**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering, Mumbai Campus**

## A.Y. 2023 - 2024

**Course: Database Management Systems Project Report**

|  |  |  |
| --- | --- | --- |
| Program: | MBA Tech. Computer Engineering | |
| Semester | IV | |
| Name of the Project: | Internet Service Provider Database Management System - A Lens Into The Free World | |
|  | | |
| Details of Project Members |  |  |
| Batch: | Roll No. | Name |
| E2 | N114 | Kanwaljit Singh |
| E2 | N121 | Sahil Sanap |
| E2 | N124 | Shreyas Kausik Prajapati |
| Date of Submission: 28th March 2024 | | |

**Contribution of each project Members:**

|  |  |  |
| --- | --- | --- |
| Roll No. | Name: | Contribution |
| N114 | Kanwaljit Singh | ER diagram, Queries |
| N121 | Sahil Sanap | Schema Diagram, PPT |
| N124 | Shreyas Kausik Prajapati | Normalisation, Report |

**Project Report on:**

Internet Service Provider Database Management System - A Lens Into The Free World

by

***Kanwaljit Sing, N117***

***Sahil Sanap, N121***

***Shreyas Kaushik Prajapati, N124***

**Course: DBMS AY: 2023-24**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Sr no.** | **Topic** | **Page no.** |
| **1** | Introduction |  |
| **2** | Database Design Make-Up |  |
| **3** | Entity Relationship Diagram |  |
| **4** | Relational Model |  |
| **5** | Normalization |  |
| **6** | SQL Queries |  |
| **7** | Learning from the Project |  |
| **8** | Challenges Faced |  |
| **9** | Conclusion |  |

# Introduction

In the digital era,

efficient management of data is crucial for businesses to thrive. Database Management Systems (DBMS) play a pivotal role in organizing, storing, retrieving, and managing vast amounts of data effectively.

This project aims to design and implement a comprehensive DBMS tailored to the needs of an ISP company. By leveraging database technologies, the company can streamline its operations, enhance decision-making processes, and improve overall efficiency.

# Components of Database Design

The entities with their attributes are as follows:

* 1. Customer

|  |  |
| --- | --- |
| * + - CustomerID | * + - varchar(25) PK |
| * + - CustomerAddress | * + - varchar(50) |
| * + - CustomerName | * + - varchar(50) |
| * + - CustomerContact | * + - varchar(50) |

* 1. Address

|  |  |
| --- | --- |
| * + - AddressID | * + - varchar(25) PK |
| * + - CustomerID | * + - varchar(25) |
| * + - Street | * + - varchar(50) |
| * + - City | * + - varchar(50) |
| * + - Location | * + - varchar(50) |
| * + - PinCode | * + - varchar(50) |
| * + - Area | * + - varchar(50) |

* 1. Account

|  |  |
| --- | --- |
| * + - Account | * + - varchar(25) PK |
| * + - CustomerID | * + - varchar(25) |
| * + - BillingInfo | * + - varchar(255) |
| * + - AccountType | * + - varchar(50) |

* 1. Accountplandetails

|  |  |
| --- | --- |
| * + - PlanID | * + - int AI PK |
| * + - AccountID | * + - varchar(25) |
| * + - PlanName | * + - varchar(50) |
| * + - PlanDescription | * + - varchar(255) |

* 1. Invoice

|  |  |
| --- | --- |
| * + - InvoiceID | * + - varchar(25) PK |
| * + - AccountID | * + - varchar(25) |
| * + - InvoiceDate | * + - date |
| * + - InvoiceTotalAmount | * + - int |

* 1. Payment

|  |  |
| --- | --- |
| * + - PaymentID | * + - varchar(25) PK |
| * + - AccountID | * + - varchar(25) |
| * + - PaymentDueDate | * + - date |
| * + - PaymentAmount | * + - int |

* 1. Subscription

|  |  |
| --- | --- |
| * + - SubscriptionID | * + - varchar(25) PK |
| * + - AccountID | * + - varchar(25) |
| * + - SubscriptionStartDate | * + - date |
| * + - SubscriptionEndDate | * + - date |
| * + - SubscriptionIDataLimit | * + - varchar(50) |
| * + - SubscriptionCost | * + - int |

* 1. Router

|  |  |
| --- | --- |
| * + - RouterID | * + - varchar(25) PK |
| * + - CustomerID | * + - varchar(25) |
| * + - MACAddress | * + - varchar(20) |
| * + - RouterModel | * + - varchar(25) |
| * + - RouterInstallationDate | * + - date |

* 1. NetworkNodes

|  |  |
| --- | --- |
| * + - NodeID | * + - int PK |
| * + - RouterID | * + - varchar(25) |
| * + - ConnectionType | * + - varchar(25) |
| * + - LOCATION | * + - varchar(50) |

* 1. ServiceRequest

|  |  |
| --- | --- |
| * + - ServiceRequestID | * + - varchar(25) PK |
| * + - CustomerID | * + - varchar(25) |
| * + - ReqDate | * + - date |
| * + - ServiceStatus | * + - varchar(25) |
| * + - ServiceDescription | * + - varchar(10 |

* 1. Tickets

|  |  |
| --- | --- |
| * + - TicketID | * + - varchar(25) PK |
| * + - ServiceRequestID | * + - varchar(25) |
| * + - TicketStatus | * + - varchar(25) |
| * + - TicketDescription | * + - varchar(25) |
| * + - TicketPriority | * + - int |
| * + - TicketDate | * + - date |

Relationship between the tables in the database:

1. Customer and Address:

- One-to-Many Relationship: Each customer can have one or more addresses (e.g., home address, billing address).

- Foreign Key: Address table's CustomerID column references the Customer table's CustomerID column.

2. Customer and Account:

- One-to-Many Relationship: Each customer can have one or more accounts.

- Foreign Key: Account table's CustomerID column references the Customer table's CustomerID column.

3. Customer and Router:

- One-to-Many Relationship: Each customer can have one or more routers installed.

- Foreign Key: Router table's CustomerID column references the Customer table's CustomerID column.

4. Router and NetworkNodes:

- One-to-Many Relationship: Each router can have one or more network nodes connected to it.

- Foreign Key: NetworkNodes table's RouterID column references the Router table's RouterID column.

5. Account and Subscription:

- One-to-Many Relationship: Each account can have one or more subscriptions.

- Foreign Key: Subscription table's AccountID column references the Account table's Account column.

6. Customer and ServiceRequest:

- One-to-Many Relationship: Each customer can raise one or more service requests.

- Foreign Key: ServiceRequest table's CustomerID column references the Customer table's CustomerID column.

7. ServiceRequest and Tickets:

- One-to-Many Relationship: Each service request can have one or more associated tickets.

- Foreign Key: Tickets table's ServiceRequestID column references the ServiceRequest table's ServiceRequestID column.

8. Account and Payment:

- One-to-Many Relationship: Each account can have one or more payments.

- Foreign Key: Payment table's AccountID column references the Account table's Account column.

9. Account and Invoice:

- One-to-Many Relationship: Each account can have one or more invoices.

- Foreign Key: Invoice table's AccountID column references the Account table's Account column.

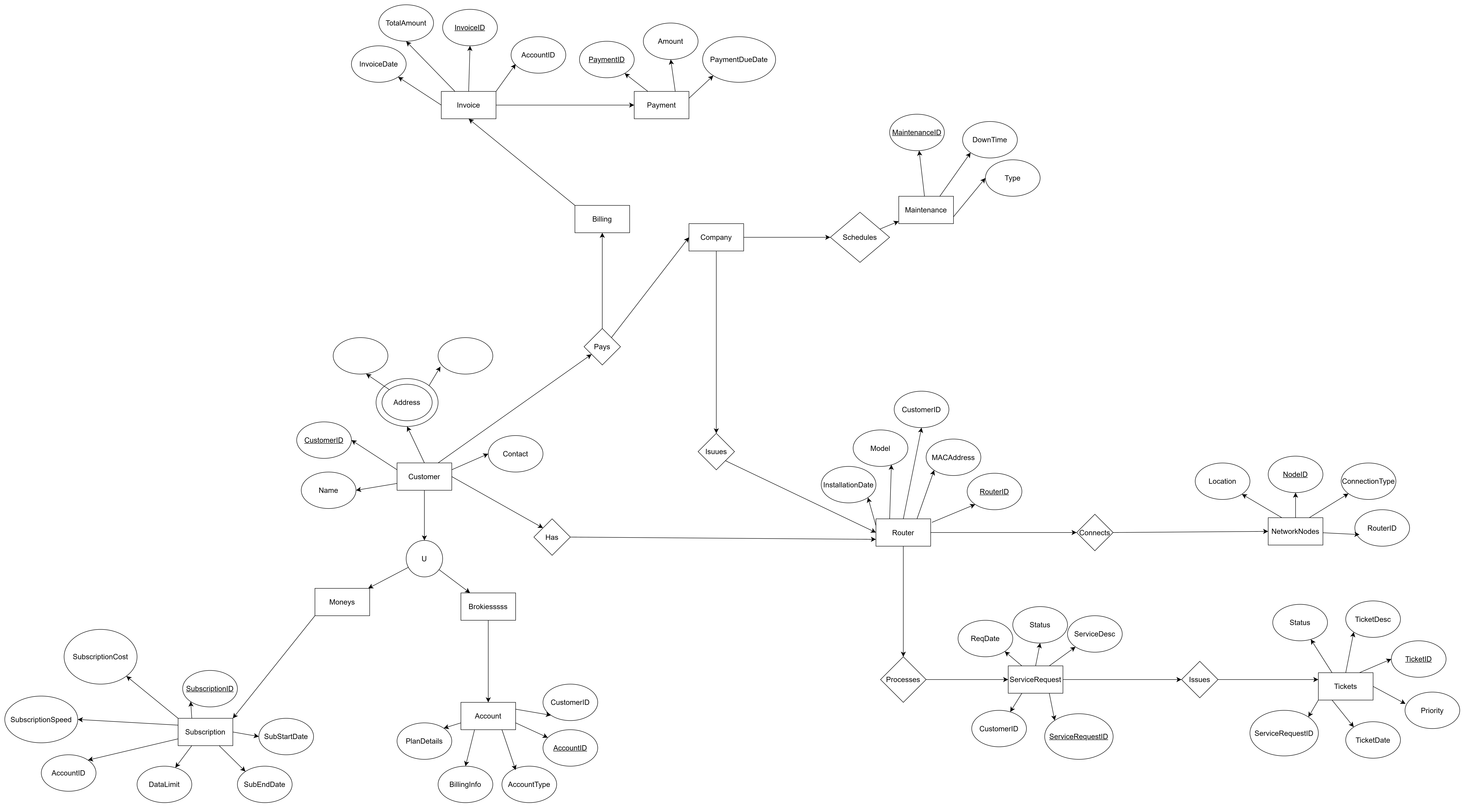
1. Account and AccountPlanDetails:

- One-to-Many Relationship: Each account can have one or more account plan details.

- Foreign Key: AccountPlanDetails table's AccountID column references the Account table's Account column.

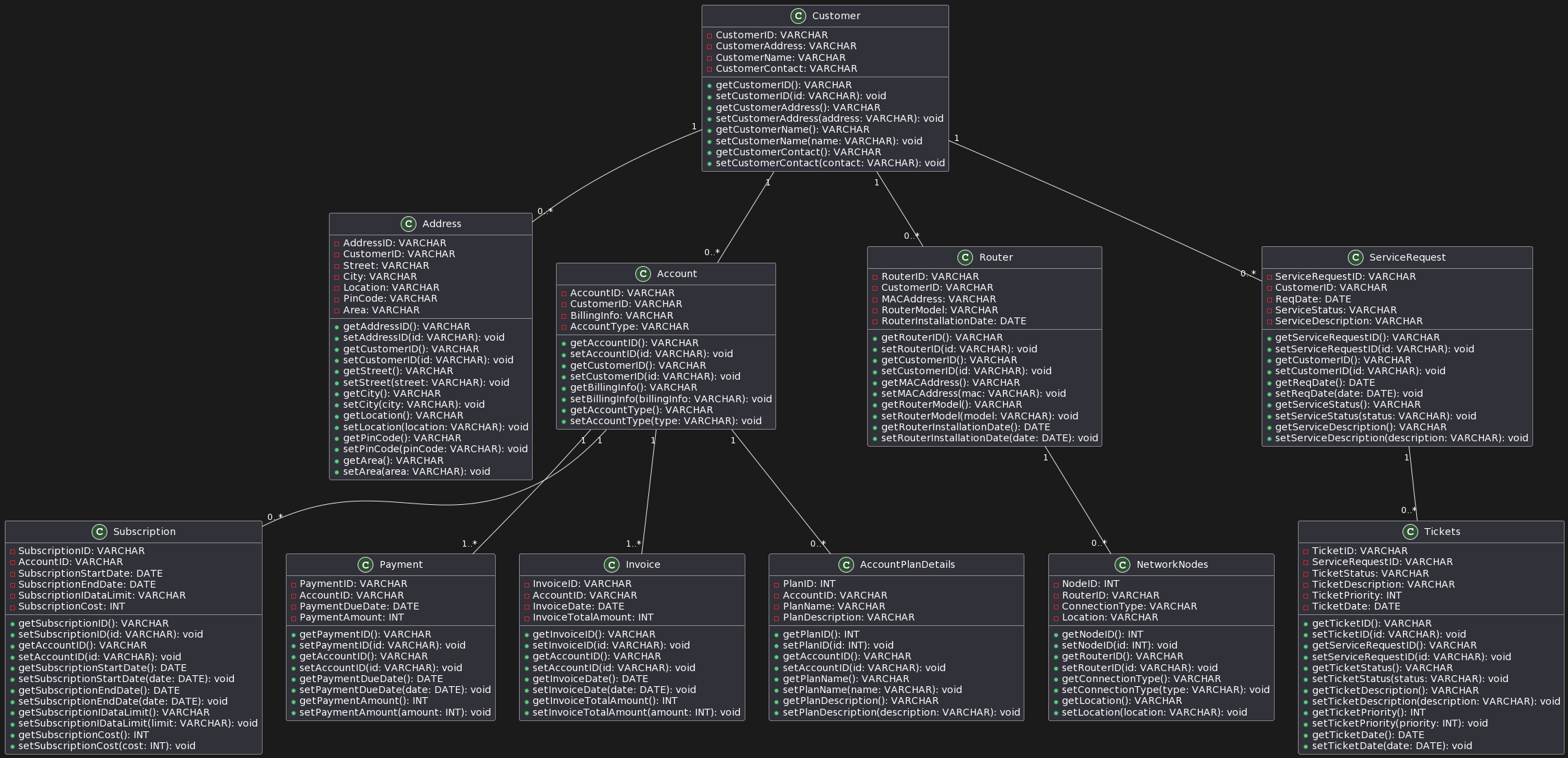
* These relationships define how data in different tables is related to each other, allowing for efficient retrieval and management of information within the database.

# Entity Relationship Diagram



1. **Relational Model**

On converting the EER to a relational model, we obtained the following schema:



# Normalization

In our project,

-- Normalization

-- 1NF to 3NF

-- We are moving from the original form (which can be considered a form of 1NF where AccountPlanDetails is a multi-valued attribute) to at least the

-- Third Normal Form (3NF) by splitting the AccountPlanDetails into a separate table.

-- Step 1: Create a new table for account plan details

CREATE TABLE AccountPlanDetails (

PlanID INT AUTO\_INCREMENT PRIMARY KEY,

AccountID VARCHAR(25),

PlanName VARCHAR(50),

PlanDescription VARCHAR(255),

FOREIGN KEY (AccountID) REFERENCES Account (Account)

);

SELECT \* FROM Account;

-- Step 2: Remove AccountPlanDetails column from the Account table

ALTER TABLE Account DROP COLUMN AccountPlanDetails;

-- Step 3: Add foreign key constraint to link AccountPlanDetails with Account

ALTER TABLE AccountPlanDetails ADD CONSTRAINT FK\_AccountPlanDetails\_Account FOREIGN KEY (AccountID) REFERENCES Account (Account);

-- Insert sample data into the AccountPlanDetails table

INSERT INTO AccountPlanDetails (AccountID, PlanName, PlanDescription)

VALUES

('ACC001', 'Unlimited Data', 'xyz'),

('ACC002', 'Basic Plan', 'abc'),

('ACC003', 'Premium Plan', 'ask'),

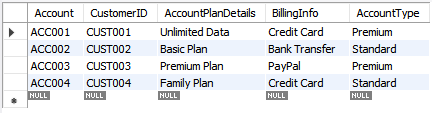
('ACC004', 'Family Plan', 'aws');

SELECT \* FROM AccountPlanDetails;

SELECT \* FROM Account;

Before Splitting the tables:

Table Account



After Splitting the tables:

Table AccountPlanDetails

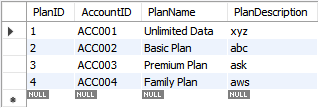
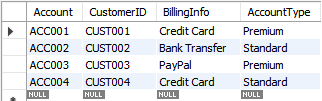


Table Account



Thus, the table has been Normalized into 3NF form.

# SQL Queries

## Behold,

## The code:

CREATE DATABASE project\_new;

USE project\_new;

CREATE TABLE Customer (

CustomerID VARCHAR(25) PRIMARY KEY,

CustomerAddress VARCHAR(50),

CustomerName VARCHAR(50),

CustomerContact VARCHAR(50)

);

CREATE TABLE Account (

Account VARCHAR(25) PRIMARY KEY,

CustomerID VARCHAR(25),

AccountPlanDetails VARCHAR(50),

BillingInfo VARCHAR(255),

AccountType VARCHAR(50),

FOREIGN KEY (CustomerID) REFERENCES Customer (CustomerID)

);

CREATE TABLE Router (

RouterID VARCHAR(25) PRIMARY KEY,

CustomerID VARCHAR(25),

MACAddress VARCHAR(20),

RouterModel VARCHAR(25),

RouterInstallationDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customer (CustomerID)

);

CREATE TABLE NetworkNodes (

NodeID INT(25) PRIMARY KEY,

RouterID VARCHAR(25),

ConnectionType VARCHAR(25),

LOCATION VARCHAR(50),

FOREIGN KEY (RouterID) REFERENCES Router (RouterID)

);

CREATE TABLE Subscription (

SubscriptionID VARCHAR(25) PRIMARY KEY,

AccountID VARCHAR(25),

SubscriptionStartDate DATE,

SubscriptionEndDate DATE,

SubscriptionIDataLimit VARCHAR(50),

SubscriptionCost INT(50),

FOREIGN KEY (AccountID) REFERENCES Account (Account)

);

CREATE TABLE ServiceRequest (

ServiceRequestID VARCHAR(25) PRIMARY KEY,

CustomerID VARCHAR(25),

ReqDate DATE,

ServiceStatus VARCHAR(25),

ServiceDescription VARCHAR(100),

FOREIGN KEY (CustomerID) REFERENCES Customer (CustomerID)

);

CREATE TABLE Tickets (

TicketID VARCHAR(25) PRIMARY KEY,

ServiceRequestID VARCHAR(25),

TicketStatus VARCHAR(25),

TicketDescription VARCHAR(25),

TicketPriority INT(5),

TicketDate DATE,

FOREIGN KEY (ServiceRequestID) REFERENCES ServiceRequest (ServiceRequestID)

);

CREATE TABLE Payment (

PaymentID VARCHAR(25) PRIMARY KEY,

AccountID VARCHAR(25),

PaymentDueDate DATE,

PaymentAmount INT(50),

FOREIGN KEY (AccountID) REFERENCES Account (Account)

);

CREATE TABLE Invoice (

InvoiceID VARCHAR(25) PRIMARY KEY,

AccountID VARCHAR(25),

InvoiceDate DATE,

InvoiceTotalAmount INT(50),

FOREIGN KEY (AccountID) REFERENCES Account (Account)

);

CREATE TABLE Address (

AddressID VARCHAR(25) PRIMARY KEY,

CustomerID VARCHAR(25),

Street VARCHAR(50),

City VARCHAR(50),

Location VARCHAR(50),

PinCode VARCHAR(50),

Area VARCHAR(50),

FOREIGN KEY (CustomerID) REFERENCES Customer (CustomerID)

);

-- Additional data for the Customer table

INSERT INTO Customer (CustomerID, CustomerAddress, CustomerName, CustomerContact)

VALUES

('CUST001', '123 Main St', 'John Doe', '123-456-7890'),

('CUST002', '456 Oak St', 'Jane Smith', '987-654-3210'),

('CUST003', '789 Pine St', 'Alice Johnson', '555-123-4567'),

('CUST004', '321 Cedar St', 'Bob Miller', '555-987-6543');

-- Additional data for the Address table

INSERT INTO Address (AddressID, CustomerID, Street, City, Location, PinCode, Area)

VALUES

('ADDR001', 'CUST001', '123 Main St', 'City1', 'Location1', '12345', 'Downtown'),

('ADDR002', 'CUST002', '456 Oak St', 'City2', 'Location2', '54321', 'Suburbia'),

('ADDR003', 'CUST003', '789 Pine St', 'City2', 'Location2', '54321', 'Suburbia'),

('ADDR004', 'CUST004', '321 Cedar St', 'City1', 'Location1', '12345', 'Downtown');

-- Additional data for the Account table

INSERT INTO Account (Account, CustomerID, AccountPlanDetails, BillingInfo, AccountType)

VALUES

('ACC001', 'CUST001', 'Unlimited Data', 'Credit Card', 'Premium'),

('ACC002', 'CUST002', 'Basic Plan', 'Bank Transfer', 'Standard'),

('ACC003', 'CUST003', 'Premium Plan', 'PayPal', 'Premium'),

('ACC004', 'CUST004', 'Family Plan', 'Credit Card', 'Standard');

-- Additional data for the Router table

INSERT INTO Router (RouterID, CustomerID, MACAddress, RouterModel, RouterInstallationDate)

VALUES

('ROUT001', 'CUST001', '12:34:56:78:90:AB', 'Model X', '2023-05-01'),

('ROUT002', 'CUST002', 'AB:CD:EF:12:34:56', 'Model Y', '2023-06-01'),

('ROUT003', 'CUST003', 'EF:CD:AB:90:78:56', 'Model Z', '2023-07-01'),

('ROUT004', 'CUST004', '56:34:12:AB:EF:CD', 'Model X', '2023-07-15');

-- Additional data for the NetworkNodes table

INSERT INTO NetworkNodes (NodeID, RouterID, ConnectionType, LOCATION)

VALUES

(1, 'ROUT001', 'Wired', 'Living Room'),

(2, 'ROUT002', 'Wireless', 'Bedroom'),

(3, 'ROUT003', 'Wireless', 'Office'),

(4, 'ROUT004', 'Wired', 'Living Room');

-- Additional data for the Subscription table

INSERT INTO Subscription (SubscriptionID, AccountID, SubscriptionStartDate, SubscriptionEndDate, SubscriptionIDataLimit, SubscriptionCost)

VALUES

('SUBS001', 'ACC001', '2023-05-01', '2024-05-01', 'Unlimited', 1200),

('SUBS002', 'ACC002', '2023-06-01', '2024-06-01', 'Basic', 600),

('SUBS003', 'ACC003', '2023-07-01', '2024-07-01', 'Basic', 900),

('SUBS004', 'ACC004', '2023-07-15', '2024-07-15', 'Unlimited', 1200);

-- Additional data for the ServiceRequest table

INSERT INTO ServiceRequest (ServiceRequestID, CustomerID, ReqDate, ServiceStatus, ServiceDescription)

VALUES

('SRVREQ001', 'CUST001', '2023-05-01', 'Open', 'Internet connectivity issue'),

('SRVREQ002', 'CUST002', '2023-06-01', 'Closed', 'Router setup'),

('SRVREQ003', 'CUST003', '2023-07-01', 'Open', 'Internet speed issue'),

('SRVREQ004', 'CUST004', '2023-07-15', 'Closed', 'Router firmware update');

-- Additional data for the Tickets table

INSERT INTO Tickets (TicketID, ServiceRequestID, TicketStatus, TicketDescription, TicketPriority, TicketDate)

VALUES

('TICK001', 'SRVREQ001', 'Open', 'Issue with connection', 1, '2023-05-01'),

('TICK002', 'SRVREQ002', 'Closed', 'Router setup', 2, '2023-06-01'),

('TICK003', 'SRVREQ003', 'Open', 'Slow connection', 1, '2023-07-01'),

('TICK004', 'SRVREQ004', 'Closed', 'Router upgrade', 2, '2023-07-15');

-- Additional data for the Payment table

INSERT INTO Payment (PaymentID, AccountID, PaymentDueDate, PaymentAmount)

VALUES

('PAY001', 'ACC001', '2023-05-31', 100),

('PAY002', 'ACC002', '2023-06-15', 50),

('PAY003', 'ACC003', '2023-07-01', 80),

('PAY004', 'ACC004', '2023-07-15', 60);

-- Additional data for the Invoice table

INSERT INTO Invoice (InvoiceID, AccountID, InvoiceDate, InvoiceTotalAmount)

VALUES

('INV001', 'ACC001', '2023-05-01', 120),

('INV002', 'ACC002', '2023-06-01', 60),

('INV003', 'ACC003', '2023-07-01', 85),

('INV004', 'ACC004', '2023-07-15', 65);

-- Show All Tables

SHOW TABLES;

-- SELECT all records from the Customer Table

SELECT \* FROM Customer;

-- SELECT all records from the Account Table

SELECT \* FROM Account;

-- SELECT all records from the Router Table

SELECT \* FROM Router;

-- SELECT all records from the NetworkNodeSs Table

SELECT \* FROM NetworkNodes;

-- SELECT all records from the Subscription Table

SELECT \* FROM Subscription;

-- SELECT all records from the ServiceRequest Table

SELECT \* FROM ServiceRequest;

-- SELECT all records from the Tickets Table

SELECT \* FROM Tickets;

-- SELECT all records from the Payment Table

SELECT \* FROM Payment;

-- SELECT all records from the Invoice Table

SELECT \* FROM Invoice;

-- SELECT all records from the Address Table

SELECT \* FROM Address;

-- Normalization

-- 1NF to 3NF

-- We are moving from the original form (which can be considered a form of 1NF where AccountPlanDetails is a multi-valued attribute) to at least the

-- Third Normal Form (3NF) by splitting the AccountPlanDetails into a separate table.

-- Step 1: Create a new table for account plan details

CREATE TABLE AccountPlanDetails (

PlanID INT AUTO\_INCREMENT PRIMARY KEY,

AccountID VARCHAR(25),

PlanName VARCHAR(50),

PlanDescription VARCHAR(255),

FOREIGN KEY (AccountID) REFERENCES Account (Account)

);

SELECT \* FROM Account;

-- Step 2: Remove AccountPlanDetails column from the Account table

ALTER TABLE Account DROP COLUMN AccountPlanDetails;

-- Step 3: Add foreign key constraint to link AccountPlanDetails with Account

ALTER TABLE AccountPlanDetails ADD CONSTRAINT FK\_AccountPlanDetails\_Account FOREIGN KEY (AccountID) REFERENCES Account (Account);

-- Insert sample data into the AccountPlanDetails table

INSERT INTO AccountPlanDetails (AccountID, PlanName, PlanDescription)

VALUES

('ACC001', 'Unlimited Data', 'xyz'),

('ACC002', 'Basic Plan', 'abc'),

('ACC003', 'Premium Plan', 'ask'),

('ACC004', 'Family Plan', 'aws');

SELECT \* FROM AccountPlanDetails;

-- Data Retrieval Queries

SELECT \* FROM Customer WHERE CustomerID = 'CUST001';

SELECT \* FROM Account;

SELECT \* FROM Router;

SELECT \* FROM Subscription;

SELECT \* FROM ServiceRequest;

-- Data Analysis Queries

SELECT SUM(SubscriptionCost) AS TotalRevenue FROM Subscription;

SELECT ServiceDescription, COUNT(\*) AS RequestCount FROM ServiceRequest GROUP BY ServiceDescription ORDER BY RequestCount DESC;

SELECT YEAR(PaymentDueDate) AS PaymentYear, MONTH(PaymentDueDate) AS PaymentMonth, SUM(PaymentAmount) AS TotalPayments FROM Payment GROUP BY PaymentYear, PaymentMonth;

-- Data Modification Queries

INSERT INTO Customer (CustomerID, CustomerName, CustomerAddress, CustomerContact) VALUES ('CUST005', 'Emily Johnson', '789 Elm St', '555-678-9012');

UPDATE Customer SET CustomerContact = '555-111-2222' WHERE CustomerID = 'CUST002';

START TRANSACTION;

DELETE FROM Tickets WHERE ServiceRequestID = 'SRVREQ004';

DELETE FROM ServiceRequest WHERE ServiceRequestID = 'SRVREQ004';

-- ROLLBACK

COMMIT;

-- Join Queries

SELECT c.\*, a.\* FROM Customer c INNER JOIN Account a ON c.CustomerID = a.Account;

-- SELECT c.\*, a.\* FROM Customer c INNER JOIN Account a ON c.CustomerID = a.Account WHERE ap.AccountPlanDetails ='Unlimited Data';

SELECT c.\*, a.\* FROM Customer c INNER JOIN Account a ON c.CustomerID = a.Account WHERE c.CustomerID ='CUST001';

SELECT \* FROM Customer NATURAL JOIN Account NATURAL JOIN AccountPlanDetails;

-- Aggregate Queries

SELECT SUM(InvoiceTotalAmount) AS TotalRevenue FROM Invoice;

SELECT AVG(SubscriptionCost) AS AverageSubscriptionCost FROM Subscription;

-- Filtering and Sorting Queries

SELECT \* FROM ServiceRequest WHERE ServiceStatus = 'Open';

SELECT \* FROM Invoice ORDER BY InvoiceDate DESC;

-- Grouping Queries

-- Subqueries

SELECT \* FROM Customer WHERE CustomerID IN (SELECT CustomerID FROM Subscription WHERE SubscriptionCost > (SELECT AVG(SubscriptionCost) FROM Subscription));

-- SELECT DISTINCT s.SubscriptionCost, c.\* FROM Subscription s INNER JOIN Account a ON s.AccountID = a.CustomerID INNER JOIN Customer c ON c.CustomerID = a.CustomerID WHERE SubscriptionCost > (SELECT AVG(SubscriptionCost) FROM Subscription);

SELECT DISTINCT s.SubscriptionCost, c.\* FROM Subscription s INNER JOIN Customer c ON c.CustomerID = s.AccountID WHERE SubscriptionCost > (SELECT AVG(SubscriptionCost) FROM Subscription);

SELECT AVG(SubscriptionCost) FROM Subscription;

SELECT \* FROM Subscription;

-- Revised Subquery

-- -- SELECT c.\*, s.SubscriptionCost FROM Customer c INNER JOIN Subscription s ON c.CustomerID = s.CustomerID WHERE s.SubscriptionCost > (SELECT AVG(SubscriptionCost) FROM Subscription);

-- -- SELECT c.\*, s.SubscriptionCost FROM Customer c INNER JOIN Account a ON c.CustomerID = a.CustomerID WHERE a.CustomerID = (SELECT s.AccountID FROM Subscription s INNER JOIN Account a on s.AccountID = a.CustomerID WHERE s.SubscriptionCost > (SELECT AVG(SubscriptionCost) FROM Subscription);

-- SELECT c.\*, s.SubscriptionCost FROM Customer c INNER JOIN Account a ON c.CustomerID = a.CustomerID WHERE a.CustomerID = (SELECT s.AccountID FROM Subscription s INNER JOIN Account a on s.AccountID = a.Account);

-- SELECT c.\*, s.SubscriptionCost FROM Customer c INNER JOIN Account a ON c.CustomerID = a.CustomerID INNER JOIN Subscription s ON a.CustomerID = s.AccountID WHERE s.SubscriptionCost = (SELECT AVG(SubscriptionCost) FROM Subscription);

-- Views

-- View to Display Customer Details with Addresses

CREATE VIEW CustomerDetails AS SELECT c.\*, a.Street, a.City, a.Location, a.PinCode, a.Area FROM Customer c LEFT JOIN Address a ON c.CustomerID = a.CustomerID;

SELECT \* FROM CustomerDetails;

DROP VIEW CustomerDetails;

-- View to Show Account Subscriptions

CREATE VIEW AccountSubscriptions AS

SELECT a.\*, s.\*

FROM Account a

LEFT JOIN Subscription s ON a.Account = s.AccountID;

SELECT \* FROM AccountSubscriptions;

DROP VIEW AccountSubscriptions;

-- View to List Service Requests with Customer Information

CREATE VIEW ServiceRequestsView AS

SELECT sr.\*, c.CustomerName, c.CustomerContact

FROM ServiceRequest sr

LEFT JOIN Customer c ON sr.CustomerID = c.CustomerID;

SELECT \* FROM ServiceRequestsView;

DROP VIEW ServiceRequestsView;

-- View to Display Payments with Account Information

CREATE VIEW PaymentsView AS

SELECT p.\*, a.AccountType, ap.PlanID, ap.PlanName, ap.PlanDescription

FROM Payment p

LEFT JOIN Account a ON p.AccountID = a.Account NATURAL JOIN AccountPlanDetails ap;

SELECT \* FROM PaymentsView;

DROP VIEW PaymentsView;

-- View to Show Invoices with Customer Details

CREATE VIEW InvoiceDetails AS

SELECT i.\*, c.CustomerName, c.CustomerAddress

FROM Invoice i

LEFT JOIN Account a ON i.AccountID = a.Account

LEFT JOIN Customer c ON a.CustomerID = c.CustomerID;

SELECT \* FROM InvoiceDetails;

DROP VIEW InvoiceDetails;

-- Advanced Join with Aggregation

SELECT

c.CustomerID,

c.CustomerName,

COUNT(s.SubscriptionID) AS NumSubscriptions,

SUM(s.SubscriptionCost) AS TotalSubscriptionCost

FROM

Customer c

LEFT JOIN

Account a ON c.CustomerID = a.CustomerID

LEFT JOIN

Subscription s ON a.Account = s.AccountID

GROUP BY

c.CustomerID, c.CustomerName

ORDER BY

TotalSubscriptionCost DESC;

-- Window Functions

SELECT

c.CustomerID,

CustomerName,

CustomerAddress,

ROW\_NUMBER() OVER (PARTITION BY CustomerID ORDER BY ServiceRequestID) AS RequestSequence

FROM

Customer c

JOIN

ServiceRequest sr ON c.CustomerID = sr.CustomerID;

-- ---------------------------------------------------------------------

-- -- Drop Tables

-- DROP TABLE Tickets;

-- DROP TABLE ServiceRequest;

-- DROP TABLE NetworkNodes;

-- DROP TABLE Router;

-- DROP TABLE Subscription;

-- DROP TABLE Payment;

-- DROP TABLE Invoice;

-- DROP TABLE AccountPlanDetails;

-- DROP TABLE Account;

-- DROP TABLE Address;

-- DROP TABLE Customer;

-- DROP DATABASE project\_new;

# Learning from the project

* Learnt about normalisation and its types, 1NF, 2NF, and 3NF. Also learnt how to normalize tables and restructure our dataset with appropriate columns and data types.

## Learnt about joins, views, nested clauses, subqueries, and keywords such as AVG, UPPER, SUBSTRING, etc

* Learnt about joins and how they can be used to manipulate tables. They can be used to create relations between tables and obtain specific rows or columns from selective tables where the specific conditions are met. There are many types of JOIN statements, mainly NATURAL JOIN, INNER JOIN, OUTER JOIN which can be LEFT or RIGHT.

## Learnt to use SQL and Workbench to effectively handle and manipulate data and tables through queries. Optimization of queries along with the correct syntax leads to the most efficient performance. However, at times, I did have to use external resources to learn about the correct syntax, or require help while troubleshooting errors. Overall, writing SQL queries firsthand has helped in understanding the management and working of Database Systems.

* Learnt about EER diagrams and the relational model. The Entity-Relationship (ER) model provides a high-level view of what data entities exist and how they are related, using concepts like entities, attributes, and relationships. On the other hand, the Relational model offers a more detailed perspective, focusing on how data is stored in tables and ensuring it follows specific rules for efficiency and consistency. Both models play crucial roles in the database design process, with the ER model providing a conceptual framework and the Relational model offering the practical implementation details. Together, they form a comprehensive approach to building well-organized and functional databases
* Learnt how to apply Data Manipulation language on the relational model and use databases with their appropriate syntax to obtain meaningful results from the available data. Using keywords such as SELECT, DESCRIBE, INSERT, UPDATE, WHERE, DELETE, SET, etc allow us to manipulate and handle the data at hand efficiently and effectively.
* Learned about new Technologies and Concepts like:
* PlantUML
* Open source resource to create diagrams based off of textual descriptions.
* SVG Files
* Used for website interfacing attempt.
* ReplIT
* Platform allowing simultaneous access to view and edit programs/codes between team members for better communication.
* Advanced Projects for Complex Queries
* Referred advanced DBMS Projects to formulate more complex queries which helped expand our knowledge and interest in the subject.
* Deep dive into industry standard database management systems and hands-on experience with the creation of mid-to-advanced tier DBMS.
* Used various sources to learn new, out-of-syllabus topics/ concepts/ Technologies such as PlantUML for Schema Diagram generation, Technologies like ReplIT for flexible code verification and team communication, Draw.io for EER Diagram, lastly, advanced projects for implementing more complex queries and functionalities.

# Challenges Faced

* Database Availability
* Modularity
* Learning Curve
* Diagram Generation
* Normalization
* Simplification
* Clean Code
* Introducing Easy Backtracking on code

# Conclusion

In this project, titled "Internet Service Provider Database Management System - A Lens Into The Free World", we have meticulously created and manipulated a series of SQL tables to deepen our understanding of the complex dynamics within the streaming and entertainment sectors. Through queries ranging from basic to advanced, including sub-queries, joins, and set operations, we have extracted, analyzed, and visualized critical data points that shed light on actor demographics, show distributions, platform preferences, and much more. These insights are invaluable for stakeholders aiming to make informed decisions in content creation, distribution, and platform development.

One significant learning from this exercise is the power of SQL in handling and querying large datasets to derive meaningful patterns and trends. We have seen how data normalization and table relationships enhance the database's efficiency and integrity, allowing for more accurate and comprehensive analyses.

However, limitations have also surfaced, particularly regarding the scope of data available and the static nature of SQL queries. Future enhancements could include integrating real-time data feeds, adopting NoSQL databases for unstructured data, or employing machine learning algorithms for predictive analysis. Such advancements would provide a more dynamic, nuanced understanding of the rapidly evolving streaming landscape and its implications for the industry as a whole.