Insurance-Management-System-SQL-based-Database-Design-and-Analysis

Introduction:

Insurance is one of the fields which is gaining a huge importance and has become one of the important components in our lives. As the number of people who are taking the insurance is increasing day by day the number of insurance companies are also increasing drastically. This increase in scale of industry, leads to a proper database management in order to store and perform the analysis on the data to draw the better insights to get good profits.

Database Design:

Our main motive is to build a database for an 'Insurance Company', which provides insurance service to 'Health', 'Car' and 'Home'.

There are multiple branches of company located in the various places across the country and each branch have a set of agents, whose responsibility is to interact with customers and increase the number of policies. Each customer will have one agent, where as one agent can deal with many customers. Each agent is given a unique ID, to identify them and along with ID, his personal details like name, phone and email id are also collected.

Company offers various policies and irrespective of the type of policy, there are few details which are in common like start date, end date, tenure, policy number, premium and coverage for each policy.

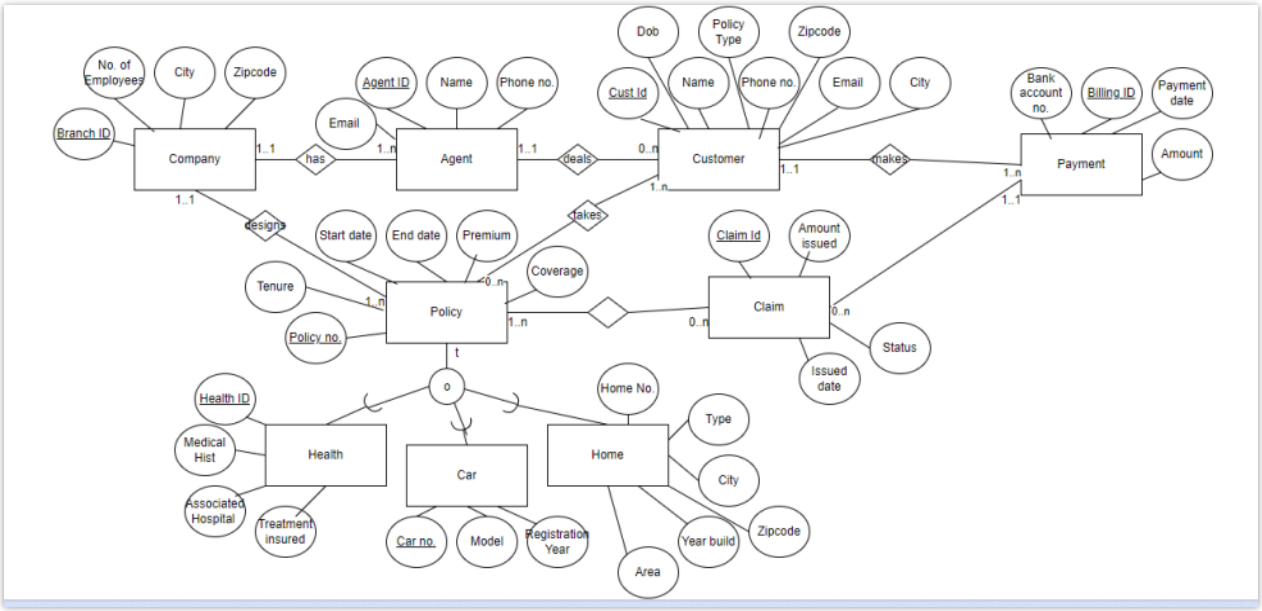
There are few details which are in specific to the type of policy, like for health insurance we must collect the customer's previous health records, hospitals and treatments that are associated to the insurance policy. When it comes to car insurance, we must store details of the car like car's number, model and year of registration. Home insurance includes details like address, area (in sq.ft) house number.

Customers can register into one or multiple policies and customer details like name, email id, mobile number, address, date of birth, policy type are stored and along with the personal details, each customer is given a unique id to identify them.

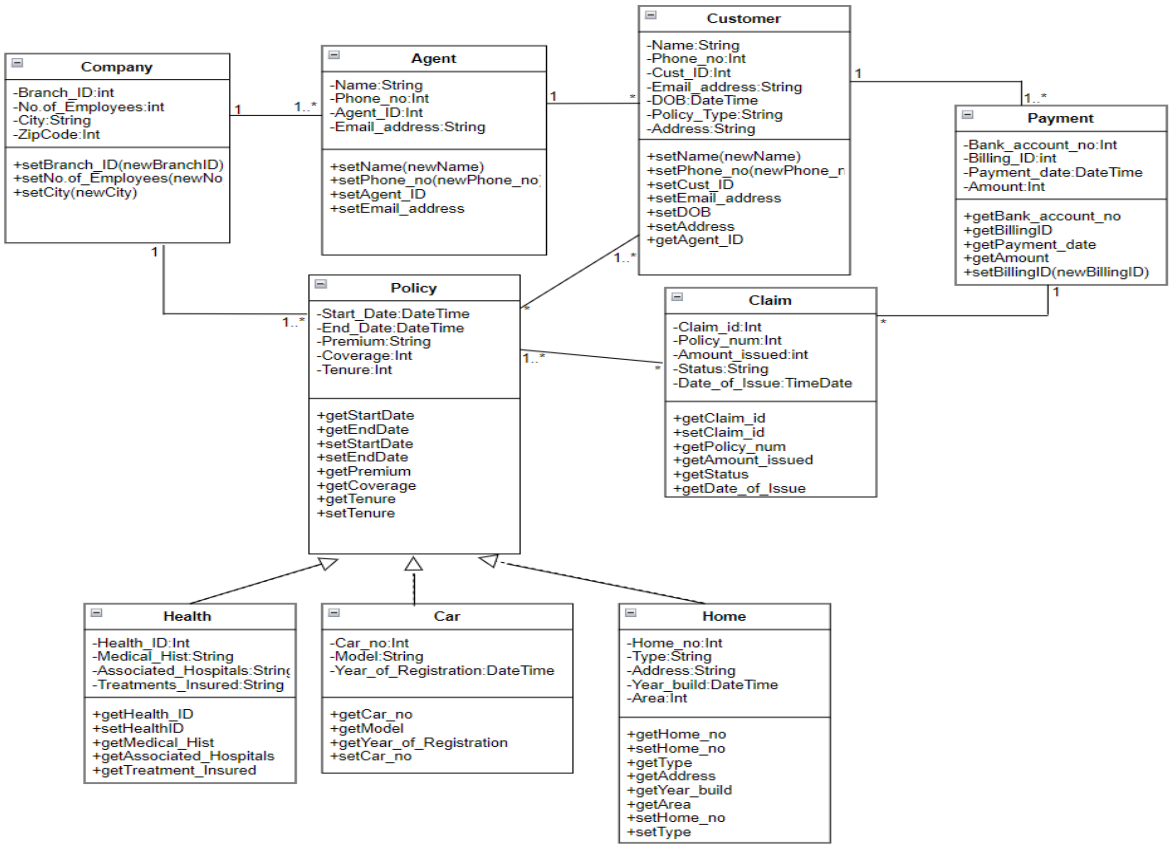
There are two types of payments involved in this. First one is the premium which customers pay to the company and second is the claim amount received by the customers from the company.

Paying premium is mandatory, but all customers need not get the claim amount from the company.

EER Diagram:



UML Diagram:



Relational Model:

Company (Branch_ID, No_of_Employees, City,Zipcode)

Agent(Agent_ID, Email, Name , Phone_no, *Branch_ID*)

- Branch_ID refers to Branch_ID in company table.

Customer (Customer_ID, DoB, Name, Phone_no, Email, Policy_type, Address, *Agent_ID*)

- Agent_ID refers to Agent_ID in agent table.

Payment (Bank_acc_no , Billing_ID, Payment_date, Amount, *Customer_ID*)

- Customer_ID refers to Customer_ID in customer table.

Policy (Policy_no, Tenure, Start_date, End_date, Premium, Coverage, *Branch_ID*)

- Branch_ID refers to Branch_ID in company table

Claim(Claim_ID, Amount_issued, Status, Issued_date, *Billing_ID*)

- Billing_ID refers to Billing_ID in payment table

Health (Health_ID, Medical_history, Associated_hospital, Treatmen_insured, *Policy_no*)

- Policy_no refers to Policy_no in policy table

Car (Car_no, Model, Registration_year, *Policy_no*)

- Policy_no refers to Policy_no in policy table

Home (Home_no, Type, Address, Year_built, Area, *Policy_no*)

- Policy_no refers to Policy_no in policy table

Cust_Policy (*Customer_ID*, *Policy_no*)

- Policy_no refers to Policy_no in policy table
- Customer_ID refers to Customer_ID in customer table.

Claim_Policy (*Claim_ID*, *Policy_no*)

- Policy_no refers to Policy_no in policy table
- Claim_ID refers to Claim_ID in claim table

Data Description:

The Company has total 7 branches. The company has 200 employees and around 400 customers. The 11 tables in the database are as follows: Company, Agent, Customer, Payment, Policy, Claim, Health, Car, Home, Cust_Policy, Claim_Policy.

Implementation of Relational model in MySQL and NoSQL:

a) MySQL Problem statements:

- No. of customers and agents present in a particular city:

```
#1. No. of customers and agents present in a particular city

select c.city, count(distinct c.Customer_id) as num_of_customers, count(distinct a.Agent_id) as num_of_agents
from customer c
left join agent a
on c.city=a.city
group by city
order by num_of_customers desc;
```

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

city	num_of_customers	num_of_agents
Boston	66	0
Hatfield	42	0
Nevada	32	16
Bothell	30	16
San Francisco	30	15
Seattle	28	15
Portland	26	14
Dallas	24	14
Duluth	22	17

Result 4

Output

Action Output

#	Time	Action	Message
3	07:57:07	select c.city, count(distinct c.Customer_id) as num_of_customers, count(distinct a.Agent_id) as num_of_age...	15 row(s) returned

- Details of top 5 performing agents:

```
#2. List of top 5 performing agents

select pl.Agent_id, concat(a.First_name,' ',a.Last_name) as Name ,count(distinct pl.Policy_no) as num_of_policy,
sum(p.amount) as revenue_generated_over_5_years
from payment p, customer_policy cp, policy pl, agent a
where pl.Policy_no=cp.Policy_no and a.Agent_id=pl.Agent_id and
cp.Customer_id=p.Customer_id and p.billing_id not in (
select Billing_id from claim)
group by pl.Agent_id
order by revenue_generated_over_5_years desc
limit 5;
```

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

Agent_id	Name	num_of_policy	revenue_generated_over_5_years
147	Alberto Errazuriz	8	371000
118	Guy Himuro	10	206000
146	Karen Partners	9	185000
166	Sundar Ande	6	141500
179	Charles Johnson	4	140000

- List of customers who took multiple policies:

```
36    #3. List of customers who took multiple policies
37
38    select cp.customer_id,concat(c.First_name,' ',c.Last_name) as Name, count(cp.customer_id) as num_of_policy
39    from customer_policy cp, customer c
40    where c.Customer_id=cp.Customer_id
41    group by customer_id
42    having count(cp.customer_id) >1;
43
44
```

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

customer_id	Name	num_of_policy
200	Schmitt Carine	3
210	Freyre Diego	2
217	Hashimoto Juri	2
481	Charles Smith	3
573	Robert Harris	2
595	Jamie McCoskey	2
596	Jerry Martin	2
599	Ray Jasper	2

- Annual change in number of health insurances taken (Is there any impact of Covid):

```
49    #4. Annual change in number of health insurances taken
50
51    select year(p.start_date) as policy_year, count(p.Policy_no) as Num_of_policy
52    from policy p, health h
53    where p.Policy_no=h.Policy_no
54    group by policy_year
55    order by policy_year;
56
```

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

policy_year	Num_of_policy
2018	43
2019	28
2020	56
2021	50
2022	27

- Year-wise revenue analysis:

```
59      #5. Year-wise revenue analysis
60
61  •   with yoy as (
62      select pa.payment_date as year,sum(pa.amount) as revenue
63      from  payment pa where
64      Billing_id not in
65      (select Billing_id from claim)
66      group by year
67      order by revenue desc)
68
69      SELECT year, revenue,
70      revenue - LAG(revenue) OVER ( ORDER BY year ) AS yoy_revenue_difference
71      FROM yoy;
```

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

year	revenue	yoy_revenue_difference
2018	298500	NULL
2019	663000	364500
2020	1047000	384000
2021	1454000	407000
2022	1780500	326500

- Categorization of insurances according to their premium amount and count of customers in each particular category:

  Health Insurance:

```
76      #6. Categorization of insurances according to their premium amount and count of customers in each particular category
77      #Health Insurance
78  •   select health_plan,count(Policy_no) as num_of_policy, sum(premium) as revenue_by_plan
79      from
80      ( select
81      case when premium>= 1000 and premium<2000 then 'basic plan'
82      when premium>=2000 and premium<3000 then 'silver plan'
83      when premium>=3000 and premium<4500 then 'gold plan'
84      when premium>=4500 then 'platinum plan'
85      end as health_plan,p.policy_no,p.premium
86      from policy p, health h
87      where p.Policy_no=h.Policy_no)d
88      group by health_plan
89      order by premium desc;
```

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

health_plan	num_of_policy	revenue_by_plan
platinum plan	55	260000
gold plan	41	143500
silver plan	53	106000
basic plan	55	82000

Car Insurance:

```
91    #Car Insurance
92 •  select car_plan,count(Policy_no) as num_of_policy, sum(premium) as premium
93    from
94    ( select
95    case when premium>= 1000 and premium<1500 then 'basic plan'
96    when premium>1500 and premium<=2000 then 'silver plan'
97    when premium>2000 and premium<=3000 then 'gold plan'
98    when premium>3000 then 'platinum plan'
99    end as car_plan,p.policy_no,p.premium
100   from policy p, car c
101   where p.Policy_no=c.Policy_no)d
102   group by car_plan
103   order by premium desc;
104
```

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

car_plan	num_of_policy	premium
gold plan	28	84000
platinum plan	17	68000
silver plan	27	54000
basic plan	31	31000

Home Insurance:

```
105   #Home Insurance
106 •  select home_plan,count(Policy_no) as num_of_policy, sum(premium) as premium
107   from
108   ( select
109   case when premium>= 4000 and premium<5000 then 'basic plan'
110   when premium>=5000 and premium<9000 then 'silver plan'
111   when premium>=9000 and premium<12000 then 'gold plan'
112   when premium>=12000 then 'platinum plan'
113   end as home_plan,p.policy_no,p.premium
114   from policy p, home ho
115   where p.Policy_no=ho.Policy_no)d
116   group by home_plan
117   order by premium desc;
118
```

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

home_plan	num_of_policy	premium
platinum plan	44	616000
gold plan	20	180000
silver plan	19	95000
basic plan	18	72000

- Which age group of customers showed more interest to take insurance?

```
#6. Which age group of customers showed more interest to take insurance?
select age_group,count(Policy_no) as num_of_policy
from
( select
case when TIMESTAMPDIFF(year, DoB, date(now()))>= 18 and TIMESTAMPDIFF(year, DoB, date(now()))<31 then '18-30'
when TIMESTAMPDIFF(year, DoB, date(now()))>=31 and TIMESTAMPDIFF(year, DoB, date(now()))<41 then '31-40'
when TIMESTAMPDIFF(year, DoB, date(now()))>=41 and TIMESTAMPDIFF(year, DoB, date(now()))<51 then '41-50'
when TIMESTAMPDIFF(year, DoB, date(now()))>=51 and TIMESTAMPDIFF(year, DoB, date(now()))<61 then '51-60'
when TIMESTAMPDIFF(year, DoB, date(now()))>=61 and TIMESTAMPDIFF(year, DoB, date(now()))<71 then '61-70'
when TIMESTAMPDIFF(year, DoB, date(now()))>=71 and TIMESTAMPDIFF(year, DoB, date(now()))<81 then '71-80'
when TIMESTAMPDIFF(year, DoB, date(now()))>=81 and TIMESTAMPDIFF(year, DoB, date(now()))<91 then '81-90'
end as age_group,p.policy_no,c.DoB
from policy p, customer_policy cp, customer c
where p.Policy_no=cp.Policy_no and cp.Customer_id=c.Customer_id)d
group by age_group
order by num_of_policy desc;
```

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

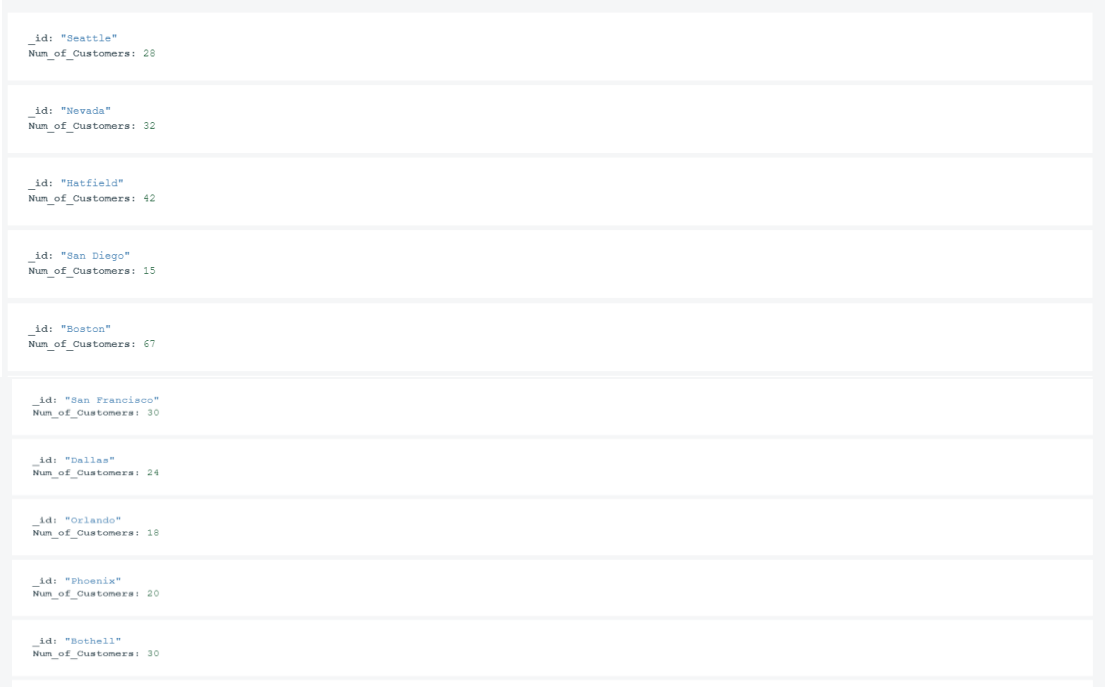
age_group	num_of_policy
18-30	158
51-60	81
41-50	59
61-70	47
71-80	31
31-40	28
81-90	5

b) NoSQL Problem statements:

The table taken for NoSQL implementation are : Customer, Policy, Home. Implentation was performed in MangoDB Compass.

- No. of customers and agents present in a particular city:

```
[{
   $project: {
      City: 1,
      Customer_ID: 1
   }}, {
   $group: {
      _id: '$City',
      Num_of_Customers: {
         $sum: 1}}
}]
```



- Details of top 5 performing agents based on most number of policies:

```
[{
 $project: {
  Policy_no: 1,
  Agent_id: 1
 }
}, {
 $group: {
  _id: '$Agent_id',
  Num_of_policy: {
   $sum: 1
  }}}, {
 $sort: {
  Num_of_policy: -1
 }}, {
 $limit: 5
}]
```

```
_id: 118
Num_of_policy: 10

_id: 146
Num_of_policy: 9

_id: 142
Num_of_policy: 8

_id: 147
Num_of_policy: 8

_id: 195
Num_of_policy: 8
```

- List of different home types along with their age and area taken for home insurance:

```
[{
 $addFields: {
  age_of_house: {
   $dateDiff: {
    startDate: '$Year_built',
    endDate: '$$NOW',
    unit: 'year'
   } } }}, {
 $project: {
  Home_type: 1,
  Home_no: 1,
  Area: 1,
  age_of_house: 1
 }}, {
 $group: {
  _id: '$Home_type',
  count: {
   $sum: 1  },
  avg_area: {
   $avg: '$Area'  },
  avg_age: {
   $avg: '$age_of_house' }}
}]
```

```
_id: "Single-Family Home"
count: 25
avg_area: 2761.92
avg_age: 12.64
```

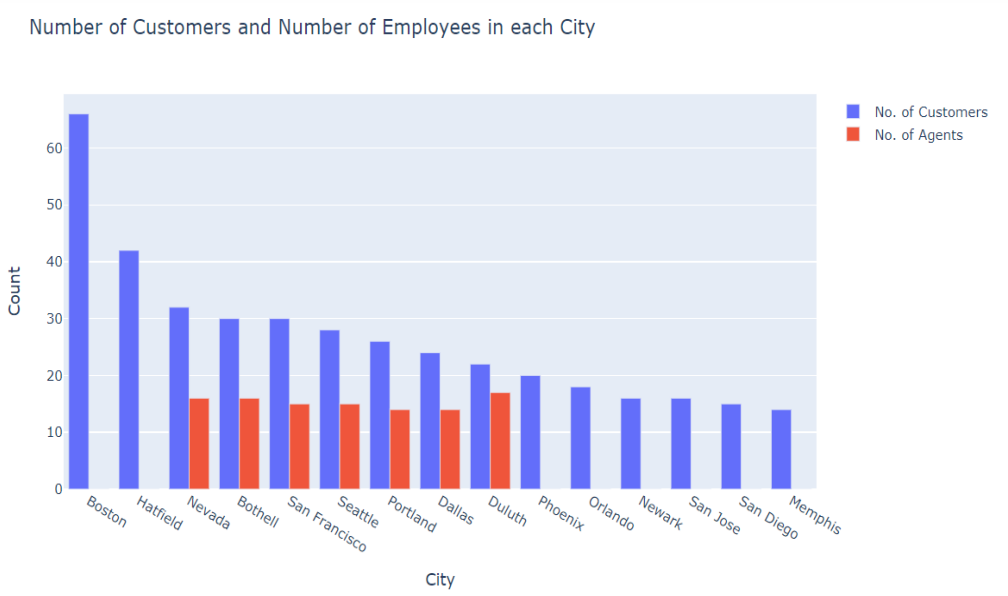
```
_id: "Multi-Family Home"
count: 24
avg_area: 3123.5833333333335
avg_age: 18.041666666666668
```

```
_id: "Townhouse"
count: 25
avg_area: 2996.32
avg_age: 16.28
```

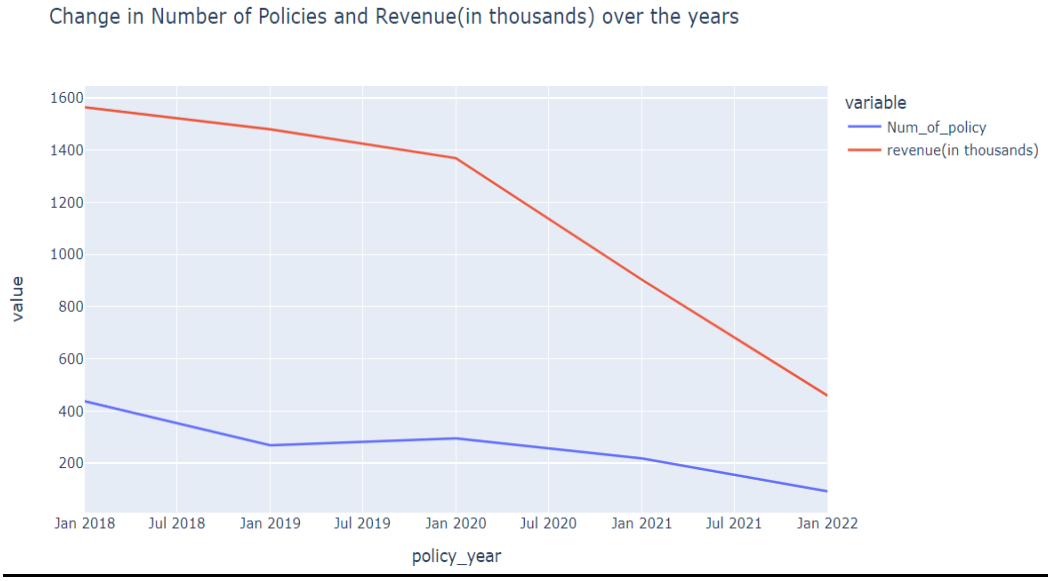
```
_id: "Condo"
count: 25
avg_area: 3043.92
avg_age: 15.56
```

Connecting database to python platform:

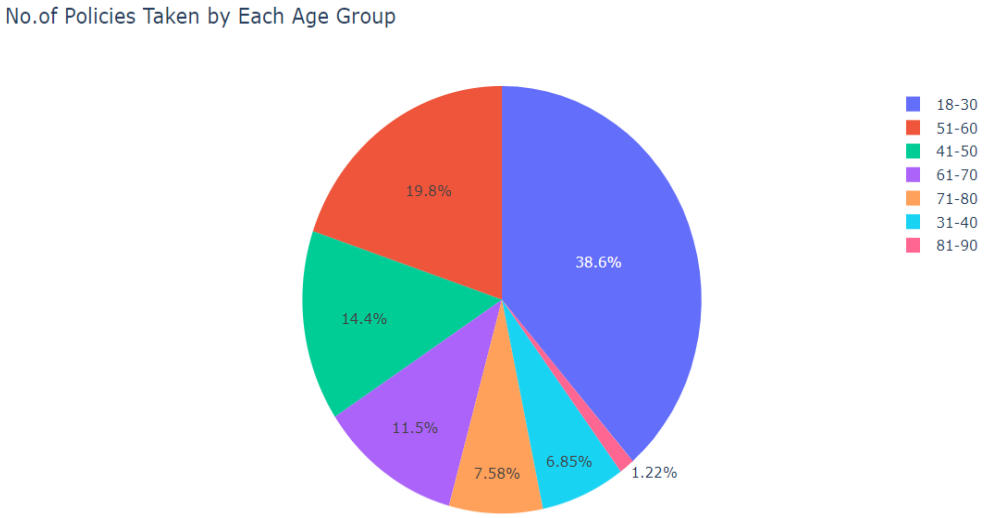
- We have connected MySQL to python using mysql.connector to access the Insurance database created.
- Visualizations have been created to analyze few of the problem statements from MySQL.
    - Visualization 1:



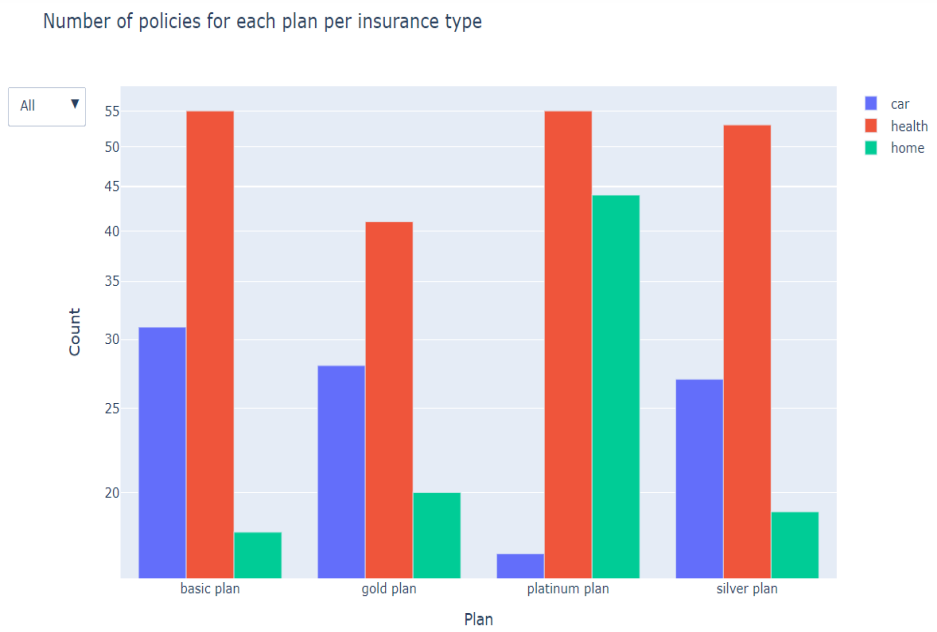
    - Visualization 2:

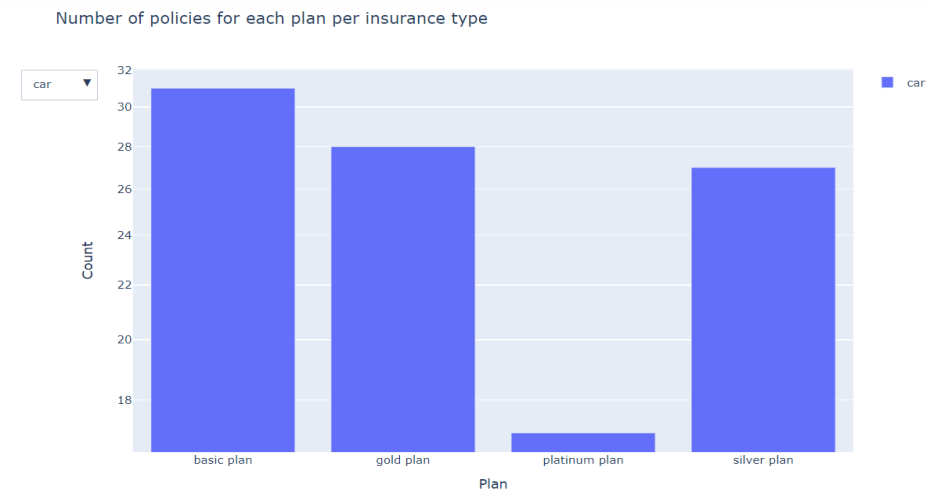


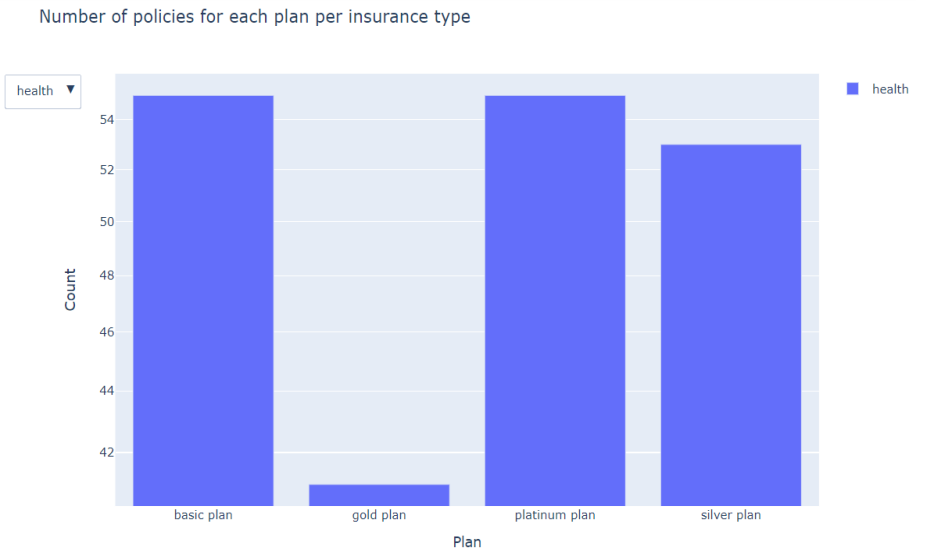
- Visualization 3:

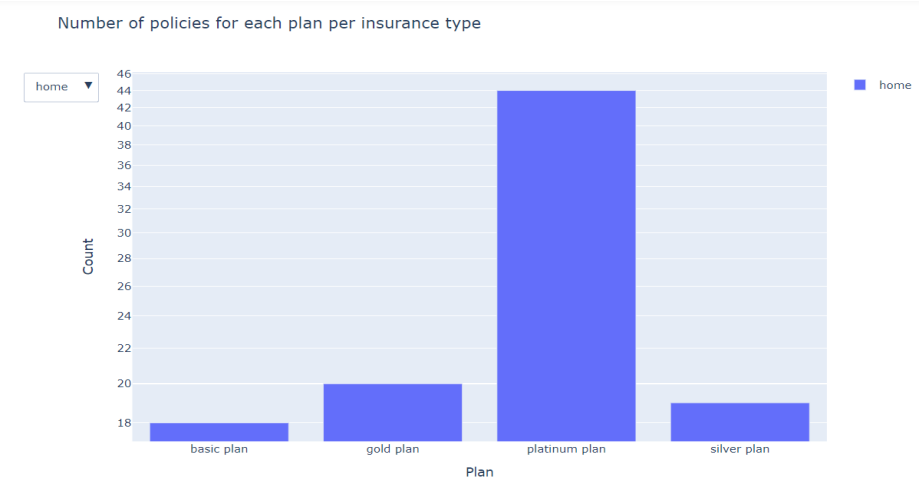


- Visualization 4: The toggle let's us choose the different plans and observe them individually.




Number of policies for each plan per insurance type
car
car
32
30
28
26
24
22
20
18
Count
basic plan
gold plan
platinum plan
silver plan
Plan


Number of policies for each plan per insurance type
health
health
54
52
50
48
46
44
42
Count
basic plan
gold plan
platinum plan
silver plan
Plan


Number of policies for each plan per insurance type
home
home
46
44
42
40
38
36
34
32
30
28
26
24
22
20
18
Count
basic plan
gold plan
platinum plan
silver plan
Plan

Features in Database:

- We have used trigger function to check the records in the payment table, when a new row is inserted.
- It checks the premium amount of the user and validates if the user is paying correct amount or not.
- We have built a script to fetch the date of birth and email of the customer from the database and sends an automated email to the customer on their birthday.
- User can choose the desired table and can enter the values into it on python platform.

Summary

- The MySQL-based Insurance Database is an industry-ready relational database that may be implemented in the insurance sector. It is designed to maintain the data of both the employees and customers without any confusions. Different insurance types have different tables, which helps to differentiate the policies and details of the respective policy taken by the customer.
- Payment table is also designed in such a way that it records both the payments that are paid by the customer and the payments customers receive from the company as part of the insurance claim.
- Above mentioned database features, helps to verify the payments made by the customers, thereby not giving any opportunity to the mismatch of the amounts.

Future scope:

- We did not include the records of the employees like, their base salary, promotion etc. We can include such columns and draw the necessary analysis from them.
- Payments table can also include the payment records of the employees, like their monthly salaries, yearly bonus etc.