Master of Computer Applications (MCA)

Database Management System Lab (DMCACO108P24)

Self-Learning Material
(SEM 1)

Jaipur National University
Centre for Distance and Online Education
_____
Established by Government of Rajasthan
Approved by UGC under Sec 2(f) of UGC ACT 1956
&
NAAC A+ Accredited

TABLE OF CONTENTS

Experiment 12 Write rite a query to drop the course registration table.	08
Experiment 13 Create a table for storing student addresses.	08– 09
Experiment 14 Write SQL queries to insert Data in to the student addresses table.	09
Experiment 15 Write a query to retrieve a specific student's address by their student ID.	09
Experiment 16 Write a query to delete a student's address as of the student addresses table.	09
Experiment 17 Create a table for storing instructor's office hours.	09
Experiment 18 Write SQL queries to add data into the instruct or office hours table.	10
Experiment 19 Write a query to retrieve all instructions or office hours.	10
Experiment 20 Write a query to retrieve a specific instructor's office hours by their employee ID.	10
Experiment 21 Write a query to update an instructor's office hours.	10
Experiment 22 Write a query to delete an instructor's office hours from the instructor office hours table.	10
Experiment 23 Create a table for storing course prerequisites.	10 – 11
Experiment 24 Consider a table named '\Employees' with the following columns: employee _id, first_ name, last_name, age, department, and salary. Write a"SQL query" to display the first name, last name, and salary of all employees working within the 'Finance' department.	11

Experiment 25 Consider a table named 'Students' by the following columns: student_id, first_name, last_name, age, grade, and course_id. Write a SQL query to compute the average age of students in grade 10.	11
Experiment 26 Consider two tables named 'Orders' and 'Order_items'. The 'orders' table has the columns order_id, , and order_date, customer_id. The 'order_items' table has the column order_id, product_id, quantity, with price. Write a "SQL query" to find the total revenue generated on a specific date (e.g., '2023-03-31').	11
Experiment 27 Consider a table name 'products' by the following columns: product_id,product_name, category, and price. write down a SQL query to display the three most expensive products in each category	11 – 12
Experiment 28 Consider a table name 'customers' by the next columns: customer_id,first_name, last_name, Email, and Phone. Write down a SQL query to update the phone numbers of customers with the lastname 'Smith' by adding a '+1' prefix.	12
Experiment 29 Consider a table name 'books' with the next columns: book_id, title, author,genre, in addition to publication_year. Write down a SQL query to count the number of books published in each genre after 2010.	12
Experiment 30 Consider two tables name 'authors' and 'books'. The 'authors' table have the columns author_id, first_name, and last_name. The 'books' table has the columns book_id,title, author_id, and publication_year. Write down a SQL query to display the list of authors who have published at least three books.	12
Experiment 31 Consider a table name 'inventory' by the following columns: product_id,product_name, quantity, and price. Write down a "SQL query" to display the total value of the inventory(quantity*price)for each product with a value greater than 1000.	12 – 13

Experiment 32 Consider a table name 'events' with the following columns: event_id, event_name,start_date, end_date, and venue. Write down a SQL query to display the events scheduled to occur between '2023-04-02' and '2023-04-30', sorted by start date.	13
Experiment 33 Consider a table named 'users' with the following columns: user_Id, username, email,and registration_dAte. Write a SQL query to delete all users who registered more than two years ago(assuming the current date is '2023-03-31').	13
Experiment 34 Consider a table name 'sales' with the following columns: sale_id, product_id,sale_date, and quantity. Write down a "SQL query" to display total number of sales foreverymonthin2022.	13
Experiment 35 Consider two tables name 'students' and 'enrollments'. The 'students' table has thecolumns student_id, first_name, and last_name.	13
Experiment 36 Consider a table name 'orders' with the following columns: order_id, customer_id,order_date, and total_amount. Write down a "SQL query" to find the total revenue generatedper monthin2022.	13 –14
Experiment 37 Consider a table name 'employees' by the following columns: employee_id,first_name, last_name, hire_date, and salary. Write down a SQL query to display the employees hired with in the last 6 months (assuming the current date is '2023-03-31').	14
Experiment 38 Consider a table name 'cities' with the following columns: city_id, city_name,country, and population. Write down a SQL query to display the top 5 most populous cities in ascending order	14
Experiment 39 Consider a table name 'employees'wIth the following columns: employee_id,first_name, last_name, department, and salary. Write down a SQL query to find the employees with the highest salary in each department	14 – 15
Experiment 40 Consider two tables named 'students' and 'courses'.	15
Experiment 41 Write a query to locate the courses offered in a specific semester.	15

Experiment	Page
Experiment 68 Create An Index on the student table to improve query performance.	25
Experiment 69 Create an Index on the course table to improve query performance.	25
Experiment 70 Analyze the performance of a specific query using the Oracle Explain Plan feature.	25
Experiment 71 Use SQL subqueries to find students who are registered for the same course as a specific student	25
Experiment 72 Use SQL subqueries to find instructors who have taught the same course as a specific instructor.	26
Experiment 73 Write a query to implement pagination for a large result set, such as a list of all students.	26
Experiment 74 Write aquery to implement full-text search for student names or course titles	26
Experiment 75 Use SQL transactions to ensure data integrity when enrolling students in courses or updating their GPA	26 – 27
Experiment 76 Optimize a slow-running query using indexes, subqueries, or other query optimization techniques.	27 – 28
Experiment 77 Use SQL join store cover a list of students and the course they are registered for	28
Experiment 78 Use SQL join store cover a record of instructors and the course they are teaching.	28
Experiment 79 Use SQL join store cover a list of courses and their prerequisites.	28
Experiment 80 Write down a SQL query to create a normalized schema for an online store, including tables for customers, products, orders, and order items. Ensure each table has appropriate primary and foreign keys.	28 – 29

COURSE INTRODUCTION

Database Management Systems (DBMS) are fundamental in today's data-centric world, offering a structured approach to storing, managing, and retrieving vast amounts of information efficiently. A DBMS serves as an intermediary between users and the database, allowing users to define, create, maintain, and control access to data. It abstracts the complexities of data management, enabling seamless interaction with databases for a wide range of applications, from enterprise solutions to personal data handling.

To begin with, students will explore different database models, focusing primarily on the relational model, which organizes data into tables with rows and columns. Understanding database models is crucial as it dictates how data is structured and queried. The course will also delve into data modeling techniques, including the creation of Entity-Relationship (ER) diagrams, which visually represent data entities, their attributes, and the relationships between them. This foundational knowledge will be translated into practical database schemas.

A significant portion of the course will be dedicated to SQL (Structured Query Language), the standard language used to interact with relational databases. Students will develop skills in writing SQL queries to perform essential operations such as data retrieval, insertion, updating, and deletion. Advanced SQL topics, including joins, subqueries, and transactions, will also be covered to provide a deeper understanding of complex data manipulation.

Transaction management is another crucial topic, focusing on how databases handle transactions to maintain data consistency and reliability. Students will learn about transaction properties (ACID properties), concurrency control mechanisms, and recovery strategies to ensure the integrity of database operations.

Additionally, the course will introduce non-relational (NoSQL) databases, which handle unstructured and semi-structured data differently from relational databases. Students will gain insights into various types of NoSQL databases, such as document stores, key-value stores, and column-family stores, and understand their applications in different scenarios.

Course Outcomes:

At the completion of the course, a student will be able to:
1. Demonstrate an understanding of the elementary & advanced features of DBMS & RDBMS.
2. Develop a clear understanding of the conceptual frameworks and definitions of specific terms that are integral to the Relational Database Management
3. Attain a good practical understanding of SQL.
4. Develop clear concepts about Relational Model.
5. Examine techniques pertaining to Database design practices
6. Prepare various database tables and joins them using SQL commands
7. Understand the basic concepts of Concurrency Control & database security
8. Understand the basic concept how storage techniques are used to backup data and maintain data access performance in peak hours
9. Evaluate options to make informed decisions that meet data storage, processing, and retrieval needs.

1. **Install OracleRDBMSand create a Database.**

Installing "Oracle RDBMS" and creating a Database is a multi-step process that can be broken down into the following steps:

● Download Oracle Database Software: Go to the Oracle Database download page(https://www.oracle.com/database/technologies/) and choose the version you want to install (e.g., Oracle Database 19c). Download the proper installer for your operating system (Linux, Windows,ormacOS).

● Install the Oracle Database Software: Follow the setting up steps specific to your Operating system. For detailed instructions, consult Database of Oracle InstallationGuide (https://docs.oracle.com/en/database/oracle/oracle-database/index.html) for your chosen version and operating system.

● create an Oracle Database: After installing "Oracle Database" software, you can createa

"Database using the Database Configuration Assistant" (DBCA). Follow these steps:

a. Open the database Configuration Assistant:

● On Windows, click "Start," search for "Database Configuration Assistant" and click on it.

● On Linux, open a terminal and enter **dbca** to launch the Database ConfigurationAssistant.

b. Choose"Create a Database" and click "Next."

Choose the appropriate template for your database (e.g., General Purpose or TransactionProcessing) and click"Next."

Enter a Global Database Name and System Identifier "SID" for your database. The GlobalDatabase Name should be in the format "database_name.domain_name" (e.g.,"mydb.example.com"). The SID is a unique identifier for your database instance. Click"Next."

c. Choose the storage options for your database, such as file system or Automatic StorageManagement (ASM).Click"Next."

d.Set the memory allocation for your database by choosing either automatic or custom memory management. Click "Next."

e. Choosetheappropriatecharactersetforyourdatabaseandclick"Next."

f. Configure security settings for your database, such as enabling Transparent DataEncryption(TDE)andsettingapasswordfortheSYSandSYSTEMaccounts.Click"Next."

g. C onfigure the management options for your database, such as enabling Oracle EnterpriseManagerDatabaseExpress and setting a password for the DBSNMP account. Click "Next."

h.   R
eview the summary of your database configuration and click "Finish" to create the
database.

- **join to the Oracle Database:** Use SQL*Plus, SQL Developer, or another database
  management tool on the way to connect to your newly created Oracle Database using the
  connection details (SID, username, and password) you provided during the database
  creation process.

Now you have installed Oracle RDBMS and created a database. You can start creating
tables,inserting data, and querying the database as needed

2. Familiarize yourself with Oracle SQL Developer or another Oracle-compatible SQLclient.

"Oracle SQL" Developer is a popular, free "Integrated Development Environment" (IDE) that
simplify working with Oracle databases. It provides a powerful and intuitive interface
formanagingdatabaseobjects, running "SQLqueries" ,and developing "PL/SQL code".

Here's how to get started with "OracleSQLDeveloper ":

1. Download "Oracle SQL" Developer: Go to the Oracle SQL Developer download
   page(https://www.oracle.com/tools/downloads/sqldev-downloads.html) and download
   the appropriate version for your operating system(Windows,macOS,or Linux).

2. Installing Oracle SQL Developer: Follow the installation instruction for your operating
   system   provided   in   the   Oracle   SQL   Developer
   documentation(https://docs.oracle.com/en/database/oracle/sql-developer/index.html).

3. Launch Oracle SQL Developer: Start Oracle SQL Developer by running the
   executable file (sqldeveloper.exe on Windows or sqldeveloper.sh on Linux/macOS)
   located in the installation folder.

4. Create a Database Connection: To connect to your Oracle Database, you need to setup
   a fresh database connection. Click the "+" icon within the "Connections" tab in the left
   pane to open the "NewConnection" dialog.

Fill in the required details, such as:

- ConnectionName:A Unique Name For The Connection.

- Username: The database user account (e.g., SYSTEM or another user account you've
  created).

- Password:The   Password   For   The   User   Account.

- Hostname:Thehostnameor "IPaddress"oftheserver hostyourOracleDatabase.

- Port :Thelistenerportfor your OracleDatabase(default is1521).

- SID or Service Name:The SID or Service Name of your Oracle Database.

Click "Test" to ensure the connection settings are correct, then click "Connect" to establish a connection to the database.

5. Explore Oracle SQL Developer Features: With Oracle SQL Developer, you can manage your database, develop and debug PL/SQL code, run SQL queries, and more.Familiarizeyourself with the following features:

- SQL Worksheet: Write, execute, and save SQL queries, PL/SQL code, and scripts.Access it by right-clicking a connection and selecting "Open SQL Worksheet" or clicking the"SQLWorksheet"button toolbar.

- Object Browser: Explore and manage database objects (tables, indexes, views, etc.)in the "Connections" tab. You can create, edit, and delete objects by right-clicking selecting the appropriate options.

- Data Import and Export: Import data from external files (CSV, Excel, XML, etc.) or export data from tables and views to various file formats. Access these options byright-clickingatableorviewandselecting"ImportData" or"ExportData."

- PL/SQL Debugging: Debug PL/SQL code by setting breakpoints, stepping through code, and examining variable values. Open a PL/SQL object (procedure, function,package, etc.) in the editor, set breakpoints, and click the "Debug" button on the toolbar to start a debugging session.

3. **Create a schema for a university, including tables for students, courses, and instructors.**

   -- Creating table for studentsCREATETABLEstudents(

   Student_id NUMBER PRIMARY KEY,

   First_nameVARCHAR2(50),Last_name VARCHAR2(50),Birth_dateDATE,

    majorVARCHAR2(50)

   );

   -- Creating table for coursesCREATE TABLE courses (

course_id   NUMBER   PRIMARY   KEY,course_name   VARCHAR2(100), course_description VARCHAR2(1000), instructor_id NUMBER);

-- Creating table for instructors CREATE TABLE instructors (instructor_id NUMBER PRIMARY KEY, first_name VARCHAR2(50), last_name VARCHAR2(50), department VARCHAR2(50)

);

-- Add foreign key constraint on courses referencing instructorsALTERTABLE courses

ADDCONSTRAINTfk_instructorFOREIGN   KEY (instructor_id)REFERENCESinstructors(instructor_id);

4. **Write SQL queries insert data into the tables created.**

--Inserting data into instructors table

INSERT   INTO   instructors   (instructor_id,   first_name,   last_name, department)VALUES(1, 'John','Doe','Computer Science');

--Inserting data into student table
INSERT   INTO   students   (student_id,   first_name,   last_name,   birth_date, major)VALUES(1,'Jane','Smith',TO_DATE('1998-05-17','YYYY-MM-

DD'),'Computer

Science');

--Inserting   data   into   courses   table

5. Writea query to retrieve every students from the student table.

SELECT*FROMstudents;

Output:

STUDENT_ID|FIRST_NAME|LAST_NAME| BIRTH_DATE|MAJOR

----

1 |Jane | Smith | 17-MAY-98| Computer Science

6. Create a database called "College" with two tables named "Students" and "Courses".Then, insert sample data into these tables and perform a simple join operation to retrieve student names along with the courses they are enrolled in.

● **Creating the"College" database:**

CREATEDATABASECollege;

● **createthe"Students"table:**

USE College;

CREATE TABLE Students

  (Student_idINTPRIMAR

  YKEY,

  student_Name VARCHAR(50)

  NOT NULL,Course_idINT

);

● **Createthe"Courses"table:**

CREATE   TABLE

  Courses

  (Course_idINTPRI

  MARYKEY,

  course_name VARCHAR(50) NOTNULL

);

● **Insert sample data into the "Students" table:**

INSERT INTO students (student_id, student_name,

Course_id)VALUES (1, 'Alice',101),

   (2,   'Bob',102),

(3,'Charlie',101);

- **insert sample data into the "Courses"table:**INSERT INTO Courses (course_id, course_name)VALUES(101, 'Mathematics'),

  (102,'Physics');

- **Performing a simple join operation to retrieve student names all along with the coursestheyareenrolledin:**

SELECT   student_nameand

course_nameFROMStudents

JOINCoursesONStudents.Course_id withCourses.course_id;

7. **Write a query to find the courses by the highest and lowest number of registered students.**

WITH Course Counts AS(

  SELECT   Course_id,   COUNT(student_id)   AS num_students

  FROM Students
  GROUPBYCourse_id

)

, MinMaxCountsAS(

  SELECT   MIN(num_students)   AS   min_students,   MAX(num_students)   AS max_students FROM Course Counts

)

SELECT   Courses.course_id,   Courses.course_name,

CourseCounts.num_students FROM Courses

JOIN CourseCounts ON Courses.course_id=CourseCounts.course_id

JOIN   MinMaxCounts   ON   CourseCounts.num_students   = MinMaxCounts.min_students   OR CourseCounts.num_students=MinMaxCounts.max_students;

This   Query   Consists   Of   Three   Parts:

1. The **CourseCounts** Common Table Expression (CTE) calculate the number of registered student for each course by grouping the **Students** table by **course_id** and counting the **student_id**s.

2. The **MinMaxCounts** CTE finds the minimum and maximum number of registered students among all courses by selecting the **MIN** and **MAX** of the **num_students** column from the **Course Counts** CTE.

3. The key query joins the **Courses**, **CourseCounts**, and **MinMaxCounts** tables to find and display the course ID, course name, and amount of registered students for thecourses withthehighestandlowestnumberofregisteredstudents.

8. Write a querytoretrievetheaverageGPAofstudentsineach course.

SELECT   Courses.course_id,   Courses.course_name,   AVG(Students.GPA)   AS average_gpa FROM Students

JOIN Courses ON Students.course_id with Courses.course_id

GROUPBY Courses.course_id,Courses.course_name;

This query performs the following operations:

1. join the **Students** and **Course s**tables on the **course_id** column.

2. group the joined records by **course_id** and **course_name**(from the **Courses**table).

3. Calculates the Average GPA students in each group using the AVG() function.

The result of this query will display the course ID, course name, and average GPA of students in each course.

9. Write a query to update a studentGPA.

To revise a student's GPA, you would first need to know the structure of your database,particularly the name of the table that holds the student information and the names of the columns for the student ID and GPA. Assuming the table name is "students" and the columns are"student_id" and"gpa", you could write a query like this:

UPDATE

studentsSETgpa=ne

w_gpa

WHEREstudent_id   =target_student_id;

Replace new_gpa with the updated GPA value (e.g., 3.5) and target_student_id with the ID of the student whose GPA you desire to update (e.g., 12345). Your final query would look like this:

> UPDATE
>
> studentsSETgpa=
>
> 3.5
>
> WHEREstudent_id=12345;

Before running the query, make sure to replace the table and column names if they are different in your database.

10. Write A query to update a course's credit hours.

To update a course's credit hours, you would need to know the structure of your database,specifically the name of the table holding the course information and the names of the columns for the course ID and credit hours. Assuming the table name is "courses" and thecolumnsare"course_id"and"credit_hours", you could write a query like this:

> UPDATE courses
>
> SET credit_hours = new_credit_hours WHERE course_id =
>
> target_course_id;

Replace new_credit_hours with the updated credit hours value (e.g., 4) andtarget_course_id with the ID of the course whose credit hours you want to update (e.g.,'CSCI101'). Your final query would look like this:

> UPDATEcoursesSE
>
> Tcredit_hours=4
>
> WHERE course_id = 'CSCI101';

11. Writea query to delete a student from the student table.

DELETE FROM student

WHEREstudent_id=<student_id_to_delete>;

12. Write a query to drop the course registration table.

> DROPTABLEcourse_registration;

13. Createatableforstoringstudentaddresses.

```
CREATE TABLE student_addresses(address_idSERIALPRIMARYKEY,
    Student_id INT REFERENCES student (student_id),streetVARCHAR(250),
    city VARCHAR(255),stateVARCHAR(255),
    postal_code VARCHAR(255),countryVARCHAR(255)
);
```

14. Write SQLqueriestoinsertDataintothe student addressestable.

```
INSERT INTO student_addresses (student_id, street, city, state, postal_code,
country)VALUES(<student_id>,                        '<street>','<city>',
'<state>','<postal_code>','<country>');
```

15. Write aquerytoretrieveaspecificstudent'saddressbytheirstudentID.

```
SELECT*FROMStudent_addresses
WHEREStudent_id=<Student_id_to_search>;
```

16. Write aquerytodeleteastudent'saddressas ofthestudentaddressestable.

```
DELETEFROMstudent_addresses
WHEREStudent_id=1;--Replace1withthedesiredstudentID
```

17. Create a table for storing instructor's office hours.

```
CREATE TABLE instructor_office_hours(id SERIAL PRIMARYKEY,
    instructor_id    INT    NOT    NULL,day_of_week    VARCHAR(15)    NOT
    NULL,start_timeTIMENOTNULL,
    End_timeTIMENOTNULL
```

18. Write SQL queries add data into the instructor office hours table.

INSERT INTO instructor_office_hours (instructor_id, Day_of_week, start_time, end_time) VALUES (1,'Monday','10:00:00', '11:00:00');

INSERT INTO instructor_office_hours (instructor_id, Day_of_week, start_time, end_time)VALUES (2, 'Tuesday', '14:00:00', '16:00:00');

INSERT INTO instructor_office_hours (instructor_id, Day_of_week, start_time, end_time)VALUES(1,'Thursday','10:00:00', '12:00:00');

19. Write a query to retrieve all instructor office hours.

SELECT*FROMinstructor_office_hours;

20. Writea query to retrieve a specific instructor's office hours their employeeID.

SELECT*FROMinstructor_office_hours

WHEREinstructor_id=1;--Replace1 with the desired instructor ID

21. Write A Query To Update An Instructor's Office Hours.

UPDATEinstructor_office_hours

SETStart_time= '11:00:00',end_time='13:00:00'

WHERE id=1;--Replace1 with thedesiredofficehours record ID

22. Write a query to delete an instructor's office hours from the instructor office hours table.

DELETEFROM instructor_office_hours

WHERE id=1;--Replace1 with the desired office hours record ID

23. Create   a   A   table   for   storing   course   prerequisites.

CREATE TABLE Course_prerequisites(id SERIAL PRIMARYKEY,

course_id INT NOT NULL, prerequisite_id INT NOT NULL

);

24. **Consider a table named '\Employees' with the following columns: employee_id,first_name, last_name, age, department, and salary. Write a "SQL query" to display the first name, last name, and salary of all employees working within the 'Finance Department.**

SELECT   First_name   ,last_name,

salaryFROMEmployees

WHERE
Department='Finance';

25. **Consider a table named 'Students' by the follow columns: student_id, first_name,last_name, age, grade, and course_id. Write a SQL query to compute the average age of students grade 10.**

SELECT AVG(age) AS average_age FROM Students

WHERE grade=10;

26. **Consider two tables named 'Orders' and 'Order_items'. The 'orders' table has the columns order_id, , and order_date, customer_id. The 'order_items' table has the column order_id, product_id, quantity, with price. Write a "SQL query" to find the total revenue generated on a specific date (e.g., '2023-03-31').**

SELECT SUM(quantity * price) AS total_revenue FROM orders
JOIN orders.order_id ON order_items = order_id.order_item WHERE order_date
='2023-03-31';

27. **Consider a table name 'products' by the following columns: product_id, product_name, category, and price. write down a SQL query to display the three most expensive products in each category.**

SELECT   p1.product_id,   p1.product_name,   p1.category,

p1.price   FROM   productsp1

WHERE (SELECT COUNT(*)

FROM productsp2

WHERE p2.category= P1.category ANDp2.price> P1.price

) < 3

ORDERBY p1.category,p1.priceDESC;

28. **Consider a table name 'customers' by the next columns: customer_id,first_name, last_name, Email, and Phone. Write down a SQL query to update the phone numbers of customers with the last name 'Smith' by adding a '+1' prefix.**

UPDATE customers
SET phone = CONCAT('+1', phone) WHERE last_name = 'Smith';

29. **Consider a table name 'books' with the next columns: book_id, title, author,genre, in addition to publication_year. Write down a SQL query to count the number of books published in each genre after 2010.**

SELECT genre, COUNT(*) AS book_count FROM books
WHERE publication_year> 2010 GROUPBYgenre;

30. **Consider two tables name 'authors' and 'books'. The 'authors' table have the columnsauthor_id, first_name, and last_name. The 'books' table has the columns book_id,title, author_id, and publication_year. Write down a SQL query to display the list of authors who have published at least three books.**

"SELECT a.author_id, a.first_name, a.last_name, COUNT(b.book_id) AS book_count FROM authors a
JOIN books b ON a.author_id = b.author_idGROUP BY a.author_id, a.first_name, a.last_nameHAVINGCOUNT(b.book_id) >=3;"

31. **ConsIder a table name 'inventory' by the following columns: product_id,product_name, quantity, and price. Write down a "SQL query" to display the total value of the inventory(quantity*price)for each product with a value greaterthan1000.**

SELECTproduct_id, product_name, quantity, price, (quantity*price) AS inventory_value

FROM inventory
WHERE (quantity*price)>1000;

32. **Consider a table name 'events' with the following columns: event_id, event_name,start_date, end_date, and venue. Write down a SQL query to display the events scheduled to occur between '2023-04-02'and'2023-04-30', sortedbystart_date.**

SELECT event_id, event_name, start_date, end_date, venue FROM events
WHERE start_date BETWEEN '2023-04-02'AND'2023-04-30'
ORDERBY start_date;

33. **Consider a table named 'users' with the following columns: user_Id, username, email,and registration_dAte. Write a SQL query to delete all users who registered more than two years ago(assuming the current date is '2023-03-31').**

DELETE FROM users
WHERE registration_date<DATE_SUB ('2023-03-31',INTERVAL 2 YEAR);

34. **Consider a table name 'sales' with the following columns: sale_id, product_id,sale_date, and quantity. Write down a "SQL query" to display the total number of sales for every month in 2022.**

"SELECT YEAR (sale_date) AS sale_y
Eear , MONTH(sale_date) AS sale_month, COUNT(*) AS sale_count
FROM sales
WHERE YEAR(sale_date) = 2022 GROUPBYsale_year,sale_month;"

35. **Consider two tables named 'students' and 'enrollments'. The 'students' table has thecolumns student_id, first_name, and last_name. The 'enrollments' table has the columns enrollment_Id, student_id, course_id, and semester. Writedown a SQL query to display the list of students who are not enrolled in a few courses for the 'Spring 2023 semester.**

"SELECT   s.student_id,   s.first_name,
s.last_name FROM students
LEFT JOIN enrollments e ON s.student_id = e.student_id AND e.semester = 'Spring 2023 WHERE e.enrollment_idISNULL;"

36. **Consider a table name 'orders' with the following columns: order_id, customer_id,order_date, and total_amount. Write down a "SQL query" to find the total   revenue   generated   per   month   in   2022.**

"SELECT YEAR(oRder_date) AS order_year, MONTH(oRder_date) AS order_month,SUM(total_amount) AS monthly_revenue
FROMoRders
WHERE YEAR(order_date) = 2022 GROUPBY order_year, order_month;"

37. **Consider a table name 'employees' by the following columns: employee_id, first_name, last_name, hire_date, and salary. Write down a SQL query to display the employees hired within the last 6 months(assuming the current date is'2023-03-31').**

"SELECT employee_id, first_name, last_name, hire_date, salary FROM employees
WHERE hire_date>DATE_SUB('2023-03-31', INTERVAL 6 MONTH);"

38. **Consider a table named 'cities' with the following columns: city_id, city_name,country, and population. Write down a SQL query to display the top 5 most populous cities in ascending order.**

SELECT city_id, city_name, country, population FROM cities
ORDER BY population DESC LIMIT 5;

39. **Consider a table name 'employees' with the following columns: employee_id, first_name, last_name, department, and salary. Write down a SQL query to find the employees with the highest salary in each department.**

Step1:
First, we need to find the highest salary for each department. To do this, we use the GROUP BY clause to group the records in the department and the MAX() function to get the greatest salary in each group.

SELECT department, MAX(salary) AS highest_salary FROM employees
GROUP BY department;

Step2:
Now that we have the highest salary for each department, we need to join the result of the previous query with the original 'employees' table to get the employee details.

SELECT e.employee_id, e.department, e.first_name, e.salary FROM employees ,e.last_name JOIN (
 SELECT   department,   MAX   (Salary)   AShighest_Salary

FROM employees GROUP BY department
)ON e.department=d.department AND e.salary=d.highest_salary;

The inner query (subquery) calculates calculates the highest salary designed for each department, and the outer query joins the 'employees' table with the result of the subquery to get the employee details.

40. **Consider two tables named 'students' and 'courses'. The 'students' table has the columns first_name, student_id, and last_name . The 'courses' table has the columns course_id, course_name, with instructor. Write a SQL query to find the students who have not taken any courses taught by a specific instructor (e.g.,'JohnSmith').**

Step 1:
Filter the 'courses' table to get the courses taught by the specific instructor.SELECT course_id
FROM courses
WHERE instructor='JohnSmith';

Step2:
Join the 'students' table by the 'courses' table using a LEFT JOIN to get the list of students who have taken courses taught by the specific instructor. Filter the result to include only students who haven't taken any of the instructor's courses.

SELECT DISTINCT s.student_id, s.first_name, s.last_name FROM students
LEFT JOIN courses c ON s.course_id = c.course_id AND c.instructor = 'John Smith' WHERE c.course_id IS NULL;

The LEFT JOIN ensures that all students are included in the result, even if they haven't taken any courses taught by the specific instructor. The DISTINCT keyword is used to remove duplicate entries in case a student is enrolled in multiple courses not taught by the instructor.

41. **Write a query to locate the courses offered in a specific semester.**

SELECT*FROM

coursesWHERE

semester='Fall2023';

42. **Write a query to find the instructor's teaching specific course.**

"SELECT i.instructor_id ,i.instructor_name FROM instructors JOIN course_instructors.ci ON i.instructor_id = ci.instructor_id WHERE ci.course_id = 'CS101' ;"

43. Write a query to locate the students by the highest GPA in a specific course.

SELECT s.student_id, s.student_name, s.gpa FROM students sJOINcourse_registrationscr ON

s.student_id = cr.student_id WHERE cr.course_id ='CS101'ANDs.gpa =(

SELECT MAX(gpa) FROM students st

JOIN course_registrationscrt ON st.student_id = crt.student_id WHERE crt.course_id =

'CS101'

);

44. Writea query to find courses by means of non registered students.

"SELECT c.course_id, c.course_name FROM courses

LEFT JOIN course_registrationscr ON c.course_id = cr.course_id WHERE cr.student_id IS

NULL;"

45. Writea Querytolocatethe top 4 students with the highest GPA.

SELECT id, name, GPAFROMstudentsORDER By GPA DESC LIMIT 4 ;

46. Write a query to find theTop 5 course with the maximum averageGPA.

"SELECT c.id, c.name, AVG(r.grade) as average_gpaFROMcourses c

JOiN registrations r ON c.id = r.course_idGROUPBYc.id,c.name

ORDER BY average_gpa DESC

LIMIT   5;"

47. Write a query to find theTop 5 instructors with the HighestaveragestudentGPA.

SELECT i.id, i.name, AVG(r.grade) as

average_gpaFROM instructors i

JOIN course_instructors ci ON i.id =

ci.instructor_idJOIN registrations r ON

ci.course_id = r.course_idGROUPBYi.id,i.name

ORDER BY average_gpa

DESC LIMIT 5;

48. Create a view to display the student ID, name, and total credit hours of the course they are registered for.

CREATEVIEWstudent_credit_hoursAS

SELECT s.id as student_id, s.name as student_name, SUM(c.credit_hours) astotal_credit_hours

FROMstudentss

JOIN registrations r ON s.id =

r.student_idJOIN courses c ON

r.course_id = c.idGROUPBYs.id,s.name;

49. Create a view to display the instructor ID, name, and total credit hours of the courses they are teaching.

CREATE VIEWinstructor_credit_hoursAS

SELECT i.id as instructor_id, i.name as instructor_name, SUM(c.credit_hours) astotal_credit_hours

FROM instructorsi

JOIN course_instructors ci ON i.id =

ci.instructor_idJOINcourses cONci.course_id= c.id

GROUPBYi.id,i.name;

50. Createastoredproceduretoenrollastudentinacourse.

```
CREATE          PROCEDURE
EnrollStudent@StudentIDINT,
@CourseID
INTAS
BEGIN
INSERT   INTO   Enrollment   (StudentID,
CourseID)VALUES(@StudentID,@CourseID)
;
END
;GO
```

51. Createastoreproceduretodropacourseused forastudent.

```
CREATE          PROCEDURE
DropCourse@studentIDINT,
@courseID
INTAS
BEGIN
DELETEFROM Enrollment
WHERE   StudentID   =   @StudentID   AND   courseID   =
@courseID;END;
GO
```

52. Create a stored procedure to add a fresh course to the course table.

```
CREATE PROCEDURE AddCourse
@CourseID INT,
@CourseNameNVARCHAR(25
5),@CreditHoursINT
ASBE
GIN
```

```
INSERT             INTO        Courses        (CourseID,        CourseName,
CreditHours)VALUES(@CourseID,@CourseName,@CreditHours);
    END;GO
```

53. Create a stored procedure to delete courses from the course table.

```
CREATE PROCEDURE DeleteCourse@courseID INT
ASBEGIN
DELETEFROMCourses
WHERE CourseID = @CourseID;END;
GO
```

54. Create a stored procedure to add an instructor to the instructor table.

```
CREATE                                              PROCEDURE
AddInstructor@InstructorIDINT,@InstructorNameNVARCHAR(255),@OfficeHoursNV
ARCHAR(255)
ASBEGIN
INSERT         INTO         Instructors      (InstructorID,      InstructorName,
OfficeHours)VALUES(@InstructorID,@InstructorName,@OfficeHours);
END;GO
```

55. Create a stored procedure to delete instructors from the instructor table.

```
CREATE PROCEDURE DeleteInstructor
@InstructorID INTAS
BEGIN
DELETEFROMInstructors
WHERE InstructorID = @InstructorID;END;
GO
```

56. Createa stored procedure to revise astudent'sGPA.

CREATE PROCEDURE UpdateStudentGPA@StudentIDINT,

@NewGPADECIMAL(4, 2)AS

BEGIN

UPDATEStudents

SET GPA=@NewGPA

WHERE StudentID = @StudentID;END;

GO

57. Createastoreproceduretobring up to dateacourse'scredithours.

CREATE PROCEDURE UpdateCourseCreditHours@CourseID INT,

@NewCreditHours INTAS

BEGIN

UPDATECourses

SET CreditHours = @NewCreditHoursWHERECourseID=@CourseID;

END

;GO

58. Createastoredproceduretoupdateaninstructor'sofficehours.

```
CREATE                    PROCEDURE
UpdateInstructorOfficeHours@InstructorIDINT,
@NewOfficeHoursNVARCHAR(2
55)AS
BEGIN
UPDATEInstructors
SET         OfficeHours        =
@NewOfficeHoursWHERE
InstructorID = @InstructorID;END;
GO
```

59. Create a function to compute the average GPA of students in a specific course.

```
CREATE      FUNCTION      AvgGPAByCourse
(@CourseID INT)RETURNS DeCIMAL(4,2)
ASBE
GIN
RETURN (
SELECT AVG(GPA)
FROMStudents
JOIN     EnrollMent     ON     Students.StudentID     =
Enrollment.StudentIDWHEREEnrollMent.CoUrseID=@CoUrse
ID
);EN
D;G
O
```

60. Createafunctiontocomputethetotalcredithoursearnedbyaspecificstudent.

Assuming you have a table enrollments by columns student_id, course_id, andcredit_hours, the function to calculate the total credit hours earned by a specific students can be created as follows:

CREATE FUNCTION total_credit_hours_student(student_idInT) RETURNS INT

AS $$DECLARE

 total_hours

INT;BEGIN

 SELECT SUM(credit_hours)INTOtotal_hoursFROMenrollmentsWHEREstudent_id= $1;

 RETURN

total_hours;END;

$$LANGUAGEplpgsql;

61. Create a function to calculate the total credit hours taught by a specific instructor.

To calculate the whole credit hours taught by a specific instructor, you can create a similar function. Assuming you have a table courses with columns instructor_id andcredit_hours:

CREATE FUNCTION total_credit_hours_instructor(instrUctor_id INT) RETuRNS INT

AS $$DECLARE

 TOTAl_hours

INT;BEiIN

 SELECT SUM(credit_hours) INTO tOtal_hours FROM courses WHERE instructor_id

 = $1;RETURNtotal_hours;

END;

$$LANGUAGEplpgsql;

62. Create a trigger to update the total credit hours earned by a student when they enroll or drop a course.

Assuming you have a table students with columns id and total_credit_hours, create a trigger to update the total credit hours earned by a student when they enroll or drop a course:

CREATEORREPLACEFUNCTIONupdate_student_credit_hours()RETURNSTRIGGERAS $$BE

GIN

IF(TG_oP='INSERT')THEN

UPDATE students SET total_credit_hours = total_credit_hours + NEW.credit_hoursWHEREid=NEW.student_id;

ELSIF (TG_OP = 'DELETE')THEN

UPDATE students SET total_credit_hours = total_credit_hours - OLD.credit_hoursWHEREid=OLD.student_id;

ENDIF;RET

URN

NULL;END;

$$LANGUAGEplpGsql;

CREATE   TRIGGER

update_student_credit_hours_triggerAFTER   INSERT

OR DELETE ONenrollments

FOR each ROW EXECUTEFUNCTIONupdate_student_credit_hours();

63. **Create a trigger to update the total credit hours earned by a student when they enroll or drop a course.**

Assuming you have a table instructors with columns id and total_credit_hours_taught,create a trigger to update the total credit hours taught by an instructor when they areassignedorremovedfrom acourse:CREATEOR REPLACEFUNCTION Update_instructor_credit_hours()RETURNS TRIGGER AS $$BE

GIN

IF(TG_OP='INSERT')THEN

UPDATE instructors SET total_credit_hours_taught = total_credit_hours_taught +NEW.credit_hoursWHERE id = NEW.instructor_id;

ELSIF (TG_OP = 'DELETE')THEN

UPDATE instructors SET total_credit_hours_taught = total_credit_hours_taught -OLD.credit_hoursWHEREid=OLD.instructor_id;

```
   ENDIF
;RETURN
  NULL;
  END ;
  $$LANGUAGEplpGsql;


  CREATE                                    TRIGGER
  update_instructor_credit_hours_triggerAFTER    INSERT
  OR DELETE ONcourses
  FOREACHROWEXECUTEFUNCTIONupdate_instructor_credit_hours();
```

64. Create a table for storing user login logs.

```
CREATE                TABLE
  user_login_logs(idSERIAl
  PRIMARYKEY ,
  user_idINTNOTNULL,
  login_timestampTIMESTAMPNOTNULL

);
```

65. Write SQL queries to add data into the user login logs table.INSERT   INTO   user_login_logs   (user_id, login_timestamp)VALUES (1, '2023-03-3110:00:00');

```
  INSERT      INTO      user_login_logs      (user_id,
  login_timestamp)VALUES (2, '2023-03-3110:05:00');


  INSERT      INTO      user_login_logs      (user_id,
  login_timestamp)VALUES (1, '2023-03-3114:00:00');
```

66. Writea query to retrieve a specific user's login logs BytheiruserID.

```
SELECT  *FROMuser_login_logs

WHERE                                                   User_id=1;--Replace1withthedesireduserID
```

67. Writea querytodeleteaspecificuser'sloginlogsfromtheuserloginlogstable.

DELETEFROMuser_login_logs

WHERE user_id=1;--Replace1withthedesireduserID

68. Create An Index on the student table to improve performance.

CREATEINDEX idx_student_last_Name

ON student (last_name); -- Replace 'student' with your actual student table and'last_name'withthedesiredcolumn

69. CreateanIndexonthecoursetableto improve query performance.

CREATE INDEXidx_course_name

ON course (name); -- Replace 'course' with your actual course table and 'name' with the desired column

70. AnalyzetheperformanceofaspecificqueryusingtheOracleExplainPlanfeature.

To analyze the concert of a specific query, you can use the EXPLAIN PLAN statement inOracle. Here is a case of how to use the EXPLAIN PLAN feature:

-- Replace the SELECT statement with your specific

queryEXPLAINPLAN FOR

SELECT *FROMuser_login_logs WHEREuser_id=1;

-- To view the output of the EXPLAIN PLAN, you can query the

PLAN_TABLE:SELECT*FROM TABLE(DBMS_XPLAN.DISPLAY());

71. Use SQL subqueries to find students who are registered for the same course as a specific student.

SELECT   DISTINCT   s2.student_id,

s2.student_nameFROM   course_registrationAS

cr1

JOIN   course_registration   AS   cr2   ON   cr1.Course_id   =

cr2.course_idJOINstudent ASs2ONcr2.stuDent_id= s2.student_id

WHEREcr1.student_id=<specific_stuDent_id>ANDcr1.student_id!=s2.student_id;

72. Use SQL subqueries to find instructors who have taught the same course as a specific instructor.

"SELECT DISTINCT i2.instructor_id,

i2.instructor_nameFROM courseAs c1

JOINcourseAS c2ONc1.Course_id=c2.course_id

JOINinstructorASi2ONc2.instructOr_Id = i2.instructor_id

WHERE Ec1.instructor_id=<specific_instructor_id>ANDc1.instructor_id!=i2.instructor_id;"

73. Write a query to implement pagination for a large result set, such as a list of all students.

SELECT * FROM

studentORDER BY

student_id

LIMIT <page_size> OFFSET <offset>;

Replace <page_size> with the number of records per page and <offset> with the starting record number forthepage (e.g.,(page_number - 1)*page_size).

74. Write a query to implement full-text search for student names or course titles.

--For

studentnamesSELECT*F

ROMstudent

WHEREto_tsvector('enGlish',student_name)@@to_tsquery('english','<search_query>');

--For

coursetitlesSELECT*F

ROMcourse

WHERE to_tsvector('english', course_title) @@ to_tsquery('english',

'<search_query>');Replace<search_query>with the text you want to search for.

75. Use SQL transactions to ensure data integrity when enrolling students in courses or updating theirGPA.

-- Enrolling a student in a

courseBEGIN;

```
INSERT    INTO    Course_registration    (student_id,
course_id)VALUES(<stuDent_id>,<course_id>);
UPDATEstudent
SET enrolled_courses = enrolled_courses +
1WHERE          student_id          =
<student_id>;COMMIT;


-- Updating   a   Student's
GPABEGIN;
UPDATEstudent
SET gpa=<new_gpa>
WHEREstudent_id=<student_id>;
-- Any other related updates should be placed here, e.g., updating class rank,
etc.COMMIT;
```

76. Optimize a slow-running query using indexes, subqueries, or other query optimization techniques.

Here's an example of optimizing a slow-running query that finds students enrolled in a precise course:

Original query:

```
"SELECT *
FROM student
WHERE    Student_id
   IN
   (SELECTstudent_id
   FROMcourse_registration
   WHERECourse_id=<specific_course_id>
);"
```

Optimized Query Using JOIN:

```
"SELECT s.*
FROM                                                           StudentASs
```

JOIN course_registration AS cr ON s.student_id = cr.student_idWHEREcr.course_id =<specific_course_id>;"

Additionally, you could create an index on course_registration(course_id) andstudent(student_id)to speed up the join operation:

"CREATE INDEX idx_course_registration_course_id O N Course_registration(course_id);

CREASTEINDEXidx_student_id ONstudent(student_id);"

77. UseSQL joins to recover a list of students and the course they are registered for.

"SELECT s.student_id, s.student_name, c.course_id, c.course_titleFROM studentASs

JOIN course_registration AS cr ON s.student_id = cr.student_idJOINcourseAS cONcr.course_id= c.course_id;"

78. UseSQLjoinstorecoverarecord ofinstructorsandthecoursetheyareteaching.

"SELECT i.instructor_id, i.instructor_name, c.course_id, c.course_titleFROM instructor AS i

JOINcourseASc ONi.instructor_id= c.instructor_id;"

79. UseSQL joins to recover a list of courses and their prerequisites.

SELECT c1.course_id AS course_id, c1.course_title AS course_title,c2.course_idASprerequisite_id,c2.course_titleASprerequisite_ title

FROMcourse A Sc1

JOIN course_prerequisite AS cp ON c1.course_id = cp.course_idJOINcourse A S c2O Ncp.prerequisite_id= c2.course_id;

80. Write down a SQL query to create a normalized schema for an online store, including tables for customers, products, orders, and order items. Ensure each table has appropriate primary and foreign keys.

"CREATE TABLE Customers (

  customer_idINTPRIMARYKEYAUTO_INCREMENT,

first_name VARCHAR(250) NOT NULL,last_name VARCHAR(250) NOT NULL,emailVARCHAR(250)UNIQUENOTNULL

);

CREATE TABLE Products (

 Product_idINT PRIMARYK E YAUTO_INCREMENT,

 product_name VARCHAR(255) NOT NULL,

 PriceDECIMAL(10,2)NOTNULL,

 stock_levelINTNOTNULL

);

CREATE TABLE Orders (

 order_idINT PRIMARYKEY AUTO_INCREMENT,

 customer_id INT NOT NULL,order_date DATE NOT NULL,total_priceDECIMAL(10 ,2)NOTNULL,

 FOREIGNKEY(customer_Id)REFERENCES Customers(customer_id)

);
CREATETABLEOrder_Items(

 order_item_id INT PRIMARY KEY AUTO_INCREMENT,order_idINT NOTNULL,

 product_id INT NOT NULL,quantityINTNOTNULL,

 FOREIGN   KEY   (order_id)   REFERENCES

 Orders(order_Id),FOREiGNKEY(product_id)REFERENCESProducts(product_id),)"

81. Write down a "SQL query " to create an index On the product name in the "Products" table,assuming the product name column is called"product_name".

CREATE INDEXidx_product_nameONProducts(product_name);

82. Write a "SQL query" to find the top 3 Best-selling products in the online store, based on the total quantity sold.

"SELECT p.product_id, p.product_name, SUM(oi.quantity) as

total_soldFROMProductsp

JOIN order_Itemsoi ON p.product_id =

oi.product_idGROUPByp.product_id,p.proDuct_name

ORDER BY total_sold DESCLIMIT 3;"

83. Write a "SQL query" to find the total revenue generated by the online store for a given date range,using the"Orders"and "Order_Items"tables.

SELECT SUM(o.total_price) as total_revenueFROMOrderso

WHEREo.order_dateBETWEEN start_dateANDend_date;

84. Create a stored procedure to insert a new customer record into the "Customers' ' table,checking for duplicate email addresses and returning an appropriate error message if duplicate is found.

DELIMITER//

CREATE PROCEDURE InsertCustomer(INp_first_nameVARCHAR(255),

 IN

 p_last_nameVARCHAR(255),INp_emailVARCHAR(255),OUTp_statusVARCHAR(255)

)BEGIN

 DECLAREemail_existsINTDEFAULT0;

SELECT COUNT(*) INTO email_existsFROMCustomers

WHERE email=p_email;

IFemail_exists>0THEN

   SET p_status = 'Error: Email address already exists.';ELSE

   INSERT   INTO   Customers   (First_name,   last_name,

   email)VALUES(p_First_name,p_last_name,p_email);

   SET p_status = 'Success: Customer added.';ENDIF;

END//DELIMITER;

85. Create a stored procedure to update the stock level meant for a product in the "Products"table,considering the stock level should not be less than zero.

DELIMITER//

CREATE PROCEDURE Update STOCK Level(INp_product_idINT,

 INp_STOCK_level INT,

 OUTp_statusVARCHAR(255)
)BEGIN

 IFp_stock_level< 0THEN

   SET p_status = 'Error: Stock level cannot be negative.';ELSE

   UPDATEproducts

   SET stock_level = p_stock_levelWHEREproduct_id=p_product_id;

   SET p_status = 'Success: Stock level

   updated.';ENDIF;

END//DELIMI

TER;

86. Write a SQL query to implement a transaction that inserts a new order and its associated order items into the "Orders" and "Order_Items" tables, ensuring data consistency.

BEGINTRANSACTION;

--InsertthenewordicerintotheOrderstable

INSERT   INTO   Orders   (order_id,   customer_id,

order_date)VALUES(NEW_ORDER_ID,CUSTOMER_ID,'Y

YYY-MM-DD');

-- Insert the associated order items into the Order_Items table
INSERT INTO Order_Items (order_id, product_id, quantity, price)
VALUES(NEW_ORDER_ID,PRODUCT_ID_1,QUANTITY_1,PR
ICE_1),

   (NEW_ORDER_ID, PRODUCT_ID_2, QUANTITY_2,

   PRICE_2),(NEW_ORDER_ID,PRODUCT_ID_3,QUANTI

   TY_3,PRICE_3);

-- Check for errors and commit the transaction if no errors
occurredIF@@ERROR=0

  COMMIT

TRANSACTION;ELSE

  ROLL BACK TRANSACTION;

87. Write a SQL query to Create a view that displays the whole revenue generated by each customer.

"CREATE VIEW Customer_RevenueAS

SELECT c.customer_id, c.customer_name, SUM(oi.price * oi.quantity) AS

total_revenue FROM Customersc

JOIN   Order   so   ONc.Customer_id=o.customer_id

JOIN Order_ItemsoI ON o.order_id =

oi.order_idGROUPBYC.customer_id,c.customer_name;"

88. Implement a trigger that automatically calculates and updates the total price of an order in the "Orders" table when a new record is inserted into the "Order_Items"table.

"CREATE TRIGGER Update_Order_Total AFTER INSERT ON Order_Items

FOR EACH ROWBEGIN

 UPDATEOrders

 SET total_price = total_price + (NEW.quantity * NEW.price)WHERE

 order_id=NEW.order_id;

END;"

89. Write a SQL query to create a schema for a many-to-many relationship among students and courses, using a junction table for enrollments.

-- Create Students table "CREATE TABLE students (student_id INT

PRIMARY KEY,

 student_nameVARCHAR(255)NOTNULL

);
-- Create Courses tableCREATE TABLE Courses

(course_idINTPRIMARYKEY,

 course_name VARCHAR(255)NOTNULL

);

--CreateEnrollmentsjunction table

CREATE TABLE Enrollments (student_idINT,

course_Id INT,enrollment_dateDATE,

PRIMARYKEY(student_id, course_id),

FOREIGN KEY (Student_id) REFERENCES Students

(student_id),FOREIGNKEY(course_id)REFERENCES Courses (course_id)

);"

90. Write a SQL query to uncover the average grade of all students in a specific course, using the "Students", "Courses", and"Enrollments"tables.

"SELECT AVG(Enrollments.grade) AS average_grade FROM Students

JOIN Enrollments ON Students.Student_id = Enrollments.student_id JOIN Courses ON

Enrollments.course_id = Courses.course_id WHERE Courses.course_name = 'SpecificCourseName';"

91. Write down a SQL query to identify clients who have placed more than 5 orders in the online store,using the "Customers" and "Orders" tables.

"SELECT Customers.customer_id, Customers.customer_name FROM Customers

JOIN Orders ON Customers.customer_id = Orders.customer_idGROUP BY

Customers.customer_id, Customers.customer_name HAVING COUNT(Orders.order_id) >

5;

92. Write a SQL query to Find the products that have not been ordered in the last 30 days,using the "Products","Orders",and "OrderItems"tables.

"SELECT Products.product_id, Products.product_name FROM Products

LEFT JOIN Order_Items ON Products.Product_id =Order_Items.product_id

LEFT JOIN Orders ON Order_Items.order_id=Orders.order_id

WHERE Orders.order_date< CURRENT_DATE - INTERVAL '30 days' OR
Orders.order_id ISNULL

GROUP BY Products.product_id,Products.product_name;"

93. Implement a trigger that checks the stock point of a product before inserting a new record into the "Order_Items" table. If the stock level is insufficient, cancel the insertion and return an error message.

"DELIMITER//

CREATE TRIGGER

check_stock_level BEFORE INSERT ON Order_Items FOR EACH ROW

BEGIN

   DECLARE stock_level INT;SELECT stock INTO stock_level FROM Products

   WHERE Products.ProdUct_id=NEW.prodUct_id;

   IF stock_level<NEW.quantity THEN

   SIGNAL SQLSTATE 45000'

     SET MESSAGE_TEXT = 'Insufficient stock level.';ENDIF;

END;

//DELIMITER :"

94. Create a stored procedure to calculate and update the total price of an order after adding,updating, or deleting an orderitem.

CREATE PROCEDURE Update Total Price(IN order_id INT)BEGIN

```
DECLARE tOtal_price DECIMAL(10,2);

SELECT SuM ( Products.price * Order_Items.quantity) INTO
toTal_priceFROM Order_Items
JOIN Products ON Order_Items.product_id = Products.product_id
WHERE Order_Items.order_id = order_id;


UPDATE OrdErs
SET tOtal_price = tOtal_price
WHERE

 ORders.order_id = ORder_id;
END;
```


95. Write a SQL query to Create an index on the "Orders" table to optimize the search for order placed within a specific date range.

CREATE INDEX idx_orders_order_date ON Orders (order_date);

96. Write a "SQL query" to find the total number of enrollments for each course, sorted by the number of enrollments in descending order.

"SELECT Courses.course_Id, Courses.course_naMe,
COUNT(Enrollments.student_id) asenrollment_count

FROM Courses

JOIN Enrollment ON Courses.course_id = Enrollments.course_id

GROUP BYCourses.course_id,Courses.course_name

ORDER BY enrollment_count DESC;"

97. Create a view to display the number of products sold in each product category,using the"Products" and "Order_Items"tables.

"CREATE VIEWproducts_sold_by_category AS

SELECT   Products.category,   COUNT(Order_Items.product_id)asproducts_sold

FROM Products

JOIN Order_Items ON Products.product_id = Order_Items.product_id GROUP BY

products.category;"

98. Implement a Foreign key constraint on the "Enrollments" table to ensure that a record can only be added if the corresponding student and course IDs exist in the "Students" and"Courses"tables.

"ALTER TABLE Enrollments

ADD FOREIGN KEY (student_id) REFERENCES

students(student_id),ADDFOREIGNKEY(course_id)

REFERENCE Scourses(course_id);"

99. Write a SQL query to identify the top three customers who have spent the most money on top of the online store, based on the total price of their orders.

"SELECT Customers.customer_id, customers.customer_name, SUM(Orders.total_price) astotal_spent

FROM Customers

JOIN Orders ON customers.customer_id = Orders.customer_idGROUP BY

customers.customer_id, Customers.customer_name ORDER BY Total_spent DESC

LIMIT 3;"

100. Write down a SQL query to find the students who have taken at least 1 course from each department, using the "Students", "Courses", "Enrollments", and "Departments" tables.Assume there is a "department_id" column in both the "Courses" and "Departments"tables.

To find the students who have taken at least 1 course from each department, you can usethefollowingSQLquery:

WITH Department Courses AS(

   SELECT DISTINCT department_id, course_idFROM Courses

),

Student Courses AS(SELECT e.student_id, dc.department_id FROM Enrollments

```
        JOIN CoursescONe.course_id =c.courSe_id

        JOIN Department CoursesdcONc.course_id=dc.course_id),

Department CountsAS(

        SELECT COUNT(*) AS department_count FROM Departments

),

StudentDepartmentCounts AS(

        SELECT student_id, COUNT(DISTINCT department_id)

        AS student_department_count FROM Student Courses

        GROUPBY Student_id

)

SELECT s.student_id, s.name FROM Students s

JOIN Student Department Counts secONds.Student_id=sdc.student_id

JOIN Department Counts cONsdc.Student_department_count=dc.department_count;
```