Name : U Ganesh Chowdari

Email : ganeshchowdary077@gmail.com

Date: 31-12-2025

Day 4 – Module 4.4 Practical Project Assignment

Create Sample Insurance database named **InsuranceDB** with tables, constraints, and initial sample data. Based on following er diagram and descriptions.

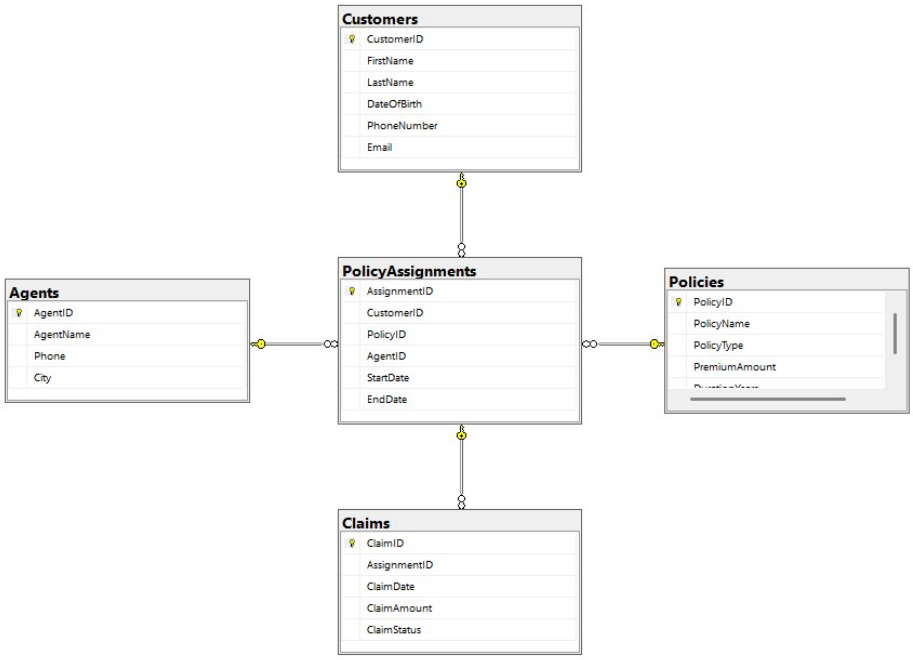
Entities we'll model:

- **Customers** – people who buy insurance
- **Policies** – insurance products (Health, Motor, Life)
- **Agents** – insurance agents
- **PolicyAssignments** – which customer bought which policy
- **Claims** – claims raised against policies

Relationship Explanation

- **One Customer → Many PolicyAssignments**
- **One Policy → Many PolicyAssignments**
- **One Agent → Many PolicyAssignments**
- **One PolicyAssignment → Many Claims**

Database Diagram:



Create Database command.

Query:
```
CREATE DATABASE InsuranceDB;
```

Create table commands for all the tables with constraints, relationships etc.

Customer Table Query:
```
-- Customers Table
CREATE TABLE Customers
(
	CustomerID INT IDENTITY NOT NULL,
	FirstName VARCHAR(30) NOT NULL,
	LastName VARCHAR(30),
	DateOfBirth DATE,
	PhoneNumber VARCHAR(15) NOT NULL,
	Email VARCHAR(50) NOT NULL UNIQUE,
	CONSTRAINT custID_pk PRIMARY KEY (CustomerID)

);
```

Policies Table Query:
```
-- Policies Table
CREATE TABLE Policies
(
	PolicyID INT IDENTITY NOT NULL,
	PolicyName VARCHAR(50) NOT NULL,
	PolicyType VARCHAR(50) NOT NULL,
	PremiumAmount DECIMAL(10,2),
	DurationYears INT
	CONSTRAINT PK_policyID PRIMARY KEY (PolicyID)

);
```

Agents Table Query:
```
-- Agents Table
CREATE TABLE Agents
(
	AgentID INT IDENTITY NOT NULL,
	AgentName VARCHAR(50),
	Phone VARCHAR(15),
	City VARCHAR(50)
	CONSTRAINT PK_agentID PRIMARY KEY (AgentID)

);
```

PolicyAssignments Table Query:

```sql
-- PolicyAssignments Table
CREATE TABLE PolicyAssignments
(
	AssignmentID INT NOT NULL IDENTITY,
	CustomerID INT NOT NULL,
	PolicyID INT NOT NULL,
	AgentID INT NOT NULL,
	StartDate DATETIME DEFAULT GetDate(),
	EndDate DATETIME,
	CONSTRAINT PK_asgnmtID PRIMARY KEY (AssignmentID),
	CONSTRAINT FK_custID FOREIGN KEY (CustomerID) REFERENCES
	Customers(CustomerID),
	CONSTRAINT FK_policyID FOREIGN KEY (PolicyID) REFERENCES
	Policies(PolicyID),
	CONSTRAINT FK_AgentID FOREIGN KEY (AgentID) REFERENCES Agents(AgentID)
);
```

Claims Table Query:

```sql
-- Claims Table
CREATE TABLE Claims
(
	ClaimID INT IDENTITY NOT NULL,
	AssignmentID INT NOT NULL,
	ClaimDate DATETIME NOT NULL DEFAULT GetDate(),
	ClaimAmount DECIMAL(10,2),
	ClaimStatus VARCHAR(30)
	CONSTRAINT PK_claimID PRIMARY KEY (ClaimID),
	CONSTRAINT FK_asgnmtID FOREIGN KEY (AssignmentID) REFERENCES
	PolicyAssignments(AssignmentID)

);
```

Insert commands for all tables.

Customers Table Insertion Query:

```sql
-- Customers Data Insertion
INSERT INTO Customers (FirstName, LastName, DateOfBirth, PhoneNumber, Email)
VALUES
('Amit', 'Sharma', '1998-05-12', '9876543210', 'amit@gmail.com'),
('Neha', 'Verma', '2003-07-20', '9876543211', 'neha@gmail.com'),
('Ravi', 'Kumar', '1995-03-15', '9876543212', 'ravi@gmail.com'),
('Sneha', 'Patel', '2001-01-10', '9876543213', 'sneha@gmail.com'),
('Arjun', 'Mehta', '2005-11-25', '9876543214', 'arjun@gmail.com'),
('Kiran', 'Rao', '1988-06-18', '9876543215', 'kiran@gmail.com');
```

Policies Table Insertion Query:

```sql
-- Policies Data Insertion
INSERT INTO Policies (PolicyName, PolicyType, PremiumAmount, DurationYears)
VALUES
('Life Shield', 'Life', 15000, 10),
('Health Plus', 'Health', 12000, 1),
('Motor Secure', 'Motor', 8000, 1),
('Health Gold', 'Health', 20000, 2),
('Life Premium', 'Life', 25000, 15);
```

Agnets Table Insertion Query:

```sql
-- Agents Data Insertion
INSERT INTO Agents (AgentName, Phone, City)
VALUES
('Rajesh', '9000011111', 'Delhi'),
('Anita', '9000011112', 'Mumbai'),
('Suresh', '9000011113', 'Bangalore'),
('Kavya', '9000011114', 'Chennai'),
('Mahesh', '9000011115', 'Jaipur');
```

PolicyAssignments Table Insertion Query:

```sql
-- PolicyAssignments Data Insertion
INSERT INTO PolicyAssignments (CustomerID, PolicyID, AgentID, StartDate,
EndDate)
VALUES
(1, 1, 1, '2020-01-01', '2030-01-01'),
(2, 2, 2, '2022-06-01', '2023-06-01'),
(3, 3, 3, '2021-03-01', '2022-03-01'),
(4, 4, 1, '2023-01-01', '2025-01-01'),
(5, 2, 4, '2024-02-01', NULL),
(6, 5, 5, '2019-05-01', '2034-05-01');
```

Claims Table Insertion Query:

```sql
-- Claims Data Insertion
INSERT INTO Claims (AssignmentID, ClaimDate, ClaimAmount, ClaimStatus)
VALUES
(1, '2022-05-10', 30000, 'Approved'),
(1, '2023-07-15', 25000, 'Rejected'),
(2, '2023-08-20', 15000, 'Approved'),
(3, '2022-01-10', 10000, 'Rejected'),
(4, '2024-06-01', 60000, 'Approved'),
(4, '2024-07-10', 45000, 'Rejected');
```

SELECT CLAUSE:

```sql
-- Display all records from the Customers table.
SELECT *
FROM Customers;

-- Display FirstName, LastName, and Email of all customers.
SELECT FirstName, LastName, Email
FROM Customers;

-- Display unique cities where agents are located.
SELECT DISTINCT City
FROM Agents;

-- Display PolicyName, PolicyType, and PremiumAmount
--    Rename PremiumAmount as AnnualPremium.
SELECT
    PolicyName,
    PolicyType,
    PremiumAmount AS AnnualPremium
FROM Policies;
```

WHERE CLAUSE:

```sql
-- Display customers whose FirstName starts with the letter 'A'.
SELECT *
FROM Customers
WHERE FirstName LIKE 'A%';

-- Display policies having PremiumAmount greater than 15000.
SELECT PolicyID, PolicyName, PremiumAmount
FROM Policies
WHERE PremiumAmount > 15000;

-- Display agents who belong to the city 'Delhi'.
SELECT AgentID, AgentName, City
FROM Agents
WHERE City = 'Delhi';

-- Display customers born between 1st Jan 2000 and 31st Dec 2010.
SELECT CustomerID, FirstName, LastName, DateOfBirth
FROM Customers
WHERE DateOfBirth BETWEEN '2000-01-01' AND '2010-12-31';

-- Display claims where ClaimStatus is 'Rejected'.
SELECT ClaimID, AssignmentID, ClaimAmount, ClaimStatus
FROM Claims
WHERE ClaimStatus = 'Rejected';
```

GROUPBY, ORDERBY & HAVING CLAUSE:

```sql
-- Display all customers ordered by LastName in ascending order.
SELECT CustomerID, FirstName, LastName
FROM Customers
ORDER BY LastName ASC;

-- Display all policies ordered by PremiumAmount in descending order.
SELECT PolicyID, PolicyName, PremiumAmount
FROM Policies
ORDER BY PremiumAmount DESC;

-- Display number of policies available for each PolicyType.
SELECT PolicyType, COUNT(*) AS PolicyCount
FROM Policies
GROUP BY PolicyType;

-- Display Agent-wise number of policy assignments
--    and show only agents having more than 1 policy assignment.
SELECT AgentID, COUNT(*) AS AssignmentCount
FROM PolicyAssignments
GROUP BY AgentID
HAVING COUNT(*) > 1
ORDER BY AssignmentCount DESC;
```

SQL FUNCTIONS:

```sql
-- Display customer full name in uppercase and show length of the full name (STRING
FUNCTIONS)
SELECT
    UPPER(CONCAT(FirstName, ' ', LastName)) AS CustomerNameUpper,
    LEN(CONCAT(FirstName, ' ', LastName)) AS NameLength
FROM Customers;

-- Display customers along with their age calculated from DateOfBirth (DATE FUNCTION)
SELECT
    CONCAT(FirstName, ' ', LastName) AS CustomerName,
    DATEDIFF(YEAR, DateOfBirth, GETDATE()) AS Age
FROM Customers;
-- Display total number of policies and average premium amount (AGGREGATE FUNCTIONS)
SELECT
    COUNT(*) AS TotalPolicies,
    AVG(PremiumAmount) AS AveragePremium
FROM Policies;

-- Display policy name and premium rounded to 2 decimals and converted to integer
(NUMERIC & CONVERSION FUNCTIONS)
SELECT
    PolicyName,
    ROUND(PremiumAmount, 2) AS RoundedPremium,
    CAST(PremiumAmount AS INT) AS PremiumAsInteger
FROM Policies;
```

Aggregation Functions:

```sql
--Count total number of customers.
SELECT COUNT(*) AS [TotalCustomers]
FROM Customers;

--Find the average premium amount of all policies.
SELECT ROUND(AVG(PremiumAmount), 2) AS AvgPremium
FROM Policies;

--Find the maximum claim amount.
SELECT MAX(claimAmount) AS 'Maximum Claim Amount'
FROM Claims;

--Calculate total claim amount for approved claims.
SELECT SUM(ClaimAmount) AS TotalApprovedClaims
FROM Claims
WHERE ClaimStatus = 'Approved';
```

JOINS:

```sql
-- Display all customers along with their assigned policy names (INNER JOIN)
SELECT
    CONCAT(c.FirstName, ' ', c.LastName) AS CustomerName,
    p.PolicyName
FROM Customers c
JOIN PolicyAssignments pa
    ON c.CustomerID = pa.CustomerID
JOIN Policies p
    ON pa.PolicyID = p.PolicyID;

-- Display all customers with or without policies (LEFT JOIN)
SELECT
    CONCAT(c.FirstName, ' ', c.LastName) AS CustomerName,
    p.PolicyName
FROM Customers c
LEFT JOIN PolicyAssignments pa
    ON c.CustomerID = pa.CustomerID
LEFT JOIN Policies p
    ON pa.PolicyID = p.PolicyID;

-- Display all policies with or without customer assignments (RIGHT JOIN)
SELECT
    p.PolicyName,
    c.FirstName
FROM Customers c
RIGHT JOIN PolicyAssignments pa
    ON c.CustomerID = pa.CustomerID
RIGHT JOIN Policies p
    ON pa.PolicyID = p.PolicyID;
```

```sql
-- Display all customers and all policies including unmatched records (FULL OUTER JOIN)
SELECT
    CONCAT(c.FirstName, ' ', c.LastName) AS CustomerName,
    p.PolicyName
FROM Customers c
FULL OUTER JOIN PolicyAssignments pa
    ON c.CustomerID = pa.CustomerID
FULL OUTER JOIN Policies p
    ON pa.PolicyID = p.PolicyID;

-- Display claim details along with customer name and policy name (MULTI-TABLE JOIN)
SELECT
    CONCAT(c.FirstName, ' ', c.LastName) AS CustomerName,
    p.PolicyName,
    cl.ClaimAmount,
    cl.ClaimStatus
FROM Customers c
JOIN PolicyAssignments pa
    ON c.CustomerID = pa.CustomerID
JOIN Policies p
    ON pa.PolicyID = p.PolicyID
JOIN Claims cl
    ON pa.AssignmentID = cl.AssignmentID;

-- Display all agents along with the number of policies they have handled (JOIN + GROUP BY)
SELECT
    a.AgentName,
    COUNT(pa.AssignmentID) AS PolicyHandled
FROM Agents a
LEFT JOIN PolicyAssignments pa
    ON a.AgentID = pa.AgentID
GROUP BY a.AgentName;
```

SUBQUERIES:

```sql
-- Display customers whose DateOfBirth is earlier than the average DateOfBirth of all customers
SELECT CustomerID, FirstName, LastName, DateOfBirth
FROM Customers
WHERE DateOfBirth <
     (SELECT AVG(DateOfBirth) FROM Customers);

-- Display policies having PremiumAmount greater than the average PremiumAmount
SELECT PolicyID, PolicyName, PremiumAmount
FROM Policies
WHERE PremiumAmount >
     (SELECT AVG(PremiumAmount) FROM Policies);
```

```sql
-- Display customers who have at least one policy assigned (IN subquery)
SELECT CustomerID, FirstName, LastName
FROM Customers
WHERE CustomerID IN (
    SELECT CustomerID
    FROM PolicyAssignments
);

-- Display customers who do NOT have any policy assigned (NOT IN subquery)
SELECT CustomerID, FirstName, LastName
FROM Customers
WHERE CustomerID NOT IN (
    SELECT CustomerID
    FROM PolicyAssignments
);

-- Display customers whose total claim amount is greater than the overall average
claim amount
SELECT DISTINCT c.CustomerID, c.FirstName, c.LastName
FROM Customers c
JOIN PolicyAssignments pa
    ON c.CustomerID = pa.CustomerID
JOIN Claims cl
    ON pa.AssignmentID = cl.AssignmentID
WHERE cl.ClaimAmount >
     (SELECT AVG(ClaimAmount) FROM Claims);
```

SET OPERATORS:

```sql
-- Display CustomerIDs who have taken Life policies OR Health policies (UNION)
SELECT pa.CustomerID
FROM PolicyAssignments pa
JOIN Policies p
    ON pa.PolicyID = p.PolicyID
WHERE p.PolicyType = 'Life'
UNION
SELECT pa.CustomerID
FROM PolicyAssignments pa
JOIN Policies p
    ON pa.PolicyID = p.PolicyID
WHERE p.PolicyType = 'Health';

-- Display all CustomerIDs from PolicyAssignments and Claims
-- including duplicates (UNION ALL)
SELECT CustomerID
FROM PolicyAssignments
UNION ALL
SELECT pa.CustomerID
FROM Claims cl
JOIN PolicyAssignments pa
    ON cl.AssignmentID = pa.AssignmentID;
```

```sql
-- Display CustomerIDs who have taken BOTH Life and Health policies (INTERSECT)
SELECT pa.CustomerID
FROM PolicyAssignments pa
JOIN Policies p
   ON pa.PolicyID = p.PolicyID
WHERE p.PolicyType = 'Life'
INTERSECT
SELECT pa.CustomerID
FROM PolicyAssignments pa
JOIN Policies p
   ON pa.PolicyID = p.PolicyID
WHERE p.PolicyType = 'Health';

-- Display CustomerIDs who have taken Life policies
-- but NOT Health policies (EXCEPT)
SELECT pa.CustomerID
FROM PolicyAssignments pa
JOIN Policies p
   ON pa.PolicyID = p.PolicyID
WHERE p.PolicyType = 'Life'
EXCEPT
SELECT pa.CustomerID
FROM PolicyAssignments pa
JOIN Policies p
   ON pa.PolicyID = p.PolicyID
WHERE p.PolicyType = 'Health';

-- Display PolicyIDs that are assigned to customers
-- but have never had any claims (EXCEPT)
SELECT PolicyID
FROM PolicyAssignments
EXCEPT
SELECT pa.PolicyID
FROM Claims cl
JOIN PolicyAssignments pa
   ON cl.AssignmentID = pa.AssignmentID;
```

CASE..ELSE, ROLLUP, CUBE & GROUPING SETS:

```sql
-- CASE..ELSE in SQL Server
SELECT CONCAT(FirstName, ' ',LastName) AS [Customer Name],
            DATEDIFF(YEAR, DateOfBirth, GetDATE()) AS Age,
            CASE
                  WHEN DATEDIFF(YEAR, DateOfBirth, GetDATE()) < 30 THEN 'Teenager'
                  WHEN DATEDIFF(YEAR, DateOfBirth, GetDATE()) BETWEEN 30 AND 40
THEN 'Adult'
                  ELSE 'Senior Citizen'
            END AS [Age Category]
FROM Customers
ORDER BY
```

```
	CASE
		WHEN DATEDIFF(YEAR, DateOfBirth, GetDATE()) > 40 THEN 1
		WHEN DATEDIFF(YEAR, DateOfBirth, GetDATE()) BETWEEN 30 AND 40 THEN 2
		ELSE 3
	END;

-- ROLLUP in SQL
SELECT PolicyType, DurationYears,COUNT(*) AS [Number Of Policies]
FROM Policies
GROUP BY ROLLUP(PolicyType,DurationYears);

-- CUBE in SQL
SELECT PolicyType, DurationYears,COUNT(*) AS [Number Of Policies]
FROM Policies
GROUP BY CUBE(PolicyType,DurationYears);

-- GROUPING SETS SQL
SELECT
    CASE
        WHEN GROUPING(PolicyType) = 0 THEN PolicyType
        ELSE 'ALL POLICIES'
    END AS PolicyType,
    CASE
        WHEN GROUPING(DurationYears) = 0 THEN CAST(DurationYears AS VARCHAR)
        ELSE 'ALL DURATIONS'
    END AS DurationYears,
    COUNT(*) AS TotalPolicies
FROM Policies
GROUP BY GROUPING SETS (
    (PolicyType),
    (DurationYears),
    ()
);
```

MERGE COMMAND:

```
-- Synchronize Policies table with a new source of policy data:
MERGE INTO Policies AS Target
USING (
    SELECT 1 AS PolicyID, 'Life Shield Plus' AS PolicyName, 'Life' AS PolicyType,
18000 AS PremiumAmount, 12 AS DurationYears
    UNION ALL
    SELECT 6, 'Health Secure Max', 'Health', 22000, 2
) AS Source
ON Target.PolicyID = Source.PolicyID
WHEN MATCHED THEN
    UPDATE SET
        Target.PolicyName = Source.PolicyName,
        Target.PolicyType = Source.PolicyType,
        Target.PremiumAmount = Source.PremiumAmount,
       Target.DurationYears = Source.DurationYears
```

```
WHEN NOT MATCHED BY TARGET THEN
    INSERT (PolicyName, PolicyType, PremiumAmount, DurationYears)
    VALUES      (Source.PolicyName,      Source.PolicyType,      Source.PremiumAmount,
Source.DurationYears);
```

CHECK CONSTRAINT:

```
-- Ensure PremiumAmount is always greater than 0
ALTER TABLE Policies
ADD CONSTRAINT CK_Policies_PremiumAmount
CHECK (PremiumAmount > 0);

-- Ensure DurationYears is between 1 and 30 years
ALTER TABLE Policies
ADD CONSTRAINT CK_Policies_DurationYears
CHECK (DurationYears BETWEEN 1 AND 30);

-- Ensure ClaimAmount is positive
ALTER TABLE Claims
ADD CONSTRAINT CK_Claims_ClaimAmount
CHECK (ClaimAmount > 0);

-- Ensure ClaimStatus allows only valid values
ALTER TABLE Claims
ADD CONSTRAINT CK_Claims_ClaimStatus
CHECK (ClaimStatus IN ('Approved', 'Rejected', 'Pending'));

-- Ensure PolicyAssignment EndDate is greater than StartDate (if EndDate is not NULL)
ALTER TABLE PolicyAssignments
ADD CONSTRAINT CK_PolicyAssignments_Dates
    CHECK (EndDate IS NULL OR EndDate > StartDate);
```

VIEWS:

```
--Create a view to see customer policy details
CREATE VIEW vw_CustomerPolicies AS
SELECT c.firstname, c.lastname, p.policyName, pa.startDate, pa.endDate
FROM customers c
JOIN policyAssignment pa ON c.customerId = pa.customerId
JOIN policies p ON pa.policyId = p.policyId;

--Display data from Customer Policies view
SELECT * FROM vw_CustomerPolicies;

--Create a view for claims report
CREATE VIEW vw_ClaimsReport AS
SELECT c.firstname, p.policyName, cl.claimAmount, cl.claimStatus, cl.claimDate
FROM claims cl
JOIN policyAssignment pa ON cl.assignmentId = pa.assignmentId
JOIN customers c ON pa.customerId = c.customerId
```

```sql
JOIN policies p ON pa.policyId = p.policyId;

--Display approved claims using the view
SELECT *
FROM vw_ClaimsReport
WHERE claimStatus = 'Approved';

--Create a view showing agent-wise policy count
CREATE VIEW vw_AgentPolicyCount AS
SELECT a.agentName, COUNT(pa.policyId) AS policyCount
FROM agents a
JOIN policyAssignment pa ON a.agentId = pa.agentId
GROUP BY a.agentName;
```

INDEXING:

```sql
--Create an index on customer email for faster search
CREATE INDEX idx_customers_email
ON customers(email);

--Create an index on policy type for filtering policies
CREATE INDEX idx_policies_policyType
ON policies(policyType);

--Create an index on agent city
CREATE INDEX idx_agents_city
ON agents(city);
```