



**DEPARTMENT OF**

**COMPUTER SCIENCE & ENGINEERING**

Discover. Learn. Empower.

## EXPERIMENT 2.1

**Student Name:** Ayush Kohli

**UID:** 23BCS11238

**Branch:** CSE

**Section/Group:** KRG-3B

**Semester:** 5<sup>th</sup>

**Date of Performance:** 22/07/25

**Subject Name:** ADBMS

**Subject Code:** 23CSP-333

### 1. Aim:

To display the details of each employee along with their manager's name and department, using a self-join on the EMPLOYEE table.

### 2. Objective:

This code helps us:

1. Understand employee-manager relationships within the same table.
2. Use self join (i.e., joining the table with itself) to fetch manager-related data.
3. Provide a clear view of each employee's:
  - Name
  - Department
  - Manager's Name
  - Manager's Department

### 3. Code:

```
CREATE TABLE EMPLOYEE(  
EMP_ID INT primary key,  
EMP_NAME VARCHAR(25),  
DEPARTMENT VARCHAR(25),  
MANAGER_ID INT);
```

```
INSERT INTO EMPLOYEE  
(EMP_ID,EMP_NAME,DEPARTMENT,MANAGER_ID) VALUES  
(1, 'alice', 'hr', NULL),  
(2, 'bob', 'finance', 1),  
(3, 'charlie', 'it', 1),  
(4, 'david', 'finance', 2),  
(5, 'eve', 'it', 3),  
(6, 'frank', 'hr', 1);
```

```
SELECT E1.EMP_NAME AS [EMPLOYEE NAME], E2.EMP_NAME AS  
[MANAGER NAME],E1.DEPARTMENT AS [EMPLOYEE_DEPT],
```

```
E2.DEPARTMENT AS [MANAGER_DEPT]
FROM EMPLOYEE AS E1
LEFT
OUTER
JOIN
EMPLOY
EE AS E2
ON
E1.MANAGER_ID = E2.EMP_ID;
```

## 4. Output:

100 %		No issues found		
Results		Messages		
	EMPLOYEE NAME	MANAGER NAME	EMPLOYEE_DEPT	MANAGER_DEPT
1	alice	NULL	hr	NULL
2	bob	alice	finance	hr
3	charlie	alice	it	hr
4	david	bob	finance	finance
5	eve	charlie	it	it
6	frank	alice	hr	hr



## EXPERIMENT 2.2

**Student Name:** Ayush kohli

**UID:** 23BCS11238

**Branch:** CSE

**Section/Group:** KRG-3B

**Semester:** 5<sup>th</sup>

**Date of Performance:** 22/07/25

**Subject Name:** ADBMS

**Subject Code:** 23CSP-333

### 1. Aim:

To create and manage a relational database that stores information about faculties and their respective subjects, and to retrieve faculties that offer more than two subjects.

### 2. Objective:

Create two related tables:

TBL\_FACULTY: Stores faculty information (like Engineering, Mathematics, etc.).

TBL\_SUBJECTS: Stores subjects offered under each faculty.

Link the two tables using a foreign key:

The FACULTY\_REF column in the TBL\_SUBJECTS table is a foreign key that refers to FACULTY\_ID in the TBL\_FACULTY table.

Insert sample data into both tables to simulate a real-world college or university faculty-subject structure.

Use a JOIN and GROUP BY with HAVING clause to:

Count the number of subjects each faculty offers.

Show only those faculties that offer more than 2 subjects.

### 3. Code:

-- 1. Create table to hold actual NPV values

```
CREATE TABLE Year_tbl (  
    ID INT,  
    YEAR INT,  
    NPV INT  
);
```

-- 2. Create table for query requests

```
CREATE TABLE Queries (  
    ID INT,  
    YEAR INT  
);
```

-- 3. Insert data into Year\_tbl

```
INSERT INTO Year_tbl (ID, YEAR, NPV) VALUES  
(1, 2018, 100),  
(7, 2020, 30),  
(13, 2019, 40),  
(1, 2019, 113),  
(2, 2008, 121),  
(3, 2009, 12),  
(11, 2020, 99),  
(7, 2019, 0);
```

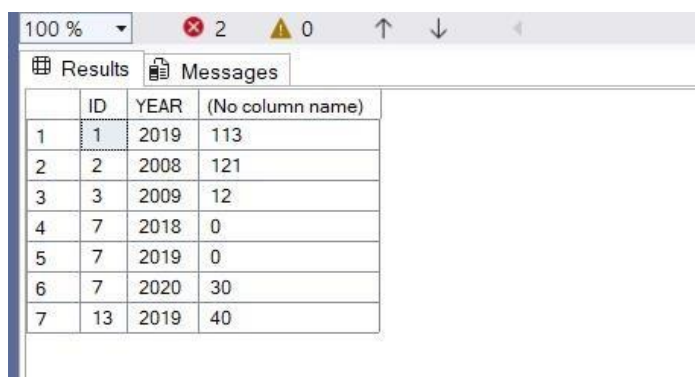
-- 4. Insert data into Queries

```
INSERT INTO Queries (ID, YEAR) VALUES  
(1, 2019),  
(2, 2008),  
(3, 2009),  
(7, 2018),  
(7, 2019),  
(7, 2020),  
(13, 2019);
```

-- 5. Final query: Return (ID, YEAR) with NPV if available, else 0

```
SELECT  
    Q.ID,  
    Q.YEAR,  
    ISNULL(Y.NPV, 0) AS NPV  
FROM  
    Queries AS Q  
LEFT OUTER JOIN  
    Year_tbl AS Y  
ON  
    Q.ID = Y.ID AND Q.YEAR = Y.YEAR;
```

#### 4. Output:



	ID	YEAR	(No column name)
1	1	2019	113
2	2	2008	121
3	3	2009	12
4	7	2018	0
5	7	2019	0
6	7	2020	30
7	13	2019	40