PROJECT REPORT

DS-5110 Introduction to Data Management and Processing, Fall 2023

INSURIT - Insurance Provider Simulation

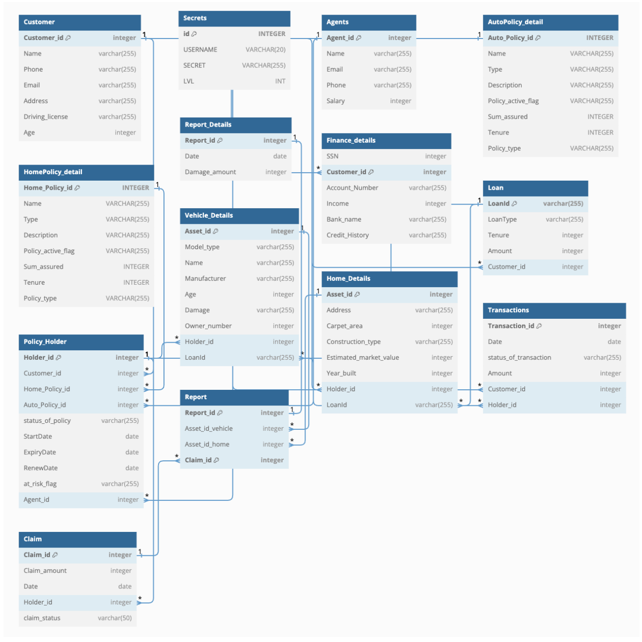
(github.com) Insureit-DB-Simulation: CLI based Insurance provider simulation

Karan Badlani, Naman Singhal, Isha Singh

Introduction

InsureIt, a sophisticated system, revolutionizes the insurance landscape by seamlessly facilitating the sale and management of automobile and homeowners' insurance policies. Beyond transactional capabilities, InsureIt introduces dynamic premium computation, private agent portals, and an analytical dashboard for leadership insights. Leveraging advanced database management and processing techniques, the user-friendly Command-Line Interface (CLI) ensures intuitive interactions. The system's robust foundation encompasses a comprehensive database design, connecting entities like Customers, Policy Holders, Auto and Home Policy Details, Vehicles, Homes, Finance Details, Transactions, Agents, Reports, Claims, and Loans. This report provides a detailed exploration of InsureIt's architecture, functionalities, and methodologies, representing a significant advancement in insurance policy management and database processing integration.

ER - Diagram



ERD - Click to view Image in separate tab

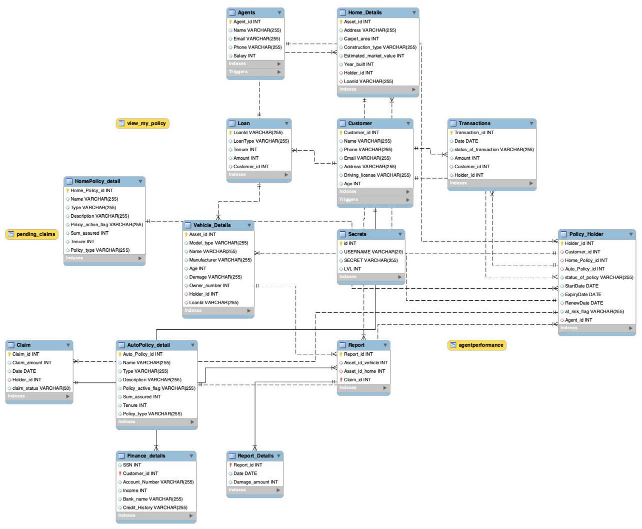
The diagram above illustrates every table/entity within the database system, showcasing their respective attributes and depicting the relationships among various entities.

Entity/Table Description:

- *Customers* - It stores all the PII and general information about customers.
- *Agents* - It stores information of employees who are designated to the role of an Agent in Insurit.
- *Secrets* - This table stores the username and hashed (SHA256) password for all users - admin, customers and agents. This table is populated by Triggers and is not connected with any other table to keep it isolated.
- *Policy_Holder* - Each customer can either hold an automotive/Vehicles or home policy. One customer can have multiple of these policies. This table stores information on policies a customer holds.
  Each policy has its unique number which maps it to a customer and agent. This is a core table which holds the system together
- *HomePolicy_Details* - Insurit provides certain policies to its users for purchase or hold until expiry. This table contains details of all Home Insurance policies offered (currently active) and the once offered in the past (inactive policies).
- *AutoPolicy_Details* - Similarly like HomePolicy this table holds details of all Auto/Vehicle Insurance policies offered (currently active) and the once offered.
- *Vehicle_Details* - Each Insurance policy Policy_Holder ID insures an asset. This table stores the details of vehicle assets for auto policies.
- *Home_Details* - Similarly, this table contains the details of Home assets for home policies.
- *Claim* - During an unfortunate loss or damage of an asset which is insured by Insurit policies, the customer can create a claim for some amount. This table stores the details of these claims and their status.
- *Report* - In case the damage to the asset has a corresponding Law Suit or Police report then we save it in this table.

- *Report_Details* - This table is an extension to the Report table with additional information.
- *Transactions* - All premiums paid to Insurit against some policy is stored in this table.
- *Loan* - This table saves all the mortgage and loan related information for the assets.
- *Finance_Details* - When a customer makes payment against some policy then we store all their payment methods in this table for quicker transactions.

Database Schema Diagram



Database Diagram - Click to view in separate tab

The above diagram is extracted from the database. Each table has a heading in blue and each view is colored in Yellow.

Database Triggers

We have the following 2 triggers for our database:

- *generate_credentials* - The purpose of the "generate_credentials" trigger is to automate the process of generating secure credentials for new customers within the insurance application. This trigger operates on the principle of enhancing data security by hashing the customer's name using the SHA2 hashing function. Upon the insertion of a new customer record, the trigger dynamically generates a hashed password and subsequently inserts a new record into the SECRETS table. This record includes a unique username and the hashed password, ensuring a robust and secure authentication mechanism for customer access to the system.
- *generate_agent_credentials* - The purpose of the "generate_agent_credentials" trigger serves a parallel purpose, specifically tailored for the generation of credentials for new agents joining the insurance system. Like its counterpart for customers, this trigger employs a secure approach to credential creation.

Database Functions

We have the following 3 functions as a part of our database:

- *CalculatePremium* - The "CalculatePremium" function serves to compute insurance premiums by considering policy type, sum assured, tenure, and the age of the policyholder. Its parameters include policy_id, age, sum_assured, tenure, and policy_type. This function applies a predefined logic to determine a base premium, subsequently adjusting it based on age categories.
- *Auth* - The "auth" function operates as an authentication mechanism for user credentials and permissions. It takes parameters such as USR (Username), PSWD (Password), and PERMISSION (Permission level). The function generates a hashed password and validates the provided credentials and permissions against the database, returning a binary result.
- *GetCustomersForAgent*- The "GetCustomersForAgent" function retrieves customer details associated with a specific agent, taking agentID as its parameter. Utilizing a JOIN operation on relevant tables, this function compiles customer information into a formatted result.

Database Stored Procedures

We have the following stored procedures in our database:

- *insertdata* - This stored procedure is used to perform the basic INSERT operation into various tables of our database. It is used to set up the initial database for performing operations on our application. The administrator can also use this stored procedure to reset the data after changes have been made to the database.
- *AddNewPolicy* - The "AddNewPolicy" procedure adds a new insurance policy dynamically to a specified policy table. With parameters like policy_name, policy_type, description, policy_active_flag, sum_assured, tenure, and policy_table, this procedure constructs and executes a SQL query to facilitate policy addition.
- *CreateNewClaim* - The "CreateNewClaim" procedure facilitates the creation of new insurance claims with parameters including p_Claim_amount, p_Date, and p_Holder_id. This straightforward procedure directly

inserts a new claim record with the provided claim amount, date, and policy holder ID.

Database Views

We have the following 3 views for our database:

- *AgentPerformance* - "AgentPerformance" provides an overview of agent performance, displaying Agent_id, Agent_Name, Num_Customers, Num_Policies, and Total_Sales. This view employs JOIN operations on various tables to calculate and aggregate performance metrics for each agent.
- *Pending_Claims* - The "Pending_Claims" view presents pending insurance claims, showing fields like customer_id, past_rejects, claim_id, claim_date, amount claimed, policy status and policy_at_risk. By retrieving relevant information and utilizing conditional logic, this view identifies and displays pending claims.
- *view_my_policy:* The "view_my_policy" view offers a comprehensive perspective on a customer's insurance policies. It amalgamates information from different tables to present details such as Holder_id, Customer_id, Home_Policy_id, Auto_Policy_id, homePolicyName, autoPolicyName, status_of_policy, StartDate, ExpiryDate, RenewDate, at_risk_flag, and Agent_id.

Data Collection and Processing

In this Section, We'll see the Sources, Methods and Strategies Involved in the process of Collecting the Data for our Insurit System and Preprocessing it for the end use.

Data Collection

Data aggregation from Kaggle, Data.gov, and Mockaroo forms the cornerstone of our comprehensive data collection, enriching our insurance management system with diverse and dynamic information sources. The packages used while collecting these data from sources were: Selenium, API requests

Data Processing

Although datasets resembling our tables were discovered, significant attributes were missing, necessitating extensive preprocessing to align with our schema. The final datasets underwent thorough cleaning and processing, executed meticulously using Jupyter Notebook. The core libraries that were used to clean and aggregate data as mentioned in the below methods were: Pandas, Numpy, BeautifulSoup, Data-profiling.

Merging Datasets

The Final Dataset of many tables were formulated by merging more than one csv files that were collected from either Kaggle or Data.gov platforms. This Merging was done through the help of Pandas Library in Python Programming Language.

Populating Child Tables

Following the completion of the primary or independent tables, we populated specific attributes in child tables by referencing their corresponding parent tables.

Generating Data

Following the Primary steps above, there were some attributes for certain tables that were missing, so to populate them, we generated random data with the help of Faker library in python.

Data Profiling / summarization

After crafting the datasets, we checked them to learn more about the scraped and generated data. We found some unusual values in the generated data and discovered some missing information in the data from platforms. We used the Pandas-Profiling library to help us with this analysis.

Handling Missing Values, Duplicates & Outliers

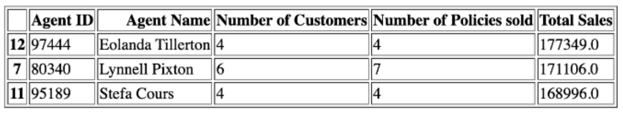
Following the analysis, attributes in datasets with numerous null or missing values were appropriately regenerated, and duplicate values in the respective columns were eliminated. Outliers were addressed by removing corresponding rows to enhance the overall generalization of the data. After all the above steps, Final datasets were created that were used to fill in the tables for our Insurit System.

Analytics and Report

In this section, we harness the wealth of data stored in our system to generate insightful graphs and reports, shedding light on key aspects of our business operations. Through these graphs, we derive meaningful insights into our business operations. It helps us identify the following:

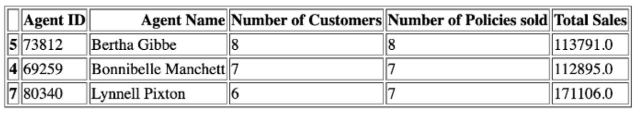
- *Top Performing Agents:* Agents with a higher number of customers and a significant number of policies sold are considered top performers.
- *Underperforming Agents:* Agents with a substantial customer base but a low percentage of policies sold might need attention.
- *Benchmarking*: Comparing the performance of each agent against a predefined benchmark for the average performance of all agents.

Top 3 Agents with Maximum Sales



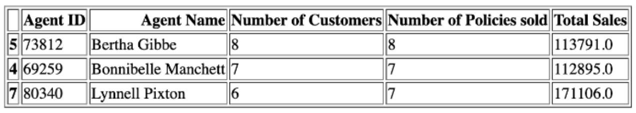
	Agent ID	Agent Name	Number of Customers	Number of Policies sold	Total Sales
12	97444	Eolanda Tillerton	4	4	177349.0
7	80340	Lynnell Pixton	6	7	171106.0
11	95189	Stefa Cours	4	4	168996.0

Top 3 Agents with Maximum Number of Customers



	Agent ID	Agent Name	Number of Customers	Number of Policies sold	Total Sales
5	73812	Bertha Gibbe	8	8	113791.0
4	69259	Bonnibelle Manchett	7	7	112895.0
7	80340	Lynnell Pixton	6	7	171106.0

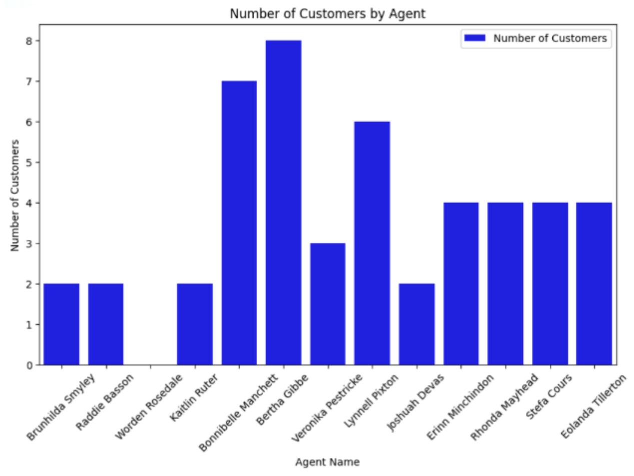
Top 3 Agents with Maximum Policies Sold



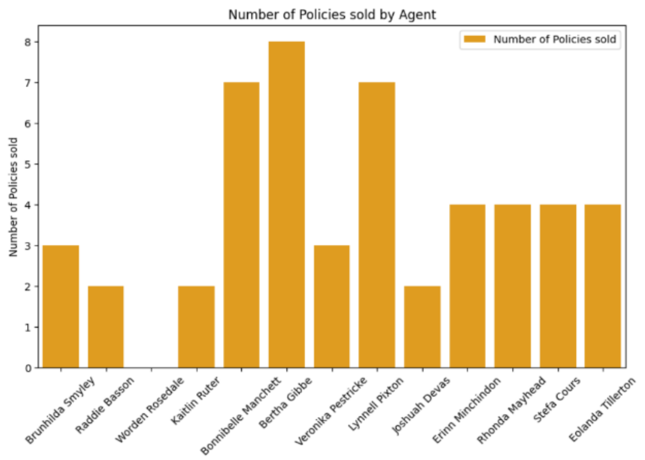
	Agent ID	Agent Name	Number of Customers	Number of Policies sold	Total Sales
5	73812	Bertha Gibbe	8	8	113791.0
4	69259	Bonnibelle Manchett	7	7	112895.0
7	80340	Lynnell Pixton	6	7	171106.0

Number of Customers by Agent

This graph elucidates the distribution of customers across various agents within our system.



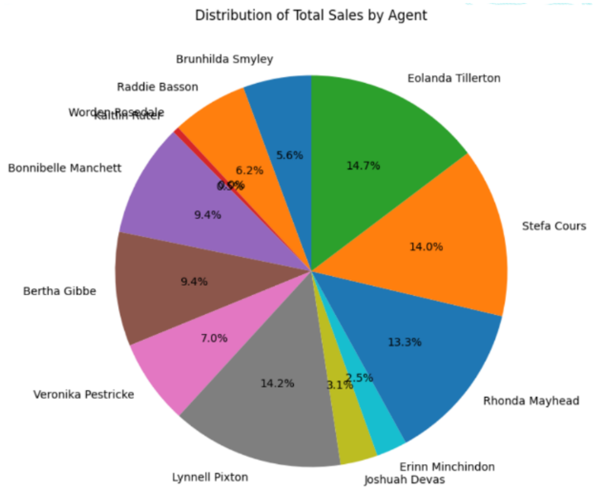
Number of Policies Sold by Agent



This graph highlights the performance of each agent in terms of the quantity of policies sold.

Distribution of Total Sales by Agent

This chart provides a visual representation of the contribution of each agent to the overall policy sales within the system.



Overall, these analytics empower us to make informed decisions, optimize our business processes, and foster a dynamic and competitive environment within our insurance system. The visual representation of data through graphs enhances our ability to interpret complex information and facilitates strategic planning for the continued success of our insurance services

Project Structure

The Github link to this project's source code can be found under the title of this report.

The project folder - Insureit-DB-Simulation - contains mainly Insurit and DataPrep folders along with other git and setup related files. The source code of the application can be found inside the insurit folder. The DataPrep folder contains .ipynb files and raw data along with processed data csv files. All the other files in this folder will be used during setup and explained in further sections.

The sub-folder insurit contains .py scripts and a folder named backend. All the .py script in this sub-folder handles the CLI frontend of this project with __main__.py being the entrypoint.

The backend sub-folder contains all the .py files which manage the interaction between the database and the frontend user. The folder sql_scripts contains all the scripts required to setup the database before the application can be started.

Setting Up the Application

Before we can begin setting up the application we need to make sure we have the following requirements met.

1. Downloaded the application from here to local.
2. MySQL Database and a client.
3. Python Installation.
4. A virtual environment preferably Conda Environment.
5. A package manager like pip or conda.

The following are the main steps required to be performed to set up the project.

Setting up Database on MySQL

To find all the SQL scripts for setup go to the local path *insurit/backend/sql_scripts* from the main folder Insureit-DB-Simulation on your system. This sub-folder contains all the DDL, Triggers, SP, Functions, Views, and DML scripts.

We will execute the following scripts before we start inserting data. All the below paths are relative to the sub-folder *insurit/backend/sql_scripts:*

1. *create_ddl.sql* - Running this script will create a schema *insurit* on the database along with DDLs of all the tables.
   Note: This script also inserts the default credentials for root user.
2. Creating all Triggers
   a. */Triggers/gen_creds_trigg.sql*
3. Creating all Functions
   a. */Functions/calPremium.sql*
   b. */Functions/create_auth_func.sql*
   c. */Functions/function_getcustomerdetails.sql*
4. Creating all Stored-Procedures
   a. */SP/CreateNewClaim_SP.sql*
   b. */SP/addNewPolicy_SP.sql*
   c. */SP/insertdata_sp.sql*
5. Creating all Views
   a. */Views/agentPerformance.sql*
   b. */Views/pending_claims.sql*
   c. */Views/view_my_policy.sql*

Once we have the basic setup we can insert all the data into our tables. To do so we can either login to insurit application as root user and execute the command

```
> database --reset RESET
```

which inserts all the data using the stored-procedure or we can execute .sql files in the following order.

1. *customers.sql*
2. *agents.sql*
3. *autopolicy_detail.sql*
4. *homepolicy_detail.sql*
5. *finance_details.sql*
6. *loan.sql*
7. *policy_holder.sql*
8. *vehicle_details.sql*
9. *home_details.sql*
10. *transactions.sql*
11. *claim.sql*
12. *report.sql*
13. *report_details.sql*

Setting up Conda Virtual Environment

The main folder of the project has an *environments.yml* file which contains a list of all the packages we need to install in the virtual environment. When working on Anaconda/Conda it is straightforward to use this file and create a virtual Environment.

In Terminal or Command Prompt run the following command

```
> conda env create -f environments.yml
```

This command uses the *environments.yml* file to create a new virtual environment in Anaconda - *insurit_env* - and downloads all the required python packages in that environment. Once it has been executed we have fulfilled all package installation requirements to run Insurit.

```
> conda activate insurit_env
```

With this command we can enable the virtual environment and follow steps further.

Installing Insurit as a Package

The main folder of the project contains two setup files - *install.bat* and *install.sh*. For windows PC we will run the *install.bat* file in command prompt and for linux based system and Macs we run the command *bash install.sh* in terminal. Make sure these files are run inside the virtual environment.

Starting the application

Once the application and all the setup steps are completed we can check the application installation by executing the following commands on terminal/command prompt

```
> conda activate insurit_env
> insurit --help
```

The --help is expected to list a help page to start the application. In case this fails we can still run the application by running the __main__.py file directly.

To start the application we can run the following command and provide it with username and password to the database.

```
> insurit connect
```

The *logout* and *exit* commands can be used to logout of a user and exit the application respectively.

Application Description

There are 3 main types of end users for the application - customer, agent, and administrator. Each of these users have access to features according to their permission levels with Admin being the highest, then agent, and customer. The admin user role is the only one who can manipulate the database from the application to reset data. Below are a list of features for each user role along with their commands.

Root/Administrator:

To login to Insurit as a root we can execute the following command after starting the application
> *root -u root -p root*

The default username and passwords are root.

As a root user the following operations can be performed.

- *Manage Agents:* Managers have the authority to manage agents, including the addition of new agents to the system and overseeing their performance.
  Commands:
  > *agent --new*
- *Access Performance Dashboard:* The application provides managers with a comprehensive performance dashboard, offering insights into agent performance and overall system metrics.
  Commands:
  > *agent --performance*
- *Manage Claims*: Managers can efficiently manage insurance claims, overseeing the processing, approval, and resolution of claims within the system.
  Commands:
  > *claims --pending*
  > *claims --approved*
- *Maintain Database:* Managers have full control over the database, allowing them to perform tasks such as resetting the database, ensuring data integrity and system functionality.
  Commands:
  > *database --reset RESET*

Agents:

To login to Insurit as an agent we can execute the following command after starting the application
> *agent -u stefa -p 'stefa cours'*

The default username is the first name of the agent in lowercase and the password is the full name in lower case. In the above command replace the username and password with any other agent's credentials to login as a different user.

As an agent user the following operations can be performed.

- *View and Manage Clients:* Agents can efficiently view and manage their client portfolio, providing them with a comprehensive overview of their customer base.
  Commands:
  > *client --new*
  > *client --show*
- *Calculate Premium*: Agents have a premium calculation feature, allowing them to assess and determine insurance premiums for potential customers based on various parameters.
  Commands:
  > *policy --calc-premium*
- *Identify Policies at Risk:* The application empowers agents to identify policies that are at risk, enabling proactive management and mitigation of potential issues.
  Commands:
  > *policy --at-risk*
- *Add New Policies:* Agents can easily add new policies to the system.
  Commands:
  > *policy --add-new*

Customers:

To login to Insurit as a customer we can execute the following command after starting the application
> *login -u rochette -p 'rochette dowglass'*

The default username is the first name of the agent in lowercase and the password is the full name in lower case. In the above command replace the username and password with any other customer's credentials to login as a different user.

As a client user the following operations can be performed.

- *View Policies:* Customers can easily access and view details of their insurance policies, providing transparency and ensuring they stay informed about their coverage.
  Commands:
  > *my-policies*
- *Pay Premium:* The application enables customers to conveniently pay their insurance premiums online, ensuring a seamless and efficient payment process.
  Commands:
  > *pay*
- *Create New Claim:* Customers have the functionality to initiate and submit new insurance claims through the application, streamlining the claims process.
  Commands:
  > *claim --new*
- *Check Claim Status:* Customers can effortlessly check the status of their insurance claims, offering real-time updates on the progress and resolution of their claims.
  Commands:
  > *claim --view*

- *Maintain Account Information:* Customers have the ability to manage and update their account information, including email addresses, phone numbers, and addresses, ensuring accurate and up-to-date records.
  Commands:
  > *account-settings*
  Or use one of the below options to skip menu
  > *account-settings [--email* ||*--phone* ||*--address]*

Future Scope

In the future, this insurance database management project has the potential to expand its capabilities by incorporating enhanced user interfaces for improved accessibility. The integration of machine learning algorithms could facilitate predictive analytics, offering insights into customer behaviors and optimizing premium calculations. Expanding the analytics and reporting capabilities by incorporating more advanced data visualization tools. This can include real-time analytics, trend analysis, and predictive modeling to empower decision-makers with deeper insights. Implementing chatbots or virtual assistants could enhance customer interaction, while policy customization features and multi-channel communication would provide a more personalized and informed insurance experience. Ongoing efforts to strengthen cybersecurity measures will be crucial for maintaining the trust of users. These future enhancements aim to make the system more sophisticated, adaptive, and aligned with evolving industry standards and technological advancements.

Bibliography

1. Home Loan prediction | Kaggle
2. https://www.kaggle.com/code/lagabralla/credit-score-analysis-visualization/input
3. https://www.mockaroo.com
4. Vehicle Insurance EDA and boosting models | Kaggle
5. Retail Transaction Data
6. https://data.gov/