Providing Unsecured Loans to Customers

Nihit Natu
SUNY Buffalo
New York, USA
nihitume@buffalo.edu

Sai Kumar Aindla
SUNY Buffalo
New York, USA
saindla@buffalo.edu

ABSTRACT

The number of loan defaulters are increasing all over the world and the banks need a secure system to thoroughly check the customer background before granting him a loan. In this project, we try to implement a system that will check through the customer current information like bank balance, credit limits, mortgages etc. and calculate the possibility of the customer defaulting the loan. We have implemented the databases in Postgres to store the customer information and written functions in Javascript to calculate the credit scores of the customers which helps in deciding the defaulting percentage of the customer.

CCS CONCEPTS

• **Database Systems**;

KEYWORDS

Bank, Credit Scores, Loans, Postgres, Javascript.

ACM Reference Format:
Nihit Natu and Sai Kumar Aindla. 2020. Providing Unsecured Loans to Customers. In *Proceedings of ACM Conference (SUNY Buffalo).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The number of cases of defaulting on loans have increased over the past few years. The main reason behind this is the lack of background check done by the banks on the customer. The bank needs to take into consideration additional factors apart from the one they consider currently to ensure these cases reduce.

A bank takes in consideration many relevant details of their customer while offering an unsecured loan because there won't be any kind of collateral documents that are needed to submit from the customer's side. The details such as customer's credit history, web browsing data, loans which he already took or any other personal data that helps him in getting an unsecured loan. This project proposes an efficient way for the bank to query the customers data and decide whether they can offer him this unsecured loan or not.

SUNY Buffalo, May 2020, Buffalo, NY, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 DATA

Partial data was taken from a kaggle dataset called 'Default of Credit Card Loans'. Since the data contains only some of the attributes that we have mentioned in our relational schema the remaining attributes were filled by writing a python script that generated some random data and filled in those attributes. The kaggle dataset mainly consists of transactions made by the customer over the past few months and his credit card bill payment activity. We need this data for the credit card schema in our database. It also contains some user information like name, address and age which will be used for customer schema. The details in the mortgage schema were filled by the script based on the data types and the range of values each attribute can take in the schema. All the data generated were converted into csv files and these files were then uploaded into the Postgres database directly after creating the relation which would be explained in the next section.

3 RELATIONS

Based on the requirements we used 4 relations in our system. The relations are Customer Details, Credit Card, Mortgage and Web browsing data. Further we divided the Customer table into Customer and Account tables because the initial Customer table was not in BCNF form. So for the final database we now have 5 relations. The attributes of each relation are given below:

3.1 Customer Details

This schema is used to store the details of all the customers in the bank.

(Customer_ID, Name, Age, Address, Account_Number)

In this relation, Customer ID is the primary key. This is not in BCNF form and this may cause problems while insertion or deletion of the data. So this relation is split into 2 and the resulting relations are Customer and Account

Relation 1: Customer
(Customer_ID, Name, Age, Address)

Relation 2: Account
(Customer_ID, Account_Number)

Now both of these relations are in 2NF. These relations are also in 3NF and BCNF form.

3.2 Credit Card

This schema stores the credit details associated with a customer. This data is used by the functions to calculate the credit score of the customer.

Relation 3: CreditCard
(Customer_ID, CCLimit, OtherCards, LastAnnPaid, LastCreditUpgrade, CardType, MonthlyBalChange)

In this Customer_ID is the foreign key. This schema does not have another primary key and the foreign key itself is used to identify the tuples uniquely. This relation is in 2NF. This relation is also in 3NF since none of the non prime attributes can determine another non prime attribute and it is also in BCNF so no changes need to be made to the original schema.

3.3 Mortgage

This schema stores the mortgage details associated with a customer. It also stores if any ongoing loan details are there for a customer. This data is used by the functions to calculate the mortgage score of the customer.

Relation 4: Mortgage
(Customer_ID, CurrentLoans, LoanAmount, AmountPaid, AmountRemain, InterestRate, LoanType, LoanPeriod, Mortgages, TotalBalance)

In this Customer_ID is the foreign key. This schema does not have another primary key and the foreign key itself is used to identify the tuples uniquely. This relation is in 2NF. This relation is also in 3NF since none of the non prime attributes can determine another non prime attribute and it is also in BCNF so no changes need to be made to the original schema.

3.4 Web Data

This schema is used to store the web browsing details associated with a customer like the amount of times he visited the bank website.

Relation 5: Web_Browsing
(Customer_ID, Device_id, Tot_time_spent, No_of_Visits, Tot_applications, No_products_Seen, Search_Query)

The candidate keys for this relation consists of only Customer_ID since that's the only attribute that can uniquely determine a row in the schema. Since there is no transitive dependency the relation is in 3NF and it is also in BCNF since CID is the only key in this schema.

4 DATABASE

For this project we used PostgreSQL for storing all the data related to the bank. Postgres is a free and open-source relational database management system (RDBMS) emphasizing extensibility and SQL compliance [1]. We used the interface PgAdmin provided by postgres to upload all the data into the schemas. The data was directly imported from the CSV files generated from the Psql shell which will come along with the PgAdmin tool.

Apart from that we also used a NoSql database to host our tables in the Mongodb which is shown in the results section.

5 IMPLEMENTATION

We used the Postgres as our database for the complete project. For creating the website we used Express server framework and used "Javascript" to access the postgre database in our backend. And we have used ReactJS as our frontend language. We have five relations in our postgres database as we already mentioned above and accessed those relations by connecting both backend with the frontend. We have implemented this website in our local system, so the ip address will be our localhost:<port_number>(ex: localhost:3001).

We performed basic queries like Insertion, Updation and Selection on the dataset and these queries were the most efficient ones in terms of computation. The computation was mainly calculated based on the time required to execute the query.

Our application calculates credit scores to decide which is one the main factors in determining whether the customer will default or repay his loan. The credit score calculation takes most of the information in the Credit Card Schema and a function is written to calculate the score based on the values obtained from the Credit Card Schema. This query of calculating the credit score was the least efficient in terms of time required to fetch the results. We used the EXPLAIN and EXPLAIN ANALYZE to find the time complexity related to queries.

The following results were obtained when we used it on the query to calculate the credit score:
Planning Time: 0.112 ms
Execution Time: 0.285 ms

5.1 Indexing

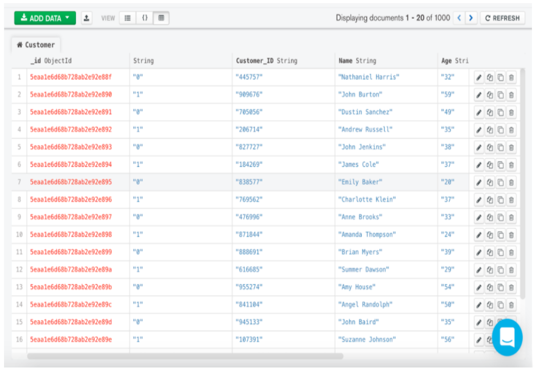
Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.The index is a type of data structure. It is used to locate and access the data in a database table quickly.

The time taken for some of the queries like credit score calculation is considerably more and can be reduced in order to optimize the system performance. Indexing was used in order to improve the execution time of this query. The credit score requires 5 attribute values from the CreditCard schema namely CC limit, Last Annual Fee paid, Last credit upgrade, Card type and Monthly balance change. We indexed The Customer_ID column which is the primary key for the schema so that the time required to access the record is reduced.

After indexing the following results were obtained:
CREATE INDEX find_customer ON CreditCard(customer_id)
Planning Time: 0.097 ms
Execution Time: 0.154 ms
There is a significant reduction in the execution time and indexing has helped to get a better time efficiency for the database system.

6 NOSQL DATABASE

We also hosted the database using a NoSQL database system. We implemented this using MongoDB. A NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases [3]. MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schema [2]. We used the interface Mongo Compass provided by MongoDB for displaying the data and performing the queries. The results for the same can be seen in the figure.



	_id ObjectId	String	Customer_ID String	Name String	Age Stri
1	5eaa1e6d68b728ab2e92e88f	"0"	"445757"	"Nathaniel Harris"	"32"
2	5eaa1e6d68b728ab2e92e890	"1"	"909676"	"John Burton"	"59"
3	5eaa1e6d68b728ab2e92e891	"0"	"705056"	"Dustin Sanchez"	"49"
4	5eaa1e6d68b728ab2e92e892	"1"	"206714"	"Andrew Russell"	"35"
5	5eaa1e6d68b728ab2e92e893	"0"	"827727"	"John Jenkins"	"38"
6	5eaa1e6d68b728ab2e92e894	"1"	"184269"	"James Cole"	"37"
7	5eaa1e6d68b728ab2e92e895	"0"	"838577"	"Emily Baker"	"20"
8	5eaa1e6d68b728ab2e92e896	"1"	"769562"	"Charlotte Klein"	"37"
9	5eaa1e6d68b728ab2e92e897	"0"	"476996"	"Anne Brooks"	"33"
10	5eaa1e6d68b728ab2e92e898	"1"	"871844"	"Amanda Thompson"	"24"
11	5eaa1e6d68b728ab2e92e899	"0"	"888691"	"Brian Myers"	"39"
12	5eaa1e6d68b728ab2e92e89a	"1"	"616685"	"Summer Dawson"	"29"
13	5eaa1e6d68b728ab2e92e89b	"0"	"955274"	"Amy House"	"54"
14	5eaa1e6d68b728ab2e92e89c	"1"	"841104"	"Angel Randolph"	"50"
15	5eaa1e6d68b728ab2e92e89d	"0"	"945133"	"John Baird"	"35"
16	5eaa1e6d68b728ab2e92e89e	"1"	"107391"	"Suzanne Johnson"	"56"

The data was loaded into MongoDB directly from the CSV files.

7 EXECUTION

For running this whole project in your system, you need to have node js or npm packages so that this whole system works in your local system. You need to set up the local postgres database before starting to run the website in your system. After that, two shells are needed, one for running backend and another for frontend. Each will run in one port each, say if the backend is running on localhost:3000, then the frontend will run on localhost:3001.

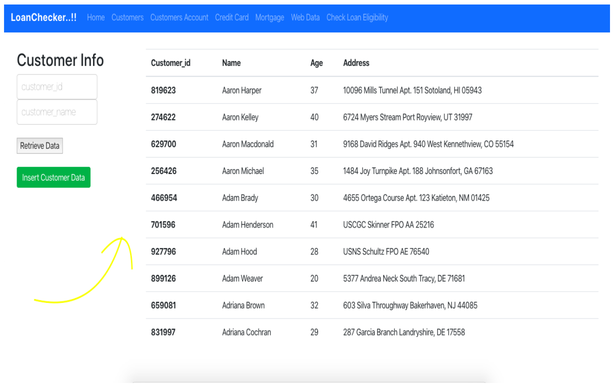
8 RESULTS

We implemented the basic query operations like insert data, select a specific record and displaying first 10 rows for each table in the website.

8.1 Selection

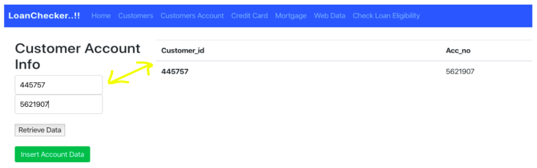
Selection is an operation of displaying all the data in a given schema. The selection can also be filtered based on a selected attribute. The filtering attribute used in this project is the primary key used in all the table Customer_ID.

Below figure shows the selection results of the first 10 rows ordered by name of the customer in customer table.



Customer_id	Name	Age	Address
819623	Aaron Harper	37	10096 Mills Tunnel Apt. 151 Sotoland, HI 05943
274622	Aaron Kelley	40	6724 Myers Stream Port Royview, UT 31997
629700	Aaron Macdonald	31	9168 David Ridges Apt. 940 West Kennethview, CO 55154
256426	Aaron Michael	35	1484 Joy Turnpike Apt. 188 Johnsonfort, GA 67163
466954	Adam Brady	30	4655 Ortega Course Apt. 123 Katieton, NM 01425
701596	Adam Henderson	41	USCGC Skinner FPO AA 25216
927796	Adam Hood	28	USNS Schultz FPO AE 76540
899126	Adam Weaver	20	5377 Andrea Neck South Tracy, DE 71681
659081	Adriana Brown	32	603 Silva Throughway Bakerhaven, NJ 44085
831997	Adriana Cochran	29	287 Garcia Branch Landryshire, DE 17558

Retrieving a specific record from the table customer based on the customer id.

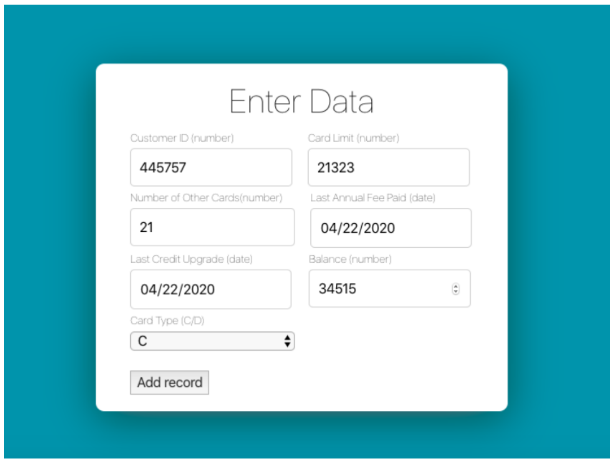


Customer_id	Acc_no
445757	5621907

8.2 Insertion

Insertion is an operation for creating new records in the schemas existing in the Postgres database. The insertion operation ensures that all the data have a unique id associated with them that is the Customer_ID.

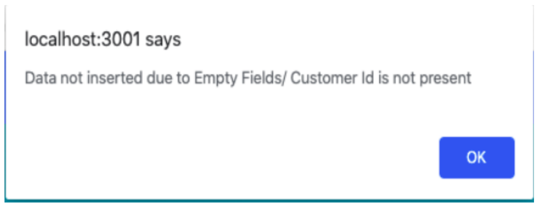
Example of insertion data in CreditCard relation is shown in the figure.



Enter Data

Customer ID (number): 445757	Card Limit (number): 21323
Number of Other Cards(number): 21	Last Annual Fee Paid (date): 04/22/2020
Last Credit Upgrade (date): 04/22/2020	Balance (number): 34515
Card Type (C/D): C	

An error message is displayed if the data is not inserted due to violations of the restrictions of having a unique Customer_ID.

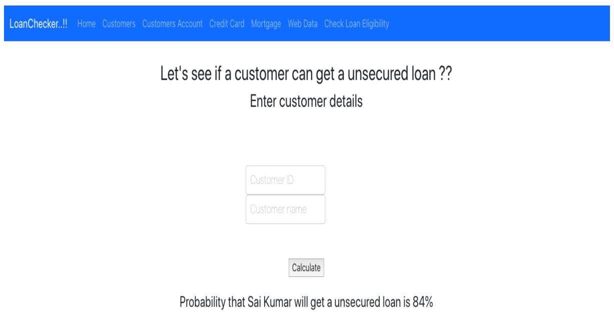


localhost:3001 says

Data not inserted due to Empty Fields/ Customer Id is not present

8.3 Defaulting Probability

Whenever a customer applies for a loan the bank can check for the probability that the customer will repay the loan. We have created a function in the backend that calculates this based on the current customer data. This function takes into consideration the data from the Customer,Credit card and the Mortgage schema. Each attribute in these schema are assigned a weight based on the values and the probability is generated based on the summation of these attribute values. Below Figure shows a customer probability of getting unsecured loan.



Let's see if a customer can get a unsecured loan ??

Enter customer details

Probability that Sai Kumar will get a unsecured loan is 84%

This probability might help the bank before giving the loan to the customer.

9 CONCLUSION

This project aims at developing a new application that might reduce the bank defaulters. We have designed a system that takes into consideration the present data in the bank database instead of using any additional data. This system would certainly decrease the number of bank defaulters and make this process simple and fast for the banks to check the background information of the customer.

10 REFERENCES

(1) PostgreSQL (https://en.wikipedia.org/wiki/PostgreSQL)
(2) MongoDB (https://en.wikipedia.org/wiki/MongoDB)
(3) NoSQL (https://en.wikipedia.org/wiki/NoSQL)