Date: 31-12-25

Module 4.4 Practical Project Assignment

1. **Database Creation**

```
CREATE DATABASE InsuranceDB;
GO

USE InsuranceDB;
GO
```

2. **Tables Creation**

```
CREATE TABLE Customers (
    CustomerID INT IDENTITY(1,1) PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    DateOfBirth DATE,
    Phone VARCHAR(10),
    Email VARCHAR(50)
);

CREATE TABLE Agents (

    AgentID INT IDENTITY(1,1) PRIMARY KEY,
    AgentName VARCHAR(50),
    Phone VARCHAR(10),
    City VARCHAR(30)
);

CREATE TABLE Policies (
    PolicyID INT IDENTITY(1,1) PRIMARY KEY,
    PolicyName VARCHAR(30),
    PolicyType VARCHAR(25),
    PremiumAmount INT CHECK (PremiumAmount > 0),
    DurationYears INT
);


CREATE TABLE PolicyAssignments (
   AssignmentID INT IDENTITY(1,1) PRIMARY KEY,
    CustomerID INT NOT NULL,
    PolicyID INT NOT NULL,
    AgentID INT NOT NULL,
    StartDate DATE,
    EndDate DATE,

    CONSTRAINT fk_pa_customers
    FOREIGN KEY (CustomerID)
```

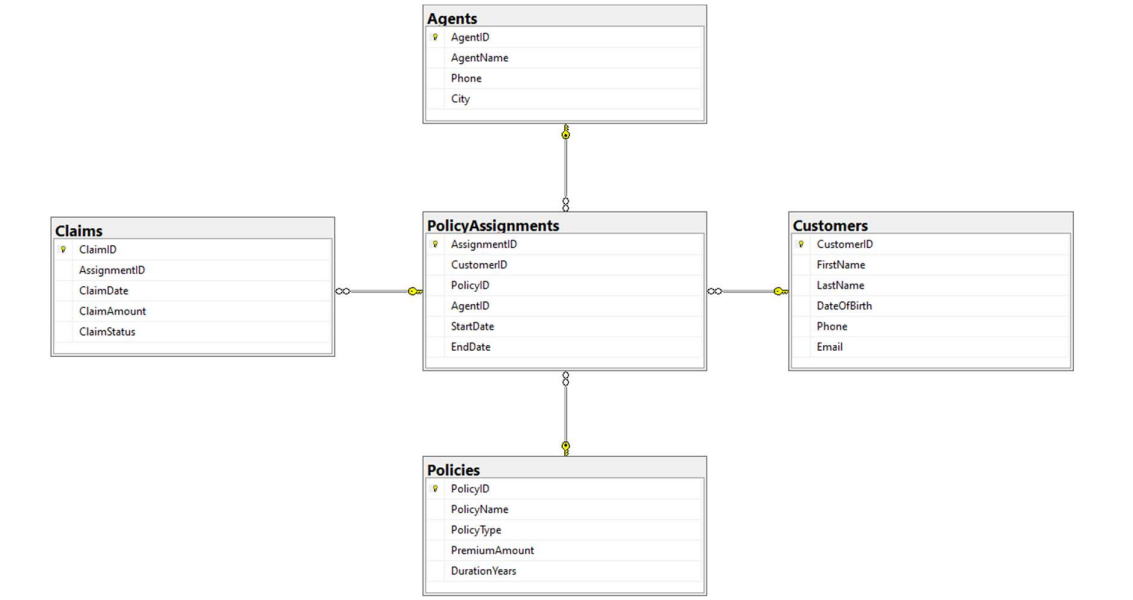
```
    REFERENCES Customers(CustomerID),

    CONSTRAINT fk_pa_policies
    FOREIGN KEY (PolicyID)
    REFERENCES Policies(PolicyID),

    CONSTRAINT fk_pa_agents
    FOREIGN KEY (AgentID)
    REFERENCES Agents(AgentID)
);


CREATE TABLE Claims (
    ClaimID INT IDENTITY(1,1) PRIMARY KEY,
    AssignmentID INT NOT NULL,
    ClaimDate DATE,
    ClaimAmount INT,
    ClaimStatus VARCHAR(20),

    CONSTRAINT fk_claims_assignments
    FOREIGN KEY (AssignmentID)
    REFERENCES PolicyAssignments(AssignmentID)
);
```



3. **INSERT Commands (Insert values)**

```
INSERT INTO Customers (FirstName, LastName, DateOfBirth, Phone,
Email)
VALUES
```

```
('Srujana', 'GL', '2003-05-12', '9876543210',
'srujana@gmail.com'),
('Harsh', 'K', '2005-03-18', '9000023456', 'harsh@gmail.com');

INSERT INTO Agents (AgentName, Phone, City)
VALUES
('Ravi Kumar', '8888888888', 'Bangalore'),
('Anita Sharma', '7777777777', 'Hyderabad');


INSERT INTO Policies (PolicyName, PolicyType, PremiumAmount,
DurationYears)
VALUES
('Life Shield', 'Life', 15000, 10),
('Health Secure', 'Health', 8000, 5);


INSERT INTO PolicyAssignments (CustomerID, PolicyID, AgentID,
StartDate, EndDate)
VALUES
(1, 1, 1, '2024-01-01', '2034-01-01'),
(2, 2, 2, '2024-02-01', '2029-02-01');


INSERT INTO Claims (AssignmentID, ClaimDate, ClaimAmount,
ClaimStatus)
VALUES
(1, '2024-06-10', 50000, 'Approved'),
(2, '2024-07-15', 20000, 'Pending');
```

4. **SELECT Commands**

   i. Select * from Customers

   ii. View all records of PolicyAssignment table with CustomerId, PolicyId, StartDate and
   EndDate columns only.
   SELECT CustomerID, PolicyID, StartDate, EndDate FROM policyAssignments;

   iii. Display unique city names from where agents belong to.
   SELECT DISTINCT City FROM Agents;

   iv. Display list of customers born after January 1 st , 2001 and before December 31 st , 2020 using >= and <= operators.
   SELECT * FROM Customers WHERE DateOfBirth >= '2001-01-01' AND DateOfBirth <= '2020-12-31';

v. Display no of claims rejected.
SELECT COUNT(*) AS RejectedClaimsCount FROM Claims WHERE ClaimStatus = 'Rejected';

vi. Delete the record of PolicyAssignments whose EndDate is before today's date.
DELETE FROM PolicyAssignments WHERE EndDate < GETDATE();

5. String Functions Commands

UPPER()
SELECT UPPER(CustomerName) AS CustomerName FROM Customer;

LOWER()
SELECT LOWER(City) AS City FROM Customer;

LENGTH() / LEN()
SELECT CustomerName, LENGTH(CustomerName) AS NameLength
FROM Customer;

SUBSTRING() / SUBSTR()
SELECT SUBSTRING(CustomerName, 1, 4) AS ShortName FROM Customer;

CONCAT()
SELECT CONCAT(CustomerName, ' - ', City) AS CustomerDetails FROM Customer;

TRIM()
SELECT TRIM(CustomerName) AS TrimmedName FROM Customer;

RIGHT()
SELECT RIGHT(CustomerName, 4) AS LastFourChars
FROM Customer;

REPLACE()
SELECT REPLACE(CustomerName, 'a', '@') AS ReplacedName
FROM Customer;

6. Date Functions Commands

CURRENT_DATE / CURDATE()
SELECT CURRENT_DATE;

YEAR()
SELECT AssignmentID

FROM PolicyAssignment
WHERE YEAR(StartDate) = 2023;

MONTH()
SELECT AssignmentID
FROM PolicyAssignment
WHERE MONTH(StartDate) = 5;

DAY()
SELECT AssignmentID
FROM PolicyAssignment
WHERE DAY(StartDate) = 12;

DATEDIFF()
SELECT AssignmentID,
DATEDIFF(CURRENT_DATE, StartDate) AS DaysSincePolicyStart
FROM PolicyAssignment;

7. JOIN Commands

i. List all Policies for a CustomerId 5.

```
SELECT p.*
FROM PolicyAssignments pa
JOIN Policies p
ON pa.PolicyID = p.PolicyID
WHERE pa.CustomerID = 5;
```

ii. View all customers with their policies.

```
SELECT
    c.FirstName,
    c.LastName,
    p.PolicyName,
    p.PolicyType
FROM Customers c
JOIN PolicyAssignments pa
ON c.CustomerID = pa.CustomerID
JOIN Policies p
ON pa.PolicyID = p.PolicyID;
```

iii. Display records of Customers with or without Policies.

```
SELECT c.FirstName, p.PolicyName  FROM Customers c
LEFT JOIN PolicyAssignments pa
ON c.CustomerID = pa.CustomerID LEFT JOIN Policies p
ON pa.PolicyID = p.PolicyID;
```

iv. Display claims report with FirstName, PolicyName, ClaimAmount, ClaimStatus, and ClaimDate from their respective tables.

```
SELECT c.FirstName, p.PolicyName, cl.ClaimAmount,
cl.ClaimStatus, cl.ClaimDate FROM Claims cl JOIN
PolicyAssignments pa ON cl.AssignmentID =
pa.AssignmentID JOIN Customers c ON pa.CustomerID =
c.CustomerID JOIN Policies p ON pa.PolicyID =
p.PolicyID;
```

v. Display FirstName, PolicyName, AgentName, StartDate and EndDate from their respective tables.

```
SELECT c.FirstName, p.PolicyName, a.AgentName,
pa.StartDate, pa.EndDate FROM PolicyAssignments pa
JOIN Customers c ON pa.CustomerID = c.CustomerID JOIN
Policies p ON pa.PolicyID = p.PolicyID JOIN Agents a
ON pa.AgentID = a.AgentID;
```

8. SUBQUERIES Commands

i. Display agents who handle at least one policy

```
SELECT DISTINCT a.AgentName FROM Agents a JOIN PolicyAssignments pa
ON a.AgentID = pa.AgentID;
```

ii. Find customers who have a policy with premium > 50,000

```
SELECT DISTINCT c.CustomerName
FROM Customers c
JOIN PolicyAssignments pa ON c.CustomerID = pa.CustomerID
JOIN Policies p ON pa.PolicyID = p.PolicyID
WHERE p.PremiumAmount > 50000;
```

iii. List agents who handle policies whose premium is greater than ANY premium of Life policies

```
SELECT DISTINCT a.AgentName
FROM Agents a
JOIN PolicyAssignments pa ON a.AgentID = pa.AgentID
JOIN Policies p ON pa.PolicyID = p.PolicyID
WHERE p.PremiumAmount > ANY (
  SELECT PremiumAmount
  FROM Policies
  WHERE PolicyType = 'Life'
);
```

iv. Find policies whose premium is greater than ANY premium of policies having claims

```
SELECT PolicyID, PolicyType, PremiumAmount
FROM Policies
WHERE PremiumAmount > ANY (
  SELECT p.PremiumAmount
  FROM Policies p
```

```
    JOIN Claims c ON p.PolicyID = c.PolicyID
);
```

v. List customers whose policy premium is greater than ALL premiums of customers having claims.

```
SELECT DISTINCT c.CustomerName
FROM Customers c
JOIN PolicyAssignments pa ON c.CustomerID = pa.CustomerID
JOIN Policies p ON pa.PolicyID = p.PolicyID
WHERE p.PremiumAmount > ALL (
   SELECT p2.PremiumAmount
   FROM Policies p2
   JOIN Claims cl ON p2.PolicyID = cl.PolicyID
);
```

9. CASE…ELSE Commands

i. Categorize Policies by Premium Amount

```
SELECT PolicyID, PolicyType, PremiumAmount,
CASE
   WHEN PremiumAmount >= 50000 THEN 'High Premium'
   WHEN PremiumAmount >= 20000 THEN 'Medium Premium'
   ELSE 'Low Premium'
END AS PremiumCategory
FROM Policy;
```

ii.

```
SELECT ClaimID, ClaimAmount,
CASE
  WHEN ClaimAmount > 50000 THEN 'Large Claim'
   WHEN ClaimAmount > 20000 THEN 'Medium Claim'
   ELSE 'Small Claim'
END AS ClaimCategory
FROM Claim;
```

10. Merge Commands

i. MERGE to Insert or Update Customers

```
MERGE INTO Customer c
USING (
   SELECT 6 AS CustomerID, 'Ananya Rao' AS CustomerName, 'Bangalore' AS City
) src
ON (c.CustomerID = src.CustomerID)

WHEN MATCHED THEN
   UPDATE SET c.City = src.City

WHEN NOT MATCHED THEN
```

```
    INSERT (CustomerID, CustomerName, City)
    VALUES (src.CustomerID, src.CustomerName, src.City);
```

ii. MERGE Claims (Insert new claims, update amount if exists)

```
MERGE INTO Claim c
USING (
    SELECT 204 AS ClaimID, 105 AS PolicyID, 75000 AS ClaimAmount
) src
ON c.ClaimID = src.ClaimID

WHEN MATCHED THEN
    UPDATE SET c.ClaimAmount = src.ClaimAmount

WHEN NOT MATCHED THEN
    INSERT (ClaimID, PolicyID, ClaimAmount)
    VALUES (src.ClaimID, src.PolicyID, src.ClaimAmount);
```

11. Roll Up Commands

i. Total Premium Amount by Policy Type with Grand Total

```
SELECT
    PolicyType,
    SUM(PremiumAmount) AS TotalPremium
FROM Policy
GROUP BY ROLLUP (PolicyType);
```

ii. Customer-wise Premium Total with Overall Total

```
SELECT
    c.CustomerName,
    SUM(p.PremiumAmount) AS TotalPremium
FROM Customer c
JOIN PolicyAssignment pa ON c.CustomerID = pa.CustomerID
JOIN Policy p ON pa.PolicyID = p.PolicyID
GROUP BY ROLLUP (c.CustomerName);
```

12. Cube Commands

i. Total Premium by Policy Type with all subtotals

```
SELECT
    PolicyType,
    SUM(PremiumAmount) AS TotalPremium
FROM Policy
GROUP BY CUBE (PolicyType);
```

ii. Claim Amount by Customer and Policy Type

```
SELECT
    c.CustomerName,
```

```
    p.PolicyType,
    SUM(cl.ClaimAmount) AS TotalClaimAmount
FROM Customer c
JOIN PolicyAssignment pa ON c.CustomerID = pa.CustomerID
JOIN Policy p ON pa.PolicyID = p.PolicyID
JOIN Claim cl ON p.PolicyID = cl.PolicyID
GROUP BY CUBE (c.CustomerName, p.PolicyType);
```

13. Grouping sets Commands

i. Premium by Policy Type and Grand Total only

```
SELECT
    PolicyType,
    SUM(PremiumAmount) AS TotalPremium
FROM Policy
GROUP BY GROUPING SETS (
    (PolicyType),
    ()
);
```

ii. Claim Amount by Policy Type + Overall Claim Total

```
SELECT
    p.PolicyType,
    SUM(cl.ClaimAmount) AS TotalClaimAmount
FROM Policy p
JOIN Claim cl ON p.PolicyID = cl.PolicyID
GROUP BY GROUPING SETS (
    (p.PolicyType),
    ()
);
```