

**DATABASE MANAGEMENT SYSTEMS (DSC221-3)**

**CIA-III**

**COMPONENT-A**

**PROJECT ON ALL THE TOPICS**

**By**

**SHUBHANKARI SHUBHAM**

**REG NO-2341334**

**3BSc Data Science and Mathematics**

**Submitted to**

**Dr. UMMESALMA M**

**INTRODUCTION:**

One of the largest and fastest-growing industries in India has been the telecommunication system, which has transformed a lot owing to new technological advancements and optimum regulatory reforms. This sector reaches millions of users in the country with mobile, fixed-line, broadband, and satellite services. Among them, BSNL and MTNL are state-owned companies, and Reliance Jio, Airtel, and Vodafone Idea are some of the major private players in the sector. It is governed by the Telecom Regulatory Authority of India that promotes fair competition but safeguards the consumer interest. India was famous for providing the cheapest rates in the world. Significant contribution has come through this sector to the growth of digital India, especially with programs such as Digital India that were designed to expand digital infrastructure and provide connectivity to far-off rural areas. Communication is a powerful driver of economic development and digital transformation for India.

**EXPLANATION ABOUT ALL THE TOPICS FROM LAB 1 TO LAB 10:**

1. DDL COMMANDS: Data Definition Language (DDL) commands are SQL operations used to define and manage the structure of database objects such as tables, schemas, and indexes. The primary DDL commands include CREATE, which is used to create new database objects; ALTER, which modifies existing objects by adding, deleting, or changing columns; DROP, which removes objects from the database; and TRUNCATE, which deletes all rows in a table but retains the table structure for future use. These commands are essential for setting up and managing the architecture of a database, allowing users to define how data is stored and organized within the system. Proper use of DDL commands is crucial for maintaining the integrity and structure of the database environment.
2. **Creation of tables:**

CREATE TABLE ServicePlan (

plan\_id INT PRIMARY KEY AUTO\_INCREMENT,

plan\_name VARCHAR(100) NOT NULL,

cost DECIMAL(10, 2),

data\_limit INT

);

describe ServicePlan;

OUTPUT:

+------------+---------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+------------+---------------+------+-----+---------+----------------+

| plan\_id | int | NO | PRI | NULL | auto\_increment |

| plan\_name | varchar(100) | NO | | NULL | |

| cost | decimal(10,2) | YES | | NULL | |

| data\_limit | int | YES | | NULL | |

+------------+---------------+------+-----+---------+----------------+

CREATE TABLE Customer (

customer\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

address VARCHAR(255),

phone\_number VARCHAR(15) UNIQUE NOT NULL,

plan\_id INT,

FOREIGN KEY (plan\_id) REFERENCES ServicePlan(plan\_id)

);

describe Customer;

OUTPUT:

+--------------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+--------------+--------------+------+-----+---------+----------------+

| customer\_id | int | NO | PRI | NULL | auto\_increment |

| name | varchar(100) | NO | | NULL | |

| address | varchar(255) | YES | | NULL | |

| phone\_number | varchar(15) | NO | UNI | NULL | |

| plan\_id | int | YES | MUL | NULL | |

+--------------+--------------+------+-----+---------+----------------+

1. **ALTERING COLUMNS**

ALTER TABLE Customer ADD email VARCHAR(100);

OUTPUT:

+--------------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+--------------+--------------+------+-----+---------+----------------+

| customer\_id | int | NO | PRI | NULL | auto\_increment |

| name | varchar(100) | NO | | NULL | |

| address | varchar(255) | YES | | NULL | |

| phone\_number | varchar(15) | NO | UNI | NULL | |

| plan\_id | int | YES | MUL | NULL | |

| email | varchar(100) | YES | | NULL | |

+--------------+--------------+------+-----+---------+----------------+

1. **MODIFYING:**

ALTER TABLE Customer MODIFY phone\_number VARCHAR(20);

+--------------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+--------------+--------------+------+-----+---------+----------------+

| customer\_id | int | NO | PRI | NULL | auto\_increment |

| name | varchar(100) | NO | | NULL | |

| address | varchar(255) | YES | | NULL | |

| phone\_number | varchar(20) | YES | UNI | NULL | |

| plan\_id | int | YES | MUL | NULL | |

| email | varchar(100) | YES | | NULL | |

+--------------+--------------+------+-----+---------+----------------+

1. **DROPPING A COLUMN:**

ALTER TABLE Customer DROP COLUMN email;

describe Customer;

OUTPUT:

+--------------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+--------------+--------------+------+-----+---------+----------------+

| customer\_id | int | NO | PRI | NULL | auto\_increment |

| name | varchar(100) | NO | | NULL | |

| address | varchar(255) | YES | | NULL | |

| phone\_number | varchar(20) | YES | UNI | NULL | |

| plan\_id | int | YES | MUL | NULL | |

+--------------+--------------+------+-----+---------+----------------+

1. **ALTER TABLE NAME:**

ALTER TABLE Customer RENAME TO TelecomCustomer;

describe TelecomCustomer;

OUTPUT:

+--------------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+--------------+--------------+------+-----+---------+----------------+

| customer\_id | int | NO | PRI | NULL | auto\_increment |

| name | varchar(100) | NO | | NULL | |

| address | varchar(255) | YES | | NULL | |

| phone\_number | varchar(20) | YES | UNI | NULL | |

| plan\_id | int | YES | MUL | NULL | |

+--------------+--------------+------+-----+---------+----------------+

1. **DML COMMANDS(INSERT , DELETE, UPDATE):** Data Manipulation Language (DML) commands are crucial SQL operations that allow users to manage and manipulate data within database tables. The primary DML commands include SELECT, which retrieves data from one or more tables, enabling users to view specific information; INSERT, which adds new rows of data into a table, facilitating data entry; UPDATE, which modifies existing records based on specified conditions, allowing for data corrections and adjustments; and DELETE, which removes one or more rows from a table according to given criteria. Together, these commands form the backbone of data management in relational databases, enabling efficient manipulation and retrieval of information as needed.
2. **INSERT:**

INSERT INTO ServicePlan (plan\_name, cost, data\_limit)

VALUES ('Prepaid', 199.99, 30);

SELECT \* FROM ServicePlan;

OUTPUT:

+---------+-----------+--------+------------+

| plan\_id | plan\_name | cost | data\_limit |

+---------+-----------+--------+------------+

| 1 | Prepaid | 199.99 | 30 |

+---------+-----------+--------+------------+

INSERT INTO TelecomCustomer(name, address, phone\_number, plan\_id)

VALUES ('Alice Smith', 'Delhi', '9812345678', 1);

SELECT \* FROM TelecomCustomer;

OUTPUT

+-------------+-------------+---------+--------------+---------+

| customer\_id | name | address | phone\_number | plan\_id |

+-------------+-------------+---------+--------------+---------+

| 1 | Alice Smith | Delhi | 9812345678 | 1 |

+-------------+-------------+---------+--------------+---------+

1. **UPDATE:**

UPDATE TelecomCustomer

SET phone\_number = '9876543210'

WHERE customer\_id = 1;

SELECT \* FROM TelecomCustomer;;

OUTPUT:

+-------------+-------------+---------+--------------+---------+

| customer\_id | name | address | phone\_number | plan\_id |

+-------------+-------------+---------+--------------+---------+

| 1 | Alice Smith | Delhi | 9876543210 | 1 |

+-------------+-------------+---------+--------------+---------+

1. **DELETE:**

DELETE FROM TeleCustomer WHERE customer\_id = 2;

SELECT \* FROM Telecustomer;

1. **CONSTRAINTS:** Constraints in SQL are rules that ensure data integrity and accuracy within a database table. Key types include PRIMARY KEY, which uniquely identifies each row and cannot be NULL; FOREIGN KEY, which establishes relationships between tables by ensuring values match; NOT NULL, which mandates that a column cannot contain NULL values; UNIQUE, which ensures all values in a column are distinct; CHECK, which enforces specific conditions on column values; DEFAULT, which provides a default value when none is specified; and AUTO\_INCREMENT, which automatically generates unique values for a column. Together, these constraints are essential for maintaining high-quality and reliable data in relational databases. Now we will see some commands through which we can implement these.

CREATE TABLE ServicesPlan (

plan\_id INT AUTO\_INCREMENT PRIMARY KEY,

plan\_name VARCHAR(100) NOT NULL,

cost DECIMAL(10, 2) CHECK (cost > 0),

data\_limit INT DEFAULT 50

);

describe ServicesPlan;

OUTPUT:

+------------+---------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+------------+---------------+------+-----+---------+----------------+

| plan\_id | int | NO | PRI | NULL | auto\_increment |

| plan\_name | varchar(100) | NO | | NULL | |

| cost | decimal(10,2) | YES | | NULL | |

| data\_limit | int | YES | | 50 | |

+------------+---------------+------+-----+---------+----------------+

1. CREATE TABLE Customer (

customer\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

address VARCHAR(255) NULL,

phone\_number VARCHAR(15) UNIQUE NOT NULL,

plan\_id INT,

FOREIGN KEY (plan\_id) REFERENCES ServicePlan(plan\_id) ON DELETE SET NULL );

describe Customer;

OUTPUT:

--------------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+--------------+--------------+------+-----+---------+----------------+

| customer\_id | int | NO | PRI | NULL | auto\_increment |

| name | varchar(100) | NO | | NULL | |

| address | varchar(255) | YES | | NULL | |

| phone\_number | varchar(15) | NO | UNI | NULL | |

| plan\_id | int | YES | MUL | NULL | |

+--------------+--------------+------+-----+---------+----------------+

1. **GROUPING FUNCTIONS:** Grouping functions in SQL, often referred to as aggregate functions, are used to perform calculations on a set of values and return a single summary value. Common grouping functions include COUNT(), which counts the number of rows; SUM(), which calculates the total of a numeric column; AVG(), which finds the average value of a numeric column; MIN(), which retrieves the smallest value, and MAX(), which finds the largest value. These functions are typically used in conjunction with the GROUP BY clause, which groups rows that have the same values in specified columns into summary rows. Grouping functions are essential for data analysis, allowing users to summarize and gain insights from large datasets efficiently.
2. SELECT plan\_id, COUNT(\*) FROM TelecomCustomer GROUP BY plan\_id;

OUTPUT:

---------+----------+

| plan\_id | COUNT(\*) |

+---------+----------+

| 1 | 1 |

+---------+----------+

1. SELECT plan\_id, COUNT(\*)

FROM TelecomCustomer

GROUP BY plan\_id

HAVING COUNT(\*) > 5;

OUTPUT :

DOES NOT EXIST.

1. SELECT \* FROM TelecomCustomer ORDER BY name ASC;

Output:

+-------------+-------------+---------+--------------+---------+

| customer\_id | name | address | phone\_number | plan\_id |

+-------------+-------------+---------+--------------+---------+

| 1 | Alice Smith | Delhi | 9876543210 | 1 |

+-------------+-------------+---------+--------------+---------+

SELECT DISTINCT plan\_name

FROM ServicePlan;

Output:

+-----------+

| plan\_name |

+-----------+

| Prepaid |

+-----------+

SELECT \*

FROM Customer

LIMIT 5;

Output :

+-------------+-------------+---------+--------------+---------+

| customer\_id | name | address | phone\_number | plan\_id |

+-------------+-------------+---------+--------------+---------+

| 1 | Alice Smith | Delhi | 9876543210 | 1 |

+-------------+-------------+---------+--------------+---------+

SELECT

SUM(cost) AS total\_cost,

AVG(cost) AS average\_cost

FROM ServicePlan;

Output :

+------------+--------------+

| total\_cost | average\_cost |

+------------+--------------+

| 199.99 | 199.990000 |

+------------+--------------+

1. SELECT \* FROM Customer WHERE name LIKE 'A%';

Output :

-------------+-------------+---------+--------------+---------+

| customer\_id | name | address | phone\_number | plan\_id |

+-------------+-------------+---------+--------------+---------+

| 1 | Alice Smith | Delhi | 9876543210 | 1 |

+-------------+-------------+---------+--------------+---------+

1. **JOINS**: A join in SQL is an operation that combines rows from two or more tables based on a related column, allowing for the retrieval of related data. The main types of joins include INNER JOIN, which returns only the rows with matching values in both tables; LEFT JOIN (or LEFT OUTER JOIN), which returns all rows from the left table along with matched rows from the right table, filling unmatched rows from the right with NULLs; RIGHT JOIN (or RIGHT OUTER JOIN), which returns all rows from the right table and matched rows from the left, filling unmatched rows from the left with NULLs; and FULL OUTER JOIN, which returns all rows from both tables, filling unmatched rows with NULLs where necessary. These various join types enable complex queries by linking related data across multiple tables, facilitating comprehensive data analysis and reporting.
2. SELECT c.customer\_id, c.name AS customer\_name, sp.plan\_name, sp.cost FROM Customer c INNER JOIN ServicePlan sp ON c.plan\_id = sp.plan\_id;

OUTPUT:

+-------------+---------------+-----------+--------+

| customer\_id | customer\_name | plan\_name | cost |

+-------------+---------------+-----------+--------+

| 1 | Alice Smith | Prepaid | 199.99 |

+-------------+---------------+-----------+--------+

1. SELECT sp.plan\_id, sp.plan\_name, c.name AS customer\_name FROM ServicePlan sp RIGHT JOIN Customer c ON sp.plan\_id = c.plan\_id;

OUTPUT:

+---------+-----------+---------------+

| plan\_id | plan\_name | customer\_name |

+---------+-----------+---------------+

| 1 | Prepaid | Alice Smith |

+---------+-----------+---------------+

1. SELECT c.customer\_id, c.name AS customer\_name, sp.plan\_name FROM Customer c LEFT JOIN ServicePlan sp ON c.plan\_id = sp.plan\_id;

OUTPUT:

+-------------+---------------+-----------+

| customer\_id | customer\_name | plan\_name |

+-------------+---------------+-----------+

| 1 | Alice Smith | Prepaid |

+-------------+---------------+-----------+

1. SELECT c.customer\_id, c.name AS customer\_name, sp.plan\_name FROM Customer c LEFT JOIN ServicePlan sp ON c.plan\_id = sp.plan\_id UNION ALL SELECT c.customer\_id, c.name AS customer\_name, sp.plan\_name FROM Customer c RIGHT JOIN ServicePlan sp ON c.plan\_id = sp.plan\_id WHERE c.customer\_id IS NULL;

OUTPUT:

+-------------+---------------+-----------+

| customer\_id | customer\_name | plan\_name |

+-------------+---------------+-----------+

| 1 | Alice Smith | Prepaid |

+-------------+---------------+-----------+

1. **STRING FUNCTIONS, NUMERIC FUNCTIONS, DATE FUNCTION:** String functions in SQL are used to manipulate and analyze character data, allowing for operations like concatenation (`CONCAT`), substring extraction (`SUBSTRING`), and pattern matching (`LIKE`). Numeric functions perform calculations on numeric data types, including functions like `ROUND`, `SUM`, and `AVG`, which facilitate mathematical operations and aggregations. Date functions enable the manipulation of date and time values, allowing for operations such as extracting parts of a date (`YEAR`, `MONTH`), calculating differences between dates (`DATEDIFF`), and formatting dates (`FORMAT`). Together, these functions provide essential tools for handling and analyzing various types of data in SQL. We will try some coding now.
2. SELECT UPPER(name) AS customer\_name\_upper, LOWER(name) AS customer\_name\_lower, CONCAT(name, ' - ', address) AS customer\_info, LENGTH(name) AS name\_length FROM Customer;

OUTPUT:

+---------------------+---------------------+---------------------+-------------+

| customer\_name\_upper | customer\_name\_lower | customer\_info | name\_length |

+---------------------+---------------------+---------------------+-------------+

| ALICE SMITH | alice smith | Alice Smith - Delhi | 11 |

+---------------------+---------------------+---------------------+-------------+

1. SELECT plan\_name, cost, ROUND(cost) AS rounded\_cost, FLOOR(cost) AS floored\_cost, CEIL(cost) AS ceiled\_cost FROM ServicePlan;

OUTPUT:

+-----------+--------+--------------+--------------+-------------+

| plan\_name | cost | rounded\_cost | floored\_cost | ceiled\_cost |

+-----------+--------+--------------+--------------+-------------+

| Prepaid | 199.99 | 200 | 199 | 200 |

+-----------+--------+--------------+--------------+-------------+

1. ALTER TABLE TelecomCustomer ADD COLUMN created\_at DATETIME DEFAULT NOW()

Describe TelecomCustomer;

OUTPUT:

+--------------+--------------+------+-----+-------------------+-------------------+

| Field | Type | Null | Key | Default | Extra |

+--------------+--------------+------+-----+-------------------+-------------------+

| customer\_id | int | NO | PRI | NULL | auto\_increment |

| name | varchar(100) | NO | | NULL | |

| address | varchar(255) | YES | | NULL | |

| phone\_number | varchar(20) | YES | UNI | NULL | |

| plan\_id | int | YES | MUL | NULL | |

| created\_at | datetime | YES | | CURRENT\_TIMESTAMP | DEFAULT\_GENERATED |

SELECT name, created\_at, DATEDIFF(NOW(), created\_at) AS days\_since\_creation, DATE\_FORMAT(created\_at, '%Y-%m-%d') AS formatted\_creation\_date FROM Customer;

OUTPUT:

+-------------+---------------------+---------------------+-------------------------+

| name | created\_at | days\_since\_creation | formatted\_creation\_date |

+-------------+---------------------+---------------------+-------------------------+

| Alice Smith | 2024-10-16 16:33:27 | 0 | 2024-10-16 |

+-------------+---------------------+---------------------+-------------------------+

1. **NESTED QUERIES/ SUB-QUERY/ SELF QUERY:** Nested queries, or subqueries, are SQL queries embedded within another SQL query. They allow for the retrieval of data based on the results of another query, enabling complex filtering and data manipulation. A nested query can be used in various clauses like `SELECT`, `WHERE`, or `FROM`. For example, a subquery can find customers who have made purchases exceeding a certain amount by first querying the total purchases in one table and then using that result to filter the customers in another table. Nested queries enhance the power of SQL by allowing for more dynamic and refined data retrieval. Now we have some example codes.
2. SELECT \*

FROM TelecomCustomer c1

WHERE plan\_id = (SELECT plan\_id FROM TelecomCustomer WHERE customer\_id = 1);

OUTPUT:

+-------------+-------------+---------+--------------+---------+---------------------+

| customer\_id | name | address | phone\_number | plan\_id | created\_at |

+-------------+-------------+---------+--------------+---------+---------------------+

| 1 | Alice Smith | Delhi | 9876543210 | 1 | 2024-10-16 16:36:41 |

+-------------+-------------+---------+--------------+---------+---------------------+

1. SELECT c.customer\_id, c.name AS customer\_name, sp.plan\_name,

(SELECT COUNT(\*) FROM TelecomCustomer WHERE plan\_id = sp.plan\_id) AS total\_customers

FROM

TelecomCustomer c

JOIN

ServicePlan sp ON c.plan\_id = sp.plan\_id;

OUTPUT:

+-------------+---------------+-----------+-----------------+

| customer\_id | customer\_name | plan\_name | total\_customers |

+-------------+---------------+-----------+-----------------+

| 1 | Alice Smith | Prepaid | 1 |

+-------------+---------------+-----------+-----------------+

1. SELECT c.customer\_id, c.name AS customer\_name, sp.plan\_name FROM TelecomCustome c

JOIN

ServicePlan sp ON c.plan\_id = sp.plan\_id

WHERE

sp.cost > (SELECT AVG(cost) FROM ServicePlan);

1. SELECT c.customer\_id, c.name AS customer\_name FROM TelecomCustomer c JOIN ServicePlan sp ON c.plan\_id = sp.plan\_id WHERE sp.cost > ANY (SELECT cost FROM ServicePlan WHERE cost < 50);
2. **CARTESIAN PRODUCT, DIVISION**: The Cartesian Product is a relational algebra operation that combines all rows from two tables, producing a new table with every possible pair of rows, resulting in a dataset with a total number of rows equal to the product of the row counts of the two tables. This operation is useful for analyzing combinations of data but can lead to large datasets. In contrast, the Division operation identifies tuples in one table that are related to all tuples in another table, effectively filtering records based on multiple criteria. For example, it can be used to find students enrolled in all courses by comparing a `Students` table with a `Courses` table. Together, these operations facilitate complex data retrieval and analysis in relational databases. Now we will implement some codes
3. Cross join:

SELECT \*

FROM TelecomCustomer

CROSS JOIN ServicePlan;

Output :

+-------------+-------------+---------+--------------+---------+---------------------+---------+-----------+--------+------------+

| customer\_id | name | address | phone\_number | plan\_id | created\_at | plan\_id | plan\_name | cost | data\_limit |

+-------------+-------------+---------+--------------+---------+---------------------+---------+-----------+--------+------------+

| 1 | Alice Smith | Delhi | 9876543210 | 1 | 2024-10-16 16:48:49 | 1 | Prepaid | 199.99 | 30 |

+-------------+-------------+---------+--------------+---------+---------------------+---------+-----------+--------+------------+

1. SELECT customer\_id, name FROM TelecomCustomer c WHERE NOT EXISTS (SELECT plan\_id

FROM ServicePlan sp

WHERE NOT EXISTS (

SELECT \*

FROM TelecomCustomer

WHERE TelecomCustomer.plan\_id = sp.plan\_id

AND TelecomCustomer.customer\_id = c.customer\_id

)

);

Output:

+-------------+-------------+

| customer\_id | name |

+-------------+-------------+

| 1 | Alice Smith |

+-------------+-------------+

1. **TCL COMMANDS:** Transaction Control Language (TCL) commands are essential for managing database transactions and ensuring data integrity. The main TCL commands include COMMIT, which saves all changes made during a transaction; ROLLBACK, which undoes changes since the last commit or savepoint; and SAVEPOINT, which marks a point within a transaction for potential rollback. Additionally, SET AUTOCOMMIT controls whether SQL statements are automatically committed. Together, these commands ensure that either all operations succeed or none are applied, maintaining a consistent and reliable database. Starting with examples and output we have:

START TRANSACTION;

SET autocommit = 0;

INSERT INTO TelecomCustomer (name, address, plan\_id, created\_at)

VALUES ('Alice Brown', '789 Pine St', 1, NOW());

SELECT \* FROM TelecomCustomer;

+-------------+-------------+-------------+--------------+---------+---------------------+

| customer\_id | name | address | phone\_number | plan\_id | created\_at |

+-------------+-------------+-------------+--------------+---------+---------------------+

| 1 | Alice Smith | Delhi | 9876543210 | 1 | 2024-10-16 16:56:05 |

| 2 | Alice Brown | 789 Pine St | NULL | 1 | 2024-10-16 16:56:05 |

+-------------+-------------+-------------+--------------+---------+---------------------+

SAVEPOINT before\_second\_insert;

INSERT INTO TelecomCustomer (name, address, plan\_id, created\_at)

VALUES ('Bob White', '321 Maple Ave', 2, NOW());

SELECT \* FROM TelecomCustomer;

ROLLBACK TO before\_second\_insert;

SELECT \* FROM TelecomCustomer;

COMMIT;

SET autocommit = 1;

1. **Views** : A virtual table representing a stored query. It can simplify complex queries and provide a way to restrict access to certain data in the underlying tables.
2. Create a View with All Fields from the Original Table

CREATE VIEW AllCustomers AS

SELECT \* FROM TelecomCustomer;

Output :

+-------------+-------------+

| customer\_id | name |

+-------------+-------------+

| 1 | Alice Smith |

+-------------+-------------+

1. Create a View with Selected Fields Satisfying Certain Conditions

CREATE VIEW PremiumCustomers AS

SELECT customer\_id, name, address

FROM TelecomCustomer

WHERE plan\_id = 1;

describe PremiumCustomers;

output:

+-------------+--------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------------+--------------+------+-----+---------+-------+

| customer\_id | int | NO | | 0 | |

| name | varchar(100) | NO | | NULL | |

| address | varchar(255) | YES | | NULL | |

+-------------+--------------+------+-----+---------+-------+

1. Create a View Using Nested Queries with Two Different Tables

CREATE VIEW CustomerPlans AS

SELECT c.customer\_id, c.name AS customer\_name,

(SELECT sp.plan\_name FROM ServicePlan sp WHERE sp.plan\_id = c.plan\_id) AS plan\_name

FROM TelecomCustomer c;

describe CustomerPlans;

output :

+---------------+--------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------------+--------------+------+-----+---------+-------+

| customer\_id | int | NO | | 0 | |

| customer\_name | varchar(100) | NO | | NULL | |

| plan\_name | varchar(100) | YES | | NULL | |

+---------------+--------------+------+-----+---------+-------+

1. Perform an Equi-Join and Create a View

CREATE VIEW CustomerServicePlans AS

SELECT c.customer\_id, c.name AS customer\_name, sp.plan\_name

FROM TelecomCustomer c

JOIN ServicePlan sp ON c.plan\_id = sp.plan\_id;

describe CustomerServicePlans;

output :

+---------------+--------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------------+--------------+------+-----+---------+-------+

| customer\_id | int | NO | | 0 | |

| customer\_name | varchar(100) | NO | | NULL | |

| plan\_name | varchar(100) | NO | | NULL | |

+---------------+--------------+------+-----+---------+-------+

1. Update a View and Verify Changes in the Master Table

UPDATE AllCustomers

SET name = 'Alice Updated'

WHERE customer\_id = 1;

SELECT \* FROM Customer WHERE customer\_id = 1;

SELECT \* FROM AllCustomers WHERE customer\_id = 1;

Output:

-------------+---------------+---------+--------------+---------+---------------------+

| customer\_id | name | address | phone\_number | plan\_id | created\_at |

+-------------+---------------+---------+--------------+---------+---------------------+

| 1 | Alice Updated | Delhi | 9876543210 | 1 | 2024-10-16 17:08:51 |

+-------------+---------------+---------+--------------+---------+---------------------+

1. Delete Multiple Records from the Master Table and Verify DELETE FROM TelecomCustomer

WHERE customer\_id IN (1, 2, 3);

SELECT \* FROM TelecomCustomer;

SELECT \* FROM AllCustomers;

1. Update the View Using IF-ELSE-Like Condition

UPDATE AllCustomers

SET plan\_id = CASE

WHEN plan\_id = 1 THEN 2

ELSE plan\_id

END

WHERE customer\_id IN (1, 2, 3);

SELECT \* FROM AllCustomers;

**Entity-Relationship Diagram:**

