SQL Queries

Q1. Create a table ORDERS and PRODUCTS with the following fields

ORDERS: {ORDER_ID, CUSTOMER_ID, PRODUCT_ID, QUANTITY, ORDER_DATE}
PRODUCTS: {PRODUCT_ID, PRODUCT_NAME, PRICE}

Perform the following queries in SQL:

1. Insert at least 10 records into the ORDERS and PRODUCTS tables. (2 marks)
2. Count the number of orders placed by each customer. (1 mark)
3. Calculate the total quantity of each product sold. (1 mark)
4. Find the top 3 products that generated the highest revenue. (2 marks)
5. Display the details of orders placed on or after '2024-05-05' sorted by ORDER_DATE in descending order. (1 mark)
6. Add a data constraint to ensure that the QUANTITY in the ORDERS table is always greater than zero. (1 mark)
7. Calculate the total revenue generated by each customer. (2 marks)

Solution 1:

```
 Creating the PRODUCTS table
CREATE TABLE PRODUCTS (
   PRODUCT_ID INT PRIMARY KEY,
   PRODUCT_NAME VARCHAR(100),
   PRICE DECIMAL(10,2)
);

– Creating the ORDERS table
CREATE TABLE ORDERS (
   ORDER_ID INT PRIMARY KEY,
   CUSTOMER_ID INT,
   PRODUCT_ID INT,
   QUANTITY INT CHECK (QUANTITY > 0), – Constraint to ensure quantity is always
greater than zero
   ORDER_DATE DATE,
   FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCTS(PRODUCT_ID)
);

– Inserting at least 10 records into PRODUCTS
INSERT INTO PRODUCTS (PRODUCT_ID, PRODUCT_NAME, PRICE) VALUES
(1, 'Laptop', 800.00),
(2, 'Smartphone', 500.00),
(3, 'Tablet', 300.00),
(4, 'Smartwatch', 200.00),
(5, 'Headphones', 100.00),
(6, 'Keyboard', 50.00),
(7, 'Mouse', 30.00),
(8, 'Monitor', 250.00),
(9, 'Printer', 150.00),
(10, 'Webcam', 80.00);

– Inserting at least 10 records into ORDERS
INSERT INTO ORDERS (ORDER_ID, CUSTOMER_ID, PRODUCT_ID, QUANTITY,
ORDER_DATE) VALUES
(1, 101, 1, 2, '2024-05-01'),
(2, 102, 3, 1, '2024-05-02'),
(3, 103, 2, 3, '2024-05-03'),
(4, 101, 5, 4, '2024-05-04'),
(5, 102, 6, 2, '2024-05-06'),
(6, 104, 4, 1, '2024-05-07'),
(7, 105, 7, 3, '2024-05-08'),
(8, 101, 8, 1, '2024-05-09'),
(9, 103, 9, 2, '2024-05-10'),
(10, 106, 10, 1, '2024-05-11');

– 2. Count the number of orders placed by each customer
SELECT CUSTOMER_ID, COUNT(*) AS TOTAL_ORDERS
FROM ORDERS
GROUP BY CUSTOMER_ID;

– 3. Calculate the total quantity of each product sold
SELECT PRODUCT_ID, SUM(QUANTITY) AS TOTAL_QUANTITY_SOLD
FROM ORDERS
GROUP BY PRODUCT_ID;

– 4. Find the top 3 products that generated the highest revenue
SELECT P.PRODUCT_ID, P.PRODUCT_NAME, SUM(O.QUANTITY * P.PRICE) AS
TOTAL_REVENUE
FROM ORDERS O
JOIN PRODUCTS P ON O.PRODUCT_ID = P.PRODUCT_ID
GROUP BY P.PRODUCT_ID, P.PRODUCT_NAME
ORDER BY TOTAL_REVENUE DESC
LIMIT 3;

– 5. Display the details of orders placed on or after '2024-05-05' sorted by
ORDER_DATE in descending order
SELECT * FROM ORDERS
WHERE ORDER_DATE >= '2024-05-05'
ORDER BY ORDER_DATE DESC;

– 6. Add a data constraint to ensure that the QUANTITY in the ORDERS table is
always greater than zero (Already included in table creation)

– 7. Calculate the total revenue generated by each customer
```

```
SELECT O.CUSTOMER_ID, SUM(O.QUANTITY * P.PRICE) AS TOTAL_REVENUE
FROM ORDERS O
JOIN PRODUCTS P ON O.PRODUCT_ID = P.PRODUCT_ID
GROUP BY O.CUSTOMER_ID;
```

Q2. Create a table STUDENT and EXAM with the following fields:

STUDENT TABLE: {STUDENT_ID, Name, Gender, Age, Class (‘10th Grade’, ‘11th Grade’, ‘12th Grade’)}
EXAM: {STUDENT_ID, MATH_SCORE, SCIENCE_SCORE, ENGLISH_SCORE}

Perform the following queries in SQL:
i. Insert at least 10 records in the tables. (2)
ii. Display the details of male students who are in ‘12th Grade’. (1)
iii. Display the details of the student who secured the highest total score. (2)
iv. Add a new Column HISTORY_SCORE in EXAM table and modify the table by inserting values to HISTORY_SCORE for the records. (1)
v. List Top 3 students of ‘11th Grade’ based on total score. (1)
vi. Display the Average Age of students in ‘10th Grade’. (1)
vii. Display the list of students who scored more than the average marks in MATH_SCORE. (2)

Solution 2:

```
- Creating the STUDENT table
CREATE TABLE STUDENT (
   STUDENT_ID INT PRIMARY KEY,
   NAME VARCHAR(100),
   GENDER VARCHAR(10),
   AGE INT,
   CLASS VARCHAR(20)
);

– Creating the EXAM table
CREATE TABLE EXAM (
   STUDENT_ID INT,
   MATH_SCORE INT,
   SCIENCE_SCORE INT,
   ENGLISH_SCORE INT,
   HISTORY_SCORE INT, – Newly added column
   FOREIGN KEY (STUDENT_ID) REFERENCES STUDENT(STUDENT_ID)
);

– Inserting at least 10 records into STUDENT
INSERT INTO STUDENT (STUDENT_ID, NAME, GENDER, AGE, CLASS) VALUES
(1, 'Sam', 'Female', 15, '10th Grade'),
(2, 'Ram', 'Male', 16, '11th Grade'),
(3, 'Charlie', 'Male', 17, '12th Grade'),
(4, 'David', 'Male', 16, '11th Grade'),
(5, 'Emma', 'Female', 17, '12th Grade'),
(6, 'Frank', 'Male', 15, '10th Grade'),
(7, 'Sia', 'Female', 16, '11th Grade'),
(8, 'Shyam', 'Male', 17, '12th Grade'),
(9, 'Jenny', 'Female', 15, '10th Grade'),
(10, 'Jack', 'Male', 16, '11th Grade');

– Inserting at least 10 records into EXAM
INSERT INTO EXAM (STUDENT_ID, MATH_SCORE, SCIENCE_SCORE,
ENGLISH_SCORE, HISTORY_SCORE) VALUES
(1, 85, 78, 90, 88),
(2, 88, 80, 85, 82),
(3, 92, 85, 88, 90),
(4, 80, 75, 78, 79),
(5, 89, 84, 86, 85),
(6, 76, 70, 72, 74),
(7, 91, 88, 90, 92),
(8, 85, 80, 78, 83),
(9, 79, 72, 75, 77),
(10, 87, 83, 86, 85);

– ii. Display the details of male students who are in ‘12th Grade’
SELECT * FROM STUDENT WHERE GENDER = 'Male' AND CLASS = '12th Grade';

– iii. Display the details of the student who secured the highest total score
SELECT S.*, (E.MATH_SCORE + E.SCIENCE_SCORE + E.ENGLISH_SCORE +
E.HISTORY_SCORE) AS TOTAL_SCORE
FROM STUDENT S
JOIN EXAM E ON S.STUDENT_ID = E.STUDENT_ID
ORDER BY TOTAL_SCORE DESC
LIMIT 1;

– v. List Top 3 students of ‘11th Grade’ based on total score
SELECT S.*, (E.MATH_SCORE + E.SCIENCE_SCORE + E.ENGLISH_SCORE +
E.HISTORY_SCORE) AS TOTAL_SCORE
FROM STUDENT S
JOIN EXAM E ON S.STUDENT_ID = E.STUDENT_ID
WHERE CLASS = '11th Grade'
ORDER BY TOTAL_SCORE DESC
LIMIT 3;

– vi. Display the Average Age of students in ‘10th Grade’
SELECT AVG(AGE) AS AVERAGE_AGE FROM STUDENT WHERE CLASS = '10th
Grade';

– vii. Display the list of students who scored more than the average marks in
```

```
MATH_SCORE
SELECT S.* FROM STUDENT S
JOIN EXAM E ON S.STUDENT_ID = E.STUDENT_ID
WHERE E.MATH_SCORE > (SELECT AVG(MATH_SCORE) FROM EXAM);
```

Q3. Create a table named SALES and ITEMS with the following fields:

SALES: {SALE_ID, CUSTOMER_ID, ITEM_ID, UNITS_SOLD, SALE_DATE}
ITEMS: {ITEM_ID, ITEM_NAME, UNIT_PRICE}

Perform the following queries in SQL:

i. Insert at least 10 records into the SALES and ITEMS tables. (2 marks)
ii. Count the number of sales made by each customer. (1 mark)
iii. Calculate the total units sold for each item. (1 mark)
iv. Find the top 3 items that generated the highest revenue. (2 marks)
v. Display the details of sales made on or after '2024-06-01' sorted by SALE_DATE in descending order. (1 mark)
vi. Add a data constraint to ensure that the UNITS_SOLD in the SALES table is always greater than zero. (1 mark)
vii. Calculate the total revenue generated by each customer. (2 marks)

Solution 3:

```
Creating the ITEMS table
CREATE TABLE ITEMS (
   ITEM_ID INT PRIMARY KEY,
   ITEM_NAME VARCHAR(100),
   UNIT_PRICE DECIMAL(10,2)
);

– Creating the SALES table
CREATE TABLE SALES (
   SALE_ID INT PRIMARY KEY,
   CUSTOMER_ID INT,
   ITEM_ID INT,
   UNITS_SOLD INT CHECK (UNITS_SOLD > 0), – Constraint to ensure units sold is
always greater than zero
   SALE_DATE DATE,
   FOREIGN KEY (ITEM_ID) REFERENCES ITEMS(ITEM_ID)
);

– Inserting at least 10 records into ITEMS
INSERT INTO ITEMS (ITEM_ID, ITEM_NAME, UNIT_PRICE) VALUES
(1, 'Television', 900.00),
(2, 'Smartphone', 500.00),
(3, 'Tablet', 300.00),
(4, 'Smartwatch', 200.00),
(5, 'Headphones', 100.00),
(6, 'Keyboard', 50.00),
(7, 'Mouse', 30.00),
(8, 'Monitor', 250.00),
(9, 'Printer', 150.00),
(10, 'Webcam', 80.00);

– Inserting at least 10 records into SALES
INSERT INTO SALES (SALE_ID, CUSTOMER_ID, ITEM_ID, UNITS_SOLD, SALE_DATE)
VALUES
(1, 101, 1, 2, '2024-06-01'),
(2, 102, 3, 1, '2024-06-02'),
(3, 103, 2, 3, '2024-06-03'),
(4, 101, 5, 4, '2024-06-04'),
(5, 102, 6, 2, '2024-06-06'),
(6, 104, 4, 1, '2024-06-07'),
(7, 105, 7, 3, '2024-06-08'),
(8, 101, 8, 1, '2024-06-09'),
(9, 103, 9, 2, '2024-06-10'),
(10, 106, 10, 1, '2024-06-11');

– ii. Count the number of sales made by each customer
SELECT CUSTOMER_ID, COUNT(*) AS TOTAL_SALES
FROM SALES
GROUP BY CUSTOMER_ID;

– iii. Calculate the total units sold for each item
SELECT ITEM_ID, SUM(UNITS_SOLD) AS TOTAL_UNITS_SOLD
FROM SALES
GROUP BY ITEM_ID;

– iv. Find the top 3 items that generated the highest revenue
SELECT I.ITEM_ID, I.ITEM_NAME, SUM(S.UNITS_SOLD * I.UNIT_PRICE) AS
TOTAL_REVENUE
FROM SALES S
JOIN ITEMS I ON S.ITEM_ID = I.ITEM_ID
GROUP BY I.ITEM_ID, I.ITEM_NAME
ORDER BY TOTAL_REVENUE DESC
LIMIT 3;

– v. Display the details of sales made on or after '2024-06-01' sorted by SALE_DATE
in descending order
SELECT * FROM SALES
WHERE SALE_DATE >= '2024-06-01'
ORDER BY SALE_DATE DESC;
```

– vi. Add a data constraint to ensure that the UNITS_SOLD in the SALES table is always greater than zero (Already included in table creation)

– vii. Calculate the total revenue generated by each customer
```
SELECT S.CUSTOMER_ID, SUM(S.UNITS_SOLD * I.UNIT_PRICE) AS TOTAL_REVENUE
FROM SALES S
JOIN ITEMS I ON S.ITEM_ID = I.ITEM_ID
GROUP BY S.CUSTOMER_ID;
```

Q4. Create a table EMPLOYEES and DEPARTMENTS with the following fields:

EMPLOYEES: {EMPLOYEE_ID, NAME, DEPARTMENT_ID, SALARY, JOIN_DATE}
DEPARTMENTS: {DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID}

Perform the following queries in SQL:

i. Insert at least 5 records into the EMPLOYEES and DEPARTMENTS tables. (1 mark)
ii. Display the names of all employees. (1 mark)
iii. Display the department names and their corresponding manager IDs. (1 mark)
iv. Find the employee with the highest salary. (1 mark)
v. Display the details of employees who joined in the year 2024. (1 mark)
vi. Ensure that the SALARY column in the EMPLOYEES table does not accept negative values. (1 mark)
vii. Calculate the total number of employees. (1 mark)
viii. List the names of employees along with their department names. (2 marks)
ix. Display the details of departments that have no employees. (1 mark)
x. Update the salary of an employee with EMPLOYEE_ID 1 to 60000. (1 mark)

Solution 4 :

Creating the ITEMS table
```
CREATE TABLE ITEMS (
   ITEM_ID INT PRIMARY KEY,
   ITEM_NAME VARCHAR(100),
   UNIT_PRICE DECIMAL(10,2)
);
```

– Creating the SALES table
```
CREATE TABLE SALES (
   SALE_ID INT PRIMARY KEY,
   CUSTOMER_ID INT,
   ITEM_ID INT,
   UNITS_SOLD INT CHECK (UNITS_SOLD > 0), – Constraint to ensure units sold is
always greater than zero
   SALE_DATE DATE,
    FOREIGN KEY (ITEM_ID) REFERENCES ITEMS(ITEM_ID)
);

– Inserting at least 10 records into ITEMS
INSERT INTO ITEMS (ITEM_ID, ITEM_NAME, UNIT_PRICE) VALUES
(1, 'Refrigerator', 900.00),
(2, 'Smartphone', 500.00),
(3, 'Tablet', 300.00),
(4, 'Smartwatch', 200.00),
(5, 'Headphones', 100.00),
(6, 'Keyboard', 50.00),
(7, 'Mouse', 30.00),
(8, 'Monitor', 250.00),
(9, 'Printer', 150.00),
(10, 'Webcam', 80.00);

– Inserting at least 10 records into SALES
INSERT INTO SALES (SALE_ID, CUSTOMER_ID, ITEM_ID, UNITS_SOLD, SALE_DATE)
VALUES
(1, 101, 1, 2, '2024-06-01'),
(2, 102, 3, 1, '2024-06-02'),
(3, 103, 2, 3, '2024-06-03'),
(4, 101, 5, 4, '2024-06-04'),
(5, 102, 6, 2, '2024-06-06'),
(6, 104, 4, 1, '2024-06-07'),
(7, 105, 7, 3, '2024-06-08'),
(8, 101, 8, 1, '2024-06-09'),
(9, 103, 9, 2, '2024-06-10'),
(10, 106, 10, 1, '2024-06-11');

– ii. Count the number of sales made by each customer
SELECT CUSTOMER_ID, COUNT(*) AS TOTAL_SALES
FROM SALES
GROUP BY CUSTOMER_ID;

– iii. Calculate the total units sold for each item
SELECT ITEM_ID, SUM(UNITS_SOLD) AS TOTAL_UNITS_SOLD
FROM SALES
GROUP BY ITEM_ID;

– iv. Find the top 3 items that generated the highest revenue
SELECT I.ITEM_ID, I.ITEM_NAME, SUM(S.UNITS_SOLD * I.UNIT_PRICE) AS
TOTAL_REVENUE
FROM SALES S
JOIN ITEMS I ON S.ITEM_ID = I.ITEM_ID
GROUP BY I.ITEM_ID, I.ITEM_NAME
ORDER BY TOTAL_REVENUE DESC
LIMIT 3;
```

– v. Display the details of sales made on or after '2024-06-01' sorted by SALE_DATE in descending order

```
SELECT * FROM SALES
WHERE SALE_DATE >= '2024-06-01'
ORDER BY SALE_DATE DESC;
```

– vi. Add a data constraint to ensure that the UNITS_SOLD in the SALES table is always greater than zero (Already included in table creation)

– vii. Calculate the total revenue generated by each customer

```
SELECT S.CUSTOMER_ID, SUM(S.UNITS_SOLD * I.UNIT_PRICE) AS TOTAL_REVENUE
FROM SALES S
JOIN ITEMS I ON S.ITEM_ID = I.ITEM_ID
GROUP BY S.CUSTOMER_ID;
```

Q5. Create a table EMPLOYEES and DEPARTMENTS with the following fields:

EMPLOYEES: {EMPLOYEE_ID, NAME, DEPARTMENT_ID, SALARY, JOIN_DATE}
DEPARTMENTS: {DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID}

Perform the following queries in SQL:

i. Insert at least 10 records into the EMPLOYEES and DEPARTMENTS tables. (2 marks)
ii. Count the number of employees in each department. (1 mark)
iii. Calculate the total salary paid to employees in each department. (1 mark)
iv. Find the top 3 departments with the highest total salary expenditure. (2 marks)
v. Display the details of employees who joined on or after '2024-01-01' sorted by JOIN_DATE in descending order. (1 mark)
vi. Add a data constraint to ensure that the SALARY in the EMPLOYEES table is always greater than zero. (1 mark)
vii. Calculate the average salary of employees managed by each manager. (2 marks)

Solution 5:

 Creating the ITEMS table

```
CREATE TABLE ITEMS (
   ITEM_ID INT PRIMARY KEY,
   ITEM_NAME VARCHAR(100),
   UNIT_PRICE DECIMAL(10,2)
);
```

– Creating the SALES table

```
CREATE TABLE SALES (
   SALE_ID INT PRIMARY KEY,
   CUSTOMER_ID INT,
   ITEM_ID INT,
   UNITS_SOLD INT CHECK (UNITS_SOLD > 0), – Constraint to ensure units sold is
always greater than zero
   SALE_DATE DATE,
   FOREIGN KEY (ITEM_ID) REFERENCES ITEMS(ITEM_ID)
);

– Inserting at least 10 records into ITEMS
INSERT INTO ITEMS (ITEM_ID, ITEM_NAME, UNIT_PRICE) VALUES
(1, 'Refrigerator', 900.00),
(2, 'Smartphone', 500.00),
(3, 'Tablet', 300.00),
(4, 'Smartwatch', 200.00),
(5, 'Headphones', 100.00),
(6, 'Keyboard', 50.00),
(7, 'Mouse', 30.00),
(8, 'Monitor', 250.00),
(9, 'Printer', 150.00),
(10, 'Webcam', 80.00);

– Inserting at least 10 records into SALES
INSERT INTO SALES (SALE_ID, CUSTOMER_ID, ITEM_ID, UNITS_SOLD, SALE_DATE)
VALUES
(1, 101, 1, 2, '2024-06-01'),
(2, 102, 3, 1, '2024-06-02'),
(3, 103, 2, 3, '2024-06-03'),
(4, 101, 5, 4, '2024-06-04'),
(5, 102, 6, 2, '2024-06-06'),
(6, 104, 4, 1, '2024-06-07'),
(7, 105, 7, 3, '2024-06-08'),
(8, 101, 8, 1, '2024-06-09'),
(9, 103, 9, 2, '2024-06-10'),
(10, 106, 10, 1, '2024-06-11');

– ii. Count the number of sales made by each customer
SELECT CUSTOMER_ID, COUNT(*) AS TOTAL_SALES
FROM SALES
GROUP BY CUSTOMER_ID;

– iii. Calculate the total units sold for each item
SELECT ITEM_ID, SUM(UNITS_SOLD) AS TOTAL_UNITS_SOLD
FROM SALES
GROUP BY ITEM_ID;

– iv. Find the top 3 items that generated the highest revenue
SELECT I.ITEM_ID, I.ITEM_NAME, SUM(S.UNITS_SOLD * I.UNIT_PRICE) AS
TOTAL_REVENUE
```

```
FROM SALES S
JOIN ITEMS I ON S.ITEM_ID = I.ITEM_ID
GROUP BY I.ITEM_ID, I.ITEM_NAME
ORDER BY TOTAL_REVENUE DESC
LIMIT 3;

– v. Display the details of sales made on or after '2024-06-01' sorted by SALE_DATE
in descending order
SELECT * FROM SALES
WHERE SALE_DATE >= '2024-06-01'
ORDER BY SALE_DATE DESC;

– vi. Add a data constraint to ensure that the UNITS_SOLD in the SALES table is
always greater than zero (Already included in table creation)

– vii. Calculate the total revenue generated by each customer
SELECT S.CUSTOMER_ID, SUM(S.UNITS_SOLD * I.UNIT_PRICE) AS TOTAL_REVENUE
FROM SALES S
JOIN ITEMS I ON S.ITEM_ID = I.ITEM_ID
GROUP BY S.CUSTOMER_ID;

– Creating the DEPARTMENTS table
CREATE TABLE DEPARTMENTS (
   DEPARTMENT_ID INT PRIMARY KEY,
   DEPARTMENT_NAME VARCHAR(100),
   MANAGER_ID INT
);
```

Q6. Create a table PATIENTS and APPOINTMENTS with the following fields:

PATIENTS: {PATIENT_ID, NAME, AGE, GENDER}
APPOINTMENTS: {APPOINTMENT_ID, PATIENT_ID, DOCTOR, APPOINTMENT_DATE, FEES}

Perform the following queries in SQL:

i. Insert at least 5 records into the PATIENTS and APPOINTMENTS tables. (1 mark)
ii. Display the names of all patients. (1 mark)
iii. Display the details of appointments for a particular doctor (e.g., 'Dr. Smith'). (1 mark)
iv. Find the patient with the highest number of appointments. (1 mark)
v. Display the details of appointments that occurred in the last month. (1 mark)
vi. Ensure that the FEES column in the APPOINTMENTS table does not accept negative values. (1 mark)
vii. Calculate the total number of appointments. (1 mark)
viii. List the names of patients along with their appointment details. (2 marks)

ix. Display the details of patients who have not had any appointments. (1 mark)
x. Update the appointment date for a specific appointment (e.g., APPOINTMENT_ID 1) to a new date. (1 mark)

Solution 6:

```
–Creating the PATIENTS table
CREATE TABLE PATIENTS (
   PATIENT_ID INT PRIMARY KEY,
   NAME VARCHAR(100),
   AGE INT,
   GENDER VARCHAR(10)
);

– Creating the APPOINTMENTS table
CREATE TABLE APPOINTMENTS (
   APPOINTMENT_ID INT PRIMARY KEY,
   PATIENT_ID INT,
   DOCTOR VARCHAR(100),
   APPOINTMENT_DATE DATE,
   FEES DECIMAL(10,2) CHECK (FEES >= 0), – Constraint to ensure fees are not
negative
   FOREIGN KEY (PATIENT_ID) REFERENCES PATIENTS(PATIENT_ID)
);

– i. Inserting at least 5 records into PATIENTS and APPOINTMENTS
INSERT INTO PATIENTS (PATIENT_ID, NAME, AGE, GENDER) VALUES
(1, 'Alice Johnson', 30, 'Female'),
(2, 'Bob Smith', 45, 'Male'),
(3, 'Charlie Brown', 29, 'Male'),
(4, 'Diana Green', 50, 'Female'),
(5, 'Ethan White', 35, 'Male');

INSERT INTO APPOINTMENTS (APPOINTMENT_ID, PATIENT_ID, DOCTOR,
APPOINTMENT_DATE, FEES) VALUES
(1, 1, 'Dr. Smith', '2024-05-10', 200.00),
(2, 2, 'Dr. Lee', '2024-05-15', 150.00),
(3, 3, 'Dr. Smith', '2024-05-20', 250.00),
(4, 4, 'Dr. Adams', '2024-05-22', 180.00),
(5, 5, 'Dr. Lee', '2024-05-25', 220.00);

– ii. Display the names of all patients
SELECT NAME FROM PATIENTS;

– iii. Display the details of appointments for a particular doctor (e.g., 'Dr. Smith')
SELECT * FROM APPOINTMENTS WHERE DOCTOR = 'Dr. Smith';

– iv. Find the patient with the highest number of appointments
SELECT P.PATIENT_ID, P.NAME, COUNT(A.APPOINTMENT_ID) AS
```

```
TOTAL_APPOINTMENTS
FROM PATIENTS P
JOIN APPOINTMENTS A ON P.PATIENT_ID = A.PATIENT_ID
GROUP BY P.PATIENT_ID, P.NAME
ORDER BY TOTAL_APPOINTMENTS DESC
LIMIT 1;

– v. Display the details of appointments that occurred in the last month
SELECT * FROM APPOINTMENTS
WHERE APPOINTMENT_DATE >= DATE_SUB(CURDATE(), INTERVAL 1 MONTH);

– vi. Ensure that the FEES column in the APPOINTMENTS table does not accept
negative values (Already included in table creation)

– vii. Calculate the total number of appointments
SELECT COUNT(*) AS TOTAL_APPOINTMENTS FROM APPOINTMENTS;

– viii. List the names of patients along with their appointment details
SELECT P.NAME, A.*
FROM PATIENTS P
LEFT JOIN APPOINTMENTS A ON P.PATIENT_ID = A.PATIENT_ID;

– ix. Display the details of patients who have not had any appointments
SELECT * FROM PATIENTS
WHERE PATIENT_ID NOT IN (SELECT DISTINCT PATIENT_ID FROM
APPOINTMENTS);

– x. Update the appointment date for a specific appointment (e.g., APPOINTMENT_ID
1) to a new date
UPDATE APPOINTMENTS
SET APPOINTMENT_DATE = '2024-06-01'
WHERE APPOINTMENT_ID = 1;
```

Q7. Create a table EMPLOYEES and SALARIES with the following fields:

EMPLOYEES TABLE: {EMPLOYEE_ID, NAME, DEPARTMENT, GENDER, AGE}
SALARIES TABLE: {EMPLOYEE_ID, BASE_SALARY, BONUS, DEDUCTIONS}

Perform the following queries in SQL:

i. Insert at least 10 records in the tables. (2 marks)

ii. Display the details of employees in the 'Finance' department. (1 mark)

iii. Display the details of the employee with the highest net salary. (2 marks)

iv. Add a new column ALLOWANCES in the SALARIES table and modify the table by inserting values to ALLOWANCES for the records. (1 mark)

v. List the top 3 employees with the highest total compensation (BASE_SALARY + BONUS + ALLOWANCES - DEDUCTIONS). (1 mark)

vi. Display the average age of employees in the 'HR' department. (1 mark)

i. Display the list of employees who have a net salary (BASE_SALARY + BONUS - DEDUCTIONS) greater than the average net salary of all employees. (2 marks)

Solution 7:

```
–– Creating the EMPLOYEES table
CREATE TABLE EMPLOYEES (
   EMPLOYEE_ID INT PRIMARY KEY,
   NAME VARCHAR(100),
   DEPARTMENT VARCHAR(50),
   GENDER VARCHAR(10),
   AGE INT
);

– Creating the SALARIES table
CREATE TABLE SALARIES (
   EMPLOYEE_ID INT PRIMARY KEY,
   BASE_SALARY DECIMAL(10,2),
   BONUS DECIMAL(10,2),
   DEDUCTIONS DECIMAL(10,2),
   ALLOWANCES DECIMAL(10,2),
   FOREIGN KEY (EMPLOYEE_ID) REFERENCES EMPLOYEES(EMPLOYEE_ID)
);

– i. Inserting at least 10 records in EMPLOYEES and SALARIES
INSERT INTO EMPLOYEES (EMPLOYEE_ID, NAME, DEPARTMENT, GENDER, AGE)
VALUES
(1, 'John Doe', 'Finance', 'Male', 35),
(2, 'Jane Smith', 'HR', 'Female', 40),
(3, 'Robert Brown', 'IT', 'Male', 28),
(4, 'Emily Davis', 'Finance', 'Female', 30),
(5, 'Michael Johnson', 'HR', 'Male', 45),
(6, 'Sarah Wilson', 'IT', 'Female', 32),
(7, 'David Lee', 'Marketing', 'Male', 38),
(8, 'Laura Adams', 'Finance', 'Female', 29),
(9, 'James White', 'IT', 'Male', 41),
(10, 'Sophia Green', 'Marketing', 'Female', 36);

INSERT INTO SALARIES (EMPLOYEE_ID, BASE_SALARY, BONUS, DEDUCTIONS,
ALLOWANCES) VALUES
(1, 60000, 5000, 2000, 3000),
(2, 55000, 4000, 1500, 2500),
(3, 70000, 6000, 2500, 4000),
(4, 62000, 5200, 2200, 3100),
(5, 58000, 4500, 1800, 2700),
(6, 73000, 7000, 2800, 4200),
(7, 56000, 4300, 1600, 2600),
(8, 61000, 5100, 2100, 3200),
(9, 75000, 7500, 3000, 4500),
(10, 57000, 4400, 1700, 2800);

– ii. Display the details of employees in the 'Finance' department
SELECT * FROM EMPLOYEES WHERE DEPARTMENT = 'Finance';

– iii. Display the details of the employee with the highest net salary
SELECT E.*, S.*
FROM EMPLOYEES E
JOIN SALARIES S ON E.EMPLOYEE_ID = S.EMPLOYEE_ID
ORDER BY (S.BASE_SALARY + S.BONUS - S.DEDUCTIONS) DESC
LIMIT 1;

– iv. List the top 3 employees with the highest total compensation
SELECT E.*, (S.BASE_SALARY + S.BONUS + S.ALLOWANCES - S.DEDUCTIONS) AS
TOTAL_COMPENSATION
FROM EMPLOYEES E
JOIN SALARIES S ON E.EMPLOYEE_ID = S.EMPLOYEE_ID
ORDER BY TOTAL_COMPENSATION DESC
LIMIT 3;

– v. Display the average age of employees in the 'HR' department
SELECT AVG(AGE) AS AVERAGE_AGE FROM EMPLOYEES WHERE DEPARTMENT =
'HR';

– vi. Display employees whose net salary is greater than the average net salary
SELECT E.*, S.*
FROM EMPLOYEES E
JOIN SALARIES S ON E.EMPLOYEE_ID = S.EMPLOYEE_ID
WHERE (S.BASE_SALARY + S.BONUS - S.DEDUCTIONS) >
   (SELECT AVG(BASE_SALARY + BONUS - DEDUCTIONS) FROM SALARIES);
```

Q8. Create a table PROFESSOR and PUBLICATION with the following fields:

PROFESSOR TABLE: {PROFESSOR_ID, NAME, DEPARTMENT, AGE, SALARY}
PUBLICATION TABLE: {PUBLICATION_ID, PROFESSOR_ID, TITLE, JOURNAL, YEAR}

Perform the following queries in SQL:

i. Insert at least 10 records into the PROFESSOR and PUBLICATION tables. (2 marks)

ii. Display the details of professors from the 'OPERATIONS RESEARCH' department. (1 mark)

iii. Display the details of the professor with the highest number of publications. (2 marks)

iv. Add a new column EMAIL in the PROFESSOR table and update the table by inserting values for EMAIL for the records. (1 mark)

v. List the top 3 journals with the highest number of publications. (1 mark)

vi. Display the average salary of professors in the 'MATHEMATICS' department. (1 mark)

i. Display the list of professors who have published in more than 2 journals. (2 marks)

9. Create a table CUSTOMER and TRANSACTION with the following fields:

CUSTOMER TABLE: {CUSTOMER_ID, NAME, AGE, CITY, PHONE_NUMBER}
TRANSACTION TABLE: {TRANSACTION_ID, CUSTOMER_ID, AMOUNT, TRANSACTION_DATE, TRANSACTION_TYPE}

Perform the following queries in SQL:

i. Insert at least 10 records into the CUSTOMER and TRANSACTION tables. (2 marks)

ii. Display the details of customers living in 'New York'. (1 mark)

iii. Display the details of the customer who made the highest total transaction amount. (2 marks)

iv. Add a new column EMAIL in the CUSTOMER table and update the table by inserting values for EMAIL for the records. (1 mark)

v. List the top 3 customers based on the number of transactions they have made. (1 mark)

vi. Display the average transaction amount of 'Deposit' transactions. (1 mark)

vii. Display the list of customers who have made transactions totalling more than $5000. (2 marks)

10. Create a table BOOK and BORROW with the following fields:

BOOK TABLE: {BOOK_ID, TITLE, AUTHOR, GENRE, PUBLISHED_YEAR}
BORROW TABLE: {BORROW_ID, BOOK_ID, MEMBER_ID, BORROW_DATE, RETURN_DATE}

Perform the following queries in SQL:

i. Insert at least 10 records into the BOOK and BORROW tables. (2 marks)

ii. Display the details of books authored by 'J.K. Rowling'. (1 mark)

iii. Display the details of the book that has been borrowed the most times. (2 marks)

iv. Add a new column ISBN in the BOOK table and update the table by inserting values for ISBN for the records. (1 mark)

v. List the top 3 members based on the number of books they have borrowed. (1 mark)

vi. Display the average number of days books are borrowed for. (1 mark)

vii. Display the list of books borrowed by members who have borrowed more than 5 books. (2 marks

Solution 8:

```
— Creating the PROFESSOR table
CREATE TABLE PROFESSOR (
   PROFESSOR_ID INT PRIMARY KEY,
   NAME VARCHAR(100),
   DEPARTMENT VARCHAR(100),
   AGE INT,
   SALARY DECIMAL(10,2),
   EMAIL VARCHAR(100)
);

– Creating the PUBLICATION table
CREATE TABLE PUBLICATION (
   PUBLICATION_ID INT PRIMARY KEY,
   PROFESSOR_ID INT,
   TITLE VARCHAR(255),
   JOURNAL VARCHAR(100),
    YEAR INT,
    FOREIGN KEY (PROFESSOR_ID) REFERENCES PROFESSOR(PROFESSOR_ID)
);

– i. Inserting at least 10 records into PROFESSOR and PUBLICATION
INSERT INTO PROFESSOR (PROFESSOR_ID, NAME, DEPARTMENT, AGE, SALARY,
EMAIL) VALUES
(1, 'Dr. Alice Brown', 'Mathematics', 45, 90000, 'alice.brown@university.edu'),
(2, 'Dr. Bob Smith', 'Physics', 50, 95000, 'bob.smith@university.edu'),
(3, 'Dr. Charlie Johnson', 'Operations Research', 40, 88000,
'charlie.johnson@university.edu'),
(4, 'Dr. Diana Green', 'Mathematics', 55, 98000, 'diana.green@university.edu'),
(5, 'Dr. Ethan White', 'Computer Science', 42, 87000, 'ethan.white@university.edu'),
(6, 'Dr. Fiona Adams', 'Operations Research', 47, 93000, 'fiona.adams@university.edu'),
(7, 'Dr. George Wilson', 'Mathematics', 60, 102000, 'george.wilson@university.edu'),
(8, 'Dr. Hannah Lee', 'Physics', 53, 91000, 'hannah.lee@university.edu'),
(9, 'Dr. Ian Brown', 'Computer Science', 39, 85000, 'ian.brown@university.edu'),
(10, 'Dr. Julia Scott', 'Operations Research', 44, 92000, 'julia.scott@university.edu');

INSERT INTO PUBLICATION (PUBLICATION_ID, PROFESSOR_ID, TITLE, JOURNAL,
YEAR) VALUES
(1, 1, 'Mathematical Theories in AI', 'Journal of Mathematics', 2023),
(2, 2, 'Quantum Mechanics and Computation', 'Physics World', 2022),
(3, 3, 'Optimization Techniques', 'OR Journal', 2024),
(4, 4, 'Statistical Models', 'Journal of Mathematics', 2021),
(5, 5, 'AI in Computer Vision', 'CS Journal', 2023),
(6, 6, 'Operations Research in Logistics', 'OR Journal', 2022),
(7, 7, 'Advanced Calculus', 'Mathematical Analysis', 2024),
(8, 8, 'Astrophysics Discoveries', 'Physics World', 2023),
(9, 9, 'Machine Learning Innovations', 'CS Journal', 2024),
(10, 10, 'Game Theory Applications', 'OR Journal', 2021);

– ii. Display the details of professors from the 'OPERATIONS RESEARCH' department
SELECT * FROM PROFESSOR WHERE DEPARTMENT = 'Operations Research';

– iii. Display the details of the professor with the highest number of publications
SELECT P.PROFESSOR_ID, P.NAME, COUNT(PB.PUBLICATION_ID) AS
TOTAL_PUBLICATIONS
FROM PROFESSOR P
JOIN PUBLICATION PB ON P.PROFESSOR_ID = PB.PROFESSOR_ID
GROUP BY P.PROFESSOR_ID, P.NAME
ORDER BY TOTAL_PUBLICATIONS DESC
LIMIT 1;

– iv. List the top 3 journals with the highest number of publications
SELECT JOURNAL, COUNT(*) AS PUBLICATION_COUNT
FROM PUBLICATION
GROUP BY JOURNAL
```

```
ORDER BY PUBLICATION_COUNT DESC
LIMIT 3;

– v. Display the average salary of professors in the 'MATHEMATICS' department
SELECT AVG(SALARY) AS AVERAGE_SALARY FROM PROFESSOR WHERE
DEPARTMENT = 'Mathematics';

– vi. Display the list of professors who have published in more than 2 journals
SELECT P.PROFESSOR_ID, P.NAME, COUNT(DISTINCT PB.JOURNAL) AS
JOURNAL_COUNT
FROM PROFESSOR P
JOIN PUBLICATION PB ON P.PROFESSOR_ID = PB.PROFESSOR_ID
GROUP BY P.PROFESSOR_ID, P.NAME
HAVING JOURNAL_COUNT > 2;
```