PES UNIVERSITY

100 feet Ring Road, BSK 3rd Stage, Bengaluru 560085

Department of Computer Science and Engineering

Jan – May 2020

UE18CS252

Database Management Systems

Project Report

Vehicle Insurance Management System

PES1201800308 Bhargav SNV

4th Sem Sec. I Roll No. 15

PROJECT SUMMARY

This project aims to implement a system to record and maintain data about vehicle insurances and all related data by an insurance agency. An apt and descriptive data model has been described under the data model section on page 4. This project aims to replicate real world use cases for data queries and has been described as such in the below sections. Triggers have been created to not only perform audit-tails to log changes for users and insurance schemes, but also to help generation of registration details easier. Overall, the project aims to build a concise and effective database solution for use by insurance agencies.

Introduction

This project aims to implement a system to record and maintain data about vehicle insurances and all related data by an insurance agency. It efficiently stores data about the agency's employees, customers and all related assets like credentials, vehicle details, insurances issued, validity of schemes, etc.

Thus, the mini-world chosen is that of an insurance agency which has employees who have job-roles and login credentials, customers who own vehicles, which have registration details and insurance. The entities in this mini world are as follows:

- Users
- User's login credentials
- User's Job roles
- Customers
- Vehicles
- Vehicle Insurance

The database has multiple relations between entities, these can be seen pictorially through the schema described in the later sections. In brief these relationships are:

- Users *have* login credentials and roles
- Users *manage* customers
- Customers *own* vehicles
- Vehicles *have* registration and insurance
- Insurance *needs* registration

Data Model

The mini-world is represented by the below schema.

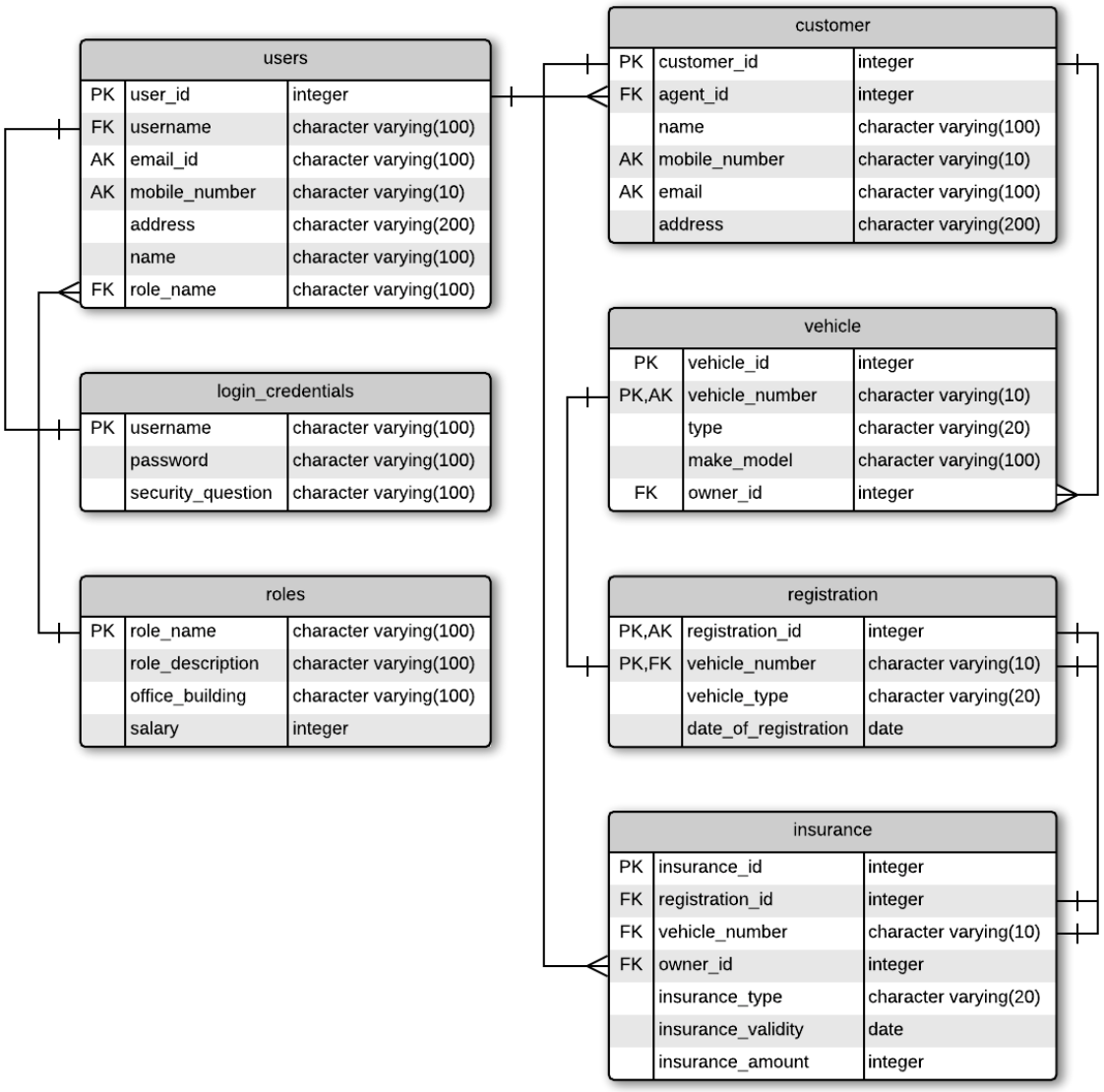

Table Keys:

- **Users**
  - Primary Key: *user_id*
  - Alternate Keys: *email_id, mobile_number*
- **Login_Credentials**
  - Primary Key: *username*
- **Roles**
  - Primary Key: *role_name*
- **Customer**

    - Primary Key: *customer_id*
    - Alternate Keys: *mobile_number, email*
- **Vehicle**
    - Primary Key: *vehicle_id, vehicle_number*
- **Registration**
    - Primary Key: *registration_id, vehicle_number*
    - Here, *vehicle_number* is chosen as primary and foreign key so that no mismatch can occur for a given pair *registration_id* and *vehicle_number*.
- **Insurance**
    - Primary Key: *insurance_id*

FD and Normalization

The database has been normalised to 3 NF. This is violated if in the schema, **Users, Login_Credentials, Roles** were stored as a single table instead of 3. This is because of transitive dependency. The primary key of **Users** can determine *roles* and *username*, which in turn can determine *role_description, office_building, salary* and *password, security_question* respectively.

The functional dependencies are now only between primary keys and other attributes of a table. These functional dependencies are:

- **Users**:
    - *user_id -> username, email_id, mobile_number, address, name, role_name*
- **Login_Credentials:**
    - *username -> password, security_question*
- **Roles:**
    - *role_name -> role_description, office_building, salary*
- **Customer**:
    - *customer_id -> agent_id, name, mobile_number, email, address*
- **Vehicle**:
    - *vehicle_id, vehicle_number -> type, make_model, owner_id*
- **Registration**:
    - *registration_id, vehicle_number -> vehicle_type, date_of_registration*
- **Insurance**:
    - *insurance_id -> registration_id, vehicle_number, owner_id, insurance_type, insurance_validity, insurance_amount*

DDL

```
CREATE TABLE Login_Credentials (
  username varchar(100),
  password varchar(100) NOT NULL CHECK (char_length(password) > 7 AND
char_length(password) <= 100),
  security_question varchar(100),
  PRIMARY KEY (username)
);

CREATE TABLE Roles (
  role_name varchar(100),
  role_description varchar(100),
  office_building varchar(100),
  salary integer,
  PRIMARY KEY (role_name)
);

CREATE TABLE Users (
  user_id integer,
  username varchar(100),
  email_id varchar(100) NOT NULL UNIQUE,
  mobile_number varchar(10) NOT NULL UNIQUE CHECK (char_length(mobile_number) =
10),
  address varchar(200) NOT NULL,
  name varchar(100) NOT NULL,
  role_name varchar(100),
  PRIMARY KEY (user_id),
  FOREIGN KEY (role_name) REFERENCES Roles (role_name) ON DELETE SET NULL ON
UPDATE CASCADE,
  FOREIGN KEY (username) REFERENCES Login_Credentials (username) ON DELETE SET
NULL ON UPDATE CASCADE
);

CREATE TABLE Customer (
  customer_id integer,
  agent_id integer,
  name varchar(100) NOT NULL,
  mobile_number varchar(10) NOT NULL UNIQUE CHECK (char_length(mobile_number) =
10),
  email varchar(100) NOT NULL UNIQUE,
  address varchar(200) NOT NULL,
  PRIMARY KEY (customer_id),
  FOREIGN KEY(agent_id) REFERENCES Users(user_id) ON DELETE SET NULL ON UPDATE
CASCADE
);

CREATE TABLE Vehicle (
  vehicle_id integer,
  vehicle_number varchar(15) UNIQUE CHECK ( vehicle_number ~ $$^[A-Z]{2}\s[0-
9]{2}\s[A-Z]{2}\s[0-9]{4}$$),  -- check with regex
  type varchar(20) NOT NULL,
  make_model varchar(100),
  owner_id integer,
  PRIMARY KEY (vehicle_id),
  FOREIGN KEY(owner_id) REFERENCES Customer(customer_id) ON DELETE RESTRICT ON
UPDATE CASCADE
);
```

```
CREATE TABLE Registration (
  registration_id SERIAL,
  vehicle_number varchar(15),
  vehicle_type varchar(20),
  date_of_registration date NOT NULL,
  PRIMARY KEY (registration_id, vehicle_number),
  FOREIGN KEY(vehicle_number) REFERENCES Vehicle(vehicle_number) ON DELETE
RESTRICT ON UPDATE CASCADE
);

CREATE TABLE Insurance (
  insurance_id integer,
  registration_id integer,
  vehicle_number varchar(15),
  owner_id integer,
  insurance_type varchar(20) NOT NULL,
  insurance_validity date NOT NULL,
  insurance_amount integer NOT NULL,
  PRIMARY KEY (insurance_id),
  FOREIGN KEY(registration_id, vehicle_number) REFERENCES
Registration(registration_id, vehicle_number) ON DELETE SET NULL ON UPDATE
CASCADE,
  FOREIGN KEY(owner_id) REFERENCES Customer(customer_id) ON DELETE SET NULL ON
UPDATE CASCADE
);
```

Triggers

3 triggers were made. They are:

- *create_registration_trigger*: This is triggered whenever data is inserted into the **vehicle** table. As data is inserted into the **vehicle** table, registration details are automatically inferred and inserted into the **Registration** table. This helps maintain the proper details of vehicles as and when they are inserted.
- *audit_user_trigger*: This is triggered whenever changes are made to the **Users** table. The *user id* and *time stamp* are logged in the table named **audit_users**.
- *audit_insurance_trigger*: This is triggered whenever changes are made to the **Insurance** table. The *customer id, agent id, insurance id* and *time stamp* are logged into the table named **audit_insurance**.

```
-- Audit Logs and triggers
--------------------------

CREATE TABLE audit_users (
  user_id integer NOT NULL,
  entry_date TIMESTAMP NOT NULL
);

CREATE OR REPLACE FUNCTION auditlogusers() RETURNS TRIGGER AS $table$
  BEGIN
    INSERT INTO audit_users
    VALUES (new.user_id, current_timestamp);
    RETURN NEW;
```

```
  END;
$table$ LANGUAGE plpgsql;

CREATE TRIGGER audit_user_trigger
AFTER INSERT OR UPDATE OR DELETE ON Users
FOR EACH ROW EXECUTE PROCEDURE auditlogusers();

CREATE TABLE audit_insurance (
  owner_id integer NOT NULL,
  agent_id integer NOT NULL,
  insurance_id integer NOT NULL,
  entry_date TIMESTAMP NOT NULL
);

CREATE OR REPLACE FUNCTION auditloginsurance() RETURNS TRIGGER AS $table$
  BEGIN
    INSERT INTO audit_insurance
    VALUES (new.owner_id, (
      SELECT agent_id
      FROM customer
      WHERE customer_id = new.owner_id), new.insurance_id,
current_timestamp);
    RETURN NEW;
  END;
$table$ LANGUAGE plpgsql;

CREATE TRIGGER audit_insurance_trigger
AFTER INSERT OR UPDATE OR DELETE ON Insurance
FOR EACH ROW EXECUTE PROCEDURE auditloginsurance();

CREATE OR REPLACE FUNCTION create_registration() RETURNS TRIGGER AS $table$
  BEGIN
    INSERT INTO Registration(vehicle_number, vehicle_type,
date_of_registration)
    VALUES (new.vehicle_number, new.type, NOW());
    RETURN NEW;
  END;
$table$ LANGUAGE plpgsql;

CREATE TRIGGER create_registration_trigger
AFTER INSERT ON Vehicle
FOR EACH ROW EXECUTE PROCEDURE create_registration();
```

SQL Queries

Aggregate Queries

1. Find the count of customers each agent manages

```
SELECT U.name, C.agent_id, COUNT(C.agent_id)
FROM customer C, users U
WHERE U.user_id = C.agent_id
GROUP BY (U.name, C.agent_id);
```

2. Find agents with salary higher than average salary

```
SELECT name, salary
FROM Users NATURAL JOIN Roles
WHERE salary > (
  SELECT AVG(salary)
  FROM Users NATURAL JOIN Roles
);
```

Nested Queries

1. Find customer name, id, vehicle number and type where vehicles have insurance lasting beyond 2023.

```
SELECT name, customer_id, vehicle_number, type
FROM Customer C INNER JOIN Vehicle V on V.owner_id = C.customer_id
WHERE C.customer_id in (
  SELECT owner_id
  FROM insurance
  WHERE insurance_validity > '2023-02-08'
);
```

2. Find agents with customers having cars registered before 2020.

```
SELECT U.name, U.user_id
FROM Users U
WHERE user_id in (
  SELECT C.agent_id
  FROM Customer C
  WHERE C.customer_id in (
    SELECT V.owner_id
    FROM vehicle V
    WHERE V.vehicle_number in (
      SELECT vehicle_number
      FROM Registration
      WHERE (date_of_registration < '2022-01-01')
    )
  )
);
```

Correlated Queries

1. Find insurance details for all non-geared motor cycles

```
SELECT I.vehicle_number, I.insurance_amount, I.insurance_validity,
I.insurance_type
FROM insurance I
WHERE I.registration_id = (
  SELECT R.registration_id
  FROM Registration R
  WHERE R.vehicle_type = 'MCWOG'
);
```

2. Update vehicle make and model based off type mentioned in Registration

```
UPDATE Vehicle V
SET make_model = CONCAT((
  SELECT R.vehicle_type
  FROM Registration R
  WHERE V.vehicle_number = R.vehicle_number), ': ', make_model
);
```

Outer Join Queries

```
SELECT * FROM Users FULL OUTER JOIN Roles on Users.role_name =
Roles.role_name;

SELECT * FROM Users FULL OUTER JOIN Insurance on Users.user_id =
Insurance.owner_id;
```

Conclusion

This system can efficiently manage vehicle insurances. If equipped with a front-end, it can prove to be a powerful application which insurance companies can use to track and maintain records. The only limitations are there are no alerting methods to inform the end users of events like expiry of insurance. This however can be tackled quite easily and implemented as future scope.