**Database Systems Project Part III -**

**Logical Schema Optimization and Machine Learning Model Creation**

Ming Gao (mg7572), Wes Wang (lw2534)

# **EDA Physical Model Design**

## **1.1 EDA Physical Model**

We created and implemented a database using MySQL server locally and we managed to deploy it on the cloud using AWS RDS service. On our local server, we created all the table schemas, with appropriate data types and restrictions. We also enforced all foreign key constraints and ensured all foreign keys are referencing the correct column in another table. We added indices where we saw fit, suiting our actual business needs.

To illustrate, see below for the overall view of our MySQL local database. Note that in our logical design, for certain relationship tables, we used a hyphen to indicate that it is a relationship table. For instance, Account-Account is a recursive relationship on Account table itself. However, to comply with MySQL database restrictions we switched to all underline instead of hyphens. Additionally, some relationship tables has its own names in the given comprehensive schema in the first place. Therefore, we chose to keep their names. Please refer to our Part II report for a more straightforward illustration regarding naming choices.

A screenshot of a computer

Description automatically generated

Please refer to the following image regarding the content that will be discussed in this paragraph. Take the “account” table as an example, it has its primary key “Account\_id”, which according to our logical design is a generated artificial integer primary key. Here in MySQL we set it to be self-incrementing integer primary key. We also implemented the foreign key referencing the “CompanyCode” column in the CompanyCode table. It appeared as bold text in the image below.

A screenshot of a computer

Description automatically generated

The same goes for all other entity tables. For our relationship tables, i.e. those which function as the middle tables in many-to-many relationships, all of them have two foreign keys referencing the participating entities. Additionally, some of them have their own other attributes. For instance, please refer to the following image. Other than the two foreign keys, the table has 16 attributes supporting this many-to-many relationship.

A screenshot of a computer

Description automatically generated

We also implemented a few indices as we saw fit. For instance, we added an index for the “CompanyCode” foreign key column as we believe for our business needs we require our queries to respond fast to our high frequent filtering based on companies. We also expect to perform lots of JOINs based on this column as it is a popular foreign key among some of the most essential tables in our database. Please refer to the following image to see how it is implemented in our database.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

We implemented a bunch of other indices based on our needs. Please refer to the “create\_indices.sql” file in our submission to view them. Below we included a comprehensive list of indices and a brief reasoning of why we created each.

**idx\_account\_companycode:** This index on the CompanyCode foreign key in the Account table will optimize joins with the referenced CompanyCode table.

**idx\_contract\_companycode:** Same explanation as above - improves joins from Contract table to CompanyCode table.

**idx\_contractbenefit\_contractnumber:** Index on foreign key to optimize joins from ContractBenefit table to Contract table based on the primary key ContractNumber.

**idx\_contractpremium\_contractbenefitid:** Index to optimize joins from ContractPremium table to ContractBenefit table.

**idx\_managercontract\_associateid:** Index on foreign key referencing Associate table to allow faster joins.

**idx\_managercontract\_sitcode:** Index on the primary key SitCode to speed up lookups and joins on this column.

**idx\_associate\_associateid**: Index on primary key of Associate table for fast lookups.

**idx\_customer\_customerid**: Same as above - index on primary key of Customer table.

**Composite Indexes (idx\_acctadmin\_accountid\_adminid, idx\_accountbilling\_accountid\_billingid, idx\_accountmember\_customerid\_accountid):** These indexes cover the foreign keys from both sides of each junction table to optimize joins from either parent table.

Finally, we deployed this database on an AWS RDS service, please refer to the following image to see our deployment certificate (crucial information is blurred to prevent security issues). It is deployed at US-EAST OHIO. It is estimated to cost 14.84 US dollars per month.

A screenshot of a computer

Description automatically generated

## **1.2 Use Cases and Workflow**

To obtain an insurance quote and policy, we leveraged the schemas in Project 2 and here is an example of the workflow of one use case step by step:

Step 1: Identify Customer and Account

Use the Customer table to get the customer's details (Customer\_id, CusFirstName, CusLastName, etc.). Join this with the Account table using Account\_Member middle table to link Customer\_id and Account\_id.

Step 2: Choose Insurance Contract

Select a contract from the Contract table, which contains details like ContractNumber, CompanyCode, CoverageType, etc.

Step 3: Determine Benefits and Premiums

Join the Contract table with ContractBenefit using ContractNumber to identify benefits (ContractBenefit\_id). Then, join ContractBenefit with ContractPremium using ContractBenefit\_id to find premium details (PremiumCode, AnnualizedPremium, etc.).

Step 4: Associate Billing Account

Use the BillingAccount table and join it with the Account table using Account\_BillingAccount middle table to link Account\_id and BillingAccount\_id, which provides billing details (BAcctName, Address, etc.).

Step 5: Finalize Policy Details

Combine all the obtained information (customer, account, contract, benefits, premiums, billing) to create a comprehensive insurance policy and quote.

Step 6: Record Policy in Database

Insert the finalized policy details into relevant tables like Customer, Account, Contract, ContractBenefit, ContractPremium, and BillingAccount.

# **Machine Learning Model Creation**

## **2.1 Background**

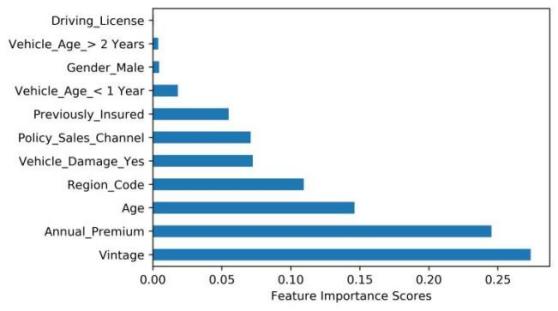
Based on the Project Part II, we further kept developing our data lake to compile various datasets. We collected a new insurance-related dataset to refine and complement the use cases in the Project Part II which will be demonstrated below.

An insurance company that has provided Health Insurance to its customers now they need some help in building a model to predict whether the customers from past year will also be interested in Vehicle Insurance provided by the company. Building a model to predict whether a customer would be interested in Vehicle Insurance is extremely helpful for the company because it can then accordingly plan its communication strategy to reach out to those customers and optimize its business model and revenue. Therefore, the company aims to make precise predictions via machine learning models to target potential customers who may be interested, in order to promote their products more efficiently and save costs.

According to the requirements, we set our goals as leveraging various machine learning models and figuring out the optimal model to predict Health Insurance owners who will be interested in Vehicle Insurance.

## **2.2 Data Processing**

The original dataset contains 381,109 records with 11 features. We dropped feature *id* since it has no practical meaning. And we performed One-Hot Encoding to process 3 categorical features (i.e. *Male*, *Vehicle Age*, *Vehicle\_Damage*). Then, we leveraged the Extra-Trees classifier for feature selection. The model calculated importance scores for each feature, which would be shown in Figure 1.



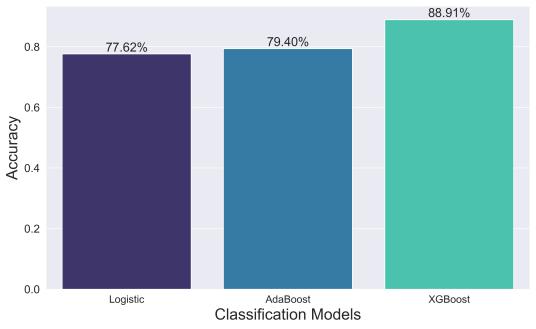
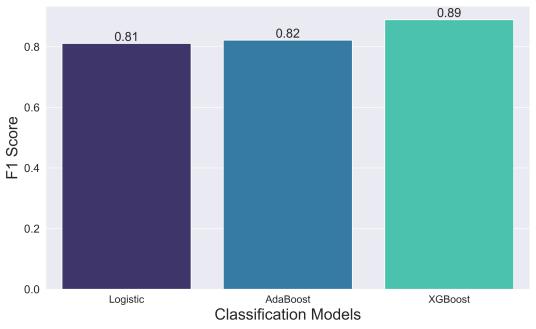
**Figure 1. Feature Importance Scores**

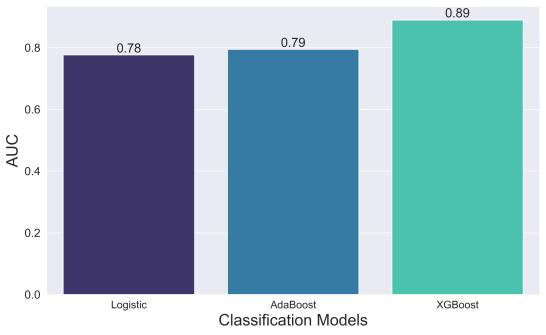
We can clearly see that the importance scores of the features under *Vehicle\_Age\_<1 Year* were all over 0.05, so we only used these top 7 important features as the independent variables for modeling. The dependent variable was *Response* where “1” represent the customer was interested and “0” represent the customer was not interested.

We also noticed that the processed data was still imbalanced that the ratio of the number of uninterested customers to the number of interested customers was approximately 7:1. Therefore, we applied SMOTE algorithm, a kind of up-sampling methods, to process the imbalanced data. The final dataset contained 668,798 records and the ratio mentioned before was 1:1. And we split the final dataset into a training set and a test set as the ratio of 7:3.

## **2.3 Modeling**

This was a binary classification problem, so we leveraged 3 advanced classifiers (Logistic Regression, AdaBoost, and XGBoost) to predict whether a customer would be interested in Vehicle Insurance. To evaluate the model performance, we calculated accuracy, F1 scores, and AUC on the test set for each model. The modeling results are as follows.



**Figure 2. Modeling Results**

From the modeling results, it's clear to see that the XGBoost model achieved the highest accuracy, F1 score, and AUC among three classifiers.

Therefore, the XGBoost classifier is the optimal model for predicting Health Insurance owners who will be interested in Vehicle Insurance. The insurance company can leverage this classifier to make precise predictions and accordingly plan its communication strategy to reach out to those customers and optimize its business model and revenue.

# **Appendix**

The source of the dataset used for machine learning is as follows:

<https://www.kaggle.com/datasets/anmolkumar/health-insurance-cross-sell-prediction/data>