USE CASE STUDY REPORT

Member **Names:** Chia-Han Chiang and Urjasvit Sinha

Executive Summary

In the project "Insurance database management system" we have developed a relational database to manage the insertion, storage and efficient retrieval of the data of the insurance company. We first gathered the domain knowledge of working of the insurance company and the challenges faced by them. Then we tried to implement part of the problem by listing the required information and assumed a health insurance product as a business case for our project. We made an Enhanced Entity Relation Diagram of the insurance product and made a Class diagram using the gathered information. We then used both of the diagrams to construct a relational model before implementing it in MySQL. Since there was no readily available dataset in the public domain, we created our own dataset which resembled the real-world scenario. The project used DML and DDL commands present in RDBMS. Using the MySQL Workbench we created several tables and imported the dataset into MySQL. We developed various real-world situations relevant to our business case and retrieved the answers to them using SQL queries. Along with that we managed our part of the database in the NoSQL (MongoDB) environment and used MapReduce and aggregate functions. Also, we accessed our database using R and made analysis of the retrieved data.

I. Introduction

Liberty life is one of fastest growing insurance companies in the USA, whose main product is health insurance. The demand for it is growing exponentially. Because of their growth they are trying to manage and separate the product line so that the demand can be met easily, and gain profit out of the product. They want to make a relational database management system which will help the company solve data management and process management issues. Here is the information the insurance company wants to manage:

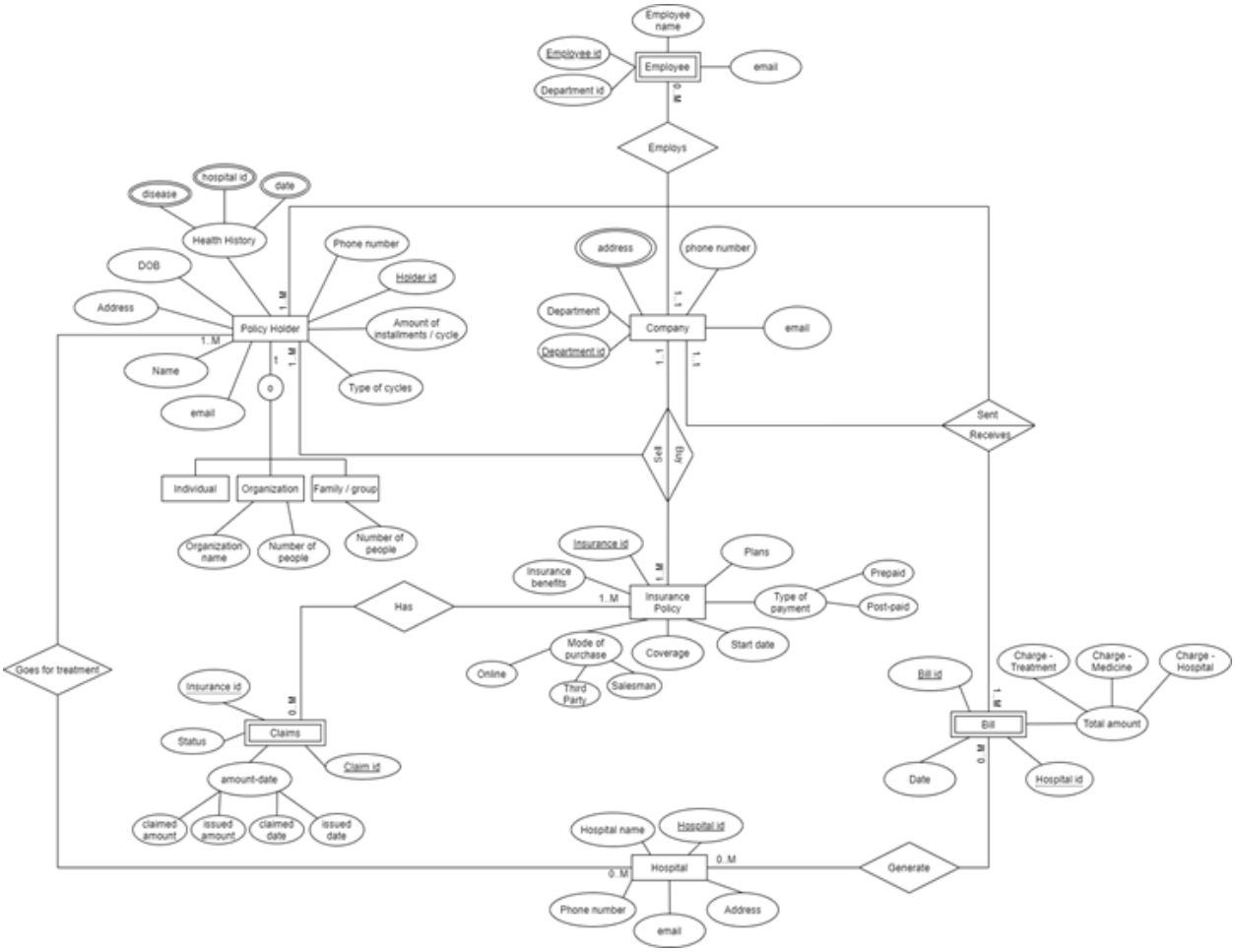
- The company has multiple addresses, department, department id, customer care number, email address.
- The company sells insurance policies to buyers. Each policyholder(buyer) has name, address, phone number, DOB, email address, amount of instalment/cycles(money paid to the company each cycle), type of cycles(how the insurance instalments get paid to the company), health history, and holder id through which it gets identified.
- Along with that holder can be categorized into an organization, an individual or a family/group. A holder can be in multiple categories at the same time.
- For an organization holder, it must specify the number of people (employee), and the organization's name.
- Each group/family holder requires a number of people.

- The health history has the name of the disease, hospital id, date which is stored inside the database.
- The company employs people. They have employee id, employee name, and email.
- A holder can buy insurance through the company's online portal, third party or company salesman.
- Every insurance has an insurance id, coverage (maximum amount that the insurance will cover), plans(the level of the insurance), type of payment to the hospital (how the insurance money gets paid to the hospital: prepaid, post-paid), start date, mode of purchase (online, third party, or company salesman), insurance benefits(what will the plan covers).
- A claim is made on insurance, it has claim id, an amount (how much is claimed by the holder and how much is provided by the company: issued and claimed), insurance id, issued date, status of the claim.
- The holder goes to the hospital for treatment. Each hospital has hospital id, name, phone number, email, address.
- The hospital when diagnoses/runs tests/treats the patient, the bill is generated along with bill id, total amount, date, hospital charges, medicine charges, and treatment charges. The bill is sent by the patient to the insurance company for claims.

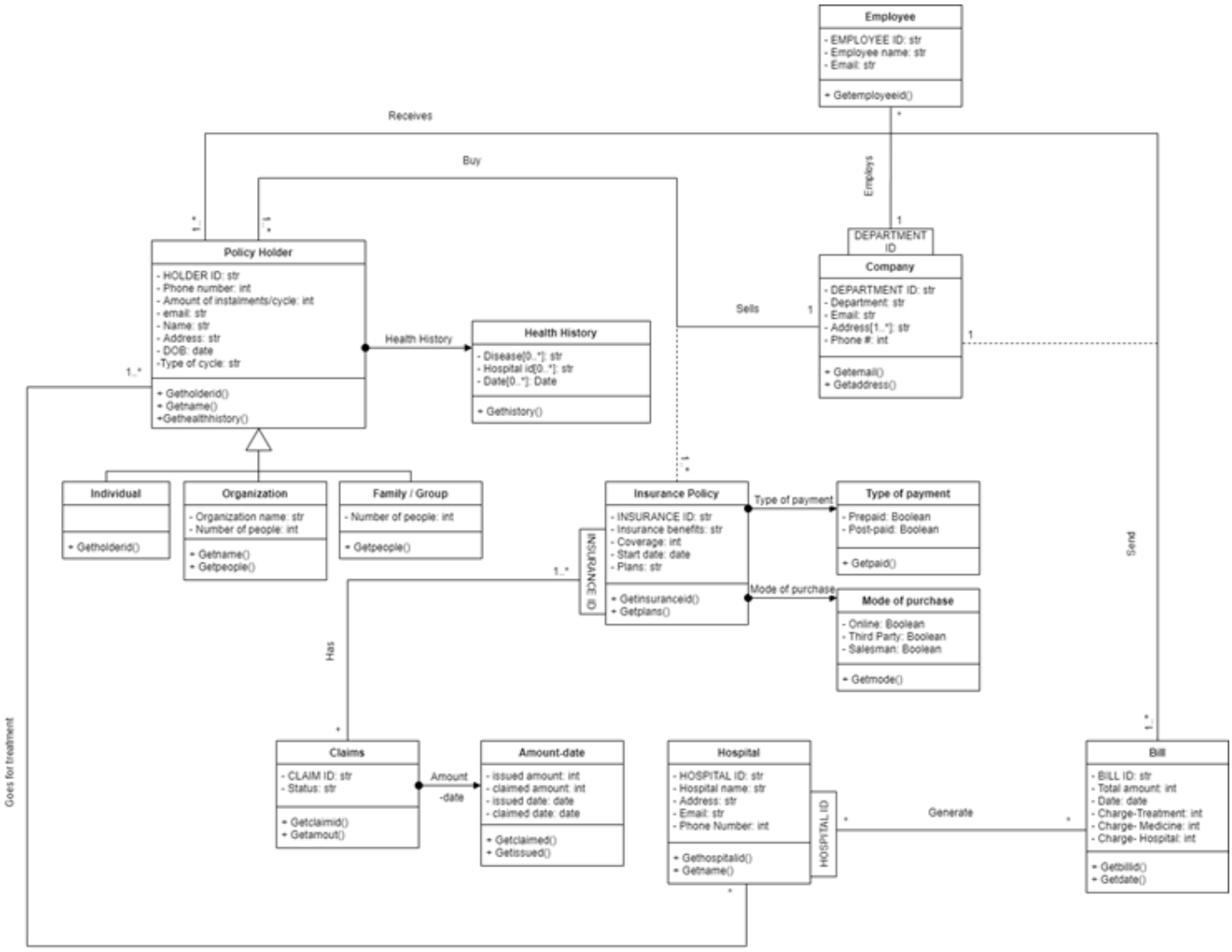
The purpose of this database is to maintain data of multiple entities which are mentioned above across the enterprise to restrict the accessibility of the resources. The RDBMS will provide permissions to the users according to the nature of restrictions they are under.

II. Conceptual Data Modeling

Enhanced Entity Relationship Model



Unified Modeling Language- Class Diagram



III. Mapping Conceptual Model to Relational Model

POLICY HOLDER: { **Holder id** | Phone number | DOB | Address | Name | Email | Insurance id | Bill id | Hospital id | Department id}
POLICYHOLDER-CYCLE: { **Holder id** | Amount of installment / cycle | Type of Cycle }
HOLDER_HEALTH_HISTORY: { **Buyer id** | Disease | hospital id | date }
INDIVIDUAL: { **Holder id** }
ORGANIZATION: { **Holder id** | Organization name | Number of people }
FAMILY / GROUP: { **Holder id** | Number of people }

EMPLOYEE: { **Employee id** | ***Department id*** | Employee name | Email }

COMPANY: { **Department id** | Department }
DEPARTMENT DETAILS: { **Department**| Address | Phone number | Email }

INSURANCE: { **Insurance id** | Plans | Start date | Mode of purchase | Type of payment}
INSURANCE-PLAN: { Insurance benefit | **Plans** | Coverage }

CLAIMS: { **Claim id** | ***Insurance id*** | Status }
CLAIMS DETAIL: { **Claim id** | Issued Amount | Claimed Amount | Issued date | Claimed date }

HOSPITAL: { **Hospital id** | Hospital name | Phone number | Email | Address }

BILL: { **Bill id** | ***Hospital id*** | Date }
BILL-TOTAL-AMOUNT: { **Bill id** | Treatment charges | Medicine charges | Hospital charges }

Buyer-hospital: { **Buyer id** | **Hospital id** }
Insurance-claims: { **Claim id** | **Insurance id** }
Hospital-bill: { **Hospital id** | **Bill id** }

*****BOLD(Primary) and *BOLD* (Foreign key) is NOT NULL**

IV. Implementation of Relation Model via MySQL and NoSQL

TABLE CREATION AND INSERTION (MYSQL)

Question: Write a query to create a holder health history table having holder id as primary key, and other attributes like health history of disease, the id of hospital and the last date of disease. Also, perform data insertion into the table.

Answer:

```
CREATE TABLE policy_holder_health_history (
  Holder_id INT NOT NULL PRIMARY KEY,
  Health_history_disease VARCHAR(255),
  Health_history_hospital_id INT,
  Health_history_date VARCHAR(255)
);
INSERT INTO policy_holder_health_history
  (Holder_id,Health_history_disease,Health_history_hospital_id,Health_history_date)
```

VALUES

(1,'heart attack',4,'2019-08-06'),

(2,'malaria',1,'2004-12-09'),

(3,'cancer',5,'2017-08-15'),

(4,'diabetes',2,'2014-08-15'),

(5,'fracture',2,'2005-08-12');

```
1 •  SELECT * FROM insurance.policy_holder_health_history;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap C

Holder_id	Health_history_disease	Health_history_hospital_id	Health_history_date
1	heart attack	4	2019-08-06
2	malaria	1	2004-12-09
3	cancer	5	2017-08-15
4	diabetes	2	2014-08-15
5	fracture	2	2005-08-12

DATA RETRIEVAL USING SUB-QUERIES (MYSQL)

Question: Display the policy holder id, name and amount of installment paid every cycle of an individual who pays the highest installment amount yearly from the insurance database.

Answer:

```
1   #Display the holder id,name and the amount of installment paid every cylce of an individual who pays highest amount
2   # of installment yearly from the insurance database
3
4 •  select c.Holder_id, p.name,c.Amount_of_installmentcycle
5   from policyholder p, policyholdercycle c
6   where p.holderid=c.holder_id and c.Amount_of_installmentcycle in (
7   select max(amount_of_installmentcycle)
8   from policyholdercycle)
9   and c.Type_of_cycle='yearly';
10
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Holder_id	name	Amount_of_installmentcycle
55	Ilysa Keneleyside	9864

QUERY USING AGGREGATE FUNCTIONS (MYSQL)

Question: Display maximum claim amount given to an individual and the average claim amount given by the insurance company per individual.

Answer:

```
10
11  #display maximum claim amount given to an individual and the average claim amount passed by the insurance company per individual
12  select max(amount_claimed) as MaximumClaimGiven,avg(amount_claimed) as AverageAmountSpent
13  from claims
14  where Claim_status='Given';
15
16
17
18
19
20
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

MaximumClaimGiven	AverageAmountSpent
25342	12161.1250

DATA RETRIEVAL USING JOINS ON TWO TABLES (MYSQL)

Question: Display the Policy holder id, name, age, insurance id and health history of the individuals born after 1900.

Answer:

```
16  #Display the Policy holder id, name, age, insurance id and health history of the individuals born after 1900
17  select p.holder_id, h.name, p.health_history_disease, h.Insurance_id,FLOOR(DATEDIFF('2020-04-16',h.dob) / 365.25) as Age
18  from policy_holder_health_history p inner join policyholder h on p.Holder_id=h.Holderid
19  where h.DOB>'1990';
20
21
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

holder_id	name	health_history_disease	Insurance_id	Age
1	Cobby Audus	heart attack	681	15
2	Roderigo Karpenko	malaria	258	23
3	Jules Edelheit	cancer	779	22
5	Tobey Giacomello	fracture	448	26
6	Bell Sweatland	dengue	625	27
10	Granny Madre	cancer	799	16
12	Rabi Hargreves	heart attack	721	27
14	Tammie Richardin	malaria	487	18
22	Kelley Pitway	dengue	227	17
32	Aggi Olanda	cancer	604	20
36	Anneliese Tremain	heart attack	141	26
37	Quinn Arkil	dengue	651	20
38	Caresse Asee	heart attack	490	25
39	Teresita Hagan	malaria	539	15
41	Teddie Rountree	heart attack	924	27
45	Ingmar Antyukhin	dengue	166	26
48	Marguerite MacSke...	diabetes	848	24
51	Leeanne Thieme	malaria	133	16

VIEW CREATION USING AGGREGATE FUNCTIONS, PERFORMING MULTIPLE JOINS ON TABLES AND DATA RETRIEVAL USING GROUP BY (MYSQL)

Question: Create a view and display details of policy holders (id,name,previous health history,age) who paid maximum money to the company for every type of cycle and their difference from average amount. Also, print the minimum of each cycle.

Answer:

```
#Create a view and display details of policy holders(id,name,previous health history,age)
#who paid maximum money to the company for every type of cycle and their difference from average amount. Also, print min of each cycle.
create view v as(
select p.holder_id, h.name, p.health_history_disease, h.Insurance_id,
FLOOR(DATEDIFF('2020-04-16',h.dob) / 365.25) as Age,
c.Amount_of_installmentcycle, c.Type_of_cycle
from policy_holder_health_history p inner join policyholder h on p.Holder_id=h.Holderid
inner join policyholdercycle c on c.Holder_id=h.Holderid
);
select holder_id,name,age,health_history_disease as previousDisease,
max(amount_of_installmentcycle) as Amount_Paid_by_holder, type_of_cycle,
min(amount_of_installmentcycle) as Cycle_Minimum_Amount,
avg(amount_of_installmentcycle) as Cycle_Average_Amount,
max(amount_of_installmentcycle)-avg(amount_of_installmentcycle) as Diff_from_average
from v
group by type_of_cycle;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

holder_id	name	Age	previousDisease	Amount_Paid_by_holder	Type_of_cycle	Cycle_Minimum_Amount	Cycle_Average_Amount	Diff_from_average
1	Cobby Audus	15	heart attack	9860	monthly	450	4920.7143	4939.2857
2	Roderigo Karpenko	23	malaria	9864	yearly	437	5137.1429	4726.8571
7	Derrick Gimbart	41	fracture	9277	quaterly	140	4700.0000	4577.0000

For **NoSQL**, we used 2 tables in MongoDB, one is “claim”, which recorded all the claims that policyholders requested for. Second one is “policy holder”, this records all the details of each holder.

TABLE CREATION AND INSERTION (NOSQL)

Question: Write a query to create and insert data in a claim table having claim id as primary key, and other attributes like insurance id of the policy holder, claim status and total amount in the claim.

Answer:

```
db.claim.insert({
Claimid: 1 ,
Insurance_id: 233,
Claim_status: "Claimed",
Amount_claimed: 28594
})
db.claim.insert({
Claimid: 2 ,
Insurance_id: 206,
Claim_status: "Given",
Amount_claimed: 20220
})
```

```
Insurance_id: 221,
Claim_status: "Claimed",
Amount_claimed: 14333
})

db.claim.insert({
Claimid: 14 ,
Insurance_id: 133,
Claim_status: "Given",
Amount_claimed:18877
})

db.claim.insert({
Claimid: 15 ,
Insurance_id: 229,
Claim_status: "Pending",
Amount_claimed: 9556
})

db.claim.find();
```

Run

Result

```
{ "_id" : ObjectId("5e9d92b001fa3db8c4533d9e"), "Claimid" : 1, "Insurance_id" : 233, "Claim_status" : "C
{ "_id" : ObjectId("5e9d92b001fa3db8c4533d9f"), "Claimid" : 2, "Insurance_id" : 206, "Claim_status" : "G
{ "_id" : ObjectId("5e9d92b001fa3db8c4533da0"), "Claimid" : 3, "Insurance_id" : 133, "Claim_status" : "C
{ "_id" : ObjectId("5e9d92b001fa3db8c4533da1"), "Claimid" : 4, "Insurance_id" : 141, "Claim_status" : "P
{ "_id" : ObjectId("5e9d92b001fa3db8c4533da2"), "Claimid" : 5, "Insurance_id" : 152, "Claim_status" : "P
```

BASIC QUERY IN MONGODB

Question: List out all the policyholders that never received the bill.

Answer:

```
db.policyholder.find({ "Bill_id": null });
```

Run

Result

```
"Holderid" : 4, "Name" : "kelly", "Address" : "24 broke ave", "Email" : "kelly@moneycont.com", "DOB" :
"Holderid" : 5, "Name" : "mandeep", "Address" : "12 leisure ave", "Email" : "mandeep@gmail.com", "DOB"
"Holderid" : 10, "Name" : "grey", "Address" : "9 mission street", "Email" : "grey@gmail.com", "DOB" : "
```

AGGREGATE PIPELINE

Question: Write a query to get the highest amount in the given claim using an aggregate pipeline in "claim" collection

Answer:

```
db.claim.aggregate([{
    $match:{Claim_status:"Given"}
},{
    $group:{
        _id: {Claim_id:"$Claimid"},
        Amount:{$push:"$Amount_claimed"}
    }
},{
    $sort:{Amount:-1}
},{
    $limit:1
}])
```

Run

Result

```
{ "_id" : { "Claim_id" : 2 }, "Amount" : [ 20220 ] }
```

MAP-REDUCE PIPELINE

Question: Write a query to find anyone who submitted more than 1 claim using a map-reduce pipeline in "claim" collection

Answer:

```
db.claim.mapReduce(
    function(){
        emit(this.Insurance_id, 1);
        },
    function(key, value)
        {return(Array.sum(value))},
        {out:"insuranceid_counts"}
    ).find({"value":{$gt:1}});
```

Run

Result

```
{ "_id" : 133, "value" : 2 }
{ "_id" : 206, "value" : 2 }
```

Question: Write a query to count the number of employee in each department using a map-reduce pipeline in “policyholder” collection

Answer:

```
db.policyholder.mapReduce(
    function(){
        emit(this.Dept_id, 1);
        },
    function(key, value)
        {return(Array.sum(value))},
        {out:"dept_counts"}
    ).find();
```

Run

Result

```
{ "_id" : null, "value" : 3 }
{ "_id" : 1, "value" : 7 }
{ "_id" : 4, "value" : 5 }
```

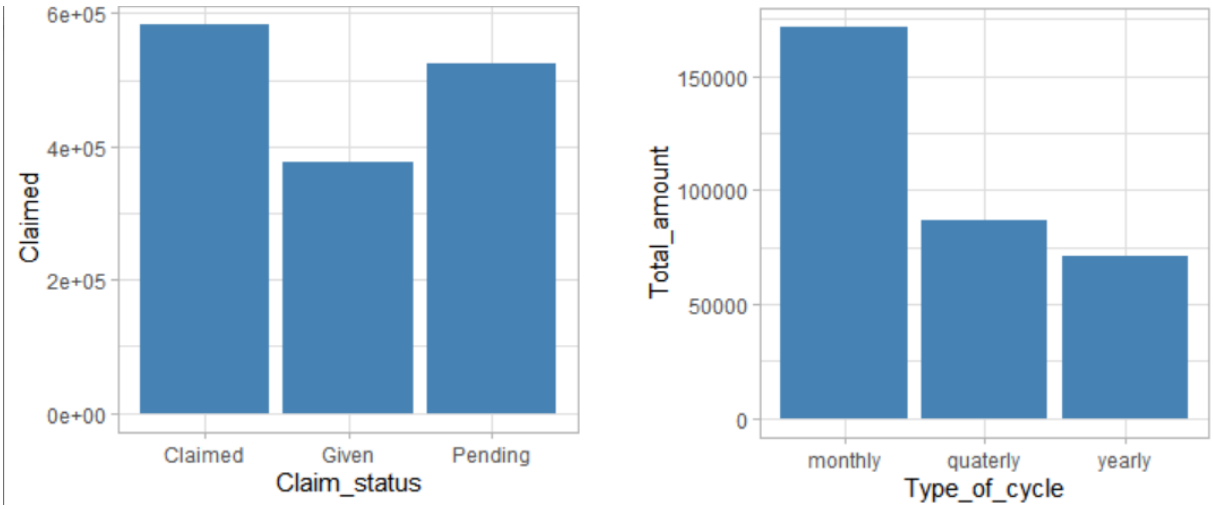
*null = the policy holder is not an employee in this company

V. Database Access via R

In order to demonstrate the MySQL database access via R, create a database connection object present in the RMySQL library. Now that the connection is established list the tables and fields in the connected database. The queries can be run using the dbSendQuery(), here we used this function to get claims and policyholdercycle tables and stored the results in rs and rs1 objects. To access the results in R we used fetch() which saves the results of the query as data frame objects. Using the data frame object, we made a histogram of claimed status and amount claimed from the claims table and another histogram of type of cycle and amount of installment per cycle.

```r
library(DBI)
library(RMySQL)
library(dbConnect)

#connecting to MySQL database
mydb <- dbConnect(MySQL(), user='dmaproject', password='password', dbname='insurance', host='localhost')

#Listing tables in Insurance database
dbListTables(mydb)

#retriving data from claims table
rs <- dbSendQuery(mydb,"select * from claims")
data <- fetch(rs)

#retriving data from claims table
rs1 <- dbSendQuery(mydb,"select * from policyholdercycle")
data1 <- fetch(rs1)

#plotting histograms of fetched data
temp= data %>% group_by(Claim_status) %>% summarise(Claimed= sum(Amount_claimed))
ggplot(temp)+ geom_bar(aes(Claim_status,Claimed), stat = "identity",fill="steelblue") + theme_light()

temp2= data1 %>% group_by(Type_of_cycle) %>% summarise(Total_amount= sum(Amount_of_installmentcycle))
ggplot(temp2)+ geom_bar(aes(Type_of_cycle,Total_amount), stat = "identity",fill="steelblue") + theme_light()
```

Screenshot of the MySQL implementation in R



Screenshots of the histogram made from the fetched data

VI. Summary and recommendation

The project was successfully implemented with all the conditions and constraints of the business case. We were able to replicate and implement the business model in SQL as well as the NoSQL environment. We also accessed the MySQL database in R and did several visualizations for making analysis. We recommend using more comprehensive structures which will resemble more towards reality and then include it in the business case and study the effectiveness of our database design.