ON mc.category_id = cat.category_id
JOIN address a ON c.address_id = a.address_id
JOIN city ct ON a.city_id = ct.city_id
WHERE cat.name = 'Sci-Fi'
GROUP BY c.first_name, c.last_name, c.email, a.phone, ct.city
HAVING COUNT(*) > 2
ORDER BY ct.city, c.first_name, c.last_name;
```

[OUTPUT SCREENSHOT]



```
 first_name | last_name |                 email                 |    phone     |           city
------------+-----------+---------------------------------------+--------------+-------------------------
 PEGGY      | MYERS     | PEGGY.MYERS@sakilacustomer.org        | 196568435814 | Abha
 TOM        | MILNER    | TOM.MILNER@sakilacustomer.org         | 985109775584 | Abu Dhabi
 SHERRI     | RHODES    | SHERRI.RHODES@sakilacustomer.org      | 510737228015 | Ahmadnagar
 CLAYTON    | BARBEE    | CLAYTON.BARBEE@sakilacustomer.org     | 380077794770 | Alvorada
 ARLENE     | HARVEY    | ARLENE.HARVEY@sakilacustomer.org      | 460795526514 | Ambattur
 JANET      | PHILLIPS  | JANET.PHILLIPS@sakilacustomer.org     | 675292816413 | Antofagasta
 CRYSTAL    | FORD      | CRYSTAL.FORD@sakilacustomer.org       | 709935135487 | Ashdod
 MAUREEN    | LITTLE    | MAUREEN.LITTLE@sakilacustomer.org     | 684192903087 | Asuncin
 LINDA      | WILLIAMS  | LINDA.WILLIAMS@sakilacustomer.org     | 448477190408 | Athenai
 JIMMY      | SCHRADER  | JIMMY.SCHRADER@sakilacustomer.org     | 604177838256 | Atinsk
 TED        | BREAUX    | TED.BREAUX@sakilacustomer.org         | 488600270038 | Baicheng
 ANDREW     | PURDY     | ANDREW.PURDY@sakilacustomer.org       | 119501405123 | Baku
 BRANDON    | HUEY      | BRANDON.HUEY@sakilacustomer.org       | 99883471275  | Balikesir
 MARIO      | CHEATHAM  | MARIO.CHEATHAM@sakilacustomer.org     | 406784385440 | Batna
 ELMER      | NOE       | ELMER.NOE@sakilacustomer.org          | 448876499197 | Battambang
 JUDY       | GRAY      | JUDY.GRAY@sakilacustomer.org          | 107137400143 | Bchar
 MARSHA     | DOUGLAS   | MARSHA.DOUGLAS@sakilacustomer.org     | 245477603573 | Beira
 WILMA      | RICHARDS  | WILMA.RICHARDS@sakilacustomer.org     | 168758068397 | Bellevue
 CARLOS     | COUGHLIN  | CARLOS.COUGHLIN@sakilacustomer.org    | 924815207181 | Bhavnagar
 ERIK       | GUILLEN   | ERIK.GUILLEN@sakilacustomer.org       | 80593242951  | Bhimavaram
 TIM        | CARY      | TIM.CARY@sakilacustomer.org           | 195337700615 | Bijapur
 JESSIE     | MILAM     | JESSIE.MILAM@sakilacustomer.org       | 383353187467 | Binzhou
 KEVIN      | SCHULER   | KEVIN.SCHULER@sakilacustomer.org      | 908029859266 | Birgunj
 JOY        | GEORGE    | JOY.GEORGE@sakilacustomer.org         | 479007229460 | Botosani
 ANNE       | POWELL    | ANNE.POWELL@sakilacustomer.org        | 720998247660 | Bradford
 MARC       | OUTLAW    | MARC.OUTLAW@sakilacustomer.org        | 465897838272 | Brindisi
 VALERIE    | BLACK     | VALERIE.BLACK@sakilacustomer.org      | 885899703621 | Brockton
 BILLY      | POULIN    | BILLY.POULIN@sakilacustomer.org       | 333390595558 | Cabuyao
 DAVID      | ROYAL     | DAVID.ROYAL@sakilacustomer.org        | 504434452842 | Callao
 WILLARD    | LUMPKIN   | WILLARD.LUMPKIN@sakilacustomer.org    | 377633994405 | Carmen
 EVA        | RAMOS     | EVA.RAMOS@sakilacustomer.org          | 9099941466   | Clarksville
 STELLA     | MORENO    | STELLA.MORENO@sakilacustomer.org      | 266798132374 | Coacalco de Berriozbal
 MAURICE    | CRAWLEY   | MAURICE.CRAWLEY@sakilacustomer.org    | 684529463244 | Coatzacoalcos
 RENEE      | LANE      | RENEE.LANE@sakilacustomer.org         | 662227486184 | Compton
 JEFFERY    | PINSON    | JEFFERY.PINSON@sakilacustomer.org     | 15273765306  | Dadu
 BRYAN      | HARDISON  | BRYAN.HARDISON@sakilacustomer.org     | 775235029633 | Dallas
 MAE        | FLETCHER  | MAE.FLETCHER@sakilacustomer.org       | 96604821070  | Donostia-San Sebastin
 VELMA      | LUCAS     | VELMA.LUCAS@sakilacustomer.org        | 558236142492 | Effon-Alaiye
 JON        | WILES     | JON.WILES@sakilacustomer.org          | 205524798287 | El Alto
 JIM        | REA       | JIM.REA@sakilacustomer.org            | 524567129902 | El Fuerte
 DUSTIN     | GILLETTE  | DUSTIN.GILLETTE@sakilacustomer.org    | 131912793873 | Emmen
 MICHEAL    | FORMAN    | MICHEAL.FORMAN@sakilacustomer.org     | 411549550611 | Escobar
 EDGAR      | RHOADS    | EDGAR.RHOADS@sakilacustomer.org       | 402630109080 | Eskisehir
 MORRIS     | MCCARTER  | MORRIS.MCCARTER@sakilacustomer.org    | 278669994384 | Fengshan
 ALICE      | STEWART   | ALICE.STEWART@sakilacustomer.org      | 171822533480 | Fontana
 DEBBIE     | REYES     | DEBBIE.REYES@sakilacustomer.org       | 581852137991 | Fukuyama
:
```



```
RONALD      | WEINER    | RONALD.WEINER@sakilacustomer.org      | 674805712553 | San Felipe del Progreso
CARL        | ARTIS     | CARL.ARTIS@sakilacustomer.org         | 20064292617  | San Lorenzo
MARION      | SNYDER    | MARION.SNYDER@sakilacustomer.org      | 391065549876 | Santa Brbara dOeste
ERIC        | ROBERT    | ERIC.ROBERT@sakilacustomer.org        | 105470691550 | Santa F
JUSTIN      | NGO       | JUSTIN.NGO@sakilacustomer.org         | 764680915323 | Santo Andr
DEBORAH     | WALKER    | DEBORAH.WALKER@sakilacustomer.org     | 196495945706 | Shikarpur
SHERRY      | MARSHALL  | SHERRY.MARSHALL@sakilacustomer.org    | 787654415858 | Shubra al-Khayma
KATHRYN     | COLEMAN   | KATHRYN.COLEMAN@sakilacustomer.org    | 821972242086 | Simferopol
WANDA       | PATTERSON | WANDA.PATTERSON@sakilacustomer.org    | 198123170793 | Sincelejo
JUNE        | CARROLL   | JUNE.CARROLL@sakilacustomer.org       | 506134035434 | Skikda
LUCILLE     | HOLMES    | LUCILLE.HOLMES@sakilacustomer.org     | 918032330119 | Soshanguve
GREGORY     | MAULDIN   | GREGORY.MAULDIN@sakilacustomer.org    | 80303246192  | Sousse
BOBBY       | BOUDREAU  | BOBBY.BOUDREAU@sakilacustomer.org     | 934352415130 | South Hill
LOUISE      | JENKINS   | LOUISE.JENKINS@sakilacustomer.org     | 800716535041 | Springs
FREDDIE     | DUGGAN    | FREDDIE.DUGGAN@sakilacustomer.org     | 644021380889 | Sullana
CHARLES     | KOWALSKI  | CHARLES.KOWALSKI@sakilacustomer.org   | 181179321332 | Sungai Petani
JAY         | ROBB      | JAY.ROBB@sakilacustomer.org           | 834061016202 | Surakarta
TERRANCE    | ROUSH     | TERRANCE.ROUSH@sakilacustomer.org     | 437829801725 | Szkesfehrvr
DIANNE      | SHELTON   | DIANNE.SHELTON@sakilacustomer.org     | 117592274996 | Tabriz
CAROLINE    | BOWMAN    | CAROLINE.BOWMAN@sakilacustomer.org    | 435785045362 | Tallahassee
PHILLIP     | HOLM      | PHILLIP.HOLM@sakilacustomer.org       | 776031833752 | Tama
BECKY       | MILES     | BECKY.MILES@sakilacustomer.org        | 648482415405 | Tambaram
MICHELLE    | CLARK     | MICHELLE.CLARK@sakilacustomer.org     | 892775750063 | Tangail
CLIFTON     | MALCOLM   | CLIFTON.MALCOLM@sakilacustomer.org    | 29341849811  | Tanshui
AUSTIN      | CINTRON   | AUSTIN.CINTRON@sakilacustomer.org     | 288241215394 | Tieli
CASEY       | MENA      | CASEY.MENA@sakilacustomer.org         | 525518075499 | Tokat
PAULINE     | HENRY     | PAULINE.HENRY@sakilacustomer.org      | 321944036800 | Torren
VIRGINIA    | GREEN     | VIRGINIA.GREEN@sakilacustomer.org     | 440512153169 | Toulouse
EDWARD      | BAUGH     | EDWARD.BAUGH@sakilacustomer.org       | 46568045367  | Trshavn
RUSSELL     | BRINSON   | RUSSELL.BRINSON@sakilacustomer.org    | 821476736117 | Tychy
BEATRICE    | ARNOLD    | BEATRICE.ARNOLD@sakilacustomer.org    | 264541743403 | Udaipur
ANGEL       | BARCLAY   | ANGEL.BARCLAY@sakilacustomer.org      | 759586584889 | Ueda
JOANNE      | ROBERTSON | JOANNE.ROBERTSON@sakilacustomer.org   | 597815221267 | Urawa
EVELYN      | MORGAN    | EVELYN.MORGAN@sakilacustomer.org      | 889318963672 | Vaduz
KRISTINA    | CHAMBERS  | KRISTINA.CHAMBERS@sakilacustomer.org  | 892523334    | Valle de la Pascua
MIKE        | WAY       | MIKE.WAY@sakilacustomer.org           | 206169448769 | Valparai
WILLIE      | HOWELL    | WILLIE.HOWELL@sakilacustomer.org      | 991802825778 | Vicente Lpez
STEPHANIE   | MITCHELL  | STEPHANIE.MITCHELL@sakilacustomer.org | 42384721397  | Yerevan
GILBERT     | SLEDGE    | GILBERT.SLEDGE@sakilacustomer.org     | 959467760895 | York
MICHELE     | GRANT     | MICHELE.GRANT@sakilacustomer.org      | 499408708580 | Yuncheng
CONSTANCE   | REID      | CONSTANCE.REID@sakilacustomer.org     | 588964509072 | Zaria
RONNIE      | RICKETTS  | RONNIE.RICKETTS@sakilacustomer.org    | 670370974122 | Ziguinchor
SETH        | HANNON    | SETH.HANNON@sakilacustomer.org        | 864392582257 | al-Manama
JOHNNY      | TURPIN    | JOHNNY.TURPIN@sakilacustomer.org      | 765957414528 | al-Qadarif
TRAVIS      | ESTEP     | TRAVIS.ESTEP@sakilacustomer.org       | 214976066017 | al-Qatif
CHARLOTTE   | HUNTER    | CHARLOTTE.HUNTER@sakilacustomer.org   | 935448624185 | guas Lindas de Gois
(152 rows)

(END)
```

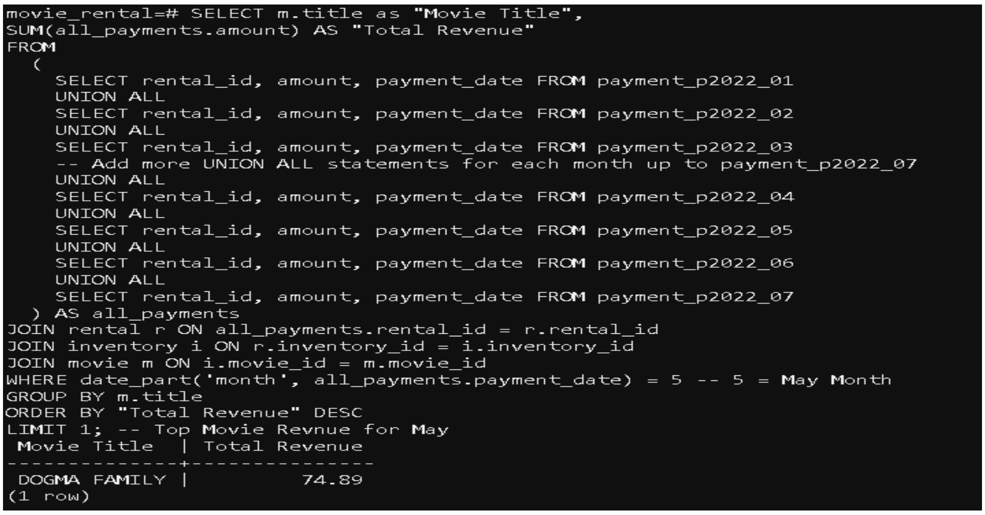
Q6. The business is on the right track, however it seems that in May a particular movie was quite popular generating a high revenue. What movie was and what was the amount generated?

A:

[EDITABLE CODE]

```
SELECT m.title as "Movie Title",
SUM(all_payments.amount) AS "Total Revenue"
FROM
  (
    SELECT rental_id, amount, payment_date FROM payment_p2022_01
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_02
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_03
    -- Add more UNION ALL statements for each month up to payment_p2022_07
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_04
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_05
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_06
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_07
  ) AS all_payments
JOIN rental r ON all_payments.rental_id = r.rental_id
JOIN inventory i ON r.inventory_id = i.inventory_id
JOIN movie m ON i.movie_id = m.movie_id
WHERE date_part('month', all_payments.payment_date) = 5 -- 5 = May Month
GROUP BY m.title
ORDER BY "Total Revenue" DESC
LIMIT 1; -- Top Movie Revenue for May
```

[OUTPUT SCREENSHOT]



```
movie_rental=# SELECT m.title as "Movie Title",
SUM(all_payments.amount) AS "Total Revenue"
FROM
  (
    SELECT rental_id, amount, payment_date FROM payment_p2022_01
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_02
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_03
    -- Add more UNION ALL statements for each month up to payment_p2022_07
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_04
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_05
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_06
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_07
  ) AS all_payments
JOIN rental r ON all_payments.rental_id = r.rental_id
JOIN inventory i ON r.inventory_id = i.inventory_id
JOIN movie m ON i.movie_id = m.movie_id
WHERE date_part('month', all_payments.payment_date) = 5 -- 5 = May Month
GROUP BY m.title
ORDER BY "Total Revenue" DESC
LIMIT 1; -- Top Movie Revnue for May
 Movie Title  | Total Revenue
--------------+---------------
 DOGMA FAMILY |         74.89
(1 row)
```

Q7. Now the manager wants to have a closer look at the general business income. He would like to see the ***top 5 categories total gross revenue for January per category***.

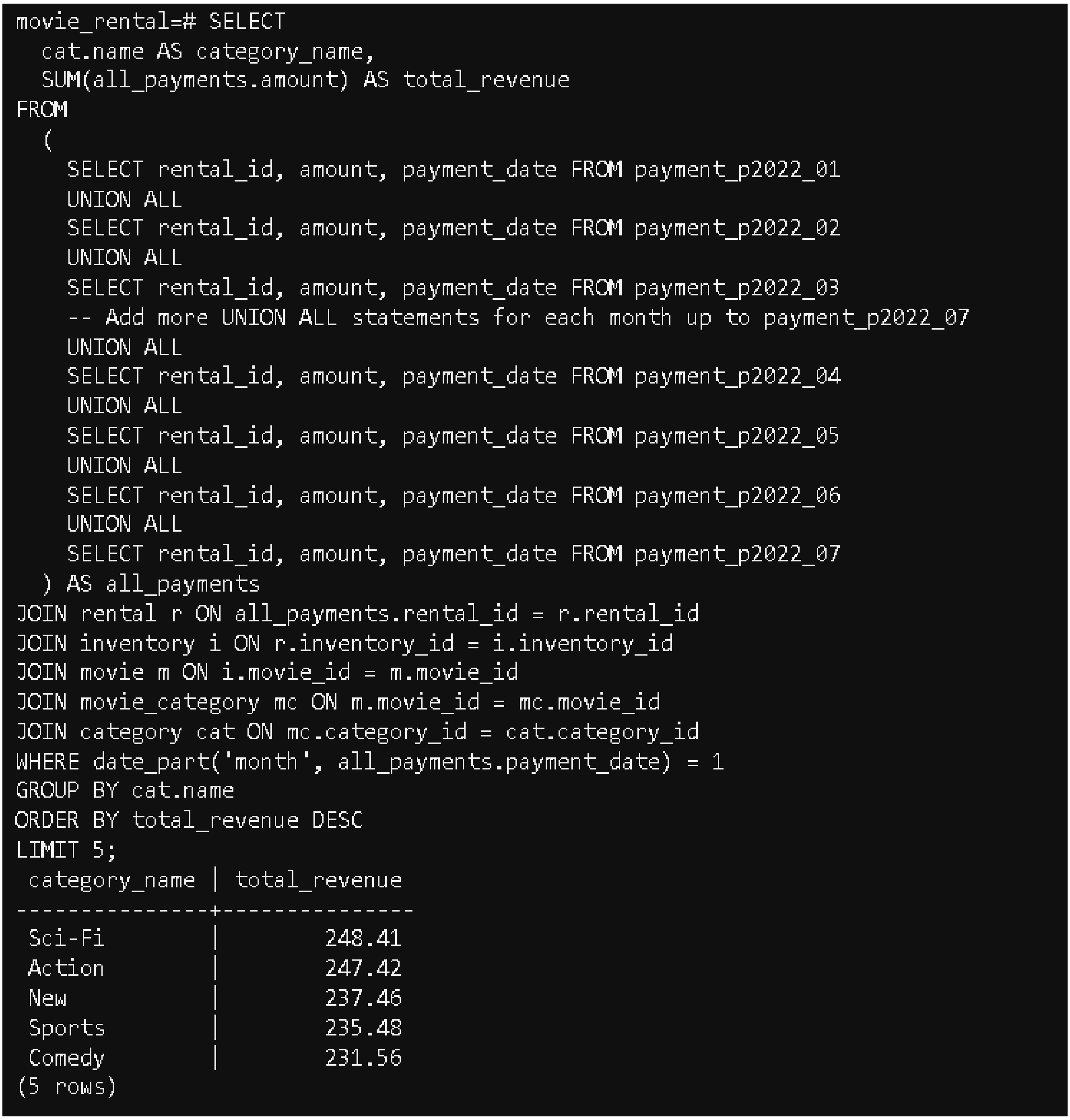
A:

[EDITABLE CODE]

```
SELECT
  cat.name AS category_name,
  SUM(all_payments.amount) AS total_revenue
FROM
  (
    SELECT rental_id, amount, payment_date FROM payment_p2022_01
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_02
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_03
    -- Add more UNION ALL statements for each month up to payment_p2022_07
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_04
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_05
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_06
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_07
  ) AS all_payments
JOIN rental r ON all_payments.rental_id = r.rental_id
JOIN inventory i ON r.inventory_id = i.inventory_id
JOIN movie m ON i.movie_id = m.movie_id
JOIN movie_category mc ON m.movie_id = mc.movie_id
JOIN category cat ON mc.category_id = cat.category_id
```

```
WHERE date_part('month', all_payments.payment_date) = 1
GROUP BY cat.name
ORDER BY total_revenue DESC
LIMIT 5;
```

[OUTPUT SCREENSHOT]



```
movie_rental=# SELECT
  cat.name AS category_name,
  SUM(all_payments.amount) AS total_revenue
FROM
  (
    SELECT rental_id, amount, payment_date FROM payment_p2022_01
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_02
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_03
    -- Add more UNION ALL statements for each month up to payment_p2022_07
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_04
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_05
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_06
    UNION ALL
    SELECT rental_id, amount, payment_date FROM payment_p2022_07
  ) AS all_payments
JOIN rental r ON all_payments.rental_id = r.rental_id
JOIN inventory i ON r.inventory_id = i.inventory_id
JOIN movie m ON i.movie_id = m.movie_id
JOIN movie_category mc ON m.movie_id = mc.movie_id
JOIN category cat ON mc.category_id = cat.category_id
WHERE date_part('month', all_payments.payment_date) = 1
GROUP BY cat.name
ORDER BY total_revenue DESC
LIMIT 5;
 category_name | total_revenue
---------------+---------------
 Sci-Fi        |        248.41
 Action        |        247.42
 New           |        237.46
 Sports        |        235.48
 Comedy        |        231.56
(5 rows)
```

Q8. The *Bottleneck* concept refers to:

a. A system that will allow easy query

b. Aspects that will limit database performance

c. Display critical business metrics and gather reports in one place

d. The absolute minimum interruption to the process
e. Process mapping the workflow visually to spot congestions

Q9. Performance of a databases is measured against:

a. Queries
b. Business rules
c. Software requirements
d. Benchmarks
e. Monitoring tools

Q10. How many tuning levels are for databases?

a. 6
b. 4
c. 2
d. 3
e. 1

Conclusion/Reflection

Lab 10 was an incredibly enriching exercise that solidified my understanding of database administration and its intersection with Linux command-line operations. Creating a database backup, deleting the database, and then restoring it from the backup file illustrated the importance of backup systems in real-world scenarios.

The lab also helped me appreciate how powerful SQL can be when it comes to extracting valuable business insights. By executing specific queries, I could determine patterns and trends that could potentially be leveraged for business decision-making and strategy planning.

Another important aspect that Lab 10 emphasized was the concept of database performance and bottleneck identification. The ability to measure and optimize database performance is a vital skill in the field of data management.

Overall, Lab 10 was a challenging but enlightening experience, effectively honing my database administration skills, enhancing my SQL expertise, and broadening my understanding of real-world database use-cases. This knowledge and these skills will undoubtedly prove to be crucial in my future data-related endeavors.

LAB 11 (26/07/2023) - Christmas Fun

Lab Description

Lab 11 was an engaging exploration into advanced PostgreSQL database management and SQL queries. It covered a wide range of topics including recursive CTEs, joins, subqueries, full-text search, and query optimization. Implemented these techniques on the Movie Rental database, which provided a practical and enriching environment for improving query writing and data manipulation skills.

In addition to these advanced techniques, also introduced the fascinating world of PostgreSQL tricks such as generating automatic number series, creating festive graphical outputs like Christmas trees and forests, generating random numbers, and simulating UUIDs.

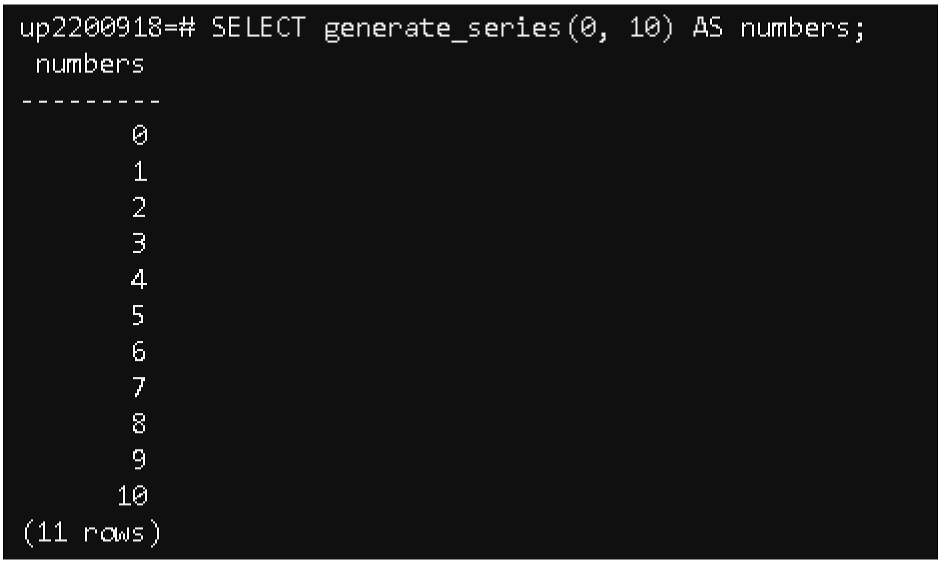
Q1. Create an automatic series of numbers from 0 to 10. The output should be generated by PgSQL without any database, table or insert. No specific database is needed.

A:

[EDITABLE CODE]

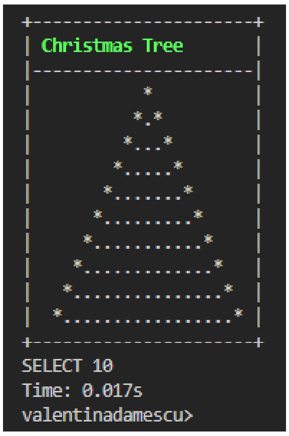
```
SELECT generate_series(0, 10) AS numbers;
```

[OUTPUT SCREENSHOT]



```
up2200918=# SELECT generate_series(0, 10) AS numbers;
 numbers
---------
       0
       1
       2
       3
       4
       5
       6
       7
       8
       9
      10
(11 rows)
```

Q2. We want to get festive with our PostgreSQL. Can you generate a Christmas Tree like below? No specific database, table or insert is required.



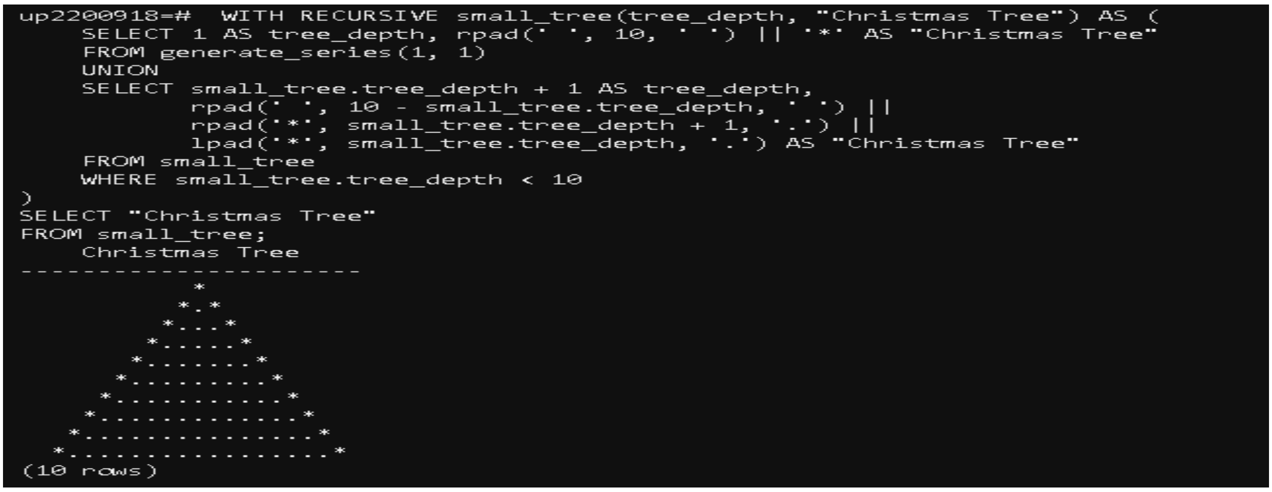
```
+----------------------+
| Christmas Tree       |
|----------------------|
|           *          |
|          *.*         |
|         *...*        |
|        *.....*       |
|       *.......*      |
|      *.........*     |
|     *...........*    |
|    *.............*   |
|   *...............*  |
|  *.................* |
+----------------------+
SELECT 10
Time: 0.017s
valentinadamescu>
```

A:

[EDITABLE CODE]

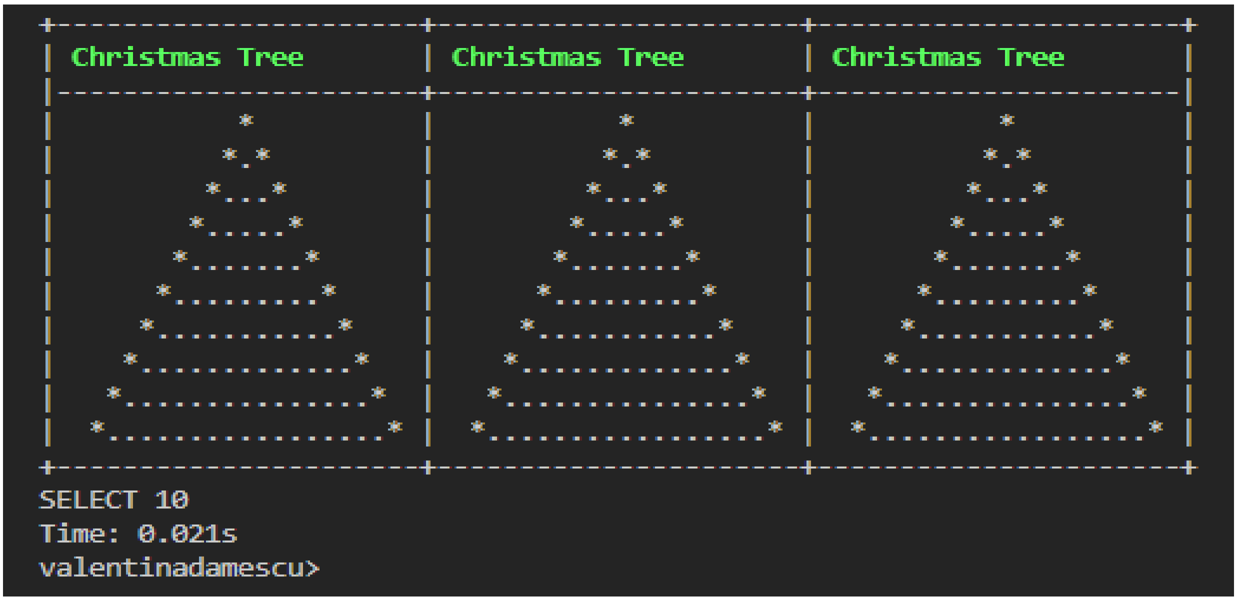
```sql
WITH RECURSIVE small_tree(tree_depth, "Christmas Tree") AS (
    SELECT 1 AS tree_depth, rpad(' ', 10, ' ') || '*' AS "Christmas Tree"
    FROM generate_series(1, 1)
    UNION
    SELECT small_tree.tree_depth + 1 AS tree_depth,
           rpad(' ', 10 - small_tree.tree_depth, ' ') ||
           rpad('*', small_tree.tree_depth + 1, '.') ||
           lpad('*', small_tree.tree_depth, '.') AS "Christmas Tree"
    FROM small_tree
    WHERE small_tree.tree_depth < 10
)
SELECT "Christmas Tree"
FROM small_tree;
```

[OUTPUT SCREENSHOT]



```
up2200918=#  WITH RECURSIVE small_tree(tree_depth, "Christmas Tree") AS (
    SELECT 1 AS tree_depth, rpad(' ', 10, ' ') || '*' AS "Christmas Tree"
    FROM generate_series(1, 1)
    UNION
    SELECT small_tree.tree_depth + 1 AS tree_depth,
           rpad(' ', 10 - small_tree.tree_depth, ' ') ||
           rpad('*', small_tree.tree_depth + 1, '.') ||
           lpad('*', small_tree.tree_depth, '.') AS "Christmas Tree"
    FROM small_tree
    WHERE small_tree.tree_depth < 10
)
SELECT "Christmas Tree"
FROM small_tree;
    Christmas Tree
----------------------
           *
          *.*
         *...*
        *.....*
       *.......*
      *.........*
     *...........*
    *.............*
   *...............*
  *.................*
(10 rows)
```

Q3. How about a small forest of them?



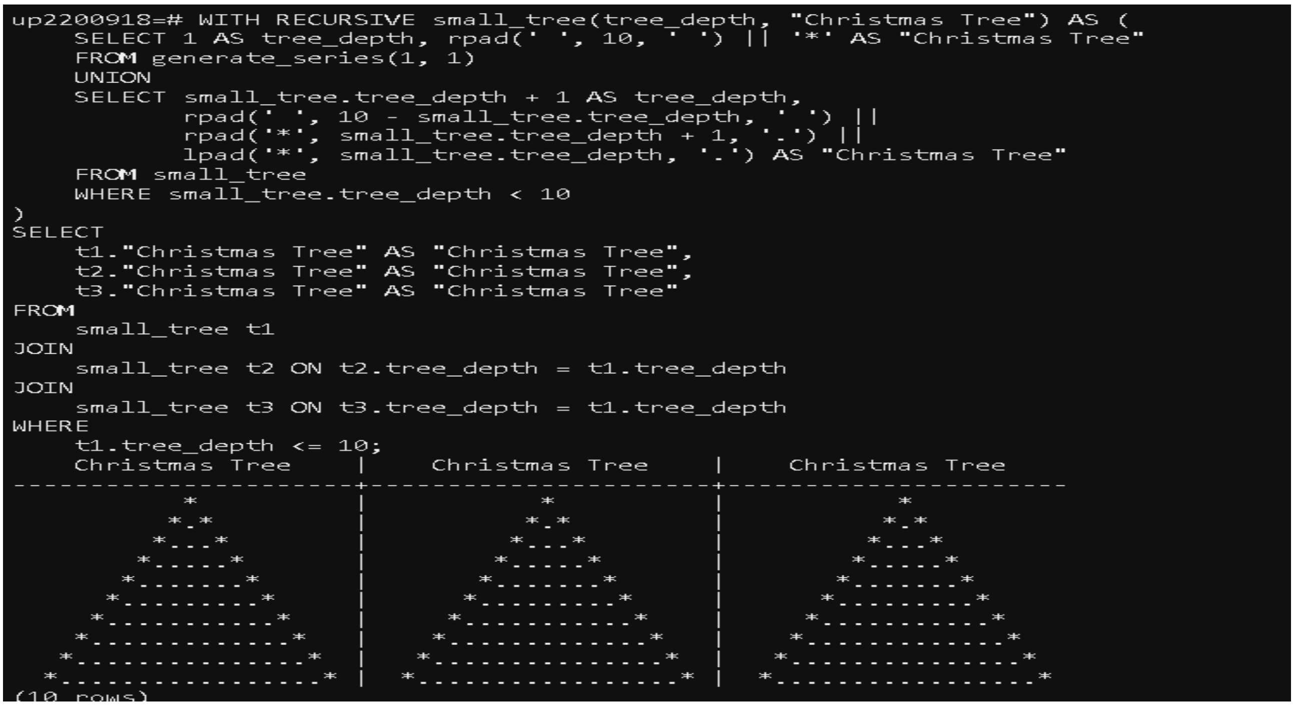
```
+----------------------+----------------------+----------------------+
| Christmas Tree       | Christmas Tree       | Christmas Tree       |
|----------------------+----------------------+----------------------|
|           *          |           *          |           *          |
|          *.*         |          *.*         |          *.*         |
|         *...*        |         *...*        |         *...*        |
|        *.....*       |        *.....*       |        *.....*       |
|       *.......*      |       *.......*      |       *.......*      |
|      *.........*     |      *.........*     |      *.........*     |
|     *...........*    |     *...........*    |     *...........*    |
|    *.............*   |    *.............*   |    *.............*   |
|   *...............*  |   *...............*  |   *...............*  |
|  *.................* |  *.................* |  *.................* |
+----------------------+----------------------+----------------------+
SELECT 10
Time: 0.021s
valentinadamescu>
```

A:

[EDITABLE CODE]

```sql
WITH RECURSIVE small_tree(tree_depth, "Christmas Tree") AS (
    SELECT 1 AS tree_depth, rpad(' ', 10, ' ') || '*' AS "Christmas Tree"
    FROM generate_series(1, 1)
    UNION
    SELECT small_tree.tree_depth + 1 AS tree_depth,
           rpad(' ', 10 - small_tree.tree_depth, ' ') ||
           rpad('*', small_tree.tree_depth + 1, '.') ||
           lpad('*', small_tree.tree_depth, '.') AS "Christmas Tree"
    FROM small_tree
    WHERE small_tree.tree_depth < 10
)
SELECT
    t1."Christmas Tree" AS "Christmas Tree",
    t2."Christmas Tree" AS "Christmas Tree",
    t3."Christmas Tree" AS "Christmas Tree"
FROM
    small_tree t1
JOIN
    small_tree t2 ON t2.tree_depth = t1.tree_depth
JOIN
    small_tree t3 ON t3.tree_depth = t1.tree_depth
WHERE
    t1.tree_depth <= 10;
```

[OUTPUT SCREENSHOT]



```
up2200918=# WITH RECURSIVE small_tree(tree_depth, "Christmas Tree") AS (
    SELECT 1 AS tree_depth, rpad(' ', 10, ' ') || '*' AS "Christmas Tree"
    FROM generate_series(1, 1)
    UNION
    SELECT small_tree.tree_depth + 1 AS tree_depth,
           rpad(' ', 10 - small_tree.tree_depth, ' ') ||
           rpad('*', small_tree.tree_depth + 1, '.') ||
           lpad('*', small_tree.tree_depth, '.') AS "Christmas Tree"
    FROM small_tree
    WHERE small_tree.tree_depth < 10
)
SELECT
    t1."Christmas Tree" AS "Christmas Tree",
    t2."Christmas Tree" AS "Christmas Tree",
    t3."Christmas Tree" AS "Christmas Tree"
FROM
    small_tree t1
JOIN
    small_tree t2 ON t2.tree_depth = t1.tree_depth
JOIN
    small_tree t3 ON t3.tree_depth = t1.tree_depth
WHERE
    t1.tree_depth <= 10;
    Christmas Tree    |    Christmas Tree    |    Christmas Tree
----------------------+----------------------+----------------------
           *          |           *          |           *
          *.*         |          *.*         |          *.*
         *...*        |         *...*        |         *...*
        *.....*       |        *.....*       |        *.....*
       *.......*      |       *.......*      |       *.......*
      *.........*     |      *.........*     |      *.........*
     *...........*    |     *...........*    |     *...........*
    *.............*   |    *.............*   |    *.............*
   *...............*  |   *...............*  |   *...............*
  *.................* |  *.................* |  *.................*
(10 rows)
```

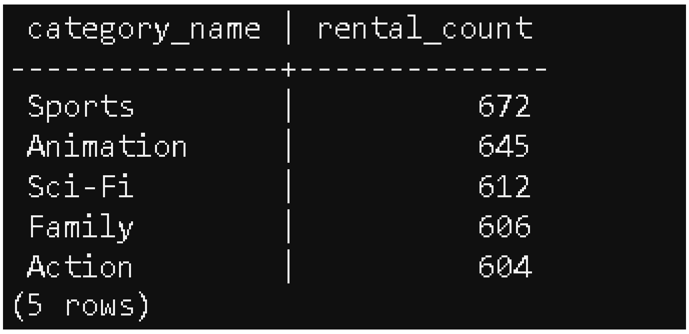
Q4. Put all your SQL knowledge at work and based on the Movie Rental database, what is the best query you can create?

A:

[EDITABLE CODE]

```
-- Top 5 most popular movie categories (genres) based on
the number of rentals in the past year:
SELECT
  cat.name AS category_name,
  COUNT(*) AS rental_count
FROM
  rental r
JOIN inventory i ON r.inventory_id = i.inventory_id
JOIN movie m ON i.movie_id = m.movie_id
JOIN movie_category mc ON m.movie_id = mc.movie_id
JOIN category cat ON mc.category_id = cat.category_id
WHERE
  r.rental_date >= CURRENT_DATE - INTERVAL '1 year'
GROUP BY
  cat.name
ORDER BY
  rental_count DESC
LIMIT 5;
```

[OUTPUT SCREENSHOT]



```
 category_name | rental_count
---------------+--------------
 Sports        |          672
 Animation     |          645
 Sci-Fi        |          612
 Family        |          606
 Action        |          604
(5 rows)
```

Q5. What “cool tricks” can you show with PostgreSQL? Similar to Q1 - Q3?

A:

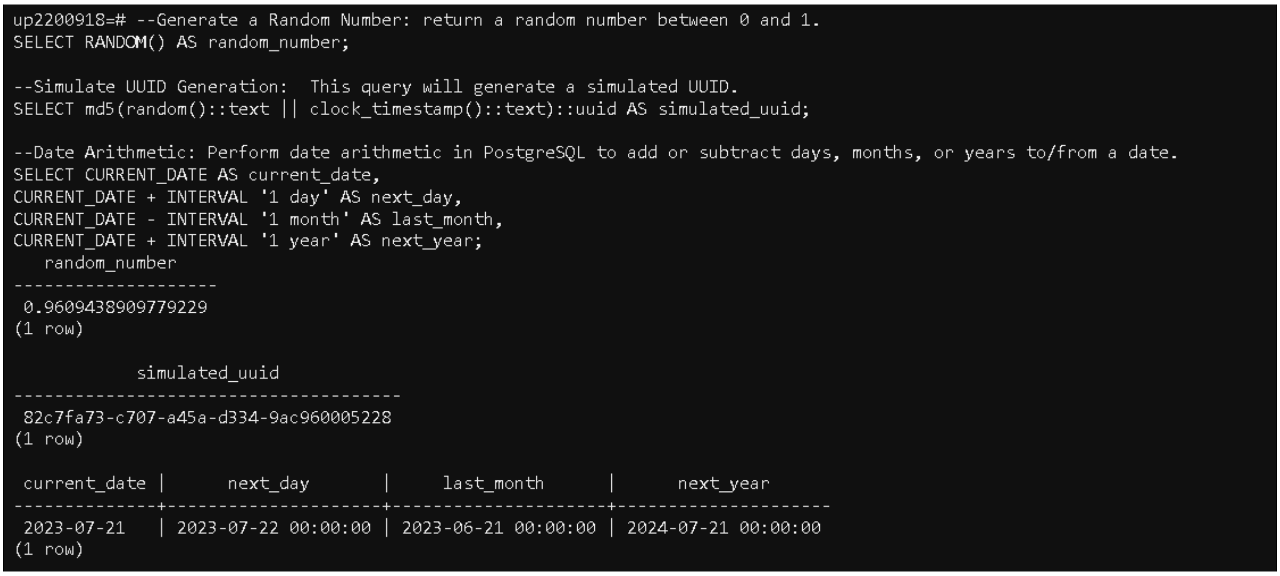
[EDITABLE CODE]

```
--Generate a Random Number: return a random number between 0 and 1.
SELECT RANDOM() AS random_number;

--Simulate UUID Generation:  This query will generate a simulated UUID.
SELECT md5(random()::text || clock_timestamp()::text)::uuid AS
simulated_uuid;

--Date Arithmetic: Perform date arithmetic in PostgreSQL to add or subtract
days, months, or years to/from a date.
SELECT CURRENT_DATE AS current_date,
CURRENT_DATE + INTERVAL '1 day' AS next_day,
CURRENT_DATE - INTERVAL '1 month' AS last_month,
CURRENT_DATE + INTERVAL '1 year' AS next_year;
```

[OUTPUT SCREENSHOT]



```
up2200918=# --Generate a Random Number: return a random number between 0 and 1.
SELECT RANDOM() AS random_number;

--Simulate UUID Generation:  This query will generate a simulated UUID.
SELECT md5(random()::text || clock_timestamp()::text)::uuid AS simulated_uuid;

--Date Arithmetic: Perform date arithmetic in PostgreSQL to add or subtract days, months, or years to/from a date.
SELECT CURRENT_DATE AS current_date,
CURRENT_DATE + INTERVAL '1 day' AS next_day,
CURRENT_DATE - INTERVAL '1 month' AS last_month,
CURRENT_DATE + INTERVAL '1 year' AS next_year;
   random_number
--------------------
 0.9609438909779229
(1 row)

            simulated_uuid
--------------------------------------
 82c7fa73-c707-a45a-d334-9ac960005228
(1 row)

 current_date |      next_day       |     last_month      |      next_year
--------------+---------------------+---------------------+---------------------
 2023-07-21   | 2023-07-22 00:00:00 | 2023-06-21 00:00:00 | 2024-07-21 00:00:00
(1 row)
```

Conclusion/Reflection

Lab 11 was a valuable experience that deepened my understanding of PostgreSQL and advanced SQL techniques. I found the topic of recursive CTEs intriguing and the opportunity to use this to generate graphical outputs like Christmas trees was fun and enlightening. It was fascinating to see how PostgreSQL can be used creatively, beyond the traditional database operations.

Working on complex queries with the Movie Rental database also allowed me to appreciate the importance of query optimization in real-world scenarios. Extracting the most popular movie categories based on the number of rentals in the past year showed how SQL can provide valuable business insights.

The "cool tricks" with PostgreSQL was another highlight of Lab 11. Generating random numbers and simulating UUIDs are techniques that I can see being quite handy in many real-world applications.

Overall, Lab 11 was a rewarding and enriching experience that improved my SQL skills, broadened my understanding of advanced database management, and exposed me to the unique capabilities of PostgreSQL. I feel more confident in handling complex database operations and optimizing performance for real-world database scenarios.