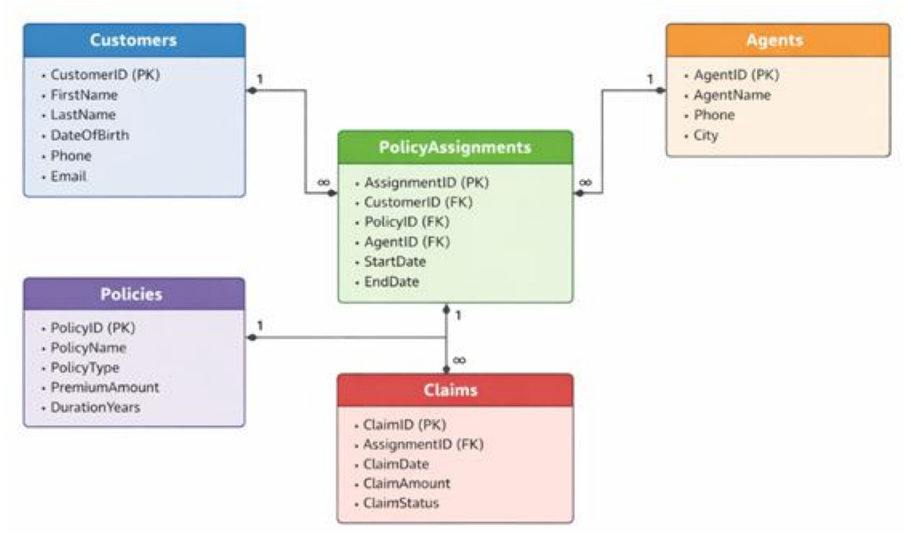
Name: Nanda kishor Gudala

1. Create Database command.
   Query: create database insuranceDB;

SCHEMA



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

```
CREATE TABLE Customers (
   CustomerID   INT IDENTITY(1,1) PRIMARY KEY,
   FirstName    VARCHAR(50) NOT NULL,
   LastName     VARCHAR(50) NOT NULL,
   DateOfBirth  DATE NULL,
   Phone        VARCHAR(20) NULL,
   Email        VARCHAR(100) NULL
);


CREATE TABLE Policies (
   PolicyID      INT IDENTITY(1,1) PRIMARY KEY,
   PolicyName    VARCHAR(100) NOT NULL,
    PolicyType     VARCHAR(50) NOT NULL,
    PremiumAmount  DECIMAL(10,2) NOT NULL,
    DurationYears  INT NOT NULL
);


CREATE TABLE Agents (
    AgentID    INT IDENTITY(1,1) PRIMARY KEY,
    AgentName  VARCHAR(100) NOT NULL,
    Phone      VARCHAR(20) NULL,
    City       VARCHAR(50) NULL
);

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

    CONSTRAINT FK_PolicyAssignments_Customers
        FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),

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

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


CREATE TABLE Claims (
    ClaimID      INT IDENTITY(1,1) PRIMARY KEY,
    AssignmentID INT NOT NULL,
    ClaimDate    DATE NOT NULL,
    ClaimAmount  DECIMAL(10,2) NOT NULL,
    ClaimStatus  VARCHAR(50) NOT NULL,

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

3. Insert commands for all tables.

```
INSERT INTO Customers (FirstName, LastName, DateOfBirth, Phone, Email) VALUES
('Koushik',     'Reddy',     '2004-03-12', '9876500011', 'koushik.reddy@gmail.com'),
('Rithvik',     'Sharma',    '2004-07-25', '9876500012', 'rithvik.sharma@gmail.com'),
```

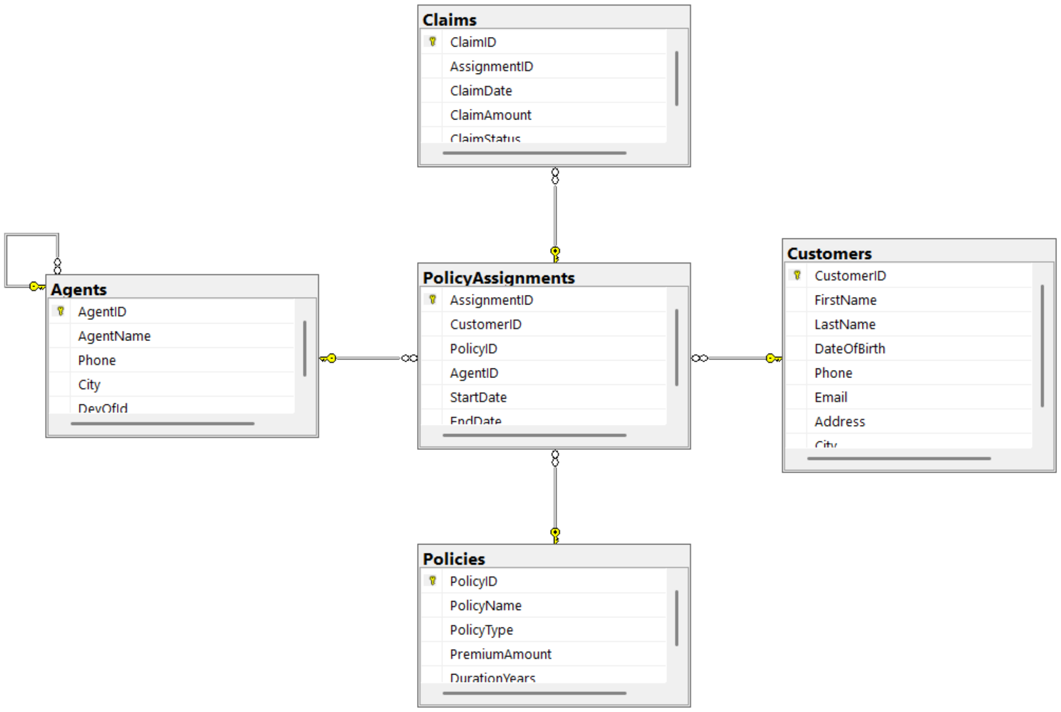
```
INSERT INTO Policies (PolicyName, PolicyType, PremiumAmount, DurationYears)
VALUES
('Health Secure Basic',      'Health',  8000.00,  1),
('Health Secure Plus',       'Health', 15000.00,  1),
```

```
INSERT INTO Agents (AgentName, Phone, City) VALUES
('Ramesh',      '9876500101', 'Hyderabad'),
('Bhoopendhar', '9876500102', 'Warangal'),
```

```
INSERT INTO PolicyAssignments (CustomerID, PolicyID, AgentID, StartDate, EndDate)
VALUES
(1, 1, 1, '2024-01-10', '2025-01-09'),
(2, 2, 2, '2024-03-15', '2025-03-14'),
```

```
INSERT INTO Claims (AssignmentID, ClaimDate, ClaimAmount, ClaimStatus) VALUES
(3,  '2023-08-21', 18500.00, 'Approved'),
(1,  '2024-04-14',  3200.00, 'Rejected'),
```

DATABASE DIAGRAM



4. **Basic commands**

1. List policies of type Life, Health, Motor use OR clause.

   Query:

```
SELECT *
FROM Policies
WHERE PolicyType = 'Life'
  OR PolicyType = 'Health'
  OR PolicyType = 'Motor';Output:
```

use IN operator.

Query:

```
Query:
SELECT *
FROM Policies
WHERE PolicyType in ('Life','Health','Motor');
```

2. Display list of customers born after January 1st, 2001 and before December 31st, 2020 using >= and <= operators.

   Query:

```
SELECT *
FROM Customers
WHERE DateOfBirth >= '2001-01-01'
  AND DateOfBirth <= '2020-12-31';
```

3. 2020 using between operator.

   Query:

```
SELECT *
FROM Customers
WHERE DateOfBirth BETWEEN '2001-01-01' AND '2020-12-31';
```

4. Display latest claim record.

   Query:

```
SELECT *
FROM Claims
ORDER BY ClaimDate DESC
OFFSET 0 ROWS FETCH NEXT 1 ROW ONLY;
```

5. Increase premium amount to 10% for all health insurance policies.

   Query:

```
UPDATE Policies
SET PremiumAmount = PremiumAmount * 1.10
WHERE PolicyType = 'Health';
```

6. Delete the record of PolicyAssignments whose EndDate is before today's date.

```
delete from claims
where policyassignmentid in (
   select policyassignmentid
   from policyassignments
   where enddate < cast(getdate() as date)
);

delete from policyassignments
where enddate < cast(getdate() as date);
```

7. Write command to make the above DevOfId as a recursive foreign key to AgentId as Parent.

Query:

```
ALTER TABLE Agents
ADD CONSTRAINT FK_Agents_DevOfId
FOREIGN KEY (DevOfId) REFERENCES Agents(AgentID);
```

Pattern Matching

1) Customers whose first name starts with 'Na'

```
SELECT *
FROM Customers
WHERE FirstName LIKE 'Na%';
```

2) Customers whose email ends with gmail.com

```
SELECT *
FROM Customers
WHERE Email LIKE '%@gmail.com';
```

3) Agents whose city contains 'd' anywhere (case-insensitive)

```
SELECT *
FROM Agents
WHERE LOWER(City) LIKE '%d%';
```

4) Policies whose name is exactly 3 letters (e.g., Car) using underscores

```
SELECT *
```

```
FROM Policies
WHERE PolicyName LIKE '___';
```

5) Claims where status starts with 'A' or 'P' (Approved / Pending)

```
SELECT *
FROM Claims
WHERE ClaimStatus LIKE 'A%' OR ClaimStatus LIKE 'P%';
```

Date Functions:

1) CUSTOMERS: Calculate exact-ish age (handles birthday not yet happened this year)

```
SELECT customerID, firstName, lastName, dateOfBirth,
      DATEDIFF(YEAR, dateOfBirth, GETDATE())
      - CASE WHEN DATEADD(YEAR, DATEDIFF(YEAR, dateOfBirth, GETDATE()),
dateOfBirth) > CAST(GETDATE() AS date)
            THEN 1 ELSE 0 END AS AgeYears
FROM Customers;
```

2) POLICYASSIGNMENTS: Find assignments ending in the next 30 days

```
SELECT AssignmentID, CustomerID, AgentID, StartDate, EndDate
FROM PolicyAssignments
WHERE EndDate BETWEEN CAST(GETDATE() AS date) AND DATEADD(DAY, 30,
CAST(GETDATE() AS date));
```

3) CLAIMS: Month-wise total claims and amount (using YEAR/MONTH)

```
SELECT YEAR(ClaimDate) AS ClaimYear,
      MONTH(ClaimDate) AS ClaimMonth,
      COUNT(*) AS TotalClaims,
      SUM(ClaimAmount) AS TotalClaimAmount
FROM Claims
GROUP BY YEAR(ClaimDate), MONTH(ClaimDate)
ORDER BY ClaimYear, ClaimMonth;
```

4) POLICIES: Compute a "sample maturity date" if a policy starts today (DATEADD)

```
SELECT PolicyID, PolicyName, DurationYears,
    CAST(GETDATE() AS date) AS SampleStartDate,
    DATEADD(YEAR, DurationYears, CAST(GETDATE() AS date)) AS SampleMaturityDate
FROM Policies;
```

5) AGENTS: Show a follow-up date 7 days from today for every agent (DATEADD + GETDATE)

```
SELECT AgentID, AgentName, City,
    CAST(GETDATE() AS date) AS Today,
    DATEADD(DAY, 7, CAST(GETDATE() AS date)) AS FollowUpDate
FROM Agents;
```

String Functions:

1) CUSTOMERS: Concatenate first + last name (CONCAT)

```
SELECT CustomerID,
    CONCAT(FirstName, ' ', LastName) AS FullName,
    Email
FROM Customers;
```

2) AGENTS: Convert agent name to UPPER and city to LOWER (UPPER, LOWER)

```
SELECT AgentID,
    UPPER(AgentName) AS AgentName_Upper,
    LOWER(City) AS City_Lower
FROM Agents;
```

3) CUSTOMERS: Get first 3 letters of first name (LEFT)

```
SELECT CustomerID, FirstName,
    LEFT(FirstName, 3) AS First3Chars
FROM Customers;
```

4) POLICIES: Find policies whose name contains 'he' anywhere (CHARINDEX)

```
SELECT PolicyID, PolicyName
FROM Policies
WHERE CHARINDEX('he', LOWER(PolicyName)) > 0;
```

5) CLAIMS: Show claim status length and remove extra spaces (LEN, LTRIM, RTRIM)

```
SELECT ClaimID,
       ClaimStatus,
       LEN(ClaimStatus) AS StatusLength,
       LTRIM(RTRIM(ClaimStatus)) AS CleanStatus
FROM Claims;
```

1) Constraints (2)

a) UNIQUE constraint on customer email

```
ALTER TABLE Customers
ADD CONSTRAINT UQ_Customers_Email UNIQUE (Email);
```

b) CHECK constraints (valid claimStatus + positive amount)

```
ALTER TABLE Claims
ADD CONSTRAINT CK_Claims_Status_Amount
CHECK (ClaimStatus IN ('Approved','Rejected','Pending') AND ClaimAmount > 0);
```

2) Calculated fields (2)

a) Policy assignment duration in days + months

```
SELECT AssignmentID, CustomerID, PolicyID, AgentID, StartDate, EndDate,
       DATEDIFF(DAY, StartDate, EndDate)   AS DurationDays,
       DATEDIFF(MONTH, StartDate, EndDate) AS DurationMonths
FROM PolicyAssignments;
```

b) Claim amount with 18% GST (example calculation)

```
SELECT ClaimID, AssignmentID, ClaimAmount,
       ClaimAmount * 0.18 AS GST_18,
       ClaimAmount * 1.18 AS TotalWithGST
FROM Claims;
```

3) CASE WHEN (2)

a) Categorize claim amount

```
SELECT ClaimID, ClaimAmount,
       CASE
         WHEN ClaimAmount >= 15000 THEN 'High'
         WHEN ClaimAmount >= 8000  THEN 'Medium'
         ELSE 'Low'
       END AS ClaimCategory
FROM Claims;
```

b) Policy status: Active/Expired (based on EndDate)

```
SELECT AssignmentID, CustomerID, PolicyID, StartDate, EndDate,
       CASE
         WHEN EndDate < CAST(GETDATE() AS date) THEN 'Expired'
         ELSE 'Active'
```

```
    END AS PolicyStatus
FROM PolicyAssignments;
```

Joins :

1. Display records of Customers with or without Policies.
   Query:
   ```
   SELECT
      c.CustomerID,
      c.FirstName,
      c.LastName,
      p.PolicyID,
      p.PolicyName,
      pa.StartDate,
      pa.EndDate
   FROM Customers c
   LEFT JOIN PolicyAssignments pa ON pa.CustomerID = c.CustomerID
   LEFT JOIN Policies p ON p.PolicyID = pa.PolicyID
   ```

2. Display all Customers with NO Claims.
   Query:
   ```
   SELECT c.CustomerID, c.FirstName, c.LastName
   FROM Customers c
   WHERE NOT EXISTS (
      SELECT 1
      FROM PolicyAssignments pa
      JOIN Claims cl ON cl.AssignmentID = pa.AssignmentID
      WHERE pa.CustomerID = c.CustomerID
   );
   ```

3. Show CustomerName with Total Claim Amount per Customer.
   Query:
   ```
   SELECT
      c.FirstName + ' ' + c.LastName AS CustomerName,
      SUM(cl.ClaimAmount) AS TotalClaimAmount
   FROM Customers c
   ```

```
JOIN PolicyAssignments pa ON pa.CustomerID = c.CustomerID
JOIN Claims cl ON cl.AssignmentID = pa.AssignmentID
GROUP BY c.FirstName, c.LastName
ORDER BY TotalClaimAmount DESC;
```

4. Show names and total claim amount of Customers With Claim Amount > 50000 (Use HAVING Clause).
   Query:
   ```
   SELECT
      c.FirstName + ' ' + c.LastName AS CustomerName,
      SUM(cl.ClaimAmount) AS TotalClaimAmount
   FROM Customers c
   JOIN PolicyAssignments pa ON pa.CustomerID = c.CustomerID
   JOIN Claims cl ON cl.AssignmentID = pa.AssignmentID
   GROUP BY c.FirstName, c.LastName
   HAVING SUM(cl.ClaimAmount) > 50000
   ORDER BY TotalClaimAmount DESC;
   ```

5. Display list with Agent Wise Policy Count.
   Query:
   ```
   SELECT
      a.AgentID,
      a.AgentName,
      COUNT(pa.AssignmentID) AS PolicyCount
   FROM Agents a
   LEFT JOIN PolicyAssignments pa ON pa.AgentID = a.AgentID
   GROUP BY a.AgentID, a.AgentName
   ORDER BY PolicyCount DESC;
   ```

Nesetd Queries:

1) Customers who have at least one claim
   ```
   SELECT CustomerID, FirstName, LastName
   FROM Customers
   WHERE CustomerID IN (
      SELECT CustomerID
      FROM PolicyAssignments
      WHERE AssignmentID IN (
   ```

```
        SELECT AssignmentID
        FROM Claims
    )
);
```

2) Agents who handled the maximum number of assignments

```
SELECT AgentID, AgentName, City
FROM Agents
WHERE AgentID IN (
    SELECT AgentID
    FROM PolicyAssignments
    GROUP BY AgentID
    HAVING COUNT(*) = (
        SELECT MAX(cnt)
        FROM (
            SELECT COUNT(*) AS cnt
            FROM PolicyAssignments
            GROUP BY AgentID
        ) x
    )
);
```

3) Policies whose premium is greater than the average premium

```
SELECT PolicyID, PolicyName, PremiumAmount
FROM Policies
WHERE PremiumAmount > (
    SELECT AVG(PremiumAmount)
    FROM Policies
);
```

4) Customers who have claims with amount greater than their own average claim amount

```
SELECT c.ClaimID, c.AssignmentID, c.ClaimAmount
FROM Claims c
WHERE c.ClaimAmount > (
    SELECT AVG(c2.ClaimAmount)
    FROM Claims c2
    WHERE c2.AssignmentID IN (
        SELECT pa.AssignmentID
        FROM PolicyAssignments pa
        WHERE pa.CustomerID = (
            SELECT pa2.CustomerID
            FROM PolicyAssignments pa2
            WHERE pa2.AssignmentID = c.AssignmentID
```

```
                )
            )
        );
```
5) Customers who have no assignments
```
        SELECT CustomerID, FirstName, LastName
        FROM Customers
        WHERE CustomerID NOT IN (
            SELECT CustomerID
            FROM PolicyAssignments
        );
```

4) MERGE (2)

a) Upsert Agents (update if exists, insert if new)
```
MERGE INTO Agents AS tgt
USING (VALUES
    (1, 'Rajeev',      '878900875', 'Hyd'),
    (5, 'Sandeep',     '900001111', 'Vizag')
) AS src(AgentID, AgentName, Phone, City)
ON tgt.AgentID = src.AgentID
WHEN MATCHED THEN
    UPDATE SET tgt.AgentName = src.AgentName, tgt.Phone = src.Phone, tgt.City = src.City
WHEN NOT MATCHED THEN
    INSERT (AgentID, AgentName, Phone, City)
    VALUES (src.AgentID, src.AgentName, src.Phone, src.City);
```
b) Upsert Policies (update/insert by PolicyName)

```
MERGE INTO Policies AS tgt
USING (VALUES
    ('Health', 5000, 3),
    ('Travel', 4500, 2)
) AS src(PolicyName, PremiumAmount, DurationYears)
ON tgt.PolicyName = src.PolicyName
WHEN MATCHED THEN
    UPDATE SET tgt.PremiumAmount = src.PremiumAmount, tgt.DurationYears =
src.DurationYears
WHEN NOT MATCHED THEN
    INSERT (PolicyName, PremiumAmount, DurationYears)
    VALUES (src.PolicyName, src.PremiumAmount, src.DurationYears);
```

5) ROLLUP (2)

a) Claims total by Status + grand total
```
SELECT ClaimStatus,
      SUM(ClaimAmount) AS TotalAmount
FROM Claims
GROUP BY ROLLUP (ClaimStatus);
```

b) Claims totals by Year, Month + subtotals + grand total

```
SELECT YEAR(ClaimDate) AS ClaimYear,
      MONTH(ClaimDate) AS ClaimMonth,
      SUM(ClaimAmount) AS TotalAmount
FROM Claims
GROU
P BY ROLLUP (YEAR(ClaimDate), MONTH(ClaimDate))
ORDER BY ClaimYear, ClaimMonth;
```

6) CUBE (2)

a) Claims totals by (Status, Year) with all combinations

```
SELECT ClaimStatus,
      YEAR(ClaimDate) AS ClaimYear,
      SUM(ClaimAmount) AS TotalAmount
FROM Claims
GROUP BY CUBE (ClaimStatus, YEAR(ClaimDate))
ORDER BY ClaimStatus, ClaimYear;
```

b) Count assignments by (Agent, Policy) with all combinations

```
SELECT AgentID, PolicyID,
      COUNT(*) AS TotalAssignments
FROM PolicyAssignments
GROUP BY CUBE (AgentID, PolicyID)
ORDER BY AgentID, PolicyID;
```

7) GROUPING SETS (2)

a) Claims summary: (Year,Month), (Year), (Grand Total)

```
SELECT YEAR(ClaimDate) AS ClaimYear,
      MONTH(ClaimDate) AS ClaimMonth,
      SUM(ClaimAmount) AS TotalAmount
FROM Claims
GROUP BY GROUPING SETS (
   (YEAR(ClaimDate), MONTH(ClaimDate)),
   (YEAR(ClaimDate)),
   ()
)
ORDER BY ClaimYear, ClaimMonth;
```

b) Agents summary: by City, by AgentName, and grand total

```
SELECT City, AgentName,
      COUNT(*) AS TotalAgents
```

```
FROM Agents
GROUP BY GROUPING
SETS (
    (City),
    (AgentName),
    ()
)
ORDER BY City, AgentName;
```