MYSQL

1. Create a db called company consist of the following tables.

1.Emp (eno,ename, job,hiredate,salary,commission,deptno)

2.dept(deptno,deptname,location)

eno is primary key in emp

deptno is primary key in dept

Solve Queries by SQL

1. List the maximum salary paid to salesman

2. List name of emp whose name start with 'I'

3. List details of emp who have joined before '30-sept-81'

4. List the emp details in the descending order of their basic salary

5. List of no. of emp & avg salary for emp in the dept no '20'

6. List the avg salary, minimum salary of the emp hiredatewise for dept no '10'.

7. List emp name and its department

8. List total salary paid to each department

9. List details of employee working in 'Dev' department

10. Update salary of all employees in deptno 10 by 5 %.

1. Maximum salary paid to salesman

```
SELECT MAX(salary) AS Max_Salary
FROM Emp
WHERE job = 'Salesman';
```

2. Employees whose name starts with 'I'

```
SELECT * FROM Emp
WHERE ename LIKE 'I%';
```

3. Employees joined before '30-SEP-81'

```
SELECT * FROM Emp
WHERE hiredate < '1981-09-30';
```

4. Employee details in descending order of salary

```
SELECT * FROM Emp
ORDER BY salary DESC;
```

5. Number of employees and average salary in deptno 20

```
SELECT COUNT(*) AS No_of_Emp, AVG(salary) AS Avg_Salary
FROM Emp
WHERE deptno = 20;
```

6. Average and minimum salary hiredate-wise for deptno 10

```
SELECT hiredate, AVG(salary) AS Avg_Salary, MIN(salary) AS Min_Salary
FROM Emp
WHERE deptno = 10
GROUP BY hiredate;
```

7. Employee name with their department

```
SELECT e.ename, d.deptname
FROM Emp e
JOIN Dept d ON e.deptno = d.deptno;
```

8. Total salary paid to each department

```
SELECT deptno, SUM(salary) AS Total_Salary
FROM Emp
GROUP BY deptno;
```

9. Employee details working in “Dev” department

```
SELECT e.*
FROM Emp e
JOIN Dept d ON e.deptno = d.deptno
WHERE d.deptname = 'Dev';
```

10. Update salary of deptno 10 employees by 5%

```
UPDATE Emp
SET salary = salary * 1.05
WHERE deptno = 10;
```

---

2. Create a database

1. employee (employee name, street, city) ,employee name is primary key

2. works (employee name, company name, salary)

3. company (company name, city) ,company name is primary key

4. manages (employee name, manager name)

Give an expression in SQL for each of the following queries.

1. Find the names of all employees who work for First Bank Corporation.

2. Find all employees who do not work for First Bank Coorporation

3. Find the company that has most employees.

4. Find all companies located in every in which small bank corporation is located

5. Find details of employee having salary greater than 10,000.

6. Update salary of all employees who work for First Bank Corporation by 10%.

7. Find employee and their managers.

8. Find the names, street and cities of all employees who work for First Bank

Corporation and earn more than 10,000.

9. Find those companies whose employees earn a higher salary,on average, than th

average salary at First Bank Corporation

1. **Employees who work for “First Bank Corporation”**

```
SELECT employee_name
FROM works
WHERE company_name = 'First Bank Corporation';
```

2. Employees who do NOT work for “First Bank Corporation”

```
SELECT employee_name
FROM works
WHERE company_name <> 'First Bank Corporation';
```

3. Company with the most employees

```
SELECT company_name, COUNT(*) AS total_employees
FROM works
GROUP BY company_name
ORDER BY total_employees DESC
LIMIT 1;
```

4. Companies in all cities where “Small Bank Corporation” is located

```
SELECT company_name
FROM company
WHERE city IN (SELECT city FROM company WHERE company_name = 'Small Bank Corporation');
```

5. Employees with salary > 10,000

```
SELECT *
FROM works
WHERE salary > 10000;
```

6. Update salary of First Bank Corporation employees by 10%

```
UPDATE works
SET salary = salary * 1.10
WHERE company_name = 'First Bank Corporation';
```

7. Employees and their managers

```
SELECT employee_name, manager_name
FROM manages;
```

8. Employees (name, street, city) who work for “First Bank” and earn > 10,000

```
SELECT e.employee_name, e.street, e.city
FROM employee e
JOIN works w ON e.employee_name = w.employee_name
WHERE w.company_name = 'First Bank Corporation' AND w.salary > 10000;
```

9. Companies whose employees earn more than avg salary at First Bank

```
SELECT company_name
FROM works
GROUP BY company_name
HAVING AVG(salary) > (
  SELECT AVG(salary)
  FROM works
  WHERE company_name = 'First Bank Corporation'
);
```

---

3. The following tables form part of a database held in a relational DBMS:

Hotel (HotelNo, Name, City) HotelNo is the primary key

Room (RoomNo, HotelNo, Type, Price)

Booking (HotelNo, GuestNo, DateFrom, DateTo, RoomNo)

Guest (GuestNo, GuestName, GuestAddress) GuestNo is primary key

Room contains room details for each hotel and (HotelNo, RoomNo) forms the

primary key.Booking contains details of the bookings and the primary key comprises

(HotelNo, GuestNo and DateFrom)

Solve following queries by SQL

1. List full details of all hotels.

2. How many hotels are there?

3. List the price and type of all rooms at the Grosvenor Hotel.

4. List the number of rooms in each hotel.

5. Update the price of all rooms by 5%.

6. List full details of all hotels in London.

7. What is the average price of a room?

8. List all guests currently staying at the Grosvenor Hotel.

9. List the number of rooms in each hotel in London.

10.Create one view on above database and query it.

1. List full details of all hotels

```
SELECT * FROM Hotel;
```

2. Count total hotels

```
SELECT COUNT(*) AS Total_Hotels FROM Hotel;
```

3. Price and type of rooms at “Grosvenor Hotel”

```
SELECT r.Price, r.Type
FROM Room r
JOIN Hotel h ON r.HotelNo = h.HotelNo
WHERE h.Name = 'Grosvenor Hotel';
```

4. Number of rooms in each hotel

```
SELECT HotelNo, COUNT(*) AS No_of_Rooms
FROM Room
GROUP BY HotelNo;
```

5. Update price of all rooms by 5%

```
UPDATE Room
SET Price = Price * 1.05;
```

6. Details of all hotels in London

```
SELECT * FROM Hotel
WHERE City = 'London';
```

7. Average price of a room

```
SELECT AVG(Price) AS Avg_Price
FROM Room;
```

8. Guests staying at “Grosvenor Hotel”

```
SELECT g.GuestName
FROM Guest g
JOIN Booking b ON g.GuestNo = b.GuestNo
JOIN Hotel h ON b.HotelNo = h.HotelNo
WHERE h.Name = 'Grosvenor Hotel';
```

9. Number of rooms in each London hotel

```
SELECT h.Name, COUNT(r.RoomNo) AS No_of_Rooms
FROM Hotel h
JOIN Room r ON h.HotelNo = r.HotelNo
WHERE h.City = 'London'
GROUP BY h.Name;
```

10. Create a view and query it

```
CREATE VIEW Hotel_View AS
SELECT h.Name, h.City, r.Type, r.Price
FROM Hotel h
JOIN Room r ON h.HotelNo = r.HotelNo;

SELECT * FROM Hotel_View;
```

---

4. The following tables form part of a database held in a relational DBMS:

Hotel (HotelNo, Name, City) HotelNo is primary key

Room (RoomNo, HotelNo, Type, Price)

Booking (HotelNo, GuestNo, DateFrom, DateTo, RoomNo)

Guest (GuestNo, GuestName, GuestAddress) GuestNo is primary key

Solve following queries by SQL

1. What is the total revenue per night from all double rooms?

2. List the details of all rooms at the Grosvenor Hotel, including the name of the guest staying in the room, if the room is occupied.

3. What is the average number of bookings for each hotel in April?

4. Create index on one of the field and show is performance in query.

5. List full details of all hotels.

6. List full details of all hotels in London.

7. Update the price of all rooms by 5%.

8. List the number of rooms in each hotel in London.

9. List all double or family rooms with a price below £40.00 per night, in ascending order of price.

1. Total revenue per night from all double rooms

```
SELECT SUM(Price) AS Total_Revenue_Per_Night
FROM Room
WHERE Type = 'Double';
```

2. Details of all rooms at the Grosvenor Hotel (with guest name if occupied)

```
SELECT r.RoomNo, r.Type, r.Price, g.GuestName
FROM Hotel h
JOIN Room r ON h.HotelNo = r.HotelNo
LEFT JOIN Booking b ON r.RoomNo = b.RoomNo AND r.HotelNo = b.HotelNo
LEFT JOIN Guest g ON b.GuestNo = g.GuestNo
```

```
WHERE h.Name = 'Grosvenor Hotel';
```

3. Average number of bookings for each hotel in April

(assuming DateFrom is a DATE column)

```
SELECT h.HotelNo, h.Name, COUNT(b.RoomNo) / 30.0 AS Avg_Bookings_April
FROM Hotel h
JOIN Booking b ON h.HotelNo = b.HotelNo
WHERE MONTH(b.DateFrom) = 4
GROUP BY h.HotelNo, h.Name;
```

4. Create index on one field and show performance

👉 Create index on City column of Hotel for faster searching:

```
CREATE INDEX idx_city ON Hotel(City);
```

Then check performance with:

```
EXPLAIN SELECT * FROM Hotel WHERE City = 'London';
```

5. List full details of all hotels

```
SELECT * FROM Hotel;
```

6. List full details of all hotels in London

```
SELECT * FROM Hotel
WHERE City = 'London';
```

7. Update the price of all rooms by 5%

```
UPDATE Room
SET Price = Price * 1.05;
```

8. Number of rooms in each hotel in London

```
SELECT h.Name, COUNT(r.RoomNo) AS No_of_Rooms
FROM Hotel h
JOIN Room r ON h.HotelNo = r.HotelNo
WHERE h.City = 'London'
GROUP BY h.Name;
```

9. All double or family rooms below £40, ascending order of price

```
SELECT *
FROM Room
WHERE (Type = 'Double' OR Type = 'Family')
  AND Price < 40
ORDER BY Price ASC;
```

5. The following tables form part of a database held in a relational DBMS:

Hotel (HotelNo, Name, City) HotelNo is the primary key

Room (RoomNo, HotelNo, Type, Price)

Booking (HotelNo, GuestNo, DateFrom, DateTo, RoomNo)

Guest (GuestNo, GuestName, GuestAddress)

Solve following queries by SQL

1. List full details of all hotels.

2. How many hotels are there?

3. List the price and type of all rooms at the Grosvenor Hotel.

4. List the number of rooms in each hotel

5. List all guests currently staying at the Grosvenor Hotel.

6. List all double or family rooms with a price below £40.00 per night, in ascending order of price.

7. How many different guests have made bookings for August?

8. What is the total income from bookings for the Grosvenor Hotel today?

9. What is the most commonly booked room type for each hotel in London?

10. Update the price of all rooms by 5%.

1. All hotels

```
SELECT * FROM Hotel;
```

2. Total hotels

```
SELECT COUNT(*) FROM Hotel;
```

3. Price & type of Grosvenor Hotel rooms

```
SELECT Type, Price
FROM Room r JOIN Hotel h ON r.HotelNo = h.HotelNo
WHERE h.Name = 'Grosvenor Hotel';
```

4. Number of rooms in each hotel

```
SELECT h.Name, COUNT(r.RoomNo)
FROM Hotel h JOIN Room r ON h.HotelNo = r.HotelNo
GROUP BY h.Name;
```

5. Guests staying at Grosvenor Hotel

```
SELECT g.GuestName
FROM Guest g JOIN Booking b ON g.GuestNo = b.GuestNo
JOIN Hotel h ON b.HotelNo = h.HotelNo
WHERE h.Name='Grosvenor Hotel'
```

AND CURRENT_DATE BETWEEN b.DateFrom AND b.DateTo;

6. Double or family rooms below £40

```
SELECT * FROM Room
WHERE (Type='Double' OR Type='Family') AND Price<40
ORDER BY Price;
```

7. Guests booked in August

```
SELECT COUNT(DISTINCT GuestNo)
FROM Booking
WHERE MONTH(DateFrom)=8 OR MONTH(DateTo)=8;
```

8. Total income today (Grosvenor Hotel)

```
SELECT SUM(r.Price)
FROM Booking b JOIN Room r ON b.RoomNo=r.RoomNo
JOIN Hotel h ON b.HotelNo=h.HotelNo
WHERE h.Name='Grosvenor Hotel'
AND CURRENT_DATE BETWEEN b.DateFrom AND b.DateTo;
```

9. Most booked room type (London hotels)

```
SELECT h.Name, r.Type, COUNT(*) AS cnt
FROM Booking b JOIN Room r ON b.RoomNo=r.RoomNo
JOIN Hotel h ON h.HotelNo=b.HotelNo
WHERE h.City='London'
GROUP BY h.Name, r.Type;
```

10. Increase all room prices by 5%

```
UPDATE Room SET Price = Price * 1.05;
```

---

6. The following tables form part of a database held in a relational DBMS:

Hotel (HotelNo, Name, City)

Room (RoomNo, HotelNo, Type, Price)

Booking (HotelNo, GuestNo, DateFrom, DateTo, RoomNo)

Guest (GuestNo, GuestName, GuestAddress)

Solve following queries by SQL

1. List full details of all hotels.

2. List full details of all hotels in London.

3. List all guests currently staying at the Grosvenor Hotel.

4. List the names and addresses of all guests in London, alphabetically ordered by name.

5. List the bookings for which no date_to has been specified.

6. How many hotels are there?

7. List the rooms that are currently unoccupied at the Grosvenor Hotel.

8. What is the lost income from unoccupied rooms at each hotel today?

9. Create index on one of the field and show is performance in query.

10. Create one view on above database and query it.

1. All hotels

```
SELECT * FROM Hotel;
```

2. Hotels in London

```
SELECT * FROM Hotel
WHERE City = 'London';
```

3. Guests staying at Grosvenor Hotel

```
SELECT g.GuestName
FROM Guest g JOIN Booking b ON g.GuestNo = b.GuestNo
JOIN Hotel h ON b.HotelNo = h.HotelNo
WHERE h.Name = 'Grosvenor Hotel'
AND CURRENT_DATE BETWEEN b.DateFrom AND b.DateTo;
```

4. Guests in London (A–Z by name)

```
SELECT GuestName, GuestAddress
FROM Guest
WHERE GuestAddress LIKE '%London%'
ORDER BY GuestName;
```

5. Bookings with no DateTo

```
SELECT * FROM Booking
WHERE DateTo IS NULL;
```

6. Total number of hotels

```
SELECT COUNT(*) AS Total_Hotels
FROM Hotel;
```

7. Unoccupied rooms at Grosvenor Hotel

```
SELECT r.RoomNo, r.Type, r.Price
FROM Room r
JOIN Hotel h ON r.HotelNo = h.HotelNo
WHERE h.Name = 'Grosvenor Hotel'
AND r.RoomNo NOT IN (
  SELECT RoomNo FROM Booking
```

```
  WHERE CURRENT_DATE BETWEEN DateFrom AND DateTo
);
```

8. Lost income from unoccupied rooms today

```
SELECT h.Name, SUM(r.Price) AS Lost_Income
FROM Hotel h
JOIN Room r ON h.HotelNo = r.HotelNo
WHERE r.RoomNo NOT IN (
  SELECT RoomNo FROM Booking
  WHERE CURRENT_DATE BETWEEN DateFrom AND DateTo
)
GROUP BY h.Name;
```

9. Create index & show performance

```
CREATE INDEX idx_city ON Hotel(City);
EXPLAIN SELECT * FROM Hotel WHERE City = 'London';
```

10. Create view & query it

Create view:

```
CREATE VIEW London_Hotels AS
SELECT * FROM Hotel WHERE City = 'London';
```

Query view:

```
SELECT * FROM London_Hotels;
```

---

7. Consider the following database

Project(project_id,proj_name,chief_arch) , project_id is primary key

Employee(Emp_id,Emp_name) , Emp_id is primary key

Assigned-To(Project_id,Emp_id)

Find the SQL queries for the following:

1. Get the details of employees working on project C353

2. Get employee number of employees working on project C353

3. Obtain details of employees working on Database project

4. Get details of employees working on both C353 and C354

5. Get employee numbers of employees who do not work on project C453

1. Get details of employees working on project C353

```
SELECT E.*
FROM Employee E
```

```
JOIN Assigned_To A ON E.Emp_id = A.Emp_id
WHERE A.Project_id = 'C353';
```

2. Get employee number of employees working on project C353

```
SELECT Emp_id
FROM Assigned_To
WHERE Project_id = 'C353';
```

3. Obtain details of employees working on Database project

```
SELECT E.*
FROM Employee E
JOIN Assigned_To A ON E.Emp_id = A.Emp_id
JOIN Project P ON P.Project_id = A.Project_id
WHERE P.proj_name = 'Database';
```

4. Get details of employees working on both C353 and C354

```
SELECT E.*
FROM Employee E
WHERE E.Emp_id IN (
  SELECT Emp_id FROM Assigned_To WHERE Project_id = 'C353'
)
AND E.Emp_id IN (
  SELECT Emp_id FROM Assigned_To WHERE Project_id = 'C354'
);
```

5. Get employee numbers of employees who do not work on project C453

```
SELECT Emp_id
FROM Employee
WHERE Emp_id NOT IN (
  SELECT Emp_id FROM Assigned_To WHERE Project_id = 'C453'
);
```

---

8. Consider the following database

Employee(emp_no,name,skill,pay-rate) eno primary key

Position(posting_no,skill) posting_no primary key

Duty_allocation(posting_no,emp_no,day,shift)

Find the SQL queries for the following:

1. Get the duty allocation details for emp_no 123461 for the month of April 1986.

2. Find the shift details for Employee ‘xyz’

3. Get employees whose rate of pay is more than or equal to the rate of pay of employee ‘xyz’

4. Get the names and pay rates of employees with emp_no less than 123460 whose rate of pay is more than the rate of pay of at least one employee with emp_no greater than or equal to 123460.

5. Find the names of employees who are assigned to all positions that require a Chef’s skill

6 .Find the employees with the lowest pay rate

7 .Get the employee numbers of all employees working on at least two dates.

8 .Get a list of names of employees with the skill of Chef who are assigned a duty

9 .Get a list of employees not assigned a duty

10.Get a count of different employees on each shift

1. Get duty allocation details for emp_no 123461 for April 1986

```
SELECT *
FROM Duty_allocation
WHERE emp_no = 123461
  AND day BETWEEN '1986-04-01' AND '1986-04-30';
```

2. Find shift details for Employee ‘xyz’

```
SELECT D.shift
FROM Duty_allocation D
JOIN Employee E ON D.emp_no = E.emp_no
WHERE E.name = 'xyz';
```

3. Get employees whose pay rate ≥ pay rate of employee ‘xyz’

```
SELECT *
FROM Employee
WHERE pay_rate >= (
  SELECT pay_rate FROM Employee WHERE name = 'xyz'
);
```

4. Get names and pay rates of employees

(emp_no < 123460) with pay_rate > at least one emp (emp_no ≥ 123460)

```
SELECT name, pay_rate
FROM Employee E1
WHERE emp_no < 123460
  AND pay_rate > ANY (
    SELECT pay_rate FROM Employee E2 WHERE emp_no >= 123460
);
```

5. Employees assigned to all positions that require Chef’s skill

```
SELECT E.name
FROM Employee E
WHERE E.skill = 'Chef'
  AND NOT EXISTS (
    SELECT P.posting_no
    FROM Position P
    WHERE P.skill = 'Chef'
      AND P.posting_no NOT IN (
        SELECT D.posting_no
        FROM Duty_allocation D
        WHERE D.emp_no = E.emp_no
      )
  );
```

6. Employees with the lowest pay rate

```
SELECT *
FROM Employee
WHERE pay_rate = (SELECT MIN(pay_rate) FROM Employee);
```

7. Employee numbers working on at least two dates

```
SELECT emp_no
FROM Duty_allocation
GROUP BY emp_no
HAVING COUNT(DISTINCT day) >= 2;
```

8. Names of employees (skill = Chef) who are assigned a duty

```
SELECT DISTINCT E.name
FROM Employee E
JOIN Duty_allocation D ON E.emp_no = D.emp_no
WHERE E.skill = 'Chef';
```

9. Employees not assigned any duty

```
SELECT E.name
FROM Employee E
WHERE E.emp_no NOT IN (SELECT emp_no FROM Duty_allocation);
```

10. Count of different employees on each shift

```
SELECT shift, COUNT(DISTINCT emp_no) AS total_employees
FROM Duty_allocation
GROUP BY shift;
```

9. Create the following tables. And Solve following queries by SQL

• Deposit (actno,cname,bname,amount,adate)

• Branch (bname,city)

• Customers (cname, city)

• Borrow(loanno,cname,bname, amount)

Add primary key and foreign key wherever applicable.

Insert data into the above created tables.

1. Display names of depositors having amount greater than 4000.

2. Display account date of customers Anil

3. Display account no. and deposit amount of customers having account opened

between dates 1-12-96 and 1-5-97

4. Find the average account balance at the Perryridge branch.

5. Find the names of all branches where the average account balance is more

than $1,200.

6. Delete depositors having deposit less than 5000

7. Create a view on deposit table.

1. Depositors having amount > 4000

```
SELECT cname, amount
FROM Deposit
WHERE amount > 4000;
```

2. Account date of customer ‘Anil’

```
SELECT adate
FROM Deposit
WHERE cname = 'Anil';
```

3. Account no. & deposit amount for customers opened between 1-Dec-96 and 1-May-97

```
SELECT actno, amount
FROM Deposit
WHERE adate BETWEEN '1996-12-01' AND '1997-05-01';
```

4. Average account balance at Perryridge branch

```
SELECT AVG(amount) AS avg_balance
FROM Deposit
WHERE bname = 'Perryridge';
```

5. Branches where avg account balance > $1200

```
SELECT bname
```

FROM Deposit

GROUP BY bname

HAVING AVG(amount) > 1200;

6. Delete depositors having deposit < 5000

DELETE FROM Deposit

WHERE amount < 5000;

7. Create a view on deposit table

CREATE VIEW HighDeposits AS

SELECT cname, bname, amount

FROM Deposit

WHERE amount > 5000;

✅ *You can now query it as:*

SELECT * FROM HighDeposits;

---

10. Create the following tables. And Solve following queries by SQL

1. Deposit (actno,cname,bname,amount,adate)

2. Branch (bname,city)

3. Customers (cname, city)

4. Borrow(loanno,cname,bname, amount)

Add primary key and foreign key wherever applicable.

Insert data into the above created tables.

a. Display names of all branches located in city Bombay.

b. Display account no. and amount of depositors.

c. Update the city of customers Anil from Pune to Mumbai

d. Find the number of depositors in the bank

e. Calculate Min,Max amount of customers.

f. Create an index on deposit table

g. Create View on Borrow table.

1. Display names of all branches located in city Bombay

SELECT bname

FROM Branch

WHERE city = 'Bombay';

2. Display account no. and amount of depositors

SELECT actno, amount

FROM Deposit;

3. Update the city of customer Anil from Pune → Mumbai

UPDATE Customers

SET city = 'Mumbai'

WHERE cname = 'Anil';

4. Find the number of depositors in the bank

SELECT COUNT(*) AS total_depositors

FROM Deposit;

5. **Calculate Min and Max amount of customers**

SELECT MIN(amount) AS Min_Amount, MAX(amount) AS Max_Amount

FROM Deposit;

6. Create an index on Deposit table

CREATE INDEX idx_deposit_bname ON Deposit(bname);

7. Create a view on Borrow table

CREATE VIEW BorrowView AS

SELECT cname, bname, amount

FROM Borrow;

**You can now query it:

SELECT * FROM BorrowView;

---

11. Create the following tables. Solve queries by SQL

• Deposit (actno,cname,bname,amount,adate)

• Branch (bname,city)

• Customers (cname, city)

• Borrow(loanno,cname,bname, amount)

Add primary key and foreign key wherever applicable.

Insert data into the above created tables.

a. Display account date of customers Anil.

b. Modify the size of attribute of amount in deposit

c. Display names of customers living in city pune.

d. Display name of the city where branch KAROLBAGH is located.

e. Find the number of tuples in the customer relation

f. Delete all the record of customers Sunil

g. Create a view on deposit table.

1. Display account date of customer Anil

```
SELECT adate
FROM Deposit
WHERE cname = 'Anil';
```

2. Modify the size of attribute amount in Deposit

```
ALTER TABLE Deposit
MODIFY amount DECIMAL(12,2);
```

3. Display names of customers living in Pune

```
SELECT cname
FROM Customers
WHERE city = 'Pune';
```

4. Display city where branch KAROLBAGH is located

```
SELECT city
FROM Branch
WHERE bname = 'KAROLBAGH';
```

5. Find number of tuples in Customer relation

```
SELECT COUNT(*) AS total_customers
FROM Customers;
```

6. Delete all records of customer Sunil

```
DELETE FROM Customers
WHERE cname = 'Sunil';
```

7. Create a view on Deposit table

```
CREATE VIEW Deposit_View AS
SELECT cname, bname, amount
FROM Deposit;
```

View check:

```
SELECT * FROM Deposit_View;
```

---

12. Create the following tables. Solve queries by SQL

• Deposit (actno,cname,bname,amount,adate)

• Branch (bname,city)

• Customers (cname, city)

• Borrow(loanno,cname,bname, amount)

Add primary key and foreign key wherever applicable.

Insert data into the above created tables.

Solve following queries by SQL

1. Display customer name having living city Bombay and branch city Nagpur

2. Display customer name having same living city as their branch city

3. Display customer name who are borrowers as well as depositors and having living city Nagpur.

4. Display borrower names having deposit amount greater than 1000 and loan amount greater than 2000

5. Display customer name living in the city where branch of depositor sunil is located.

6. Create an index on deposit table

1. Customer living city = *Bombay* and branch city = *Nagpur**

```
SELECT d.cname
FROM Deposit d
JOIN Customers c ON d.cname = c.cname
JOIN Branch b ON d.bname = b.bname
WHERE c.city = 'Bombay' AND b.city = 'Nagpur';
```

2. Customers having same living city as their branch city

```
SELECT d.cname
FROM Deposit d
JOIN Customers c ON d.cname = c.cname
JOIN Branch b ON d.bname = b.bname
WHERE c.city = b.city;
```

3. Customers who are borrowers & depositors and live in Nagpur

```
SELECT DISTINCT d.cname
FROM Deposit d
JOIN Borrow br ON d.cname = br.cname
JOIN Customers c ON d.cname = c.cname
WHERE c.city = 'Nagpur';
```

4. Borrowers having deposit > 1000 and loan > 2000

```
SELECT d.cname
FROM Deposit d
JOIN Borrow b ON d.cname = b.cname
WHERE d.amount > 1000 AND b.amount > 2000;
```

5. Customers living in the same city as branch of depositor Sunil

```
SELECT c.cname
```

```
FROM Customers c
WHERE c.city = (
  SELECT b.city
  FROM Branch b
  JOIN Deposit d ON b.bname = d.bname
  WHERE d.cname = 'Sunil'
);
```

6. Create an index on Deposit table

```
CREATE INDEX idx_deposit_city
ON Deposit(bname, amount);
```

---

13) Create the following tables.

1)PUBLISHER(PID , PNAME ,ADDRESS ,STATE ,PHONE ,EMAILID);

2)BOOK(ISBN ,BOOK_TITLE , CATEGORY , PRICE , COPYRIGHT_DATE , YEAR ,PAGE_COUNT ,PID);

3) AUTHOR(AID,ANAME,STATE,CITY ,ZIP,PHONE,URL)

4) AUTHOR_BOOK(AID,ISBN);

5) REVIEW(RID,ISBN,RATING);

Solve following queries by SQL

1. Retrieve city, phone, url of author whose name is ‘CHETAN BHAGAT’.

2. Retrieve book title, reviewable id and rating of all books.

3. Retrieve book title, price, author name and url for publishers ‘MEHTA’.

4. In a PUBLISHER relation change the phone number of ‘MEHTA’ to 123456

5. Calculate and display the average, maximum, minimum price of each publisher.

6. Delete details of all books having a page count less than 100.

7. Retrieve details of all authors residing in city Pune and whose name begins with character

‘C’.

8. Retrieve details of authors residing in same city as ‘Korth’.

9. Create a procedure to update the value of page count of a book of given ISBN.

10. Create a function that returns the price of book with a given ISBN.

1. Retrieve city, phone, url of author whose name is ‘CHETAN BHAGAT’.

```
SELECT CITY, PHONE, URL
FROM AUTHOR
WHERE ANAME = 'CHETAN BHAGAT';
```

2. Retrieve book title, review id and rating of all books.

```
SELECT B.BOOK_TITLE, R.RID, R.RATING
```

```
FROM BOOK B
JOIN REVIEW R ON B.ISBN = R.ISBN;
```

3. Retrieve book title, price, author name and url for publisher ‘MEHTA’.

```
SELECT B.BOOK_TITLE, B.PRICE, A.ANAME, A.URL
FROM BOOK B
JOIN PUBLISHER P ON B.PID = P.PID
JOIN AUTHOR_BOOK AB ON B.ISBN = AB.ISBN
JOIN AUTHOR A ON AB.AID = A.AID
WHERE P.PNAME = 'MEHTA';
```

4. Update phone number of publisher ‘MEHTA’ to 123456.

```
UPDATE PUBLISHER
SET PHONE = 123456
WHERE PNAME = 'MEHTA';
```

5. Display average, maximum, and minimum price of each publisher.

```
SELECT P.PNAME,
    AVG(B.PRICE) AS AVG_PRICE,
    MAX(B.PRICE) AS MAX_PRICE,
    MIN(B.PRICE) AS MIN_PRICE
FROM PUBLISHER P
JOIN BOOK B ON P.PID = B.PID
GROUP BY P.PNAME;
```

6. Delete books having page count < 100.

```
DELETE FROM BOOK
WHERE PAGE_COUNT < 100;
```

7. Retrieve authors living in city ‘Pune’ and name begins with ‘C’.

```
SELECT *
FROM AUTHOR
WHERE CITY = 'Pune'
  AND ANAME LIKE 'C%';
```

8. Retrieve authors residing in the same city as ‘Korth’.

```
SELECT *
FROM AUTHOR
WHERE CITY = (SELECT CITY FROM AUTHOR WHERE ANAME = 'Korth');
```

9. Procedure → Update page count of a book (given ISBN).

```
CREATE PROCEDURE UpdatePageCount(
```

```
    IN p_ISBN VARCHAR(20),
    IN newPageCount INT
)
BEGIN
    UPDATE BOOK
    SET PAGE_COUNT = newPageCount
    WHERE ISBN = p_ISBN;
END;
```

Call Example

```
CALL UpdatePageCount('B101', 250);
```

10. Function → Return price of book (given ISBN).

```
CREATE FUNCTION GetBookPrice(p_ISBN VARCHAR(20))
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE b_price DECIMAL(10,2);
    SELECT PRICE INTO b_price
    FROM BOOK
    WHERE ISBN = p_ISBN;
    RETURN b_price;
END;
```

** **Use Example**

```
SELECT GetBookPrice('B101');
```

PL/SQL

14.

a) Consider table Stud(Roll, Att,Status)

Write a PL/SQL block for following requirement and handle the exceptions. Roll no. of student will be entered by user. Attendance of roll no. entered by user will be checked in Stud table. If attendance is less than 75% then display the message “Term not granted” and set the status in stud table as “D”. Otherwise display message “Term granted” and set the status in stud table as “ND”

ANS:

```
DECLARE
    v_roll STUD.Roll%TYPE;
```

```
   v_att  STUD.Att%TYPE;
BEGIN
   -- Accept roll number from user
   v_roll := &Roll;

   SELECT Att INTO v_att
   FROM STUD
   WHERE Roll = v_roll;

   IF v_att < 75 THEN
      DBMS_OUTPUT.PUT_LINE('Term not granted');
      UPDATE STUD SET Status = 'D' WHERE Roll = v_roll;
   ELSE
      DBMS_OUTPUT.PUT_LINE('Term granted');
      UPDATE STUD SET Status = 'ND' WHERE Roll = v_roll;
   END IF;

EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Student not found');
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Unexpected error occurred');
END;
/
```

b) Write a PL/SQL block for following requirement using user defined exception handling. The account_master table records the current balance for an account, which is updated whenever, any deposits or withdrawals takes place. If the withdrawal attempted is more than the current balance held in the account. The user defined exception is raised, displaying an appropriate message.

Write a PL/SQL block for above requirement using user defined exception handling.

ANS:

Table:
ACCOUNT_MASTER (ACCNO, BALANCE)

PLSQL Block:

```
DECLARE
   v_accno ACCOUNT_MASTER.ACCNO%TYPE;
```

```
  v_amt   NUMBER;
  v_bal   ACCOUNT_MASTER.BALANCE%TYPE;
  e_insufficient_balance EXCEPTION;
BEGIN
  v_accno := &Account_No;
  v_amt := &Withdraw_Amount;

  SELECT BALANCE INTO v_bal
  FROM ACCOUNT_MASTER
  WHERE ACCNO = v_accno;

  IF v_amt > v_bal THEN
    RAISE e_insufficient_balance;
  ELSE
    UPDATE ACCOUNT_MASTER
    SET BALANCE = BALANCE - v_amt
    WHERE ACCNO = v_accno;
    DBMS_OUTPUT.PUT_LINE('Withdrawal Successful');
  END IF;

EXCEPTION
  WHEN e_insufficient_balance THEN
    DBMS_OUTPUT.PUT_LINE('Error: Insufficient Balance');
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Account not found');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Unexpected error occurred');
END;
/
```

15.

a) Write an SQL code block these raise a user defined exception where business rule is voilated. BR for client_ master table specifies when the value of bal_due field is less than 0 handle the exception.

Table: Client_Master(cid, cname, bal_due)

PLSQL block:

```
DECLARE
  v_bal  CLIENT_MASTER.bal_due%TYPE;
  e_invalid_balance EXCEPTION;
BEGIN
  SELECT bal_due INTO v_bal FROM CLIENT_MASTER WHERE cid = &cid;

  IF v_bal < 0 THEN
    RAISE e_invalid_balance;
  ELSE
    DBMS_OUTPUT.PUT_LINE('Balance valid');
  END IF;

EXCEPTION
  WHEN e_invalid_balance THEN
    DBMS_OUTPUT.PUT_LINE('Error: Balance due cannot be negative');
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Client not found');
END;
/
```

b) Write an SQL code block

Borrow(Roll_no, Name, DateofIssue, NameofBook, Status)

Fine(Roll_no,Date,Amt)

Accept roll_no & name of book from user. Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day. If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day. After submitting the book, status will change from I to R. If condition of fine is true, then details will be stored into fine table. Also handles the exception by named exception handler or user define exception handler.

```
DECLARE
  v_roll Borrow.Roll_no%TYPE;
  v_book Borrow.NameOfBook%TYPE;
  v_days NUMBER;
  v_fine NUMBER := 0;
```

```
BEGIN
  v_roll := &Roll_no;
  v_book := '&Book_Name';
  SELECT (SYSDATE - DateOfIssue) INTO v_days
  FROM Borrow
  WHERE Roll_no = v_roll AND NameOfBook = v_book;
  IF v_days BETWEEN 15 AND 30 THEN
    v_fine := v_days * 5;
  ELSIF v_days > 30 THEN
    v_fine := v_days * 50;
  END IF;
  UPDATE Borrow
  SET Status = 'R'
  WHERE Roll_no = v_roll;
  IF v_fine > 0 THEN
    INSERT INTO Fine VALUES(v_roll, SYSDATE, v_fine);
    DBMS_OUTPUT.PUT_LINE('Fine amount = ' || v_fine);
  ELSE
    DBMS_OUTPUT.PUT_LINE('No fine');
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Book record not found');
END;
/
```

---

16. Cursor (Any Two)

a) The bank manager has decided to activate all those accounts which were previously marked as inactive for performing no transaction in last 365 days. Write a PL/SQ block (using implicit cursor) to update the status of account, display an approximate message based on the no. of rows affected by the update. (Use of %FOUND, %NOTFOUND, %ROWCOUNT)

```
BEGIN
  UPDATE Account
  SET Status = 'Active'
  WHERE Status = 'Inactive'
```

```
  AND Last_Transaction < SYSDATE - 365;
  IF SQL%FOUND THEN
    DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' Accounts Activated');
  ELSE
    DBMS_OUTPUT.PUT_LINE('No accounts updated');
  END IF;
END;
/
```

b)Organization has decided to increase the salary of employees by 10% of existing salary, who are having salary less than average salary of organization, Whenever such salary updates takes place, a record for the same is maintained in the increment_salary table.

```
DECLARE
  v_avg NUMBER;
BEGIN
  SELECT AVG(Salary) INTO v_avg FROM Employee;
  UPDATE Employee
  SET Salary = Salary * 1.1
  WHERE Salary < v_avg;
  INSERT INTO Increment_Salary
  SELECT e_no, Salary FROM Employee WHERE Salary < v_avg;
  DBMS_OUTPUT.PUT_LINE('Salary updated successfully');
END;
/
```

c) Write PL/SQL block using explicit cursor for following requirements: College has decided to mark all those students detained (D) who are having attendance less than 75%. Whenever such update takes place, a record for the same is maintained in the D_Stud table. create table stud21(roll number(4), att number(4), status varchar(1));

---

17. Cursor (Any Two)

a) The bank manager has decided to activate all those accounts which were previously marked as inactive for performing no transaction in last 365 days. Write a PL/SQ block (using implicit cursor) to update the status of account, display an approximate message based on the no. of rows affected by the update. (Use of %FOUND, %NOTFOUND, %ROWCOUNT)

```
BEGIN
  UPDATE Account
  SET Status = 'Active'
  WHERE Status = 'Inactive'
  AND Last_Transaction < SYSDATE - 365;
  IF SQL%FOUND THEN
    DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' Accounts Activated');
  ELSE
    DBMS_OUTPUT.PUT_LINE('No accounts updated');
  END IF;
END;
/
```

b)Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped. output:

c)Write the PL/SQL block for following requirements using parameterized Cursor: Consider table EMP(e_no, d_no, Salary), department wise average salary should be inserted into new table dept_salary(d_no, Avg_salary)

```
DECLARE
  CURSOR c1 IS
    SELECT d_no, AVG(Salary) AS avg_sal
    FROM Emp GROUP BY d_no;
  v_dno Emp.d_no%TYPE;
  v_avg NUMBER;
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO v_dno, v_avg;
    EXIT WHEN c1%NOTFOUND;
    INSERT INTO Dept_Salary VALUES(v_dno, v_avg);
  END LOOP;
  CLOSE c1;
  DBMS_OUTPUT.PUT_LINE('Department-wise average inserted');
```

```
END;
/
```

18.Trigger

a) Write a update, delete trigger on clientmstr table. The System should keep track of the records that ARE BEING updated or deleted. The old value of updated or deleted records should be added in audit_trade table. (separate implementation using both row and statement triggers).

** Row-Level Trigger

```
CREATE OR REPLACE TRIGGER trg_client_audit_row
AFTER UPDATE OR DELETE ON ClientMstr
FOR EACH ROW
BEGIN
  INSERT INTO Audit_Trade(cid, cname, bal_due, action_date, action_type)
  VALUES(:OLD.cid, :OLD.cname, :OLD.bal_due, SYSDATE,
      CASE WHEN UPDATING THEN 'UPDATE' ELSE 'DELETE' END);
END;
/
```

** Statement-Level Trigger

```
CREATE OR REPLACE TRIGGER trg_client_audit_stmt
AFTER UPDATE OR DELETE ON ClientMstr
BEGIN
  INSERT INTO Audit_Trade_Log(action_done, action_time)
  VALUES('ClientMstr modified', SYSDATE);
END;
/
```

b) Write a before trigger for Insert, update event considering following requirement: Emp(e_no, e_name, salary) I) Trigger action should be initiated when salary is tried to be inserted is less than Rs. 50,000/- II) Trigger action should be initiated when salary is tried to be updated for value less than Rs. 50,000/- Action should be rejection of update or Insert operation by displaying appropriate error message. Also the new values expected to be inserted will be stored in new table Tracking(e_no, salary).

```
CREATE OR REPLACE TRIGGER trg_salary_check
BEFORE INSERT OR UPDATE ON Emp
FOR EACH ROW
BEGIN
  IF :NEW.salary < 50000 THEN
    INSERT INTO Tracking(e_no, salary)
    VALUES(:NEW.e_no, :NEW.salary);

    RAISE_APPLICATION_ERROR(-20001,
      'Error: Salary must be at least Rs. 50000/-');
  END IF;
END;
/
```

Mongodb

19.

Create Database DYPIT using MongoDB

Create following Collections

Teachers(Tname,dno,dname,experience,salary,date_of_joining)

Students(Sname,roll_no,class)

1. Find the information about all teachers

2. Find the information about all teachers of computer department

3. Find the information about all teachers of computer,IT,and e&TC department

4. Find the information about all teachers of computer,IT,and E&TC department having

salary greate than or equl to 10000/-

5. Find the student information having roll_no = 2 or Sname=xyz

6. Update the experience of teacher-praveen to 10years, if the entry is not available in

database consider the entry as new entry.

7. Update the deparment of all the teachers working in IT deprtment to COMP

8. find the teachers name and their experience from teachers collection

9. Using Save() method insert one entry in department collection

10. Using Save() method change the dept of teacher Rajesh to IT

11. Delete all the doccuments from teachers collection having IT dept.

12. display with pretty() method, the first 3 doccuments in teachers collection in ascending

Order

1. All teachers info

```
db.Teachers.find().pretty()
```

2. Teachers of Computer department

```
db.Teachers.find({dname:"Computer"}).pretty()
```

3. Teachers of Computer, IT, and E&TC dept

```
db.Teachers.find({dname:{$in:["Computer","IT","E&TC"]}}).pretty()
```

4. Teachers of those depts with salary ≥ 10000

```
db.Teachers.find({
  dname:{$in:["Computer","IT","E&TC"]},
  salary:{$gte:10000}
}).pretty()
```

5. Student info with roll_no=2 or name='xyz'

```
db.Students.find({
  $or:[{roll_no:2},{Sname:"xyz"}]
}).pretty()
```

6. Update experience of Praveen → 10 years

(if not exist, insert new)

```
db.Teachers.updateOne(
  {Tname:"Praveen"},
  {$set:{experience:10}},
  {upsert:true}
)
```

7. Update dept of all IT teachers → COMP

```
db.Teachers.updateMany(
  {dname:"IT"},
  {$set:{dname:"COMP"}}
)
```

8. Show teacher name & experience only

```
db.Teachers.find(
  {},
  {Tname:1, experience:1, _id:0}
)
```

9. Insert new department using save()

```
db.Department.save({dno:4, dname:"Mechanical"})
```

10. Change dept of teacher Rajesh → IT using save()

```
db.Teachers.save({
  Tname:"Rajesh", dno:2, dname:"IT",
  experience:4, salary:9000, date_of_joining:"2020-03-12"
})
```

11. Delete all teachers from IT dept

```
db.Teachers.deleteMany({dname:"IT"})
```

12. Display first 3 teachers (ascending order)

```
db.Teachers.find().sort({Tname:1}).limit(3).pretty()
```

20

1.Create Database DYPIT

2. Create following Collections

Teachers(Tname,dno,dname,experience,salary,date_of_joining)

Students(Sname,roll_no,class)

3. Find the information about two teachers

4. Find the information about all teachers of computer department

5. Find the information about all teachers of computer,IT,and e&TC department

6. Find the information about all teachers of computer,IT,and E&TC department having salary greate than or equl to 25000/-

7. Find the student information having roll_no = 25 or Sname=xyz

8. Update the experience of teacher-praveen to 10years, if the entry is not available in database consider the entry as new entry.

9. Update the deparment of all the teachers working in IT deprtment to COMP

10. find the teachers name and their experience from teachers collection

11. Using Save() method insert one entry in department collection

13. Delete all the doccuments from teachers collection having IT dept.

14. display with pretty() method, the first 5 documents in teachers collection in ascending order

3. Find info about two teachers

```
db.Teachers.find().limit(2).pretty()
```

4. Find all teachers of Computer department

```
db.Teachers.find({dname:"Computer"}).pretty()
```

5. Teachers of Computer, IT, and E&TC departments

```
db.Teachers.find({dname:{$in:["Computer","IT","E&TC"]}}).pretty()
```

6. Teachers of those depts with salary ≥ 25000

```
db.Teachers.find({
  dname:{$in:["Computer","IT","E&TC"]},
  salary:{$gte:25000}
}).pretty()
```

7. Student info with roll_no = 25 or name = xyz

```
db.Students.find({
  $or:[{roll_no:25},{Sname:"xyz"}]
}).pretty()
```

8. Update experience of Praveen to 10 years (upsert = true)

```
db.Teachers.updateOne(
  {Tname:"Praveen"},
  {$set:{experience:10}},
  {upsert:true}
)
```

9. Update dept of all IT teachers → COMP

```
db.Teachers.updateMany(
  {dname:"IT"},
  {$set:{dname:"COMP"}}
)
```

10. Show teacher name & experience only

```
db.Teachers.find(
  {},
  {Tname:1, experience:1, _id:0}
)
```

11. Insert entry in Department collection using save()

```
db.Department.save({dno:5, dname:"Civil"})
```

13. Delete all teachers having IT dept

```
db.Teachers.deleteMany({dname:"IT"})
```

14. Display first 5 teachers (ascending order)

```
db.Teachers.find().sort({Tname:1}).limit(5).pretty()
```

---

21. Create Database DYPIT using MongoDB

Create following Collections

Teachers(Tname,dno,dname,experience,salary,date_of_joining)

Students(Sname,roll_no,class)

1. Find the information about all teachers

2. Find the average salary teachers of computer department

3. Find the minimum and maximum salary of e&TC department teachers

4. Find the information about all teachers of computer,IT,and E&TC department having

salary greate than or equl to 10000/-

5. Find the student information having roll_no = 2 or Sname=xyz

6. Update the experience of teacher-praveen to 10years, if the entry is not available in

database consider the entry as new entry.

7. Update the deparment of all the teachers working in IT deprtment to COMP

8. find the teachers name and their experience from teachers collection

9. Using Save() method insert one entry in department collection

10. Find the total salary all teachers.

**** Create Database and Collections**

```
use DYPIT
db.createCollection("Teachers")
db.createCollection("Students")
```

**** Insert Sample Documents**

```
db.Teachers.insertMany([
  { Tname: "Praveen", dno: 1, dname: "COMP", experience: 5, salary: 12000, date_of_joining: "2015-06-01" },
  { Tname: "Rajesh", dno: 2, dname: "IT", experience: 7, salary: 25000, date_of_joining: "2014-05-01" },
  { Tname: "Sneha", dno: 3, dname: "E&TC", experience: 6, salary: 18000, date_of_joining: "2016-04-10" }
])
db.Students.insertMany([
  { Sname: "Rohit", roll_no: 2, class: "SE" },
  { Sname: "XYZ", roll_no: 5, class: "TE" }
])
```

1. Find the information about all teachers

```
db.Teachers.find().pretty()
```

2. Find the average salary of Computer department teachers

```
db.Teachers.aggregate([
```

```
 { $match: { dname: "COMP" } },
 { $group: { _id: null, avg_salary: { $avg: "$salary" } } }
])
```

3. Find minimum and maximum salary of E&TC department teachers

```
db.Teachers.aggregate([
 { $match: { dname: "E&TC" } },
 {
   $group: {
     _id: "$dname",
     min_salary: { $min: "$salary" },
     max_salary: { $max: "$salary" }
   }
 }
])
```

4. Find teachers of Computer, IT, and E&TC departments having salary ≥ 10000

```
db.Teachers.find({
 dname: { $in: ["COMP", "IT", "E&TC"] },
 salary: { $gte: 10000 }
})
```

5. Find student info with roll_no = 2 or Sname = 'xyz'

```
db.Students.find({
 $or: [{ roll_no: 2 }, { Sname: "xyz" }]
})
```

6. Update experience of teacher 'Praveen' to 10 years (insert if not present)

```
db.Teachers.updateOne(
 { Tname: "Praveen" },
 { $set: { experience: 10 } },
 { upsert: true }
)
```

7. Update department of all IT teachers to COMP

```
db.Teachers.updateMany(
 { dname: "IT" },
 { $set: { dname: "COMP" } }
)
```

8. Find teachers' names and experience only

```
db.Teachers.find({}, { Tname: 1, experience: 1, _id: 0 })
```

9. Insert one entry in Department collection using save()

```
db.Department.save({ dno: 4, dname: "AIDS" })
```

10. Find total salary of all teachers

```
db.Teachers.aggregate([
  { $group: { _id: null, total_salary: { $sum: "$salary" } } }
])
```

---

22. Create Database DYPIT using MongoDB

Create following Collections

Teachers(Tname,dno,dname,experience,salary,date_of_joining)

Students(Sname,roll_no,class)

1. Display the department wise average salary

2. display the no. Of employees working in each department

3. Display the department wise total salary of departments having total salary greater than

or equals to 50000/-

4. Write the queries using the different operators like max, min. Etc.

5. Create unique index on any field for above given collections

6. Create compound index on any fields for above given collections

7. Show all the indexes created in the database DYPIT

8. Show all the indexes created in above collections.

1. Display department-wise average salary

```
db.Teachers.aggregate([
  { $group: { _id: "$dname", avg_salary: { $avg: "$salary" } } }
])
```

2. Display number of employees working in each department

```
db.Teachers.aggregate([
  { $group: { _id: "$dname", total_employees: { $sum: 1 } } }
])
```

3. Department-wise total salary of departments having total ≥ 50000

```
db.Teachers.aggregate([
  { $group: { _id: "$dname", total_salary: { $sum: "$salary" } } },
  { $match: { total_salary: { $gte: 50000 } } }
```

```
])
```

4. Use operators like $max and $min

```
db.Teachers.aggregate([
 {
  $group: {
   _id: "$dname",
   max_salary: { $max: "$salary" },
   min_salary: { $min: "$salary" }
  }
 }
])
```

5. Create a unique index on a field

```
db.Teachers.createIndex({ Tname: 1 }, { unique: true })
```

6. Create a compound index on multiple fields

```
db.Teachers.createIndex({ dname: 1, salary: -1 })
```

7. Show all indexes created in Database DYPIT

```
db.getCollectionNames().forEach(function(name){
 print("Indexes for Collection: " + name);
 printjson(db.getCollection(name).getIndexes());
});
```

8. Show all indexes created in Teachers collection

```
db.Teachers.getIndexes()
```

---

23. Create index and fire queries with MongoDB

1. Import zip.json.

2. Create single field, composite and multikey indexes.

3. Fire queries given below again and write your analysis.

1. Display all cities having population above 1600.

2. Display all cities in state “KS”.

3. Display location of city "TIMKEN"

Import zip.json

```
mongoimport --db mydb --collection zips --file zip.json
```

**** Create Indexes**

db.zips.createIndex({ pop: 1 })

db.zips.createIndex({ state: 1, city: 1 })

db.zips.createIndex({ loc: 1 })

****Fire Queries**

1. Cities having population above 1600

db.zips.find({ pop: { $gt: 1600 } })

2. Cities in state “KS”

db.zips.find({ state: "KS" })

3. Location of city “TIMKEN”

db.zips.find({ city: "TIMKEN" }, { loc: 1, _id: 0 })

---

24. Design and Implement following query using MongoDB

1. Create a collection called ‘games’.

2. Add 5 games to the database. Give each document the following properties:

name, gametype, rating (out of 100)

3. Write a query that returns all the games

4. Write a query that returns the 3 highest rated games.

5. Update your two favourite games to have two achievements called ‘Game

Master’ and ‘Speed Demon’.

6. Write a query that returns all the games that have both the ‘Game Maser’ and

7. the ‘Speed Demon’ achievements.

8. Write a query that returns only games that have achievements.

1. Create Collection

db.createCollection("games")

2. Insert 5 Games

```
db.games.insertMany([
  { name: "Valorant", gametype: "Shooter", rating: 90 },
  { name: "GTA V", gametype: "Action", rating: 95 },
  { name: "Minecraft", gametype: "Adventure", rating: 88 },
  { name: "FIFA 22", gametype: "Sports", rating: 80 },
  { name: "NFS Heat", gametype: "Racing", rating: 85 }
])
```

3. Display all games

```
db.games.find().pretty()
```

4. Display 3 highest-rated games

```
db.games.find().sort({ rating: -1 }).limit(3)
```

5. Update two favorite games with achievements

```
db.games.updateOne(
  { name: "GTA V" },
  { $set: { achievements: ["Game Master", "Speed Demon"] } }
)
db.games.updateOne(
  { name: "Valorant" },
  { $set: { achievements: ["Game Master", "Speed Demon"] } }
)
```

6. Find games having both achievements

```
db.games.find({ achievements: { $all: ["Game Master", "Speed Demon"] } })
```

7. Find only games that have achievements

```
db.games.find({ achievements: { $exists: true } })
```

---

25. Using MapReduce in mongodb solve following queries on given below collection.

```
{
“id” : 0,
“name” : “Leanne Flinn”,
“email” : “leanne.flinn@unilogic.com”,
“work” :”Unilogic” ,
“age” :27
“gender” :”Male”
“Salary” :16660
“hobbies”:”Acrobatics,Photography,Papier-Mache”
}
```

1. Get the count of Males and Females

****Map Function**

```
var mapGender = function() {
  emit(this.gender, 1);
};
```

****Reduce Function**

```
var reduceGender = function(key, values) {
```

```
  return Array.sum(values);
};
```

**Run MapReduce

```
db.users.mapReduce(
  mapGender,
  reduceGender,
  { out: "gender_count" }
);
```

**View Result

```
db.gender_count.find();
```

2. Count the number of users in each hobby

**Map Function

```
var mapHobby = function() {
  var hobbies = this.hobbies.split(",");
  hobbies.forEach(function(h) {
    emit(h.trim(), 1);
  });
};
```

**Reduce Function

```
var reduceHobby = function(key, values) {
  return Array.sum(values);
};
```

**Run MapReduce

```
db.users.mapReduce(
  mapHobby,
  reduceHobby,
  { out: "hobby_count" }
);
```

**View Result

```
db.hobby_count.find();
```

---

26. Using MapReduce in mongodb solve following queries on given below collection.

1. Import zip.json.

2. Find total population in each state.

in **terminal / cmd**, run:

mongoimport --db DYPIT --collection zip --file zip.json --jsonArray

**Verify Data

use DYPIT

db.zip.findOne()

****Map Function**

var mapState = function() {

  emit(this.state, this.pop);

};

****Reduce Function**

var reduceState = function(key, values) {

  return Array.sum(values);

};

****Execute MapReduce**

db.zip.mapReduce(mapState, reduceState, { out: "state_population" })

**** Display Result**

db.state_population.find().pretty()

27.Create a database called ‘library’, create a collection called ‘books’.find the number of books having pages less 250 pages and consider ad small book and greater than 250 consider as Big book using Map Reduce function.

MapReduce — Count Small and Big Books

****Map Function**

var mapBooks = function() {

  if (this.pages < 250)

    emit("Small Books", 1);

  else

    emit("Big Books", 1);

};

****Reduce Function**

var reduceBooks = function(key, values) {

  return Array.sum(values);

};

**Execute

db.books.mapReduce(mapBooks, reduceBooks, { out: "book_size_count" })

db.book_size_count.find()