SaaS SUBSCRIPTION PLATFORM

PROJECT PROPOSAL

This project is a data-driven ASP.NET Web Forms application designed to manage customer subscriptions, billing, and payments for Software-as-a-Service (SaaS) products. It supports role-based functionalities for administrators and multiple types of managers, ensuring secure and streamlined access to relevant modules.

KEY FEATURES

Data-Driven Functionality:
All modules, including customer registration, plan subscription, payments, and admin operations, interact with a centralized SQL Server database (SAAS), ensuring persistent and structured data storage.

User Roles & Access Control:
The application defines roles such as:

- Administrator
- Customer Subscription Manager
- Customer Payment Manager
- Subscription Billing Manager
- Customer

Role-specific pages and functionalities are implemented using cookie-based state tracking and controlled redirection.

State Management:
User sessions and login identity are maintained using cookies (e.g., Customer_Id, Admin_Id) to persist authenticated state across pages.

Input Validation:
Both client-side (via ASP.NET validators) and server-side validation ensure input integrity (e.g., required fields, correct formats for payment IDs, phone numbers, etc.).

Data Security:

- All SQL queries are parameterized queries to prevent SQL injection.
- Password fields are masked and validated.
- Unauthorized access to protected pages is restricted based on login status and user role.

Subscription & Payment Workflow:
Customers can subscribe to plans, make payments, and view history. Admins and managers can view and edit records such as customers, subscriptions, payments, and login credentials.

Extensibility:
The system is modular and scalable, allowing for future integration of email notifications, payment gateways, or analytics dashboards.

CONCLUSION

This project successfully meets the criteria of a secure, validated, role-based, and state-managed data-driven web application, aligning well with real-world SaaS business operations.

SOFTWARE REQUIREMENTS SPECIFICATIONS

FUNCTIONAL REQUIREMENTS

User Authentication

- The system allows customers and admins/managers to log in.
- Role is selected (for admin) and verified at login.
- Invalid login attempts are tracked; lockout is implemented after 3 failures for 2 minutes.

Customer Registration

- Customers can register with their details.
- System auto-generates a Customer_Id (e.g., C001, C002…).
- Credentials are stored in Login table.

Subscription Management

- Customers can view and subscribe to plans.
- System auto-generates a Subscription_Id.
- Discounts and total price are calculated based on duration.

Payment Management

- Customers can view unpaid subscriptions and make payments.
- System auto-generates a Payment_Id.
- Payments are stored and linked to the subscription.

Role-Based Admin Functionalities

- Administrator can view and edit Admin and Login tables.
- Customer Subscription Manager (CSM) can view/edit Customer and Subscription data.
- Customer Payment Manager (CPM) can view/edit Payment and Method tables.
- Subscription Billing Manager (SBM) can be extended to manage plan pricing and invoicing.

NON-FUNCTIONAL REQUIREMENTS

Usability

- Intuitive navigation and role-based redirection.
- Clear form labels, validation error messages, and instructions.

Security

- Parameterized queries prevent SQL injection.
- Cookies are used for session persistence.
- Passwords are validated and masked on input.
- Role-based page access control.

Maintainability

- Modular codebase with dedicated event handlers.
- Simple and structured ASP.NET page organization.

Compatibility

- Runs on ASP.NET Framework 4.0+.
- Hosted on Somee with SQL Server backend.

Scalability

- System can scale to more roles and modules.

COMPONENT MODEL

FRONTEND COMPONENTS

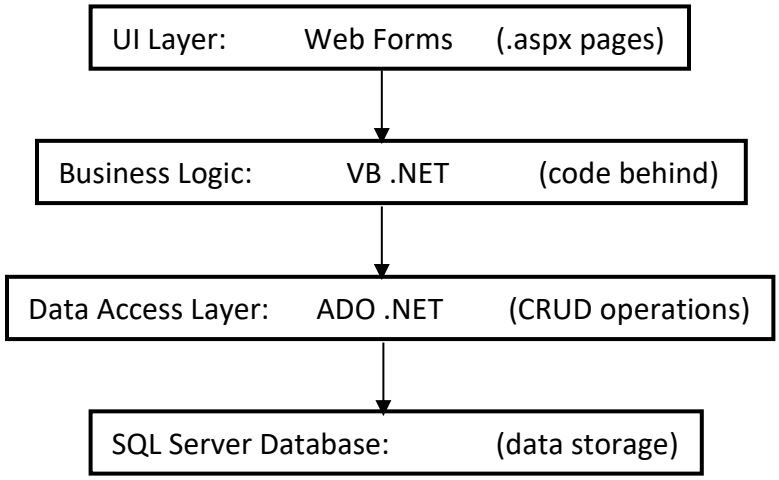
Component	Description
User Interface (ASP.NET Web Forms)	- Home.aspx (Landing page) - Login.aspx (Authentication) - Dashboard.aspx (User dashboard) - Admin.aspx (Admin panel)
CSS & Styling	- MasterPage.master (Consistent layout) - styles.css (Responsive design)

DATABASE COMPONENTS

Component	Description
SQL Server Database	- Login (User credentials) - Customer (User profiles) - Plans (Subscription packages) - Subscription (User subscriptions) - Payment (Transaction records)

BACKEND COMPONENTS

Component	Description
Business Logic (VB.NET)	- Login authentication - Role-based authorization - Subscription management - Payment processing
Data Access Layer	- SQL queries (parameterized)
Session Management	- Cookie-based authentication - Session timeout handling

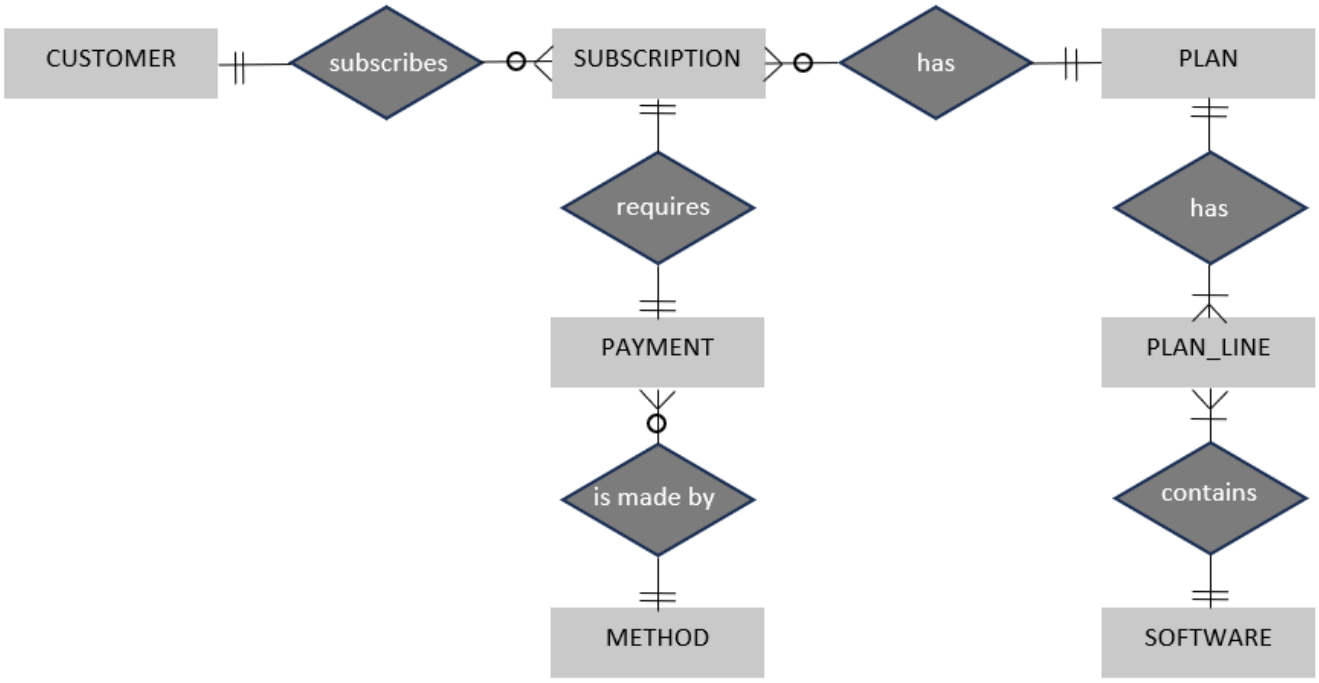


DEPLOYMENT MODEL

Development: Microsoft Visual Studio (ASP .NET with VB .NET)
Server: IIS/somee's online server
Database: Microsoft SQL Server
Client: Web Browser
Hosting: somee.com

DATABASE MODEL

ENTERPRISE DATA MODEL



- A customer subscribes a subscription
- A subscription has a plan that is subscribed
- A plan has plan_lines
- Plan_lines contain software
- A subscription requires payment
- A payment is made by a method

ENTITIES AND ATTRIBUTES

Following are the attributes for each entity and their properties:

CUSTOMER

Attribute	Purpose	Nullable	Key
Customer_Id	Unique identifier for the customer	No	Primary Key
Email	Customer's email address	No	Unique
Name	Customer's full name	No	
Cell_NO	Customer's mobile phone number	No	
Country	Customer's country code (2 characters)	Yes	
City	Customer's city	Yes	

METHOD

Attribute	Purpose	Nullable	Key
Method_Id	Unique identifier for the payment method	No	Primary Key
Method_Name	Name of the payment method	No	Unique

PLAN

Attribute	Purpose	Nullable	Key
Plan_Id	Unique identifier for the subscription plan	No	Primary Key
Plan_Name	Name of the subscription plan	No	Unique
Price_Per_Month	Monthly cost for the plan	No	

SOFTWARE

Attribute	Purpose	Nullable	Key
Software_Id	Unique identifier for the software	No	Primary Key
Software_Name	Name of the software	No	Unique

SUBSCRIPTION

Attribute	Purpose	Nullable	Key
Subscription_Id	Unique identifier for the subscription	No	Primary Key
Customer_Id	Identifier for the customer subscribing	No	Foreign Key (Customer)
Plan_Id	Identifier for the chosen subscription plan	No	Foreign Key (Plan)
Duration_Months	Duration of the subscription in months	No	
Start_Date	Date when the subscription starts	No	
Percent_Discount	Discount percentage applied to the subscription	Yes	
Amount_ToBe_Paid	Total amount to be paid after discount	No	
Paid_Status	Status of the payment ("Yes" or "No")	Yes	

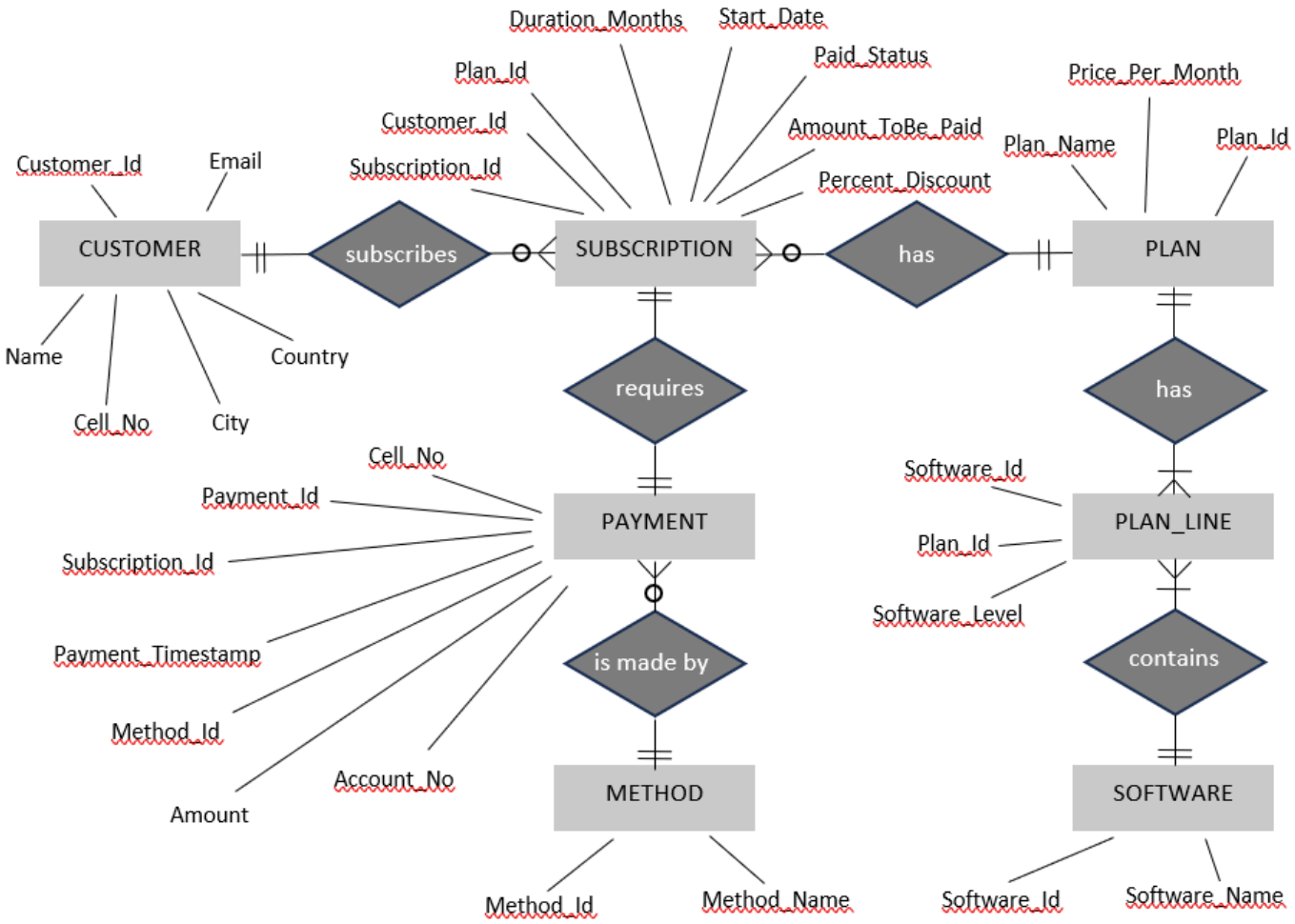
PAYMENT

Attribute	Purpose	Nullable	Key
Payment_Id	Unique identifier for the payment	No	Primary Key
Subscription_Id	Identifier for the associated subscription	No	Foreign Key (Subscription)
Method_Id	Identifier for the payment method	No	Foreign Key (Method)
Cell_NO	Mobile number used for payment	No	
Amount	Total payment amount	No	
Account_No	Account number (optional) for payment	Yes	
Payment_Timestamp	Date and time when the payment was made	Yes	

PLAN_LINE

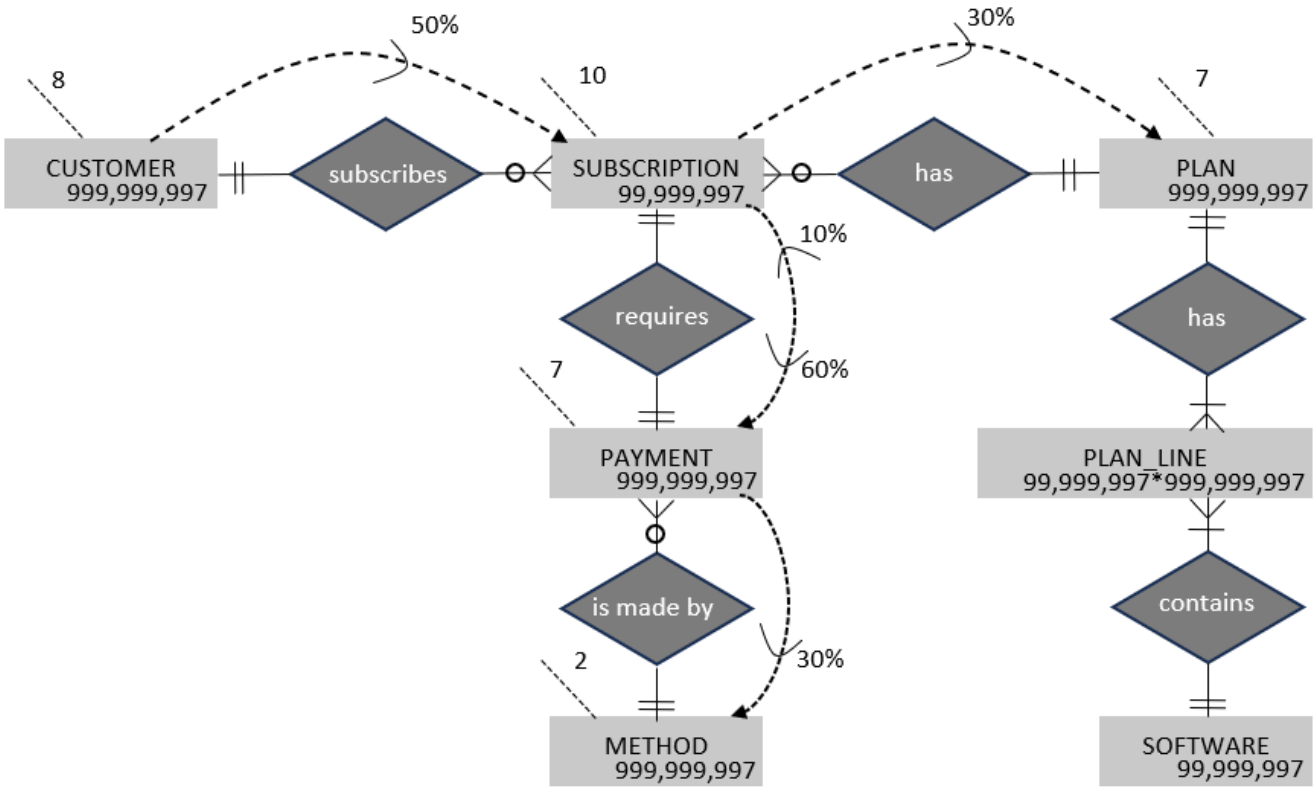
Attribute	Purpose	Nullable	Key
Software_Id	Identifier for the software in the plan	No	Foreign Key (Software)
Plan_Id	Identifier for the plan associated with the software	No	Foreign Key (Plan)
Software_Level	Level of the software in the plan (e.g. Basic, Premium)	No	

ENTITY RELATIONSHIP MODEL

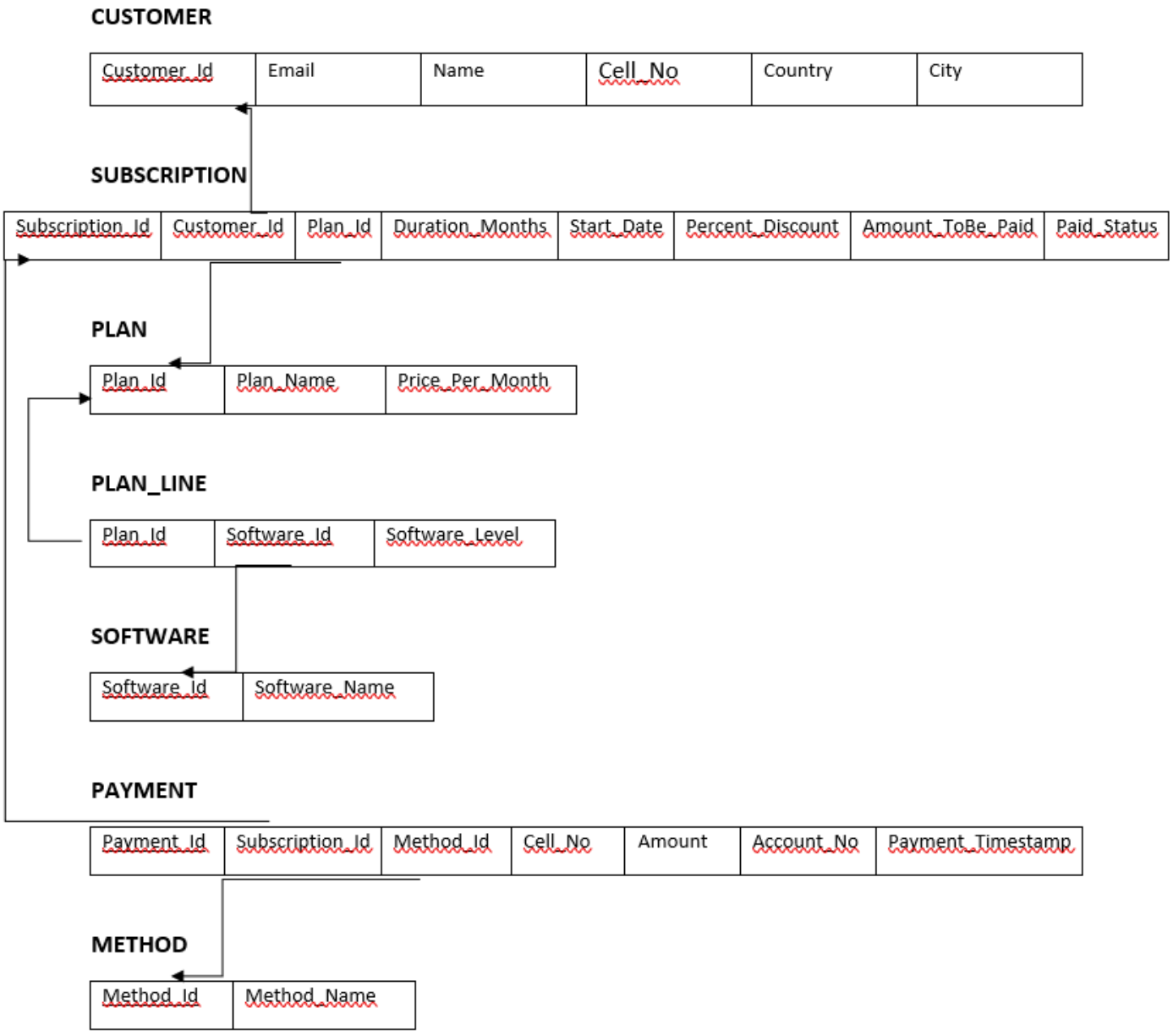


COMPOSITE USAGE MODEL

The Composite Usage Model relative to the queries applied on the database is as follows:
~Access Frequency (per minute)



RELATIONAL SCHEMA



NORMALIZED SCHEMA

- All the tables do not have any non-atomic value, so they are in 1 NF.
- All the tables do not have any partial functional dependency, so they are in 2 NF.
- All the tables do not have any transitive dependency, so they are in 3 NF.

DATABASE CREATION

-- Creating Relations

```
CREATE TABLE Customer (
   Customer_Id   VARCHAR(10)      NOT NULL,
   Email        VARCHAR(30)   NOT NULL,
   Name         VARCHAR(30)    NOT NULL,
   Cell_NO      VARCHAR(11)    NOT NULL,
   Country      VARCHAR(2),
   City         VARCHAR(20),
   CONSTRAINT Customer_PK PRIMARY KEY (Customer_Id),
   CONSTRAINT CellNo_Check CHECK (REGEXP_LIKE(Cell_NO, '^\d{11}$')),
   CONSTRAINT Email_Unique UNIQUE (Email),
   CONSTRAINT Customer_Id_Format CHECK (REGEXP_LIKE(Customer_Id, '^C\d{3,9}$'))
);

CREATE TABLE Method (
   Method_Id     VARCHAR(10)    NOT NULL,
   Method_Name    VARCHAR(20)    NOT NULL,
   CONSTRAINT Method_PK PRIMARY KEY (Method_Id),
   CONSTRAINT Method_Name_Unique UNIQUE (Method_Name),
   CONSTRAINT Method_Id_Format CHECK (REGEXP_LIKE(Method_Id, '^M\d{3,9}$'))
);

CREATE TABLE Plan (
   Plan_Id         VARCHAR(10)    NOT NULL,
   Plan_Name       VARCHAR(30)    NOT NULL,
   Price_per_month   DECIMAL(6, 3)  NOT NULL,
   CONSTRAINT Plan_PK PRIMARY KEY (Plan_Id),
   CONSTRAINT Plan_Name_Unique UNIQUE (Plan_Name),
   CONSTRAINT Plan_Id_Format CHECK (REGEXP_LIKE(Plan_Id, '^P\d{3,9}$'))
);

CREATE TABLE Software (
   Software_Id    VARCHAR(10)    NOT NULL,
   Software_Name  VARCHAR(50)    NOT NULL,
   CONSTRAINT Software_PK PRIMARY KEY (Software_Id),
   CONSTRAINT Software_Name_Unique UNIQUE (Software_Name),
   CONSTRAINT Software_Id_Format CHECK (REGEXP_LIKE(Software_Id, '^SW\d{3,8}$'))
);

CREATE TABLE Subscription (
   Subscription_Id    VARCHAR(10)       NOT NULL,
   Customer_Id       VARCHAR(10)       NOT NULL,
```

```
  Plan_Id          VARCHAR(10)       NOT NULL,
  Duration_Months  NUMBER            NOT NULL,
  Start_Date       DATE              NOT NULL,
  Percent_Discount DECIMAL(5,2),
  Amount_ToBe_Paid DECIMAL(10,2),
  Paid_Status      VARCHAR(3)        DEFAULT 'No',
  CONSTRAINT Subscription_PK PRIMARY KEY (Subscription_Id),
  CONSTRAINT Subscription_FK1 FOREIGN KEY (Plan_Id) REFERENCES Plan(Plan_Id),
  CONSTRAINT Subscription_FK2 FOREIGN KEY (Customer_Id) REFERENCES Customer(Customer_Id),
  CONSTRAINT Subscription_Id_Format CHECK (REGEXP_LIKE(Subscription_Id, '^SB\d{3,8}$'))
);

CREATE TABLE Payment (
  Payment_Id       VARCHAR(10)    NOT NULL,
  Subscription_Id  VARCHAR(10)    NOT NULL,
  Method_Id        VARCHAR(10)    NOT NULL,
  Cell_NO          VARCHAR(11)    NOT NULL,
  Amount           DECIMAL(10, 3)  NOT NULL,
  Account_No       VARCHAR(20)    NOT NULL,
  Payment_Timestamp  TIMESTAMP,
  CONSTRAINT Payment_PK PRIMARY KEY (Payment_Id),
  CONSTRAINT Payment_FK1 FOREIGN KEY (Method_Id) REFERENCES Method(Method_Id),
  CONSTRAINT Payment_FK2 FOREIGN KEY (Subscription_Id) REFERENCES Subscription(Subscription_Id)
);

CREATE TABLE Plan_Line (
  Software_Id  VARCHAR(10)   NOT NULL,
  Plan_Id      VARCHAR(10)   NOT NULL,
  Software_Level    VARCHAR(100)    NOT NULL,
  CONSTRAINT Plan_Line_PK PRIMARY KEY (Plan_Id, Software_Id),
  CONSTRAINT Plan_Line_FK1 FOREIGN KEY (Plan_Id) REFERENCES Plan(Plan_Id),
  CONSTRAINT Plan_Line_FK2 FOREIGN KEY (Software_Id) REFERENCES Software(Software_Id)
);
```

-- Creating Triggers

```
CREATE OR REPLACE TRIGGER Prevent_Duplicate_Payments
BEFORE INSERT ON Payment
FOR EACH ROW
DECLARE
  v_exists NUMBER;
BEGIN
  SELECT COUNT(*)
  INTO v_exists
  FROM Payment
  WHERE Subscription_Id = :NEW.Subscription_Id
   AND Method_Id = :NEW.Method_Id
   AND Amount = :NEW.Amount
   AND Payment_Timestamp = :NEW.Payment_Timestamp;

  IF v_exists > 0 THEN
    DBMS_OUTPUT.PUT_LINE('Error: Duplicate payment detected.');
    RAISE_APPLICATION_ERROR(-20001, 'Duplicate payment detected.');
  END IF;
END;
/

CREATE OR REPLACE TRIGGER Validate_Discount_Range
BEFORE INSERT OR UPDATE ON Subscription
FOR EACH ROW
BEGIN
  IF :NEW.Percent_Discount < 0 OR :NEW.Percent_Discount > 100 THEN
    DBMS_OUTPUT.PUT_LINE('Error: Discount percent must be between 0 and 100.');
    RAISE_APPLICATION_ERROR(-20002, 'Discount percent out of range.');
  END IF;
END;
/

CREATE OR REPLACE TRIGGER Calculate_Amount_ToBe_Paid
BEFORE INSERT OR UPDATE ON Subscription
FOR EACH ROW
DECLARE
  v_Price_per_month DECIMAL(10,2);
  v_Total DECIMAL(10,2);
  v_Discount_amount DECIMAL(10,2);
BEGIN
  SELECT Price_per_month INTO v_Price_per_month
  FROM Plan
  WHERE Plan_Id = :NEW.Plan_Id;
  v_Total := v_Price_per_month * :NEW.Duration_Months;
  IF :NEW.Percent_Discount IS NOT NULL THEN
    v_Discount_amount := (v_Total * :NEW.Percent_Discount) / 100;
  ELSE
    v_Discount_amount := 0;
  END IF;
  :NEW.Amount_ToBe_Paid := v_Total - v_Discount_amount;
END;
```

```
/

CREATE OR REPLACE TRIGGER Validate_Start_Date
BEFORE INSERT OR UPDATE ON Subscription
FOR EACH ROW
BEGIN
  IF :NEW.Start_Date > SYSDATE THEN
    DBMS_OUTPUT.PUT_LINE('Error: Start Date cannot be in the future.');
    RAISE_APPLICATION_ERROR(-20001, 'Start Date cannot be in the future.');
  END IF;
END;
/

CREATE OR REPLACE TRIGGER Validate_Unique_Plan
BEFORE INSERT ON Plan_Line
FOR EACH ROW
DECLARE
  v_conflict NUMBER;
BEGIN
  SELECT COUNT(*)
  INTO v_conflict
  FROM Plan_Line
  WHERE Plan_Id = :NEW.Plan_Id
   AND Software_Id = :NEW.Software_Id
   AND Software_Level = :NEW.Software_Level;
  IF v_conflict > 0 THEN
    DBMS_OUTPUT.PUT_LINE('Error: Plan must have unique software or different levels.');
    RAISE_APPLICATION_ERROR(-20006, 'Non-unique plan detected.');
  END IF;
END;
/

CREATE OR REPLACE TRIGGER Validate_Payment_Amount
BEFORE INSERT OR UPDATE ON Payment
FOR EACH ROW
DECLARE
  v_amount_to_be_paid DECIMAL(10, 2);
BEGIN
  SELECT Amount_ToBe_Paid
  INTO v_amount_to_be_paid
  FROM Subscription
  WHERE Subscription_Id = :NEW.Subscription_Id;
    IF :NEW.Amount <= 0 THEN
    DBMS_OUTPUT.PUT_LINE('Error: Payment amount must be greater than zero.');
    RAISE_APPLICATION_ERROR(-20004, 'Invalid payment amount.');
  ELSIF :NEW.Amount != v_amount_to_be_paid THEN
    DBMS_OUTPUT.PUT_LINE('Error: Incorrect amount being paid. Expected amount is ' || v_amount_to_be_paid);
    RAISE_APPLICATION_ERROR(-20005, 'Incorrect payment amount.');
  END IF;
END;
/
```

```
CREATE OR REPLACE TRIGGER Update_Subscription_Status
AFTER INSERT ON Payment
FOR EACH ROW
BEGIN
   UPDATE Subscription
   SET Paid_Status = 'Yes'
   WHERE Subscription_Id = :NEW.Subscription_Id
    AND Paid_Status != 'Yes'; -- Only update if Paid_Status is not already 'Yes'
END;
/

CREATE OR REPLACE TRIGGER Validate_Customer_Subscription
BEFORE INSERT ON Subscription
FOR EACH ROW
DECLARE
   v_Unpaid_Sub_Count NUMBER;
BEGIN
   SELECT COUNT(*)
   INTO v_Unpaid_Sub_Count
   FROM Subscription
   WHERE Customer_Id = :NEW.Customer_Id
    AND Paid_Status != 'Yes'; -- Payment not completed
   IF v_Unpaid_Sub_Count > 0 THEN
      DBMS_OUTPUT.PUT_LINE('Error: Cannot add a new subscription until the previous subscription is fully paid.');
      RAISE_APPLICATION_ERROR(-20007, 'Previous subscription has not been paid.');
   END IF;
END;
/
```

-- Creating Views

```
CREATE VIEW Customer_Payment_Manager_View AS
SELECT
   C.Customer_Id, C.Name, C.Email, C.City,
   P.Payment_Id, P.Amount, P.Payment_Timestamp, P.Method_Id
FROM
   Customer C
JOIN
   Subscription S ON C.Customer_Id = S.Customer_Id
JOIN
   Payment P ON S.Subscription_Id = P.Subscription_Id;

CREATE VIEW Subs_Bill_Manager_View AS
SELECT
   S.Subscription_Id, S.Customer_Id, S.Plan_Id, S.Amount_ToBe_Paid,
   P.Plan_Name, P.Price_per_month, S.Paid_Status
FROM
   Subscription S
JOIN
   Plan P ON S.Plan_Id = P.Plan_Id;
```

```
CREATE VIEW Cust_Subs_Manager_View AS
SELECT
   C.Customer_Id, C.Name, C.Email, C.City,
   S.Subscription_Id, S.Plan_Id, S.Duration_Months, S.Start_Date
FROM
   Customer C
JOIN
   Subscription S ON C.Customer_Id = S.Customer_Id;
```

-- Creating Roles and Assigning Privileges

```
CREATE ROLE Customer_Payment_Manager;
CREATE ROLE Subscription_Billing_Manager;
CREATE ROLE Customer_Subscription_Manager;

GRANT SELECT, INSERT, UPDATE ON Customer_Payment_Manager_View TO Customer_Payment_Manager;
GRANT SELECT, INSERT, UPDATE ON Subs_Bill_Manager_View TO Subscription_Billing_Manager;
GRANT SELECT, INSERT, UPDATE ON Cust_Subs_Manager_View TO Customer_Subscription_Manager;
```

-- Creating Users and Assigning Roles

```
CREATE USER user1 IDENTIFIED BY pwd1;
CREATE USER user2 IDENTIFIED BY pwd2;
CREATE USER user3 IDENTIFIED BY pwd3;
CREATE USER user4 IDENTIFIED BY pwd4;
CREATE USER user5 IDENTIFIED BY pwd5;
CREATE USER user6 IDENTIFIED BY pwd6;
CREATE USER user7 IDENTIFIED BY pwd7;

GRANT Customer_Payment_Manager TO user1;
GRANT Customer_Payment_Manager TO user2;
GRANT Customer_Payment_Manager TO user7;
GRANT Subscription_Billing_Manager TO user3;
GRANT Subscription_Billing_Manager TO user4;
GRANT Subscription_Billing_Manager TO user7;
GRANT Customer_Subscription_Manager TO user5;
GRANT Customer_Subscription_Manager TO user6;
GRANT Customer_Subscription_Manager TO user7;
```

DATA INSERTION

-- CUSTOMER (40 rows)

```
INSERT INTO Customer (Customer_Id, Email, Name, Cell_NO, Country, City) VALUES
('C001', 'alex.johnson@example.com', 'Alex Johnson', '10000000001', 'US', 'New York');
INSERT INTO Customer (Customer_Id, Email, Name, Cell_NO, Country, City) VALUES
('C002', 'maria.garcia@example.com', 'Maria Garcia', '10000000002', 'US', 'Los Angeles');
INSERT INTO Customer (Customer_Id, Email, Name, Cell_NO, Country, City) VALUES
('C003', 'li.chen@example.com', 'Li Chen', '10000000003', 'CN', 'Beijing');
```

EDIT	CUSTOMER_ID	EMAIL	NAME	CELL_NO	COUNTRY	CITY
	C001	alex.johnson@example.com	Alex Johnson	10000000001	US	New York
	C002	maria.garcia@example.com	Maria Garcia	10000000002	US	Los Angeles
	C003	li.chen@example.com	Li Chen	10000000003	CN	Beijing
	C004	william.brown@example.com	William Brown	10000000004	UK	London
	C005	sophia.miller@example.com	Sophia Miller	10000000005	CA	Toronto
	C006	noah.davis@example.com	Noah Davis	10000000006	US	Chicago
	C007	emma.wilson@example.com	Emma Wilson	10000000007	AU	Sydney
	C008	oliver.thomas@example.com	Oliver Thomas	10000000008	US	Houston
	C009	ava.taylor@example.com	Ava Taylor	10000000009	IN	Mumbai
	C010	lucas.rodriguez@example.com	Lucas Rodriguez	10000000010	MX	Mexico City
	C011	mia.martinez@example.com	Mia Martinez	10000000011	ES	Madrid
	C012	elijah.white@example.com	Elijah White	10000000012	US	San Francisco
	C013	amelia.anderson@example.com	Amelia Anderson	10000000013	NZ	Auckland
	C014	ethan.moore@example.com	Ethan Moore	10000000014	US	Dallas
	C015	harper.jones@example.com	Harper Jones	10000000015	US	Seattle

row(s) 1 - 15 of 40

-- METHOD (8 rows)

```
INSERT INTO Method (Method_Id, Method_Name) VALUES ('M101', 'Credit Card');
INSERT INTO Method (Method_Id, Method_Name) VALUES ('M102', 'PayPal');
INSERT INTO Method (Method_Id, Method_Name) VALUES ('M103', 'Bank Transfer');
```

EDIT	METHOD_ID	METHOD_NAME
	M101	Credit Card
	M102	PayPal
	M103	Bank Transfer
	M104	Bitcoin
	M105	Apple Pay
	M106	Google Pay
	M107	Venmo
	M108	Stripe

row(s) 1 - 8 of 8

Download

-- SOFTWARE (30 rows)

```
INSERT INTO Software (Software_Id, Software_Name) VALUES ('SW101', 'Microsoft Word');
INSERT INTO Software (Software_Id, Software_Name) VALUES ('SW102', 'Microsoft Excel');
INSERT INTO Software (Software_Id, Software_Name) VALUES ('SW103', 'Microsoft PowerPoint');
```

EDIT	SOFTWARE_ID	SOFTWARE_NAME
	SW101	Microsoft Word
	SW102	Microsoft Excel
	SW103	Microsoft PowerPoint
	SW104	Adobe Photoshop
	SW105	Adobe Premiere Pro
	SW106	Google Docs
	SW107	Google Sheets
	SW108	Google Slides
	SW109	Apple Pages
	SW110	Apple Numbers
	SW111	Apple Keynote
	SW112	Notion
	SW113	Slack
	SW114	Trello
	SW115	Asana

row(s) 1 - 15 of 30

Download

-- PLAN (10 rows)

INSERT INTO Plan (Plan_Id, Plan_Name, Price_per_month) VALUES ('P001', 'Basic Plan', 9.990);
INSERT INTO Plan (Plan_Id, Plan_Name, Price_per_month) VALUES ('P002', 'Standard Plan', 19.990);
INSERT INTO Plan (Plan_Id, Plan_Name, Price_per_month) VALUES ('P003', 'Premium Plan', 29.990);

EDIT	PLAN_ID	PLAN_NAME	PRICE_PER_MONTH
	P001	Basic Plan	9.99
	P006	Pro Plan	24.99
	P002	Standard Plan	19.99
	P003	Premium Plan	29.99
	P004	Enterprise Plan	49.99
	P005	Starter Plan	7.49
	P007	Developer Plan	14.99
	P008	Business Plan	39.99
	P009	Ultimate Plan	59.99
	P010	Student Plan	4.99

row(s) 1 - 10 of 10

Download

-- PLAN_LINE (50 rows)

INSERT INTO Plan_Line (Plan_Id, Software_Id, Software_Level) VALUES ('P001', 'SW101', 'Standard');
INSERT INTO Plan_Line (Plan_Id, Software_Id, Software_Level) VALUES ('P002', 'SW102', 'Advanced');
INSERT INTO Plan_Line (Plan_Id, Software_Id, Software_Level) VALUES ('P003', 'SW103', 'Enterprise');
INSERT INTO Plan_Line (Plan_Id, Software_Id, Software_Level) VALUES ('P004', 'SW104', 'Professional');

EDIT	SOFTWARE_ID	PLAN_ID	SOFTWARE_LEVEL
	SW101	P001	Standard
	SW102	P002	Advanced
	SW103	P003	Enterprise
	SW104	P004	Professional
	SW105	P005	Starter
	SW106	P006	Advanced
	SW107	P007	Developer
	SW108	P008	Business
	SW109	P009	Ultimate
	SW110	P010	Student
	SW102	P001	Basic
	SW103	P002	Professional
	SW104	P003	Enterprise
	SW105	P004	Standard
	SW106	P005	Starter

row(s) 1 - 15 of 50

Download

-- SUBSCRIPTION (32 rows)

INSERT INTO Subscription (Subscription_Id, Customer_Id, Plan_Id, Duration_Months, Start_Date, Percent_Discount, Paid_Status) VALUES ('SB001', 'C001', 'P001', 6, TO_DATE('2023-01-01', 'YYYY-MM-DD'), 10, 'Yes');
INSERT INTO Subscription (Subscription_Id, Customer_Id, Plan_Id, Duration_Months, Start_Date, Percent_Discount, Paid_Status) VALUES ('SB002', 'C002', 'P002', 8, TO_DATE('2023-02-01', 'YYYY-MM-DD'), 15, 'No');
INSERT INTO Subscription (Subscription_Id, Customer_Id, Plan_Id, Duration_Months, Start_Date, Percent_Discount, Paid_Status) VALUES ('SB003', 'C003', 'P003', 13, TO_DATE('2024-03-01', 'YYYY-MM-DD'), 30, 'Yes');
INSERT INTO Subscription (Subscription_Id, Customer_Id, Plan_Id, Duration_Months, Start_Date, Percent_Discount, Paid_Status) VALUES ('SB004', 'C004', 'P004', 25, TO_DATE('2024-04-01', 'YYYY-MM-DD'), 45, 'No');

EDIT	SUBSCRIPTION_ID	CUSTOMER_ID	PLAN_ID	DURATION_MONTHS	START_DATE	PERCENT_DISCOUNT	AMOUNT_TOBE_PAID	PAID_STATUS
	SB001	C001	P001	6	01-JAN-24	10	53.95	Yes
	SB002	C002	P002	8	01-FEB-24	15	135.93	No
	SB003	C003	P003	13	01-MAR-24	30	272.91	Yes
	SB004	C004	P004	25	01-APR-24	45	687.36	No
	SB005	C001	P001	5	01-MAY-24	0	49.95	No
	SB007	C003	P003	12	01-JUL-24	15	305.9	Yes
	SB011	C003	P003	14	01-NOV-24	30	293.9	Yes
	SB008	C008	P004	24	01-AUG-24	30	839.83	No
	SB009	C009	P001	7	01-SEP-24	15	59.44	Yes
	SB010	C010	P002	9	01-OCT-24	15	152.92	No
	SB012	C012	P004	26	01-DEC-24	45	714.86	No
	SB013	C013	P001	3	01-JAN-23	0	29.97	No
	SB015	C011	P003	6	01-MAR-24	10	161.95	Yes
	SB016	C016	P004	7	01-APR-22	15	297.44	No
	SB017	C017	P001	10	01-MAY-22	15	84.91	Yes

row(s) 1 - 15 of 32

Download

-- PAYMENT (13 rows)

INSERT INTO Payment (Payment_Id, Subscription_Id, Method_Id, Cell_NO, Amount, Account_No, Payment_Timestamp)
VALUES ('P001', 'SB001', 'M101', '10000000001', 53.950, 'ACC001', TO_TIMESTAMP('2024-01-01 10:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Payment (Payment_Id, Subscription_Id, Method_Id, Cell_NO, Amount, Account_No, Payment_Timestamp)
VALUES ('P002', 'SB003', 'M101', '10000000003', 272.910, 'ACC002', TO_TIMESTAMP('2024-03-01 12:00:00', 'YYYY-MM-DD HH24:MI:SS'));
INSERT INTO Payment (Payment_Id, Subscription_Id, Method_Id, Cell_NO, Amount, Account_No, Payment_Timestamp)
VALUES ('P003', 'SB007', 'M103', '10000000007', 305.900, 'ACC003', TO_TIMESTAMP('2024-07-01 14:00:00', 'YYYY-MM-DD HH24:MI:SS'));

EDIT	PAYMENT_ID	SUBSCRIPTION_ID	METHOD_ID	CELL_NO	AMOUNT	ACCOUNT_NO	PAYMENT_TIMESTAMP
	P001	SB001	M101	10000000001	53.95	ACC001	01-JAN-24 10.00.00.000000 AM
	P002	SB003	M101	10000000003	272.91	ACC002	01-MAR-24 12.00.00.000000 PM
	P003	SB007	M103	10000000007	305.9	ACC003	01-JUL-24 02.00.00.000000 PM
	P004	SB009	M107	10000000008	59.44	ACC004	01-SEP-24 04.00.00.000000 PM
	P005	SB011	M107	10000000012	293.9	ACC005	01-NOV-24 06.00.00.000000 PM
	P006	SB017	M107	10000000017	84.91	ACC006	01-MAY-22 08.00.00.000000 PM
	P007	SB023	M102	10000000017	287.9	ACC007	01-NOV-23 10.00.00.000000 PM
	P008	SB027	M103	10000000027	280.41	ACC008	01-MAR-23 10.30.00.000000 AM
	P009	SB029	M108	10000000029	59.44	ACC009	01-MAY-24 12.30.00.000000 PM
	P010	SB031	M101	10000000031	272.91	ACC010	01-JUL-23 02.30.00.000000 PM
	P011	SB033	M101	10000000033	53.95	ACC011	01-SEP-22 04.30.00.000000 PM
	P012	SB037	M107	10000000037	49.95	ACC012	01-JAN-23 06.30.00.000000 PM
	P013	SB039	M103	10000000039	269.91	ACC013	01-MAR-24 08.30.00.000000 PM

row(s) 1 - 13 of 13

Download

APPLYING QUERIES

1) Get the total amount paid by customers of every country

```
SELECT
   C.Country,
   SUM(P.Amount) AS Total_Amount_Paid
FROM
   Customer C
JOIN
   Subscription S ON C.Customer_Id = S.Customer_Id
JOIN
   Payment P ON S.Subscription_Id = P.Subscription_Id
GROUP BY
   C.Country
ORDER BY
   C.Country;
```

COUNTRY	TOTAL_AMOUNT_PAID
AU	280.41
CA	372.81
CN	872.71
ID	269.91
IN	59.44
MY	49.95
NL	53.95
RU	272.91
SG	59.44
US	53.95

10 rows returned in 0.00 seconds CSV

2) List the plans and their total revenue, including the number of subscribers

```
SELECT
   P.Plan_Id,
   P.Plan_Name,
   COUNT(S.Subscription_Id) AS Number_of_Subscribers,
   SUM(Payment.Amount) AS Total_Revenue
FROM
   Plan P
JOIN
   Subscription S ON P.Plan_Id = S.Plan_Id
JOIN
   Payment ON S.Subscription_Id = Payment.Subscription_Id
GROUP BY
   P.Plan_Id, P.Plan_Name
ORDER BY
   Total_Revenue DESC;
```

PLAN_ID	PLAN_NAME	NUMBER_OF_SUBSCRIBERS	TOTAL_REVENUE
P003	Premium Plan	7	1983.84
P001	Basic Plan	6	361.64

3) List top 5 customers who have received the maximum discount on a plan

```
SELECT * FROM (
   SELECT
      C.Customer_Id,
      C.Name,
      S.Plan_Id,
      S.Percent_Discount
   FROM
      Customer C
   JOIN
      Subscription S ON C.Customer_Id = S.Customer_Id
   ORDER BY
      S.Percent_Discount DESC
)
WHERE ROWNUM <= 5;
```

CUSTOMER_ID	NAME	PLAN_ID	PERCENT_DISCOUNT
C004	William Brown	P004	45
C012	Elijah White	P004	45
C032	Lucas Perez	P004	35
C003	Li Chen	P003	30
C003	Li Chen	P003	30

4) Get customers who have never made a payment but have active subscriptions

```
SELECT
   C.Customer_Id,
   C.Name,
   S.Subscription_Id,
   S.Paid_Status
FROM
   Customer C
JOIN
   Subscription S ON C.Customer_Id = S.Customer_Id
WHERE
   S.Subscription_Id NOT IN (SELECT Subscription_Id FROM Payment)
   AND S.Paid_Status = 'No';
```

CUSTOMER_ID	NAME	SUBSCRIPTION_ID	PAID_STATUS
C001	Alex Johnson	SB005	No
C002	Maria Garcia	SB002	No
C003	Li Chen	SB036	No
C004	William Brown	SB004	No
C008	Oliver Thomas	SB008	No
C010	Lucas Rodriguez	SB010	No
C011	Mia Martinez	SB028	No
C012	Elijah White	SB012	No
C013	Amelia Anderson	SB013	No
C016	Benjamin Harris	SB016	No
C018	Jackson Lee	SB018	No
C024	Daniel Scott	SB024	No
C026	Sebastian Adams	SB026	No
C032	Lucas Perez	SB032	No
C033	Sophia Ward	SB034	No
C035	Ella Collins	SB038	No
C040	Lily Wright	SB040	No

5) List all subscriptions with the calculated total amount after discount, and the difference between the actual and expected amounts

```
SELECT
   S.Subscription_Id,
   S.Customer_Id,
   S.Plan_Id,
   (P.Price_per_month * S.Duration_Months) AS Actual_Amount,
   S.Amount_ToBe_Paid,
   (P.Price_per_month * S.Duration_Months) - S.Amount_ToBe_Paid AS Discount_Amount
FROM
   Subscription S
JOIN
   Plan P ON S.Plan_Id = P.Plan_Id;
```

SUBSCRIPTION_ID	CUSTOMER_ID	PLAN_ID	ACTUAL_AMOUNT	AMOUNT_TOBE_PAID	DISCOUNT_AMOUNT
SB001	C001	P001	59.94	53.95	5.99
SB002	C002	P002	159.92	135.93	23.99
SB003	C003	P003	389.87	272.91	116.96
SB004	C004	P004	1249.75	687.36	562.39
SB005	C001	P001	49.95	49.95	0
SB007	C003	P003	359.88	305.9	53.98
SB011	C003	P003	419.86	293.9	125.96
SB008	C008	P004	1199.76	839.83	359.93
SB009	C009	P001	69.93	59.44	10.49
SB010	C010	P002	179.91	152.92	26.99
SB012	C012	P004	1299.74	714.86	584.88
SB013	C013	P001	29.97	29.97	0
SB015	C011	P003	179.94	161.95	17.99
SB016	C016	P004	349.93	297.44	52.49
SB017	C017	P001	99.9	84.91	14.99
SB018	C018	P002	259.87	181.91	77.96
SB023	C017	P003	359.88	287.9	71.98
SB024	C024	P004	899.82	674.86	224.96
SB026	C026	P002	179.91	161.92	17.99
SB027	C027	P003	329.89	280.41	49.48

More than 20 rows available. Increase rows selector to view more rows.

6) Find customers who have made payments to all plans they are subscribed to

```
SELECT
   C.Customer_Id,
   C.Name
FROM
   Customer C
JOIN
   Subscription S ON C.Customer_Id = S.Customer_Id
GROUP BY
   C.Customer_Id, C.Name
HAVING
   COUNT(DISTINCT S.Plan_Id) =
   (SELECT COUNT(DISTINCT S2.Plan_Id)
    FROM Subscription S2
    WHERE S2.Customer_Id = C.Customer_Id);
```

CUSTOMER_ID	NAME
C001	Alex Johnson
C002	Maria Garcia
C003	Li Chen
C004	William Brown
C008	Oliver Thomas
C009	Ava Taylor
C010	Lucas Rodriguez
C011	Mia Martinez
C012	Elijah White
C013	Amelia Anderson
C016	Benjamin Harris
C017	Evelyn King
C018	Jackson Lee
C024	Daniel Scott
C026	Sebastian Adams
C027	Zoe Baker
C029	Nora Carter
C031	Bella Hughes
C032	Lucas Perez
C033	Sophia Ward
More than 20 rows available. Increase rows selector to view more rows.	

7) Get the total number of payments for each payment method, along with the total sum of payments

```
SELECT
   M.Method_Name,
   COUNT(P.Payment_Id) AS Total_Payments,
   SUM(P.Amount) AS Total_Amount
FROM
   Payment P
JOIN
   Method M ON P.Method_Id = M.Method_Id
GROUP BY
   M.Method_Name
ORDER BY
   Total_Amount DESC;
```

METHOD_NAME	TOTAL_PAYMENTS	TOTAL_AMOUNT
Bank Transfer	3	856.22
Credit Card	4	653.72
Venmo	4	488.2
PayPal	1	287.9
Stripe	1	59.44

8) Get the plan with the highest number of active subscriptions in 2023-01

```
SELECT
   S.Plan_Id,
   P.Plan_Name,
   COUNT(*) AS Active_Subscription_Count
FROM
   Subscription S
JOIN
   Plan P ON S.Plan_Id = P.Plan_Id
WHERE
```

```
  TO_CHAR(S.Start_Date, 'YYYY-MM') = '2023-01'
GROUP BY
  S.Plan_Id, P.Plan_Name
HAVING
  COUNT(*) = (
    SELECT MAX(Active_Subscription_Count)
    FROM (
      SELECT COUNT(*) AS Active_Subscription_Count
      FROM Subscription S2
      WHERE TO_CHAR(S2.Start_Date, 'YYYY-MM') = '2023-01'
      GROUP BY S2.Plan_Id
    )
  )
ORDER BY
  Active_Subscription_Count DESC;
```

PLAN_ID	PLAN_NAME	ACTIVE_SUBSCRIPTION_COUNT
P001	Basic Plan	2

9) Find the average subscription duration for each plan

```
SELECT
  P.Plan_Id,
  P.Plan_Name,
  AVG(S.Duration_Months) AS Average_Duration
FROM
  Subscription S
JOIN
  Plan P ON S.Plan_Id = P.Plan_Id
GROUP BY
  P.Plan_Id, P.Plan_Name;
```

PLAN_ID	PLAN_NAME	AVERAGE_DURATION
P003	Premium Plan	11.444444444444444444444444444444444444
P002	Standard Plan	9.333333333333333333333333333333333333
P004	Enterprise Plan	17.777777777777777777777777777777777778
P001	Basic Plan	6.125

10) Identify customers who have not used credit card payment method but have subscriptions

```
SELECT
  C.Customer_Id,
  C.Name
FROM
  Customer C
WHERE
  C.Customer_Id NOT IN (
    SELECT DISTINCT S.Customer_Id
    FROM Subscription S
    JOIN Payment P ON S.Subscription_Id = P.Subscription_Id
    JOIN Method M ON P.Method_Id = M.Method_Id
    WHERE M.Method_Name = 'Credit Card'
  );
```

CUSTOMER_ID	NAME
C039	Julian Wood
C017	Evelyn King
C037	Ryan Morgan
C013	Amelia Anderson
C034	Michael Bell
C006	Noah Davis
C005	Sophia Miller
C021	Grace Hall
C024	Daniel Scott
C026	Sebastian Adams
C015	Harper Jones
C012	Elijah White
C027	Zoe Baker
C023	Chloe Young
C035	Ella Collins
C028	Matthew Hill
C025	Violet Green
C030	Leo Mitchell
C022	Henry Allen
C019	Scarlett Walker
More than 20 rows available. Increase rows selector to view more rows.	

11) Find the top 2 most expensive plans based on their total amount of payments made

```
SELECT *
FROM (
   SELECT
      P.Plan_Id,
      SUM(PM.Amount) AS Total_Payments
   FROM
      Payment PM
   JOIN
      Subscription S ON PM.Subscription_Id = S.Subscription_Id
   JOIN
      Plan P ON S.Plan_Id = P.Plan_Id
   GROUP BY
      P.Plan_Id
   ORDER BY
      Total_Payments DESC
)
WHERE ROWNUM <= 2;
```

PLAN_ID	TOTAL_PAYMENTS
P003	1983.84
P001	361.64

12) Calculate the total discount amount provided across all plans in the last 6 months

```
SELECT
   SUM((P.Price_per_month * S.Duration_Months) - S.Amount_ToBe_Paid) AS Total_Discount
FROM
   Payment PM
JOIN
```

```
   Subscription S ON PM.Subscription_Id = S.Subscription_Id
JOIN
   Plan P ON S.Plan_Id = P.Plan_Id
WHERE
   PM.Payment_Timestamp >= ADD_MONTHS(SYSDATE, -6);
```

TOTAL_DISCOUNT
190.43

13) Identify customers who have never received a discount on their subscription

```
SELECT
   C.Customer_Id
FROM
   Customer C
WHERE
   C.Customer_Id NOT IN (
      SELECT DISTINCT S.Customer_Id
      FROM Subscription S
      JOIN Payment PM ON S.Subscription_Id = PM.Subscription_Id
      WHERE S.Percent_Discount > 0
   );
```

CUSTOMER_ID
C037
C013
C034
C006
C005
C021
C024
C026
C015
C012
C023
C035
C028
C025
C030
C022
C019
C020
C038
C008
More than 20 rows available. Increase rows selector to view more rows.

14) Find the most frequent payment method used by customers in US

```
SELECT
   payment_method,
   payment_count
FROM (
   SELECT
      p.Method_Id AS payment_method,
      COUNT(*) AS payment_count
   FROM
      Payment p
```

```
    JOIN
        Subscription s ON p.Subscription_Id = s.Subscription_Id
    JOIN
        Customer c ON s.Customer_Id = c.Customer_Id
    WHERE
        c.Country = 'US'
    GROUP BY
        p.Method_Id
)
WHERE
    payment_count = (
        SELECT
            MAX(COUNT(*))
        FROM
            Payment p
        JOIN
            Subscription s ON p.Subscription_Id = s.Subscription_Id
        JOIN
            Customer c ON s.Customer_Id = c.Customer_Id
        WHERE
            c.Country = 'US'
        GROUP BY
            p.Method_Id
    );
```

PAYMENT_METHOD	PAYMENT_COUNT
M101	1

WEB APPLICATION IMPLEMENTATION

The created web application is available at:
https://iqramehmood.somee.com/LAB%2011/Home.aspx