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:

1. Find the name of the company that has the smallest payroll.
2. 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});