

**Course Code: DSC 201-3**

**Course Name: *Database Management Systems***

**CIA: 3**

**Topic: Indian Postal Service**

**Submitted to:**

Dr. Ummesalma M.

Associate Professor

**Submitted By:**

**Name:** Delson Bincy Thomas

**Reg No:** 2341310

3 BSc. DM

**Submitted on:** October 2024

**Content**

1. Introduction
2. About the Domain
3. E R Diagram
4. Database Design
5. SQL concepts
6. Conclusion
7. References

**Introduction**

The Indian Postal Service, commonly referred to as **India Post**, is one of the largest and oldest postal services in the world. Established in 1854, it has grown to be the backbone of communication and logistics in India, serving over 1.5 billion people across urban and rural areas. With its vast network of over 150,000 post offices, India Post provides a wide range of services, including letter and parcel deliveries, financial services such as savings accounts and money transfers, and various e-commerce services.

Managing such an extensive network and high volume of transactions requires an efficient data management system to handle information regarding customers, employees, packages, post offices, and services. A **Database Management System (DBMS)** helps streamline operations by efficiently organizing and processing large amounts of data, ensuring smooth postal operations and customer satisfaction. This project aims to design and implement a DBMS that supports the core functions of the Indian Postal Service using SQL concepts, such as Data Definition Language (DDL), Data Manipulation Language (DML), and Transaction Control Language (TCL).

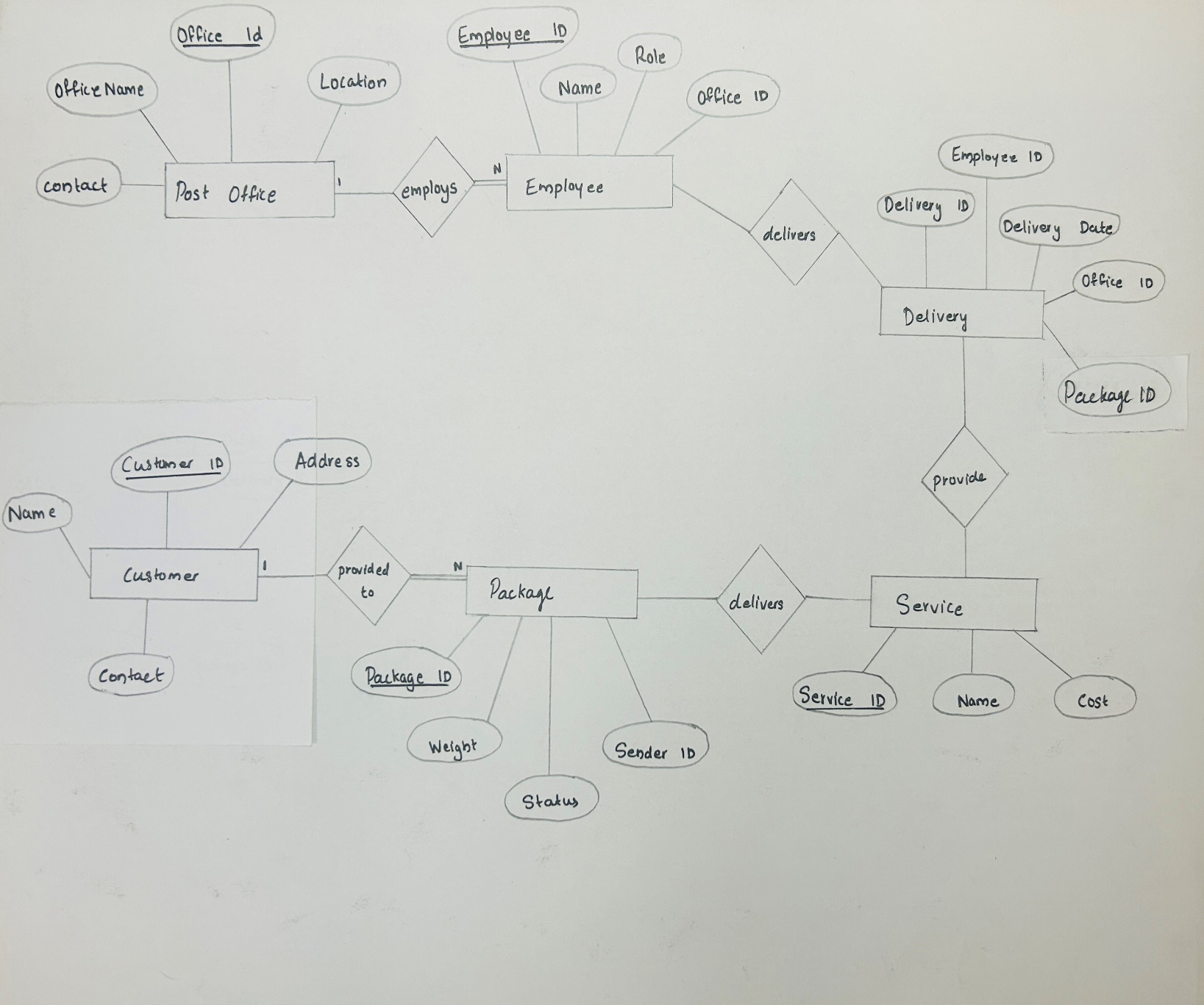
The system will facilitate the management of data regarding post offices, employees, customers, packages, and deliveries. It will provide tools for postal workers to track packages, maintain delivery records, and handle customer information efficiently, enabling India Post to meet the demands of a growing and diversifying market.

**About My Domain: Indian Postal Service**

The **Indian Postal Service**, operated by the Government of India, plays a vital role in connecting people across the country. It serves not only as a mail service but also offers various financial and retail services to millions of citizens. The domain of this project revolves around the logistical and operational side of postal services, specifically focusing on:

* **Mail and Parcel Handling**: Handling both domestic and international mail, with tracking and timely delivery being essential components of the service.
* **Financial Services**: Offering savings schemes, money orders, and banking services, especially in rural areas where banking penetration is low.
* **E-Commerce and Retail Solutions**: India Post also plays a crucial role in last-mile delivery for e-commerce, especially in rural India.
* **Employee Management**: With over 400,000 employees, India Post is one of the largest employers in India. Managing employee records, work assignments, and contact details is crucial for efficient service delivery.

**ER Diagram**

****

**Database Design**

**1.PostOffice Table**

**Attributes**:

* **OfficeID**: Unique identifier for each post office.
* **OfficeName**: Name of the post office.
* **Location**: The geographical location of the office.
* **EstablishedYear**: The year the post office was established.
* **ContactNumber**: Phone number of the post office.

**2. Employee Table**

**Attributes**:

* **EmployeeID**: Unique identifier for each employee.
* **Name**: Name of the employee.
* **Role**: The employee’s role (e.g., Postman, Clerk).
* **Contact**: Contact number of the employee.
* **OfficeID**: References the post office the employee is assigned to.

**3.Package Table**

**Attributes**:

* **PackageID**: Unique identifier for each package.
* **Weight**: The weight of the package.
* **DeliveryStatus**: Current status of the package (Pending, In Transit, Delivered).
* **SenderID**: Reference to the customer sending the package.
* **ReceiverID**: Reference to the customer receiving the package.

**4.Customer Table**

**Attributes**:

* **CustomerID**: Unique identifier for each customer.
* **Name**: Name of the customer.
* **Address**: Postal address of the customer.
* **Contact**: Contact number of the customer.

**5.Delivery Table**

**Attributes**:

* **DeliveryID**: Unique identifier for each delivery.
* **DeliveryDate**: The date of the delivery.
* **OfficeID**: The post office that handled the delivery.
* **EmployeeID**: The employee responsible for the delivery.
* **PackageID**: The package being delivered.

**6. Service Table**

**Attributes**:

* **ServiceID**: Unique identifier for each service.
* **ServiceName**: Name of the service (e.g., Speed Post, Registered Post).
* **Cost**: The cost associated with the service.

**SQL concepts**

***1.Data Definition Language (DDL)***

**DDL** is used to define and manage the structure of database objects such as tables and indexes.

**a) CREATE Command**

The CREATE command is used to create a new table or database object.

**Example**: Creating the PostOffice table.

CREATE TABLE PostOffice (

OfficeID INT AUTO\_INCREMENT PRIMARY KEY,

OfficeName VARCHAR(100) NOT NULL,

Location VARCHAR(100) NOT NULL,

EstablishedYear YEAR,

ContactNumber VARCHAR(15)

);

**Explanation**:

* This command creates a table called PostOffice with columns OfficeID, OfficeName, Location, EstablishedYear, and ContactNumber.
* AUTO\_INCREMENT generates a unique identifier for each record automatically.**Output**:

Query OK, 0 rows affected (0.03 sec)

**b)ALTER Command**

The ALTER command is used to modify an existing table structure by adding, modifying, or dropping columns.

**Example**: Adding an Email column to the PostOffice table.

ALTER TABLE PostOffice ADD COLUMN Email VARCHAR(100);

**Explanation**:

* This command adds a new column Email to the PostOffice table, allowing each post office to store an email address.

**Output**:

Query OK, 0 rows affected (0.02 sec)

Records: 0 Duplicates: 0 Warnings: 0

**c) DROP Command**

The DROP command is used to delete a table or database object permanently.

**Example**: Dropping the PostOffice table.

DROP TABLE PostOffice;

**Explanation**:

* This command removes the entire PostOffice table from the database, including all the data within it.

**Output**:

Query OK, 0 rows affected (0.01 sec)

**d) TRUNCATE Command**

The TRUNCATE command is used to delete all the rows from a table but retain the table structure.

**Example**: Truncating the PostOffice table.

TRUNCATE TABLE PostOffice;

**Explanation**:

* This command deletes all rows in the PostOffice table, but the table itself remains intact.

**Output**:

Query OK, 0 rows affected (0.01 sec)

***2. Data Manipulation Language (DML)***

**DML** commands are used to manipulate the data within database tables, including inserting, updating, and deleting data.

**a) INSERT Command**

The INSERT command is used to add new rows of data to a table.

**Example**: Inserting data into the Customer table.

INSERT INTO Customer (Name, Address, Contact)

VALUES ('John Doe', '123 Main St, Mumbai', '9876543210');

**Explanation**:

* This command inserts a new customer record with the name 'John Doe', address '123 Main St, Mumbai', and contact number '9876543210' into the Customer table.

**Output**:

Query OK, 1 row affected (0.02 sec)

**b) UPDATE Command**

The UPDATE command is used to modify existing data in a table.

**Example**: Updating the contact number of a customer.

UPDATE Customer SET Contact = '9123456789' WHERE CustomerID = 1;

**Explanation**:

* This command updates the contact number for the customer with CustomerID = 1 to '9123456789'.

**Output**:

Query OK, 1 row affected (0.01 sec)

Rows matched: 1 Changed: 1 Warnings: 0

**c) DELETE Command**

The DELETE command is used to remove one or more rows from a table based on a condition.

**Example**: Deleting a customer record.

DELETE FROM Customer WHERE CustomerID = 1;

**Explanation**:

* This command deletes the record of the customer with CustomerID = 1 from the Customer table.

**Output**:

Query OK, 1 row affected (0.01 sec)

**d) SELECT Command**

The SELECT command retrieves data from a table.

**Example**: Retrieving all customer records.

SELECT \* FROM Customer;

**Explanation**:

* This command retrieves all columns for all rows from the Customer table.

**Output**:

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

| CustomerID | Name | Address | Contact |

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

| 1 | John Doe | Main St, Mumbai | 9876543210 |

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

***3.Types of Join***

1. **Inner Join**

An **INNER JOIN** returns only the rows where there is a match between the columns in both tables.

**Example**: Fetch employee names and the post office they are assigned to.

SELECT e.Name, o.OfficeName

FROM Employee e

INNER JOIN PostOffice o ON e.OfficeID = o.OfficeID;

**Explanation**:

* The query joins the Employee and PostOffice tables based on the matching OfficeID.
* Only the employees who have been assigned to a post office (i.e., a match exists) will be shown.

**Output**:

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

| Name | OfficeName |

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

| Alice | Main Post Office |

| Bob | South Branch Office |

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

**b) Left Join (Left Outer Join)**

A Left Join returns all rows from the left table (Employee), and matched rows from the right table (PostOffice). If there is no match, NULL values are returned for columns from the right table.

**Example**: Fetch all employees and their assigned post offices, including those without an assigned office.

SELECT e.Name, o.OfficeName

FROM Employee e

LEFT JOIN PostOffice o ON e.OfficeID = o.OfficeID;

**Explanation**:

* This query shows all employees and their post offices, including employees who are not assigned to any post office (the post office will show as NULL in those cases).

**Output**:

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

| Name | OfficeName |

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

| Alice | Main Post Office |

| Bob | South Branch Office |

| Charlie | NULL |

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

**c) Right Join (Right Outer Join)**

A **RIGHT JOIN** returns all rows from the right table (PostOffice), and the matched rows from the left table (Employee). If there is no match, NULL values are returned for columns from the left table.

**Example**: Fetch all post offices and the employees assigned to them, including offices with no employees.

SELECT e.Name, o.OfficeName

FROM Employee e

RIGHT JOIN PostOffice o ON e.OfficeID = o.OfficeID;

**Explanation**:

* This query shows all post offices and their employees. If a post office has no employees assigned, the employee name will be NULL.

**Output**:

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

| Name | OfficeName |

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

| Alice | Main Post Office |

| Bob | South Branch Office |

| NULL | East Branch Office |

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

**d) Full Outer Join**

A **FULL OUTER JOIN** returns all rows when there is a match in either the left or the right table. If no match is found, NULL values are returned for columns from the unmatched table.

**Note**: MySQL does not support FULL OUTER JOIN natively, but you can achieve the same result using a **UNION** of a **LEFT JOIN** and a **RIGHT JOIN**.

**Example**: Fetch all employees and post offices, even if no match exists.

SELECT e.Name, o.OfficeName

FROM Employee e

LEFT JOIN PostOffice o ON e.OfficeID = o.OfficeID

UNION

SELECT e.Name, o.OfficeName

FROM Employee e

RIGHT JOIN PostOffice o ON e.OfficeID = o.OfficeID;

**Explanation**:

* This query combines the results of a **LEFT JOIN** and a **RIGHT JOIN** to display all employees and post offices, even if they don’t have a match.

**Output**:

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

| Name | OfficeName |

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

| Alice | Main Post Office |

| Bob | South Branch Office |

| Charlie | NULL |

| NULL | East Branch Office |

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

**e) Cross Join**

A **CROSS JOIN** returns the Cartesian product of two tables, meaning it will combine every row from the first table with every row from the second table.

**Example**: Combine all employees with all post offices.

SELECT e.Name, o.OfficeName

FROM Employee e

CROSS JOIN PostOffice o;

**Explanation**:

* Every employee will be combined with every post office, regardless of whether they are assigned to that office.

**Output**:

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

| Name | OfficeName |

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

| Alice | Main Post Office |

| Alice | South Branch Office |

| Alice | East Branch Office |

| Bob | Main Post Office |

| Bob | South Branch Office |

| Bob | East Branch Office |

| Charlie | Main Post Office |

| Charlie | South Branch Office |

| Charlie | East Branch Office |

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

**f) Self Join**

A **SELF JOIN** is a join in which a table is joined with itself. This is useful for hierarchical relationships or when comparing rows in the same table.

**Example**: Fetch employees who work in the same post office.

SELECT e1.Name AS Employee1, e2.Name AS Employee2, o.OfficeName

FROM Employee e1

JOIN Employee e2 ON e1.OfficeID = e2.OfficeID AND e1.EmployeeID <> e2.EmployeeID

JOIN PostOffice o ON e1.OfficeID = o.OfficeID;

**Explanation**:

* This query matches employees who work at the same post office, excluding pairs of the same employee.

**Output**:

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

| Employee1 | Employee2 | OfficeName |

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

| Alice | Bob | Main Post Office |

| Bob | Alice | Main Post Office |

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

***4.Transaction Control Language (TCL*)**

**TCL** commands are used to manage the execution of transactions within the database.

**a) COMMIT Command**

The COMMIT command saves all the changes made during the current transaction to the database.

**Example**: Committing a transaction.

START TRANSACTION;

INSERT INTO Employee (Name, Role, OfficeID) VALUES ('Alice', 'Postman', 1);

COMMIT;

**Explanation**:

* This sequence first starts a transaction, inserts a new employee 'Alice' into the Employee table, and then commits the transaction, making the insertion permanent.

**Output**:

Query OK, 1 row affected (0.01 sec)

**b) ROLLBACK Command**

The ROLLBACK command undoes the changes made during the current transaction.

**Example**: Rolling back a transaction.

START TRANSACTION;

INSERT INTO Employee (Name, Role, OfficeID) VALUES ('Bob', 'Clerk', 1);

ROLLBACK;

**Explanation**:

* This sequence starts a transaction, inserts a new employee 'Bob', but rolls back the transaction, undoing the insertion.

**Output**:

Query OK, 0 rows affected (0.01 sec)

**c) SAVEPOINT Command**

The SAVEPOINT command sets a point within a transaction to which a rollback can be performed.

**Example**: Using a savepoint.

START TRANSACTION;

SAVEPOINT SP1;

INSERT INTO Employee (Name, Role, OfficeID) VALUES ('Charlie', 'Manager', 1);

ROLLBACK TO SP1;

COMMIT;

**Explanation**:

* This sequence creates a savepoint SP1 within a transaction. If the insertion of 'Charlie' is not desired, the transaction can be rolled back to SP1, discarding the insertion but keeping all previous changes.

**Output**:

Query OK, 0 rows affected (0.01 sec)

***5. View Definition Language (VDL*)**

**VDL** commands manage views in the database. Views are virtual tables created by querying one or more tables.

**a) CREATE VIEW Command**

The CREATE VIEW command creates a virtual table based on a SQL query.

**Example**: Creating a view for customer package details.

CREATE VIEW CustomerPackages AS

SELECT c.CustomerID, c.Name, p.PackageID, p.DeliveryStatus

FROM Customer c

JOIN Package p ON c.CustomerID = p.SenderID;

**Explanation**:

* This command creates a view CustomerPackages that shows each customer and their package details, including the delivery status of the packages they sent.

**Output**:

Query OK, 0 rows affected (0.01 sec)

**b) DROP VIEW Command**

The DROP VIEW command removes a view from the database.

**Example**: Dropping the CustomerPackages view.

DROP VIEW CustomerPackages;

**Explanation**:

* This command deletes the CustomerPackages view, so it can no longer be used.

**Output**:

Query OK, 0 rows affected (0.01 sec)

**Conclusion**

This project focused on building a comprehensive Database Management System (DBMS) for the Indian Postal Service, demonstrating how SQL can effectively manage data related to post offices, employees, customers, packages, and deliveries. By applying core SQL concepts, including DDL, DML, TCL, and VDL, the project showcased the design, creation, and management of relational databases.

Using DDL, we created and structured database tables such as `PostOffice`, `Employee`, and `Package`. Through ML commands, we performed data manipulation tasks like inserting, updating, and deleting records. TCL commands were used to ensure that transactions were managed securely and efficiently, allowing us to commit or rollback changes as needed. We also demonstrated how VDL commands create and manage views to simplify data retrieval and enhance query performance.

Additionally, the use of various types of Joins (INNER, LEFT, RIGHT) illustrated how to fetch data from multiple related tables, providing insights into the relationships between entities like customers, packages, and post offices. This project highlighted the practical significance of SQL in optimizing the management of large datasets in a real-world system like the Indian Postal Service.

Overall, this project reinforced the importance of structured and efficient data management in organizations, and showcased the power of SQL in maintaining data integrity, optimizing queries, and supporting the operations of large-scale services.

**References**

1. **Indian Postal Service - Government of India**  
   Official website of India Post.  
   Retrieved from: <https://www.indiapost.gov.in>
2. **Database System Concepts**  
   Abraham Silberschatz, Henry F. Korth, S. Sudarshan (7th Edition).  
   McGraw-Hill Education, 2019.
3. **SQL: The Complete Reference**  
   James R. Groff and Paul N. Weinberg (3rd Edition).  
   McGraw-Hill Education, 2014.
4. **W3Schools - SQL Tutorial**  
   A comprehensive resource for learning SQL.  
   Retrieved from: <https://www.w3schools.com/sql/>
5. **Stack Overflow - SQL Operations**  
   Community discussions and best practices for SQL queries and database management.  
   Retrieved from: <https://stackoverflow.com/>