Advance Database Management Systems

EXPERIMETN 02: JOINS

Organizational Hierarchy Explorer (medium)

You are a **Database Engineer** at **TalentTree Inc.**, an enterprise HR analytics platform that stores employee data, including their reporting relationships. The company maintains a centralized **Employee** relation that holds:
Each employee’s ID, name, department, and manager ID (who is also an employee in the same table).
Your task is to generate a report that **maps employees to their respective managers**, showing:
The employee’s name and department
Their manager’s name and department (if applicable)
This will help the HR department visualize the internal reporting hierarchy.

Input Table: Employee

EmpID	Ename	Department	ManagerID
1	Alice	HR	NULL
2	Bob	Finance	1
3	Charlie	IT	1
4	David	Finance	2
5	Eve	IT	3
6	Frank	HR	1

OUTPUT

EmployeeName	EmployeeDept	Manager Name	ManagerDept
Alice	HR	NULL	NULL
Bob	Finance	Alice	HR
Charlie	IT	Alice	HR
David	Finance	Bob	Finance
Eve	IT	Charlie	IT
Frank	HR	Alice	HR

Financial Forecast Matching with Fallback Strategy (hard)

You are a Data Engineer at **FinSight Corp**, a company that models Net Present Value (NPV) projections for investment decisions. Your system maintains two key datasets:

1. **Year_tbl:** Actual recorded NPV's of various financial instruments over different years:
   - **ID:** Unique Financial instrument identifier.
   - **YEAR:** Year of record
   - **NPV:** Net Present Value in that year

2. **Queries_tbl:** A list of instrument-year pairs for which stakeholders are requesting NPV values:
   - **ID:** Financial instrument identifier
   - **YEAR:** Year of interest.

Find the NPV of each query from the Queries table. Return the output order by ID and Year in the sorted form. However, not all **ID-YEAR combinations** in the Queries table are present in the Year_tbl. If an NPV is missing for a requested combination, assume it to be 0 to maintain a consistent financial report.

Year_tbl

ID	YEAR	NPV
1	2018	100
7	2020	30
13	2019	40
1	2019	113
2	2008	121
3	2009	12
11	2020	99
7	2019	0

ID	YEAR
1	2019
2	2008
3	2009
7	2018
7	2019
7	2020
13	2019

Queries_tbl

OUTPUT:

ID	YEAR	NPV
1	2019	113
2	2008	121
3	2009	12
7	2018	0
7	2019	0
7	2020	30
13	2019	40

EXPERIMENT 03: SUB-QUERIES

Department Salary Champions

In a bustling corporate organization, each department strives to retain the most talented (and well-compensated) employees. You have access to two key records: **one lists every employee along with their salary and department**, **while the other details the names of each department**. Your task is to identify the **top earners in every department**.

If multiple employees share the same highest salary within a department, all of them should be celebrated equally. The final result should present the **department name, employee name, and salary of these top-tier professionals** arranged by department.

Input Table: **Employee**

ID	NAME	SALARY	DEPT_ID
1	JOE	70000	1
2	JIM	90000	1
3	HENRY	80000	2
4	SAM	60000	2
4	MAX	90000	1

Department

ID	DEPT_NAME
1	IT
2	SALES

OUTPUT

DEPT_NAME	NAME	SALARY
IT	MAX	90000
IT	JIM	90000
SALES	HENRY	80000

Merging Employee Histories: Who Earned Least? (Hard)

Two legacy HR systems (A and B) have separate records of employee salaries. These records may overlap. Management wants to **merge these datasets** and identify **each unique employee (**by EmpID) along with their **lowest recorded salary** across both systems.

Objective

1. Combine two tables A and B.
2. Return each EmpID with their **lowest salary,** and the corresponding **Ename.**

Table A

EmpID	Ename	Salary
1	AA	1000
2	BB	300

Table B

EmpID	Ename	Salary
2	BB	400
3	CC	100

OUTPUT:

EmpID	Ename	Salary
1	AA	1000
2	BB	300
3	CC	100

IF-ELSE & ELSE-IF LADDER (SQL SERVER)

```sql
IF condition
    BEGIN
        -- statements
    END
ELSE
    BEGIN
        -- statements
    END
```

```sql
IF condition1
    -- statements
ELSE IF condition2
    -- statements
ELSE IF condition3
    -- statements
ELSE
    -- statements
```

Examples:

```
DECLARE @Salary INT = 6000;

IF @Salary > 10000
    PRINT 'High Salary';
ELSE
    PRINT 'Average or Low Salary';
```

Examples:

```sql
DECLARE @Marks INT = 78;

IF @Marks >= 90
    PRINT 'Grade A';
ELSE IF @Marks >= 75
    PRINT 'Grade B';
ELSE IF @Marks >= 60
    PRINT 'Grade C';
ELSE
    PRINT 'Grade D';
```

IF-ELSE & ELSE-IF LADDER (POSTGRES)

```
DO $$
BEGIN
    IF condition THEN
        -- statements
    ELSE
        -- statements
    END IF;
END $$;
```

```
DO $$
BEGIN
    IF condition1 THEN
        -- statements
    ELSIF condition2 THEN
        -- statements
    ELSIF condition3 THEN
        -- statements
    ELSE
        -- default
    END IF;
END $$;
```

Examples:

```
DO $$
BEGIN
    IF 100 > 50 THEN
        RAISE NOTICE '100 is greater than 50';
    ELSE
        RAISE NOTICE '50 is greater';
    END IF;
END $$;
```

Examples:

```
DO $$
BEGIN
    IF 85 >= 90 THEN
        RAISE NOTICE 'Grade A';
    ELSIF 85 >= 75 THEN
        RAISE NOTICE 'Grade B';
    ELSE
        RAISE NOTICE 'Grade C';
    END IF;
END $$;
```

CASE STATEMENTS (SQL SERVER)

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ELSE resultN
END
```

Example:

```sql
SELECT
Name,
Salary,
    CASE
        WHEN Salary >= 10000 THEN 'High Earner'
        WHEN Salary BETWEEN 7000 AND 9999 THEN 'Medium Earner'
        ELSE 'Low Earner'
    END AS SalaryCategory
FROM Employee;
```

Practice Set:

```
CREATE TABLE Department (
   DeptID INT PRIMARY KEY,
   DeptName VARCHAR(50)
);

CREATE TABLE Employee (
   EmpID INT PRIMARY KEY,
   Name VARCHAR(50),
   DeptID INT,
   Salary INT,
   Marks INT
);
```

```
INSERT INTO Department VALUES
(1, 'IT'), (2, 'HR'), (3, 'Finance');

INSERT INTO Employee VALUES
(1, 'John', 1, 12000, 88),
(2, 'Alice', 2, 9000, 72),
(3, 'Bob', 1, 6000, 55),
(4, 'Sarah', 3, 15000, 95),
(5, 'David', 2, 7000, 65),
(6, 'Tom', 1, 8000, 45);
```

Practice Set:

Sno	Problem Statement
1	Write an IF-ELSE block to check if the average salary of all the employees is greater than 1000. (SQL SERVER + POSTGRES) If yes, print ‘Highest Salary Company’, else ‘Average Salary Company’
2	Use IF-ELSE Ladder to classify marks of a given employee (second class => 50/ first class >= 70/ distinction > 85)
3	Write a query to classify employees based on salary using CASE statements: 1. >=12000 – Platinum 2. 8000 – 11999 – Gold 3. Else Silver

Functional Dependency

$$\alpha \longrightarrow \beta$$

Determinant Dependent

*If we know the value of **α**, we can get the value of **β** from the relation.*

t1 ->	a	1
t2 ->	b	2
	c	3
tn ->	d	4

For each value of **α** we should have same value of **β.**

If t1[**α**] = t2[**α**]
then
t1[**β**] = t2[**β**]
should be equal

a	1
a	2
C	3
d	4

Here in this case, on same value of **α** we have different value of **β.**

Practice Q/A

A	B	C	D	E
a	2	3	4	5
2	a	3	4	5
a	2	3	6	5
a	2	3	6	6

Which of the following functional dependency holds true as per the rule and why?

- A-> BC
- DE -> C
- C -> DE
- BC -> A

Closure of Functional Dependencies

- Closure method helps in finding all the CANDIDATE KEYS in a relation.
- While dealing with the normalization concept and functional dependencies, this is the most important and crucial concept.
- With the help of candidate keys, we can identify PRIME and NON PRIME attributes which further helps in dealing with the normal forms.
- E.g. R(A, B, C, D)   F.D. Set – { A->B, B->C, C->D }

A+ = ABCD	A can determine each attribute of relation R, Hence it is a candidate key.
B+ = BCD	B can not determine each attribute of relation R, Hence it is not candidate key.
C+ = CD	C can not determine each attribute of relation R, Hence it is not candidate key.
D+ = D	D can not determine each attribute of relation R, Hence it is not candidate key.

Also have to check the combination, with ‘A’

(AB)+ = ABCD Hence, it is also a candidate key

But according to rule, candidate key is minimal in nature. So we cannot take AB as candidate key.
Hence , only ‘A’ will be the only candidate key.

PRIME ATTRIBUTE – {A}
NON-PRIME ATTRIBUTE – {B,C,D}

Tips and Tricks (Finding Candidate Key in Seconds)
Check the attribute which is not on the RHS, that attribute will sure be the part of candidate key.

Problem Statements

1. Consider a relation R having attributes as R(ABCD), functional dependencies are given below:

AB->C, C->D, D->A

Identify the set of candidate keys possible in relation R. List all the set of prime and non prime attributes.

2. Relation R(ABCDE) having functional dependencies as :
A->D, B->A, BC->D, AC->BE

Identify the set of candidate keys possible in relation R. List all the set of prime and non prime attributes.

3. Consider a relation R having attributes as R(ABCDE), functional dependencies are given below:

B->A, A->C, BC->D, AC->BE

Identify the set of candidate keys possible in relation R. List all the set of prime and non prime attributes.

4. Consider a relation R having attributes as R(ABCDEF), functional dependencies are given below:

A->BCD, BC->DE, B->D, D->A

Identify the set of candidate keys possible in relation R. List all the set of prime and non prime attributes.

Problem Statement

- Designing a student database involves certain dependencies which are listed below:

X ->Y

WZ ->X

WZ ->Y

Y ->W

Y ->X

Y ->Z

The task here is to remove all the redundant FDs for efficient working of the student database management system.

Problem Statement

Debix Pvt Ltd needs to maintain database having dependent attributes ABCDEF. These attributes are functionally dependent on each other for which functionally dependency set F given as:

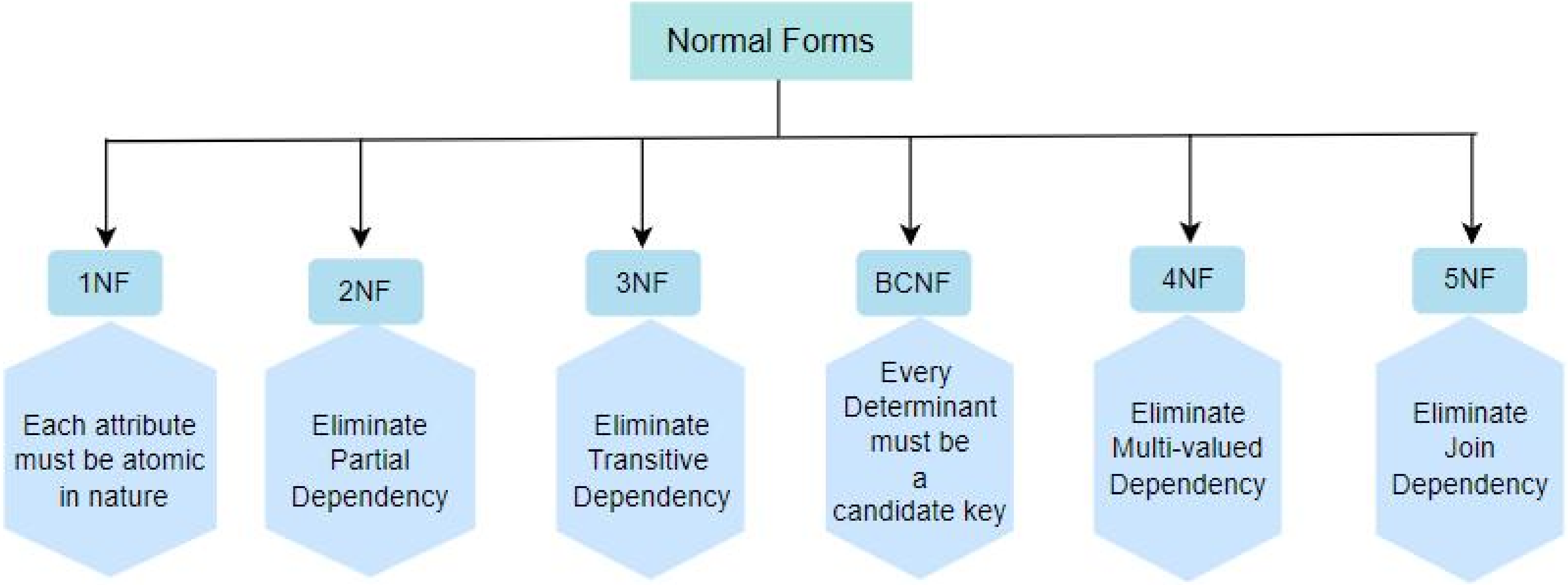
{A -> BC, D -> E, BC -> D, A -> D} Consider a universal relation R1(A, B, C, D, E, F) with functional dependency set F, also all attributes are simple and take atomic values only. Find the highest normal form along with the candidate keys with prime and non-prime attribute.

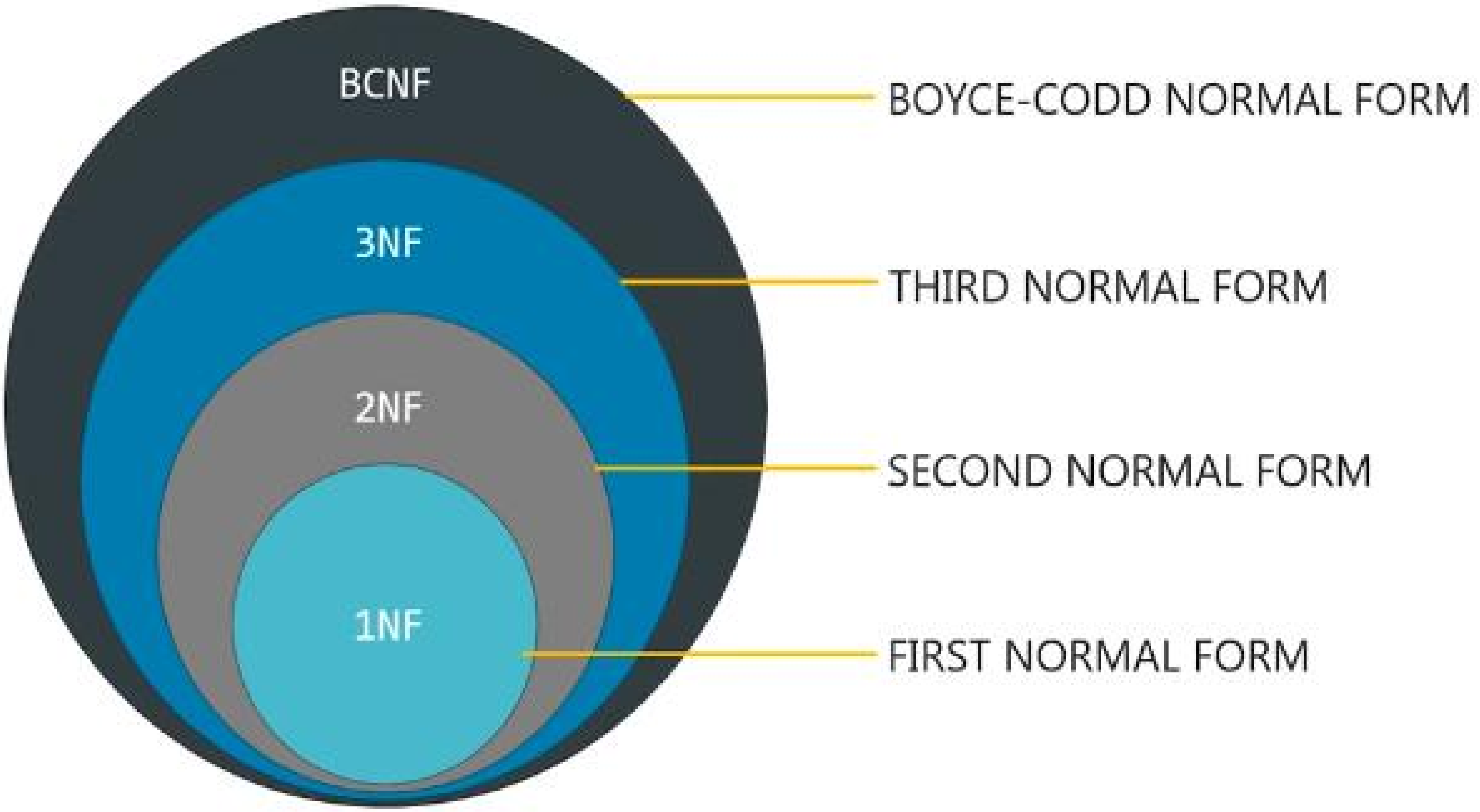
Normalization

- When we define a large database as a single relation, there are chances of data redundancy and inconsistency.
- This leads to various problems such as maintaining the huge amount of data, improper utilization of disk space and resources and many more.
- In-order to handle these issues, NORMALIZATION comes into the picture.
- Normalization helps us to reduce the redundancy from relations, divides the larger table into smaller tables and helps in dealing with insertion, deletion and updatation anomalies.

Data Anomalies

Normalization




BCNF
3NF
2NF
1NF
BOYCE-CODD NORMAL FORM
THIRD NORMAL FORM
SECOND NORMAL FORM
FIRST NORMAL FORM

1ST NORMAL FORM

Removes repeating groups from the table

Create a separate table for each set of related data

Identify each set of related data with a primary key

1NF

Employee ID	Employee Name	Phone Number	Salary
1EDU001	Alex	+91 8553206126 +91 9449424949	60,131
1EDU002	Barry	+91 8762989672	48,302
1EDU003	Clair	+91 9916255225	22,900
1EDU004	David	+91 6363625811 +91 8762055007	81,538

1NF - Decomposition

Employee ID	Employee Name	Phone Number	Salary
1EDU001	Alex	+91 8553206126	60,131
1EDU001	Alex	+91 9449424949	60,131
1EDU002	Barry	+91 8762989672	48,302
1EDU003	Clair	+91 9916255225	22,900
1EDU004	David	+91 6363625811	81,538
1EDU004	David	+91 8762055007	81,538

2nd NORMAL FORM

It has to be in 1st Normal Form

Table also should not contain partial dependency

2-NF

Employee Id	Department Id	Office Location
1EDU001	ED-T1	Pune
1EDU002	ED-S2	Bengaluru
1EDU003	ED-M1	Delhi
1EDU004	ED-T3	Mumbai

Employee ID and Department ID forms the P.K in this case.

Department ID determines Office Location – Clear cut case of Partial Dependency

2-NF - Decomposition

Employee Id	Department Id	Office Location
1EDU001	ED-T1	Pune
1EDU002	ED-S2	Bengaluru
1EDU003	ED-M1	Delhi
1EDU004	ED-T3	Mumbai

→

Employee Id	Department Id
1EDU001	ED-T1
1EDU002	ED-S2
1EDU003	ED-M1
1EDU004	ED-T3

Department Id	Office Location
ED-T1	Pune
ED-S2	Bengaluru
ED-M1	Delhi
ED-T3	Mumbai

It has to be in 2nd Normal Form

There should be no transitive dependency for non-prime attributes

Transitivity?

3-NF

Student Id	Student Name	Subject Id	Subject	Address
1DT15ENG01	Alex	15CS11	SQL	Goa
1DT15ENG02	Barry	15CS13	JAVA	Bengaluru
1DT15ENG03	Clair	15CS12	C++	Delhi
1DT15ENG04	David	15CS13	JAVA	Kochi

Student ID → Subject ID → Subject --- Transitive Dependency

3-NF - Decomposition

Student Id	Student Name	Subject Id	Subject	Address
1DT15ENG01	Alex	15CS11	SQL	Goa
1DT15ENG02	Barry	15CS13	JAVA	Bengaluru
1DT15ENG03	Clair	15CS12	C++	Delhi
1DT15ENG04	David	15CS13	JAVA	Kochi

Student Id	Student Name	Subject Id	Address
1DT15ENG01	Alex	15CS11	Goa
1DT15ENG02	Barry	15CS13	Bengaluru
1DT15ENG03	Clair	15CS12	Delhi
1DT15ENG04	David	15CS13	Kochi

Subject Id	Subject
15CS11	SQL
15CS13	JAVA
15CS12	C++
15CS13	JAVA

BC NORMAL FORM

It has to be in 3rd Normal Form

Higher version 3NF and was developed by Raymond F. Boyce and Edgar F. Codd

Every functional dependency A → B, then A has to be the Super Key of that particular table

BCNF

Student ID	Subject	Professor
1DT15ENG01	SQL	Prof. Mishra
1DT15ENG02	JAVA	Prof. Anand
1DT15ENG02	C++	Prof. Kanthi
1DT15ENG03	JAVA	Prof. Anand
1DT15ENG04	DBMS	Prof. Lokesh

One student can enroll for multiple subjects, there can be multiple professors teaching one subject, For each subject professor has been assigned to the student.

Student ID and Subject are the PA here and Professor is NPA.
Professor → Subject – Clear cut case for BCNF

BCNF - Decomposition

Student ID	Professor ID
1DT15ENG01	1DTPF01
1DT15ENG02	1DTPF02
1DT15ENG02	1DTPF03
1DT15ENG03	1DTPF02
1DT15ENG04	1DTPF04

Professor ID	Subject	Professor
1DTPF01	SQL	Prof. Mishra
1DTPF02	JAVA	Prof. Anand
1DTPF03	C++	Prof. Kanthi
1DTPF02	JAVA	Prof. Anand
1DTPF04	DBMS	Prof. Lokesh

Why we added new column as Professor ID?

1. By doing this we are removing the NPA → PA dependency.
2. Professor ID will be the primary key or super key for the second table for identifying the records uniquely.

Normalization

- **<u>Tips and Tricks:</u>** How to identify quickly that relation is in which form.
- 2NF (when is it not): if there is dependency from X -> Y

  (X is a subset of candidate key AND Y is non-prime attribute) – then R is not in 2NF
- 3NF (when is it): if there is a dependency from X -> Y

  (X is a super key or candidate key OR Y is a prime attribute)

  (If all attributes comes out to be prime – R is in 3NF)
- BCNF (when is it): if there is a dependency from X -> Y

  (X is super key or candidate key)

Views: Performance Benchmarking : Normal View vs. Materialized View

(Medium)

1. Create a large dataset:
   - Create a table names transaction_data (id , value) with 1 million records.
     - take id 1 and 2, and for each id, generate 1 million records in value column
   - Use Generate_series () and random() to populate the data.

2. Create a normal view and materialized view to for sales_summary, which includes total_quantity_sold, total_sales, and total_orders with aggregation.

3. Compare the performance and execution time of both.

Views: Securing Data Access with Views and Role-Based Permissions (Hard)

The company **TechMart Solutions** stores all sales transactions in a central database.
A new reporting team has been formed to analyze sales but **they should not have direct access to the base tables** for security reasons.
The database administrator has decided to:

1. Create **restricted views** to display only summarized, non-sensitive data.

2. Assign access to these views to specific users using **DCL commands** (GRANT, REVOKE).

Normal View vs Materialized View

Feature	Normal View	Materialized View
Storage	No storage, computed on-the-fly	Stores the result physically on disk
Query Execution	Slow for complex queries	Fast, because data is precomputed
Always Up-to-date	Yes, reflects latest table data	No, needs `REFRESH` to update
Use Case	Lightweight queries	Heavy aggregations, analytics, dashboards

When to go for Materialized Views?

- **Analytics Dashboards / BI Reports**
  - Example: Sales dashboard showing total sales by product/category.
  - Data refresh interval: nightly, hourly, or on-demand.
  - Why: Aggregations like SUM, AVG on millions of rows are expensive to compute every request.
- **Caching Frequently Accessed Data**
  - Example: Leaderboard for a gaming app or trending posts in a social media app.
  - Refresh periodically (e.g., every 10 minutes) instead of recomputing every time.
- **Complex Joins or Aggregations**
  - Example: Multiple large tables joined with filters for reporting.
  - Materialized view stores **precomputed results**, speeding up front-end API calls.
- **ETL / Data Warehousing**
  - Precompute summary tables for faster downstream analytics.
  - Refresh after batch data load.

Stored Procedures

HR-Analytics: Employee count based on dynamic gender passing (Medium)

TechSphere Solutions, a growing IT services company with offices across India, wants to **track and monitor gender diversity** within its workforce. The HR department frequently needs to know the **total number of employees by gender** (Male or Female) .

To solve this problem, the company needs an **automated database-driven solution** that can instantly return the count of employees by gender through a stored procedure that:

1. Create a PostgreSQL stored procedure that:
2. Takes a **gender** (e.g., 'Male' or 'Female') as input.
3. Calculates the **total count of employees** for that gender.
4. Returns the result as an **output parameter**.
5. Displays the result clearly for HR reporting purposes.

Stored Procedures

SmartStore Automated Purchase System (Hard)

SmartShop is a modern retail company that sells electronic gadgets like smartphones, tablets, and laptops.
The company wants to **automate its ordering and inventory management process**.
Whenever a customer places an order, the system must:

1. **Verify stock availability** for the requested product and quantity.
2. If sufficient stock is available:
   - **Log the order** in the sales table with the ordered quantity and total price.
   - **Update the inventory** in the products table by reducing quantity_remaining and increasing quantity_sold.
   - Display a **real-time confirmation message**: "Product sold successfully!“
3. If there is **insufficient stock**, the system must:
   - **Reject the transaction** and display: Insufficient Quantity Available!"