

LAB 1 (1/10/2023)

Lab Description

This lab follows up on what was covered in the first lecture, a recap of last year's content.

— LAB ANSWERS —

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

- Develop the initial ERD.
- Create a detailed narrative of the organisation's description of operations.
- Identify the attributes and primary keys that adequately describe the entities.
- Identify the business rules based on the description of operations.







A: b

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

- Many-to-many (M:M)
- One-to-many (1:M)
- One-to-one (1:1)
- Many-to-one (M:1)
- Many-to-zero OR one (M:0-1)
- One-to-one OR many (1:1-M)
- Many-to-zero

A: a

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

- 
- 
- 
- 
- 
- 

A: c

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

- An ER model is concerned primarily with the view of the attributes that will be used in physical implementation
- An ER model is concerned primarily with a physical implementation of the data and secondly with the logical view
- An ER model provides a view of the logic of the data and not the physical implementation
- An ER model is entirely concerned with modelling the physical implementation

A: c

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

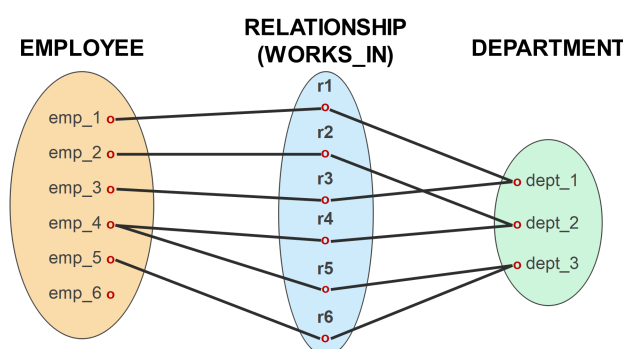


Figure.1

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

- 1:1** relationship between EMPLOYEE and DEPARTMENT and total participation from EMPLOYEE and total participation from DEPARTMENT
- 1:M** relationship between EMPLOYEE and DEPARTMENT and partial participation from EMPLOYEE and partial participation from DEPARTMENT
- M:M** relationship between EMPLOYEE and DEPARTMENT and partial participation from EMPLOYEE and total participation from DEPARTMENT
- M:1** relationship between EMPLOYEE and DEPARTMENT and total participation from EMPLOYEE and total participation from DEPARTMENT

A: c

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).
```

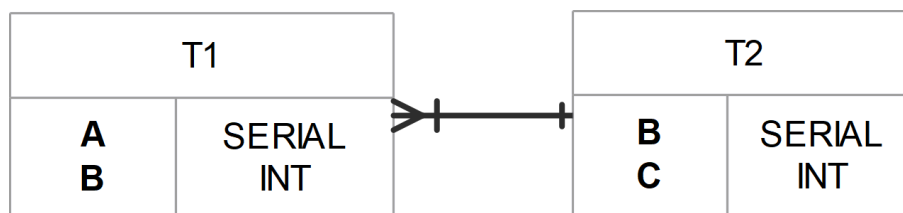


Figure. 2

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

- `CREATE TABLE T1(A SERIAL PRIMARY KEY UNIQUE, B INT, FOREIGN KEY NOT NULL);`
- `CREATE TABLE T1(A SERIAL PRIMARY KEY, B INT NOT NULL REFERENCES T2(B));`
- `CREATE TABLE T1(A SERIAL UNIQUE, B REFERENCES T2(B));`
- `CREATE TABLE T1(A SERIAL, B INT);`

A: b

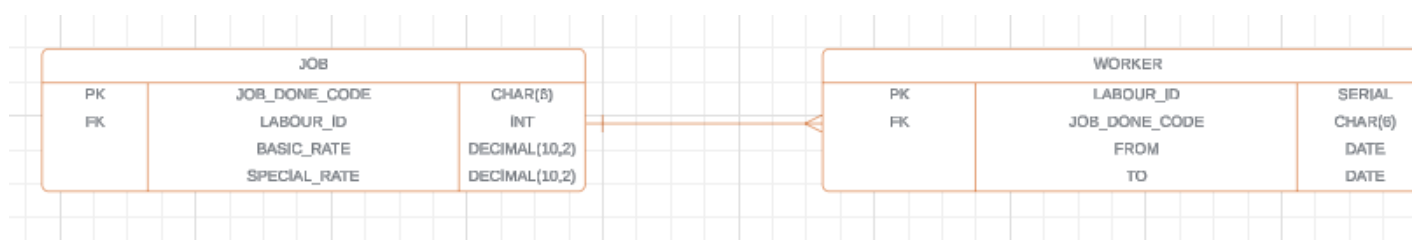
Q7. In a manufacturing industry labourers are given different jobs on different days and each job has its own monthly basic and monthly special rates as wages to be paid to labourers. A worker is not given more than one type of job on a day. A database designer is given the job to design database for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

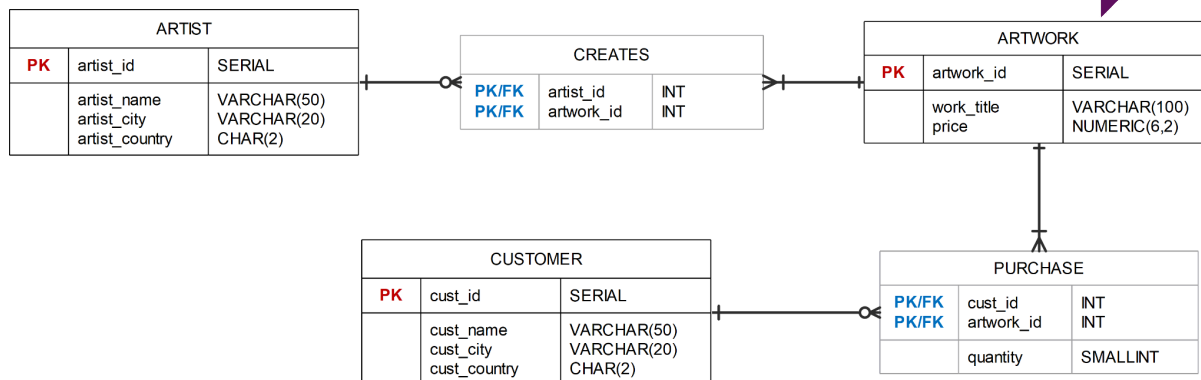
FIELD	TYPE	REMARKS
From date	DATE	From this date to
To date	DATE	This date
Labour ID	SERIAL	The worker ID
Job done code	CHAR(6)	The job ID
At Basic Rate	DECIMAL(10,2)	At this basic rate
At Special Rate	DECIMAL(10,2)	At this special rate

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).

A: Type of Job to Worker = One to Many





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:

```

SELECT artist_name AS "Artist Name"
FROM artist a
JOIN creates c ON c.artist_id = a.artist_id
JOIN artwork ak ON ak.artwork_id = c.artwork_id
WHERE work_title = 'Rainbow';
  
```

```

Artist Name
-----
Ringo
(1 row)
  
```

Q9. Find the titles of the Artworks created by '**Lolo**'.

A:

```

SELECT work_title AS "Art Title"
FROM artwork ak
JOIN creates c ON c.artwork_id = ak.artwork_id
JOIN artist a ON a.artist_id = c.artist_id
WHERE artist_name = 'Lolo';
  
```

```

    Art 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 name, artwork title and the price. Add the £ sign to the output and order on price from lowest to highest..

A:

```

SELECT c.cust_name AS "Customer Name",
       COALESCE(a.work_title, 'No Purchases') AS "Artwork Title",
       CONCAT('£', COALESCE(a.price, 0)) AS "Price"
FROM customer c
LEFT JOIN purchase p ON p.cust_id = c.cust_id
LEFT JOIN artwork a ON a.artwork_id = p.artwork_id
WHERE c.cust_id NOT IN (
    SELECT DISTINCT p.cust_id
    FROM purchase p
    JOIN artwork a ON a.artwork_id = p.artwork_id
    WHERE a.price < 200
)
ORDER BY a.price ASC;

```

Customer Name	Artwork Title	Price
Harry	My lovely song	£200.00
Harry	Lady in the sun	£250.00
Sarah	No Purchases	£0
Adda	No Purchases	£0

(4 rows)

And you continue following the same pattern

LAB 2 - Normalisation

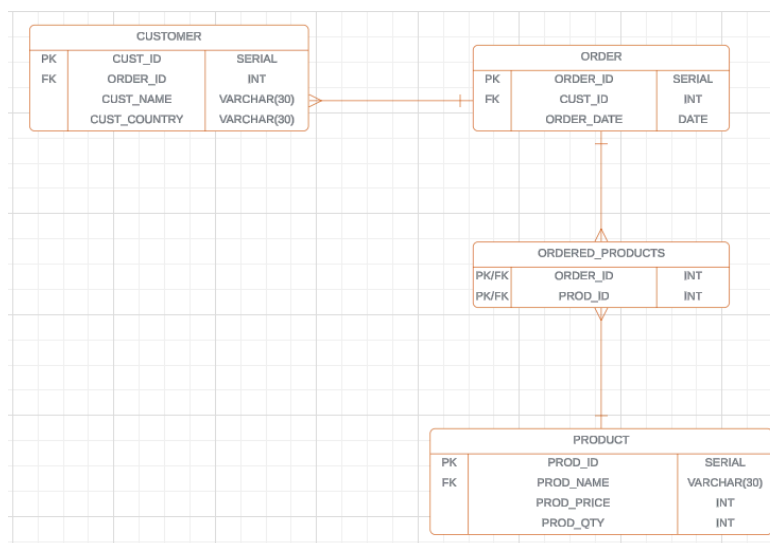
order_id	order_date	cust_id	cust_name	cust_country	prod_id	prod_name	prod_price	prod_qty
1001	01/10/2022	1	Apple	US	7, 5, 4	table, desk, chair	800, 325, 200	1, 1, 5
1001	01/10/2022		Apple					
1001	01/10/2022		Apple					
1002	02/10/2022	2	Samsung	KO	11, 4	dresser, chair	500, 200	4, 2
1002	02/10/2022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

Answer:

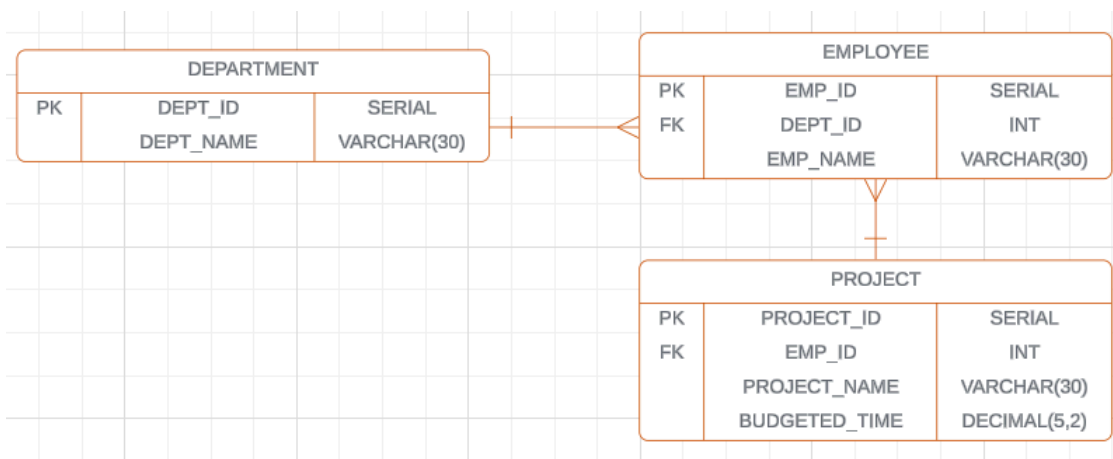
order_id	order_date	cust_id	cust_name	cust_country	prod_id	prod_name	prod_price	prod_qty
1001	01/10/2022	1	Apple	US	7	Table	800	1
1001	01/10/2022	1	Apple	US	5	Desk	325	1
1001	01/10/2022	1	Apple	US	4	Chair	200	5
1002	02/10/2022	2	Samsung	KO	11	Dresser	500	4
1002	02/10/2022	2	Samsung	KO	4	Chair	200	2
1003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



Q2. You are given the following table in 1NF. Normalise data in 3NF and create the ERD with all data type and size.

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

Answer:



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)

in a relation:

Student(up_number, stu_name, stu_addr, tut_id, tut_name, course_id, course_name, (module_code, module_name, module_results))

The functional dependencies are:

course_code → course_name
 tut_id → tut_name
 up_number, module_id → module_results
 module_code → module_name

Which of the following is a first step of the normalisation of the relation (table) **Student** to **2NF**?

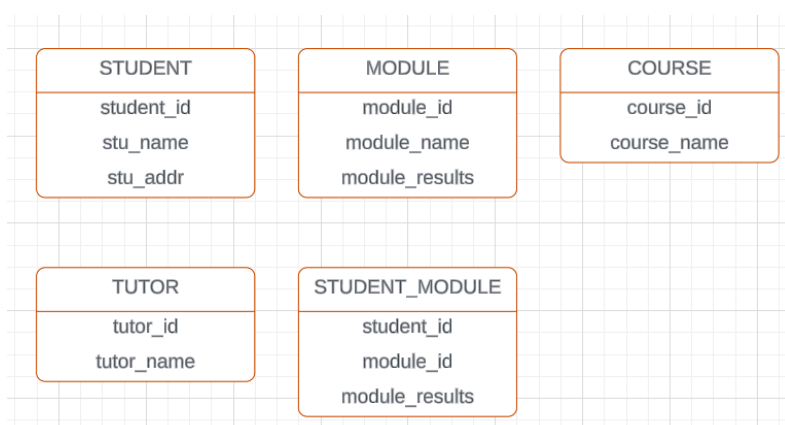
Hint: you can input a quick sample output in a spreadsheet (like Q1) and remove the repeating group.

- STUDENT**(up_number, stu_name, stu_addr, tutor_id, tutor_name, course_id, course_name)
MODULE(up_number, module_id, module_name, module_results)
- STUDENT**(up_number, stu_name, stu_addr, tutor_id, tutor_name, course_id, course_name, (module_code, module_results))
MODULE (module_code, module_name)
- STUDENT**(up_number, stu_name, stu_addr, tutor_id, tutor_name, course_id, (module_id, module_name, module_results))
COURSE(course_id, course_name)
- STUDENT**(student_id, stu_name, stu_addr, tutor_id, tutor_name, course_id)
MODULE(student_id, module_id, module_name, module_results)
COURSE(course_id, course_name)

Answer = d

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

Answer = Student_Module table is created to store every result a student would have for their various modules

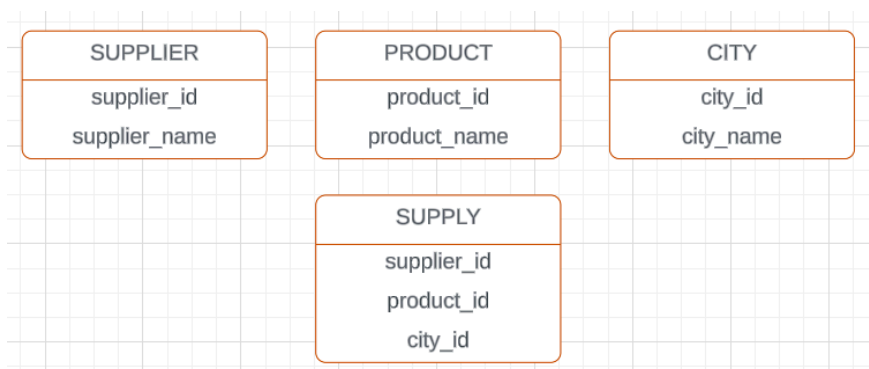


Q5. You have a SUPPLY table that records details about suppliers, products, and cities where the products are supplied. Acknowledge that a supplier will supply in many cities and a city will have many suppliers.

SUPPLY(supplier_id, supplier_name, product_id, product_name, city_id, city_name)

Normalize the table into 3NF. Assume that the table is already in 1NF, explain why certain attributes had to be separated.

Answer = Supply table was created as Supplier would have a M:M relationship with product and product the same with city and supplier the same with city too. As a intersection table linking the three of them is created

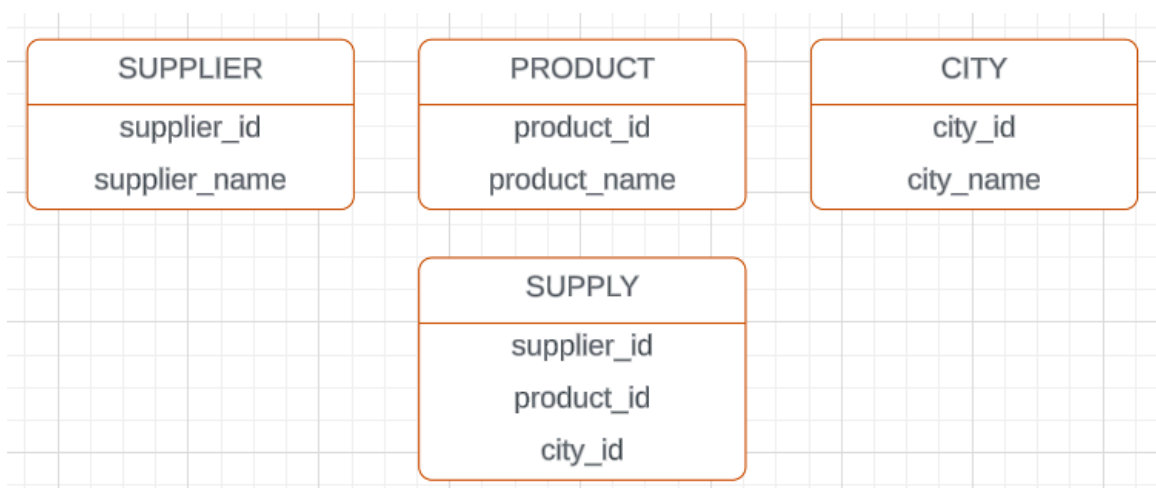


Q6. Consider the following ORDER table that records:

ORDER(order_id, customer_id, customer_name, product_id, product_name, quantity, date)

Normalise the ORDER table to 3NF.

Answer = While product holds all possible products that CAN be ordered, ordered products are put into the ordered_product table to establish which order it belongs in, the product, and how many of the product has been ordered. This is an intersection table created to handle the many to many relationship between products and order tables.



Q7. Open your VM and create a new database (lab2). You are given the following data dictionary.

ANIMAL_TYPE						
Attribute 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 scientific name of the animal
conservation_status	VARCHAR	50			NOT NULL	('Endangered', 'Least Concern')

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

and the data sample output:

animal_type_id	common_name	scientific_name	conservation_status
1	Bengal Tiger	Panthera tigris tigris	Endangered
2	Arctic Wolf	Canis lupus arctos	Least Concern

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

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

ANSWER =

```

CREATE TABLE animal_type(
    animal_type_id SERIAL 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 PRIMARY KEY,
    common_name VARCHAR(50) 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,
    FOREIGN KEY (common_name) REFERENCES animal_type(common_name)
);

INSERT INTO animal_type
    (common_name, scientific_name, conservation_status)
VALUES
    ('Bengal Tiger', 'Panthera tigris tigris', 'Endangered'),
    ('Arctic Wolf', 'Canis lupus arctos', 'Least Concern');

INSERT INTO menagerie
    (common_name, date_aquired, gender, aquired_from, name, notes)
VALUES
    ('Bengal Tiger', '2011/07/14', 'F', 'Dhaka Zoo', 'Ariel', 'Healthy coat at last exam.'),
    ('Arctic Wolf', '2008/09/30', 'F', 'National Zoo', 'Freddy', 'Strong appetite.'),
    ('Bengal Tiger', '2006/06/01', 'M', 'Scotland Zoo', 'Spark', 'Likes to play'),
    ('Arctic Wolf', '2007/06/12', 'F', 'Southampton National Park', 'Mia', 'Doesn't like sun');

```

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)

menagerie_id	common_name	date_aquired	gender	aquired_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

(4 rows)

Q8. 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.

Answer:

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

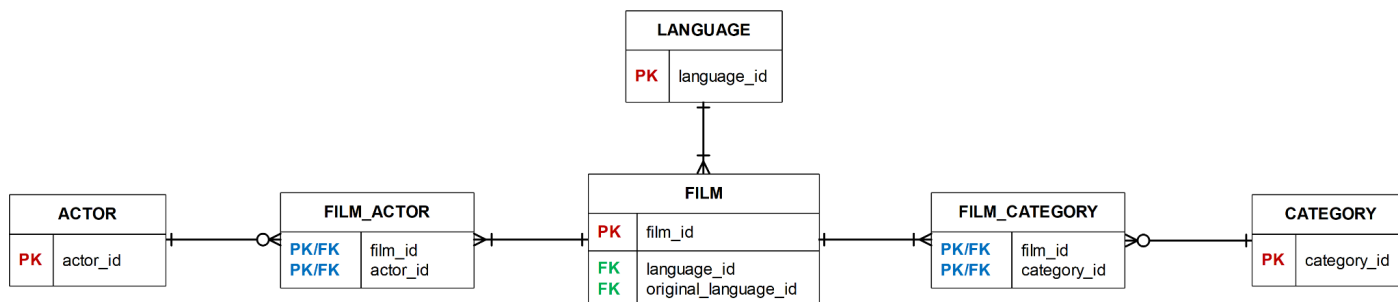
Common Name	Scientific Name	Animal Name	Date Acquired
Bengal Tiger	Panthera tigris tigris	Ariel	2011-07-14
Bengal Tiger	Panthera tigris tigris	Spark	2006-06-01

LAB 3 (18/10/2023)

— LAB ANSWERS —

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 should add some attributes to make the database functional (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';
  
```

A:

```

INSERT INTO actor
    (first_name, last_name, age)
VALUES
    ('Andrew', 'Garfield', 41);

SELECT
    f.title AS "Film",
    f.release_year AS "Release Year"
FROM film f
JOIN film_actor fa ON f.film_id = fa.film_id
WHERE fa.actor_id = 1;

```

Film	Release Year
Avengers: Endgame (French)	2019
Spider-Man: Homecoming (English)	2017
Captain America: Civil War (French)	2016

(3 rows)

```

UPDATE film
SET title = CONCAT(title, ' (' , l.language_name, ')')
FROM language l
WHERE film.language_id = l.language_id;

```

```

DELETE FROM film_actor
WHERE actor_id IN (
    SELECT actor_id
    FROM actor
    WHERE age < 40
);

```

```

DELETE FROM actor
WHERE age < 40;

```

TRANSACTION /	ACTOR	FILM_ACTOR	FILM	LANGUAGE	FILM_CATEGORY	CATEGORY
---------------	-------	------------	------	----------	---------------	----------

TABLE																								
	C	R	U	D	C	R	U	D	C	R	U	D	C	R	U	D	C	R	U	D	C	R	U	D
Insert New Actor	X	x																						
Read Actor's Films		x				x																		
Update Movie Title Based On Language										x	x			X										
Delete Actor		x		x		x		x																

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 = Where a transaction is completed simultaneously and successfully or not executed at all and if the transaction fails the database goes back to its former state before the transaction began.
- Consistency = Where after a transaction, all data still follows the rules and constraints of the database, so the database keeps its valid state from before the transaction.
- Isolation = Where transactions occur independently, unaffected by other transactions that may occur simultaneously. Even if transactions occur at the same time, concurrency control ensures that the system treats it as if it was sequential.
- Durability = Where once a transaction is completed, its results are permanently saved in the database, remaining intact even if the system crashes.
- Sequence = Where transactions are arranged in a sequential order, illustrating the order of operations to manage concurrency control and dictate how data is processed.
- Blind Write = Where a transaction updates specific data without reading its value beforehand.
- Dirty Read = Where a transaction reads data that is currently being updated by another ongoing transaction that hasn't been committed yet.
- Unrepeatable Read = Where a transaction reads the same data multiple times but the value differs each time due to updates made by another transaction in between the reads.
- Serializable Schedule = Where transactions are executed in a manner that ensures the final outcome result is the same as if they had been processed sequentially.
- Recoverable Schedule = Where if a transaction fails, the database will be restored to its state before the transaction, ensuring consistency is maintained.
- Avoids-Cascading-Aborts Schedule = Where the failure of one transaction doesn't trigger a chain reaction, preventing other transactions from also failing.

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		
t ₄		read(A3)	
t ₅		A3 = A3 - 10	
t ₆		write(A3)	
t ₇		read(A1)	
t ₈		A1 = A1 + 10	
t ₉		write(A1)	
t ₁₀		commit;	
t ₁₁	read(A3)		
t ₁₂	sum = sum + A3		
t ₁₃	read(A2)		
t ₁₄	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
t ₄		read(A3)	40	50	30	40
t ₅		A3 = A3 - 10	40	50	20	40

t6		write(A3)	40	50	20	40
t7		read(A1)	40	50	20	40
t8		A1 = A1 + 10	50	50	20	40
t9		write(A1)	50	50	20	40
t10		commit;	50	50	20	40
t11	read(A3)		50	50	20	40
t12	sum = sum + A3		50	50	20	60
t13	read(A2)		50	50	20	60
t14	sum = sum + A2		50	50	20	110

The summary value should be 100. The reason why it has become 110 is due to an unrepeatable read where A3 is read in t4 and then in t11 but what's returned is two different values. Because of this, the summary value is added up incorrectly as A3 has an additional 10 on its value and this is added on to the sum. The correct sum would be 10 less, 100.

Description of any problem encountered (if any).

For example, date format from a .csv is not working correctly in PostgreSQL (*US date/time display vs UK date/time) hence some adjustments must be made. Here you can describe how you fixed this issue - very brief.

Conclusion/Reflection

Once all questions are answered a short conclusion or reflection of what you have learned from this lab should be included.

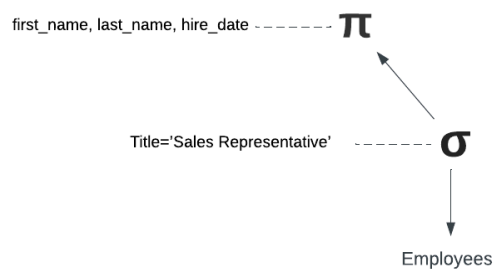
LAB 4 (25/10/2023)

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

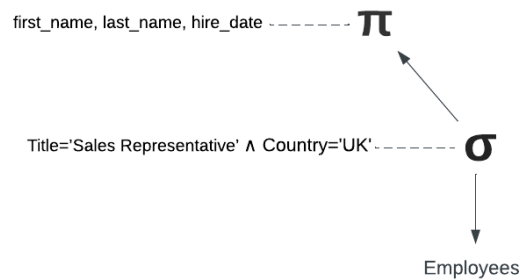
A: $\pi_{\text{first_name, last_name, hire_date}}(\sigma_{\text{Title='Sales Representative'}}(\text{Employees}))$



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: $\pi_{\text{first_name, last_name, hire_date}}(\sigma_{\text{Title='Sales Representative'} \wedge \text{Country = 'UK'}}(\text{Employees}))$



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: $\pi_{\text{order_id, order_date, shipper}}(\sigma_{\text{order_id} < 10300}(\text{orders} \bowtie_{\text{orders.ship_via} = \text{shippers.shipper_id}} \text{shippers}))$

Q4. Consider a database with the following schema:

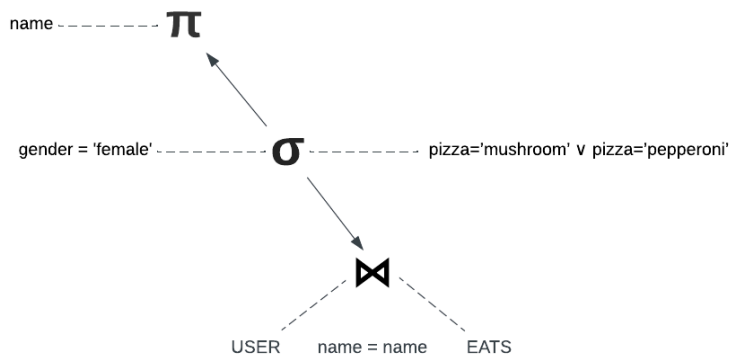
USER	(<u>name</u> , age, gender) PK - name
FREQUENTS	(<u>name</u> , <u>pizzeria</u>) PK name, pizzeria
EATS	(<u>name</u> , <u>pizza</u>) PK - name, pizza
SERVES	(<u>pizzeria</u> , <u>pizza</u> , price) PK - pizzeria, pizza

Write the 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: $\pi_{\text{name}}(\sigma_{\text{gender}='female'}(\text{USER}) \bowtie_{\text{USER.name}=\text{EATS.name}} \sigma_{\text{pizza}='mushroom' \vee \text{pizza}='pepperoni'}(\text{EATS}))$



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:

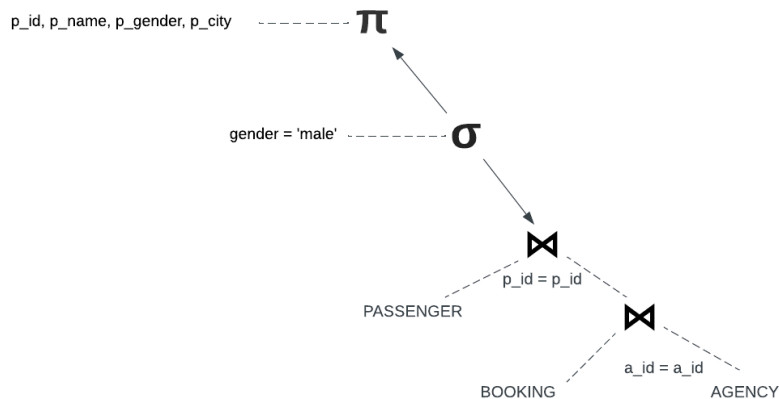
“List details of all male passengers who are booked through this agency”

A:

$\pi_{p_id, p_name, p_gender, p_city}$

$(\sigma_{p_gender = \text{'male'}}(\text{PASSENGER}) \bowtie_{\text{PASSENGER.p_id} = \text{BOOKING.p_id}} (\text{BOOKING} \bowtie_{\text{BOOKING.a_id} = \text{AGENCY.a_id}} \text{AGENCY}))$

Split it into separate lines for readability.



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

- a. Declarative
- b. Object-oriented
- c. Procedural
- d. Static
- e. None of the above

A: c

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 be there in the result of the operation $R_1 \times R_2$ (cartesian product)?

- a. 10 Records
- b. 5 Records
- c. 15 Records
- d. 50 Records
- e. None

A: d

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

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

A: Size estimate is 0, as 'physics' is not defined as a school.

LAB 5

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.

ANSWER:

1. Line 3, Movie_Title should be in quotation marks
2. Line 3 after "Movie_Title", there should be a comma
3. Line 4, LANGUAGE should be in double quotation marks = "LANGUAGE"
4. In the select part of the query, what table the attributes come from should be referenced for clarity, e.g m.movie_id instead of movie_id
5. Line 10, on the last line, at the end of the line, there should be ";

```
SELECT
  m.movie_id AS "Movie ID",
  m.movie_title AS "Movie Title",
  m.movie_lang AS "LANGUAGE",
  c.cat_name AS "Category"
FROM
  movie m
  INNER JOIN category c ON movie.movie_id = category.movie_id
WHERE
  movie.movie_title = 'ALADDIN CALENDAR';
```

- Q2.** You are given the following two queries. The table employee 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?

A	B
<pre>SELECT * FROM employee WHERE employee_id = 1;</pre>	<pre>SELECT emp_name, emp_last_name, emp_nin, emp_hire_date from employee WHERE employee_id = 1;</pre>

ANSWER = First query has better performance since it can actually execute.
Second query would be better if “employee_id = 1” was “employee_id = 1” since second query extracts the specific attributes needed

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

ANSWER:

```
\dv
```

List of relations			
Schema	Name	Type	Owner
public	actor_info	view	up2109066
public	customer_list	view	up2109066
public	movie_list	view	up2109066
public	nicer_but_slower_movie_list	view	up2109066
public	sales_by_movie_category	view	up2109066
public	sales_by_store	view	up2109066
public	staff_list	view	up2109066
(7 rows)			

7 VIEWS

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](#).

ANSWER:

```
SELECT category_id FROM category WHERE name = 'Documentary';
```

```
movie_rental=# SELECT category_id FROM category WHERE name = 'Documentary';
```

```
category_id
```

```
-----
```

```
6
```

```
(1 row)
```

```
EXPLAIN ANALYZE
```

```
SELECT
```

```
m.title AS "Movie Title"
```

```
FROM
```

```
movie m
```

```
JOIN movie_category mc ON m.movie_id = mc.movie_id
```

```
JOIN category c ON mc.category_id = c.category_id
```

```
WHERE c.name = 'Documentary';
```

```
QUERY PLAN
```

```
-----
Nested Loop (cost=16.69..38.26 rows=6 width=15) (actual time=0.051..0.425 rows=68 loops=1)
-> Hash Join (cost=16.41..35.06 rows=6 width=4) (actual time=0.045..0.242 rows=68 loops=1)
    Hash Cond: (mc.category_id = c.category_id)
    -> Seq Scan on movie_category mc (cost=0.00..16.00 rows=1000 width=8) (actual time=0.014..0.111 rows=1000 loops=1)
    -> Hash (cost=16.38..16.38 rows=3 width=4) (actual time=0.020..0.020 rows=1 loops=1)
        Buckets: 1024 Batches: 1 Memory Usage: 9kB
        -> Seq Scan on category c (cost=0.00..16.38 rows=3 width=4) (actual time=0.013..0.014 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: 0.387 ms
Execution Time: 0.478 ms
(13 rows)
```



```
EXPLAIN ANALYSE
```

```
SELECT m.title
```

```
FROM movie m
```

```
JOIN movie_category mc ON m.movie_id = mc.movie_id
```

```
WHERE mc.category_id = 6;
```

QUERY PLAN

```
-----
Hash Join (cost=19.35..88.78 rows=68 width=15) (actual time=0.132..0.445 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=0.008..0.218 rows=1000 loops=1)
    -> Hash (cost=18.50..18.50 rows=68 width=4) (actual time=0.118..0.118 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=4) (actual time=0.011..0.097 rows=68 loops=1)
                Filter: (category_id = 6)
                Rows Removed by Filter: 932
Planning Time: 0.250 ms
Execution Time: 0.501 ms
(10 rows)
```

	QUERY 1	QUERY 2	FASTER
PLANNING TIME	0.387	0.250	QUERY 2
EXECUTION TIME	0.478	0.501	QUERY 1

Planning Time:

Query 2 is faster because it uses `category_id` (integer), which PostgreSQL can process more efficiently than the string filtering in Query 1.

Execution Time:

Query 1 is slightly faster than Query 2 because it avoids the hash join for filtering. It filters the `movie_category` table using the string-based condition on the category table and then performs an indexed scan on the movie table, improving performance.

Query 2 uses a hash join to directly join the `movie_category` and `movie` tables, then filters by `category_id`, which adds extra overhead (additional resources such as time, memory, or processing) and slows down its execution time.

OUTPUT IS SAME FOR BOTH QUERIES:

```

      Movie Title
-----
ACADEMY DINOSAUR
ADAPTATION HOLES
ARMY FLINTSTONES
BEACH HEARTBREAKERS
BED HIGHBALL
BILL OTHERS
BONNIE HOLOCAUST
BROTHERHOOD BLANKET
CAUSE DATE
CHICKEN HELLFIGHTERS
CIDER DESIRE
CLERKS ANGELS
COAST RAINBOW
CUPBOARD SINNERS
DANCING FEVER
DEEP CRUSADE
DELIVERANCE MULHOLLAND
DOZEN LION
DUFFEL APOCALYPSE
EGG IGBY
EXPENDABLE STALLION
FRENCH HOLIDAY
HALLOWEEN NUTS
HARDLY ROBBERS
HAWK CHILL
HEAVYWEIGHTS BEAST
HOMeward CIDER
HUNTER ALTER
INDEPENDENCE HOTEL
INTOLERABLE INTENTIONS
KILL BROTHERHOOD
MADISON TRAP
MAJESTIC FLOATS
METAL ARMAGEDDON
MIDSUMMER GROUNDHOG
MIGHTY LUCK
MOD SECRETARY
MODERN DORADO
NATIONAL STORY
NEWSIES STORY
NORTH TEQUILA
NOTORIOUS REUNION
PACIFIC AMISTAD
PELICAN COMFORTS
POCUS PULP
PRINCESS GIANT
QUILLS BULL
RAIDERS ANTITRUST
RAINBOW SHOCK
ROAD ROXANNE
SAGEBRUSH CLUELESS
SECRET GROUNDHOG
SHIP WONDERLAND
SHOW LORD
SMOKING BARBARELLA
SPOILERS HELLFIGHTERS
STREAK RIDGEMONT
THIN SAGEBRUSH
UNITED PILOT
UNITED PILOT
UNTOUCHABLES SUNRISE
VILLAIN DESPERATE
VIRGINIAN PLUTO
WAGON JAWS
WARS PLUTO
WEDDING APOLLO
WIFE TURN
:[]

```

- 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.

ANSWER=

```
SELECT m.title, m.release_year
FROM movie m
JOIN language l ON m.original_language_id = l.language_id
WHERE l.name = 'Italian'
ORDER BY m.release_year ASC;
```

ALTER VICTORY	2018
WILD APOLLO	2019
GRIT CLOCKWORK	2020
DOCTOR GRAIL	2021
HIGHBALL POTTER	2021
HARPER DYING	2021
KENTUCKIAN GIANT	2022

(153 rows)

(FIRST 5 WITH COLUMN)

- 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

ANSWER:

```
SELECT c.name AS category, l.name AS language, COUNT(m.movie_id) AS movie_count
FROM movie m
JOIN language l ON m.original_language_id = l.language_id
JOIN movie_category mc ON m.movie_id = mc.movie_id
JOIN category c ON mc.category_id = c.category_id
GROUP BY c.name, l.name
ORDER BY c.name, l.name;
```

OUTPUT:

category	language	movie_count
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

:[] COLUMN NAME START WITH CAPITAL LETTER

LAB 6

- 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.

ANSWER:

BEFORE:

```

      phone
-----
 6171054059
(1 row)

```

```

UPDATE address
SET phone = '02392844444'
WHERE address_id = 100;

```

AFTER:

```

      phone
-----
 02392844444
(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.

ANSWER:

```
SELECT * FROM address
```

```
WHERE phone = '02392844444';
```

address_id	address	address2	district	city_id	postal_code	phone	last_update
100	1308 Arecibo Way		Georgia	41	30695	02392844444	2024-11-14 13:26:59.467389+00

```
EXPLAIN SELECT * FROM address
```

```
WHERE phone = '02392844444';
```

QUERY PLAN

```

-----
Seq Scan on address (cost=0.00..15.54 rows=1 width=63)
  Filter: ((phone)::text = '02392844444'::text)

```

Q3. Apply an INDEX on the **phone** attribute.

ANSWER:

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

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

ANSWER:

```
EXPLAIN SELECT * FROM address
WHERE phone = '0239284444';
```

QUERY PLAN

```
-----
Index Scan using phone_index on address (cost=0.28..8.29 rows=1 width=63)
  Index Cond: ((phone)::text = '0239284444'::text)
```

Where before indexing the planning was 0, now the planning is 0.28.

Where before indexing the execution time was 15.54, it is now almost halved at 8.29.

After indexing the performance is much better because the amount of data that needs to be filtered is reduced considerably. Rather than sequentially scanning, the phone number can be retrieved directly from the index, saving the cost of reading the entire table.

Q5. Delete the INDEX you have created in Q3.

ANSWER:

```
DROP INDEX phone_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 Naberezhnye 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
MICHAEL FORMAN	MICHAEL.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	Crdoaba	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 Tucum	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

ANSWER:

CREATE VIEW customer_details AS

SELECT

```

CONCAT(c.first_name, ' ', c.last_name) AS "Customer",
CONCAT(c.email, ' | ', a.phone) AS "Contact Details",
a.address AS "Customer Address",
ci.city AS "Customer City",
co.country AS "Customer Country"

```

FROM

```
customer c
```

JOIN

```
address a ON c.address_id = a.address_id
```

JOIN

```
city ci ON a.city_id = ci.city_id
```

JOIN

```
country co ON ci.country_id = co.country_id;
```

SELECT * FROM customer_details

ORDER BY "Customer Country" ASC;

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 Naberezhnye 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
MICHAEL FORMAN	MICHAEL.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

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

ANSWER:

```
SELECT ci.city_id, ci.city
FROM city ci
WHERE ci.city_id IN (
    SELECT a.city_id
    FROM address a
    JOIN city ci ON a.city_id = ci.city_id
    JOIN country co ON ci.country_id = co.country_id
    WHERE co.country = 'United Kingdom'
);
```

city_id	city
494	Southampton
589	York
495	Southend-on-Sea
88	Bradford
312	London
500	Stockport
496	Southport
149	Dundee

ALIASES USE

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.

ANSWER:

```
SELECT ci.city_id, ci.city, co.country
FROM city ci
JOIN country co ON ci.country_id = co.country_id
WHERE ci.city_id IN (
    SELECT a.city_id
    FROM address a
    JOIN city ci ON a.city_id = ci.city_id
    JOIN country co ON ci.country_id = co.country_id
    WHERE co.country IN ('United Kingdom', 'France')
)
ORDER BY co.country, ci.city;
```


city_id	city	country
92	Brest	France
297	Le Mans	France
543	Toulon	France
544	Toulouse	France
88	Bradford	United Kingdom
149	Dundee	United Kingdom
312	London	United Kingdom
494	Southampton	United Kingdom
495	Southend-on-Sea	United Kingdom
496	Southport	United Kingdom
500	Stockport	United Kingdom
589	York	United Kingdom

ALIAS

C1 - Using subqueries, list the concatenated first name and last name along with the email address - as a separate column - of all customers that have rented an action movie. List them in alphabetical order first name then last name.

Note: This query is a difficult one and can be solved in multiple ways and multiple subqueries. As long as you are using at least one subquery for the correct output, it will be a valid query. Look for multiple usage of **IN** and **USING** operators.

ANSWER:

```

SELECT
    CONCAT(c.first_name, ' ', c.last_name) AS "Full Name",
    c.email AS "Email Address"
FROM
    customer c
WHERE
    c.customer_id IN (
        SELECT r.customer_id
        FROM rental r
        JOIN inventory i ON r.inventory_id = i.inventory_id
        JOIN movie m ON i.movie_id = m.movie_id
        WHERE m.movie_id IN (
            SELECT mc.movie_id
            FROM movie_category mc
            JOIN category cat ON mc.category_id = cat.category_id
            WHERE cat.name = 'Action'
        )
    )
ORDER BY
    c.first_name, c.last_name;

```

Full Name	Email Address
AARON SELBY	AARON.SELBY@sakilacustomer.org
ADAM GOOCH	ADAM.GOOCH@sakilacustomer.org
AGNES BISHOP	AGNES.BISHOP@sakilacustomer.org
ALAN KAHN	ALAN.KAHN@sakilacustomer.org
ALBERT CROUSE	ALBERT.CROUSE@sakilacustomer.org
ALBERTO HENNING	ALBERTO.HENNING@sakilacustomer.org
ALEX GRESHAM	ALEX.GRESHAM@sakilacustomer.org
ALEXANDER FENNELL	ALEXANDER.FENNELL@sakilacustomer.org
ALFRED CASILLAS	ALFRED.CASILLAS@sakilacustomer.org
ALFREDO MCADAMS	ALFREDO.MCADAMS@sakilacustomer.org
ALICE STEWART	ALICE.STEWART@sakilacustomer.org
ALICIA MILLS	ALICIA.MILLS@sakilacustomer.org
ALLEN BUTTERFIELD	ALLEN.BUTTERFIELD@sakilacustomer.org
ALLISON STANLEY	ALLISON.STANLEY@sakilacustomer.org
ALMA AUSTIN	ALMA.AUSTIN@sakilacustomer.org
ALVIN DELOACH	ALVIN.DELOACH@sakilacustomer.org
AMANDA CARTER	AMANDA.CARTER@sakilacustomer.org
AMBER DIXON	AMBER.DIXON@sakilacustomer.org
ANA BRADLEY	ANA.BRADLEY@sakilacustomer.org
ANDRE RAPP	ANDRE.RAPP@sakilacustomer.org
ANDY VANHORN	ANDY.VANHORN@sakilacustomer.org
ANGEL BARCLAY	ANGEL.BARCLAY@sakilacustomer.org

LAB 7

Q1. Create read-only access to the "staff" table to a new role called "junior_analyst". In order to achieve this you will need to follow a couple of steps. The steps order is given but you will need to write the code (syntaxes). Use 2 terminals - one for super admin (your UP number, connected to the movie_rental database) and one for the created role.

- a. -- Create the Role *junior_analyst* with a Password and login rights

■ CODE:

```
CREATE ROLE junior_analyst WITH LOGIN PASSWORD 'password123';
```

- b. -- Login with junior_analyst role (using 2nd terminal)

■ `psql -h localhost -p 5432 -U junior_analyst -d movie_rental`

- c. -- Try to select the staff table

■ `SELECT * FROM staff;`

■ You should have the following error: "permission denied for table staff"

- d. -- Grant SELECT on the table STAFF to the role *junior_analyst*:

■ CODE:

```
GRANT SELECT ON staff to junior_analyst;
```

- e. -- Select staff table again. You should be able to see staff table data.

staff_id	first_name	last_name	address_id	email	store_id	active	username	password	last_update	picture
1	Mike	Hillyer	3	Mike.Hillyer@sakilastaff.com	1	t	Mike	8cb2237d0679ca88db6464eac60da96345513964	2022-05-16 15:13:11.79328+00	\x89504e470d0a5a0a
2	Jon	Stephens	4	Jon.Stephens@sakilastaff.com	2	t	Jon	8cb2237d0679ca88db6464eac60da96345513964	2022-05-16 15:13:11.79328+00	

Q2. Following a similar process as above, how would you grant **INSERT** access to the "payment_p2022_01" table to a new role called "*cashier*"?

Use 2 terminals again

- a. Create the role with correct logins and access level

■ CODE:

```
CREATE ROLE cashier WITH LOGIN PASSWORD 'password123';
```

- b. Grant the insert access for *cashier* role for the table *payment_p2022_01*.

■ CODE:

```
GRANT INSERT ON payment_p2022_01 to cashier;
```

- c. Login with junior_analyst role (using 2nd terminal)

■ `psql -h localhost -p 5432 -U cashier -d movie_rental`

- d. Try to make the following INSERT. Depending on how you granted the permissions, you might have an error. Fix the error on grant. If no error is generated and the INSERT was made, move to question 3.

```
INSERT INTO payment_p2022_01(customer_id, staff_id, rental_id, amount,
payment_date) VALUES (1, 1, 1, 5.99, CURRENT_TIMESTAMP);
```

Q3. Using your cashier role, try to `SELECT * FROM rental;`. You should not be able to do so.

- Switch to your superuser role and allow the role cashier to make SELECT and INSERT from the rental table.

■ CODE:

```
GRANT SELECT, INSERT ON rental TO cashier;
```

- Using the 2nd terminal (with *cashier* role) make the following INSERT:

```
INSERT INTO rental (rental_id, rental_date, inventory_id, customer_id,
return_date, staff_id, last_update)
VALUES (17000, CURRENT_DATE, 1, 1, NULL, 1, CURRENT_TIMESTAMP);
```

- see if the insert was made (you should have the today date)

```
SELECT * FROM rental WHERE rental_id = 17000;
```

rental_id	rental_date	inventory_id	customer_id	return_date	staff_id	last_update
17000	2024-11-23 00:00:00+00	1	1		1	2024-11-23 10:39:03.596165+00

- Using the role of cashier, DELETE the record with ID 17000

```
DELETE FROM rental WHERE rental_id = 17000;
```

- It should fail. How can you fix this and delete the record using the cashier role?

ANSWER = You have to grant delete privileges to the cashier role to enable deletion.

```
GRANT DELETE ON rental TO cashier;
```

```
(on second terminal) DELETE FROM rental WHERE rental_id = 17000;
```

```
SELECT * FROM rental WHERE rental_id = 17000;
```

```
rental_id | rental_date | inventory_id | customer_id | return_date | staff_id | last_update
-----+-----+-----+-----+-----+-----+-----
(0 rows)
```

Q4.How would you give a role named "*manager*" the ability to update the "*country*" table and also give them permission to grant this privilege to another role called "*sales*"?

- a. Create the role **manager** with all necessary permissions (login, update and to grant permissions to others)

■ CODE:

```
CREATE ROLE manager WITH LOGIN PASSWORD 'password123';
GRANT UPDATE ON country TO manager;
GRANT UPDATE ON country TO manager WITH GRANT OPTION;
```

- b. Create a role **sales** but only login permissions

■ CODE:

```
CREATE ROLE sales WITH LOGIN PASSWORD 'password123';
```

- c. Login as **manager** and grant the UPDATE on COUNTRY permission for **sales** role

```
psql -h localhost -p 5432 -U manager -d movie_rental
```

■ CODE:

```
GRANT UPDATE ON country TO sales;
```

- d. Login as **sales** make the following update

```
psql -h localhost -p 5432 -U sales -d movie_rental
UPDATE country SET last_update = NOW() WHERE country_id = 102;
```

Most likely you will have an error - Why? Think of the Transaction Analysis. Adjust the privileges for 'manager' and 'sales' and perform the UPDATE from the 'sales' account.

ANSWER= The reason is because although sales has update privileges, in order for the sql query to work it also needs to access the table through SELECT querying, which it doesn't have permissions to.

```
GRANT SELECT ON country TO manager;
GRANT SELECT ON country TO sales;
(FROM SUPERUSER)
```

- e. Check if the update was successful as **sales** - for last_update you should have the current data and time.

```
SELECT * FROM country WHERE country_id=102;
```

country_id	country	last_update
102	United Kingdom	2024-11-23 11:12:13.202956+00

Q5. Using the superuser role, create the following VIEW:

```
CREATE VIEW customer_view AS SELECT * FROM customer;
```

- a. How will you allow the "sales" role the ability to access the view without granting them direct access to the table?

■ CODE:

```
GRANT SELECT ON customer_view TO sales;
```

- b. `SELECT * FROM customer;` should fail with *"ERROR: permission denied for table customer"*

```
movie_rental=> SELECT * FROM customer;
ERROR: permission denied for table customer
```

`SELECT * FROM customer_view;` Should work without any issue

customer_id	store_id	first_name	last_name	email	address_id	activebool	create_date	last_update	active
1	1	MARY	SMITH	MARY.SMITH@sakilacustomer.org	5	t	2022-02-14	2022-02-15 09:57:20+00	1
2	1	PATRICIA	JOHNSON	PATRICIA.JOHNSON@sakilacustomer.org	6	t	2022-02-14	2022-02-15 09:57:20+00	1
3	1	LINDA	WILLIAMS	LINDA.WILLIAMS@sakilacustomer.org	7	t	2022-02-14	2022-02-15 09:57:20+00	1
4	2	BARBARA	JONES	BARBARA.JONES@sakilacustomer.org	8	t	2022-02-14	2022-02-15 09:57:20+00	1
5	1	ELIZABETH	BROWN	ELIZABETH.BROWN@sakilacustomer.org	9	t	2022-02-14	2022-02-15 09:57:20+00	1
6	2	JENNIFER	DAVIS	JENNIFER.DAVIS@sakilacustomer.org	10	t	2022-02-14	2022-02-15 09:57:20+00	1

Q6. As superuser add a new staff member

```
INSERT INTO staff (staff_id, first_name, last_name, address_id, email, store_id,
active, username, password, last_update, picture)
VALUES (3, 'Val', 'Adam', 5, 'val.adam@sakilastaff.com', 1, 't', 'valadam',
'8cb2237d0679ca88db6464eac60da96345513964', CURRENT_TIMESTAMP, NULL);
```

- a. Create a new role as "admin" that will have the capability to create new roles.

```
CREATE ROLE admin WITH LOGIN PASSWORD 'password123' CREATEROLE;
```

- b. Assign the staff named "Val" to admin group.

```
CREATE ROLE val WITH LOGIN PASSWORD 'ValAdam';
GRANT admin to val;
```

- c. If everything was executed correctly, running the `\du` command you should have the following entry

```
movie_rental=> \du
```

Role name	List of roles Attributes	Member of
admin	Create role	{}
cashier		{}
junior_analyst		{}
manager		{}
postgres	Superuser, Create role, Create DB, Replication, Bypass RLS	{}
sales		{}
uon-1sps-2021	Superuser	{}
val		{admin}
valentinadamescu	Superuser	{}

Role name	Attributes	Member of
admin	Create role	{}
cashier		{}
junior_analyst		{}
manager		{}
postgres	Superuser, Create role, Create DB, Replication, Bypass RLS	{}
sales		{}
test-image-2	Superuser	{}
uop	Superuser	{}
up2109066	Superuser	{}
val		{admin}

Q7. How would you create a role "senior_analyst" that is able to create new tables, drop the created tables, view data in those tables but without the ability to read or modify the data from existing tables?

```
CREATE ROLE senior_analyst WITH LOGIN PASSWORD 'password123';

GRANT CREATE ON DATABASE movie_rental TO senior_analyst;
GRANT USAGE ON SCHEMA public TO senior_analyst;
GRANT CREATE ON SCHEMA public TO senior_analyst;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO senior_analyst;

REVOKE SELECT ON ALL TABLES IN SCHEMA public FROM senior_analyst;
REVOKE INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public FROM senior_analyst;
```

- a. After role creation try to `CREATE TABLE new_table ();` It should work
- b. `SELECT * FROM customer;` It should fail

```
movie_rental=> CREATE TABLE new_table ();
CREATE TABLE
movie_rental=> SELECT * FROM customer;
ERROR: permission denied for table customer
```

LAB 8

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.*

ANSWER=

```
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 names** are in actors' names? (e.g. If they are 3 Smith's will be counted only once).

ANSWER=

```
SELECT COUNT(DISTINCT last_name) AS "Unique Last Name"
FROM actor;
```

Unique Last Name
121

(1 row)

Q3. Following from the previous query, we know how many unique last names are in the database. However, how many of them are appearing only once? E.g. OLIVIER will be twice, so is not that unique, but PESCI will be only once making truly unique. List all the true unique names)

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).

ANSWER=

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


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	
LOLLOBRIGI	
MALDEN	
MANSFIELD	
MARX	
MCDORMAND	
MIRANDA	SOBIESKI
NICHOLSON	STALLONE
PESCI	SUVARI
PFEIFFER	SWANK
PHOENIX	TAUTOU
PINKETT	TOMEI
PITT	VOIGHT
POSEY	WALKEN
PRESLEY	WAYNE
REYNOLDS	WILSON
RYDER	WITHERSPOON
SINATRA	WRAY

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**.

ANSWER=

```
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?

ANSWER=

```
SELECT
    i.inventory_id,
    COUNT(i.inventory_id) AS available_copies
FROM inventory AS i
JOIN movie AS m ON i.movie_id = m.movie_id
LEFT JOIN rental r ON i.inventory_id = r.inventory_id AND r.return_date IS NULL
WHERE m.title = 'PURPLE MOVIE'
    AND i.store_id = 1
    AND r.rental_id IS NULL
GROUP BY i.inventory_id;
```

inventory_id	available_copies
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 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

ANSWER=

```
SELECT
    CONCAT(s.first_name, ' ', s.last_name) AS "Staff Name",
    CONCAT(COALESCE(a.address, ''), ' ', COALESCE(a.address2, '')) AS "Staff Address",
    c.city AS "Staff City",
    s.email AS "Staff Email"
FROM staff s
JOIN address a ON s.address_id = a.address_id
JOIN city c ON a.city_id = c.city_id;
```

Staff Name	Staff Address	Staff City	Staff Email
Mike Hillyer	23 Workhaven Lane	Lethbridge	Mike.Hillyer@sakilastaff.com
Val Adam	1913 Hanoi Way	Sasebo	val.adam@sakilastaff.com
Jon Stephens	1411 Lillydale Drive	Woodridge	Jon.Stephens@sakilastaff.com

(3 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.

ANSWER=

```
SELECT
    CONCAT(a.first_name, ' ', a.last_name) AS "ACTOR"
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 a.last_name ASC;
```

ACTOR
KENNETH HOFFMAN
SANDRA KILMER
JAYNE NEESON
WARREN NOLTE
KIRSTEN PALTROW
REESE WEST
MORGAN WILLIAMS

(7 rows)

C1. - Question 7 displays a column of all actors in a movie, which is not well formatted.

List the movie "Agent Truman" and all the actors as one row. The output should be exactly as below.

Hint: look for arrays and strings display.

Movie Title	Actors in the movie
Agent Truman	Kenneth Hoffman, Sandra Kilmer, Jayne Neeson, Warren Nolte, Kirsten Paltrow, Reese West, Morgan Williams

SELECT 1

ANSWER=

```
SELECT
    m.title AS "Movie Title",
    STRING_AGG(CONCAT(a.first_name, ' ', a.last_name), ', ' ORDER BY a.last_name ASC,
a.first_name ASC) AS "Actors in the movie"
FROM movie m
JOIN movie_actor ma ON m.movie_id = ma.movie_id
JOIN actor a ON ma.actor_id = a.actor_id
WHERE m.title = 'AGENT TRUMAN'
GROUP BY m.title;
```

Movie Title	Actors in the movie
AGENT TRUMAN	KENNETH HOFFMAN, SANDRA KILMER, JAYNE NEESON, WARREN NOLTE, KIRSTEN PALTROW, REESE WEST, MORGAN WILLIAMS

(1 row)

LAB 9 - Advanced SQL

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.

ANSWER:

```
\df
```

Schema	Name	Result data type	Argument data types	Type
public	get_customer_balance	numeric	p_customer_id integer, p_effective_date timestamp with time zone	func
public	group_concat	text	text	agg
public	group_concat	text	text, text	func
public	inventory_held_by_customer	integer	p_inventory_id integer	func
public	inventory_in_stock	boolean	p_inventory_id integer	func
public	last_day	date	timestamp with time zone	func
public	last_updated	trigger		func
public	movie_in_stock	SETOF integer	p_movie_id integer, p_store_id integer, OUT p_movie_count integer	func
public	movie_not_in_stock	SETOF integer	p_movie_id integer, p_store_id integer, OUT p_movie_count integer	func
public	rewards_report	SETOF customer	min_monthly_purchases integer, min_dollar_amount_purchased numeric	func

(10 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.

```
SELECT 4
```

Movies in Stock
124
125
126
127

ANSWER=

```
SELECT movie_in_stock(
  p_movie_id => (
    SELECT movie_id
    FROM movie
    WHERE title = 'ANGELS LIFE'
  ),
  p_store_id => 1
) AS "Movies in Stock";
```

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).

ANSWER=

```
CREATE PROCEDURE sp_add_new_actor(p_first_name VARCHAR, p_last_name VARCHAR)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO actor (first_name, last_name)
    VALUES (p_first_name, p_last_name);
END;
$$;

CALL sp_add_new_actor('Basit', 'Sediqi');

SELECT * FROM actor
WHERE first_name = 'Basit' AND last_name = 'Sediqi';
```

actor_id	first_name	last_name	last_update
201	Basit	Sediqi	2024-12-07 11:31:05.184816+00

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

ANSWER=

```
ALTER TABLE country
ADD COLUMN country_code VARCHAR;

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.

ANSWER=

```
ALTER TABLE country
ALTER COLUMN country_code TYPE CHAR(2),
ADD CONSTRAINT country_code_unique UNIQUE (country_code); \d country
```

Column	Type	Table "public.country"	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)				

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.

ANSWER=

```
UPDATE country
SET country_code = 'UK'
WHERE country = 'United Kingdom';

SELECT country_id AS "country id",
       country AS "country name",
       country_code AS "country code"
FROM country
WHERE country = 'United Kingdom';
```

country id	country name	country code
102	United Kingdom	UK

(1 row)

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.

ANSWER=

```
CREATE TABLE new_staff (
    staff_id INT,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    email VARCHAR(255)
);

INSERT INTO new_staff (staff_id, first_name, last_name, email)
SELECT staff_id, first_name, last_name, email
FROM staff;

SELECT * FROM new_staff;
```

staff_id	first_name	last_name	email
1	Mike	Hillyer	Mike.Hillyer@sakilastaff.com
2	Jon	Stephens	Jon.Stephens@sakilastaff.com
3	Val	Adam	val.adam@sakilastaff.com

(3 rows)

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
ROUND 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 MESON	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 JERYLL	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	
(10)	

ANSWER=

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

Movie Title	Number of Copies
ACADEMY DINOSAUR	8
APACHE DIVINE	8
BEVERLY OUTLAW	8
BINGO TALENTED	8
BOOGIE AMELIE	8
ROUND 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
GRITTY CLOWWORK	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

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

ANSWER=

```
SELECT c.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 c ON mc.category_id = c.category_id
GROUP BY c.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.

ANSWER=

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

Category	Average Movie Length
Sports	128.28
Games	127.84
Foreign	121.78
Drama	120.84
Comedy	115.83

(5 rows)