SQL SECTION :

1) Consider the relation employee (emp_id,e_name,salary ,Date of Joining,Dapt_no,Designation) perform basic SQL operations.

1. Create table employee.
2. Insert 10 records in table.
3. Create a view emp_vl of table employee which has emp_id , name and dept-attributes.
4. Create view of table.
5. Update dept of any employee in view. Check whether it gets updated or not.
6. Create emp_id as primary key and show indices on table employee.
7. Show indices on table.
8. Create user defined index on any column.

ANS :

SQL Queries :

1. Create table `employee`:

```sql
CREATE TABLE employee (
    emp_id INT,
    e_name VARCHAR(50),
    salary DECIMAL(10, 2),
    Date_of_Joining DATE,
    Dept_no INT,
    Designation VARCHAR(50)
);
```

2. Insert 10 records into the table:

```sql
INSERT INTO employee (emp_id, e_name, salary, Date_of_Joining, Dept_no, Designation)
VALUES
```

```
(1, 'John Doe', 50000.00, '2022-01-01', 1, 'Manager'),
(2, 'Jane Smith', 45000.00, '2022-02-01', 2, 'Developer'),
(3, 'Bob Johnson', 40000.00, '2022-03-01', 1, 'Analyst'),
(4, 'Bob Johnson', 40000.00, '2022-03-01', 1, 'Analyst'),
(5, 'Bob Johnson', 40000.00, '2022-03-01', 1, 'Analyst'),
(6, 'Bob Johnson', 40000.00, '2022-03-01', 1, 'Analyst'),
(7, 'Bob Johnson', 40000.00, '2022-03-01', 1, 'Analyst'),
(8, 'Bob Johnson', 40000.00, '2022-03-01', 1, 'Analyst'),
(9, 'Bob Johnson', 40000.00, '2022-03-01', 1, 'Analyst'),
(10, 'Bob Johnson', 40000.00, '2022-03-01', 1, 'Analyst'),

-- Add more records as needed
;
```

3. Create a view `emp_vl` with `emp_id`, `e_name`, and `Dept_no` attributes:

```sql
CREATE VIEW emp_vl AS
SELECT emp_id, e_name, Dept_no
FROM employee;
```

4. To create a view of a table, you've already done it in step 3.

5. Update the department of an employee in the view and check if it gets updated:

```sql
```

```
UPDATE emp_vl
SET Dept_no = 3
WHERE emp_id = 1;
```

This will update the department of the employee with `emp_id` 1 in the `emp_vl` view.

6. Create `emp_id` as the primary key and show indices:

```sql
ALTER TABLE employee
ADD CONSTRAINT pk_employee PRIMARY KEY (emp_id);

-- To show indices, it depends on the specific database system you're using. For example, in MySQL, you can use:

SHOW INDEX FROM employee;
```

7. Showing indices on a table depends on the specific database system you're using. The command might be different for MySQL, PostgreSQL, SQLite, etc. Please let me know which database system you're using for more specific instructions.

8. Create a user-defined index on any column. Here's an example for MySQL:

```sql
CREATE INDEX idx_salary ON employee (salary);
```

This will create an index named `idx_salary` on the `salary` column of the `employee` table.

2) Consider the relation employee (emp_id,e_name,salary ,Date of Joining,Dapt_no,Designation) perform basic SQL operations.

1. Display employees whose name contains letter ‘e’.
2. Display different types of designation
3. Display name and salary of employee whose location is Mumbai
4. Display name and department of employee working in Manager or Marketing department
5. Display the department name whose employees are more than one
6. Rename employee table as emp1
7. Add a new column city in the employee table.

ANS :

SQL Queries :

To perform the requested operations, let's start by creating the `employee` table and then proceed with the queries:

1. Create the `employee` table:

```sql
CREATE TABLE employee (
    emp_id INT PRIMARY KEY,
    e_name VARCHAR(50),
    salary DECIMAL(10, 2),
    Date_of_Joining DATE,
    Dept_no INT,
    Designation VARCHAR(50)
);
```

2. Insert some sample data (you can replace this with actual data):

```sql
INSERT INTO employee (emp_id, e_name, salary, Date_of_Joining, Dept_no, Designation)
VALUES
    (1, 'John Doe', 50000.00, '2022-01-01', 1, 'Manager'),
    (2, 'Jane Smith', 45000.00, '2022-02-01', 2, 'Developer'),
    (3, 'Bob Johnson', 40000.00, '2022-03-01', 1, 'Analyst'),
    (4, 'Emily Brown', 55000.00, '2022-04-01', 3, 'Manager'),
    (5, 'Michael Davis', 60000.00, '2022-05-01', 3, 'Analyst'),
    (6, 'Sarah Wilson', 48000.00, '2022-06-01', 2, 'Developer'),
    (7, 'Daniel Lee', 52000.00, '2022-07-01', 1, 'Marketing'),
    (8, 'Emma Clark', 42000.00, '2022-08-01', 2, 'Analyst'),
    (9, 'Olivia Taylor', 47000.00, '2022-09-01', 1, 'Developer'),
    (10, 'Liam Williams', 51000.00, '2022-10-01', 2, 'Manager');
```

Now, let's proceed with the operations:

3. Display employees whose name contains the letter 'e':

```sql
SELECT *
FROM employee
WHERE e_name LIKE '%e%';
```

4. Display different types of designations:

```sql
SELECT DISTINCT Designation
FROM employee;
```

5. Display the name and salary of employees whose location is Mumbai (assuming 'location' is a column in the table):

```sql
-- Assuming there is a 'location' column, update this query with the actual column name.
SELECT e_name, salary
FROM employee
WHERE designation = 'Manager';
```

6. Display the name and department of employees working in the Manager or Marketing department:

```sql
SELECT e_name, Designation
FROM employee
WHERE Designation IN ('Manager', 'Marketing');
```

7. Display the department name with more than one employee:

```sql
SELECT designation
FROM employee
```

```
GROUP BY designation
HAVING COUNT(*) > 1;
```

8. Rename the `employee` table as `emp1`:

```sql
ALTER TABLE employee RENAME TO emp1;
```

9. Add a new column `city` in the `employee` table:

```sql
ALTER TABLE emp1
ADD COLUMN city VARCHAR(50);
```

3)Consider the relation employee(emp_id,e_name,salary ,Date of Joining,Dapt_no,Designation) perform basic SQL operations.

1. Find department in which maximum employees work.
2. Display name, designation and department no of employees whose name starts with either ‘A’ or ‘P’.
3. Display max. salary from department 2 and min. salary from department 4.
4. Display employee data where salary is less than average salary from department 3.
5. Display employees who were hired earliest or latest.
6. Display name and department no of employees who are manager, market analysts. Use prediactes
7. List employees hired in August.
8. List employees who are hired after 31/12/2006.
9. Find average annual salary per department

To perform these SQL operations with example values inserted, you can use the following SQL queries with sample data:

Let's assume a simplified "employee" table with the following columns and example data:

```sql
CREATE TABLE employee (
    emp_id INT,
    e_name VARCHAR(255),
    salary DECIMAL(10, 2),
    Date_of_Joining DATE,
    Dapt_no INT,
    Designation VARCHAR(255)
);

-- Insert sample data
INSERT INTO employee (emp_id, e_name, salary, Date_of_Joining, Dapt_no, Designation)
VALUES
    (1, 'Alice', 55000.00, '2023-01-15', 1, 'Manager'),
    (2, 'Bob', 60000.00, '2023-02-10', 2, 'Developer'),
    (3, 'Charlie', 52000.00, '2023-03-20', 1, 'Analyst'),
    (4, 'David', 59000.00, '2023-04-05', 3, 'Market Analyst'),
    (5, 'Eve', 54000.00, '2023-05-12', 2, 'Developer'),
    (6, 'Frank', 58000.00, '2023-06-30', 3, 'Manager'),
    (7, 'Grace', 53000.00, '2023-07-14', 4, 'Analyst'),
    (8, 'Hannah', 61000.00, '2023-08-18', 4, 'Manager');
```

Now, you can perform the SQL operations:

1. Find the department in which the maximum employees work:

```sql
SELECT Dapt_no, COUNT(emp_id) AS employee_count
FROM employee
GROUP BY Dapt_no
ORDER BY employee_count DESC
LIMIT 1;
```

2. Display the name, designation, and department no of employees whose name starts with either 'A' or 'P':

```sql
SELECT e_name, Designation, Dapt_no
FROM employee
WHERE e_name LIKE 'A%' OR e_name LIKE 'P%';
```

3. Display the maximum salary from department 2 and the minimum salary from department 4:

```sql
SELECT MAX(salary) AS max_salary_dept_2, MIN(salary) AS min_salary_dept_4
FROM employee
WHERE Dapt_no = 2 OR Dapt_no = 4;
```

4. Display employee data where the salary is less than the average salary from department 3:

```sql
SELECT *
FROM employee
WHERE salary < (SELECT AVG(salary) FROM employee WHERE Dapt_no = 3);
```

5. Display employees who were hired earliest or latest:

```sql
SELECT *
FROM employee
WHERE Date_of_Joining = (SELECT MIN(Date_of_Joining) FROM employee) OR
    Date_of_Joining = (SELECT MAX(Date_of_Joining) FROM employee);
```

6. Display the name and department no of employees who are managers or market analysts:

```sql
SELECT e_name, Dapt_no
FROM employee
WHERE Designation IN ('Manager', 'Market Analyst');
```

7. List employees hired in August:

```sql
```

~~~
SELECT *
FROM employee
WHERE MONTH(Date_of_Joining) = 8;
```
~~~

8. List employees who were hired after 31/12/2006:

~~~
```sql
SELECT *
FROM employee
WHERE Date_of_Joining > '2006-12-31';
```
~~~

9. Find the average annual salary per department:

~~~
```sql
SELECT Dapt_no, AVG(salary) AS average_salary
FROM employee
GROUP BY Dapt_no;
```
~~~

These queries should work with the sample data provided. You can adjust the sample data and table structure to match your specific scenario..

4)Consider two tables Customer(c_id, c_name , email , city , pincode)Order(order_id , date , amount , cust_id.

1. Create both the tables with primary key and foreign key constraints.
2. insert 10 records each.
3. Find all orders placed by customers with cust_id 2
4. Find list of customers who placed their order and details of order

5. List of customers who haven’t placed order
6. List all orders and append to customer table
7. Display all records
8. Display customer that are from same city8

Sure, here are the updated queries with the renamed "CustomerOrder" table:

1. Create both the tables with primary key and foreign key constraints:

```sql
CREATE TABLE Customer (
    c_id INT PRIMARY KEY,
    c_name VARCHAR(255),
    email VARCHAR(255),
    city VARCHAR(255),
    pincode INT
);

CREATE TABLE CustomerOrder (
    order_id INT PRIMARY KEY,
    date DATE,
    amount DECIMAL(10, 2),
    cust_id INT,
    FOREIGN KEY (cust_id) REFERENCES Customer(c_id)
);
```

2. Insert 10 records each (You can replace the values with your own data):

```sql
-- Insert 10 customers
INSERT INTO Customer (c_id, c_name, email, city, pincode)
VALUES
    (1, 'Customer 1', 'customer1@email.com', 'City A', 12345),
    (2, 'Customer 2', 'customer2@email.com', 'City B', 54321),
    -- Add 8 more customers...

-- Insert 10 orders
INSERT INTO CustomerOrder (order_id, date, amount, cust_id)
VALUES
    (1, '2023-01-01', 100.00, 1),
    (2, '2023-01-02', 200.00, 2),
    -- Add 8 more orders...
```

3. Find all orders placed by customers with cust_id 2:

```sql
SELECT * FROM CustomerOrder WHERE cust_id = 2;
```

4. Find the list of customers who placed their order and details of the order:

```sql
SELECT c.*, o.order_id, o.date, o.amount
FROM Customer c
INNER JOIN CustomerOrder o ON c.c_id = o.cust_id;
```

5. List of customers who haven't placed an order:

```sql
SELECT c.*
FROM Customer c
LEFT JOIN CustomerOrder o ON c.c_id = o.cust_id
WHERE o.cust_id IS NULL;
```

6. List all orders and append to the customer table:

```sql
SELECT c.*, o.order_id, o.date, o.amount
FROM Customer c
LEFT JOIN CustomerOrder o ON c.cust_id = c.c_id;
```

7. Display all records from both tables:

```sql
-- All records from the Customer table
SELECT * FROM Customer;

-- All records from the CustomerOrder table
SELECT * FROM CustomerOrder;
```

8. Display customers that are from the same city:

To display customers from the same city, you can use a self-join on the Customer table. For example, to find customers from the same city as Customer 2:

```sql
SELECT c1.*
FROM Customer c1
JOIN Customer c2 ON c1.city = c2.city
WHERE c2.c_id = 2; -- Assuming you want customers from the same city as Customer 2 (change the ID as needed).
```

5) Consider tables Borrower (RollNo, Name, DateofIssue, NameofBook, Status) and
Fine (Roll_no,Date,Amt). Status is either Issued or Returned.
1. Create both the tables with primary key.
2. Insert 10 records each.
3. Find count of books with Issued status.
4. Display all records.
5. Display RollNo whose date of issue is same.

SQL Queries :

1. Create both tables with primary keys:

```sql
-- Borrower table
CREATE TABLE Borrower (
    RollNo INT PRIMARY KEY,
    Name VARCHAR(50),
    DateofIssue DATE,
    NameofBook VARCHAR(50),
    Status VARCHAR(10)
);

-- Fine table
CREATE TABLE Fine (
    Roll_no INT,
    Date DATE,
    Amt DECIMAL(10, 2),
    PRIMARY KEY (Roll_no, Date),
    FOREIGN KEY (Roll_no) REFERENCES Borrower(RollNo)
);
```

This creates two tables: `Borrower` and `Fine`, with their respective primary keys.

2. Insert 10 records each (sample data, you can replace with actual data):

For Borrower table:

```sql
INSERT INTO Borrower (RollNo, Name, DateofIssue, NameofBook, Status)
VALUES
    (1, 'John Doe', '2022-01-05', 'Book A', 'Issued'),
    (2, 'Jane Smith', '2022-01-10', 'Book B', 'Issued'),
            (3, 'sanket', '2022-01-10', 'Book c', 'Issued');
    -- Add more records as needed
;
```

For Fine table:

```sql
INSERT INTO Fine (Roll_no, Date, Amt)
VALUES
    (1, '2022-01-15', 10.00),
    (2, '2022-01-12', 8.00),
    -- Add more records as needed
```

```
;
```

3. Find the count of books with Issued status:

```sql
SELECT COUNT(*)
FROM Borrower
WHERE Status = 'Issued';
```

4. Display all records:

For Borrower table:

```sql
SELECT * FROM Borrower;
```

For Fine table:

```sql
SELECT * FROM Fine;
```

5. Display RollNo whose date of issue is the same:

```sql
SELECT b1.RollNo
FROM Borrower b1
INNER JOIN (
    SELECT DateofIssue
    FROM Borrower
    GROUP BY DateofIssue
    HAVING COUNT(*) > 1
) b2 ON b1.DateofIssue = b2.DateofIssue;
```

Please note that these queries assume that you have the necessary permissions to create tables and insert data. Additionally, replace the sample data with actual data as per your requirements.

6) Consider student (roll_no, name, marks, class) table. Column roll_no is primary key. Perform any 3 DLL and any 3 DML operations on the table.

```
-- DDL Operations

-- 1. Create the student table

CREATE TABLE student (

    roll_no INTEGER PRIMARY KEY,

    name TEXT,

    marks INTEGER,

    class TEXT

);


-- 2. Add a new column to the table

ALTER TABLE student

ADD COLUMN gender TEXT;


-- 3. Delete the "gender" column from the table

ALTER TABLE student

DROP COLUMN gender;


-- DML Operations

-- 1. Insert records into the student table

INSERT INTO student (roll_no, name, marks, class) VALUES

    (1, 'Alice', 95, 'A'),

    (2, 'Bob', 88, 'B'),

    (3, 'Charlie', 75, 'C'),

    (4, 'David', 92, 'A');
```

```
-- 2. Update a student's marks

UPDATE student

SET marks = 85

WHERE roll_no = 2;


-- 3. Delete a student's record

DELETE FROM student

WHERE roll_no= 1;
```

7) Write a SQL statement to create a table job_history including columns employee_id, start_date, end_date, job_id and department_id and make sure that, the employee_id column does not contain any duplicate value at the time of insertion and the foreign key column job_id contain only those values which are exists in the jobs table. Consider table Job (job_id,job_title.min_sal,max_sal)

Certainly! Here is the SQL statement to create the `job_history` table with the specified constraints:

```
CREATE TABLE jobs (
   job_id VARCHAR(10) PRIMARY KEY,
   job_title VARCHAR(50),
   min_sal DECIMAL(10, 2),
   max_sal DECIMAL(10, 2)
);

-- Insert sample data into the jobs table
INSERT INTO jobs (job_id, job_title, min_sal, max_sal) VALUES
   ('JOB001', 'Software Developer', 50000.00, 90000.00),
   ('JOB002', 'Database Administrator', 55000.00, 95000.00),
   ('JOB003', 'Business Analyst', 60000.00, 100000.00),
   ('JOB004', 'Project Manager', 70000.00, 120000.00),
   ('JOB005', 'Network Engineer', 55000.00, 95000.00);

CREATE TABLE job_history (
   employee_id INT,
   start_date DATE,
   end_date DATE,
   job_id VARCHAR(10),
   department_id INT,
   PRIMARY KEY (employee_id, start_date),
   FOREIGN KEY (job_id) REFERENCES jobs (job_id)
);
```

Explanation:

- `employee_id INT`: This column stores the unique identifier for each employee.

- `start_date DATE`: This column stores the start date of the job history record.

- `end_date DATE`: This column stores the end date of the job history record.

- `job_id VARCHAR(10)`: This column stores the job ID, referencing the `job_id` in the `jobs` table.

- `department_id INT`: This column stores the department ID.

- `PRIMARY KEY (employee_id, start_date)`: This makes sure that the combination of `employee_id` and `start_date` is unique.

- `FOREIGN KEY (employee_id) REFERENCES employees(employee_id)`: This establishes a foreign key relationship with the `employees` table, ensuring that `employee_id` in `job_history` refers to a valid `employee_id` in the `employees` table.

- `FOREIGN KEY (job_id) REFERENCES jobs(job_id)`: This ensures that `job_id` in `job_history` refers to a valid `job_id` in the `jobs` table.

Make sure you have the `employees` and `jobs` tables created with their respective columns (`employee_id`, `job_id`, etc.) before creating the `job_history` table, as there are foreign key references to these tables.

8) For the given relation schema: employee(employee-name, street, city)
works (employee-name, company-name, salary)
company (company-name, city)
manages (employee-name, manager-name)

Give an expression in SQL for each of the following queries:
a) Find the names, street address, and cities of residence for all employees who work for same company and earn more than $10,000.
b) Find the names of all employees in the database who live in the same cities as the companies for which they work.
c) Find the names of all employees who earn more than the average salary of all employees of their company. Assume that all people work for at most one company.

1. **Creating Tables and Inserting Sample Data:**

```sql
-- Create employee table
CREATE TABLE employee (
    employee_name VARCHAR(50),
    street VARCHAR(100),
    city VARCHAR(50)
);

-- Create works table
CREATE TABLE works (
    employee_name VARCHAR(50),
    company_name VARCHAR(50),
    salary DECIMAL(10, 2)
);

-- Create company table
CREATE TABLE company (
    company_name VARCHAR(50),
    city VARCHAR(50)
);

-- Create manages table
CREATE TABLE manages (
    employee_name VARCHAR(50),
    manager_name VARCHAR(50)
);
```

```
-- Insert sample data into employee table
INSERT INTO employee (employee_name, street, city)
VALUES
    ('John Doe', '123 Main Street', 'New York'),
    ('Jane Smith', '456 Elm Street', 'Los Angeles'),
    ('Bob Johnson', '789 Oak Street', 'Chicago');

-- Insert sample data into works table
INSERT INTO works (employee_name, company_name, salary)
VALUES
    ('John Doe', 'Company A', 12000.00),
    ('Jane Smith', 'Company B', 15000.00),
    ('Bob Johnson', 'Company A', 10000.00);

-- Insert sample data into company table
INSERT INTO company (company_name, city)
VALUES
    ('Company A', 'New York'),
    ('Company B', 'Los Angeles'),
    ('Company C', 'Chicago');

-- Insert sample data into manages table
INSERT INTO manages (employee_name, manager_name)
VALUES
    ('John Doe', 'Jane Smith'),
    ('Jane Smith', 'Bob Johnson');
```

Now that we have the tables and some sample data, let's proceed with the SQL expressions for the queries:

a) **Find the names, street address, and cities of residence for all employees who work for the same company and earn more than $10,000:**

```sql
SELECT e.employee_name, e.street, e.city
FROM employee e
JOIN works w ON e.employee_name = w.employee_name
WHERE w.salary > 10000
AND w.company_name IN (
    SELECT DISTINCT w1.company_name
    FROM works w1
    JOIN works w2 ON w1.company_name = w2.company_name
    WHERE w1.employee_name <> w2.employee_name
);
```

b) **Find the names of all employees in the database who live in the same cities as the companies for which they work:**

```sql
SELECT DISTINCT e.employee_name
FROM employee e
JOIN works w ON e.employee_name = w.employee_name
JOIN company c ON w.company_name = c.company_name
WHERE e.city = c.city;
```

c) **Find the names of all employees who earn more than the average salary of all employees of their company. Assume that all people work for at most one company:**

```sql
SELECT e.employee_name
FROM employee e
JOIN works w ON e.employee_name = w.employee_name
WHERE w.salary > (
    SELECT AVG(w2.salary)
    FROM works w2
    WHERE w2.company_name = w.company_name
);
```

These SQL expressions should work with the provided tables and sample data.

9) For the given relation schema: employee(employee-name, street, city)
works (employee-name, company-name, salary)
company (company-name, city)
manages (employee-name, manager-name)
Give an expression in SQL for each of the following queries:

a) Find the name of the company that has the smallest payroll.
b) Find the names of all employees in the database who live in the same cities and on the same streets as do their managers.

Certainly! Let's first create the tables and then add some sample values. After that, we'll provide the SQL expressions for the queries.

Creating Tables:

```sql
-- Create employee table
CREATE TABLE employee (
    employee_name VARCHAR(50),
    street VARCHAR(100),
    city VARCHAR(50)
);

-- Create works table
CREATE TABLE works (
    employee_name VARCHAR(50),
    company_name VARCHAR(50),
    salary DECIMAL(10, 2)
);

-- Create company table
CREATE TABLE company (
    company_name VARCHAR(50),
    city VARCHAR(50)
);

-- Create manages table
CREATE TABLE manages (
    employee_name VARCHAR(50),
    manager_name VARCHAR(50)
);
```

Adding Sample Values:

```sql
-- Insert sample data into employee table
INSERT INTO employee (employee_name, street, city)
VALUES
    ('John Doe', '123 Main Street', 'New York'),
    ('Jane Smith', '456 Elm Street', 'Los Angeles'),
    ('Bob Johnson', '789 Oak Street', 'Chicago');

-- Insert sample data into works table
INSERT INTO works (employee_name, company_name, salary)
VALUES
    ('John Doe', 'Company A', 12000.00),
    ('Jane Smith', 'Company B', 15000.00),
    ('Bob Johnson', 'Company A', 10000.00);

-- Insert sample data into company table
INSERT INTO company (company_name, city)
VALUES
    ('Company A', 'New York'),
    ('Company B', 'Los Angeles'),
    ('Company C', 'Chicago');

-- Insert sample data into manages table
INSERT INTO manages (employee_name, manager_name)
VALUES
    ('John Doe', 'Jane Smith'),
    ('Jane Smith', 'Bob Johnson');
```

Queries:

a) **Find the name of the company that has the smallest payroll:**

```sql
SELECT w.company_name
FROM works w
GROUP BY w.company_name
HAVING SUM(w.salary) = (
    SELECT MIN(total_salary)
    FROM (
        SELECT SUM(salary) as total_salary
        FROM works
        GROUP BY company_name
    ) as min_salaries
);
```

b) **Find the names of all employees who live in the same cities and on the same streets as do their managers:**

```sql
SELECT e.employee_name
FROM employee e
JOIN manages m ON e.employee_name = m.employee_name
JOIN employee manager ON m.manager_name = manager.employee_name
WHERE e.street = manager.street AND e.city = manager.city;
```

These SQL statements should work with the provided schema and sample data.

MONGODB SECTION :

10) Implement CRUD operations. SAVE method. Use following Collection. Perform Map Reduce to count quantity of each item.

Item: Item ID, Item quantity, price, brand, discount

1. Display the count of item brand wise.
2. Dsiplay item with minimum price.
3. Display maximum discount given for item.

```
db.createCollection("Item");

db.Item.insertMany([
 { ItemID: 1, ItemQuantity: 10, Price: 20, Brand: "Brand1", Discount: 5 },
 { ItemID: 2, ItemQuantity: 5, Price: 15, Brand: "Brand2", Discount: 10 },
 { ItemID: 3, ItemQuantity: 8, Price: 25, Brand: "Brand1", Discount: 7 },
 { ItemID: 4, ItemQuantity: 12, Price: 30, Brand: "Brand2", Discount: 12 },
 { ItemID: 5, ItemQuantity: 15, Price: 22, Brand: "Brand1", Discount: 8 }
 ]);


var mapfunction=function(){emit(this.Brand,this.ItemQuantity)};

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

db.Item.mapReduce( mapfunction,reducefunction,{out:"Brand_quantity"});

db.Brand_quantity.find();
```

1. Display the count of item brand wise.

```
db.Item.aggregate([{$group:{_id:"$Brand",count:{$sum:1}}}]);
```

2. Dsiplay item with minimum price.

```
db.Item.find().sort({Price:1}).limit(1);
```

3. Display maximum discount given for item.

```
db.Item.find().sort({Discount:-1}).limit(1);
```

11) Implement CRUD operations. SAVE method. Use following Collection.

Item: Item ID, Item quantity, price, brand, discount

1. Display the count of item brand wise.
2. Dsiplay item with minimum price.
3. Display maximum discount given for item.

.

12) Implement CRUD operations. SAVE method. Use following Collection.

Item: Item ID, Item quantity, price, brand, discount
1. Display the count of item brand wise.
2. Dsiplay item with minimum price.
3. Display maximum discount given for item.

13) Implement Map reduces operation for counting the marks of students.

Use: student (roll_no, name marks, class)

Expected output: student name or roll no and total marks.

```
db.createCollection('students');

db.student.insertMany([

     { roll_no: 1, name: "Alice", marks: [85, 90, 78], class: "10th" },

     { roll_no: 2, name: "Bob", marks: [75, 80, 92], class: "10th" },

     { roll_no: 3, name: "Charlie", marks: [90, 88, 86], class: "10th" },

      { roll_no: 4, name: "David", marks: [92, 95, 89], class: "10th" },

     { roll_no: 5, name: "Eve", marks: [78, 85, 88], class: "10th" }

 ]);


var mapFunction=function(){

 emit(this.name,Array.sum(this.marks));

 };


var reduceFunction=function(key,values){

 return Array.sum(values);

 };


db.student.mapReduce(mapFunction,reduceFunction,{out:"total_marks"});


db.total_marks.find();
```

14) Implement Map reduces operation for displaying persons with same profession.

Use: person (person_id, name, addr, profession)

```
db.createCollection("person")

db.person.insertMany([{ person_id: 1, name: "Alice", addr: "123 Main St", profession:
"Engineer" },

    { person_id: 2, name: "Bob", addr: "456 Elm St", profession: "Doctor" },

    { person_id: 3, name: "Charlie", addr: "789 Oak St", profession: "Engineer" },

    { person_id: 4, name: "David", addr: "101 Pine St", profession: "Lawyer" },

    { person_id: 5, name: "Eve", addr: "202 Maple St", profession: "Doctor" }

]);


var mapFunction = function () {

    emit(this.profession, this.name);

};


var reduceFunction = function () {

    emit(this.profession, this.name);

};

db.person.mapReduce(mapFunction,reduceFunction,{out:"person_by_profession"});


db.person_by_profession.find();
```

15) Perform CRUD operation in mongo db –

Use : person(person_id, name, addr, profession)

1.Create Collection.

2.Inserting data in collection.

3.Reading data of collection.

4.Updating data of collection.

5.Deleting data from collection.

1.Create Collection.

```
db.createCollection("person");
```

2.Inserting data in collection.

```
db.person.insertOne({
   person_id: 1,
   name: "Alice",
   addr: "123 Main St",
   profession: "Engineer"
})

db.person.insertMany([
   {
       person_id: 2,
       name: "Bob",
       addr: "456 Elm St",
       profession: "Doctor"
   },
   {
       person_id: 3,
       name: "Charlie",
       addr: "789 Oak St",
       profession: "Engineer"
   }
]);
```

3.Reading data of collection.

```
db.person.find();
db.person.findOne({ person_id: 1 });
```

4.Updating data of collection.

```
db.person.updateOne(
    { person_id: 1 },
    { $set: { profession: "Software Developer" } }
);
```

5.Deleting data from collection.

```
db.person.deleteOne({ person_id: 2 });
```

16) Perform CRUD operation and Aggregation in mongo db

employee(emp_id,e_name,salary ,Date of Joining,Dapt_no,Designation)

1. Display the count of employee department wise.

2. Dsiplay the average salary of employee in sales department.

3. Dsiplay minimum salary to employees joins in June 2016

4. Display maximum salary given to employee in production department.

5. Display record of first and last employee department wise.

```
db.createCollection("employee");


db.employee.insertMany([
    { emp_id: 1, e_name: "Alice", salary: 60000, Date_of_Joining: "2016-05-15", Dept_no: 101,
Designation: "Manager" },
    { emp_id: 2, e_name: "Bob", salary: 55000, Date_of_Joining: "2017-03-20", Dept_no: 102,
Designation: "Sales Associate" },
    { emp_id: 3, e_name: "Charlie", salary: 75000, Date_of_Joining: "2016-06-10", Dept_no: 101,
Designation: "Engineer" },
    { emp_id: 4, e_name: "David", salary: 80000, Date_of_Joining: "2018-01-10", Dept_no: 103,
Designation: "Analyst" },
    { emp_id: 5, e_name: "Eve", salary: 90000, Date_of_Joining: "2016-06-01", Dept_no: 102,
Designation: "Sales Manager" }
]);
```

1. Display the count of employee department wise.

```
db.employee.aggregate([
    {
        $group: {
            _id: "$Dept_no",
            count: { $sum: 1 }
        }
    }
]);
```

2. Dsiplay the average salary of employee in sales department.

```
db.employee.aggregate([
    {
        $match: { Dept_no: 102 }
    },
    {
        $group: {
            _id: null,
            averageSalary: { $avg: "$salary" }
        }
    }
]);
```

3. Dsiplay minimum salary to employees joins in June 2016

```
db.employee.find({
    Date_of_Joining: {
        $gte: "2016-06-01",
        $lt: "2016-07-01"
    }
}).sort({ salary: 1 }).limit(1);
```

4. Display maximum salary given to employee in production department.

```
db.employee.find({
    Dept_no: 103
}).sort({ salary: -1 }).limit(1);
```

5. Display record of first and last employee department wise.

```
db.employee.aggregate([
    {
        $sort: { Dept_no: 1, emp_id: 1 }
    },
    {
        $group: {
            _id: "$Dept_no",
            firstEmployee: { $first: "$$ROOT" },
            lastEmployee: { $last: "$$ROOT" }
        }
    }
]);
```

17) Consider student (roll_no, name ,marks, class) table. Perform add update and delete operation on same table through java program. Write menu driven program.

PL/SQL SECTION :

18) Implement Stored Procedure namely proc_Grade for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and900 category is first class, if marks 899 and 825 category is Higher Second Class. Write a PL/SQL block for using procedure created with above requirement. Stud_Marks(name, total_marks) Result(Roll,Name, Class).

```
1)-- Create a table to store the student marks
CREATE TABLE Stud_Marks (
    Roll NUMBER PRIMARY KEY,
    Name VARCHAR2(50),
    Total_Marks NUMBER
);
```

2)-- Create a table to store the results

```
CREATE TABLE Result (
    Roll NUMBER PRIMARY KEY,
    Name VARCHAR2(50),
    Class VARCHAR2(50)
);
```

3)-- Create the stored procedure proc_Grade

```
CREATE OR REPLACE PROCEDURE proc_Grade AS
BEGIN
    FOR student IN (SELECT Roll, Name, Total_Marks FROM Stud_Marks) LOOP
        IF student.Total_Marks >= 990 AND student.Total_Marks <= 1500 THEN
            INSERT INTO Result (Roll, Name, Class) VALUES (student.Roll, student.Name, 
'Distinction');
        ELSIF student.Total_Marks >= 900 AND student.Total_Marks <= 989 THEN
            INSERT INTO Result (Roll, Name, Class) VALUES (student.Roll, student.Name, 'First 
Class');
        ELSIF student.Total_Marks >= 825 AND student.Total_Marks <= 899 THEN
            INSERT INTO Result (Roll, Name, Class) VALUES (student.Roll, student.Name, 
'Higher Second Class');
        END IF;
    END LOOP;
    COMMIT;
END;
/
```

4)-- Insert sample data into the Stud_Marks table

```
INSERT INTO Stud_Marks (Roll, Name, Total_Marks) VALUES (1, 'John', 1450);
INSERT INTO Stud_Marks (Roll, Name, Total_Marks) VALUES (2, 'Alice', 990);
INSERT INTO Stud_Marks (Roll, Name, Total_Marks) VALUES (3, 'Bob', 899);
INSERT INTO Stud_Marks (Roll, Name, Total_Marks) VALUES (4, 'Eve', 1000);
INSERT INTO Stud_Marks (Roll, Name, Total_Marks) VALUES (5, 'Charlie', 850);
```

5)-- Execute the stored procedure to categorize the students

```
BEGIN
proc_Grade;
END;
/
SELECT * FROM Result;
```

19) Write a database trigger on customer(cust_id, c_name, addr) 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 cust_Audit table.

```
CREATE TABLE customer2 (
    cust_id NUMBER PRIMARY KEY,
    c_name VARCHAR2(50),
    addr VARCHAR2(100)
);

CREATE TABLE cust_Audit2 (
    cust_id NUMBER,
    c_name VARCHAR2(50),
    addr VARCHAR2(100)
);
INSERT INTO customer2 (cust_id, c_name, addr) VALUES (1, 'John Doe', '123 Main St');
INSERT INTO customer2 (cust_id, c_name, addr) VALUES (2, 'Alice Smith', '456 Elm St');
INSERT INTO customer2 (cust_id, c_name, addr) VALUES (3, 'Bob Johnson', '789 Oak St');

CREATE OR REPLACE TRIGGER customer2_audit_trigger
BEFORE DELETE OR UPDATE ON customer2
FOR EACH ROW
BEGIN
    IF UPDATING THEN
        -- Store the old values in cust_Audit2 before updating
        INSERT INTO cust_Audit2 (cust_id, c_name, addr)
        VALUES (:OLD.cust_id, :OLD.c_name, :OLD.addr);
    ELSIF DELETING THEN
        -- Store the old values in cust_Audit2 before deleting
        INSERT INTO cust_Audit2 (cust_id, c_name, addr)
        VALUES (:OLD.cust_id, :OLD.c_name, :OLD.addr);
    END IF;
END;
/

UPDATE customer2
SET addr = '456 New Address'
WHERE cust_id = 1;

DELETE FROM customer2
WHERE cust_id = 2;
```

```
select * from customer2;

select * from cust_Audit2;
```

20) Implement a database trigger on client_master(c_id, c_name, acc_no) table. The System should keep track of the records that are being updated or inserted. The old value of updated or deleted records should be added in client_Audit table.

```
CREATE TABLE client_master (
    c_id NUMBER PRIMARY KEY,
    c_name VARCHAR2(50),
    acc_no NUMBER
);

CREATE TABLE client_Audit (
    c_id NUMBER,
    c_name VARCHAR2(50),
    acc_no NUMBER
);

CREATE OR REPLACE TRIGGER client_master_audit_trigger
AFTER INSERT OR UPDATE OR DELETE ON client_master
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        -- Store the inserted values in client_Audit
        INSERT INTO client_Audit (c_id, c_name, acc_no)
        VALUES (:NEW.c_id, :NEW.c_name, :NEW.acc_no);
    ELSIF UPDATING THEN
        -- Store the old values in client_Audit before updating
        INSERT INTO client_Audit (c_id, c_name, acc_no)
        VALUES (:OLD.c_id, :OLD.c_name, :OLD.acc_no);
    ELSIF DELETING THEN
        -- Store the old values in client_Audit before deleting
        INSERT INTO client_Audit (c_id, c_name, acc_no)
        VALUES (:OLD.c_id, :OLD.c_name, :OLD.acc_no);
    END IF;
END;
/

INSERT INTO client_master (c_id, c_name, acc_no)
VALUES (1, 'John Doe', 12345);

-- Insert another client
INSERT INTO client_master (c_id, c_name, acc_no)
VALUES (2, 'Alice Smith', 67890);

UPDATE client_master
SET acc_no = 54321
WHERE c_id = 1;

DELETE FROM client_master
WHERE c_id = 2;

select * from client_master;
select * from client_Audit;
```

21) Implement a PL/SQL block of code using explicit 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.

```
-- Create N_RollCall table
CREATE TABLE N_RollCall (
    rollcall_id NUMBER PRIMARY KEY,
    rollcall_date DATE,
    student_id NUMBER,
    status VARCHAR2(20)
);

-- Create O_RollCall table
CREATE TABLE O_RollCall (
    rollcall_id NUMBER PRIMARY KEY,
    rollcall_date DATE,
    student_id NUMBER,
    status VARCHAR2(20)
);



INSERT INTO N_RollCall (rollcall_id, rollcall_date, student_id, status)
VALUES (1, TO_DATE('2023-11-01', 'YYYY-MM-DD'), 101, 'Present');

1 row created.

INSERT INTO N_RollCall (rollcall_id, rollcall_date, student_id, status)
```

```
VALUES (2, TO_DATE('2023-11-01', 'YYYY-MM-DD'), 102, 'Absent');

1 row created.

INSERT INTO O_RollCall (rollcall_id, rollcall_date, student_id, status)
VALUES (3, TO_DATE('2023-11-01', 'YYYY-MM-DD'), 103, 'Present');

1 row created.

INSERT INTO O_RollCall (rollcall_id, rollcall_date, student_id, status)
VALUES (4, TO_DATE('2023-11-01', 'YYYY-MM-DD'), 104, 'Absent');



INSERT INTO O_RollCall (rollcall_id, rollcall_date, student_id, status)
VALUES (1, TO_DATE('2023-11-01', 'YYYY-MM-DD'), 101, 'Present');

1 row created.

INSERT INTO O_RollCall (rollcall_id, rollcall_date, student_id, status)
VALUES (2, TO_DATE('2023-11-01', 'YYYY-MM-DD'), 102, 'Absent');

 DECLARE
       CURSOR O_rollcall_cursor IS
           SELECT *
           FROM O_RollCall;

       O_rollcall_rec O_RollCall%ROWTYPE;
       existing_count NUMBER;
```

```
BEGIN
    FOR O_rollcall_rec IN O_rollcall_cursor LOOP
        -- Check if the data already exists in O_RollCall
        -- If not, insert it into O_RollCall
        SELECT COUNT(*)
        INTO existing_count
        FROM N_RollCall
        WHERE rollcall_id = O_rollcall_rec.rollcall_id;

         IF existing_count = 0 THEN
                      INSERT INTO N_RollCall (rollcall_id, rollcall_date, student_id, status)
                      VALUES(O_rollcall_rec.rollcall_id,O_rollcall_rec.rollcall_date,
                     O_rollcall_rec.student_id, O_rollcall_rec.status);
          END IF;
         END LOOP;

    COMMIT;
END;
/
```

```
select * O_RollCall;

select * N_RollCall;
```

22) Write a PL/SQL block of code for the following requirements:- Schema: Borrower(Rollin, Name, DateofIssue, NameofBook, Status) 2. 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 condition of fine is true, then details will be stored into fine table.

```
CREATE TABLE borrower2(roll_no NUMBER , name VARCHAR2(25), dateofissue

DATE, name_of_book VARCHAR2(25), status VARCHAR2(20));


INSERT INTO borrower2 VALUES(45,'ASHUTOSH',TO_DATE('01-08-2022','DD-MMYYYY'),'HARRY POTTER','PENDING');


INSERT INTO borrower2 VALUES(46,'ARYAN',TO_DATE('15-08-2022','DD-MMYYYY'),'DARK MATTER','PENDING');

1 row created.

INSERT INTO borrower2 VALUES(47,'ROHAN',TO_DATE('24-08-2022','DD-MMYYYY'),'SILENT HILL','PENDING');

1 row created.

INSERT INTO borrower2 VALUES(48,'SANKET',TO_DATE('26-08-2022','DD-MMYYYY'),'GOD OF WAR','PENDING');

1 row created.

INSERT INTO borrower2 VALUES(49,'SARTHAK',TO_DATE('09-09-2022','DD-MMYYYY'),'SPIDER-MAN','PENDING');

1 row created.


CREATE TABLE fine2 (

  roll_no NUMBER,

  return_date DATE,

  fine NUMBER

  );
```

```
DECLARE
  i_roll_no NUMBER;
  name_of_book VARCHAR2(25);
  no_of_days NUMBER;
  return_date DATE := TO_DATE(SYSDATE,'DD-MM-YYYY');
  temp NUMBER;
  doi DATE;
  fine NUMBER;
  BEGIN
 i_roll_no := &i_roll_no;
 name_of_book := '&nameofbook';
 dbms_output.put_line(return_date);
 SELECT to_date(borrower2.dateofissue,'DD-MM-YYYY') INTO doi FROM
borrower2 WHERE
 borrower2.roll_no = i_roll_no AND borrower2.name_of_book =
name_of_book;
 no_of_days := return_date-doi;
 dbms_output.put_line(no_of_days);
 IF (no_of_days >15 AND no_of_days <=30) THEN
 fine := 5*no_of_days;
 ELSIF (no_of_days>30 ) THEN
 temp := no_of_days-30;
 fine := 150 + temp*50;
 END IF;
 dbms_output.put_line(fine);

 INSERT INTO fine2 VALUES(i_roll_no,return_date,fine);

 UPDATE borrower2 SET status = 'RETURNED' WHERE borrower2.roll_no =

i_roll_no;

 END;

 /
```

```
output-

Enter value for i_roll_no: 46

Enter value for nameofbook: DARK MATTER

02-OCT-23

413

19300

PL/SQL procedure successfully completed.
```

```
SQL> select * from BORROWER;


SQL> select * from FINE;
```

23) Implement Basic SQL queries.

1. Create table employee.
2. Insert 10 records in table.
3. Create a view emp_vl of table employee which has emp_id , name and dept-attributes.
4. Display name and department of employee working in Manager or Marketing department

5. Display employees who were hired earliest or latest.
6. Display name and department no of employees who are manager, market analysts. Use Predicates

List employees hired in August.

List employees who are hired after 31/12/2006.

Here are the basic SQL queries to implement the tasks you mentioned:

1. Create a table named `employee`:

```sql
CREATE TABLE employee (
    emp_id INT PRIMARY KEY,
    name VARCHAR(50),
    department VARCHAR(50),
    hire_date DATE
);
```

2. Insert 10 records into the `employee` table:

```sql
INSERT INTO employee (emp_id, name, department, hire_date)
VALUES
    (1, 'John Doe', 'Manager', '2022-01-15'),
    (2, 'Jane Smith', 'Marketing', '2021-11-10'),
    (3, 'Bob Johnson', 'HR', '2022-03-05'),
    (4, 'Alice Brown', 'Marketing', '2020-09-20'),
    (5, 'Charlie Lee', 'Manager', '2019-12-18'),
    (6, 'Eve White', 'Sales', '2021-04-25'),
    (7, 'Frank Black', 'Marketing', '2022-08-30'),
    (8, 'Grace Davis', 'Manager', '2018-07-12'),
    (9, 'Henry Green', 'Sales', '2020-06-02'),
    (10, 'Isabel Reed', 'HR', '2021-02-14');
```

3. Create a view named `emp_vw`:

```sql
CREATE VIEW emp_vw AS
SELECT emp_id, name, department
FROM employee;
```

4. Display the name and department of employees working in the Manager or Marketing department:

```sql
SELECT name, department
FROM employee
WHERE department IN ('Manager', 'Marketing');
```

5. Display employees who were hired earliest or latest:

To display employees hired earliest:

```sql
```

```
SELECT name, department, hire_date
FROM employee
WHERE hire_date = (SELECT MIN(hire_date) FROM employee);
```

To display employees hired latest:

```sql
SELECT name, department, hire_date
FROM employee
WHERE hire_date = (SELECT MAX(hire_date) FROM employee);
```

6. Display the name and department number of employees who are managers or market analysts:

```sql
SELECT name, department
FROM employee
WHERE department IN ('Manager', 'Marketing');
```

7. List employees hired in August:

```sql
SELECT name, department, hire_date
FROM employee
WHERE EXTRACT(MONTH FROM hire_date) = 8;
```

8. List employees who were hired after December 31, 2006:

```sql
SELECT name, department, hire_date
FROM employee
WHERE hire_date > '2006-12-31';
```

These SQL queries should help you achieve the tasks you mentioned. Please adapt the table structure and data as needed for your specific use case.

24)) Indexing and join: Consider the relation

employee (emp_id,e_name,salary ,Date of Joining,Dapt_no,Designation)

Customer(c_id, c_name , email , city , pincode)Order(order_id , date , amount , cust_id.

a. create empid as primary key and indices on table employee.

b. create user defined index on any column

c. create sequence using auo-increment.

d. truncate table.

e. find list of customers who placed order and details of their orders.

f. find info of customers and append order details to the table/

g. list down customers who haven't placed order.

To accomplish the tasks you mentioned, we'll work with the "employee," "Customer," and "Order" tables and perform various operations as described:

a. Create the "employee" table with "emp_id" as the primary key and create indices on the table. We'll also insert 10 values into each of the "employee," "Customer," and "Order" tables:

~~~
```sql
-- Create the employee table with the primary key and indices
CREATE TABLE employee (
    emp_id INT PRIMARY KEY,
    e_name VARCHAR(50),
    salary DECIMAL(10, 2),
    Date_of_Joining DATE,
    Dept_no INT,
    Designation VARCHAR(50)
);

-- Create an index on the salary column
CREATE INDEX idx_salary ON employee(salary);

-- Insert 10 values into the employee table
INSERT INTO employee (emp_id, e_name, salary, Date_of_Joining, Dept_no, Designation)
VALUES
    (1, 'John Doe', 60000.00, '2022-01-15', 101, 'Manager'),
    (2, 'Jane Smith', 55000.00, '2021-11-10', 102, 'Marketing');
    -- Add more records as needed.

-- Create the Customer table and insert 10 values
CREATE TABLE Customer (
    c_id INT PRIMARY KEY,
    c_name VARCHAR(50),
    email VARCHAR(100),
    city VARCHAR(50),
    pincode VARCHAR(10)
~~~

```
);

-- Insert 10 values into the Customer table
INSERT INTO Customer (c_id, c_name, email, city, pincode)
VALUES
    (1, 'Alice Brown', 'alice@example.com', 'New York', '10001'),
    (2, 'Bob Johnson', 'bob@example.com', 'Los Angeles', '90001');
    -- Add more records as needed.

-- Create the Order table and insert 10 values
CREATE TABLE cust_order (
    order_id INT PRIMARY KEY,
    date DATE,
    amount DECIMAL(10, 2),
    cust_id INT
);

-- Insert 10 values into the Order table
INSERT INTO cust_order (order_id, date, amount, cust_id)
VALUES
    (1, '2022-01-01', 100.00, 1),
    (2, '2022-01-02', 150.00, 2);
    -- Add more records as needed.
```

b. Create a user-defined index on any column, for example, we'll create an index on the "email" column in the "Customer" table:

```sql
```

CREATE INDEX idx_email ON Customer(email);
~~~

c. Create a sequence using auto-increment:

You can create a sequence for generating unique IDs. The specific SQL for creating a sequence may vary depending on the database system you are using. Here's an example using PostgreSQL:

```sql
ALTER TABLE Customer
MODIFY c_id INT AUTO_INCREMENT;```

d. Truncate a table. You can use the `TRUNCATE` statement to remove all rows from a table without deleting the table itself. For example:

```sql
TRUNCATE TABLE employee;
~~~

e. Find a list of customers who placed orders and details of their orders:

```sql
SELECT c.c_name, o.order_id, o.date, o.amount
FROM customer c
INNER JOIN cust_order o ON c.c_id = o.cust_id;
```

f. Find info of customers and append order details to the table:

You can't directly append order details to the Customer table because the Customer table and the Order table have a one-to-many relationship. However, you can retrieve the data together in a query, or you can create a new table to store the combined data if needed.

Here's how to retrieve the data in a query:

```sql
CREATE TABLE customer_order_info AS
SELECT c.c_name, c.email, c.city, c.pincode, o.order_id, o.date, o.amount
FROM customer c
LEFT JOIN cust_order o ON c.c_id = o.cust_id;
```

g. List down customers who haven't placed orders:

```sql
SELECT c.c_name, c.email
FROM customer c
LEFT JOIN cust_order o ON c.c_id = o.cust_id
WHERE o.cust_id IS NULL;
```

This query will return customers who haven't placed orders because we use a LEFT JOIN and check for cases where the order_id is NULL, indicating no matching order for the customer.

25) Implement aggregation and indexing with suitable example in mongodb.

```
db.createCollection("sales")
db.sales.insertMany([
    { product: "A", category: "Electronics", amount: 100 },
  { product: "B", category: "Books", amount: 50 },
   { product: "C", category: "Electronics", amount: 75 },
    { product: "D", category: "Books", amount: 40 },
    { product: "E", category: "Clothing", amount: 120 },
 ]);

db.sales.aggregate([{$group:{_id:"$category",totalSales:{$sum:"$amount"}}}]) ;

db.sales.aggregate([{$group:{_id:"$category",totalSales:{$avg:"$amount"}}}]) ;

db.sales.aggregate([{$group:{_id:"$category",totalSales:{$min:"$amount"}}}]) ;

db.sales.aggregate([{$group:{_id:"$category",totalSales:{$max:"$amount"}}}]) ;

db.sales.aggregate([{$group:{_id:"$category",totalSales:{$first:"$amount"}}}]) ;

db.sales.aggregate([{$group:{_id:"$category",totalSales:{$last:"$amount"}}}]) ;


 db.sales.aggregate([{$group:{_id:"$category",totalSales:{$push:"$amount"}}}]) ;

db.sales.aggregate([{$group:{_id:"$category",count:{$sum:1}}}]) ;

db.sales.createIndex({ product: 1 });
db.sales.createIndex({ amount: -1 });

db.sales.getIndexes();

db.sales.dropIndex({amount:-1});
```