**Online Banking Database Design**

**TABLE OF CONTENTS**

1. Executive Summary 3
2. Introduction 4
3. Logical Design 5
4. Physical Design 7
5. Sample Data 8
6. Queries 9
7. Future extensions and lessons learnt 16

**EXECUTIVE SUMMARY**

The purpose of this project is to develop an online banking database design that provides banks with the facility to organize information related to the employees, customers and other relevant information in an efficient manner. A database design is an important part of any system. Application design traditionally consists of two steps: you develop a logical model of the business process you're automating, then you map that model to the database by creating a physical model, which is implemented as a series of tables. Thus, a database design makes an impact on the overall efficiency of a system.

In this technologically advanced era, most of the banking transactions are done online. Hence, this is an increasingly interesting topic to study as this project deals with banking database design. This report consists of information on logical and physical design of the database, tables created, queries used to generate different views. Recommendations to improve the database and problems faced and lessons learnt are also a part of this report.

**INTRODUCTION**

In this project, we have developed a database for an online banking system. This database has been created to assist the banks to save the information on customers, employees, various accounts and transactions in an organized manner. This database shall help banks to maintain their information at one place and provide an overview of the bank operations. Additionally, banks can generate several meaningful reports that will help them to improve customer service and banking transactions.

This database is designed such that it could be incorporated when developing a fully operating online banking systems in real world. Although this would require adding many more entities to the database, this project can be used as a baseline while defining the database for such a system. A good database design is essential for efficient performance of a system. The database is designed so that it

The database could also be used to perform statistical analysis for decision making such as what type of loans are mostly granted to customers, what should be the ideal rate of interest taking into various parameters such as customer income etc. This kind of analysis is important for banks so as to ensure smooth functioning.

For the scope of this project, the aspects that were focused on were - banking transactions, customer information, employee information, bank branch details, information about loans, customer account details.

The project has explored various relations that exist between different tables. A database devoid of redundancies was developed to provide data integrity solutions to the banking systems. Create, Read, Update, Delete operations have been performed on the database. Different views have been created so that the banks can view the records for specific information requested. To eliminate redundancies data normalization was carried out.

This report will further delve into the details of the different stages of the database development.

**LOGICAL DESIGN**

A logical design is a conceptual, abstract design. The process of logical design involves arranging data into a series of logical relationships called entities and attributes. An *entity* represents a chunk of information. In relational databases, an entity often maps to a table. An *attribute* is a component of an entity and helps define the uniqueness of the entity. In relational databases, an attribute maps to a column.

Generally, the first stage of a database design is to develope an Entity-Relationship diagram to describe information needs or the type of information that is to be stored in a database. An ER diagram is used as a blueprint to build a physical database. It is a graphical representation of the information system, its entities and various relationships that can exist between them. An entity represents a table in the relational database. During the development of this project, ER diagram helped in identifying the important entities and mapping the requirements in a more clear and concise manner.

For the initial ERD, the basic requirements of the online banking system were captured and below entities were identified:

1. Employees
2. Branches
3. Customers
4. Accounts
5. Credit\_Cards
6. Banking\_Transactions
7. CC\_Transactions
8. Loan\_Details

Once the tables were identified, attributes for each table were defined. Primary keys were established to ensure every table has a unique identifier column for their records. Following relationships between the tables were defined.

* Many to many relationship: This type of relation was established between Branches and Employees, Accounts and Customers. An employee could work for any number of branches and a branch could have any number of employees. Similarly, a customer could have many accounts and an account could be a joint account between more than two customers.

These relationships were resolved in MySQL Workbench automatically and two more

identities were generated as follows -

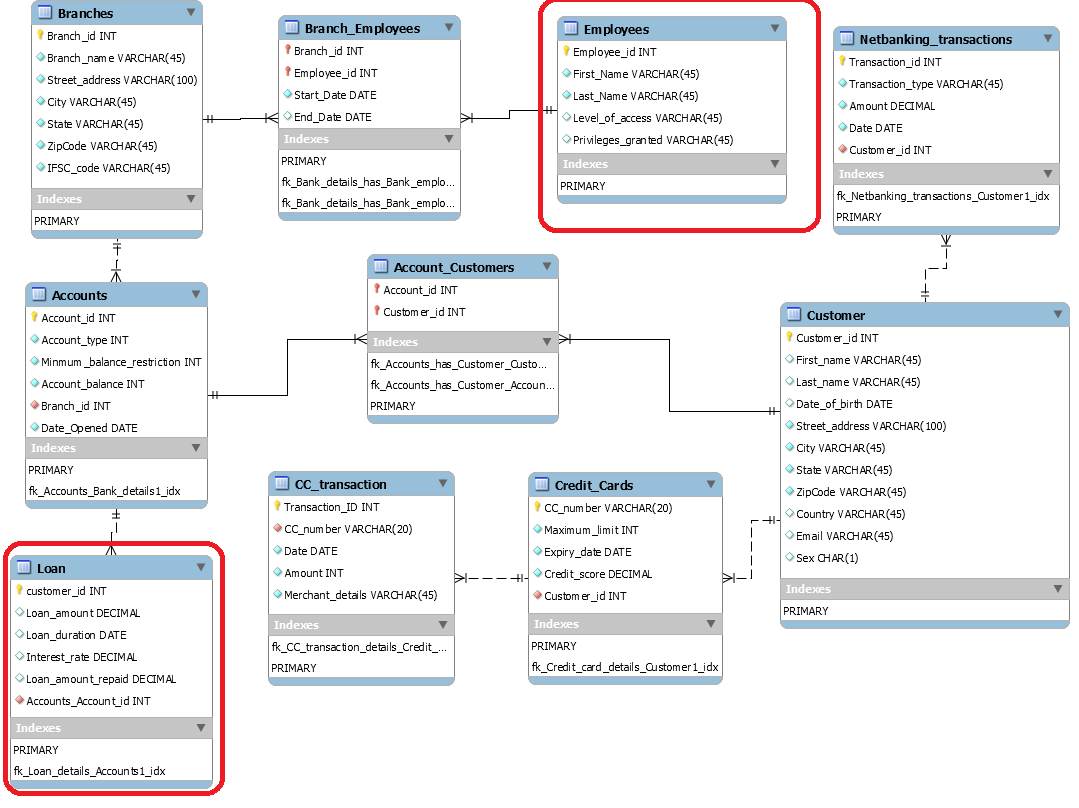
1. Account\_Customers
2. Branch\_Employees

Since the primary key of the source table is also the primary key of join table, the relation

was thought to be an identifying relation.

* One to many relationship - This type of relation was established between Branches and Accounts, customer and banking transaction, customer and credit cards, credit cards and cc\_transactions, accounts and loan. These relationships were non-identifying relationships as the primary key of the parent table is only used as part of the foreign key in the child table.

During the process of ERD design, it was ensured that the primary keys are auto incremented and unsigned variables and variable types are consistent, table names and variable names are clear and consistent. Variables which cannot have blank/null values were marked “Not Null”.



Fig(a). ERD\_Initial\_Version

However during the physical design of the database a few issues were identified and there was a need to alter the entities. Some of the relations were wrongly identified, many attributes were missing and certain attributes needed to be modified to match real world conventions. This has been further explained in the next section. The final version of the ERD was reverse engineered from the physical database.

**PHYSICAL DESIGN**

During the physical design process, data gathered during the logical design phase is converted into a description of the physical database, including tables and constraints. ERD was forward engineered and converted to physical database.

When database was being populated an issue was identified in the Accounts table. Account\_minimum balance attribute was not defined in the account table, also the account\_type attribute was repetitive. Hence another table account\_type was created which had- account\_minmum\_balance and account\_type as the attributes. One to one non-identifying relationship between accounts and account\_type was established.

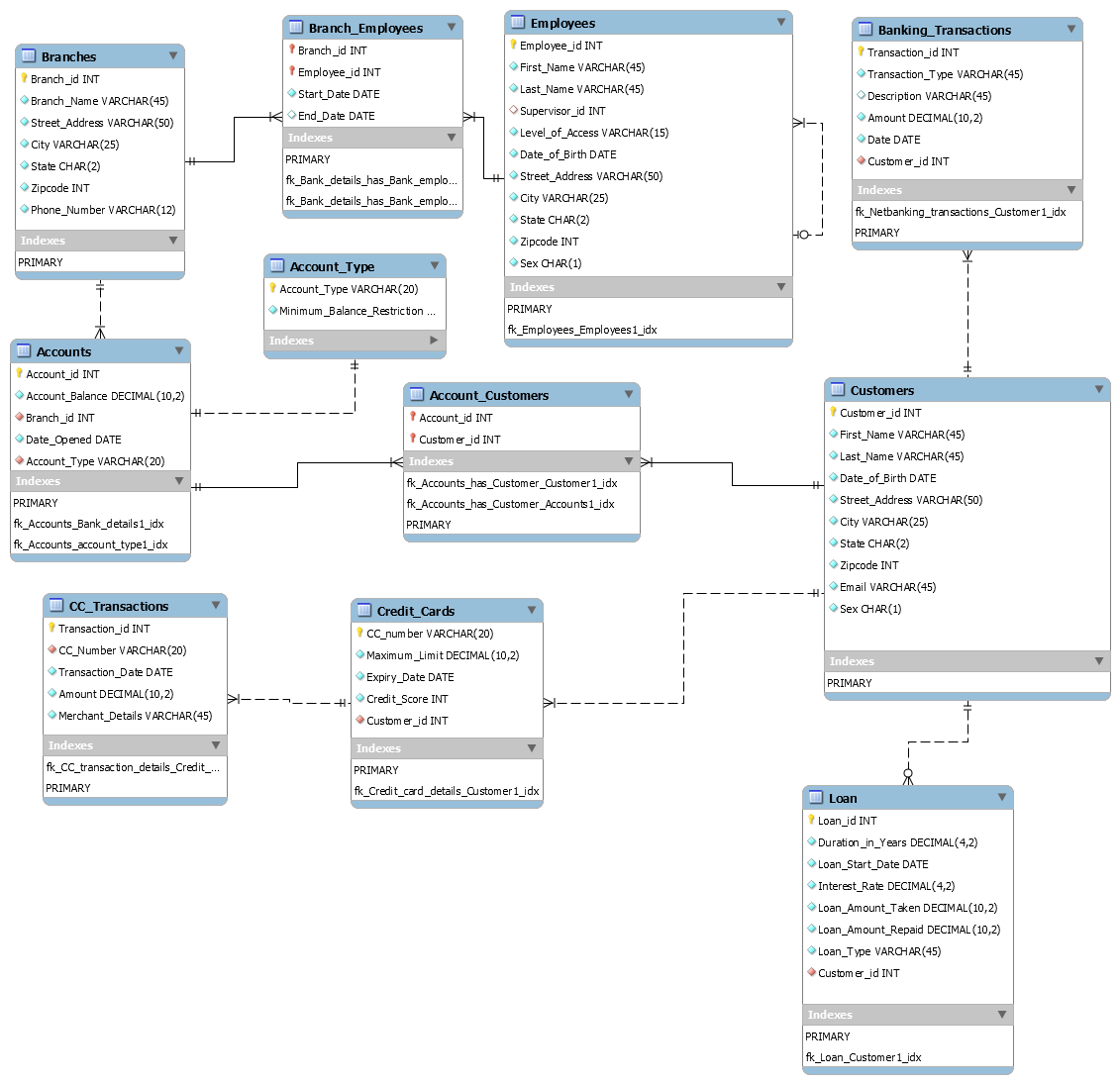
In the ERD, a relation was incorrectly established between the accounts and loan tables. Ideally it should have been between customers and loans as a customer takes loans. And it wasn’t necessary that all customers had taken loans. Thus, there was supposed to be non identifying one to many relation between customers and loans with zero or 1 cardinality associated with loans table. The attributes in the loan table were modified so as to contain customer\_id as the foreign key, loan\_id as the primary key. Some of the attribute names were changed to make it more understandable.

For employees, there were many missing attributes such as date\_of\_birth, supervisor id, street\_address, city, state, zipcode, sex. These attributes were added to the table. Privileges\_granted was a redundant attribute that was removed. The level\_of\_access attribute was sufficient to convey what privileges were granted to the employee accessing the database. Technically, a supervisor is also an employee. It might be possible that the employee won’t have any other supervisor. So a self non-identifying relation was established in this table with zero or 1 cardinality associated with supervisor\_id.

Some of the data types were changed and made uniform across the tables.

* Sex was defined as CHAR(1)
* State as CHAR(2)
* Street\_Address as VARCHAR(50)
* City as VARCHAR(25)

Once the tables were altered and populated with data, reverse engineering was performed to obtain the final ERD.



Fig(b) ERD\_Final\_Version

**SAMPLE DATA**

The database uses hypothetical data. Below websites were used to generate random data:

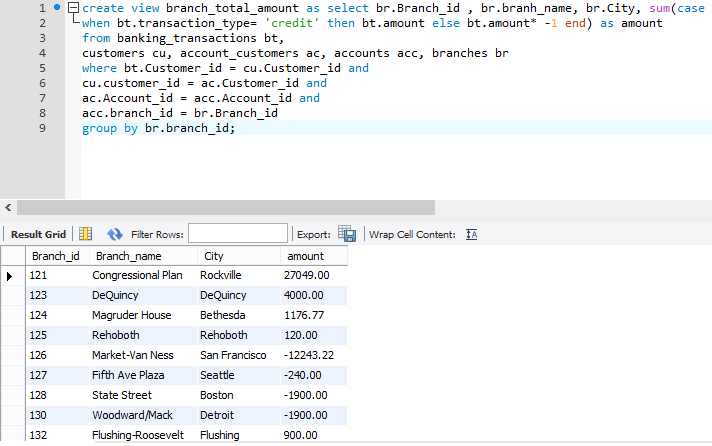
* <http://www.generatedata.com/>
* <https://www.randomlists.com/random-addresses>

Through these websites, random information is generated. Variables - ID’s, names, addresses, date of birth, account balance etc. have been generated through these websites. Remaining attributes have been populated manually.

**QUERIES**

1. Obtaining branch-wise total amount in the bank:

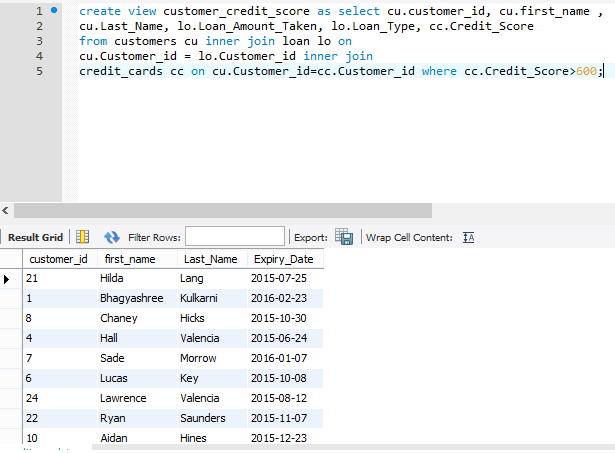
Here, banking transactions per customer belonging to the same branch are identified, then if the transaction type is credit the amount is added and if it is debit the amount is subtracted from the total amount. Here, equi join is performed on banking\_transactions, customer, account\_customers, accounts and branches. Select query displays branch information- branch\_id and total bank branch amount.



fig(1): Branch\_total\_amount

2. Customers who have a loan and having credit score above 600:

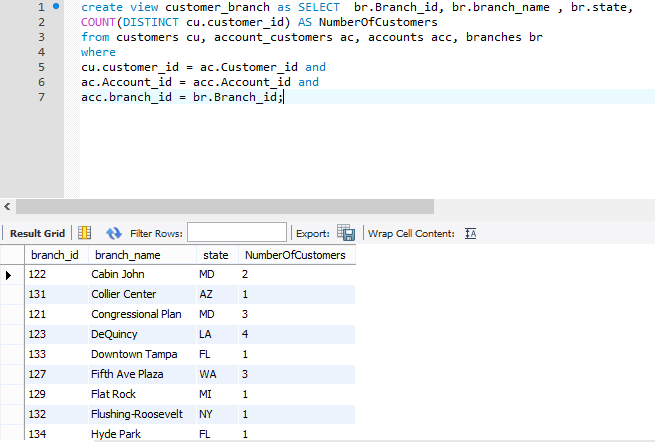
Here, inner join is performed on customers and loan and credit cards table on customer\_id. Customers who have taken loans and who have credit score higher than 600 are displayed. Select query displays customer\_id, first\_name, last\_name, loan\_amount\_taken, loan\_type, and credit\_score.



fig(2): customer\_credit\_score

3. Number of customers present in a bank branch

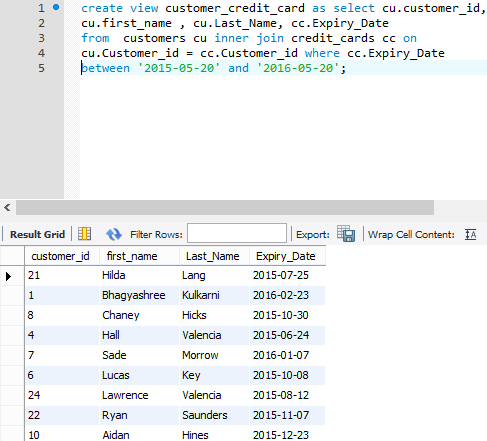
Here equi join is performed on customers, accounts, account\_customers, branches. DISTINCT keyword is used to get unique customer\_ids. Select query displays branch\_id, branch\_name, state and number of customers.



fig(3): customer\_branch

4. Customers with credit card information between 2015-05-20 and 2016-05-20

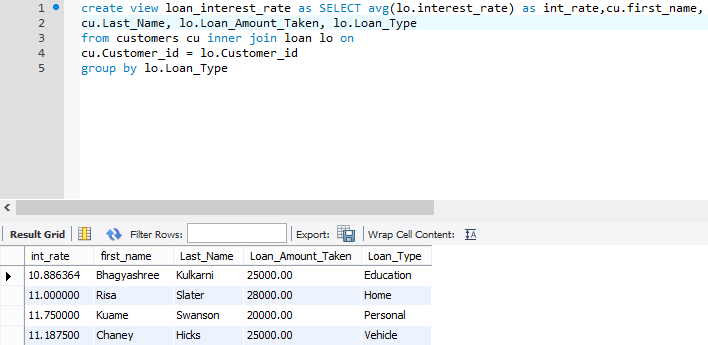
Inner join is performed on customers and credit\_cards. Customer\_id, first\_name, last\_name, expiry date of credit card is displayed.



fig(4): customer\_credit\_cards

5. Average interest rate for loan\_type

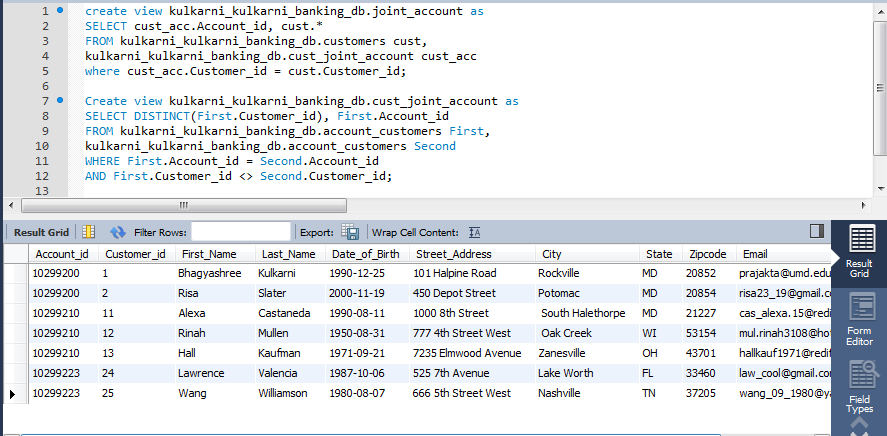
Here inner join is performed on customers and loan. avg function is used to calculate average interest\_rate and group by is used to group the loan\_types.



fig(5): loan\_interest\_average

6. Customers with joint account:

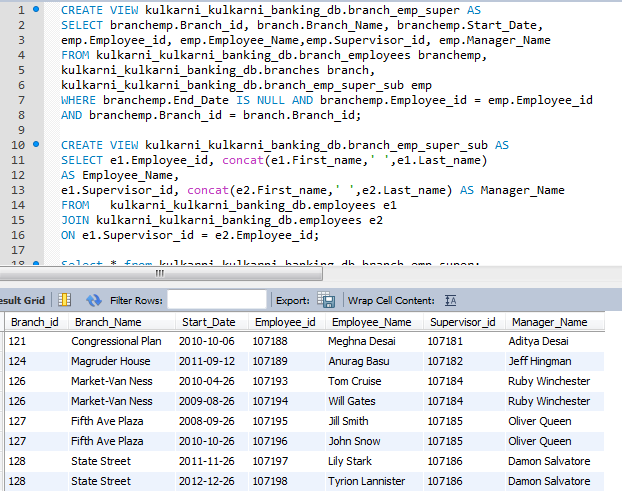
This query shows customers with joint account. Since we cannot create a view when there is a select statement in the FROM clause, we had to create a different view for it and then call it in the main view.



fig(6): joint\_account

7. Employees with supervisors and their current branch:

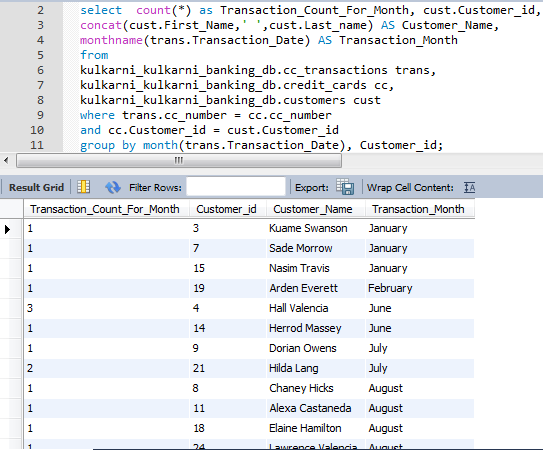
This query tells us the supervisors of the employees and the current branch the employee is working in. Since we cannot create a view when there is a select statement in the FROM clause, we had to create a different view for it and then call it in the main view.



fig(7): branch\_emp\_super

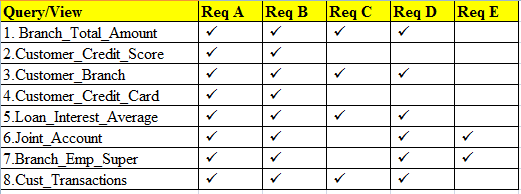
8. Credit card transactions of customers per month:

This query tells us the count of credit card transactions per month for the customers.



fig(8): cust\_transactions

**REQUIREMENTS SATISFACTION TABLE**



**FUTURE EXTENSIONS**

* The bank would have different types of users with different administrator access, users with lower level access who can only view a certain part of the database and not all the information.
* Different privileges will be granted to different levels of users to modify the database. At this point only a general access has been given to the records. This would increase security and would allow privileged access to the database.
* In the table branch\_employees, information about previous supervisor was not included. Information related to branch assets was also not considered.
* Employees’ information such as salary, job type was not taken into consideration.
* Some entities such customer\_complaints, check were not implemented.
* Banking operations such as withdrawal, deposit, atm transactions were not explicitly addressed and were included as a part of banking\_transactions table.

**LESSONS LEARNT**

* Table names should be small and precise to avoid confusion and unnecessary long names.
* When tables having FK are connected to another table, the FK is automatically included in the new table, this is not always needed. This is unnecessary extra work.
* The join tables can have attributes other than FK
* The identifying and non-identifying relations should be established appropriately.
* Columns of the same type should have
* It is easy to insert data in MySQL
* If database is altered after forward engineering from the ERD, it needs to be reverse engineered to get the new ERD.