1A) Imagine a College Management System which manages student information, classes, subjects, grades, attendance, and teacher assignments.

Students: Stores personal details of each student (student_ID, name, date_of_birth, gender, class_ID).

Classes: Lists the different classes in the school (class_ID, class_name, teacher_ID - optional, could be in a separate assignment table).

Subjects: Lists the subjects taught in the school (subject_ID, subject_name).(No. Of. Subjects= 4, Each carries 25 Marks).

Teachers: Stores personal details of teachers (teacher_ID, name, subject_ID - can teach multiple, so might need a linking table).

Marks: Records the marks obtained by students in different subjects for various exams (student_ID, class_ID,subject_ID, exam_name, marks).

Solve the following statements with the help of SQL Queries.

1) Create a stored procedure to get the average marks of a specific class.
2) Create a View to Get Student marks for All Subjects.
3)Display the student ID, Name, Class Name and Marks of a student having highest Marks in given subject.
4) List the names and dates of birth of all students born in the month of March.
5) Find the average marks for each subject across all students for a specific exam.
6)Write an SQL query to find the total number of students in each class.

1B) Imagine a College Management System which manages student information, classes, subjects, grades, attendance, and teacher assignments.

Students: Stores personal details of each student (student_ID, name, date_of_birth, gender, class_ID).

Classes: Lists the different classes in the school (class_ID, class_name, teacher_ID - optional, could be in a separate assignment table).

Subjects: Lists the subjects taught in the school (subject_ID, subject_name). (No. Of. Subjects= 4, Each carries 25 Marks).

Teachers: Stores personal details of teachers (teacher_ID, name, subject_ID - can teach multiple, so might need a linking table).

Marks: Records the marks obtained by students in different subjects for various exams (student_ID, class_ID, subject_ID, exam_name, marks).

Attendance: Tracks the daily attendance of students (student ID, Month, No_Of_Days_Present).

Solve the following statements with the help of SQL Queries.

1) Define a function to calculate the number of absent days for a given student in a specific month and year.
2) Combine a list of all students in class 'SE' who have scored more than 20 marks in any subject.
3) Identify students whose names have the second letter as 'a'.
4) Calculate the average attendance percentage for all students in class 'FE' for the month of March.
5) List the names of students who have scored below 10 marks in any subject.
6)List the names of all students in the 'SE' class and their corresponding total marks in the "Midterm" exam.

1C) Imagine a School Management System which manages student information, classes, subjects, grades, attendance, and teacher assignments.
Students: Stores personal details of each student (student ID, name, date of birth, gender, class ID).
Classes: Lists the different classes in the school (class ID, class name, teacher ID - optional, could be in a separate assignment table).
Subjects: Lists the subjects taught in the school (subject ID, subject name).
Teachers: Stores personal details of teachers (teacher ID, name, subject ID - can teach multiple, so might need a linking table).
Grades: Records the marks obtained by students in different subjects for various exams (grade ID, student ID, subject ID, exam name, marks).
Attendance: Tracks the daily attendance of students (student ID, attendance date, status - Present, Absent, Late).
Solve the following statements with the help of SQL Queries.

1) Create a View to Get Students' Attendance Summary
2) Find the number of students in each class.
3) Retrieve the names and student IDs of all students whose names begin with the letter 'D'.
4) Write a query to find the total number of subjects offered in the school.
5) Create a trigger to insert the student's name, old class id, new class id and timestamp automatically in StudentClassHistorytablewhen a student is moved to a different class.
6) Generate a combined list of the top 3 highest marks in the "Midterm" exam across all subjects

2A) Imagine a popular mobile application called "Fitness Tracking" used by fitness enthusiasts in the city. This application allows users to log their workouts, track their steps, monitor their heart rate, and set fitness goals. The application uses a MySQL database to store user data. Entities are given below:

- **Users:** Stores user profiles (user_ID, name, email, registration_date, goal_ID).
- **Goals:** Stores info of fitness goals (goal_ID,goal_type)
- **Users_Goal**: (user_ID, goal_ID, goal_type)
- **Workouts:** Records details of each workout session (workout_ID, user_ID, workout_type, start_time, end_time, duration_in_minutes).
- **WorkoutDetails:** Stores specific details for each workout, such as exercises performed, sets, reps, and weight used (workout_ID, exercise_name, sets, repetitions, weight).
- **Steps:** Tracks daily step counts for each user (step_ID, user_ID, tracking_date, step_count).

Write and Execute MySQL Queries for the following.

1. Find the total duration of workouts for each user.
2. Create a stored procedure to retrieve all workouts for a specific user within a given date range.
3. Retrieve a list of users along with the total number of workouts they have logged.
4. Create a view that shows a summary of each user's workouts. This should include the user's name, workout type and total duration of all workouts.
5. Retrieve all users who have set a goal to lose weight (use the goal_type field to identify weight loss goals).
6. Retrieve the total number of workouts a user performed.

2B) Imagine a popular mobile application called "Fitness Tracking" used by fitness enthusiasts in the city. This application allows users to log their workouts, track their steps, monitor their heart rate, and set fitness goals. The application uses a MySQL database to store user data. Entities are given below:

- **Users:** Stores user profiles (user_ID, name, email, registration_date, goal_ID).
- **Goals:** Stores info of fitness goals (goal_ID,goal_type)
- **Users_Goal**: (user_ID, goal_ID, goal_type)
- **Workouts:** Records details of each workout session (workout_ID, user_ID, workout_type, start_time, end_time, duration_in_minutes).
- **WorkoutDetails:** Stores specific details for each workout, such as exercises performed, sets, reps, and weight used (workout_ID, exercise_name, sets, repetitions, weight).
- **Steps:** Tracks daily step counts for each user (step_ID, user_ID, tracking_date, step_count).

Write and Execute MySQL Queries for the following.

1) Retrieve users who have a 'Improve Endurance' goal and have logged 'Running' or 'Cycling' workouts.
2) Calculate the average daily step count for each user.
3) List users who logged a workout on a day they also tracked more than 8,000 steps.
4) Create a view showing each user's name and the date of their most recent workout.
5) Create a Trigger to ensure that the end_time of a workout in the Workouts table is always after the start_time.
6) Identify users who have set a 'Lose Weight' goal and have logged more than 1 workout.

2C) Imagine a popular mobile application called "Fitness Tracking" used by fitness enthusiasts in the city. This application allows users to log their workouts, track their steps, monitor their heart rate, and set fitness goals. The application uses a MySQL database to store user data. Entities are given below:

- **Users:** Stores user profiles (user_ID, name, email, registration_date, goal_ID).
- **Goals:** Stores info of fitness goals (goal_ID,goal_type)
- **Users_Goal**: (user_ID, goal_ID, goal_type)
- **Workouts:** Records details of each workout session (workout_ID, user_ID, workout_type, start_time, end_time, duration_in_minutes).
- **WorkoutDetails:** Stores specific details for each workout, such as exercises performed, sets, reps, and weight used (workout_ID, exercise_name, sets, repetitions, weight).
- **Steps:** Tracks daily step counts for each user (step_ID, user_ID, tracking_date, step_count).

Write and Execute MySQL Queries for the following.

1. Find the user who has taken the most total steps.
2. Find all users who have set a specific fitness goal type (e.g., "weight loss").
3. Find the average duration of each workout type
4. Create a Function to calculate the total steps taken by a user within a specific date range.
5. Retrieve all workouts where the duration was greater than 60 minutes.
6. Retrieve all workout sessions performed by a specific user on a given date, including the workout type, start time, and end time.

2D) Imagine a popular mobile application called "Fitness Tracking" used by fitness enthusiasts in the city. This application allows users to log their workouts, track their steps, monitor their heart rate, and set fitness goals. The application uses a MySQL database to store user data. Entities are given below:

- **Users:** Stores user profiles (user_ID, name, email, registration_date, goal_ID).
- **Goals:** Stores info of fitness goals (goal_ID,goal_type)
- **Users_Goal**: (user_ID, goal_ID, goal_type)
- **Workouts:** Records details of each workout session (workout_ID, user_ID, workout_type, start_time, end_time, duration_in_minutes).
- **WorkoutDetails:** Stores specific details for each workout, such as exercises performed, sets, reps, and weight used (workout_ID, exercise_name, sets, repetitions, weight).
- **Steps:** Tracks daily step counts for each user (step_ID, user_ID, tracking_date, step_count).

Write and Execute MySQL Queries for the following.

1) Identify Users Active on a Specific Date.
2) Determine the most common goal_type
3) Find Users Who Registered in the Last Month.
4) Create a view to get a summary of each user's workouts, including their name, the workout type, and the duration.
5) Create a cursor for the first three users listed in the Users table (based on their user_ID), display their user_ID and name

3A) Imagine a online restaurant needs a system to manage the menu, track customer orders and generate sales reports. They use a MySQL database for this. Entities are as follows:

- **Menu:** Stores details of all the items offered by the restaurant (item_ID, item_name, category, price).
- **Customers:** Stores information about registered customers (customer_ID, name, contact_number).
- **Orders:** Records details of each order placed (order_ID, customer_ID, order_date_time).
- **OrderItems:** Lists the individual items included in each order (order_ID, item_ID, quantity, unit_price).

Write and execute the SQL Queries for the following

1. Find the total sales for each menu category.
2. Create a stored procedure to retrieve the details of a specific order given its order ID.
3. Retrieve a list of all orders along with the customer's name.
4. Identify menu items that have never been ordered.
5. Create a view to display top 3 selling items.
6. Retrieve all orders placed on the current day.

3B) Imagine a online restaurant needs a system to manage the menu, track customer orders and generate sales reports. They use a MySQL database for this. Entities are as follows:

- **Menu:** Stores details of all the items offered by the restaurant (item_ID, item_name, category, price).
- **Customers:** Stores information about registered customers (customer_ID, name, contact_number).
- **Orders:** Records details of each order placed (order_ID, customer_ID, order_date_time).
- **OrderItems:** Lists the individual items included in each order (order_ID, item_ID, quantity, unit_price).

Write and execute the SQL Queries for the following

1) Calculate the total price of each item in order_ID 1.
2) Find the names of all customers who have placed an order
3) Create a view that shows the order ID, item name, quantity, and the total price for each item in an order.
4) Find the names of customers who have placed orders containing items from both the 'Pizza' and 'Burgers' categories.
5) Find the menu items that have never been included in any order.
6) Create a function to check if an item is in a specific category.

3C) Imagine a online restaurant needs a system to manage the menu, track customer orders and generate sales reports. They use a MySQL database for this. Entities are as follows:

- **Menu:** Stores details of all the items offered by the restaurant (item_ID, item_name, category, price).
- **Customers:** Stores information about registered customers (customer_ID, name, contact_number).
- **Orders:** Records details of each order placed (order_ID, customer_ID, order_date_time).
- **OrderItems:** Lists the individual items included in each order (order_ID, item_ID, quantity, unit_price).

Write and execute the SQL Queries for the following

1) Create a stored procedure that takes an item name, category, and price as input and inserts a new record into the Menu table.
2) Find the highest and lowest price from the Menu table, along with the names of the items.
3) Count the number of items in each category in the Menu table.
4) Calculate the average number of items per order.
5) Create a view that summarizes the sales of each menu item, including the item name, the total quantity sold, and the total revenue generated.
6) Get the names of all items in a specific order, given the order ID.

4A) Imagine an online marketplace manages a vast inventory of Home appliances products, process customer orders, and track the fulfillment process. They use a MySQL database to manage their products, inventory, orders, and shipping information. Entities are listed below:

- **Products:** Stores detailed information about each product listed on the platform (product_ID, name, description, price, category_ID).
- **Categories:** Lists the different product categories (category_ID, category_name).
- **Inventory:** Tracks the stock levels for each product in their warehouse(s) (inventory_ID, product_ID, warehouse_ID, quantity_in_stock, last_stock_update).
- **Warehouses:** Lists the different warehouse locations (warehouse_ID, warehouse_name, address).
- **Orders:** Records customer orders (order_ID, customer_ID, order_date, shipping_address, order_status).
- **OrderItems:** Lists the individual products included in each order (order_item_ID, order_ID, product_ID, quantity_ordered, unit_price).
- **Shipments:** Tracks the shipment details for each order (shipment_ID, order_ID, shipping_carrier, tracking_number, shipment_date).

Write and execute the SQL Queries for the following

1. Find how many products are listed under each category.
2. Create a stored procedure to retrieve the current stock level of a specific product in a given warehouse.
3. Retrieve a list of all orders along with the names of the products ordered in each order.
4. Create a view to show products with low stock (less than 10 units total)
5. Count how many orders exist in each status (e.g., Pending, Shipped, Delivered).
6. Calculate the total number of products ordered by each customer.

4B) Imagine an online marketplace manages a vast inventory of Home appliances products, process customer orders, and track the fulfillment process. They use a MySQL database to manage their products, inventory, orders, and shipping information. Entities are listed below:

- **Products:** Stores detailed information about each product listed on the platform (product_ID, name, description, price, category_ID).
- **Categories:** Lists the different product categories (category_ID, category_name).
- **Inventory:** Tracks the stock levels for each product in their warehouse(s) (inventory_ID, product_ID, warehouse_ID, quantity_in_stock, last_stock_update).
- **Warehouses:** Lists the different warehouse locations (warehouse_ID, warehouse_name, address).
- **Orders:** Records customer orders (order_ID, customer_ID, order_date, shipping_address, order_status).
- **OrderItems:** Lists the individual products included in each order (order_item_ID, order_ID, product_ID, quantity_ordered, unit_price).
- **Shipments:** Tracks the shipment details for each order (shipment_ID, order_ID, shipping_carrier, tracking_number, shipment_date).

Write and execute the SQL Queries for the following

1. Find the total quantity in stock for each product across all warehouses.
2. Create a view to show the total price of each order.
3. Combine a list of products in the 'Refrigerators' category and a list of products that have a price greater than 1000.
4. Create a cursor to loop through products and display their total stock.
5. Show products with their stock, ordered by product name.
6. Calculate the total sales for each product based on the quantity ordered and unit price.

4C) Imagine an online marketplace manages a vast inventory of Home appliances products, process customer orders, and track the fulfillment process. They use a MySQL database to manage their products, inventory, orders, and shipping information. Entities are listed below:

- **Products:** Stores detailed information about each product listed on the platform (product_ID, name, description, price, category_ID).
- **Categories:** Lists the different product categories (category_ID, category_name).
- **Inventory:** Tracks the stock levels for each product in their warehouse(s) (inventory_ID, product_ID, warehouse_ID, quantity_in_stock, last_stock_update).
- **Warehouses:** Lists the different warehouse locations (warehouse_ID, warehouse_name, address).
- **Orders:** Records customer orders (order_ID, customer_ID, order_date, shipping_address, order_status).
- **OrderItems:** Lists the individual products included in each order (order_item_ID, order_ID, product_ID, quantity_ordered, unit_price).
- **Shipments:** Tracks the shipment details for each order (shipment_ID, order_ID, shipping_carrier, tracking_number, shipment_date).

Write and execute the SQL Queries for the following

1. Calculate how many shipments were handled by each carrier.
2. List products, ordered by price in ascending order.
3. Create a trigger to prevent orders from being created if the quantity of products ordered exceeds the quantity in stock.
4. Create a view to display the product name, current quantity in stock, and the name of the warehouse for all products.
5. List all orders placed on a specific date.
6. Calculate the average quantity ordered for each product.

4D) Imagine an online marketplace manages a vast inventory of Home appliances products, process customer orders, and track the fulfillment process. They use a MySQL database to manage their products, inventory, orders, and shipping information. Entities are listed below:

- **Products:** Stores detailed information about each product listed on the platform (product_ID, name, description, price, category_ID).
- **Categories:** Lists the different product categories (category_ID, category_name).
- **Inventory:** Tracks the stock levels for each product in their warehouse(s) (inventory_ID, product_ID, warehouse_ID, quantity_in_stock, last_stock_update).
- **Warehouses:** Lists the different warehouse locations (warehouse_ID, warehouse_name, address).
- **Orders:** Records customer orders (order_ID, customer_ID, order_date, shipping_address, order_status).
- **OrderItems:** Lists the individual products included in each order (order_item_ID, order_ID, product_ID, quantity_ordered, unit_price).
- **Shipments:** Tracks the shipment details for each order (shipment_ID, order_ID, shipping_carrier, tracking_number, shipment_date).

Write and execute the SQL Queries for the following

1) Define a function to check if a product is currently out of stock (total quantity across all warehouses is zero).
2) Find the total sales for each category
3) Implement a trigger to automatically decrease the quantity_in_stock in the Inventory table when a new order item is added.
4) Combine a list of products in the 'Washing Machines' category and a list of products that have a price greater than 900.
5) Get all products in a specific category.
6) Find the total revenue generated by each product.

5A) Imagine a **Hospital Management System** which stores Patients information, Doctors information, Appointments, Departments.

Patients: Stores personal details of each patient (PatientID, PatientName, DOB, Address, PhoneNumber)
Doctors: Stores details of doctors (DoctorID, DoctorName, Specialty, DepartmentID)
Appointments: Tracks patient appointments with doctors (AppointmentID, PatientID, DoctorID, AppointmentDate, Created_At)
Departments: Lists various departments in the hospital (DepartmentID, DepartmentName, Location)

Solve the following statements with the help of SQL Queries

1.Find doctors who specialize in 'Cardiology', 'Neurology', or 'Orthopedics'
2.List patients ordered by date of birth (youngest to oldest)
3.Count how many appointments were made today
4.Get doctor names with their respective department names.
5.Procedure to add a new patient:
6.Set default department for a new doctor (if none provided)

/***/

5B) Imagine a **Hospital Management System** which stores Patients information, Doctors information, Appointments, Departments.

Patients: Stores personal details of each patient (PatientID, PatientName, DOB, Address, PhoneNumber)
Doctors: Stores details of doctors (DoctorID, DoctorName, Specialty, DepartmentID)
Appointments: Tracks patient appointments with doctors (AppointmentID, PatientID, DoctorID, AppointmentDate, Created_At)
Departments: Lists various departments in the hospital (DepartmentID, DepartmentName, Location)

Solve the following statements with the help of SQL Queries

1.List all doctors ordered by their specialties alphabetically.
2.How many doctors work in each department?
3. Find the total number of appointments per doctor.
4.Show patient names with their appointment dates.
5. Create a procedure to get doctor information by ID.
6.Create a trigger to set appointment date when a new record is added

/***/

5C) Imagine a **Hospital Management System** which stores Patients information, Doctors information, Appointments, Departments.

Patients: Stores personal details of each patient (PatientID, PatientName, DOB, Address, PhoneNumber)
Doctors: Stores details of doctors (DoctorID, DoctorName, Specialty, DepartmentID)
Appointments: Tracks patient appointments with doctors (AppointmentID, PatientID, DoctorID, AppointmentDate, Created_At)
Departments: Lists various departments in the hospital (DepartmentID, DepartmentName, Location)

Solve the following statements with the help of SQL Queries

1.Which patients are considered senior citizens (over 65 years old) and might need special care?
2.What is the average age of all patients?
3. List all doctors with their department names.
4.Create a procedure to find appointments for a specific doctor.
5.Find the youngest patient in the hospital.
6.Write a stored procedure using a cursor that goes through all patients and displays the names and phone numbers of those who are above 60 years old (senior citizens).

/**/

5D) Imagine a **Hospital Management System** which stores Patients information, Doctors information, Appointments, Departments.

Patients: Stores personal details of each patient (PatientID, PatientName, DOB, Address, PhoneNumber)
Doctors: Stores details of doctors (DoctorID, DoctorName, Specialty, DepartmentID)
Appointments: Tracks patient appointments with doctors (AppointmentID, PatientID, DoctorID, AppointmentDate, Created_At)
Departments: Lists various departments in the hospital (DepartmentID, DepartmentName, Location)

Solve the following statements with the help of SQL Queries

1.List all doctors who work in the Cardiology department.
2.List patients between 30 and 50 years old.
3.Who are the 3 oldest patients in the hospital?
4. Create a procedure that returns patient counts by age groups
5.Create a view that shows patient ID, name, age, and phone number for all patients.
6.Create a function that counts how many appointments a patient has.

/**/

6A) **Imagine a Library Management System** which manages member information, library resources, physical loans, digital access, and membership tiers.

Members: Stores personal details of each member (MemberID, Name, Address, Phone, Email, TierID).
Resources: Lists the different resources available in the library, including books and digital content (ResourceID, Title, Author, Type, Format, AvailabilityStatus).
Loans: Records the physical loans of resources issued to members (LoanID, MemberID, ResourceID, LoanDate, ReturnDate).
DigitalAccess: Tracks the digital access of members to online resources (AccessID, MemberID, ResourceID, AccessDate, ExpiryDate).
MembershipTiers: Defines the privileges of different types of memberships (TierID, TierName, MaxLoans, MaxDigitalAccess).
DigitalAccess: Tracks digital access to online resources (AccessID, MemberID, ResourceID, AccessDate, ExpiryDate)
MembershipTiers: Defines borrowing and access privileges based on membership tier (TierID, TierName, MaxLoans, MaxDigitalAccess)

Solve the following statements with the help of SQL Queries
1.Retrieve all members who belong to TierID 2.
2.List members who have borrowed more than 5 resources.
3.What is the total of all MaxLoans defined in the Membership Tiers?
4. List member names along with the date they borrowed each resource.
5.Create a trigger to automatically mark a resource as 'Not Available' when it's loaned
6.Add a new column DateOfBirth to the Members table.

/***/

6 B) **Imagine a Library Management System** which manages member information, library resources, physical loans, digital access, and membership tiers.

Members: Stores personal details of each member (MemberID, Name, Address, Phone, Email, TierID).
Resources: Lists the different resources available in the library, including books and digital content (ResourceID, Title, Author, Type, Format, AvailabilityStatus).
Loans: Records the physical loans of resources issued to members (LoanID, MemberID, ResourceID, LoanDate, ReturnDate).
DigitalAccess: Tracks the digital access of members to online resources (AccessID, MemberID, ResourceID, AccessDate, ExpiryDate).
MembershipTiers: Defines the privileges of different types of memberships (TierID, TierName, MaxLoans, MaxDigitalAccess).
DigitalAccess: Tracks digital access to online resources (AccessID, MemberID, ResourceID, AccessDate, ExpiryDate)
MembershipTiers: Defines borrowing and access privileges based on membership tier (TierID, TierName, MaxLoans, MaxDigitalAccess)

Solve the following statements with the help of SQL Queries
1. Get all resources that are either Books or Magazines.
2.List all members sorted alphabetically by their names.
3.Count the number of members in each membership tier.
4.Create a procedure to retrieve all loans of a given member.
5.Update the phone number of the member with MemberID = 1.
6. Show a list of all loans along with member names and emails. Include loans even if the member details are missing.

/***/

6 C) **Imagine a Library Management System** which manages member information, library resources, physical loans, digital access, and membership tiers.

Members: Stores personal details of each member (MemberID, Name, Address, Phone, Email, TierID).
Resources: Lists the different resources available in the library, including books and digital content (ResourceID, Title, Author, Type, Format, AvailabilityStatus).
Loans: Records the physical loans of resources issued to members (LoanID, MemberID, ResourceID, LoanDate, ReturnDate).
DigitalAccess: Tracks the digital access of members to online resources (AccessID, MemberID, ResourceID, AccessDate, ExpiryDate).
MembershipTiers: Defines the privileges of different types of memberships (TierID, TierName, MaxLoans, MaxDigitalAccess).
DigitalAccess: Tracks digital access to online resources (AccessID, MemberID, ResourceID, AccessDate, ExpiryDate)
MembershipTiers: Defines borrowing and access privileges based on membership tier (TierID, TierName, MaxLoans, MaxDigitalAccess)

Solve the following statements with the help of SQL Queries
1.Find all digital resources that are currently accessible (i.e., not expired yet).
2.Find members who accessed more than 2 digital resources.
3.Find the average number of digital accesses allowed across all tiers.
4.Show each resource and who currently has it on loan (if anyone).
5.Automatically update the availability status to 'Available' when a loan is deleted (resource returned).
6.Create a procedure to list all resources based on **availability status** (e.g., "Available", "Not Available").

/***/

6 D) **Imagine a Library Management System** which manages member information, library resources, physical loans, digital access, and membership tiers.

Members: Stores personal details of each member (MemberID, Name, Address, Phone, Email, TierID).
Resources: Lists the different resources available in the library, including books and digital content (ResourceID, Title, Author, Type, Format, AvailabilityStatus).
Loans: Records the physical loans of resources issued to members (LoanID, MemberID, ResourceID, LoanDate, ReturnDate).
DigitalAccess: Tracks the digital access of members to online resources (AccessID, MemberID, ResourceID, AccessDate, ExpiryDate).
MembershipTiers: Defines the privileges of different types of memberships (TierID, TierName, MaxLoans, MaxDigitalAccess).
DigitalAccess: Tracks digital access to online resources (AccessID, MemberID, ResourceID, AccessDate, ExpiryDate)
MembershipTiers: Defines borrowing and access privileges based on membership tier (TierID, TierName, MaxLoans, MaxDigitalAccess)

Solve the following statements with the help of SQL Queries
1.Find the number of resources grouped by type and format
2.Retrieve members and their membership tier name (LEFT JOIN)
3.Get list of resources that are currently unavailable
4.Create a function that returns the total number of resources available in a given format (e.g., 'Digital' or 'Physical').

5.Create a procedure to list all resources based on availability status
6.Find total digital accesses per resource