INSURANCE AUTOMATION

GROUP MEMBERS

CHITRALEKHA.CH	– MBA202325-067
MOUNIKA BODEPU	– MBA202325-119
SAI SRI PONNURU	– MBA202325-188
PRAVEEN KR	– MBA202325-158

TABLES CREATED:

1) **Customers**

Store customer details

2) **Policies**

Stores different types of policies (Life, Health, Vehicle, Asset)

3) **Customer_Policies**

Links customers to the policies they own

4) **Customer_History**

Tracks historical records of deleted customers for future reference

5) **Agents**

Stores details of agents who help customers with policy payments and claims

6) **Employees**

Stores employee details for the insurance company

7) **Claims**

Stores claim requests made by customers

8) **Payments**

Stores payment information for customer policies

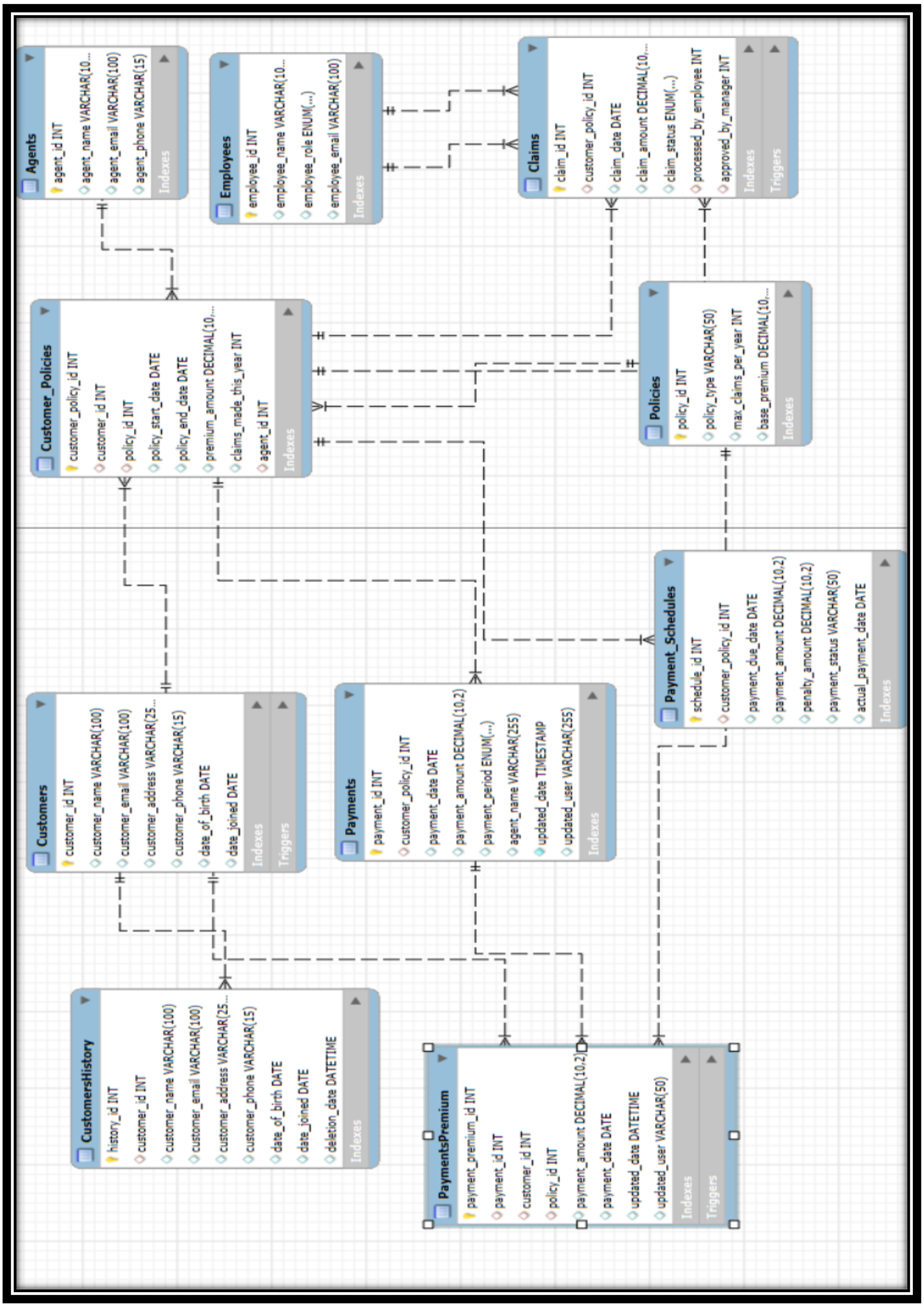
9) Payment_Schedules

Payment Schedules table to track payment dues

10) Premium_Payments

Stores the schedule of premium payments

ER DIAGRAM:



Agents
agent_id INT
agent_name VARCHAR(10...
agent_email VARCHAR(100)
agent_phone VARCHAR(15)
Employees
employee_id INT
employee_name VARCHAR(10...
employee_role ENUM(...)
employee_email VARCHAR(100)
Claims
claim_id INT
customer_policy_id INT
claim_date DATE
claim_amount DECIMAL(10,...
claim_status ENUM(...)
processed_by_employee INT
approved_by_manager INT
Customer_Policies
customer_policy_id INT
customer_id INT
policy_id INT
policy_start_date DATE
policy_end_date DATE
premium_amount DECIMAL(10,...
claims_made_this_year INT
agent_id INT
Policies
policy_id INT
policy_type VARCHAR(50)
max_claims_per_year INT
base_premium DECIMAL(10,...
Customers
customer_id INT
customer_name VARCHAR(100)
customer_email VARCHAR(100)
customer_address VARCHAR(25...
customer_phone VARCHAR(15)
date_of_birth DATE
date_joined DATE
Payments
payment_id INT
customer_policy_id INT
payment_date DATE
payment_amount DECIMAL(10,2)
payment_period ENUM(...)
agent_name VARCHAR(255)
updated_date TIMESTAMP
updated_user VARCHAR(255)
Payment_Schedules
schedule_id INT
customer_policy_id INT
payment_due_date DATE
payment_amount DECIMAL(10,2)
penalty_amount DECIMAL(10,2)
payment_status VARCHAR(50)
actual_payment_date DATE
CustomersHistory
history_id INT
customer_id INT
customer_name VARCHAR(100)
customer_email VARCHAR(100)
customer_address VARCHAR(25...
customer_phone VARCHAR(15)
date_of_birth DATE
date_joined DATE
deletion_date DATETIME
PaymentsPremium
payment_premium_id INT
payment_id INT
customer_id INT
policy_id INT
payment_amount DECIMAL(10,2)
payment_date DATE
updated_date DATETIME
updated_user VARCHAR(50)
Indexes
Triggers

SQL CODE :

CREATE AND USE A DATABASE:

```
create database INSAUT;

use INSAUT;
```

CREATE CUSTOMERS TABLE:

```
CREATE TABLE Customers (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_name VARCHAR(100),
    customer_email VARCHAR(100),
    customer_address VARCHAR(255),
    customer_phone VARCHAR(15),
    date_of_birth DATE,
    date_joined DATE
);
```

INSERT SOME CUSTOMERS:

```
INSERT INTO Customers (customer_name, customer_email, customer_address,
customer_phone, date_of_birth, date_joined)

VALUES ('Chitra', 'chitra@gmail.com', '123 Main St', '555-1234', '2001-03-26', CURDATE());

INSERT INTO Customers (customer_name, customer_email, customer_address,
customer_phone, date_of_birth, date_joined)

VALUES ('Saisri', 'saisri@gmail.com', '234 Main St', '555-2345', '1999-08-03', CURDATE());

INSERT INTO Customers (customer_name, customer_email, customer_address,
customer_phone, date_of_birth, date_joined)

VALUES ('Mounika', 'mounika@gmail.com', '345 Main St', '555-3456', '2001-08-21',
CURDATE());

INSERT INTO Customers (customer_name, customer_email, customer_address,
customer_phone, date_of_birth, date_joined)
```

```
VALUES ('Praveen', 'praveen@gmail.com', '456 Main St', '555-4567', '1999-08-11',
CURDATE());

INSERT INTO Customers (customer_name, customer_email, customer_address,
customer_phone, date_of_birth, date_joined)

VALUES ('Chaitanya', 'chaitanya@gmail.com', '567 Main St', '555-5678', '1999-09-25',
CURDATE());

INSERT INTO Customers (customer_name, customer_email, customer_address,
customer_phone, date_of_birth, date_joined)

VALUES ('Mohan', 'mohan@gmail.com', '678 Main St', '555-6789', '2002-06-15', CURDATE());

INSERT INTO Customers (customer_name, customer_email, customer_address,
customer_phone, date_of_birth, date_joined)

VALUES ('Moulika', 'moulika@gmail.com', '789 Main St', '555-7890', '2000-03-23',
CURDATE());

INSERT INTO Customers (customer_name, customer_email, customer_address,
customer_phone, date_of_birth, date_joined)

VALUES ('Shanthi', 'shanthi@gmail.com', '890 Main St', '555-8901', '2000-05-25',
CURDATE());

INSERT INTO Customers (customer_name, customer_email, customer_address,
customer_phone, date_of_birth, date_joined)

VALUES ('MohanY', 'mohany@gmail.com', '901 Main St', '555-9012', '2002-10-10',
CURDATE());

INSERT INTO Customers (customer_name, customer_email, customer_address,
customer_phone, date_of_birth, date_joined)

VALUES ('Vikas', 'vikas@gmail.com', '012 Main St', '555-0123', '2000-08-17', CURDATE());
```

CREATE POLICIES TABLE:

```
CREATE TABLE Policies (

    policy_id INT PRIMARY KEY,

    policy_type VARCHAR(50),

    max_claims_per_year INT,

    base_premium DECIMAL(10, 2)

);
```

INSERT SOME POLICIES:

```
INSERT INTO Policies (policy_id, policy_type, max_claims_per_year, base_premium)
VALUES (1, 'Life', 2, 1200.00),
    (2, 'Health', 4, 1000.00),
    (3, 'Asset', 3, 1500.00),
    (4, 'Vehicle', 2, 800.00);
```

CREATE CUSTOMER POLICIES TABLE:

-- This table links customers with their policies,

-- As a customer can have multiple policies. It also keeps track of the renewal dates and premium amounts.

```
CREATE TABLE Customer_Policies (
  customer_policy_id INT PRIMARY KEY AUTO_INCREMENT,
  customer_id INT,
  policy_id INT,
  policy_start_date DATE,
  policy_end_date DATE,
  premium_amount DECIMAL(10, 2),
  claims_made_this_year INT DEFAULT 0,
  agent_id INT,
      FOREIGN KEY (agent_id) REFERENCES Agents(agent_id),
  FOREIGN KEY (customer_id) REFERENCES Customers(customer_id),
  FOREIGN KEY (policy_id) REFERENCES Policies(policy_id)
);
```

ASSIGNING POLICIES TO CUSTOMERS:

```
INSERT INTO Customer_Policies (customer_id, policy_id, policy_start_date, policy_end_date,
premium_amount, claims_made_this_year)

VALUES

-- Chitra gets a Life policy

((SELECT customer_id FROM Customers WHERE customer_name = 'Chitra'), (SELECT
policy_id FROM Policies WHERE policy_type = 'Life'), CURDATE(), DATE_ADD(CURDATE(),
INTERVAL 10 day), (SELECT base_premium FROM Policies WHERE policy_type = 'Life'), 0),

((SELECT customer_id FROM Customers WHERE customer_name = 'Chitra'), (SELECT
policy_id FROM Policies WHERE policy_type = 'Health'), CURDATE(), DATE_ADD(CURDATE(),
INTERVAL 1 YEAR), (SELECT base_premium FROM Policies WHERE policy_type = 'Asset'), 1),

-- Saisri gets a Health policy

((SELECT customer_id FROM Customers WHERE customer_name = 'Saisri'), (SELECT policy_id
FROM Policies WHERE policy_type = 'Health'), CURDATE(), DATE_ADD(CURDATE(), INTERVAL
1 YEAR), (SELECT base_premium FROM Policies WHERE policy_type = 'Health'), 0),

-- Mounika gets an Asset policy

((SELECT customer_id FROM Customers WHERE customer_name = 'Mounika'), (SELECT
policy_id FROM Policies WHERE policy_type = 'Asset'), CURDATE(), DATE_ADD(CURDATE(),
INTERVAL 3 YEAR), (SELECT base_premium FROM Policies WHERE policy_type = 'Asset'), 3),

-- Praveen gets a Vehicle policy

((SELECT customer_id FROM Customers WHERE customer_name = 'Praveen'), (SELECT
policy_id FROM Policies WHERE policy_type = 'Vehicle'), CURDATE(), DATE_ADD(CURDATE(),
INTERVAL 1 YEAR), (SELECT base_premium FROM Policies WHERE policy_type = 'Vehicle'), 0),

-- Chaitanya gets a Health policy

((SELECT customer_id FROM Customers WHERE customer_name = 'Chaitanya'), (SELECT
policy_id FROM Policies WHERE policy_type = 'Health'), CURDATE(), DATE_ADD(CURDATE(),
INTERVAL 2 YEAR), (SELECT base_premium FROM Policies WHERE policy_type = 'Health'), 1),

-- Mohan gets a Life policy

((SELECT customer_id FROM Customers WHERE customer_name = 'Mohan'), (SELECT
policy_id FROM Policies WHERE policy_type = 'Life'), CURDATE(), DATE_ADD(CURDATE(),
INTERVAL 3 YEAR), (SELECT base_premium FROM Policies WHERE policy_type = 'Life'), 0),
```

CREATE AN AGENTS TABLE:

```
CREATE TABLE Agents (
    agent_id INT PRIMARY KEY AUTO_INCREMENT,
    agent_name VARCHAR(100),
    agent_email VARCHAR(100),
    agent_phone VARCHAR(15)
);
```

INSERT SOME AGENTS:

```
INSERT INTO Agents (agent_name, agent_email, agent_phone)
VALUES ('Agent_A', 'agenta@agent.com', '123-9876');
INSERT INTO Agents (agent_name, agent_email, agent_phone)
VALUES ('Agent_B', 'agentb@agent.com', '234-9876');
```

CREATE EMPLOYEES TABLE TABLE:

```
CREATE TABLE Employees (
    employee_id INT PRIMARY KEY AUTO_INCREMENT,
    employee_name VARCHAR(100),
    employee_role ENUM('Employee', 'Manager'),
    employee_email VARCHAR(100)
);
```

INSERT SOME EMPLOYEES:

```
INSERT INTO Employees (employee_name, employee_role, employee_email)
VALUES ('Alice', 'Employee', 'alice@insurance.com');
INSERT INTO Employees (employee_name, employee_role, employee_email)
VALUES ('Bob', 'Manager', 'bob@insurance.com');
```

CREATE CLAIMS TABLE TABLE:

```
-- Customers request claims and can only be approved by a Manager
CREATE TABLE Claims (
    claim_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_policy_id INT,
    claim_date DATE,
    claim_amount DECIMAL(10, 2),
    claim_status ENUM('Pending', 'Approved', 'Rejected') DEFAULT 'Pending',
    processed_by_employee INT,
    approved_by_manager INT,
    FOREIGN KEY (customer_policy_id) REFERENCES Customer_Policies(customer_policy_id),
    FOREIGN KEY (processed_by_employee) REFERENCES Employees(employee_id),
    FOREIGN KEY (approved_by_manager) REFERENCES Employees(employee_id),
    FOREIGN KEY (customer_policy_id) REFERENCES Customer_Policies(customer_policy_id)
);
```

CREATE PAYMENTS TABLE:

```
-- This table stores the details of premium payments made by the customer or agent.
CREATE TABLE Payments (
    payment_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_policy_id INT,
    payment_date DATE,
    payment_amount DECIMAL(10, 2),
    payment_period ENUM('Monthly', 'Quarterly', 'Yearly'),
    agent_name VARCHAR(255),
    updated_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    updated_user VARCHAR(255),
    FOREIGN KEY (customer_policy_id) REFERENCES Customer_Policies(customer_policy_id)
);
```

INSERTING PAYMENTS:

```
INSERT INTO Payments (customer_policy_id, payment_date, payment_amount,
payment_period)

VALUES

((SELECT customer_policy_id FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Chitra') AND policy_id = (SELECT
policy_id FROM Policies WHERE policy_type = 'Life')), CURDATE(), 1200.00, 'Yearly',
'Agent_A'),

((SELECT customer_policy_id FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Chitra') AND policy_id = (SELECT
policy_id FROM Policies WHERE policy_type = 'Health')), CURDATE(), 1500.00, 'Yearly',
'Agent_B');

INSERT INTO Payments (customer_policy_id, payment_date, payment_amount,
payment_period, agent_name)

VALUES

((SELECT customer_policy_id FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Saisri') AND policy_id = (SELECT
policy_id FROM Policies WHERE policy_type = 'Health')), CURDATE(), 800.00, 'Monthly',
'Agent_C');

INSERT INTO Payments (customer_policy_id, payment_date, payment_amount,
payment_period, agent_name)

VALUES

((SELECT customer_policy_id FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Mounika') AND policy_id =
(SELECT policy_id FROM Policies WHERE policy_type = 'Asset')), CURDATE(), 3000.00,
'Yearly', 'Agent_D');

INSERT INTO Payments (customer_policy_id, payment_date, payment_amount,
payment_period, agent_name)

VALUES

((SELECT customer_policy_id FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Praveen') AND policy_id = (SELECT
policy_id FROM Policies WHERE policy_type = 'Vehicle')), CURDATE(), 1200.00, 'Yearly',
'Agent_A' );

INSERT INTO Payments (customer_policy_id, payment_date, payment_amount,
payment_period, agent_name)

VALUES
```

```
((SELECT customer_policy_id FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Chaitanya') AND policy_id =
(SELECT policy_id FROM Policies WHERE policy_type = 'Health')), CURDATE(), 1000.00,
'Quarterly', 'Agent_B');

INSERT INTO Payments (customer_policy_id, payment_date, payment_amount,
payment_period, agent_name)

VALUES

((SELECT customer_policy_id FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Mohan') AND policy_id = (SELECT
policy_id FROM Policies WHERE policy_type = 'Life')), CURDATE(), 2000.00,
'Yearly','Agent_C');

INSERT INTO Payments (customer_policy_id, payment_date, payment_amount,
payment_period, agent_name)

VALUES

((SELECT customer_policy_id FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Moulika') AND policy_id = (SELECT
policy_id FROM Policies WHERE policy_type = 'Asset')), CURDATE(), 2500.00, 'Yearly',
'Agent_D');

INSERT INTO Payments (customer_policy_id, payment_date, payment_amount,
payment_period, agent_name)

VALUES

((SELECT customer_policy_id FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Shanthi') AND policy_id = (SELECT
policy_id FROM Policies WHERE policy_type = 'Vehicle')), CURDATE(), 1300.00, 'Quarterly',
'Agent_A');

INSERT INTO Payments (customer_policy_id, payment_date, payment_amount,
payment_period, agent_name)

VALUES

((SELECT customer_policy_id FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'MohanY') AND policy_id =
(SELECT policy_id FROM Policies WHERE policy_type = 'Life')), CURDATE(), 1100.00, 'Yearly',
'Agent_B');

((SELECT customer_policy_id FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Vikas') AND policy_id = (SELECT
policy_id FROM Policies WHERE policy_type = 'Health')), CURDATE(), 1500.00, 'Yearly',
'Agent_C');
```

HANDLING PREMIUM PAYMENT SCHEDULE AND PENALTIES

```
-- Calculate the penalty for late payment (5% penalty if payment is overdue)

-- Let’s create a query to calculate premium payment schedules and

-- Check if there’s a penalty due for late payments


SELECT

   cp.customer_policy_id,

   c.customer_name,

   p.policy_type,

   cp.premium_amount,

   CASE

      WHEN DATEDIFF(CURDATE(), MAX(py.payment_date)) > 30 THEN cp.premium_amount
* 0.05

      ELSE 0

   END AS penalty

FROM

   Customer_Policies cp

JOIN

   Customers c ON cp.customer_id = c.customer_id

JOIN

   Policies p ON cp.policy_id = p.policy_id

LEFT JOIN

   Payments py ON cp.customer_policy_id = py.customer_policy_id

GROUP BY

   cp.customer_policy_id;
```

CLAIM REQUEST AND PROCESSING

```
-- Inserting claims for customers

-- A customer submits a claim, and it needs to be processed by an employee
and then approved by a manager

-- Find all customer policies


SELECT customer_policy_id FROM Customer_Policies

WHERE customer_id = (SELECT customer_id FROM Customers WHERE customer_name =
'Chitra');


-- Find all employees named Alice

SELECT employee_id FROM Employees WHERE employee_name = 'Alice';


SELECT DISTINCT customer_policy_id

FROM Customer_Policies

WHERE customer_id = (SELECT customer_id FROM Customers WHERE customer_name =
'Chitra' LIMIT 1);


SELECT MIN(customer_policy_id)

FROM Customer_Policies

WHERE customer_id = (SELECT customer_id FROM Customers WHERE customer_name =
'Chitra');
```

```
INSERT INTO Claims (customer_policy_id, claim_date, claim_amount, claim_status,
processed_by_employee, approved_by_manager)

VALUES

-- Chitra's claim is approved

((SELECT MIN(customer_policy_id) FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Chitra')), CURDATE(), 500.00,
'Approved', (SELECT MIN(employee_id) FROM Employees WHERE employee_name = 'Alice'),
(SELECT MIN(employee_id) FROM Employees WHERE employee_name = 'Bob')),


-- Saisri's claim is pending

((SELECT MIN(customer_policy_id) FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Saisri')), CURDATE(), 300.00,
'Pending', (SELECT MIN(employee_id) FROM Employees WHERE employee_name = 'Alice'),
NULL),


-- Mounika's claim is rejected

((SELECT MIN(customer_policy_id) FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Mounika')), CURDATE(), 700.00,
'Rejected', (SELECT MIN(employee_id) FROM Employees WHERE employee_name = 'Alice'),
(SELECT MIN(employee_id) FROM Employees WHERE employee_name = 'Bob')),


-- Praveen's claim is approved

((SELECT MIN(customer_policy_id) FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Praveen')), CURDATE(), 250.00,
'Approved', (SELECT MIN(employee_id) FROM Employees WHERE employee_name = 'Alice'),
(SELECT MIN(employee_id) FROM Employees WHERE employee_name = 'Bob')),


-- Chaitanya's claim is pending

((SELECT MIN(customer_policy_id) FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Chaitanya')), CURDATE(), 450.00,
'Pending', (SELECT MIN(employee_id) FROM Employees WHERE employee_name = 'Alice'),
NULL),
-- Mohan's claim is approved

((SELECT MIN(customer_policy_id) FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Mohan')), CURDATE(), 600.00,
'Approved', (SELECT MIN(employee_id) FROM Employees WHERE employee_name = 'Alice'),
(SELECT MIN(employee_id) FROM Employees WHERE employee_name = 'Bob')),


-- Moulika's claim is rejected

((SELECT MIN(customer_policy_id) FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Moulika')), CURDATE(), 800.00,
'Rejected', (SELECT MIN(employee_id) FROM Employees WHERE employee_name = 'Alice'),
(SELECT MIN(employee_id) FROM Employees WHERE employee_name = 'Bob')),


-- Shanthi's claim is pending

((SELECT MIN(customer_policy_id) FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Shanthi')), CURDATE(), 350.00,
'Pending', (SELECT MIN(employee_id) FROM Employees WHERE employee_name = 'Alice'),
NULL),


-- MohanY's claim is approved

((SELECT MIN(customer_policy_id) FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'MohanY')), CURDATE(), 900.00,
'Approved', (SELECT MIN(employee_id) FROM Employees WHERE employee_name = 'Alice'),
(SELECT MIN(employee_id) FROM Employees WHERE employee_name = 'Bob')),


-- Vikas' claim is pending

((SELECT MIN(customer_policy_id) FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Vikas')), CURDATE(), 400.00,
'Pending', (SELECT MIN(employee_id) FROM Employees WHERE employee_name = 'Alice'),
NULL);
```

CREATE TABLE FOR PAYMENT SCHEDULES AND PENALTIES

```
-- Payment Schedules table to track payment dues

CREATE TABLE IF NOT EXISTS Payment_Schedules (

    schedule_id INT PRIMARY KEY AUTO_INCREMENT,

    customer_policy_id INT,

    payment_due_date DATE,

    payment_amount DECIMAL(10, 2),

    penalty_amount DECIMAL(10, 2) DEFAULT 0.00,  -- penalty applied for late payment

    payment_status VARCHAR(50) DEFAULT 'Pending', -- default payment status

    actual_payment_date DATE, -- date when payment is made

    FOREIGN KEY (customer_policy_id) REFERENCES Customer_Policies(customer_policy_id)

);
```

INSERT PAYMENT SCHEDULE

```
-- You can create a schedule for a customer's policy with due dates.

-- Insert payment schedule for a customer

INSERT INTO Payment_Schedules (customer_policy_id, payment_due_date,
payment_amount)

VALUES

((SELECT customer_policy_id FROM Customer_Policies WHERE customer_id = (SELECT
customer_id FROM Customers WHERE customer_name = 'Chitra')

    AND policy_id = (SELECT policy_id FROM Policies WHERE policy_type = 'Life')),

 '2024-10-01',  -- Example payment due date

 500.00);      -- Example payment amount
```

-- Step 3: Apply Penalty for Late Payment

```
-- To apply a penalty when the payment is late,

-- you can write a trigger or periodically check if the due date has passed and the payment is
still pending.

-- Update penalty for late payment
```

```
SET SQL_SAFE_UPDATES = 0;

UPDATE Payment_Schedules

SET penalty_amount = 50.00,   -- Example penalty for late payment

    payment_status = 'Late'    -- Mark as late

WHERE payment_due_date < CURDATE() AND payment_status = 'Pending';
```

-- Step 4: Payment Submission

-- When the customer makes a payment (whether on time or late),

-- update the actual payment date and calculate the total amount paid, including any penalties.

-- Update payment record after customer pays

```
UPDATE Payment_Schedules

SET actual_payment_date = CURDATE(),

    payment_status = 'Paid',

    penalty_amount = CASE

                   WHEN payment_due_date < CURDATE() THEN 50.00  -- Apply penalty if late

                   ELSE 0.00  -- No penalty for on-time payment

                 END

WHERE schedule_id = 1;  -- Example schedule ID
```

-- Step 5: Notify Customer of Payment Schedule

-- If you want to "publish" or send the payment schedule to the customer, you can fetch and display the schedule with a query:

```
SELECT ps.payment_due_date, ps.payment_amount, ps.penalty_amount,
ps.payment_status

FROM Payment_Schedules ps

JOIN Customer_Policies cp ON ps.customer_policy_id = cp.customer_policy_id

JOIN Customers c ON cp.customer_id = c.customer_id

WHERE c.customer_name = 'Mounika';
```

FOR CUSTOMERS TO ACCESS POLICY DETAILS ONLINE THROUGH A WEBSITE

-- Backend PHP is required for further steps

```
SELECT

   c.customer_name,

   p.policy_type,

   cp.policy_start_date,

   cp.policy_end_date,

   cp.premium_amount,

   cp.claims_made_this_year,

   py.payment_amount,

   py.payment_date,

   py.payment_period

FROM

   Customers c

JOIN

   Customer_Policies cp ON c.customer_id = cp.customer_id

JOIN

   Policies p ON cp.policy_id = p.policy_id

LEFT JOIN

   Payments py ON cp.customer_policy_id = py.customer_policy_id

WHERE

   c.customer_id = (SELECT customer_id FROM Customers WHERE customer_name =
'SAISRI') -- Replace with actual customer ID or name

GROUP BY

   c.customer_name,

   p.policy_type,

   cp.policy_start_date,
   cp.policy_end_date,
   cp.premium_amount,
   cp.claims_made_this_year,
   py.payment_amount,
   py.payment_date,
   py.payment_period;
```

GENERATING Monthly/Quarterly/Yearly STATEMENTS

```
SELECT
   c.customer_name,
   cp.premium_amount,
   p.policy_type,
   py.payment_period,
   SUM(py.payment_amount) AS total_paid
FROM
   Payments py
JOIN
   Customer_Policies cp ON py.customer_policy_id = cp.customer_policy_id
JOIN
   Customers c ON cp.customer_id = c.customer_id
JOIN
   Policies p ON cp.policy_id = p.policy_id
WHERE
   py.payment_period = 'Yearly' -- or 'Monthly' 'Quarterly', 'Yearly'
GROUP BY
c.customer_name, cp.premium_amount, p.policy_type, py.payment_period;
```

DISCOUNT FOR NO CLAIMS ON POLICY RENEWAL

```
-- When renewing a policy, if no claims were made during the previous year
-- The customer will get a discount on their premium.
-- Calculate the premium for the next year, including discount if no claims were made

SELECT
    cp.customer_policy_id,
    c.customer_name,
    p.policy_type,
    cp.premium_amount,
    CASE
        WHEN cp.claims_made_this_year = 0 THEN cp.premium_amount * 0.9 -- 10% discount
        ELSE cp.premium_amount
    END AS next_year_premium
FROM
    Customer_Policies cp
JOIN
    Customers c ON cp.customer_id = c.customer_id
JOIN
    Policies p ON cp.policy_id = p.policy_id;
```

REGULATION OF CLAIMS PER YEAR

-- Each type of policy has a limitation on how many claims can be made in a year

-- Check if a customer can make a new claim based on their policy

```
SELECT
    cp.customer_policy_id,
    p.policy_type,
    p.max_claims_per_year,
    cp.claims_made_this_year,
    CASE
        WHEN cp.claims_made_this_year < p.max_claims_per_year THEN 'Allowed'
        ELSE 'Not Allowed'
    END AS claim_status
FROM
    Customer_Policies cp
JOIN
    Policies p ON cp.policy_id = p.policy_id;
```

PROCEDURES AND FUNCTIONS

1. LIST CUSTOMER DETAILS FOR A GIVEN CUSTOMER

This procedure will fetch all the customer details based on their customer_id.

```
DELIMITER $$
CREATE PROCEDURE GetCustomerDetail(IN customerId INT)
BEGIN
    SELECT customer_id, customer_name, customer_email, customer_address,
customer_phone, date_of_birth, date_joined
    FROM Customers
    WHERE customer_id = customerId;
END$$
DELIMITER ;
```

-- How to Execute the Procedure:

```
CALL GetCustomerDetail(3);  -- Replace '1' with the desired Customer ID
```

2. CALCULATE POLICY AMOUNT AT THE TIME OF RENEWAL

-- This function will calculate the premium for policy renewal.

-- The function will check whether the customer has made any claims during the year.

-- If no claims were made, a discount is applied; otherwise, the premium remains the same.

```
DELIMITER $$
CREATE FUNCTION RenewalAmountCalculation(customerId INT, policyId INT)
RETURNS DECIMAL(10,2)
BEGIN
-- Four variables are declared to store intermediate values
```

```
    DECLARE claimsMade INT;
    DECLARE basePremium DECIMAL(10, 2);
    DECLARE discount DECIMAL(10, 2);
    DECLARE finalAmount DECIMAL(10, 2);
    -- Get the number of claims made by the customer for this policy in the current year
    SELECT claims_made_this_year INTO claimsMade
    FROM Customer_Policies
    WHERE customer_id = customerId AND policy_id = policyId;
    -- Get the base premium for the policy
    SELECT base_premium INTO basePremium
    FROM Policies
    WHERE policy_id = policyId;
    -- Check if claims were made
    IF claimsMade = 0 THEN
        SET discount = basePremium * 0.10; -- 10% discount if no claims were made
    ELSE
        SET discount = 0; -- No discount if claims were made
    END IF;
    -- Calculate final amount
    SET finalAmount = basePremium - discount;
    RETURN finalAmount;
END$$
DELIMITER ;
SET GLOBAL log_bin_trust_function_creators = 1;
-- to check whether the function is added in the database or not
```

-- How to Execute the Function:

SELECT RenewalAmountCalculation(1, 2);

-- Replace '1' with the Customer ID and '2' with the Policy ID

3. LIST ALL POLICY DETAILS OWNED BY A CUSTOMER

-- This procedure will return all the policies owned by a customer,

-- along with their details, such as policy type, start date, end date, premium amount, and claims made this year.

-- Stored Procedure to List All Policies for a Customer

```
DELIMITER $$


CREATE PROCEDURE GetCustomerPolicies(IN customerId INT)
BEGIN
   SELECT cp.customer_policy_id, p.policy_type, cp.policy_start_date, cp.policy_end_date,
cp.premium_amount, cp.claims_made_this_year
   FROM Customer_Policies cp
   JOIN Policies p ON cp.policy_id = p.policy_id
   WHERE cp.customer_id = customerId;
END$$


DELIMITER ;
```

-- How to Execute the Procedure:

```
CALL GetCustomerPolicies(1);  -- Replace '1' with the desired Customer ID
```

TRIGGERS

1. Trigger for Recording Deletions in the Customers Table

```
-- When a record is deleted from the Customers table,

-- The deleted data will be inserted into a Customer_History table for future reference.


-- Step 1: Create the Customer_History Table


CREATE TABLE CustomersHistory (

    history_id INT PRIMARY KEY AUTO_INCREMENT,

    customer_id INT,

    customer_name VARCHAR(100),

    customer_email VARCHAR(100),

    customer_address VARCHAR(255),

    customer_phone VARCHAR(15),

    date_of_birth DATE,

    date_joined DATE,

    deletion_date DATETIME,

    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)

);


-- Step 2: Create the Trigger for Deletion

DELIMITER $$

CREATE TRIGGER AfterCustomerDelete

AFTER DELETE ON Customers -- trigger will execute after a row is deleted from the
Customers table.

FOR EACH ROW -- trigger will execute once for each row that is deleted

BEGIN
```

```
    INSERT INTO CustomerHistory (customer_id, customer_name, customer_email,
customer_address, customer_phone, date_of_birth, date_joined, deletion_date)

    VALUES (OLD.customer_id, OLD.customer_name, OLD.customer_email,
OLD.customer_address, OLD.customer_phone, OLD.date_of_birth, OLD.date_joined,
NOW());

END$$


DELIMITER ;


-- OLD refers to the values of the row that was just deleted from the Customers table

-- NOW(): Captures the current timestamp of when the deletion occurred.


-- The AfterCustomerDelete trigger automatically inserts a record into the CustomersHistory
table

-- whenever a customer is deleted from the Customers table.

-- This helps maintain a historical customer data log, even after they are removed from the
primary customers list.
```

2. Trigger for Handling Insertions and Updates in the Premium_Payment Table

```
-- This trigger will update the updated_date and updated_user fields

-- whenever a new record is inserted or updated in the Premium_Payment table.


-- Step 1: Assume the Premium_Payment Table Structure


CREATE TABLE PaymentsPremium (

    payment_premium_id INT PRIMARY KEY AUTO_INCREMENT,

    payment_id INT,

    customer_id INT,

    policy_id INT,

    payment_amount DECIMAL(10, 2),
    payment_date DATE,

    updated_date DATETIME,

    updated_user VARCHAR(50),

    FOREIGN KEY (payment_id) REFERENCES Payments(payment_id),

    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id),

    FOREIGN KEY (policy_id) REFERENCES Policies(policy_id)

);

-- Step 2: Trigger for Insertions


DELIMITER $$


CREATE TRIGGER BeforePaymentsPremiumInsert

BEFORE INSERT ON PaymentsPremium -- -- trigger will execute before a row is inserted into
the PaymentsPremium table.

FOR EACH ROW

BEGIN

    SET NEW.updated_date = NOW(); -- Sets the updated_date field to the current date and
time at the moment of insertion.

    SET NEW.updated_user = USER();  --  Sets the updated_user field to the current username,
assuming USER() retrieves the logged-in user's name.

END$$


DELIMITER ;


-- Step 3 : Trigger for updates


DELIMITER $$


CREATE TRIGGER BeforePaymentsPremiumUpdate

BEFORE UPDATE ON PaymentsPremium -- trigger will execute before a row is updated into
the PaymentsPremium table.
```

```
FOR EACH ROW
BEGIN
   SET NEW.updated_date = NOW();
   SET NEW.updated_user = USER();
END$$

DELIMITER ;
```

-- BEFORE INSERT OR UPDATE: This trigger fires before an insert or update operation on the Premium_Payment table.

-- USER(): Retrieves the logged-in user.

3. Trigger to Regulate the Number of Claims for Each Policy

-- This trigger will ensure that the number of claims for each policy type

-- does not exceed the allowed limit when a new claim request is created.

-- Step 1: Assume the Policies Table Stores the Max Claims

-- Step 2: Create the Trigger to Regulate Claims

```
DELIMITER $$

CREATE TRIGGER BeforeClaimInsert
BEFORE INSERT ON Claims --  trigger will execute before a new row is inserted into the Claims table.
FOR EACH ROW -- trigger will execute for each row that is being inserted.
BEGIN
   DECLARE claimCount INT;
   DECLARE maxAllowedClaims INT;

   -- Get the current claim count for this customer policy
   SELECT claims_made_this_year INTO claimCount
```

```
    FROM Customer_Policies
    WHERE customer_policy_id = NEW.customer_policy_id; -- NEW.customer_policy_id is the
customer policy ID associated with the claim being newly inserted.

    -- Get the maximum allowed claims for the policy type
    SELECT p.max_claims INTO maxAllowedClaims
    FROM Policies p
    JOIN Customer_Policies cp ON p.policy_id = cp.policy_id
    WHERE cp.customer_policy_id = NEW.customer_policy_id;

    -- Check if the number of claims has exceeded the maximum allowed
    IF claimCount >= maxAllowedClaims THEN
        SIGNAL SQLSTATE '45000' -- This raises a generic error with the specified SQL state code
if the IF condition is not met.
        SET MESSAGE_TEXT = 'Claim limit exceeded for this policy type'; -- This sets the error
message that will be returned to the user.
    END IF;
END$$

DELIMITER ;

-- NEW: Refers to the data being inserted into the Claims table.
-- SIGNAL SQLSTATE '45000': Throws a custom error if the maximum allowed claims for a
policy are exceeded.
```