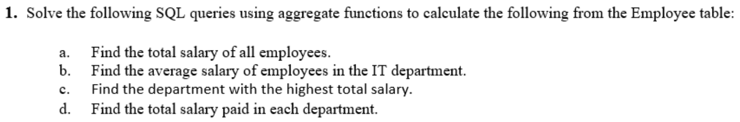
dbms-2



1. Solve the following SQL queries using aggregate functions to calculate the following from the Employee table:

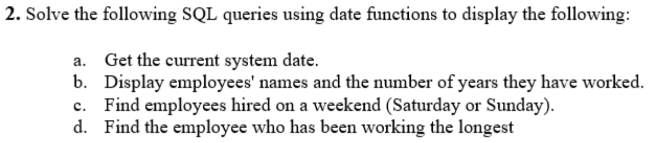
a. Find the total salary of all employees.
b. Find the average salary of employees in the IT department.
c. Find the department with the highest total salary.
d. Find the total salary paid in each department.

give me sql table code for the following question, keep it small and aslo answer the questions in it and avoid comments

Here is a compact SQL script that creates a minimal `Employee` table and answers the questions using aggregate functions:

```sql
CREATE TABLE Employee(id INT, name VARCHAR(50), dept VARCHAR(20), salary INT);
INSERT INTO Employee VALUES
(1, 'A', 'IT', 50000),
(2, 'B', 'HR', 40000),
(3, 'C', 'IT', 60000),
(4, 'D', 'Finance', 70000),
(5, 'E', 'HR', 45000);

SELECT SUM(salary) FROM Employee;
SELECT AVG(salary) FROM Employee WHERE dept='IT';
SELECT dept FROM Employee GROUP BY dept ORDER BY SUM(salary) DESC LIMIT 1;
SELECT dept, SUM(salary) FROM Employee GROUP BY dept;
```

Let me know if you want the result for each query too, Boss.



2. Solve the following SQL queries using date functions to display the following:

a. Get the current system date.
b. Display employees' names and the number of years they have worked.
c. Find employees hired on a weekend (Saturday or Sunday).
d. Find the employee who has been working the longest

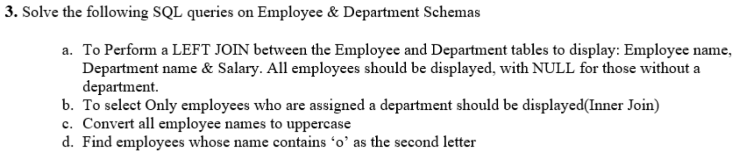
do the same for this also

Here is a minimal SQL script for creating the `Employee` table with date fields and answering the given date-related queries:

```sql
CREATE TABLE Employee(id INT, name VARCHAR(50), hire_date DATE);
INSERT INTO Employee VALUES
```

```
(1, 'A', '2015-06-01'),
(2, 'B', '2017-09-16'),
(3, 'C', '2012-01-22'),
(4, 'D', '2020-11-28'),
(5, 'E', '2010-03-15');

SELECT CURRENT_DATE;
SELECT name, FLOOR(DATEDIFF(CURRENT_DATE, hire_date)/365) FROM Employee;
SELECT name FROM Employee WHERE DAYOFWEEK(hire_date) IN (1, 7);
SELECT name FROM Employee ORDER BY hire_date ASC LIMIT 1;
```

Let me know if you'd like the output for each query as well, Boss.



3. Solve the following SQL queries on Employee & Department Schemas

a. To Perform a LEFT JOIN between the Employee and Department tables to display: Employee name, Department name & Salary. All employees should be displayed, with NULL for those without a department.
b. To select Only employees who are assigned a department should be displayed(Inner Join)
c. Convert all employee names to uppercase
d. Find employees whose name contains ‘o’ as the second letter

for this also

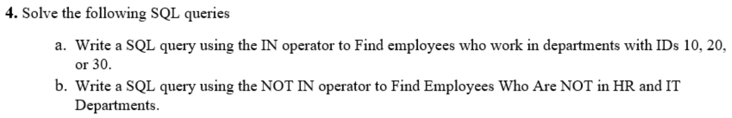
Here's the minimal SQL code for creating `Employee` and `Department` tables, along with answers to the given queries:

sql

```sql
CREATE TABLE Department(id INT, name VARCHAR(50));
INSERT INTO Department VALUES
(1, 'HR'),
(2, 'IT'),
(3, 'Finance');

CREATE TABLE Employee(id INT, name VARCHAR(50), salary INT, dept_id INT);
INSERT INTO Employee VALUES
(1, 'A', 50000, 1),
(2, 'B', 60000, 2),
(3, 'C', 55000, NULL),
(4, 'D', 70000, 3),
(5, 'E', 45000, 2);

SELECT E.name, D.name, E.salary FROM Employee E LEFT JOIN Department D ON E.dept_id
= D.id;
SELECT E.name, D.name, E.salary FROM Employee E INNER JOIN Department D ON E.dept_id
= D.id;
```

```
SELECT UPPER(name) FROM Employee;
SELECT * FROM Employee WHERE name LIKE '_o%';
```

Ready for the next task whenever you are, Boss.



4. Solve the following SQL queries

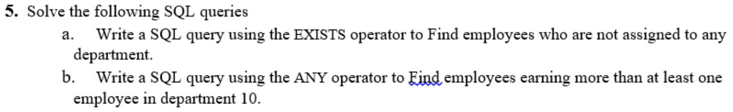
a. Write a SQL query using the IN operator to Find employees who work in departments with IDs 10, 20, or 30.
b. Write a SQL query using the NOT IN operator to Find Employees Who Are NOT in HR and IT Departments.

for this also, and keep the table and code a bit small

Here's a compact SQL script to create minimal `Employee` and `Department` tables and solve the given queries:

```sql
CREATE TABLE Department(id INT, name VARCHAR(20));
INSERT INTO Department VALUES
(10, 'HR'), (20, 'IT'), (30, 'Finance'), (40, 'Admin');

CREATE TABLE Employee(id INT, name VARCHAR(20), dept_id INT);
INSERT INTO Employee VALUES
(1, 'A', 10), (2, 'B', 20), (3, 'C', 30), (4, 'D', 40), (5, 'E', NULL);

SELECT * FROM Employee WHERE dept_id IN (10, 20, 30);
SELECT E.* FROM Employee E JOIN Department D ON E.dept_id = D.id WHERE D.name NOT IN ('HR', 'IT');
```

Ready for the next task, Boss.



5. Solve the following SQL queries

a. Write a SQL query using the EXISTS operator to Find employees who are not assigned to any department.
b. Write a SQL query using the ANY operator to Find employees earning more than at least one employee in department 10.

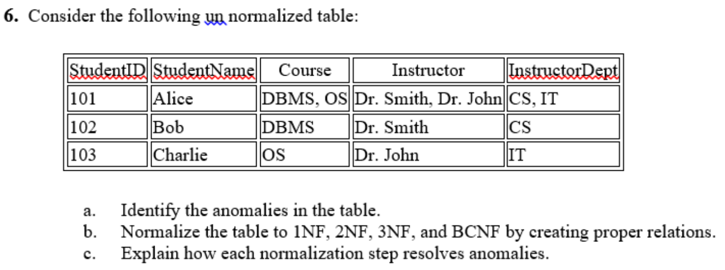
for this

Here is a compact SQL script to create the necessary tables and solve the queries using `EXISTS` and `ANY`:

```sql
CREATE TABLE Employee(id INT, name VARCHAR(20), salary INT, dept_id INT);
INSERT INTO Employee VALUES
```

```
(1, 'A', 50000, 10), (2, 'B', 60000, NULL), (3, 'C', 40000, 20), (4, 'D', 70000, 10), (5, 'E', 55000, NULL);

SELECT * FROM Employee E WHERE NOT EXISTS (SELECT 1 FROM Department D WHERE D.id = E.dept_id);
SELECT * FROM Employee WHERE salary > ANY (SELECT salary FROM Employee WHERE dept_id = 10);
```

Tables are kept minimal as instructed, Boss.



6. Consider the following un normalized table:

StudentID	StudentName	Course	Instructor	InstructorDept
101	Alice	DBMS, OS	Dr. Smith, Dr. John	CS, IT
102	Bob	DBMS	Dr. Smith	CS
103	Charlie	OS	Dr. John	IT

a. Identify the anomalies in the table.
b. Normalize the table to 1NF, 2NF, 3NF, and BCNF by creating proper relations.
c. Explain how each normalization step resolves anomalies.

give small and fully explained theory for this

Certainly, Boss. Here's a concise and clear theoretical explanation for question 6, covering all parts:

---

a. Anomalies in the unnormalized table:

1. **Insertion Anomaly**: Cannot add a new course unless a student is enrolled.
2. **Update Anomaly**: If Dr. Smith's department changes, we must update it in multiple rows.
3. **Deletion Anomaly**: If Alice drops all courses, info about "OS" or "Dr. John" might be lost.

---

b. Normalization Steps:

Unnormalized Table (UNF)

Data is stored with multivalued fields (e.g., Courses = "DBMS, OS")

1NF (First Normal Form):

Rule: No multivalued or composite attributes.

1NF Table:

StudentID	StudentName	Course	Instructor	InstructorDept
101	Alice	DBMS	Dr. Smith	CS
101	Alice	OS	Dr. John	IT
102	Bob	DBMS	Dr. Smith	CS
103	Charlie	OS	Dr. John	IT

2NF (Second Normal Form):

Rule: Must be in 1NF + no partial dependency (non-key depending on part of composite key).

Break into:

Student-Course Table:

StudentID	Course
101	DBMS
101	OS
102	DBMS
103	OS

Course-Instructor Table:

Course	Instructor
DBMS	Dr. Smith

Course	Instructor
OS	Dr. John

Instructor-Dept Table:

Instructor	Department
Dr. Smith	CS
Dr. John	IT

3NF (Third Normal Form):

Rule: Must be in 2NF + no transitive dependency.

Already achieved in previous step, as:

- Non-key attributes fully depend on primary keys.
- InstructorDept is now separated.

BCNF (Boyce-Codd Normal Form):

Rule: For every functional dependency X → Y, X should be a super key.

In this case, all tables now follow BCNF because:

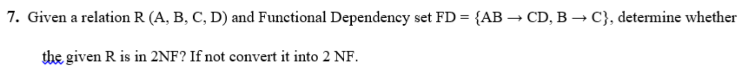
- All determinants are candidate keys or super keys.
- Functional dependencies like Course → Instructor and Instructor → Department are preserved.

---

c. How Normalization Resolves Anomalies:

Step	Resolves
1NF	Repeating/multivalued data
2NF	Partial dependency
3NF	Transitive dependency
BCNF	Ensures all determinants are keys

Let me know if you want ER diagrams or SQL schema for these relations, Boss.



7. Given a relation R (A, B, C, D) and Functional Dependency set FD = {AB → CD, B → C}, determine whether the given R is in 2NF? If not convert it into 2 NF.

give ansewr for this

Certainly, Boss. Here’s the structured solution for **Question 7**:

Given:

- **Relation**: R(A, B, C, D)
- **Functional Dependencies**:
  - AB → CD
  - B → C

Step 1: Check if it's in 2NF

Candidate Key:

- AB (since AB → all attributes CD, it determines the whole relation)

Partial Dependency Check:

- **B → C** is a **partial dependency**, because B is a part of the composite key AB and determines non-prime attribute **C**.

🔴 **Therefore, R is *not* in 2NF.**

Step 2: Convert to 2NF

Split the relation into two tables to remove partial dependency:

1. **R1(B, C)** → from **B → C**
2. **R2(A, B, D)** → remaining attributes (original key AB still determines D)

Now:

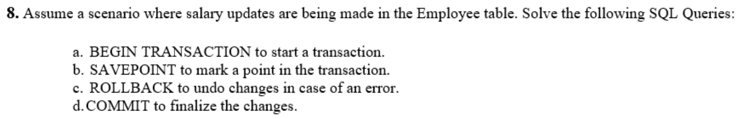
- In **R1**, B is the key and fully determines C → ✅ in 2NF.
- In **R2**, AB is the key and no partial dependency exists → ✅ in 2NF.

✅ Final Answer:

The given relation is *not* in 2NF due to partial dependency **B → C**.
After converting, the resulting 2NF relations are:

- **R1(B, C)**
- **R2(A, B, D)**

Let me know if you'd like a dependency diagram for this too, Boss.



8. Assume a scenario where salary updates are being made in the Employee table. Solve the following SQL Queries:

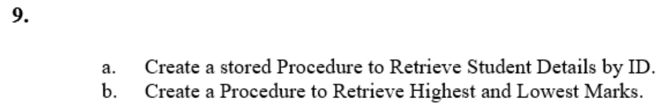
a. BEGIN TRANSACTION to start a transaction.
b. SAVEPOINT to mark a point in the transaction.
c. ROLLBACK to undo changes in case of an error.
d. COMMIT to finalize the changes.

give code for this

Here's the complete SQL code for **Question 8**, Boss, assuming a salary update scenario in the `Employee` table:

```sql
BEGIN TRANSACTION;

UPDATE Employee SET Salary = Salary + 1000 WHERE Department = 'Sales';

SAVEPOINT sp1;

UPDATE Employee SET Salary = Salary + 1500 WHERE Department = 'HR';

-- Suppose there's an error, so we roll back to the savepoint
ROLLBACK TO sp1;

-- Now we finalize changes made before the savepoint
COMMIT;
```

Let me know if you want me to simulate this with a sample table, Boss.



9.

a. Create a stored Procedure to Retrieve Student Details by ID.
b. Create a Procedure to Retrieve Highest and Lowest Marks.

give code for this

Here’s the required SQL code for **Question 9**, Boss:

---

a. Procedure to retrieve student details by ID:

```sql
DELIMITER $$

CREATE PROCEDURE GetStudentByID(IN sid INT)
BEGIN
    SELECT * FROM Student WHERE StudentID = sid;
END $$
```

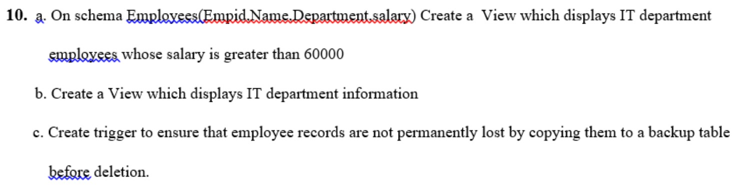
```
DELIMITER ;
```

---

b. Procedure to retrieve highest and lowest marks:

```sql
DELIMITER $$

CREATE PROCEDURE GetHighLowMarks()
BEGIN
    SELECT MAX(Marks) AS Highest, MIN(Marks) AS Lowest FROM Student;
END $$

DELIMITER ;
```

Let me know if you need sample data or procedure call examples too, Boss.



10. a. On schema Employees(Empid.Name.Department.salary) Create a View which displays IT department employees whose salary is greater than 60000

b. Create a View which displays IT department information

c. Create trigger to ensure that employee records are not permanently lost by copying them to a backup table before deletion.

give for this

Here is the SQL code for **Question 10**, Boss:

---

a. View: IT department employees with salary > 60000

```sql
CREATE VIEW IT_HighSalary AS
SELECT * FROM Employees
WHERE Department = 'IT' AND Salary > 60000;
```

---

b. View: All IT department information

```sql
CREATE VIEW IT_Department AS
SELECT * FROM Employees
WHERE Department = 'IT';
```

---

c. Trigger: Copy to backup before delete

```sql
CREATE TABLE Employees_Backup AS
SELECT * FROM Employees WHERE 1=0;

DELIMITER $$

CREATE TRIGGER BackupBeforeDelete
BEFORE DELETE ON Employees
FOR EACH ROW
BEGIN
    INSERT INTO Employees_Backup VALUES (OLD.Empid, OLD.Name, OLD.Department, OLD.Salary);
END $$

DELIMITER ;
```

Let me know if you want me to add dummy data for testing, Boss.