Bachelor of Computer Applications (BCA)

Database Management System Lab

(OBCACO206P24)

Self-Learning Material
(SEM -II)

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 13 Write SQL queries to insert data into the student addresses table.	9
Experiment 14 Write a query to retrieve a specific student's address by their student ID.	9
Experiment 15 Write a query to delete a student's address from the student addresses table.	9
Experiment 16 Create a table for storing instructor's office hours.	9
Experiment 17 Write SQL queries to insert data into the instructor office hours table.	10
Experiment 18 Write a query to retrieve all instructor office hours.	10
Experiment 19 Write a query to retrieve a specific instructor's office hours by their employee ID.	10
Experiment 20 Write a query to update an instructor's office hours.	10
Experiment 21 Write a query to delete an instructor's office hours from the instructor office hours table.	10
Experiment 22 Create a table for storing course prerequisites.	10
Experiment 23 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 in the 'Finance' department”.	11
Experiment 24 Consider a table named “students” with the following columns: “student_id, first_name, last_name, age, grade, and course_id”. Write a SQL query to calculate the average age of students in grade 10.	11

Experiment 25 Consider two tables named “orders” and “order_items”. The “orders” table has the columns “order_id, customer_id, and order_date. The 'order_items' table has the columns order_id, product_id, quantity, and price.” Write a SQL query to find the total revenue generated on a specific date (e.g., '2023-03-31').	11
Experiment 26 Consider a table named 'products' with the following columns: product_id,product_name, category, and price. Write a SQL query to display the three mostexpensive products in each category.	11
Experiment 27 Consider a table named 'customers' with the following columns: customer_id, first_name, last_name, email, and phone. Write a SQL query to update the phonenumbers of customers with the last name 'Smith' by adding a '+1' prefix.	12
Experiment 28 Consider a table named 'books' with the following columns: book_id, title, author, genre, and publication_year. Write a SQL query to count the number of books published in each genre after 2010.	12
Experiment 29 Consider two tables named 'authors' and 'books'. The 'authors' table has the columns author_id, first_name, and last_name. The 'books' table has the columns book_id, title, author_id, and publication_year. Write a SQL query to display the list of authors who have published at least three books.	12
Experiment 30 Consider a table named “inventory” with the following columns: “product_id, product_name, quantity, and price”. Write a SQL query to display the total value of theinventory (quantity * price) for each product with a value greater than 1000.	12
Experiment 31 Consider a table named “events” with the following columns: “event_id, event_name, start_date, end_date, and venue”. Write a SQL query to display the events scheduled to occur between '2023-04-01' and '2023-04-30', sorted by start_date.	13
Experiment 32 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

EXPERT COMMITTEE

Prof. Sunil Gupta
(Computer and Systems Sciences, JNU Jaipur)

Dr. Satish Pandey
(Computer and Systems Sciences, JNU Jaipur)

Dr. Shalini Rajawat
(Computer and Systems Sciences, JNU Jaipur)

COURSE COORDINATOR

Swarnima Gupta
(Computer and Systems Sciences, JNU Jaipur)

UNIT PREPARATION

Unit Writer(s)	Assisting & Proofreading	Unit Editor
Shish Dubey (Computer and Systems Sciences, JNU Jaipur))	Mrs. Rashmi Choudhary (Computer and Systems Sciences, JNU Jaipur)	Dr. Deepak Shekhawat (Computer and Systems Sciences, JNU Jaipur)

Secretarial Assistance

Mr. Mukesh Sharma

COURSE INTRODUCTION

Welcome to the Database Management Systems (DBMS) Lab, a dynamic course designed to bridge the gap between theoretical knowledge and practical application in the field of database management. This lab course provides an immersive experience in working with database technologies and tools that are essential for designing, implementing, and managing modern database systems.

Throughout the course, you will engage in hands-on activities that cover a range of topics from basic SQL queries to advanced database design and administration techniques. By applying concepts learned in lectures to real-world scenarios, you will develop critical skills in data modeling, query optimization, and system performance tuning. This practical approach will not only deepen your understanding of database principles but also prepare you for the challenges faced in professional database management roles.

As you progress through the lab, you'll have the opportunity to work on projects that simulate real-life data management problems, fostering both technical proficiency and problem-solving abilities. The DBMS Lab is designed to equip you with the tools and experience needed to excel in the field of database management and to tackle complex data-driven tasks with confidence.

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. Understand the basic concepts of Concurrency Control & database security
4. Understand the basic concept how storage techniques are used to backup data and maintain data access performance in peak hours
5. Attain a good practical understanding of SQL.
6. Develop clear concepts about Relational Model.
7. Examine techniques pertaining to Database design practices and prepare various database tables and joins them using SQL commands
8. Evaluate options to make informed decisions that meet data storage, processing, and retrieval needs.

Acknowledgements:

The content we have utilized is solely educational in nature. The copyright proprietors of the materials reproduced in this book have been tracked down as much as possible. The editors apologize for any violation that may have happened, and they will be happy to rectify any such material in later versions of this book.

Database Management System Lab

1. Install Oracle RDBMS and 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 appropriate installer for your operating system (Linux, Windows, or macOS).
- Install the Oracle Database Software: Follow the installation steps specific to your operating system. For detailed instructions, consult the Oracle Database Installation Guide "(https://docs.oracle.com/en/database/oracle/oracle-database/index.html)" foryour chosen version and operating system.
- Create an Oracle Database: After installing the Oracle Database software, you can create a

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 Configuration Assistant.

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

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

d. Enter a Global Database Name and SID (System Identifier) for your database. The Global Database 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."

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

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

g. Choose the appropriate character set for your database and click "Next."

h. Configure security settings for your database, such as enabling Transparent Data Encryption (TDE) and setting a password for the SYS and SYSTEM accounts. Click "Next."

i. Configure the management options for your database, such as enabling Oracle Enterprise Manager Database Express and setting a password for the DBSNMP account. Click "Next."

j. Review the summary of your database configuration and click "Finish" to create the database.

- **Connect to the Oracle Database:** Use SQL*Plus, SQL Developer, or another database management tool 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.

Familiarize yourself with Oracle SQL Developer or another Oracle-compatible SQL client.

Oracle SQL Developer is a popular, free integrated development environment (IDE) that simplifies working with Oracle databases. It provides a powerful and intuitive interface for managing database objects, running SQL queries, and developing PL/SQL code.

Here's how to get started with Oracle SQL Developer:

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. Install Oracle SQL Developer: Follow the installation instructions 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 set up a new database connection. Click the "+" icon in the "Connections" tab in the left pane to open the "New Connection" dialog.

Fill in the required details, such as:

- Connection Name: 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: The hostname or IP address of the server hosting your Oracle Database.
- Port: The listener port for your Oracle Database (default is 1521).
- 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. Familiarize yourself 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 "SQL Worksheet" button on the 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 and 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 by right-clicking a table or view and selecting "Import Data" or "Export Data."
- 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.

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

```
-- Creating table for studentsCREATE TABLE students (
 student_id NUMBER PRIMARY KEY,first_name
 VARCHAR2(50), last_name VARCHAR2(50), birth_date
 DATE,
 major VARCHAR2(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
instructorsALTER TABLE courses
ADD CONSTRAINT fk_instructor
FOREIGN KEY (instructor_id)
REFERENCES
instructors(instructor_id);”
```

3. Write SQL queries to insert data into the tables you 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 students 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
INSERT INTO courses (course_id, course_name, course_description, instructor_id)
```

VALUES (1, 'Introduction to Programming', 'Learn the basics of programming in Python.', 1);

4. **Write a query to retrieve all students from the student table.**

   SELECT * FROM students;

   Output:

   STUDENT_ID | FIRST_NAME | LAST_NAME | BIRTH_DATE | MAJOR

   ---

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

5. **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:**

```
CREATE DATABASE College;
```

- **Creating the "Students" table:**

```
 USE College;
CREATE TABLE Students (
   student_id INT PRIMARY KEY,
   student_name VARCHAR(50) NOT NULL,
   course_id INT
);
```

- **Creating the "Courses" table:**

```
 CREATE TABLE Courses (
   course_id INT PRIMARY KEY,
   course_name VARCHAR(50) NOT NULL
);
```

- **Inserting 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);”

- **Inserting 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 along with the courses they are enrolled in:**

“SELECT student_name,

course_nameFROM Students

JOIN Courses ON Students.course_id = Courses.course_id;”

6. Write a query to find the courses with the highest and lowest number of registered students.

```
 WITH CourseCounts AS (
   SELECT course_id, COUNT(student_id) AS num_students
   FROM Students
   GROUP BY course_id
)
, MinMaxCounts AS (
   SELECT MIN(num_students) AS min_students, MAX(num_students) AS max_students
   FROM CourseCounts
)
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) calculates the number of registered students 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 **CourseCounts** CTE.
3. The main query joins the **Courses**, **CourseCounts**, and **MinMaxCounts** tables to find and display the course ID, course name, and number of registered students for the courses with the highest and lowest number of registered students.

7. Write a query to retrieve the average GPA of students in each course.

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

average_gpa FROM Students

JOIN Courses ON Students.course_id = Courses.course_id

GROUP BY Courses.course_id, Courses.course_name;”

This query performs the following operations:

1. Joins the **Students** and **Courses** tables on the **course_id** column.
2. Groups the joined records by **course_id** and **course_name** (from the **Courses** table).
3. Calculates the average GPA of 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.

8. Write a query to update a student's GPA.

To update 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 students
SET gpa = new_gpa
WHERE student_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 want to update (e.g., 12345). Your final query would look like this:

```
UPDATE students
SET gpa = 3.5
WHERE student_id = 12345;
```

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

9. 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 the columns are "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) and target_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:

```
UPDATE  courses
SET credit_hours = 4
WHERE course_id = 'CSCI101';
```

10. Write a query to delete a student from the student table.

```
DELETE FROM student
WHERE student_id = <student_id_to_delete>;
```

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

```
DROP TABLE course_registration;
```

12. Create a table for storing student addresses.

“CREATE TABLE student_addresses

(address_id SERIAL PRIMARY

KEY,

student_id INT REFERENCES student(student_id),

street VARCHAR(255),

city VARCHAR(255),

state VARCHAR(255),

postal_code VARCHAR(255),

country VARCHAR(255)

);”

13. Write SQL queries to insert data into the student addresses table.

INSERT INTO student_addresses (student_id, street, city, state, postal_code, country)

VALUES (<student_id>, '<street>', '<city>', '<state>', '<postal_code>', '<country>');

14. Write a query to retrieve a specific student's address by their student ID.

SELECT * FROM student_addresses

WHERE student_id = <student_id_to_search>;

15. Write a query to delete a student's address from the student addresses table.

“DELETE FROM student_addresses

WHERE student_id = 1; -- Replace 1 with the desired student ID”

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

“CREATE TABLE instructor_office_hours

(id SERIAL PRIMARY KEY,

instructor_id INT NOT NULL,

day_of_week VARCHAR(10) NOT NULL,

start_time TIME NOT NULL,

end_time TIME NOT NULL

);”

17. Write SQL queries to insert data into the instructor office hours table.

“INSERT INTO instructor_office_hours (instructor_id, day_of_week, start_time, end_time)VALUES (1, 'Monday', '09:00:00', '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');”

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

SELECT * FROM instructor_office_hours;

19. Write a query to retrieve a specific instructor's office hours by their employee ID.

SELECT * FROM instructor_office_hours
WHERE instructor_id = 1; -- Replace 1 with the desired instructor ID

20. Write a query to update an instructor's office hours.

UPDATE instructor_office_hours
SET start_time = '11:00:00', end_time = '13:00:00'
WHERE id = 1; -- Replace 1 with the desired office hours record ID

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

DELETE FROM instructor_office_hours
WHERE id = 1; -- Replace 1 with the desired office hours record ID

22. Create a table for storing course prerequisites.

“CREATE TABLE course_prerequisites

(id SERIAL PRIMARY KEY,

course_id INT NOT NULL,

prerequisite_id INT NOT NULL

);”

23. 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 “thefirst name, last name, and salary of all employees working in the 'Finance' department”.

SELECT first_name, last_name, salary

FROM employees

WHERE department = 'Finance';

24. Consider a table named “students” with the following columns: “student_id, first_name,last_name, age, grade, and course_id”. Write a SQL query to calculate the average age of students in grade 10.

SELECT AVG(age) AS average_age

FROM students

WHERE grade = 10;

25. Consider two tables named “orders” and “order_items”. The “orders” table has the columns “order_id, customer_id, and order_date. The 'order_items' table has the columns order_id, product_id, quantity, and price.” Write a SQL query to find the totalrevenue generated on a specific date (e.g., '2023-03-31').

“SELECT SUM(quantity * price) AS total_revenueFROM orders
JOIN order_items ON orders.order_id = order_items.order_idWHERE order_date = '2023-03-31';”

26. Consider a table named 'products' with the following columns: product_id, product_name, category, and price. Write a SQL query to display the three most expensive products in each category.

“SELECT p1.product_id, p1.product_name, p1.category, p1.priceFROM products p1

WHERE (

SELECT COUNT(*)

FROM products p2

WHERE p2.category = p1.category AND p2.price > p1.price

) < 3

ORDER BY p1.category, p1.price DESC;”

27. Consider a table named 'customers' with the following columns: customer_id, first_name, last_name, email, and phone. Write 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';
```

28. Consider a table named 'books' with the following columns: book_id, title, author, genre, and publication_year. Write 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
GROUP BY genre;
```

29. Consider two tables named 'authors' and 'books'. The 'authors' table has the columns author_id, first_name, and last_name. The 'books' table has the columns book_id, title, author_id, and publication_year. Write 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_countFROM authors a
JOIN books b ON a.author_id = b.author_id
GROUP BY a.author_id, a.first_name, a.last_name
HAVING COUNT(b.book_id) >= 3;”
```

30. Consider a table named “inventory” with the following columns: “product_id, product_name, quantity, and price”. Write a SQL query to display the total value of theinventory (quantity * price) for each product with a value greater than 1000.

```
SELECT product_id, product_name, quantity, price, (quantity * price) AS inventory_value
```

```
FROM inventory
WHERE (quantity * price) > 1000;
```

31. **Consider a table named “events” with the following columns: “event_id, event_name, start_date, end_date, and venue”. Write a SQL query to display the events scheduled tooccur between '2023-04-01' and '2023-04-30', sorted by start_date.**

```
SELECT event_id, event_name, start_date, end_date, venue
FROM events
WHERE start_date BETWEEN '2023-04-01' AND '2023-04-30'
ORDER BY start_date;
```

32. **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 thantwo years ago (assuming the current date is '2023-03-31').**

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

33. **Consider a table named “sales” with the following columns: “sale_id, product_id, sale_date, and quantity”. Write a SQL query to display the total number of sales foreach month in 2022.**

```
“SELECT YEAR(sale_date) AS sale_year, MONTH(sale_date) AS sale_month, COUNT(*)
ASsale_count
FROM sales
WHERE YEAR(sale_date) = 2022
GROUP BY sale_year, sale_month;”
```

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

```
“SELECT s.student_id, s.first_name,
s.last_nameFROM students s
LEFT JOIN enrollments e ON s.student_id = e.student_id AND e.semester = 'Spring 2023'
WHERE e.enrollment_id IS NULL;”
```

35. **Consider a table named 'orders' with the following columns: order_id, customer_id, order_date, and total_amount. Write 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
FROM orders
WHERE YEAR(order_date) = 2022
GROUP BY order_year, order_month;
```

36. Consider a table named 'employees' with the following columns: employee_id, first_name, last_name, hire_date, and salary. Write 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,
salaryFROM employees
WHERE hire_date > DATE_SUB('2023-03-31', INTERVAL 6 MONTH);”
```

37. Consider a table named 'cities' with the following columns: city_id, city_name, country, and population. Write 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;
```

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

Step 1:
First, we need to find the highest salary for each department. To do this, we use the GROUP BY clause to group the records by department and the MAX() function to get the maximum salary in each group.

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

Step 2:
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.first_name, e.last_name, e.department, e.salary
FROM employees e
JOIN (
  SELECT department, MAX(salary) AS highest_salary
```

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

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

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

Step 1:
Filter the 'courses' table to get the courses taught by the specific instructor.

```
SELECT course_id
FROM courses
WHERE instructor = 'John Smith';
```

Step 2:
Join the 'students' table with 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_nameFROM students s
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.

40. Write a query to find the courses offered in a specific semester.

```
SELECT * FROM courses

WHERE semester = 'Fall 2023';
```

41. Write a query to find the instructors teaching a specific course.

```
“SELECT i.instructor_id, i.instructor_name FROM instructors i
```

JOIN course_instructors ci ON i.instructor_id = ci.instructor_id

WHERE ci.course_id = 'CS101';”

42. Write a query to find the students with the highest GPA in a specific course.

“SELECT s.student_id, s.student_name, s.gpa FROM students

sJOIN course_registrations cr ON s.student_id = cr.student_id

WHERE cr.course_id = 'CS101' AND s.gpa = (

SELECT MAX(gpa) FROM students st

JOIN course_registrations crt ON st.student_id = crt.student_id

WHERE crt.course_id = 'CS101'

);”

43. Write a query to find courses with no registered students.

“SELECT c.course_id, c.course_name FROM courses c

LEFT JOIN course_registrations cr ON c.course_id = cr.course_id

WHERE cr.student_id IS NULL;”

44. Write a query to find the top 5 students with the highest GPA.

“SELECT id, name,

GPAFROM students

ORDER BY GPA DESC

LIMIT 5;”

45. Write a query to find the top 5 courses with the highest average GPA.

“SELECT c.id, c.name, AVG(r.grade) as

average_gpaFROM courses c

JOIN registrations r ON c.id = r.course_id

GROUP BY c.id, c.name

ORDER BY average_gpa DESC

LIMIT 5;”

46. Write a query to find the top 5 instructors with the highest average student GPA.

“SELECT i.id, i.name, AVG(r.grade) as

average_gpaFROM instructors i

JOIN course_instructors ci ON i.id = ci.instructor_id

JOIN registrations r ON ci.course_id = r.course_id

GROUP BY i.id, i.name

ORDER BY average_gpa DESC

LIMIT 5;”

47. Create a view that displays the student ID, name, and total credit hours of the courses they are registered for.

CREATE VIEW student_credit_hours AS

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

FROM students s

JOIN registrations r ON s.id = r.student_id

JOIN courses c ON r.course_id = c.id

GROUP BY s.id, s.name;

48. Create a view that displays the instructor ID, name, and total credit hours of the courses they are teaching.

CREATE VIEW instructor_credit_hours AS

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

FROM instructors i

JOIN course_instructors ci ON i.id = ci.instructor_id

JOIN courses c ON ci.course_id = c.id

GROUP BY i.id, i.name;

49. Create a stored procedure to enroll a student in a course.

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

50. Create a stored procedure to drop a course for a student.

```
CREATE PROCEDURE DropCourse
@StudentID INT,
@CourseID INT
AS
BEGIN
DELETE FROM Enrollment
WHERE StudentID = @StudentID AND CourseID = @CourseID;
END;
GO
```