**PROJECT REPORT**

**Insurance Management System**

Group Members:

Nisarg Kapkar – 18BCP069

Nisarg Dave – 18BCP070

Parth Choksi – 18BCP075

Parth Prajapati – 18BCP076

Shreedhar Bhatt – 18BCP150D

**INTRODUCTION**

The Insurance Management Systems (IMS) is a management system created to manage all aspects of running your insurance agency from chasing your policy count to operating your expiration list and everything in between. The appliance that will permit admin to feature, edit and review documents in spite of their physical location. Insurance Management System (IMS) can decrease your load of work and increase your potency, therefore permitting you time to focus on increasing production. Our goal is to eliminate redundancies by automating several of your regular tasks; scale back expenses with such tools as going paperless; and assist you properly manage your agency by providing you with the data you would like to form solid business selections.

Database designed by us contains the data for 6 types of insurance:

1. Life Insurance
2. Health Insurance
3. Vehicle Insurance
4. Travel Insurance
5. Fire Insurance
6. Theft Insurance

**OBJECTIVE**

* To handle data redundancy
* To provide faster and efficient searching of information
* To reduce time, maintenance and cost
* To assure data privacy and security

**Experiment 1:**

**DBMS (INSURANCE MANAGEMENT SYSTEM) VS FILE SYSTEM**

* **DATA REDUNDANCY:**

There are many types of insurance like life insurance, health insurance, motor insurance, travel insurance, home insurance, fire insurance etc.

Suppose that a person has taken insurance for many of the above stated policies. In such case information of the registered person (like his phone number, address etc) will appear in many files that consists records of different insurances. So this is a case of data redundancy where the same information is repeated again and again and this leads to higher storage

* **DATA INCONSISTENCY:**

Suppose that in the example stated above if the person who has payed for health insurance as well as motor insurance wishes to change his address or mobile number Than it may be possible that required change may happen in health insurance record file but not in motor insurance file This leads to data inconsistency

* **DIFFICULTY IN ACCESSING THE DATA:**

Suppose an insurance company needs to get the list of all policy holders of life insurance who live in a particular place like Gujarat. Since the designers of the original system did not anticipate this request, there is no application program in hand to meet it

* **DATA ISOLATION:**

In file system, data will be scattered among multiple files of different formats such as Excel, PPT, Word, etc Here, it would be difficult to create a program to carry out an operation

* **DATA INTEGRITY:**

The data value stored in the database must satisfy certain types of consistency constraint For example, a person whose term of insurance is over must not be given the benefits of insurance

* **SECURITY PROBLEMS:**

Not every user of the database system should be able to access all the data For example, a policy holder if given access to update the database may increase his terms of service without paying for that much time period In such case insurance company is at loss Enforcing such security constraints is difficult in the file system

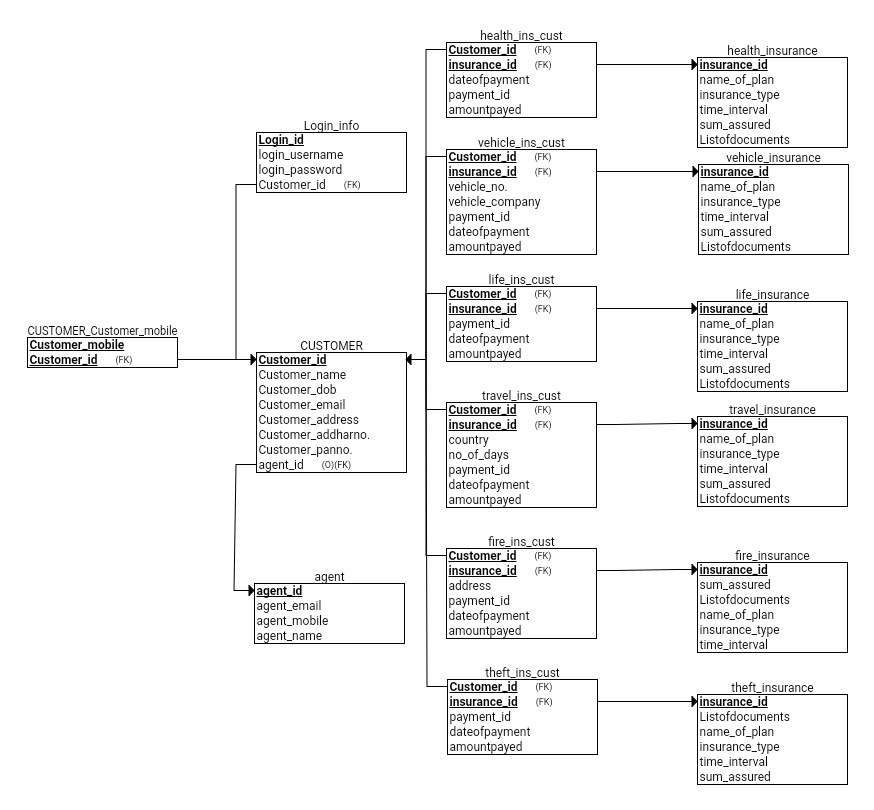
* **ATOMICITY PROBLEMS:**

Atomic means it must happen in entirety or not at all Failures may leave the database in an inconsistent state with partial updates carried out For example an insurance holder pays his monthly amount regularly But one such time in case of power failure, his account may not be updated and not reflected in the database In such a case his policy will be cancelled

* **CONCURRENT ACCESS ANOMALIES:**

For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously This may lead to inconsistency For example two users simultaneously make an entry of a new policy holder This would lead to two entries of the same person

**Experiment 2:**

**RELATIONAL MODEL:**

**Experiment 3:**

**QUERIES**

**SELECTION:**

1)Select login\_id where id is 110.

- σlogin\_id="110"(login\_info)

2)Select all customer\_id.

- σcustomer\_id(customer)

3)Select insurance\_id of any two tables.

- σinsurance\_id(life\_insurance ⋈ travel\_insurance)

4)Select insurance\_id of theft\_ins\_cust.

- σinsurance\_id(theft\_ins\_cust)

**PROJECTION:**

1)Display login\_id.

- πlogin\_id(login\_info)

2)Display customer\_id of fire\_ins\_cust table.

- πcustomer\_id(fire\_ins\_cust)

3)Display id and payment\_id of vehicle insurance.

- πcustomer\_id,payment\_id(vehicle\_ins\_cust)

4)Displayinsurance\_id of health insurance table.

- πinsurance\_id(health\_insurance)

**CARTESIAN PRODUCT:**

1)Select customer name and agent\_id.

- σcustomer\_name ^ agent\_id(customer X agent)

2)Select payment from two different table.

- σpayment\_id="110" v payment\_id="1102"(health\_ins\_cust X life\_ins\_cust)

3)Select name of plan of "XYZ".

- σname\_of\_plan="XYZ"(travel\_insurance X life\_insurance)

4)Slect time\_interval of 2 years.

- σtime\_interval\_years="2"(fire\_insurance X theft\_insurance)

**UNION:**

1)Display those customer\_id who has emi\_desc "50000" in life and travel insurance.

- πcustomer\_id(σemi\_desc="50000")(life\_ins\_cust) U πcustomer\_id(σemi\_desc="50000")(travel\_ins\_cust)

2)Display all the agent and customer name.

- πcustomer\_name(customer) U πagent\_name(agent)

3)Display customer mobile number and emial address.

- πcustomer\_mobile(customer\_customer\_mobile) U πcustomer\_email(customer)

4)Display common customer\_id and login\_id.

- πcustomer\_id(customer) U πlogin\_id(login\_info)

**SET DIFFERENCE:**

1)Select customer having health insurance that are not in travel insurance

- ∏customer\_name(health\_ins\_cust) - ∏customer\_name(travel\_ins\_cust)

2)Select customer having life insurance that are not in theft insurance

- ∏customer\_name(life\_ins\_cust) - ∏customer\_name(theft\_ins\_cust)

3)Select customer having theft insurance that are not in travel insurance

- ∏customer\_name(theft\_ins\_cust) - ∏customer\_name(travel\_ins\_cust)

4)Select customer having travel insurance that are not in health insurance

- ∏customer\_name(travel\_ins\_cust) - ∏customer\_name(health\_ins\_cust)

**NATURAL JOIN:**

1)Select customer name with login id ‘xyz’.

- ∏customer\_name( σlogin\_id=’xyz’(customer⋈login\_info))

2)Select customer id of customers having travel insurance of more than 5 days.

- ∏customer\_id(σno\_of\_days>5(customer⋈travel\_ins\_cust))

3)Select customer id of all customers having agent with agent\_id=’abc’.

- ∏customer\_id(σagent\_id=’abc’(customer⋈agent))

4)Select Customer\_mobile of customer\_name ’yqz’.

- ∏customer\_mobile(σcustomer\_name=’yqz’(customer⋈customer\_customer\_mobile))

**COMPOSITION OF ANY TWO FROM (1-6) OPERATORS:**

1)Display those customer\_id who has emi\_desc "50000" in life and travel insurance.

- πcustomer\_id(σemi\_desc="50000")(life\_ins\_cust) U πcustomer\_id(σemi\_desc="50000")(travel\_ins\_cust)

2)Display all the agent and customer name.

- πcustomer\_name(customer) U πagent\_name(agent)

3)Select name of plan of "XYZ".

- σname\_of\_plan="XYZ"(travel\_insurance X life\_insurance)

4)Slect time\_interval of 2 years.

- σtime\_interval\_years="2"(fire\_insurance X theft\_insurance)

**COMPOSITION OF ANY THREE FROM (1-6) OPERATORS:**

1)Select customer having health and theft insurance that are not in travel insurance

- ∏customer\_name(health\_ins\_cust⋈theft\_ins\_cust) - ∏customer\_name(travel\_ins\_cust)

2)Select customer having life and fire insurance that are not in theft insurance

- ∏customer\_name(life\_ins\_cust⋈fire\_ins\_cust) - ∏customer\_name(theft\_ins\_cust)

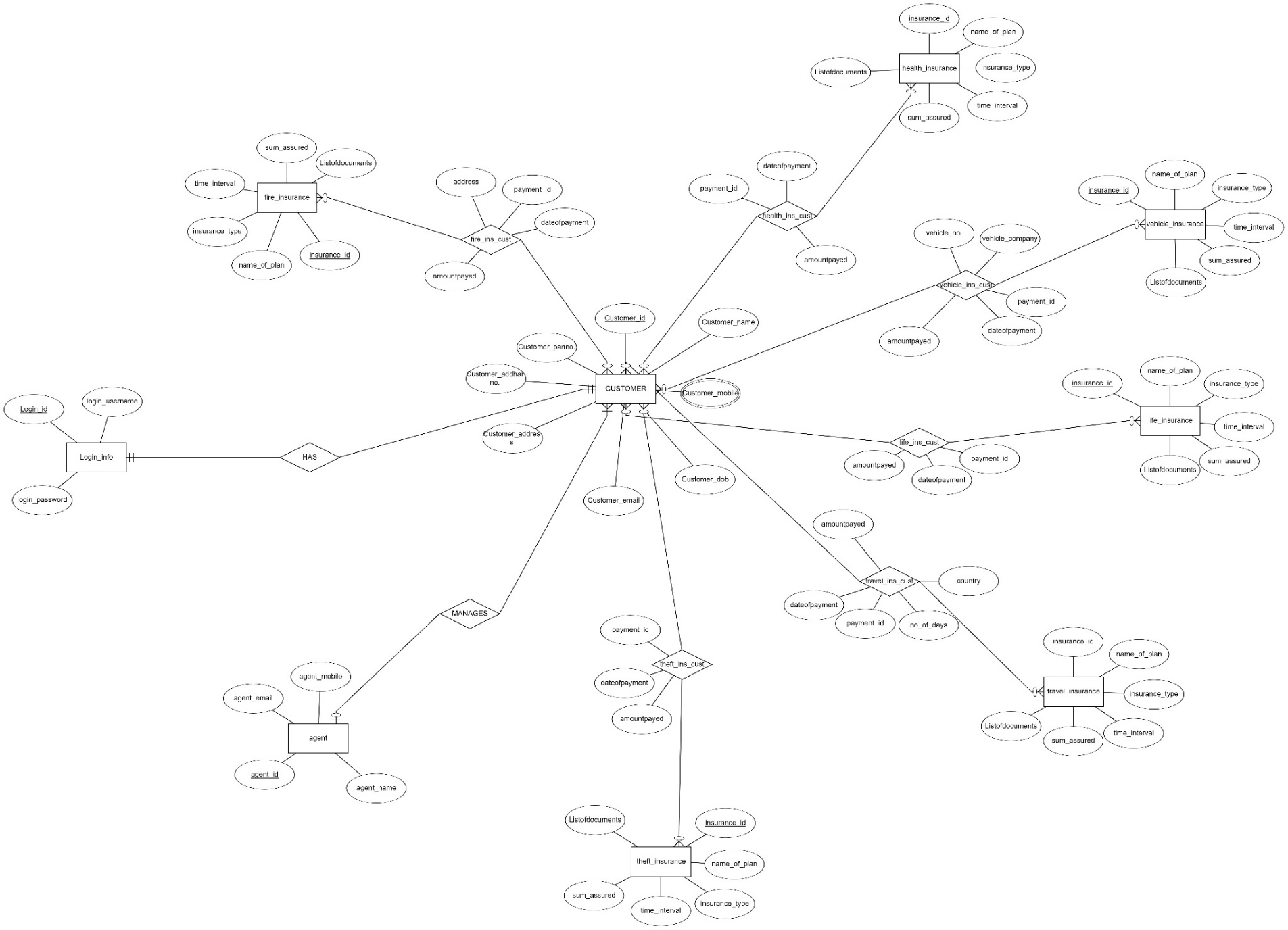
3)Select customer having theft insurance that are not in travel and life insurance

- ∏customer\_name(theft\_ins\_cust) - ∏customer\_name(travel\_ins\_cust⋈life\_ins\_cust)

4)Select customer having travel insurance that are not in health and vehicle insurance

- ∏customer\_name(travel\_ins\_cust) - ∏customer\_name(health\_ins\_cust⋈vehicle\_ins\_cust)

**Experiment 4:**

**ENTITY-RELATIONSHIP MODEL:**

**PRIMARY KEY AND FOREIGN KEY OF EACH TABLE:**

|  |  |  |
| --- | --- | --- |
| **Name of the table** | **Primary Key** | **Foreign Key** |
| Customer | customer\_id | agent\_id |
| Agent | agent\_id |  |
| login\_info | login\_id | customer\_id |
| customer\_customer\_mobile | customer\_mobile | customer\_id |
| health\_insurance | insurance\_id |  |
| life\_insurance | insurance\_id |  |
| travel\_insurance | insurance\_id |  |
| theft\_insurance | insurance\_id |  |
| fire\_insurance | insurance\_id |  |
| vehicle\_insurance | insurance\_id |  |
| health\_ins\_cust | payment\_id | customer\_id, insurance\_id |
| life\_ins\_cust | payment\_id | customer\_id, insurance\_id |
| travel\_ins\_cust | payment\_id | customer\_id, insurance\_id |
| theft\_ins\_cust | payment\_id | customer\_id, insurance\_id |
| fire\_ins\_cust | payment\_id | customer\_id, insurance\_id |
| vehicle\_ins\_cust | payment\_id | customer\_id, insurance\_id |

**Experiment 5:**

Softwares and online platforms that can be used to run SQL queries are as follows –

WAMP, XAMP, Oracle, sqljs.org , etc.

We have used sqljs.org for our project. It is an open source online platform to create and implement SQL database as well as queries.

IMS DB file: <https://github.com/PARTH-4399/Insurance_Management_System>

**TABLES:**

* Customer :

| Customer\_id | Customer\_name | Customer\_dob | Customer\_email | Customer\_address | Customer\_addharno | Customer\_panno | agent\_id |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Meredith | 2017-11-05 00:00:00 | [mchatres0@xrea.com](mailto:mchatres0@xrea.com) | 40 Comanche Point | 2596046113 | 907515827 | 70 |
| 2 | Bennie | 2019-02-24 00:00:00 | [bhutchins1@networksolutions.com](mailto:bhutchins1@networksolutions.com) | 7 Namekagon Park | 8547218068 | 1249047412 | 36 |
| 3 | Cammy | 2019-06-29 00:00:00 | [corys2@omniture.com](mailto:corys2@omniture.com) | 2380 Corben Terrace | 2668488478 | 819914576 | 115 |
| 4 | Rachele | 2016-06-29 00:00:00 | [rbatrim3@amazon.co.uk](mailto:rbatrim3@amazon.co.uk) | 8 Carberry Crossing | 5749083470 | 106490486 | 117 |
| 5 | Hally | 2019-01-13 00:00:00 | [hsanthouse4@ucoz.com](mailto:hsanthouse4@ucoz.com) | 18345 Lillian Place | 1967111987 | 9676578061 | 43 |

* Agent :

| agent\_email | agent\_mobile | agent\_name | agent\_id |
| --- | --- | --- | --- |
| [edjekic0@state.tx.us](mailto:edjekic0@state.tx.us) | 1096575353 | Elisabet | 1 |
| gmaccole1@shop-pro.jp | 7353596090 | Gnni | 2 |
| gredmain2@time.com | 176691286 | Gerrie | 3 |
| sgerrit3@symantec.com | 5816916807 | Sabrina | 4 |
| [slampl4@mlb.com](mailto:slampl4@mlb.com) | 3560438683 | Selia | 5 |

* customer\_customer\_mobile :

| Customer\_mobile | Customer\_id |
| --- | --- |
| 2614170402 | 1 |
| 5866479753 | 2 |
| 5411125871 | 3 |
| 1806247577 | 4 |
| 4335496753 | 5 |

* login\_info :

| Login\_id | login\_username | login\_password | Customer\_id |
| --- | --- | --- | --- |
| 1001 | tdanigel0 | QDSUrNNVRpH | 1 |
| 1002 | javrahamoff1 | 04IS2g0GT | 2 |
| 1003 | mtonsley2 | qiPNkv9E | 3 |
| 1004 | gpalk3 | ACwjhIl8Yyl | 4 |
| 1005 | nevershed4 | EXEjn7V2VPf | 5 |

* vehicle\_insurance :

| name\_of\_plan | vehicle\_type | time\_interval\_years | sum\_assured | Listofdocuments | insurance\_id |
| --- | --- | --- | --- | --- | --- |
| Legros Inc | Lotstring | 9 | 1597270 | Driving License | 1 |
| Wiegand-Wolf | Tres-Zap | 14 | 1007670 | Driving License | 2 |
| Wolf-Reynolds | Rank | 9 | 172630 | Pan Card | 3 |
| Leffler-Ferry | Bigtax | 8 | 1032170 | Driving License | 4 |
| Grady Group | Latlux | 15 | 880470 | Driving License | 5 |

* theft\_insurance :

| Listofdocuments | insurance\_id | name\_of\_plan | theft\_items | time\_interval\_years | sum\_assured |
| --- | --- | --- | --- | --- | --- |
| Aadhar Card | 1 | Miller-Huel | jewellery | 1 | 275260 |
| Aadhar Card | 2 | Mayert Inc | cash | 15 | 146810 |
| Aadhar Card | 3 | Nader-Barton | curios | 8 | 506050 |
| Aadhar Card | 4 | Harber Group | title deeds | 6 | 314490 |
| Passport | 5 | Price-Schimmel | jewellery | 1 | 258090 |

* fire\_insurance :

| sum\_assured | Listofdocuments | insurance\_id | name\_of\_plan | property\_damaged | time\_interval\_years |
| --- | --- | --- | --- | --- | --- |
| 348700 | Passport | 1 | Purdy, Hane and Gaylord | electronics | 6 |
| 488890 | Pan Card | 2 | Lynch and Sons | electronics | 3 |
| 289750 | Pan Card | 3 | Simonis and Sons | electronics | 10 |
| 250670 | Driving License | 4 | Kunze and Sons | electronics | 6 |
| 563300 | Aadhar Card | 5 | Schultz Group | clothing | 6 |

* travel\_insurance :

| insurance\_id | name\_of\_plan | traveller\_type | time\_interval\_years | sum\_assured | Listofdocuments |
| --- | --- | --- | --- | --- | --- |
| 1 | Schroeder Inc | couple | 2 | 102620 | Passport |
| 2 | Green, Connelly and Rodriguez | family | 9 | 128840 | Aadhar Card |
| 3 | Fay, Davis and Krajcik | couple | 6 | 495090 | Aadhar Card |
| 4 | Gislason, Fisher and Huels | family | 7 | 364240 | Passport |
| 5 | Krajcik, Breitenberg and Hane | couple | 8 | 474900 | Aadhar Card |

* life\_insurance :

| name\_of\_plan | insurance\_type | time\_interval\_years | sum\_assured | Listofdocuments | insurance\_id |
| --- | --- | --- | --- | --- | --- |
| Mertz-Block | Whole | 12 | 136300 | Passport | 1 |
| Reichel-Kub | Whole | 3 | 100500 | Aadhar Card | 2 |
| Price Inc | Whole | 3 | 156600 | Pan Card | 3 |
| Will-Wehner | Whole | 13 | 192200 | Driving License | 4 |
| Huels Inc | Term | 1 | 121000 | Passport | 5 |

* health\_insurance :

| insurance\_id | name\_of\_plan | insurance\_type | time\_interval\_years | sum\_assured | Listofdocuments |
| --- | --- | --- | --- | --- | --- |
| 1 | Rippin, Hodkiewicz and Christiansen | No-claim benefit | 2 | 194800 | Passport |
| 2 | Simonis Group | Health check-ups | 15 | 215000 | Aadhar Card |
| 3 | Kemmer, Halvorson and Larkin | Health check-ups | 7 | 342900 | Aadhar Card |
| 4 | Kreiger, Heaney and Simonis | No-claim benefit | 14 | 421300 | Aadhar Card |
| 5 | Schneider-Lemke | Health check-ups | 6 | 478900 | Pan Card |

* vehicle\_ins\_cust :

| vehicle\_no | vehicle\_company | payment\_id | dateofpayment | mode\_of\_payment | emi\_desc | fullpayment\_desc | Customer\_id | insurance\_id |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| AB1254 | Honda | PA452 | 2017-11-05 00:00:00 | full payment |  | ₹1597270 | 5 | 1 |
| AM1654 | Audi | PA862 | 2015-10-08 00:00:00 | EMI | ₹88047 per month |  | 12 | 5 |
| GH6544 | BMW | PJ552 | 2016-05-15 00:00:00 | full payment |  | ₹1007670 | 15 | 2 |
| IH6854 | Bajaj | PI652 | 2019-06-25 00:00:00 | full payment |  | ₹519580 | 26 | 9 |
| KS3544 | Honda | OH952 | 2015-07-30 00:00:00 | EMI | ₹21542 per month |  | 65 | 22 |

* theft\_ins\_cust :

| payment\_id | dateofpayment | mode\_of\_payment | emi\_desc | fullpayment\_desc | Customer\_id | insurance\_id |
| --- | --- | --- | --- | --- | --- | --- |
| PA658 | 2015-12-15 00:00:00 | full payment |  | ₹506050 | 12 | 3 |
| DF842 | 2016-02-06 00:00:00 | EMI | ₹24135 per month |  | 23 | 6 |
| WE952 | 2016-06-02 00:00:00 | full payment |  | ₹280740 | 34 | 9 |
| WF945 | 2018-03-09 00:00:00 | full payment |  | ₹280740 | 54 | 12 |
| WF546 | 2019-02-16 00:00:00 | full payment |  | ₹280740 | 24 | 13 |

* fire\_ins\_cust :

| address | payment\_id | dateofpayment | mode\_of\_payment | emi\_desc | fullpayment\_desc | Customer\_id | insurance\_id |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 850 Delaware Way | AS451 | 2019-08-24 00:00:00 | full payment |  | ₹417800 | 14 | 34 |
| 01 Kim Circle | AY41 | 2019-02-21 00:00:00 | full payment |  | ₹226810 | 25 | 19 |
| 145 Helena Hill | PF564 | 2019-05-18 00:00:00 | EMI | ₹40044 per month |  | 63 | 25 |
| 8152 Mesta Terrace | VH542 | 2018-12-24 00:00:00 | EMI | ₹41780 per month |  | 90 | 34 |
| 7266 Gulseth Drive | OK425 | 2019-06-17 00:00:00 | full payment |  | ₹672030 | 2 | 33 |

**Experiment 6:**

**SQL**

It is a computer language for storage, manipulation and retrieving data stored in a relational database. SQL is used to communicate with database and handling data.

**DDL**

DDL is acronym for Data Definition Language. It allows the user to define the database schema by using some SQL commands which includes creating and modification of the structures of database objects in database.

**DML**

DML is acronym for Data Manipulation Language. It is used for adding, deleting and modifying data in a database. It is kind of a sublanguage of SQL.

**DCL**

DCL is acronym for Data Control Language. It is a language used to control authorization i.e. controlling the access of the data stored in the database.

**DATABASE**

Database is a collection of data in a systematic manner represented in tabular form. It supports manipulation and storage of data and makes data management easy and efficient as compared to file system.

**TABLE CREATION**

Tables in DBMS are created by CREATE TABLE statement

Example:

CREATE TABLE CUSTOMER

(

Customer\_id VARCHAR NOT NULL, Customer\_name VARCHAR NOT NULL,

Customer\_dob DATE NOT NULL, Customer\_email VARCHAR,

Customer\_address VARCHAR NOT NULL, Customer\_addharno INT NOT NULL,

Customer\_panno VARCHAR NOT NULL, agent\_id VARCHAR,

PRIMARY KEY (Customer\_id),

FOREIGN KEY (agent\_id) REFERENCES agent(agent\_id),

UNIQUE (Customer\_addharno ),

UNIQUE (Customer\_panno )

);

**ALTER TABLE**

A command named alter in SQL is used for altering the table structure such as adding column to existing table, to rename any existing column, to change datatype of any column or to modify its size or to drop a column from the table.

Example:

ALTER TABLE Customer  
ADD Middle\_Name varchar(25);

**CONSTRAINTS**

Constraints are the rules use to limit the type of data entering the data to maintain the accuracy and integrity.

**PRIMARY KEY**

Primary key is a constraint used in DBMS, used to defined all the rows uniquely identify all table records.

**FOREIGN KEY**

Foreign key is a constraint in which a column or a group of column in a table provides a link between data in two tables.

**UNIQUE**

Unique is constraint in DBMS, in which one field/ columns uniquely identify a record in a database table.

**NOT NULL**

NOT NULL is a constraint in DBMS which only accepts not null values in that record.

**CHECK**

Check is a constraints which specifies to a requirement to be fulfilled in order to enter the record.

**IN OPERATOR**

IN operator allows us to easily test if an expression matches any values in a list of values.

**Experiment 7:**

**AGGREATE FUNCTION:**

1) Sum()

2) Avg()

3) Count()

4) Min()

5) Max()

* Sum(): The sum function calculates the sum of (non-zero) values.
  + Example: Sum(fullpayment\_desc) returns the total payment of customer for the particular insurance.
* Count():The count function returns the no of records in a table or colounm.
  + Example: Count(\*) when applied on customer table returns no of customer.
* Avg(): The Avg returns the average of all values of a column.
  + Example : Avg(sum\_assured) on any insurance table returns average sum assured.
* Min(): The Min returns the minimum value in a column.
  + Example : Min(emi\_desc) in any insurance table returns the minimum emi description among the column of the table.
* Max() : The Max returns the maximum value in a column.
  + Example: Max(emi\_desc) in any insurance table returns the maximum emi description among the column of the table.

### **IN BUILT FUNCTION:**

# Date Function: The date function is one of the most important function as it helps in extraction of details such as date, month and year and also maintains a proper format for storing date.

Example: Extract() Function returns the single part of date/time

Select Extract(DAY FROM dateofpayment) ; will give us day of the payment of insurance of any particular employee.

# NUMERIC FUNCTION:

# The numeric function performs operation on numbers and returns numbers as output .It helps in rounding and converting numbers in database.

# Example:

# ABS(): Helps in converting into signed number.

* Example: Abs(-439) gives 439
* CEILING(): Helps in converting smallest integer value into greater or equal to that number.
  + CEILING helps in rounding the scale quantity of chemical quantity in that factory.

**Experiment 8:**

**SET OPERATIONS**

A table is basically a subset of Cartesian product among domains (sets) of its columns. So, various set operations, such as Union Intersection, Difference (Minus), can be applied to tables also.

The UNION clause combines the output of query1 and query2. The output will be as a single set of columns and rows. The output will contain all the records from query1 and query2. Here, output will contain all the records from query1 and query2, but the common records will appear only once The number of columns, name of every single column and their data types in both the query need to be the same NULL value will not be ignored while-comparing records from both queries. The output will contain sorted records in ascending order according to columns present in SELECT statement.

Example :

* Select the customers who have taken any of one of travel and fire insurance.

(SELECT customer\_name FROM travel\_ins\_cust)

UNION

(SELECT customer\_name FROM fire\_ins\_cust)

The INTERSECT clause combines the output of query1 and query2. The output will contain records from query1 and query2. Here, output will contain those records which are common to query1 and query2. The number of columns, name of every single column and their data types in both the query need to be the same. The output will be same even if the orders of query1 and query2 are altered NULL value will not be ignored while comparing records from both queries. The output will contain sorted records in ascending order according to columns present in SELECT statement

Example :

* Select the customers who have taken both of travel and fire insurance.

(SELECT customer\_name FROM travel\_ins\_cust)

INTERSECT

(SELECT customer\_name FROM fire\_ins\_cust)

**SUB-QUERY**

A sub-query is a nested SQL query that occurs inside the main query. It may occur inside SELECT clause, or WHERE clause or FROM clause. It is also referred to as inner query. The inner query is executed first and then the parent query is executed (the query in which the sub query is present). So, the output of the sub query is passed to the parent query (also known as outer query).

* Select those vehicle insurance whose sum assured is strictly greater then the sum assured by the fire insurance with insurance id = ‘6001’.

SELECT sum\_assured FROM vehicle\_insurance WHERE sum\_assured >

(SELECT sum\_assured FROM fire\_insurance WHERE ins\_id = ‘6001’);

**Experiment 9:**

**GROUP BY:**

It is used to arrange identical data into a specific group with the help of some functions like COUNT, etc. Generally, the data is grouped by a column name of that table.

Example:

* + Show the list of customers for each agent having atleast one customer.

SELECT COUNT(customer\_id), agent\_id FROM customers  
GROUP BY agent\_id;

**HAVING:**

HAVING clause is used with aggregate functions. It is generally used with GROUP BY clause so as to arrange data "having" a particular condition. Since we cannot use WHERE clause in this situation, we use HAVING clause.

Example:

* + Select agent\_id of agents having atleast 5 customers.

SELECT agent\_id FROM customers GROUP BY agent\_id HAVING  
COUNT(customer\_id) > 4;

**ORDER BY:**

ORDER BY is used for the sorting of resultant data in either descending order. By default it sorts the result in descending order.

Example

* + List all the fire insurance with increasing sum assured.

SELECT name\_of\_plan FROM fire\_insurance ORDER BY sum\_assured ASC;

**Experiment 10:**

**JOIN:**

JOIN is used to combine rows from two or more tables. There are various kind of JOIN operators like Inner Join, Left Join, Right Join, Full Join, Self join.

**INNER JOIN:**

INNER JOIN selects those rows which are common in the target tables if there are common rows in both the tables.

**LEFT JOIN:**

LEFT JOIN on 2 tables returns all the rows of left table (which includes those values too which are common in both left and right table) if there are common rows in both the tables.

**RIGHT JOIN:**

RIGHT JOIN on 2 tables returns all the rows from the right table (which also includes the rows which are common in both the tables) if there are common rows in both the tables.

**FULL JOIN:**

FULL JOIN practically returns all the records of both the tables if there are common rows in both the tables.

**SELF JOIN:**

SELF JOIN just joins a table with itself.

**EXISTS:**

EXISTS operator is used to check whether a certain value or record is existing or not in a subquery. It simply returns TRUE or FALSE as per the result.

**ANY:**

ANY operator is used with HAVING or WHERE clause. It simply returns TRUE if "any" of the subquery values meet the given condition.

**ALL:**

ALL operator is also used with HAVING or WHERE clause. It returns TRUWE if "all" of the subquery values meet the given condition.

**TRIGGERS (EXTRA TOPIC)**

TRIGGER: A trigger may be a hold on procedure in information that mechanically invokes whenever a special event within the system happens. For instance. A trigger is invoked once a row is inserted into a table such that table or once bound table columns area unit being updated.

SYNTAX:

create trigger {name\_of\_trigger}

{after | before}

{update | delete | insert}

on {name\_of\_table}

{for each row}

{body\_of\_trigger}

1. create trigger {name\_of\_trigger}: It is used to create or replace an existing trigger with the name\_of\_trigger.

2. {after | before}: It is used to specify when will the trigger be executed.

3. {update | delete | insert}: It is used to specify the Data Manupulation Language operation.

4. on {name\_of\_table}: It is used to specify the name of the table associated with the trigger.

5. [for each row]: It is used to specify a trigger of row-level, i.e., trigger will be executed for each row being affected.

6. {body\_of\_trigger}: It is the combination of logical operation to be performed as trigger is fired

EXAMPLE:

Create trigger remaining\_value

after insert on health\_ins\_cust natural join health\_insurance

for each row set

health\_ins\_cust.remaining\_amount=health\_insurance.sum\_assured - emi\_payed

ADVANTAGES:

1. Triggers are generalized action that are performed after a condition. Hence, by coding once and storing we can use it in future.

2. Triggers in SQL provides execution of multiple scheduling/different tasks while performing the trigger synchronously.

DISADVANTAGES:

1. Decrement in performance of the respective database Because of complex nature of triggers as it is executed after the execution of a particular query.

2. Can get complex if error is generated

**FUNCTIONAL DEPENDENCIES, CANDIDATE KEYS AND NORMAL FORMS**

* **Table :** Customer\_mobile

**FD's**

b->a

**Find Minimal Cover**

b->a

**Candidate Keys Found**

b

**second,third,bcnf**

b->a

* **Table :** Login\_info

**FD's**

a->d

b->d

c->d

**Find Minimal Cover**

a->d

b->d

c->d

**Candidate Keys Found**

a b c

**second Normal form**

**Table – 1 :**

**Attributes**

a b c

**Table – 2 :**

**Attributes**

a d

**Functional Dependencies**

a->d

**Third normal form**

**Table – 1 :**

**Attributes**

a d

**Functional Dependencies**

a->d

**Table – 2 :**

**Attributes**

b d

**Functional Dependencies**

b->d

**BCNF form**

**Table – 1 :**

**Attributes**

a b c

**Table – 2 :**

**Attributes**

a d

**Functional Dependencies**

a->d

* **Table :** Customer

**FD's**

a->bcdefgh

f->abcdegh

g->abcdefh

**Minimal cover**

a->g

f->g

g->a

g->b

g->c

g->d

g->e

g->f

g->h

**Candidate Keys Found**

a f g

**Already in second,third and BCNF**

* **Table :** Agent

**FD's**

a->bcd

**Minimal Cover**

a->b

a->c

a->d

**Candidate Keys Found**

a

**Already in all the normal form.**

* **Table :** health\_insurance

**FD's**

a->bcdef

**Minimal Cover**

a->b

a->c

a->d

a->e

a->f

**Candidate Keys Found**

a

**Already in all the normal form.**

* **Table :** vehicle\_insurance

**FD's**

a->bcdef

**Minimal Cover**

a->b

a->c

a->d

a->e

a->f

**Candidate Keys Found**

a

**Already in all the normal form.**

* **Table :** life\_insurance

**FD's**

a->bcdef

**Minimal Cover**

a->b

a->c

a->d

a->e

a->f

**Candidate Keys Found**

a

**Already in all the normal form.**

* **Table :** travel\_insurance

**FD's**

a->bcdef

**Minimal Cover**

a->b

a->c

a->d

a->e

a->f

**Candidate Keys Found**

a

**Already in all the normal form.**

* **Table :** fire\_insurance

**FD's**

a->bcdef

**Minimal Cover**

a->b

a->c

a->d

a->e

a->f

**Candidate Keys Found**

a

**Already in all the normal form.**

* **Table :** theft\_insurance

**FD's**

a->bcdef

**Minimal Cover**

a->b

a->c

a->d

a->e

a->f

**Candidate Keys Found**

a

**Already in all the normal form.**

* **Table :** health\_ins\_cust

**FD's**

c->abdefg

**Minimal Cover**

c->a

c->b

c->d

c->e

c->f

c->g

**Candidate Keys Found**

c

**Already in all the normal form.**

* **Table :** vehicle\_ins\_cust

**FD's**

c->abdefghi

**Minimal Cover**

c->a

c->b

c->d

c->e

c->f

c->g

c->h

c->i

**Candidate Keys Found**

c

**Already in all the normal form.**

* **Table :** life\_ins\_cust

**FD's**

c->abdefg

**Minimal Cover**

c->a

c->b

c->d

c->e

c->f

c->g

**Candidate Keys Found**

c

**Already in all the normal form.**

* **Table :** travel\_ins\_cust

**FD's**

c->abdefghi

**Minimal Cover**

c->a

c->b

c->d

c->e

c->f

c->g

c->h

c->i

**Candidate Keys Found**

c

**Already in all the normal form.**

* **Table :** fire\_ins\_cust

**FD's**

c->abdefgh

**Minimal Cover**

c->a

c->b

c->d

c->e

c->f

c->g

c->h

**Candidate Keys Found**

c

**Already in all the normal form.**

* **Table :** theft\_ins\_cust

**FD's**

c->abdefg

**Minimal Cover**

c->a

c->b

c->d

c->e

c->f

c->g

**Candidate Keys Found**

c

**Already in all the normal form.**