INSURANCE DATABASE MANAGEMENT

Problem statement:

In the insurance database,

Relation between client and his policies is a one to many relationship, but policy type to clients is a many to many relationship.

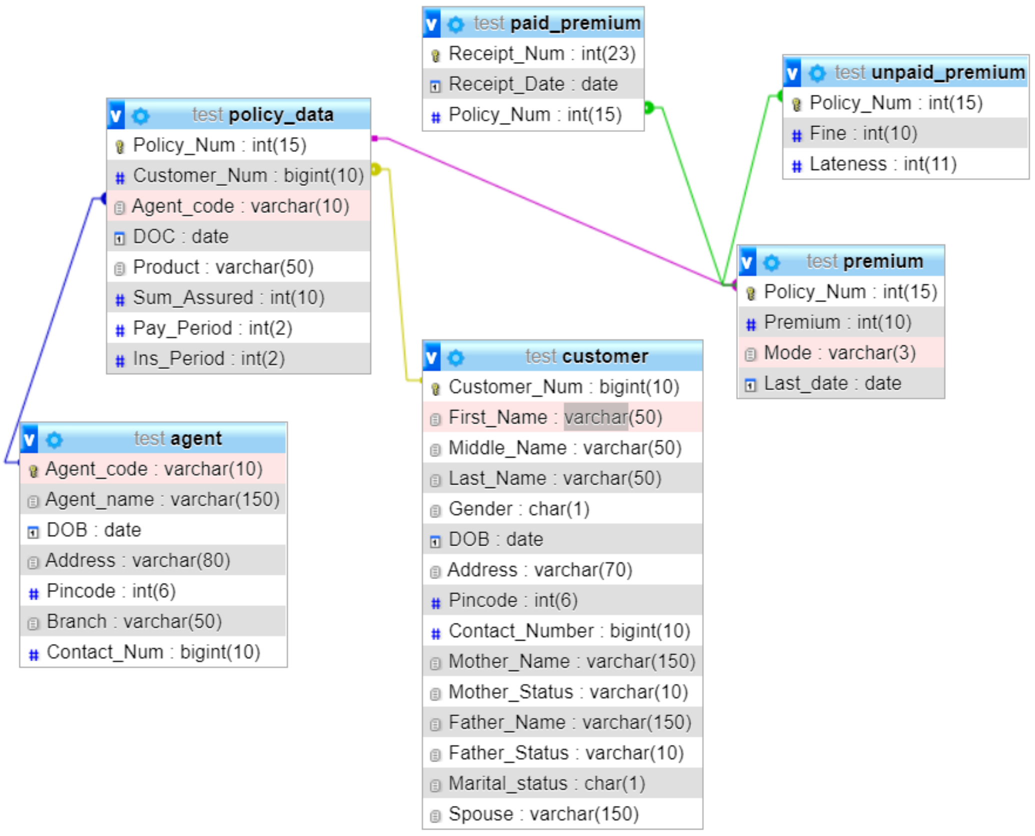
• Data would be handled using different tables and relations using MySQL.

• A policy taken by a client has attributes like premium, sum assured, date of commencement, etc.

• A client has attributes including personal details as well as details about the policy he/she has taken.

• Create many other tables where records of policies taken by different clients would be present depending on its status like active, lapsed, etc.
The developed system should allow admin users to register insured persons with their name, date of birth, residence address and also policy details.
Every agent and the customer’s phone number must be unique among the rest in their category.
Once the intial entries are done for the table containing agent details, create triggers to block new entries and also block the option for the already existing agents to update their details.

Database Schema: (show all the tables and the constraints)



test paid_premium
Receipt_Num : int(23)
Receipt_Date : date
Policy_Num : int(15)

test unpaid_premium
Policy_Num : int(15)
Fine : int(10)
Lateness : int(11)

test policy_data
Policy_Num : int(15)
Customer_Num : bigint(10)
Agent_code : varchar(10)
DOC : date
Product : varchar(50)
Sum_Assured : int(10)
Pay_Period : int(2)
Ins_Period : int(2)

test premium
Policy_Num : int(15)
Premium : int(10)
Mode : varchar(3)
Last_date : date

test customer
Customer_Num : bigint(10)
First_Name : varchar(50)
Middle_Name : varchar(50)
Last_Name : varchar(50)
Gender : char(1)
DOB : date
Address : varchar(70)
Pincode : int(6)
Contact_Number : bigint(10)
Mother_Name : varchar(150)
Mother_Status : varchar(10)
Father_Name : varchar(150)
Father_Status : varchar(10)
Marital_status : char(1)
Spouse : varchar(150)

test agent
Agent_code : varchar(10)
Agent_name : varchar(150)
DOB : date
Address : varchar(80)
Pincode : int(6)
Branch : varchar(50)
Contact_Num : bigint(10)

Functional Dependencies: (List based on your application constraints)

1. Agent_code , Contact_Num-> Agent_name, DOB, Address, Pincode, Branch
2. Customer_Num, Contact_Number -> First_Name, Middle_Name, Last_Name, Gender, DOB, Address, Pincode, Mother_Name, Mother_Status, Father_Status, Marital_Status, Spouse
3. Policy_Num ->Customer_Num, Agent_code, DOC, Product, Sum_Assured, Pay_period, Ins_Period
4. Policy_Num -> Premium, Mode, Last_date
5. Policy_Num -> Fine, Lateness
6. Receipt_Num -> Receipt_Date, Policy_Num

Candidate keys: (Justify how did you get these as keys)

Relation	Simple Candidate keys	Composite Candidate keys
agent	Agent_code Contact_Num	(Agent_code,Contact_Num)
customer	Customer_Num Contact_Number	(Customer_Num,Contact_Number)
paid_premium	Receipt_Num	(Receipt_Date,Policy_Num) (Receipt_Num, Policy_Num)
policy_data	Policy_Num	(Policy_Num, Customer_Num)
premium	Policy_Num	--
unpaid_premium	Policy_Num	--

Normalization and testing for lossless join property:

The whole database was designed using ER diagrams. So, the model is already in normalized form.

- There is only one column in every primary key attribute. So, there are no partial dependencies possible and 2nd NF is achieved.

- There are no transitive dependencies present. Hence 3rd NF is achieved.

Since there is a very specific id for each row in tables that have been created, and primary keys in each table are all simple attributes, violating the rules of 2nd NF is very hard, since there cannot be a case of partial dependency in this setting.
But there is case where we can violate the 3rd NF.

> If we insert new columns district and state in customer and agent details table, we would obtain a transitive dependency between those columns and the attribute 'pin'.
> In that case, we would have to take those three columns to make a new table and keep only the attribute 'pin' in the tables 'agent' and 'customer'.

LOSSLESS JOIN:

The conditions are

1. Union of Attributes of R1 and R2 must be equal to attribute of R. Each attribute of R must be either in R1 or in R2.
   ```
   Att(R1) U Att(R2) = Att(R)
   ```
2. Intersection of Attributes of R1 and R2 must not be NULL.
   ```
   Att(R1) ∩ Att(R2) ≠ Φ
   ```
3. Common attribute must be a key for at least one relation (R1 or R2)
   ```
   Att(R1) ∩ Att(R2) -> Att(R1) or Att(R1) ∩ Att(R2) -> Att(R2)
   ```

Since this schema is already in appropriate normal form, there is no question of decomposing any relation into smaller relations.

DDL:

```sql
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";



CREATE TABLE `agent` (
  `Agent_code` varchar(10) NOT NULL,
  `Agent_name` varchar(150) NOT NULL,
  `DOB` date NOT NULL,
  `Address` varchar(80) NOT NULL,
  `Pincode` int(6) NOT NULL,
  `Branch` varchar(50) NOT NULL,
  `Contact_Num` bigint(10) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `agent`
```

```sql
--

INSERT INTO `agent` (`Agent_code`, `Agent_name`, `DOB`, `Address`, `Pincode`, `Branch`,
`Contact_Num`) VALUES
('234abc231', 'Sanjay', '1966-02-21', '21/694, Satyam Apartment, Refinery Road, Gorwa',
390016, 'Vadodara', 7016636685);

-- --------------------------------------------------------

--
-- Table structure for table `customer`
--

CREATE TABLE `customer` (
  `Customer_Num` bigint(10) NOT NULL,
  `First_Name` varchar(50) NOT NULL,
  `Middle_Name` varchar(50) NOT NULL,
  `Last_Name` varchar(50) NOT NULL,
  `Gender` char(1) NOT NULL,
  `DOB` date NOT NULL,
  `Address` varchar(70) NOT NULL,
  `Pincode` int(6) NOT NULL,
  `Contact_Number` bigint(10) NOT NULL,
  `Mother_Name` varchar(150) NOT NULL,
  `Mother_Status` varchar(10) NOT NULL,
  `Father_Name` varchar(150) NOT NULL,
  `Father_Status` varchar(10) NOT NULL,
  `Marital_status` char(1) NOT NULL,
  `Spouse` varchar(150) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `customer`
--

INSERT INTO `customer` (`Customer_Num`, `First_Name`, `Middle_Name`, `Last_Name`, `Gender`,
`DOB`, `Address`, `Pincode`, `Contact_Number`, `Mother_Name`, `Mother_Status`,
`Father_Name`, `Father_Status`, `Marital_status`, `Spouse`) VALUES
(10002, 'Devam', 'Sanjay', 'Sheth', 'M', '2018-10-02', '21/694, Satyam Apartment, Refinery
Road, Gorwa', 390016, 7016636683, 'Harsha Sheth', 'A', 'Sanjay Sheth', 'A', 'S', '');

-- --------------------------------------------------------

--
-- Table structure for table `paid_premium`,
Also has Check constraint in Create statement
--
```

```sql
CREATE TABLE `paid_premium` (
  `Receipt_Num` int(23) NOT NULL CHECK(Receipt_Num>=1),
  `Receipt_Date` date NOT NULL,
  `Policy_Num` int(15) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `paid_premium`
--

INSERT INTO `paid_premium` (`Receipt_Num`, `Receipt_Date`, `Policy_Num`) VALUES
(325256815, '2018-10-31', 123564789),
(325284137, '2018-11-01', 284049583),
(325289940, '2018-11-01', 123564789);

-- --------------------------------------------------------

--
-- Table structure for table `policy_data`,
Also has Check constraint in Create statement
--

CREATE TABLE `policy_data` (
  `Policy_Num` int(15) NOT NULL,
  `Customer_Num` bigint(10) NOT NULL CHECK(Customer_Num>=1),
  `Agent_code` varchar(10) NOT NULL,
  `DOC` date NOT NULL,
  `Product` varchar(50) NOT NULL,
  `Sum_Assured` int(10) NOT NULL,
  `Pay_Period` int(2) NOT NULL,
  `Ins_Period` int(2) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `policy_data`
--

INSERT INTO `policy_data` (`Policy_Num`, `Customer_Num`, `Agent_code`, `DOC`, `Product`,
`Sum_Assured`, `Pay_Period`, `Ins_Period`) VALUES
(123564789, 10002, '234abc231', '2018-10-02', 'Jeevan Labh', 35000, 5, 10),
(284049583, 10002, '234abc231', '2007-06-20', 'Jeevan Lakshya', 450000, 35, 80);

-- --------------------------------------------------------

--
-- Table structure for table `premium`
--
```

```sql
CREATE TABLE `premium` (
  `Policy_Num` int(15) NOT NULL,
  `Premium` int(10) NOT NULL,
  `Mode` varchar(3) NOT NULL,
  `Last_date` date NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `premium`
--

INSERT INTO `premium` (`Policy_Num`, `Premium`, `Mode`, `Last_date`) VALUES
(123564789, 3500, 'YLY', '2018-12-01'),
(284049583, 469, 'MLY', '2018-12-01');

-- --------------------------------------------------------

--
-- Table structure for table `unpaid_premium`
--

CREATE TABLE `unpaid_premium` (
  `Policy_Num` int(15) NOT NULL,
  `Fine` int(10) NOT NULL,
  `Lateness` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `unpaid_premium`
--

INSERT INTO `unpaid_premium` (`Policy_Num`, `Fine`, `Lateness`) VALUES
(123564789, 0, 0),
(284049583, 0, 0);

--
-- Indexes for dumped tables
--

--
-- Indexes for table `agent`
--
ALTER TABLE `agent`
  ADD PRIMARY KEY (`Agent_code`);

--
-- Indexes for table `customer`
--
```

```sql
ALTER TABLE `customer`
  ADD PRIMARY KEY (`Customer_Num`);

--
-- Indexes for table `paid_premium`
--
ALTER TABLE `paid_premium`
  ADD PRIMARY KEY (`Receipt_Num`),
  ADD KEY `paid_premium_ibfk_1` (`Policy_Num`);

--
-- Indexes for table `policy_data`
--
ALTER TABLE `policy_data`
  ADD PRIMARY KEY (`Policy_Num`),
  ADD KEY `Agent_code` (`Agent_code`),
  ADD KEY `Customer_Num` (`Customer_Num`);

--
-- Indexes for table `premium`
--
ALTER TABLE `premium`
  ADD PRIMARY KEY (`Policy_Num`);

--
-- Indexes for table `unpaid_premium`
--
ALTER TABLE `unpaid_premium`
  ADD PRIMARY KEY (`Policy_Num`);

--
-- AUTO_INCREMENT for dumped tables
--

--
-- AUTO_INCREMENT for table `customer`
--
ALTER TABLE `customer`
  MODIFY `Customer_Num` bigint(10) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=10003;

--
-- Constraints for dumped tables
--

--
-- Constraints for table `paid_premium`
--
ALTER TABLE `paid_premium`
```

```
  ADD CONSTRAINT `paid_premium_ibfk_1` FOREIGN KEY (`Policy_Num`) REFERENCES `premium`
(`Policy_Num`) ON DELETE CASCADE ON UPDATE CASCADE;

--
-- Constraints for table `policy_data`
--
ALTER TABLE `policy_data`
  ADD CONSTRAINT `Agent_code` FOREIGN KEY (`Agent_code`) REFERENCES `agent` (`Agent_code`)
ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `Customer_Num` FOREIGN KEY (`Customer_Num`) REFERENCES `customer`
(`Customer_Num`) ON DELETE CASCADE ON UPDATE CASCADE;

--
-- Constraints for table `premium`
--
ALTER TABLE `premium`
  ADD CONSTRAINT `premium_ibfk_1` FOREIGN KEY (`Policy_Num`) REFERENCES `policy_data`
(`Policy_Num`) ON DELETE CASCADE ON UPDATE CASCADE;

--
-- Constraints for table `unpaid_premium`
--
ALTER TABLE `unpaid_premium`
  ADD CONSTRAINT `Policy` FOREIGN KEY (`Policy_Num`) REFERENCES `premium` (`Policy_Num`) ON
DELETE CASCADE ON UPDATE CASCADE;
COMMIT;
```

The above code has Data definition statements, definition of integrity constraints, sample input statements.
The relations ‘policy_data’ and ‘paid_premium’ have check constraints in their Create table statements.

Triggers:

To demonstrate this, it is given that the agent registrations are closed for now and the details of existing agents cannot be updated currently.

To say the registrations are closed:

```
DELIMITER $$

CREATE TRIGGER agent_create
BEFORE INSERT
ON agent FOR EACH ROW
BEGIN
    DECLARE errorMessage VARCHAR(255);
    SET errorMessage = CONCAT('new registrations ',' not required right now ');

    IF 4> 3 THEN
```

```
        SIGNAL SQLSTATE'45000'
            SET MESSAGE_TEXT = errorMessage;
    END IF;
END $$

DELIMITER ;
```

To say the updating of details are not allowed:

```
DELIMITER $$

CREATE TRIGGER agent_update
BEFORE UPDATE
ON agent FOR EACH ROW
BEGIN
    DECLARE errorMessage VARCHAR(255);
    SET errorMessage = CONCAT('changes','currently not allowed ');

    IF 4> 3 THEN
        SIGNAL SQLSTATE'45000'
            SET MESSAGE_TEXT = errorMessage;
    END IF;
END $$

DELIMITER ;
```

SQL Queries:

To find the sum of premium amount from the relation premium:

```
SELECT SUM(Premium) FROM premium;
```

To find the average of premium amount from the relation premium:

```
SELECT AVG(Premium) FROM premium;
```

To find the maximum of premium amount in the relation premium:

```
SELECT MAX(Premium) FROM premium;
```

To find the number of policies who’s premium is yet to be paid:

```
SELECT COUNT(Policy_Num) FROM unpaid_premium;
```

Select the customers’ details who are currently policy holders:

```
SELECT *
FROM customer
WHERE EXISTS
  ( SELECT 1
     FROM policy_data
     WHERE customer.Customer_Num=policy_data.Customer_Num);
```

Select the entries of policies for which the premium amount is paid:

```
SELECT * FROM premium WHERE EXISTS ( SELECT 1 FROM paid_premium WHERE
premium.Policy_Num=paid_premium.Policy_Num)
```

Select the policy number whose premium amount is greater than the average premium amount:

```
SELECT Policy_Num FROM premium WHERE Premium>=(SELECT AVG(Premium) FROM premium);
```

Select both agents and customers who reside in areas with the same Pincode:

```
SELECT Agent_Name FROM agent WHERE Pincode='390016'
UNION
SELECT FIRST_Name FROM customer WHERE Pincode='390016';
```