***VEHICLE INSURANCE DATABASE***

***DBMS (CS310)***

**PREPARED BY:**

Group:

* B. Deepak (20BCS027)
* Chippagiri Karthik (20BCS036)
* Bhukya Jagadeesh (20BCS032)
* Ankati Vivek (20BCS016)
* Kasse Lokesh Babu (20BCS067)
* K. Nirmal Swaroop(20BCS068)
* K. Balaji (20BCS073)
* P. Vivek Yadav (20BCS105)
* Ravula Veekshith Reddy(20BCS111)
* Rangam Sai Charan(20bcs108)

# Under the Guidance of-

**Dr. Uma Seshadri,**

Professor, HOD, Department of Computer Science

Indian Institute of Information Technology Dharwad.

**Dr. Pramod Yelmewad**

Assistant Professor, Department of Computer Science

Indian Institute of Information Technology Dharwad.

**Dr. Supriya Nadiger**

Assistant Professor, Department of Computer Science

Indian Institute of Information Technology Dharwad.

## 

## TABLE OF CONTENTS

## ACKNOWLEDGEMENT

## AIM

## OBJECTIVE

## ROLES AND RESPONSIBILITIES

## TIMELINE

## PROJECT IMPLEMENTATION

→ Tables Creation

→ Conceptual data model (CDM)

→ Physical data model (PDM)

→ Understanding Database

→ Insertion of valid data

→ Creating functions and stored procedures

→ Writing the queries

→ Execution of queries

→ Working on the errors

🡪CONCLUSION

## ACKNOWLEDGEMENT

We would like to sincerely and profusely thank **Dr Uma Seshadri,** for her able guidance and for giving us the opportunity to take up this project. We would also like to thank **Dr. Pramod Yelmewad** for his guidance. and giving us the opportunity to take up this project.

**AIM:**

Project is meant for helping our learning skills improve in understanding relational database management system (DBMS). We will learn some theoretical and practical concepts of DBMS and how they are implemented in real life.

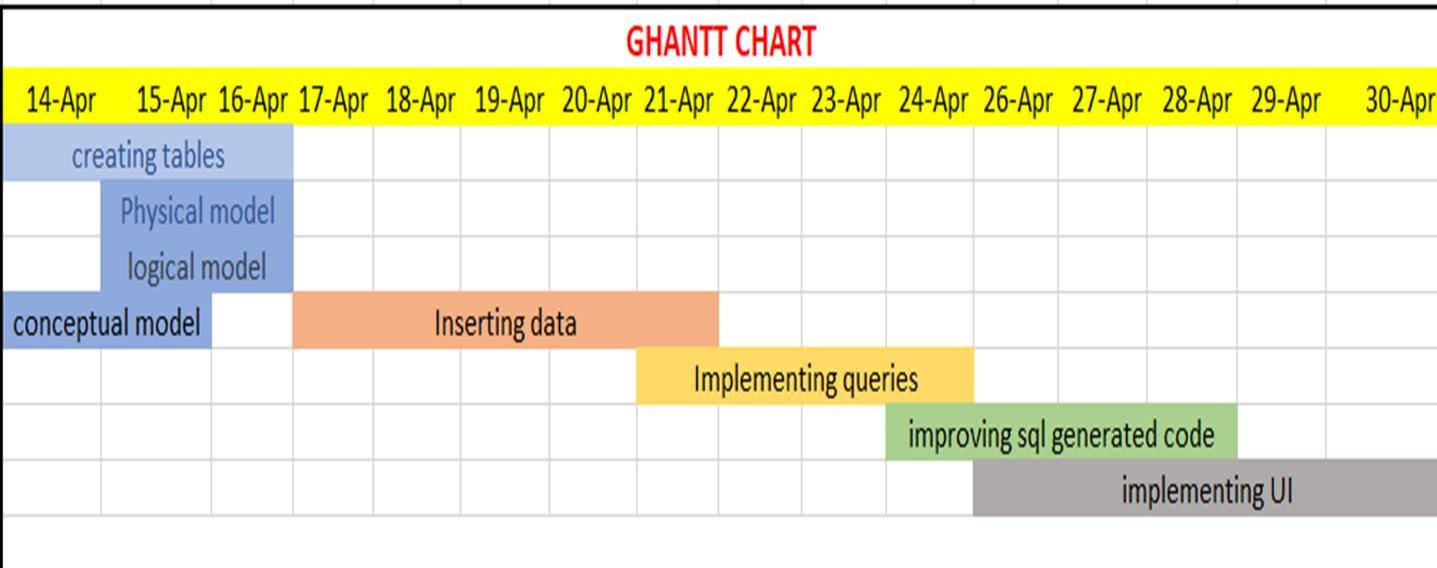
**OBJECTIVE:**

The main objective is to gain practical knowledge on working with a database, encountering and overcoming the problems and challenges faced during its implementation. We need to create a database for a vehicle insurance company. To understand all the relationships, functions, constraints, operators,.etc. which are used while using MySQL. Lastly, an important objective of this project is to be able to collaborate and work as team and bring out value for everybody.

**ROLES AND RESPONSIBILITES:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| STUDENT NAME | ROLE | | ROLLNO | Email |
| B. Deepak | Data insertion, Conceptual DM, | 20BCS027 | | 20bcs027@iiitdwd.ac.in |
| C. Karthik | Data Insertion,  Queries | 20BCS036 | | 20bcs036@iiitdwd.ac.in |
| Jagadeesh | Data Insertion, Physical Model | 20BCS032 | | 20bcs032@iiitdwd.ac.in |
| A. Vivek | Data Insertion,  Queries | 20BCS016 | | 20bcs016@iiitdwd.ac.in |
| K. Lokesh | Data Insertion,  Physical Model | 20BCS067 | | 20bcs067@iiitdwd.ac.in |
| K. Nirmal Swaroop | Data Insertion,  Queries | 20BCS068 | | 20bcs068@iiitdwd.ac.in |
| K. Balaji | Data Insertion,  Physical model | 20BCS073 | | 20bcs073@iiitdwd.ac.in |
| P. Vivek | Data Insertion,  Queries | 20BCS105 | | 20bcs105@iiitdwd.ac.in |
| Sai Charan | Data Insertion,  Queries | 20BCS108 | | 20bcs027@iiitdwd.ac.in |
| Veekshith | Data Insertion,  Queries | 20BCS111 | | 20bcs027@iiitdwd.ac.in |

**TIMELINE:**



**PROJECT IMPLEMENTATION:**

→ Tables Creation

→ Conceptual data model (CDM)

→ Physical data model (PDM)

→ Understanding Database

→ Insertion of valid data

→ Creating functions and stored procedures

→ Writing the queries

→ Execution of queries

→ Working on the errors, if they occur while improving the data base.

**TABLES CREATION:**

According to database the tables which are required are created without inserting any data initially. Table with the column names and rows are created.

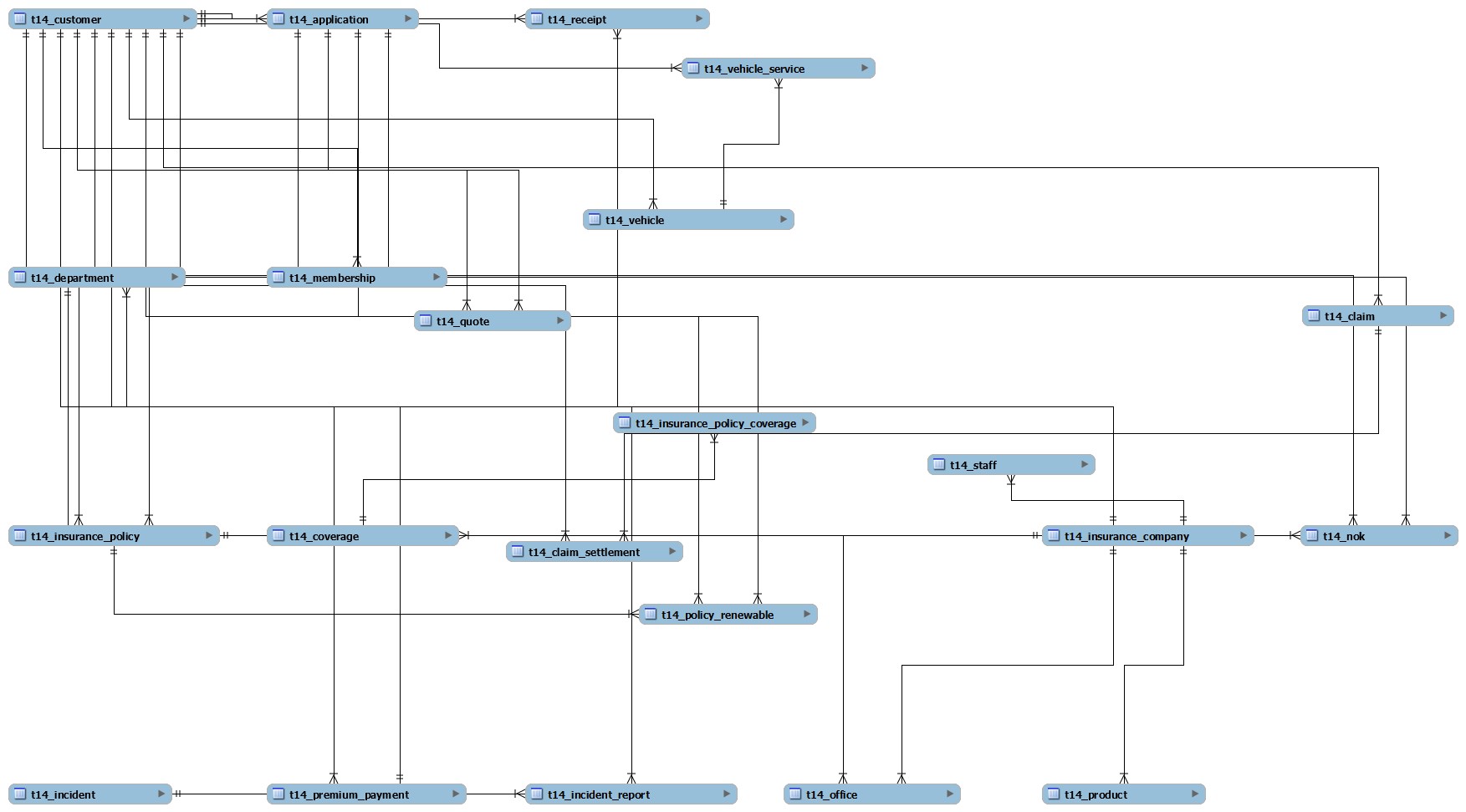
Here are the following tables and the names of the tables,

* t14\_customer;
* t14\_application;
* t14\_quote;
* t14\_insurance\_policy;
* t14\_premium\_payment;
* t14\_vehicle;
* t14\_claim;
* t14\_claim\_settlement;
* t14\_staff;
* t14\_department;
* t14\_office;
* t14\_membership;
* t14\_vehicle\_service;
* t14\_nok;
* t14\_insurance\_companies;
* t14\_policy\_renewable;
* t14\_incident;
* t14\_incident\_report;
* t14\_coverage;
* t14\_product;
* t14\_receipt;
* t14\_insurance\_Policy\_coverage;

**CONCEPTUAL DATA MODEL(CDM):**

The other name for the conceptual data model is a business model. This model focuses on identifying the data used in the business but not its processing flow or physical characteristics. As the conceptual data model is of high level it usually not contains attributes in its structure. This model is used to define the relationship among the data entities but not provide information about cardinality properties.

Below is the **CDM** of vehicle insurance database.



**PHYSICAL DATA MODEL (PDM):**

Physical data model represents how the model will be built in the database. A physical database model shows all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships between tables. Features of a physical data model include:

→ Specification all tables and columns.

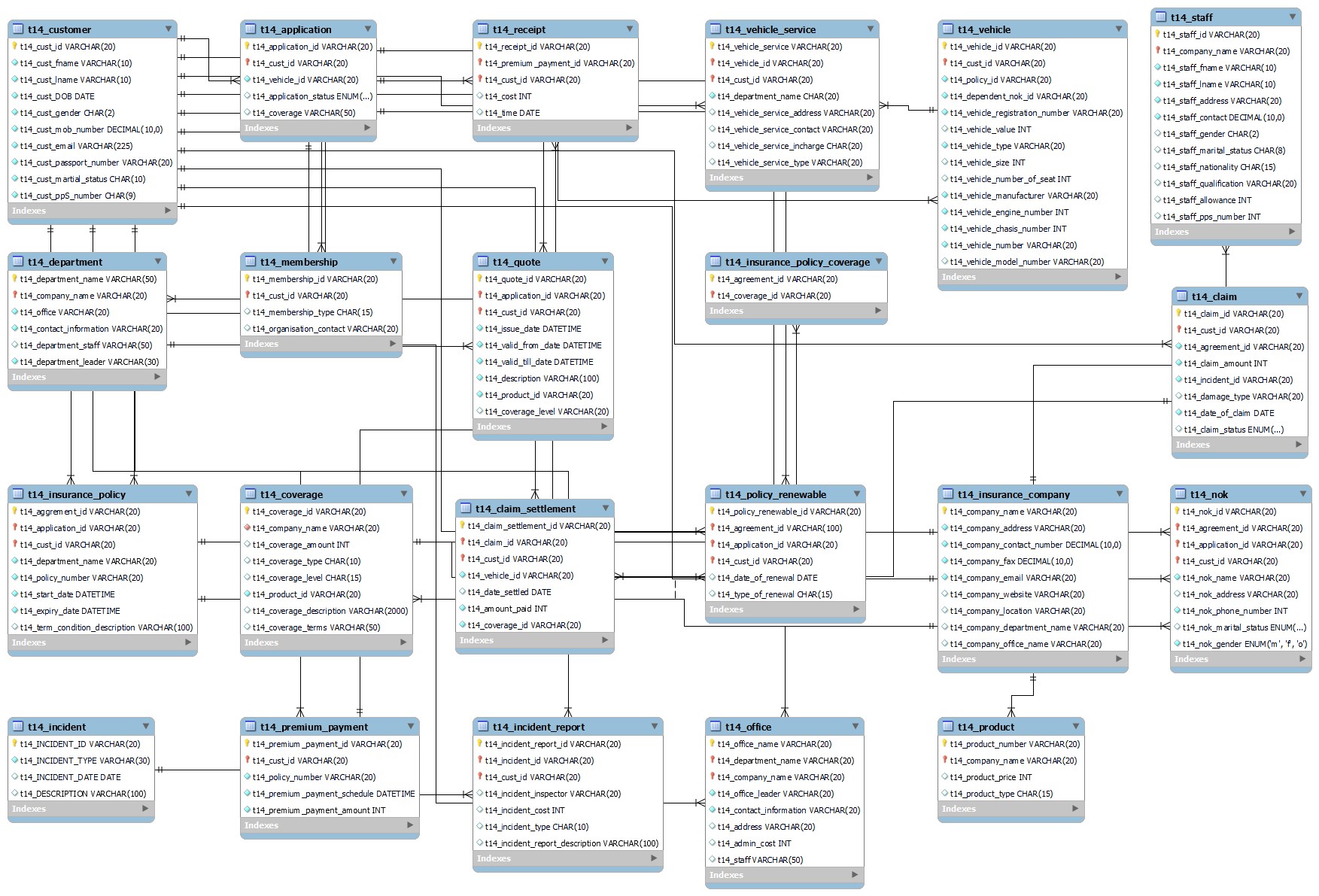
→ Foreign keys are used to identify relationships between tables.

→ Denormalization may occur based on user requirements.

→ Physical considerations may cause the physical data model to be quite different from the logical data model.

→ Physical data model will be different for different RDBMS. For example, data type for a column may be different between MySQL and SQL Server.

Below is the PDM of Vehicle database management system.



**Understanding Relations between tables:**

The main purpose of the Physical data model and Logical Data Model is to understand the cardinality, type of relation and relational constraints on relationship and many more. From the physical and logical data models following are the inference made

Relationship types:

One to one relation: only one mapping can be done from parent entity to child entity i.e. the foreign key in the chid table must be unique. To implement one to one relation in sql we use unique constraint on foreign key in child table. Following are the one-to-one relationships in the database 🡪 Application to Quote

🡪 Customer to membership

🡪 Premium payment to Receipt

🡪 Claim to Claim settlement

Many to Many relation: many mappings can be done from parent entity to child entity and vice versa. To implement many to many relation in sql we use a bridged table with foreign keys from both the table and created a composite primary key to prevent the repetition of same combined attribute. Following are the Many to Many relationships in the database

🡪Insurance policy to coverage,

Bridged\_table:t14\_insurance\_Policy\_coverage;

One to Many relation: many mapping can be done from parent entity to child entity. To implement one to many relation in sql we just reference the foreign key in child table. Following are the Many to Many relationships in the database

🡪 all Remaining relations except the above relations

**Creating Functions and Stored Procedures**

A function is compiled and executed every time whenever it is called. A function must return a value and cannot modify the data received as parameters. In MySQL the creation of function is as followed

CREATE FUNCTION function\_name [ (parameter datatype [, parameter datatype])]

RETURNS return\_datatype

BEGIN

declaration\_section

executable\_section

END;

Stored Procedures are pre-compiled objects which are compiled for the first time and its compiled format is saved, which executes (compiled code) whenever it is called.

DELIMITER &&

CREATE PROCEDURE procedure\_name [[IN | OUT | INOUT] parameter\_name datatype [, parameter datatype]) ]

BEGIN

Declaration\_section

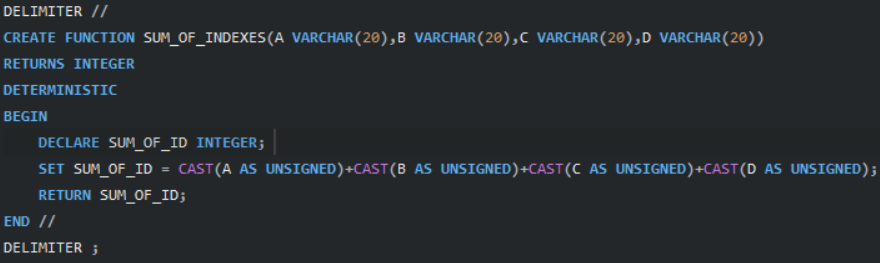
Executable\_section

END &&

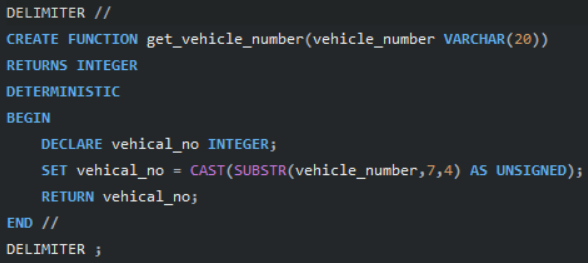
DELIMITER ;

The following are the functions used in the vehicle insurance database:

SUM\_OF\_INDEXES() – To sum up all the indexes and designed for query 6



GET\_VEHICLE\_NUMBER () – To get vehicle number and designed for query 4

****

To get data stored procedure Get data is used and to delete data stored procedure

Delete data is used.

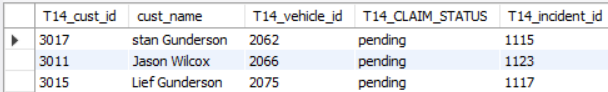
**QUERIES:**

Query 1:

Retrieve Customer and Vehicle details who has been involved in an incident and claim status is pending – Customer, vehicle, claim status, incident

Sol: SELECT c.T14\_cust\_id, CONCAT (T14\_cust\_fname, ' ', T14\_cust\_lname) AS cust\_name, v.T14\_vehicle\_id, T14\_CLAIM\_STATUS, T14\_incident\_id FROM T14\_claim cl JOIN T14\_vehicle v ON v.T14\_policy\_id = cl. T14\_agreement\_id JOIN T14\_customer c ON c.T14\_cust\_id = v.T14\_cust\_id WHERE T14\_incident\_id IS NOT NULL AND T14\_claim\_status LIKE 'pending';

RESULT OF THE QUERY:

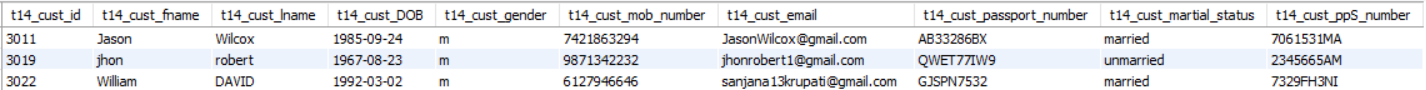


Query 2:

Retrieve customer details who has premium payment amount greater than the sum of all the customer\_Ids in the database – premium payment, customer

Sol: CREATE VIEW cust\_sum AS SELECT SUM(T14\_cust\_id) FROM T14\_customer; SELECT \* FROM T14\_CUSTOMER WHERE T14\_CUST\_ID IN (SELECT T14\_CUST\_ID FROM T14\_PREMIUM\_PAYMENT WHERE T14\_premium\_payment\_amount > (SELECT \* FROM cust\_sum)); **(OR)**

SELECT \* FROM T14\_CUSTOMER WHERE T14\_CUST\_ID IN (SELECT T14\_CUST\_ID FROM T14\_PREMIUM\_PAYMENT WHERE T14\_premium\_payment\_amount > (SELECT SUM(T14\_cust\_id) FROM T14\_customer));

RESULT OF THE QUERY:

Query 3:

Retrieve Company details whose number of products is greater than departments, where the departments are located in more than one location— company, product, departments, office Query 3:

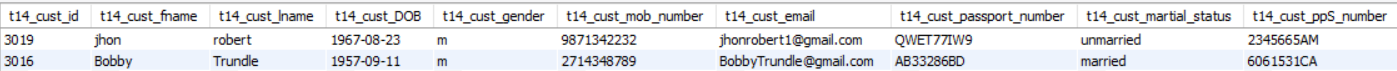
SELECT \* FROM t14\_insurance\_companies WHERE t14\_company\_name in(select o.t14\_company\_name from t14\_PRODUCT p inner join t14\_OFFICE o on o.t14\_Company\_Name = p.t14\_Company\_Name group by o.t14\_Company\_Name having Count(distinct(t14\_Product\_Number)) <count(distinct(t14\_Department\_Name) ) and count(t14\_address)>1);

RESULT OF THE QUERY:

Query 4:

Select Customers who have more than one Vehicle, where the premium for one of the Vehicles is not paid and it is involved in accident

select T14\_customer.\* from T14\_Customer where T14\_customer.T14\_cust\_id IN( SELECT c.T14\_cust\_id from T14\_customer c join t14\_incident\_report IR on c.T14\_cust\_id = IR.T14\_cust\_id left join t14\_receipt R on c.T14\_cust\_id = r.T14\_cust\_id where c.T14\_cust\_id in ( select v.T14\_cust\_id from T14\_vehicle V group by T14\_cust\_id having count(V.T14\_cust\_id)>1) and R.t14\_receipt\_id is null and T14\_incident\_type like "%accident%");

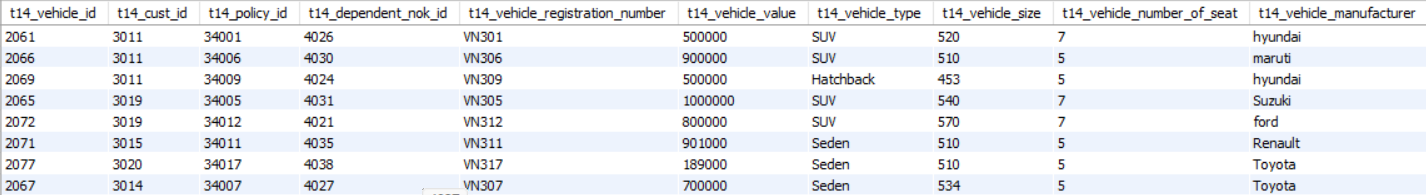
RESULT OF THE QUERY:

Query 5:

Select all vehicles which have premium more than its vehicle number.

SELECT t14\_vehicle.\* FROM t14\_vehicle JOIN t14\_premium\_payment ON t14\_vehicle.T14\_cust\_id = t14\_premium\_payment.T14\_cust\_id WHERE t14\_premium\_payment\_amount > GET\_VEHICLE\_NUMBER(t14\_vehicle\_number);

RESULT OF THE QUERY:



Query 6:

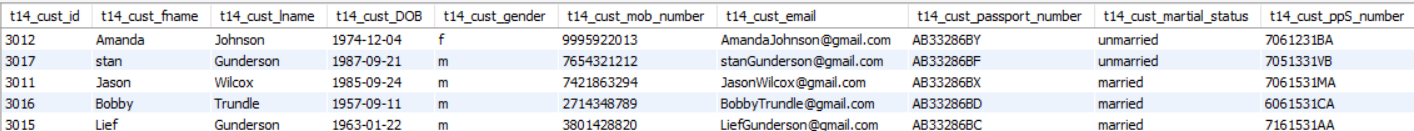
Retrieve Customer details whose Claim Amount is less than Coverage

Amount and Claim Amount is greater than Sum of (CLAIM\_SETTLEMENT\_ID,

VEHICLE\_ID, CLAIM\_ID, CUST\_ID)

SELECT \* FROM T14\_CUSTOMER WHERE T14\_CUST\_ID IN (SELECT CL.T14\_CUST\_ID FROM T14\_COVERAGE CV JOIN T14\_INSURANCE\_POLICY\_COVERAGE `IPC` ON `IPC`.T14\_COVERAGE\_ID=CV.T14\_COVERAGE\_ID JOIN T14\_INSURANCE\_POLICY IP ON `IPC`.T14\_AGREEMENT\_ID=IP.T14\_AGGREMENT\_ID JOIN T14\_CUSTOMER C ON IP.T14\_CUST\_ID = C.T14\_CUST\_ID JOIN T14\_CLAIM CL ON CL.T14\_CUST\_ID = C.T14\_CUST\_ID JOIN T14\_CLAIM\_SETTLEMENT CLS ON CL.T14\_CLAIM\_ID = CLS.T14\_CLAIM\_ID JOIN T14\_VEHICLE V ON V.T14\_CUST\_ID = C.T14\_CUST\_ID WHERE T14\_COVERAGE\_AMOUNT>T14\_CLAIM\_AMOUNT AND T14\_CLAIM\_AMOUNT>(SUM\_OF\_INDEXES(CL.T14\_CUST\_ID,CLS.T14\_CLAIM\_ID,CLS.T14\_CLAIM\_SETTLEMENT\_ID,V.T14\_VEHICLE\_ID)));

RESULT OF THE QUERY:



**CONCLUSION:**

A complete Car Vehicle Insurance company database is completely implemented and all the given project queries are executed and are completely working fine giving at least one line of output. Each table consists of at least 10 tuples of data. Further developments for this project can be making a user interface for this database to perform INSERT, UPDATE, DELETE operations, and Normalisation of a few tables till 4th NF. To adjust the data according to queries functions are used ad stored procedures are used to remove and select the data. Views also assist in the implementation of queries.