



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 Employees 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	3
Data Model	4
FD and Normalization	6
DDL	7
Sample Insert statements	8
Triggers	9
SQL Queries	11
Conclusion	14

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:

- Employees
- Employee's login credentials
- Employee'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:

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

Data Model

The mini-world is represented by the below ER-Diagram.

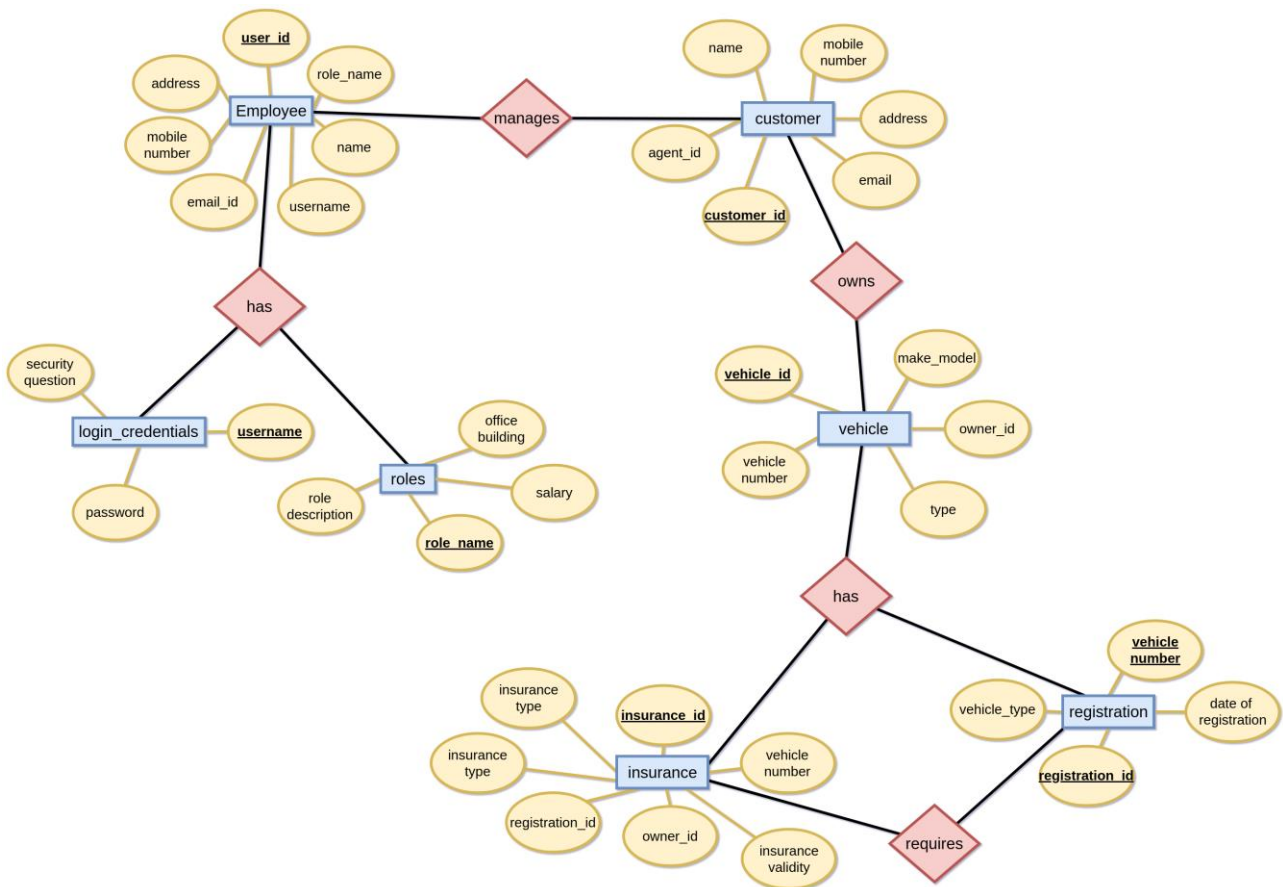
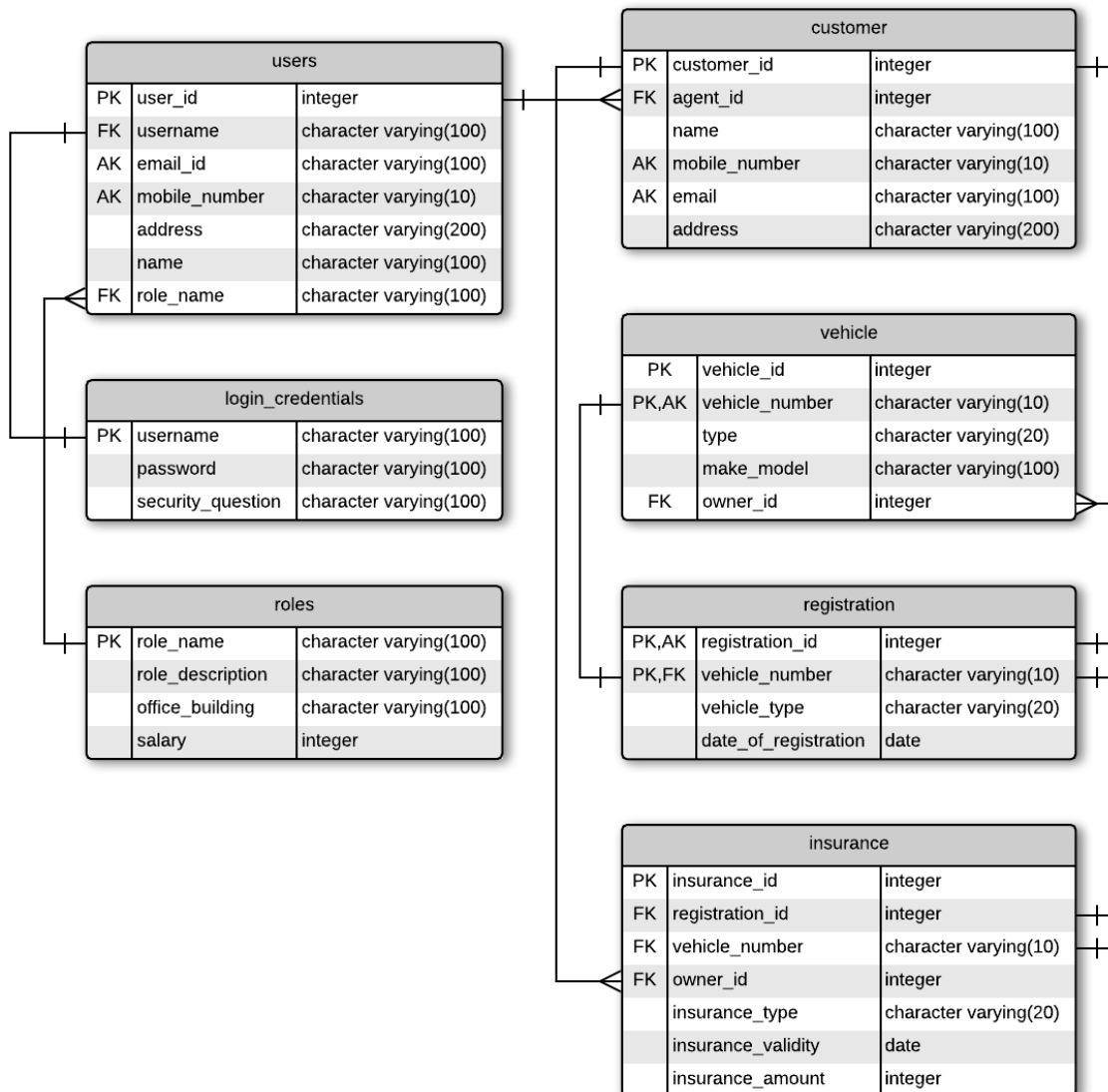


Table Keys:

- **Employee**
 - 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*

The mini-world is represented by the below schema.



FD and Normalization

The database has been normalised to 3 NF. This is violated if in the schema, **Employees**, **Login_Credentials**, **Roles** were stored as a single table instead of 3. This is because of transitive dependency. The primary key of **Employees** can determine *roles* and *Employeeenname*, 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:

- **Employee:**
 - *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 CHECK (salary > 0),  
    PRIMARY KEY (role_name)  
);  
  
CREATE TABLE Employee (  
    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 Employee (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 CHECK (insurance_amount > 0),
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
);

```

Sample Insert statements

- **Login_credentials:**

```
INSERT INTO Login_Credentials VALUES ('user1', 'password1', 'what is your
mothers name?');
```

- **Roles:**

```
INSERT INTO Roles VALUES ('Agent', 'Simple user Agent', 'B01', 50000);
```

- **Employee:**

```
INSERT INTO Employee VALUES (0001, 'user1', 'user1@gmail.com',
'9988776655', '#1, Bank street, Bangalore', 'agent1', 'Agent');
```

- **Customer:**

```
INSERT INTO Customer VALUES (00101, 0001, 'customer1', '8899776655',
'customer1@gmail.com', '#7, UB City, Bangalore');
```

- **Vehicle:**

```
INSERT INTO Vehicle VALUES (00201, 'KA 05 MC 1234', 'LMV', 'Blue Honda
SUV', 00101);
```

- **Insurance:**

```
INSERT INTO Insurance VALUES (00301, 1, 'KA 05 MC 1234', 00101, 'Basic',
'2022-01-08', 40000);
```


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_Employee_trigger*: This is triggered whenever changes are made to the **Employees** table. The *Employee id* and *time stamp* are logged in the table named **audit_Employees**.
- *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 Employee
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 auditlogininsurance();

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, Employee U
WHERE U.Employee_id = C.agent_id
GROUP BY (U.name, C.agent_id);
```

name	agent_id	number_of_customers
agent2	2	2
agent1	1	3
(2 rows)		

2. Find agents with salary higher than average salary

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

name	salary
agent4	500000
agent5	2500000
(2 rows)	

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'
);
```

name	customer_id	vehicle_number	type
customer4	104	KA 05 MC 1134	HMV
customer5	105	KA 05 MC 1255	LMV
customer5	105	KA 05 MK 1295	MCWG
customer5	105	KA 05 MK 1355	MCWOG

(4 rows)

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

```
SELECT U.name, U.Employee_id
FROM Employee U
WHERE Employee_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')
        )
    )
);
```

name	user_id
agent1	1
agent2	2

(2 rows)

Correlated Queries

1. Find insurance details for all non-gear 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'
);
```

vehicle_number	insurance_amount	insurance_validity	insurance_type
KA 05 MK 1355	27000	2018-01-08	Premium

(1 row)

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 Employee FULL OUTER JOIN Roles on Employee.role_name =
Roles.role_name;
```

user_id	username	email_id	mobile_number	address	name	role_name	role_name	role_description	office_building	salary
7	user7	user7@gmail.com	9988776654	#7, Bank street, Bangalore	agent7	Agent	Agent	Simple user Agent	B01	50000
6	user6	user6@gmail.com	9988776645	#6, Bank street, Bangalore	agent6	Agent	Agent	Simple user Agent	B01	50000
1	user1	user1@gmail.com	9988776655	#1, Bank street, Bangalore	agent1	Agent	Agent	Simple user Agent	B01	50000
2	user2	user2@gmail.com	9922776655	#2, Church street, Bangalore	agent2	Manager	Manager	Manages Agents	B02	75000
3	user3	user3@gmail.com	9933776655	#3, Food street, Bangalore	agent3	Assistant Manager	Assistant Manager	Helps Managers	B02	60000
4	user4	user4@gmail.com	9944776655	#4, Bank Lane, Bangalore	agent4	System Administrator	System Administrator	Handles Servers	B04	500000
5	user5	user5@gmail.com	9966776655	#5, Bank street, Bangalore	agent5	CEO	CEO	Runs the company	B03	2500000

(7 rows)

```
SELECT * FROM Employee FULL OUTER JOIN Insurance on Employee.Employee_id = Insurance.owner_id;
```

user_id	username	email_id	mobile_number	address	name	role_name	insurance_id	registration_id	vehicle_number	owner_id	insurance_type	insurance_validity	insurance_amount
							381	1	KA 05 MC 1234	101	Basic	2022-01-08	40000
							382	2	KA 05 MC 1235	102	Premium	2023-01-08	50000
							383	3	KA 05 MC 2234	103	Premium	2023-01-08	50000
							384	4	KA 05 MC 1134	104	Gold-Package	2024-01-08	60000
							385	5	KA 05 MC 1255	105	Gold-Package	2024-01-08	70000
							386	6	KA 05 MK 1295	105	Gold-Package	2019-01-08	28000
							387	7	KA 05 HK 1355	105	Premium	2018-01-08	27000
2	user2	user2@gmail.com	9922776655	#2, Church street, Bangalore	agent2	Manager							
5	user5	user5@gmail.com	9966776655	#5, Bank street, Bangalore	agent5	CEO							
6	user6	user6@gmail.com	9988776645	#6, Bank street, Bangalore	agent6	Agent							
4	user4	user4@gmail.com	9944776655	#4, Bank Lane, Bangalore	agent4	System Administrator							
1	user1	user1@gmail.com	9988776655	#1, Bank street, Bangalore	agent1	Agent							
3	user3	user3@gmail.com	9933776655	#3, Food street, Bangalore	agent3	Assistant Manager							
7	user7	user7@gmail.com	9988776654	#7, Bank street, Bangalore	agent7	Agent							

(14 rows)

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.