**TABLE OF CONTENTS**

[1. Introduction 2](#_Toc146532267)

[2. Scope 2](#_Toc146532268)

[3. Creation of ER Diagram 3](#_Toc146532269)

[4. Conversion of ER Diagram to Relation 3](#_Toc146532270)

[5. Normalization 4](#_Toc146532271)

[6. Implementation 6](#_Toc146532272)

[7. Conclusion 13](#_Toc146532273)

[8. Result 13](#_Toc146532274)

## Introduction

This project demonstrates a basic car insurance system that allows users to do a variety of car insurance-related tasks, such as entering owner and vehicle information, setting insurance policies, reporting accidents, submitting claims, and creating reports.

This project involves connecting to a database and using stored procedures to define the function of each activity, such as adding owner information, generating insurance policies. It also utilizes triggers to manage insurance status when a claim is filed, approved, or rejected. The database has views that are created to generate reports for car owners, accidents, claims, insurances, etc.

The program used to provide a user interface is written in python as a CLI application. Users can choose their preferred action from a menu, and the appropriate function is executed based on their selection.

The project repository can be found here: [suparthghimire/car-insurance-project (github.com)](https://github.com/suparthghimire/car-insurance-project)

## Scope

1. Allow users to add information about a car owner and car details.
2. Implement a feature to create insurance for cars.
3. Allow users to report car accidents, providing details like date, time, location, and cause.
4. Enable car owners to file insurance claims for accidents or damages.
5. Provide a mechanism for managing insurance claims.
6. Generate reports that provide information about car owners.
7. Create reports detailing car accidents, including involved vehicles.
8. Generate reports specific to insurance claims and accidents.
9. Produce reports summarizing successful insurance claims.
10. Generate reports that show successful insurance claims grouped by the agent responsible.
11. Automatic updates on insurance status when a claim is filed, approved, or rejected.

## Creation of ER Diagram

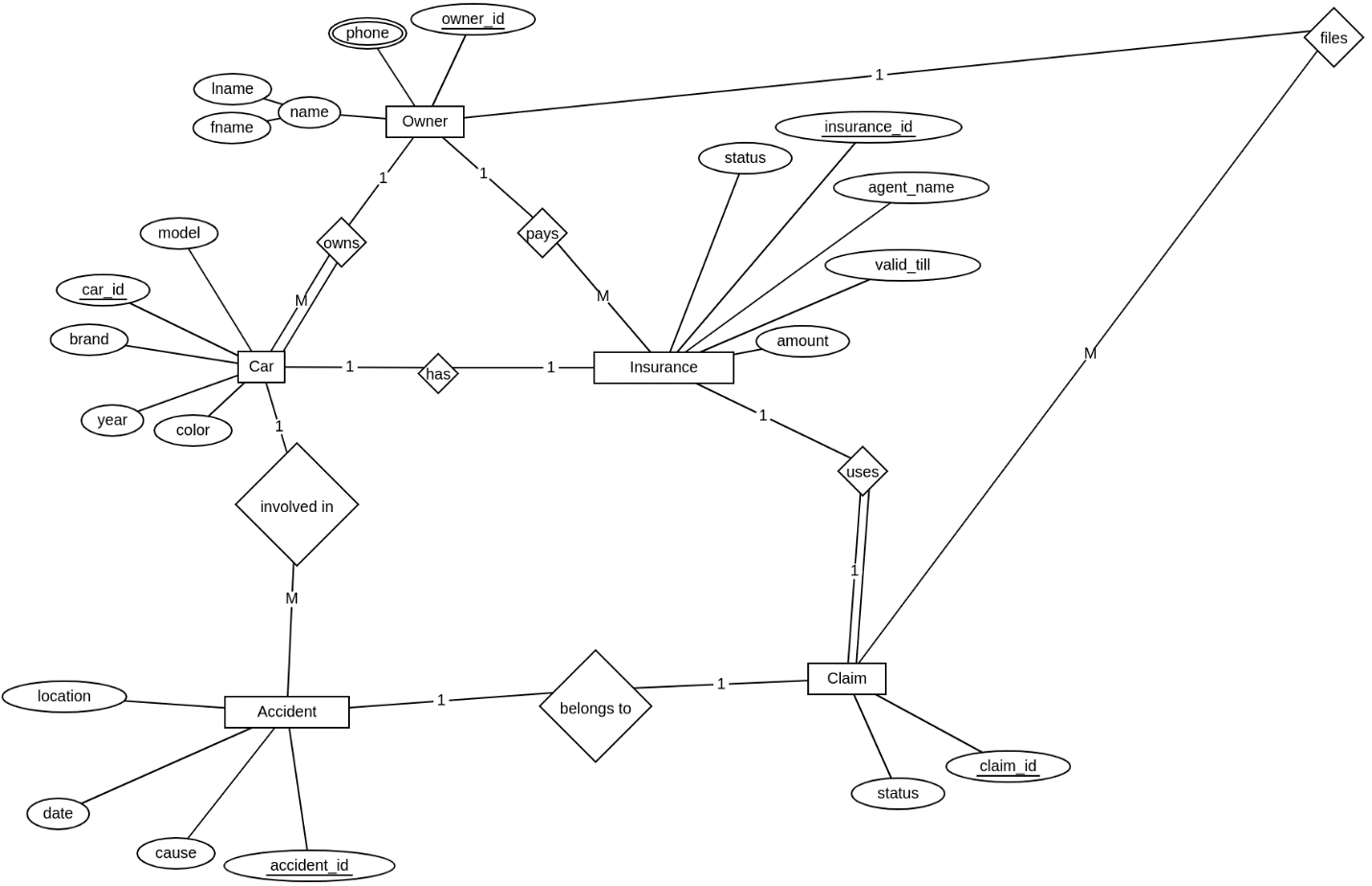


Figure ER Diagram of Car Insurance System

## Conversion of ER Diagram to Relation

1. **Conversion of strong entities**: For each strong entity create a separate table with the same name. Include all attributes as relation columns, if there is any composite attribute divide them into simple attributes add them as relation columns. Multivalued attributes are ignored at this stage. A primary key is also selected in this stage.
2. **Conversion of weak entity:** For each weak entity create a separate table with the same name. Include all attributes as table columns. Include the Primary key of a strong entity as foreign key of the weak entity. Declare the combination of foreign key and decimator attribute as Primary key from the weak entity.
3. **Conversion of one-to-one relationship:** For each one-to-one relationships take primary key of either relation and include it in another. In case of full participation, entity with full participation must contain the primary key of other relation as foreign key.
4. **Conversion of one-to-many relationship:** For each one-to-many relationships take primary key of relation with relationship cardinality 1 and include it to relation with relation cardinality M as foreign key.
5. **Conversion of many-many relationships:** For each many-to-many relationships make a new table that includes primary key of both tables as its foreign key.
6. **Conversion of multivalued attributes:** For each multivalued attribute create a separate table and include the primary key of the parent relation as foreign key.
7. **Conversion of n-ary relationship:** For each n-ary relationships create a separate table and include the primary key of all participating entities as foreign key. The combination of foreign keys is the primary key of the new relation.

The relations created after conversion are as follows.

**Owner**

|  |  |  |  |
| --- | --- | --- | --- |
| owner\_id | fname | lname | phone |

**Insurance**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| insurance\_id | status | amount | valid\_till | agent\_name | owner\_id |

**Car**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| car\_id | model | year | brand | color | owner\_id | insurance\_id |

**Claim**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| claim\_id | status | owner\_id | accident\_id | insurance\_id |

**Accident**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| accident\_id | cause | date | location | car\_id |

## Normalization

Normalization of relation is done based on normalization degree.

1. **First Normal Form (1NF):**
2. A relation will be 1NF if it contains an atomic value.
3. It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attributes.
4. First normal form disallows the multi-valued attribute, composite attribute, and their combinations.
5. **Second Normal Form (2NF):**
6. In the 2NF, relation must be in 1NF.
7. In the second normal form, all non-key attributes are fully functional dependent on the primary-key.
8. **Third Normal Form (3NF):**
9. It satisfies the First Normal Form and the Second Normal form.
10. And, it doesn't have Transitive Dependency.
11. **Boyce Codd Normal Form (BCNF):**
12. BCNF is the advanced version of 3NF. It is stricter than 3NF.
13. A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
14. For BCNF, the table should be in 3NF, and for every FD, LHS is a super key.

After normalization for each normal form, the relations are normalized upto BCNF.

Tables after normalization are as follows:

**Owner**

|  |  |  |
| --- | --- | --- |
| owner\_id | fname | lname |

**Owner**\_phone

|  |  |
| --- | --- |
| owner\_id | phone |

**Insurance**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| insurance\_id | status | amount | valid\_till | agent\_name | paid\_by |

**Car**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| car\_id | model | year | brand | color | owner\_id | insurance\_id |

**Accident**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| accident\_id | cause | date | location | car\_id |

**Claim**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| claim\_id | status | filed\_by | insurance\_id | accident\_id |

## Implementation

1. **Database**
   1. **Code for Database creation:**

DROP DATABASE IF EXISTS insurance;

CREATE DATABASE insurance;

USE insurance;

* 1. **Code to create tables:**

CREATE TABLE tbl\_owner(

owner\_id INT AUTO\_INCREMENT,

fname VARCHAR (255),

lname VARCHAR (255),

PRIMARY KEY(owner\_id)

);

CREATE TABLE tbl\_owner\_phone(

phone BIGINT,

owner\_id INT,

FOREIGN KEY(owner\_id) REFERENCES tbl\_owner (owner\_id)

);

CREATE TABLE tbl\_insurance(

insurance\_id INT AUTO\_INCREMENT,

status ENUM(

'OPEN',

'CLAIMED',

'PENDING',

'REJECTED'

) DEFAULT 'OPEN',

valid\_till DATETIME,

amount INT,

agent\_name VARCHAR(255),

paid\_by INT,

FOREIGN KEY(paid\_by) REFERENCES tbl\_owner (owner\_id),

PRIMARY KEY (insurance\_id)

);

CREATE TABLE tbl\_car(

car\_id INT AUTO\_INCREMENT,

model VARCHAR(255),

color VARCHAR(255),

brand VARCHAR(255),

year YEAR,

owner\_id INT,

FOREIGN KEY(owner\_id) REFERENCES tbl\_owner (owner\_id),

insurance\_id INT,

FOREIGN KEY(insurance\_id) REFERENCES tbl\_insurance (insurance\_id),

PRIMARY KEY(car\_id)

);

CREATE TABLE tbl\_accident(

accident\_id INT AUTO\_INCREMENT,

cause TEXT,

date DATETIME,

location VARCHAR(255),

car\_id INT,

FOREIGN KEY(car\_id) REFERENCES tbl\_car (car\_id),

PRIMARY KEY (accident\_id)

);

CREATE TABLE tbl\_claim(

claim\_id INT AUTO\_INCREMENT,

status ENUM('PENDING', 'APPROVED','REJECTED') DEFAULT 'PENDING',

filed\_by INT NOT NULL,

FOREIGN KEY(filed\_by) REFERENCES tbl\_owner (owner\_id),

insurance\_id INT NOT NULL,

FOREIGN KEY(insurance\_id) REFERENCES tbl\_insurance (insurance\_id),

accident\_id INT NOT NULL,

FOREIGN KEY(accident\_id) REFERENCES tbl\_accident (accident\_id),

PRIMARY KEY (claim\_id),

UNIQUE KEY (insurance\_id, accident\_id, filed\_by)

);

* 1. **Code for Stored Procedures**

DELIMITER //

DROP PROCEDURE IF EXISTS insert\_owner;

CREATE PROCEDURE insert\_owner(IN fname VARCHAR(255), IN lname VARCHAR(255))

BEGIN

INSERT INTO tbl\_owner(fname, lname)

VALUES(fname, lname);

DROP PROCEDURE IF EXISTS insert\_car;

CREATE PROCEDURE insert\_car(IN model VARCHAR(255), IN color VARCHAR(255), IN brand VARCHAR(255), IN year YEAR, IN owner\_id INT)

BEGIN

INSERT INTO tbl\_car(model, color, brand, year, owner\_id)

VALUES(model, color, brand, year, owner\_id);

DROP PROCEDURE IF EXISTS report\_accident;

CREATE PROCEDURE report\_accident(IN cause TEXT, IN date DATETIME, IN location VARCHAR(255), IN car\_id INT)

BEGIN

INSERT INTO tbl\_accident(cause, date, location, car\_id)

VALUES(cause, date, location, car\_id);  
  
DROP PROCEDURE IF EXISTS file\_claim;

CREATE PROCEDURE file\_claim(IN filed\_by INT, IN p\_insurance\_id INT, IN accident\_id INT)

BEGIN

IF (SELECT (SELECT valid\_till from tbl\_insurance WHERE insurance\_id =p\_insurance\_id ) > NOW()) THEN

INSERT INTO tbl\_claim(filed\_by, insurance\_id, accident\_id)

VALUES( filed\_by, p\_insurance\_id, accident\_id);

ELSE

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Insurance is not valid';

END IF;

DROP PROCEDURE IF EXISTS approve\_claim;

CREATE PROCEDURE approve\_claim(IN p\_claim\_id INT)

BEGIN

UPDATE tbl\_claim

SET status = 'APPROVED'

WHERE claim\_id = p\_claim\_id;

DROP PROCEDURE IF EXISTS reject\_claim;

CREATE PROCEDURE reject\_claim(IN p\_claim\_id INT)

BEGIN

UPDATE tbl\_claim

SET status = 'REJECTED'

WHERE claim\_id = p\_claim\_id;

DROP PROCEDURE IF EXISTS insert\_phone;

CREATE PROCEDURE insert\_phone(IN phone BIGINT, IN owner\_id INT)

BEGIN

INSERT INTO tbl\_owner\_phone(phone, owner\_id)

VALUES(phone, owner\_id);

DROP PROCEDURE IF EXISTS delete\_phone;

CREATE PROCEDURE delete\_phone(IN phone BIGINT, IN owner\_id INT)

BEGIN

DELETE FROM tbl\_owner\_phone

WHERE phone = phone AND owner\_id = owner\_id;

END // DELIMITER;

* 1. **Code for Triggers**

DELIMITER //  
  
DROP TRIGGER IF EXISTS update\_insurance\_status;

CREATE TRIGGER update\_insurance\_status

AFTER INSERT ON tbl\_claim

FOR EACH ROW

BEGIN

UPDATE tbl\_insurance

SET status = 'PENDING'

WHERE insurance\_id = NEW.insurance\_id;  
  
DROP TRIGGER IF EXISTS update\_insurance\_status\_claimed;

CREATE TRIGGER update\_insurance\_status\_claimed

AFTER UPDATE ON tbl\_claim

FOR EACH ROW

BEGIN

IF NEW.status = 'APPROVED' THEN

UPDATE tbl\_insurance

SET status = 'CLAIMED'

WHERE insurance\_id = NEW.insurance\_id;

END IF;  
  
DROP TRIGGER IF EXISTS update\_insurance\_status\_rejected;

CREATE TRIGGER update\_insurance\_status\_rejected

AFTER UPDATE ON tbl\_claim

FOR EACH ROW

BEGIN

IF NEW.status = 'REJECTED' THEN

UPDATE tbl\_insurance

SET status = 'REJECTED'

WHERE insurance\_id = NEW.insurance\_id;

END IF;

END // DELIMITER;

* 1. **Code for Views**

CREATE VIEW vw\_cars\_owners AS

SELECT c.car\_id, c.model, c.color, c.brand, c.year, o.fname, o.lname

FROM tbl\_car c

INNER JOIN tbl\_owner o

ON c.owner\_id = o.owner\_id;

CREATE VIEW vw\_accidents\_cars AS

SELECT a.accident\_id, a.cause, a.date, a.location, c.model, c.color, c.brand, c.year

FROM tbl\_accident a

INNER JOIN tbl\_car c

ON a.car\_id = c.car\_id;

CREATE VIEW vw\_claims\_accidents AS

SELECT cl.claim\_id, cl.status, cl.filed\_by, cl.insurance\_id, cl.accident\_id, a.cause, a.date, a.location

FROM tbl\_claim cl

INNER JOIN tbl\_accident a

ON cl.accident\_id = a.accident\_id;

CREATE VIEW vw\_total\_claims AS

SELECT i.insurance\_id, i.agent\_name, COUNT(c.claim\_id) AS total\_claims

FROM tbl\_insurance i

INNER JOIN tbl\_claim c

ON i.insurance\_id = c.insurance\_id

GROUP BY i.insurance\_id;

CREATE VIEW vw\_successful\_claims AS

SELECT i.agent\_name, COUNT(

CASE WHEN i.status = 'CLAIMED' THEN 1 ELSE NULL END

) AS total\_claims

FROM tbl\_insurance i

GROUP BY i.agent\_name;

1. **Python App**

import mysql.connector

import os

from dotenv import load\_dotenv

import matplotlib.pyplot as plt

# functions

def \_exit(\_):

print("Exiting... ")

exit()

def insertOwner(cursor):

fname = input("Enter first name: ")

lname = input("Enter last name: ")

sql = "CALL insert\_owner(%s, %s)"

val = (fname, lname)

cursor.execute(sql, val)

print("Owner inserted")

def insertOwnerPhoneNumber(cursor):

phone = input("Enter phone number: ")

owner\_id = input("Enter owner id: ")

sql = "CALL insert\_phone(%s, %s)"

val = (phone, owner\_id)

cursor.execute(sql, val)

print("Owner phone number inserted")

def deleteOwnerPhoneNumber(cursor):

phone = input("Enter phone number: ")

owner\_id = input("Enter owner id: ")

sql = "CALL delete\_phone(%s, %s)"

val = (phone, owner\_id)

cursor.execute(sql, val)

print("Owner phone number deleted")

def insertCar(cursor):

model = input("Enter car model: ")

color = input("Enter car color: ")

brand = input("Enter car brand: ")

year = input("Enter car year: ")

owner\_id = input("Enter owner id: ")

sql = "CALL insert\_car(%s, %s, %s, %s, %s)"

val = (model, color, brand, year, owner\_id)

cursor.execute(sql, val)

print("Car inserted")

def createInsurance(cursor):

valid\_till = input("Enter valid till date (YYYY-MM-DD HH:MM:SS): ")

amount = input("Enter amount: ")

agent\_name = input("Enter agent name: ")

paid\_by = input("Enter paid by: ")

sql = "CALL create\_insurance(%s, %s, %s, %s)"

val = (valid\_till, amount, agent\_name, paid\_by)

cursor.execute(sql, val)

print("Insurance created")

def reportAccident(cursor):

cause = input("Enter cause: ")

date = input("Enter date (YYYY-MM-DD HH:MM:SS): ")

location = input("Enter location: ")

car\_id = input("Enter car id: ")

sql = "CALL report\_accident(%s, %s, %s, %s)"

val = (cause, date, location, car\_id)

cursor.execute(sql, val)

print("Accident reported")

def fileClaim(cursor):

filed\_by = input("Enter filed by: ")

insurance\_id = input("Enter insurance id: ")

accident\_id = input("Enter accident id: ")

sql = "CALL file\_claim(%s, %s, %s)"

val = (filed\_by, insurance\_id, accident\_id)

cursor.execute(sql, val)

print("Claim filed")

def approveClaim(cursor):

claim\_id = input("Enter claim id: ")

sql = "CALL approve\_claim(%s)"

val = (claim\_id,)

cursor.execute(sql, val)

print("Claim approved")

def rejectClaim(cursor):

claim\_id = input("Enter claim id: ")

sql = "CALL reject\_claim(%s)"

val = (claim\_id,)

cursor.execute(sql, val)

def generateCarOwnerReport(cursor):

sql = "SELECT \* FROM vw\_cars\_owners"

cursor.execute(sql)

result = cursor.fetchall()

print(result)

def generateCarAccidentReport(cursor):

sql = "SELECT \* FROM vw\_accidents\_cars"

cursor.execute(sql)

result = cursor.fetchall()

print(result)

def generateClaimAccidentReport(cursor):

sql = "SELECT \* FROM vw\_claims\_accidents"

cursor.execute(sql)

result = cursor.fetchall()

print(result)

def generateSuccessClaimsReport(cursor):

sql = "SELECT \* FROM vw\_total\_claims"

cursor.execute(sql)

result = cursor.fetchall()

print(result)

def generateSuccessClaimsByAgentReport(cursor):

sql = "SELECT \* FROM vw\_successful\_claims"

cursor.execute(sql)

result = cursor.fetchall()

print(result)

# app

menus = [

{"label": "Insert Owner", "action": insertOwner},

{"label": "Insert Owner Phone Number", "action": insertOwnerPhoneNumber},

{"label": "Delete Owner Phone Number", "action": deleteOwnerPhoneNumber},

{"label": "Insert Car", "action": insertCar},

{"label": "Create Insurance", "action": createInsurance},

{"label": "Report accident", "action": reportAccident},

{"label": "File a Claim", "action": fileClaim},

{"label": "Approve Claim", "action": approveClaim},

{"label": "Reject Claim", "action": rejectClaim},

{"label": "Generate Car Owner Report", "action": generateCarOwnerReport},

{"label": "Generate Car Accident Report", "action": generateCarAccidentReport},

{"label": "Generate Claim Accident Report", "action": generateClaimAccidentReport},

{"label": "Generate Success Claims Report", "action": generateSuccessClaimsReport},

{"label": "Generate Success Claims by Agent Report", "action": generateSuccessClaimsByAgentReport},

{"label": "Exit", "action": \_exit},

]

def menu():

print ("Welcome to the Car Insurance App. Please select an option:")

for i in range(len(menus)):

print(str(i+1) + ". " + menus[i]["label"])

option = input("> ")

return int(option)

def back():

print("Press any key to go back")

input()

return

def main():

load\_dotenv()

# Connect to database

db\_host = os.environ.get('DB\_HOST')

db\_username = os.environ.get('DB\_USERNAME')

db\_password = os.environ.get('DB\_PASSWORD')

db\_name = os.environ.get('DB\_NAME')

try:

db = mysql.connector.connect(

host=db\_host,

user=db\_username,

password=db\_password,

database=db\_name

)

cursor = db.cursor()

print("DATABASE CONNECTED " + str(db.is\_connected()))

while(True):

try:

# clear screen and show menu

os.system('cls' if os.name == 'nt' else 'clear')

option = menu()

# see if option is valid

if len(menus) < option or option < 1:

print("Invalid option")

back()

continue

selected\_menu = menus[option-1]

#execute the action

selected\_menu["action"](cursor)

db.commit()

back()

except Exception as e:

print("Error: " + str(e))

back()

continue

except Exception as e:

print("Error: " + str(e))

main(

## Conclusion

In conclusion, this project represents a successful integration of database management, stored procedures, triggers, and a user interface to create a functional car Users can easily perform essential tasks, such as inputting owner and vehicle details, reporting accidents, submitting claims and generating informative reports. Additionally, it showcases the practical application of database normalization principles to ensure data integrity reduction of redundancy. This project acts as a solid foundation that can be further enhanced and customized to meet the specific needs of insurance providers.

## Result

|  |  |
| --- | --- |
| 1. Insert Owner | 1. Insert Owner Phone |
| 1. Delete Owner Phone | 1. Create Insurance |

|  |  |
| --- | --- |
| 1. Insert Car | 1. Report Accident |
| 1. File a Claim | 1. Approve Claim |

|  |  |  |
| --- | --- | --- |
| 1. Reject Claim | | 1. Car Owner Report |
| 1. Car Accident Report | | |
| 1. Total Claim Report | 1. Success Claim Report by Agent | |