HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

Faculty of Computer Science & Engineering

Bank Transaction Database System Assignment 1+2 Report

Instructor: Mr. Tran Minh Quang

Group: *3 - CC07*
Students: *Phan Tuan Khai – 1952780*
Cao Hoang Kiet - 2053165
Cao Tuan Kiet - 2053166

Ho Chi Minh City, November 2021

Contents

1 Member list & Workload 3

2 Instruction 3

3 Requirement description for Bank Transaction Database System 4
3.1 Brief Introduction 4
3.2 Requirement description 4

4 EER Diagram for our Bank Transaction Database System 5
4.1 About the tool we use 5
4.2 Our Diagram 6
4.3 Explanation 7

5 Logical design using relational database model for our database system 9
5.1 About the tool we use 9
5.2 Physical Design 10
5.3 Our relational DB model 10

6 Implement the DBs into a particular DBMS 12
6.1 Our Code 12
6.2 Result 15
6.2.1 We successfully run our code on MySQL Workbench . 15
6.2.2 The database created from our code 15

7 Conclusion for assignment 1 16

8 Complete implementation of the DB to DBMS 17
8.1 Customer Relation: 17
8.2 BankStaff Relation: 18
8.3 BankAccount Relation: 18
8.4 BankCard Relation: 19
8.5 CreditCard Relation: 19
8.6 DebitCard Relation: 20
8.7 CardTransaction Relation: 20
8.8 OnlAccTransaction Relation: 21

8.9 OffAccTransaction Relation: 21

9 The normal form (NF) and solutions for improving the NF 22
9.1 Informal Design Guidelines for Relation Schemas 22
9.2 The Normal Form 23
9.2.1 Functional Dependency 23
9.2.2 Definition of the Normal Form 23
9.2.3 First Normal Form(1NF) 24
9.2.4 Second Normal Form(2NF) 24
9.2.5 Third Normal Form(3NF) 24
9.2.6 Boyce-Codd normal form(BCNF) 24
9.2.7 Fourth Normal Form 25
9.2.8 Fifth Normal Form 25
9.3 Normal form of our relations 25
9.3.1 Dependencies 25
9.3.2 First Normal Form 29
9.3.3 Second Normal Form 30
9.3.4 Third Normal Form 30
9.3.5 Boyce-Codd Normal Form 30
9.3.6 Normal Form Conclusion 30

10 Database security and Solution 30
10.1 Definition of database security 30
10.2 Demonstration and Solution for Bank_Transaction database . 31

11 Tuning and Manipulating Bank_Transaction database 33
11.1 Retrieve data from our database 33
11.2 Trigger & Stored Procedure 34
11.2.1 Trigger for Online Transaction: 35
11.2.2 Trigger procedure for adding staff: 39
11.3 Indexing 42
11.3.1 Advantages of using indexes 42
11.3.2 Demonstration for using indexes in Bank_Transaction database 42

12 An application for our Bank_Transaction database 44

13 Conclusion for assignment 2: 54

14 References 54

1 Member list & Workload

No.	Fullname	Student ID	Percentage of work
1	Phan Tuan Khai	1952780	100%
2	Cao Hoang Kiet	2053165	100%
3	Cao Tuan Kiet	2053166	100%

2 Instruction

With most of the world operating remotely, online and offline banking transactions may create a huge help to our life and businesses using web services and mobile apps for online banking transactions as well as banking cards for transferring money from ATM to our hand

In addition to being able to bank at any time, from anywhere, there are many advantages to banking online.

- Accessibility: Internet banking is available at any time and it provides 24 hours access instead of visiting banks only during working hours.
- Customer Service: the customers do not have to stand in queues to carry out certain bank transactions.
- Pay bills online: This might be one of the top advantages of online banking because you don't have to take time out of your day to go to the bank. You can simply log into your account and pay your bill online right away.
- Transfer money: You may need to do a rapid money transfer to a client or vendor, or you may need to transfer money from one account to another.

However, the most important problem that we must encounter is security. Valuable information is always prone to hacks, but online banking tries for solving e-security threats.

In contrast, offline banking transaction is also the most famous transaction service in the world about approximately a few decades ago. If we compare its transaction with the first one(online transaction), we will see there are a lot of disadvantages when we use it in the technology era 4.0 and 19-covid pandemic which requires the use of the Internet everywhere. However, Traditional banking does not encounter e-security threats and this is a good benefit when trading offline transactions.

From the reason above, thus we choose to implement and design DB banking transactions in order to understand more about the management of banking transactions.

Our report will include a requirement description for bank transaction database system, EER Diagram of our design, relational database design and implementation of our code into the Database management system (DBMS).

3 Requirement description for Bank Transaction Database System

3.1 Brief Introduction

We choose reference most from OCB. From OCB, we get information about Account, Card. About Customer and Staff attributes, we take typical information about a human.

3.2 Requirement description

BANK_TRANSACTION database system in which every bank transaction, which are made bank account and bank card, is recorded. The data requirements for this system are summarized as follows:

Bank customers are identified by their customer's ID. The bank also stores customers' name, age, phone, email, address. A customer can own multiple accounts and several bank cards.

A banking account has unique account number, create day and balance. The account balance will depends on amount of money of its relating transaction (include AccountTransaction and its CardTransaction). The account is owned by only one customer. Moreover, a bank account may have several account transactions. Banking accounts must be saving account or checking account:

- Saving account has interest rate but is not involved in bank card.
- Checking account may involve several bank cards.

Bank Card has a unique card number, start date, expiry date and each card must associate with a checking account. A bank card also has card transactions. Every bank card is divided into 2 types: Credit card and Debit card.

- The credit card has a security code, interest rate and withdrawn amount of money, which will be the total amount of money withdrawn since the last time it is paid.
- The debit card has an PIN Code and account balance, which equals the balance of the associated account.

Every account transaction must involve a banking account. A transaction has an ID Number, which is unique in the scope of the account. The transaction also includes type, the associated amount of money, the date when carrying out the transaction, destination of the transaction. Account transactions will be offline or online:

- The offline transaction has the location and must be involved with bank staff.
- The online transaction has the name of the app used to carry out the transaction.

The card transaction must involve a bank card. A transaction has an ID Number that is unique in the scope of the card. The transaction also includes type, the associated amount of money, the date when carrying out the transaction, payment machine, destination of the transaction.

Bank staff is identified by their unique staff ID. Staffs also have their name, age, email, telephone number, address, start working date, and working location. A bank staff may be responsible for many offline transaction or not.

4 EER Diagram for our Bank Transaction Database System

4.1 About the tool we use

We draw EER Diagram using draw.io tool. Draw.io is an open source technology stack for building diagramming applications, and the world's most widely used browser-based end-user diagramming application. It's lets you create a wide range of diagrams, from simple tree and flow diagrams, to highly technical network, rack and electrical diagrams by using a wide variety of shapes, icons, connector and templates to help you get started quickly.

The tool also has a desktop application so that it is always available for us to work with. Moreover, the online version gives us online workspace for better team-work. Draw.io provides us everything to draw EER Diagram from EER notation to draw-supporting functions. Finally, the tool is very popular for us to learn nice tips and search for solutions, hence improvement in our productivity.

4.2 Our Diagram

We submited drawio file for better resolution of our diagram: BankTransaction.drawio

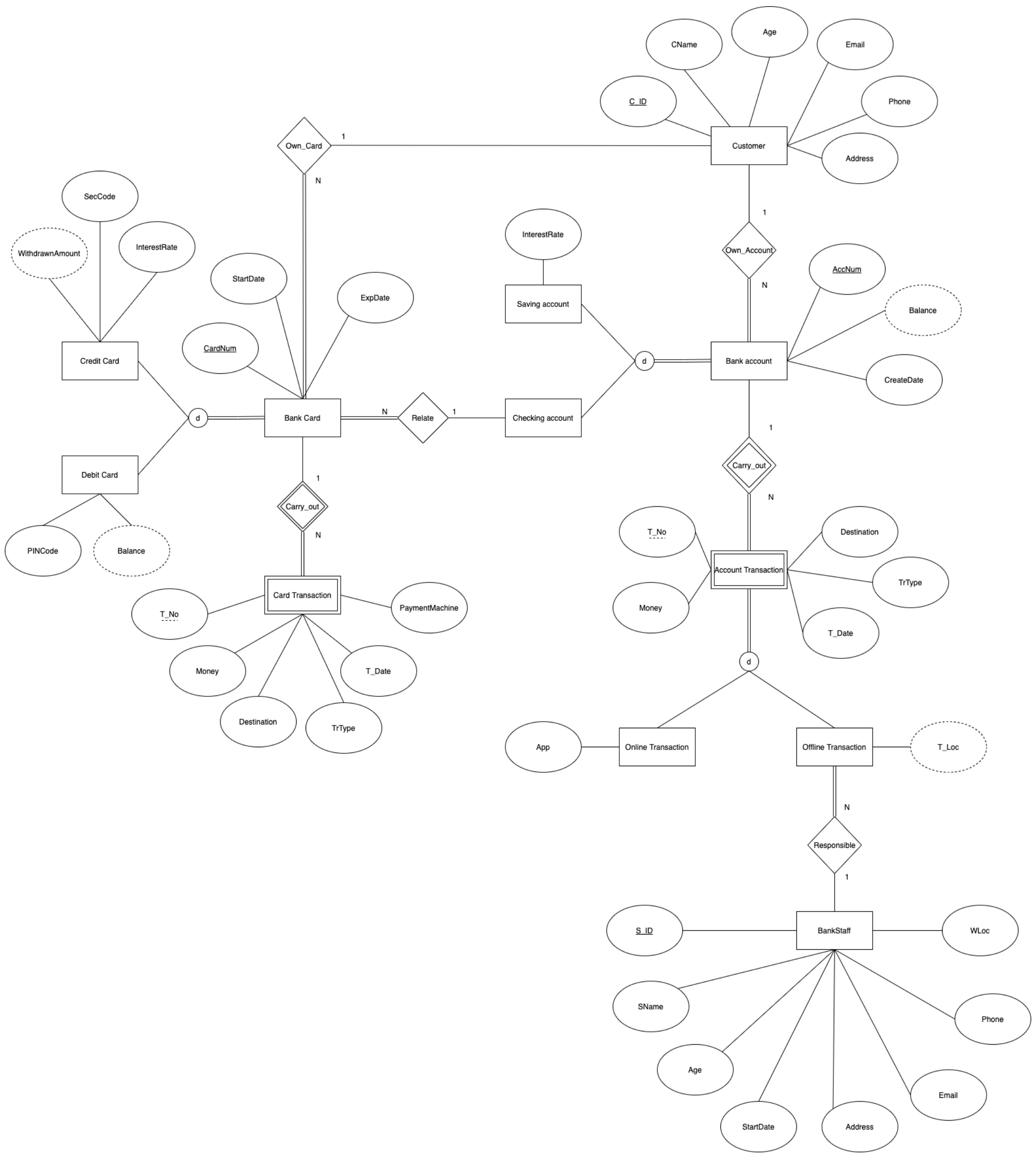


Figure 1: EER Diagram for Bank Transaction database system

4.3 Explanation

In this part, we will explain more about relationships between entities and derived attribute in some entity.

In the Bank Account entity, we have Balance as an derived attribute since

Balance must be update every time a transaction related to the account are carried out

Own_Account relationship between Customer and Bank Account: because each Bank Account has to be own by one customer, we have total participation here. In the opposite side, the customer may have many bank accounts or nothing.

Own_Card relationship between Customer and Bank Account: similar to the Own_Account relationship, each Bank Card has to be own by one customer so we have total participation here. In the opposite side, the customer may have many bank cards or they may not want to have one.

We have 2 identify relationship Carry_out between Bank Account and Account Transaction as well as Bank Card and Card Transaction. We have 2 different entity for transaction here due its different attributes. Here, Card can only work with machine but Account can involve in many transactions except for payment machine which is only for cards.

And about the Responsible relationship between Offline Account Transaction and Bank Staff. The offline transaction happens when customer working with bank staff at the bank office. Therefore, with every offline transaction, there are always a bank staff who responsible for that transaction. Moreover, the online transaction will be carried out by the system so there will be no bank staff involve.

The reason why we do not consider customer and employee as a sub-class of a superclass, for example, People: we want so separate key of customer and staff. We think we cannot have a superclass with no key and we want to have different type and format for key of customer and staff. About querying records of an account: Here we have 2 entity considered as records of an account is CardTransaction and AccountTransaction. We may create many views to customer when they want to query, for example, their account's transaction history. When customer want to see all history transaction, we can look for all account information and all card transaction (a account may have many card) and the No. of may be modified when display in UI. Additionally, this design will be more convenient when customer only want to see transaction of their particular card.

5 Logical design using relational database model for our database system

5.1 About the tool we use

We choose MySQL Workbench as a tool for designing relational Database model and implementing our Database into a DBMS.
MySQL Workbench is a unified visual tool for database architects, developers, and DBAs. MySQL Workbench provides data modeling, SQL development, and comprehensive administration tools for server configuration, user administration, backup, and much more. MySQL Workbench is available on Windows, Linux and Mac OS X.

The reason for our choice is that MySQL Workbench provides us the following benefits:

- The world's most popular open-source database.
- Has a community edition.
- Support cross-platform (Windows, Linux, and MAC OS versions).
- Wide learning resources.
- Support ER/EER Diagram.
- Allow reverse and forward engineering.
- About visual database design.
- Simplifies database design and maintenance.
- Automate time-consuming and error-prone tasks.
- Enable model-driven database design.
- Changes management:
  - MySQL Workbench helps DBAs and developers manage the complex and difficult database change processes involving different versions of database schemas.
  - Schema Synchronization and Comparison utilities allow DBA can compare two databases or a database and a model visually to see the differences.

5.2 Physical Design

For preparation, if the data types have been declared clearly, we can let attributes store data. In the following sentences, we will shows which kind of data types is available for each group of attributes.

- The primary key (AccNum or CardNum) in most of our tables (should use CHAR(12) type. We choose CHAR instead of VARCHAR because AccNum and CardNum always have 12 character hence same storage size but faster query time(CHAR query time is faster than VARCHAR due to its fixed-size feature). The primary key of OffAccTranction, OnlAccTranction and CradTransaction using INT type since we expect it to be count up for every new tuples.

- For a name of a customer or a staff as well as an email address, we use VARCHAR datatype with a the length of 45 characters since Name and Email length is flexible and 45 characters is sufficient.

- Because of the sequences of 10 digits for the telephone number, the good option for this attribute is CHAR(10).

- With regard to address or work location for customers and staff or even destination for a transaction, so we need the length VARCHAR of 100 to store them.

- In term of balance, withdraw , money and age can use INT data type for counting. And date transaction, this attribute will use DATETIME type, which will also store to time when transaction are carried out. While the startDate of Staff, Account and Card are DATE because store time is unnecessary.

5.3 Our relational DB model

We draw the relational database base on EER diagram in the previous section For better resolution you can download the file in our submisstion with the filename: Relational_BankTransaction.mwb.

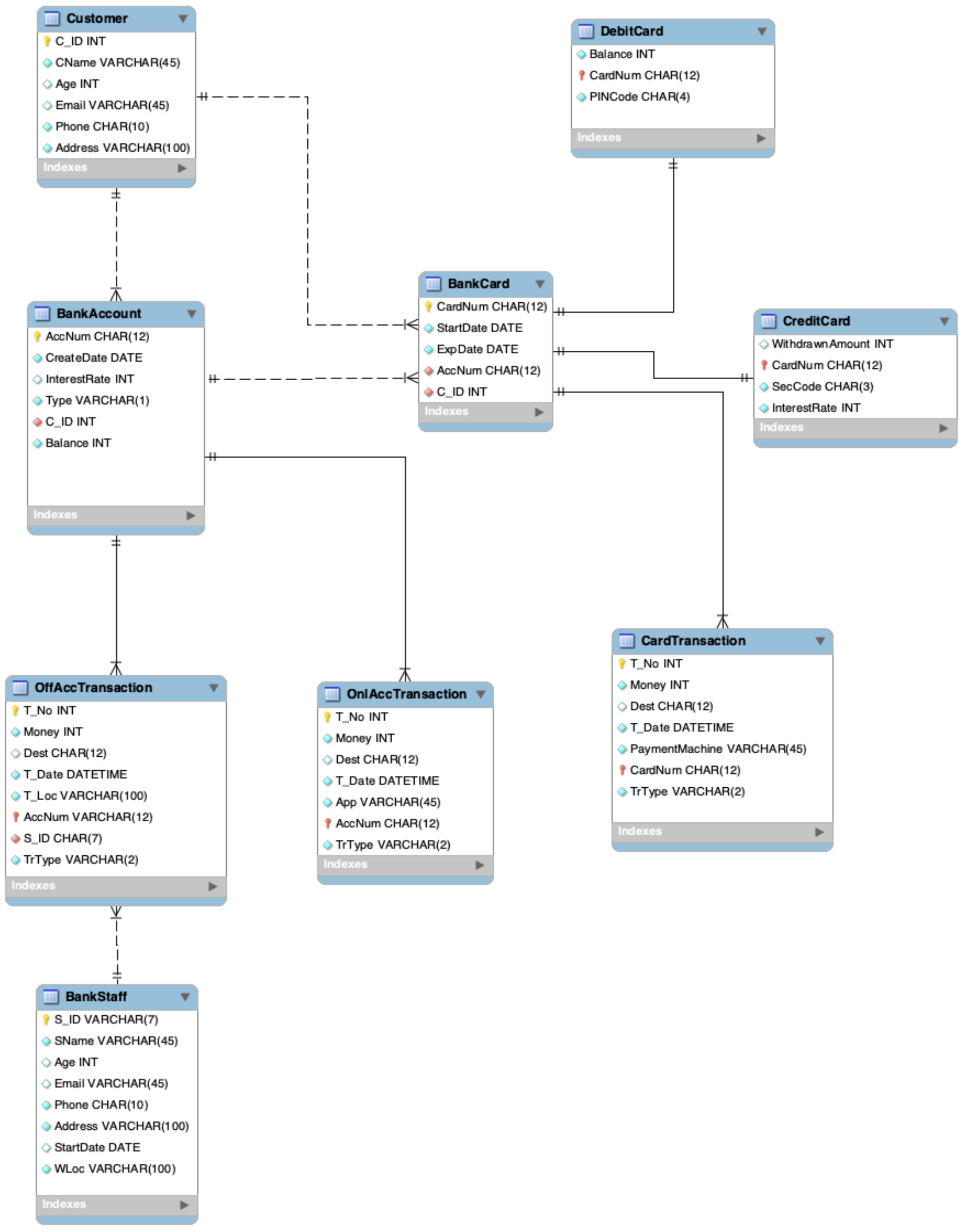


Figure 2: Relational database design for Bank Transaction database system

6 Implement the DBs into a particular DBMS

6.1 Our Code

Our code file BankTransaction.sql is submitted in the drive folder.

```
create database if not exists bank_transaction;
use bank_transaction;

create table Customer
    (C_ID    int             not null,
    CName    varchar(45)     not null,
    Age      int,
    Email    varchar(45),
    Phone    char(10)        not null,
    Address varchar(100)     not null,
        primary key (C_ID));
```

```
create table BankStaff
    (S_ID       varchar(7)       not null,
    SName       varchar(45)      not null,
    Age         int,
    Email       varchar(45),
    Phone       char(10)         not null,
    Address     varchar(100)     not null,
    StartDate   date,
    WLoc        varchar(100)     not null,
        primary key (S_ID));
```

```
create table BankAccount
    (AccNum         char(12)        not null,
    CreateDate      date            not null,
    InterestRate    int,
    Balance         int             not null,
    AccType         varchar(1)      not null,
    C_ID            int             not null,
        primary key (AccNum),
    foreign key (C_ID) references Customer(C_ID)
                on delete cascade on update cascade);
```

```
create table BankCard
    (CardNum     char(12)     not null,
    StartDate    date         not null,
    ExpDate      date         not null,
    AccNum       char(12)     not null,
    C_ID         int          not null,
    primary key (CardNum),
    foreign key (C_ID) references Customer(C_ID)
       on delete cascade on update cascade,
    foreign key (AccNum) references BankAccount(AccNum)
       on delete cascade on update cascade);
```

```
create table CreditCard
    (CardNum                 char(12)        not null,
     WithdrawnAmount         int             default 0,
     SecCode                 char(3)         not null,
     InterestRate            int             not null,
     primary key (CardNum),
     foreign key (CardNum) references BankCard(CardNum)
        on delete cascade on update cascade);
```

```
create table DebitCard
    (CardNum     char(12)        not null,
    Balance      int             not null,
    PINCode      char(4)         not null,
    primary key (CardNum),
    foreign key (CardNum) references BankCard(CardNum)
       on delete cascade on update cascade);
```

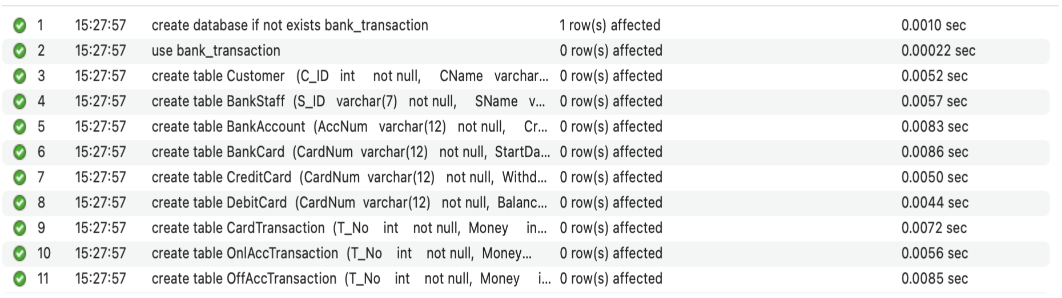
```
create table CardTransaction
    (T_No                int             not null,
     Money               int             not null,
     Dest                char(12),
     T_Date              datetime        not null,
     PaymentMachine      varchar(45)     not null,
     CardNum             char(12)        not null,
     TrType              varchar(2)      not null,
     primary key (CardNum,T_No),
     foreign key (CardNum) references BankCard(CardNum)
        on delete cascade on update cascade);
```

```
create table OnlAccTransaction
    (T_No       int             not null,
    Money       int             not null,
    Dest        char(12),
    T_Date      datetime        not null,
    App         varchar(45)     not null,
    AccNum      char(12)        not null,
    TrType      varchar(2)      not null,
    primary key (AccNum,T_No),
    foreign key (AccNum) references BankAccount(AccNum)
       on delete cascade on update cascade);
```

```
create table OffAccTransaction
    (T_No           int             not null,
     Money          int             not null,
     Dest           var(12),
     T_Date         datetime        not null,
     T_Loc          varchar(45)     not null,
     AccNum         char(12)        not null,
     S_ID           varchar(7)      not null,
     TrType         varchar(2)      not null,
     primary key (AccNum,T_No),
     foreign key (AccNum) references BankAccount(AccNum)
        on delete  cascade on update cascade,
     foreign key (S_ID) references BankStaff(S_ID)
        on delete cascade on update cascade);
```

6.2 Result

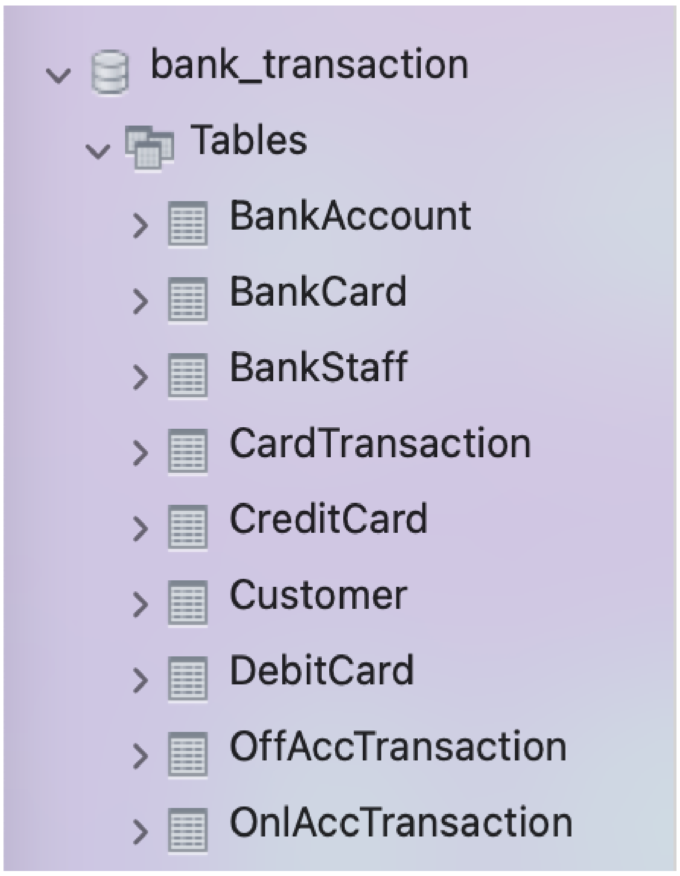
6.2.1 We successfully run our code on MySQL Workbench



```
1   15:27:57  create database if not exists bank_transaction                 1 row(s) affected   0.0010 sec
2   15:27:57  use bank_transaction                                           0 row(s) affected   0.00022 sec
3   15:27:57  create table Customer  (C_ID  int   not null,    CName  varchar...  0 row(s) affected   0.0052 sec
4   15:27:57  create table BankStaff  (S_ID  varchar(7)  not null,    SName  v...  0 row(s) affected   0.0057 sec
5   15:27:57  create table BankAccount  (AccNum  varchar(12)  not null,    Cr...  0 row(s) affected   0.0083 sec
6   15:27:57  create table BankCard  (CardNum  varchar(12)  not null,  StartDa...  0 row(s) affected   0.0086 sec
7   15:27:57  create table CreditCard  (CardNum  varchar(12)  not null,  Withd...  0 row(s) affected   0.0050 sec
8   15:27:57  create table DebitCard  (CardNum  varchar(12)  not null,  Balanc...  0 row(s) affected   0.0044 sec
9   15:27:57  create table CardTransaction  (T_No   int   not null,  Money    in...  0 row(s) affected   0.0072 sec
10  15:27:57  create table OnlAccTransaction  (T_No   int   not null,  Money...  0 row(s) affected   0.0056 sec
11  15:27:57  create table OffAccTransaction  (T_No   int   not null,  Money    i...  0 row(s) affected   0.0085 sec
```

Figure 3: Execute successfully from MySQL Workbench.

6.2.2 The database created from our code



bank_transaction
- Tables
  - BankAccount
  - BankCard
  - BankStaff
  - CardTransaction
  - CreditCard
  - Customer
  - DebitCard
  - OffAccTransaction
  - OnlAccTransaction

Figure 4: Our database when creating successfully from the code above.

7 Conclusion for assignment 1

In our report, we have written requirement description for bank transaction database. The requirement description is referenced from various sources from different banks but we take most information from OCB, which is the bank incorporating with our university.

In this assignment, we use drawio tool to draw EER diagram for our database design. The drawio tool provides us efficient features to draw a complete EER diagram. About our relational database design and implementation of our database into DBMS, we choose MySQL Workbench tool to work with. MySQL Workbench provides us many useful features for design relational database. Moreover, when we draw a complete relational database we can use forward engineering to create MySQL script, which will be very useful for us in the future. However, we manually write the SQL script without using the forward engineering feature to create our SQL script. Writing sql script manually help us improve our skill on mySQL.

In conclusion, after finished the first assignment, we are more familiar with mySQL workbench. We also gain more knowledge about bank transaction and know how to implement a transaction database system with basic information. In next assignment, we may optimize the database system using normalization. We may discuss more about security for the system and develop an application on top of the designed DB system.

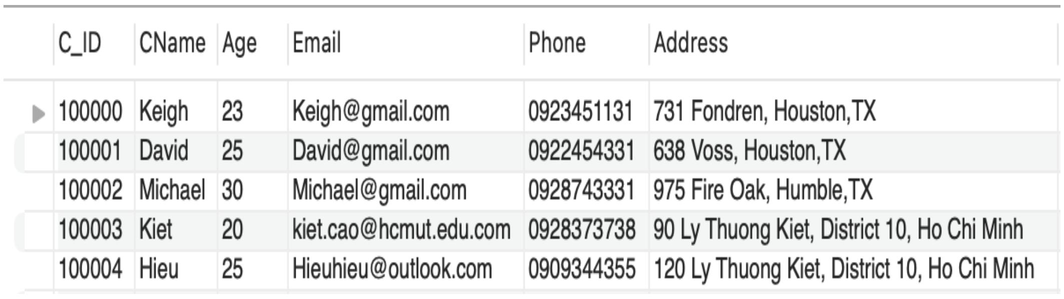
8 Complete implementation of the DB to DBMS

In this section, we will insert initial data to the database and show you the result. The data inserted will be used for query and demonstrating with triggers and procedures.

8.1 Customer Relation:

```
insert into Customer values (100000, 'Keigh', 23,
'Keigh@gmail.com', '0923451131',
'731 Fondren, Houston,TX');
insert into Customer values (100001, 'David', 25,
'David@gmail.com', '0922454331',
'638 Voss, Houston,TX');
insert into Customer values (100002, 'Michael', 30,
'Michael@gmail.com', '0928743331',
'975 Fire Oak, Humble,TX');
insert into Customer values (100003, 'Kiet', 20,
'kiet.cao@hcmut.edu.com', '0928373738',
'90 Ly Thuong Kiet, District 10, Ho Chi Minh');
insert into Customer values (100004, 'Hieu', 25,
'Hieuhieu@outlook.com', '0909344355',
'120 Ly Thuong Kiet, District 10, Ho Chi Minh');
select * from Customer;
```

Result:



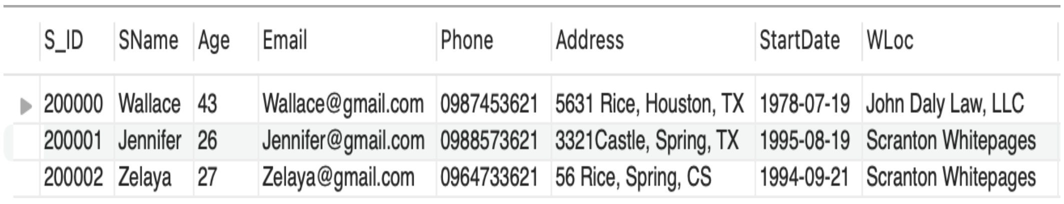
C_ID	CName	Age	Email	Phone	Address
100000	Keigh	23	Keigh@gmail.com	0923451131	731 Fondren, Houston,TX
100001	David	25	David@gmail.com	0922454331	638 Voss, Houston,TX
100002	Michael	30	Michael@gmail.com	0928743331	975 Fire Oak, Humble,TX
100003	Kiet	20	kiet.cao@hcmut.edu.com	0928373738	90 Ly Thuong Kiet, District 10, Ho Chi Minh
100004	Hieu	25	Hieuhieu@outlook.com	0909344355	120 Ly Thuong Kiet, District 10, Ho Chi Minh

Figure 5: Customer Data.

8.2 BankStaff Relation:

```
insert into BankStaff values ('200000', 'Wallace', 43,
'Wallace@gmail.com', '0987453621', '5631 Rice, Houston, TX',
'1978-07-19', 'John Daly Law, LLC');
insert into BankStaff values ('200001', 'Jennifer', 26,
'Jennifer@gmail.com', '0988573621', '3321Castle, Spring, TX',
'1995-08-19', 'Scranton Whitepages');
insert into BankStaff values ('200002', 'Zelaya', 27,
'Zelaya@gmail.com', '0964733621', '56 Rice, Spring, CS',
'1994-09-21', 'Scranton Whitepages');
select * from BankStaff;
```

Result:



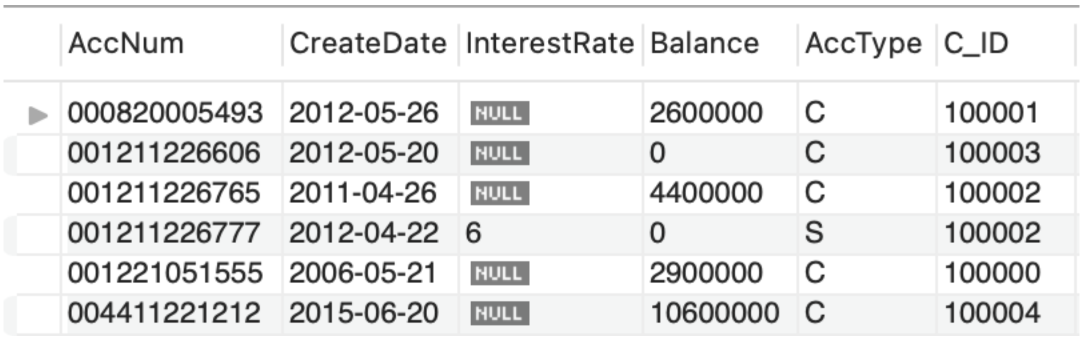
S_ID	SName	Age	Email	Phone	Address	StartDate	WLoc
200000	Wallace	43	Wallace@gmail.com	0987453621	5631 Rice, Houston, TX	1978-07-19	John Daly Law, LLC
200001	Jennifer	26	Jennifer@gmail.com	0988573621	3321Castle, Spring, TX	1995-08-19	Scranton Whitepages
200002	Zelaya	27	Zelaya@gmail.com	0964733621	56 Rice, Spring, CS	1994-09-21	Scranton Whitepages

Figure 6: BankStaff Data.

8.3 BankAccount Relation:

```
insert into BankAccount values ('001221051555', '2006-05-21', null,
2900000, 'C', 100000),
('000820005493', '2012-05-26', null, 2600000, 'C', 100001),
('001211226765', '2011-04-26', null, 4400000, 'C', 100002),
('001211226777', '2012-04-22', 6, 0, 'S', 100002),
('001211226606', '2012-05-20', null, 0, 'C', 100003),
('004411221212', '2015-06-20', null, 10600000, 'C', 100004);
select * from BankAccount;
```

Result:



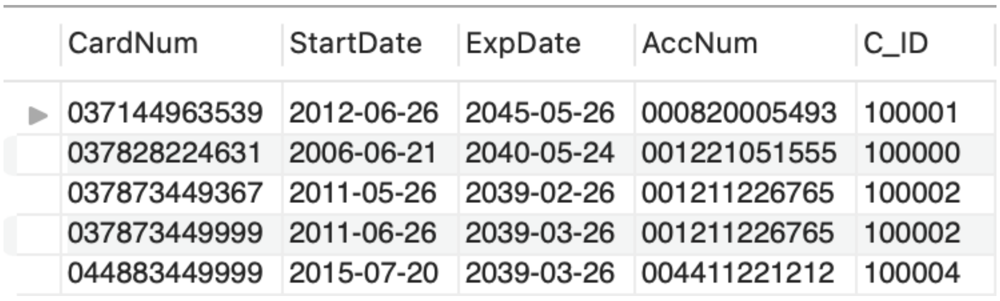
AccNum	CreateDate	InterestRate	Balance	AccType	C_ID
000820005493	2012-05-26	NULL	2600000	C	100001
001211226606	2012-05-20	NULL	0	C	100003
001211226765	2011-04-26	NULL	4400000	C	100002
001211226777	2012-04-22	6	0	S	100002
001221051555	2006-05-21	NULL	2900000	C	100000
004411221212	2015-06-20	NULL	10600000	C	100004

Figure 7: BankAccount Data.

8.4 BankCard Relation:

```
insert into BankCard values('037828224631', '2006-06-21',
'2040-05-24', '001221051555', 100000),
('037144963539', '2012-06-26', '2045-05-26','000820005493', 100001),
('037873449367', '2011-05-26', '2039-02-26','001211226765', 100002),
('037873449999', '2011-06-26', '2039-03-26','001211226765', 100002),
('044883449999', '2015-07-20', '2039-03-26','004411221212', 100004);
```

Result:



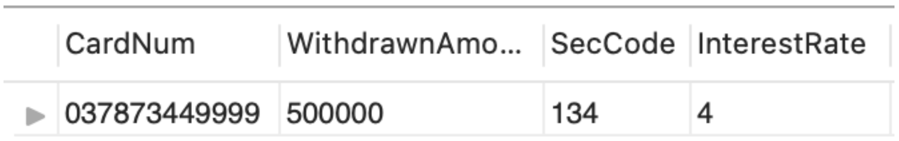
CardNum	StartDate	ExpDate	AccNum	C_ID
037144963539	2012-06-26	2045-05-26	000820005493	100001
037828224631	2006-06-21	2040-05-24	001221051555	100000
037873449367	2011-05-26	2039-02-26	001211226765	100002
037873449999	2011-06-26	2039-03-26	001211226765	100002
044883449999	2015-07-20	2039-03-26	004411221212	100004

Figure 8: BankCard Data.

8.5 CreditCard Relation:

```
insert into CreditCard values ('037873449999', 500000, '134', 4);
select * from CreditCard;
```

Result:



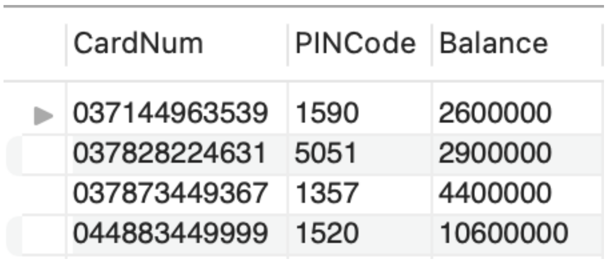
CardNum	WithdrawnAmo...	SecCode	InterestRate
037873449999	500000	134	4

Figure 9: CreditCard Data.

8.6 DebitCard Relation:

```
insert into DebitCard values('037828224631', '5051', 2900000),
('037144963539', '1590', 2600000),
('037873449367', '1357', 4400000),
('044883449999', '1520', 10600000);
select * from DebitCard;
```

Result:



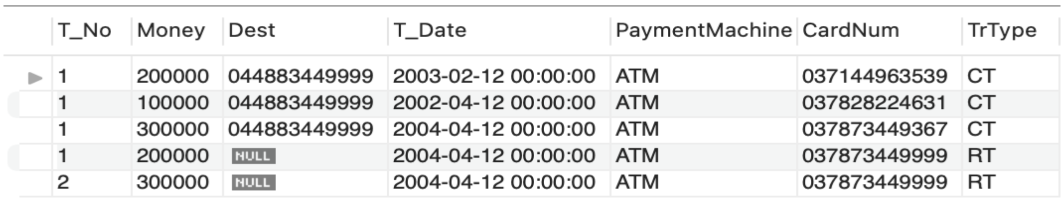
CardNum	PINCode	Balance
037144963539	1590	2600000
037828224631	5051	2900000
037873449367	1357	4400000
044883449999	1520	10600000

Figure 10: DebitCard Data.

8.7 CardTransaction Relation:

```
insert into CardTransaction values(1, 100000, '044883449999',
'2002-04-12', 'ATM', '037828224631', 'CT'),
(1, 200000, '044883449999', '2003-02-12', 'ATM', '037144963539', 'CT'),
(1, 300000, '044883449999', '2004-04-12', 'ATM', '037873449367', 'CT'),
(1, 200000, null, '2004-04-12', 'ATM', '037873449999', 'RT'),
(2, 300000, null, '2004-04-12', 'ATM', '037873449999', 'RT');
select * from CardTransaction;
```

Result:



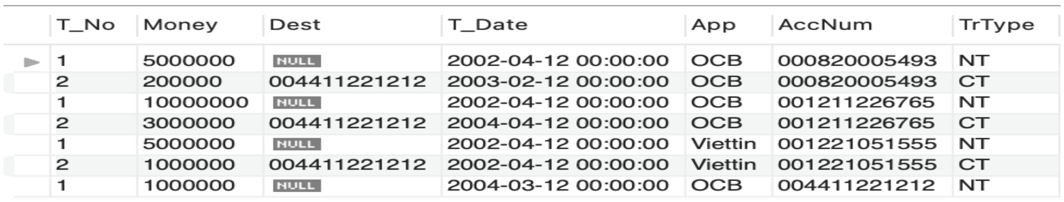
T_No	Money	Dest	T_Date	PaymentMachine	CardNum	TrType
1	200000	044883449999	2003-02-12 00:00:00	ATM	037144963539	CT
1	100000	044883449999	2002-04-12 00:00:00	ATM	037828224631	CT
1	300000	044883449999	2004-04-12 00:00:00	ATM	037873449367	CT
1	200000	NULL	2004-04-12 00:00:00	ATM	037873449999	RT
2	300000	NULL	2004-04-12 00:00:00	ATM	037873449999	RT

Figure 11: CardTransaction Data.

8.8 OnlAccTransaction Relation:

```
insert into OnlAccTransaction values (1, 5000000, null,
'2002-04-12', 'Viettin', '001221051555', 'NT'),
(2, 1000000, '004411221212', '2002-04-12', 'Viettin',
'001221051555', 'CT'),
(1, 5000000, null, '2002-04-12', 'OCB', '000820005493', 'NT'),
(2, 200000, '004411221212', '2003-02-12', 'OCB',
'000820005493', 'CT'),
(1, 10000000, null, '2002-04-12', 'OCB', '001211226765', 'NT'),
(2, 3000000, '004411221212', '2004-04-12', 'OCB',
'001211226765', 'CT'),
(1, 1000000, null, '2004-03-12', 'OCB', '004411221212', 'NT');
select * from OnlAccTransaction;
```

Result:



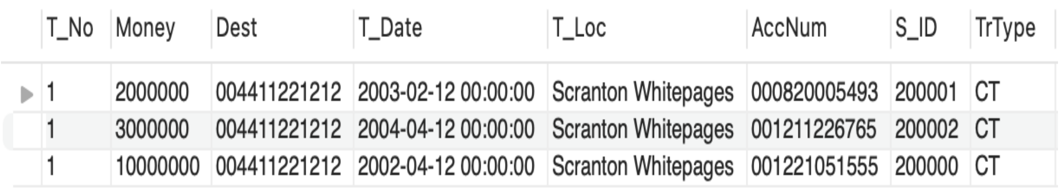
T_No	Money	Dest	T_Date	App	AccNum	TrType
1	5000000	NULL	2002-04-12 00:00:00	OCB	000820005493	NT
2	200000	004411221212	2003-02-12 00:00:00	OCB	000820005493	CT
1	10000000	NULL	2002-04-12 00:00:00	OCB	001211226765	NT
2	3000000	004411221212	2004-04-12 00:00:00	OCB	001211226765	CT
1	5000000	NULL	2002-04-12 00:00:00	Viettin	001221051555	NT
2	1000000	004411221212	2002-04-12 00:00:00	Viettin	001221051555	CT
1	1000000	NULL	2004-03-12 00:00:00	OCB	004411221212	NT

Figure 12: OnlAccTransaction Data.

8.9 OffAccTransaction Relation:

```
insert into OffAccTransaction values(1, 10000000, '004411221212',
'2002-04-12', 'Scranton Whitepages', '001221051555', '200000', 'CT'),
(1, 2000000, '004411221212', '2003-02-12', 'Scranton Whitepages',
'000820005493', '200001', 'CT'),
(1, 3000000, '004411221212', '2004-04-12', 'Scranton Whitepages',
'001211226765', '200002', 'CT');
select * from OffAccTransaction;
```

Result:



T_No	Money	Dest	T_Date	T_Loc	AccNum	S_ID	TrType
1	2000000	004411221212	2003-02-12 00:00:00	Scranton Whitepages	000820005493	200001	CT
1	3000000	004411221212	2004-04-12 00:00:00	Scranton Whitepages	001211226765	200002	CT
1	10000000	004411221212	2002-04-12 00:00:00	Scranton Whitepages	001221051555	200000	CT

Figure 13: OffAccTransaction Data.

9 The normal form (NF) and solutions for improving the NF

In this section, we will briefly review theory of dependency and Normal Form first. After that, we will analyze dependency and Normal Form of our database.

9.1 Informal Design Guidelines for Relation Schemas

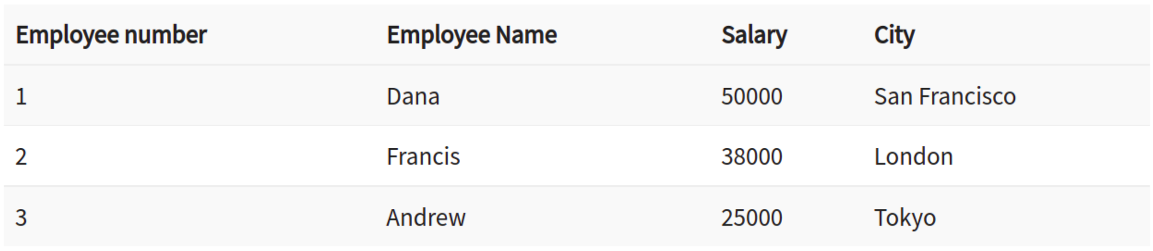
- Making sure that the semantics of the attributes is clear in the schema:
  - Design a relation schema so that it is easy to explain its meaning.
  - Do not combine attributes from multiple entity types and relationship types into a single relation.
- Avoiding update anomalies such as modification, deletion, insertion anomalies which cause redundant work to be done and may lead to accidental loss of information.
- Reducing the NULL values in tuples as well as redundant information in tuples in order to minimize the storage space.
- Disallowing the possibility of generating invalid and spurious tuples, which means that we should design relation schemas so that relations can be joined with equality conditions on attributes.

9.2 The Normal Form

9.2.1 Functional Dependency

Functional Dependency (FD) is a constraint that determines the relation of one attribute to another attribute in a Database Management System (DBMS). Functional Dependency helps to maintain the quality of data in the database, and denoted by an arrow "→".

To understand more about FD, take a look at example below:



Employee number	Employee Name	Salary	City
1	Dana	50000	San Francisco
2	Francis	38000	London
3	Andrew	25000	Tokyo

Figure 14: Example about functional dependecy.

In this example, if we know the value of Employee number, we can obtain Employee Name, city, salary, etc. By this, we can say that the city, Employee Name, and salary are functionally depended on Employee number.

9.2.2 Definition of the Normal Form

Having introduced functional dependencies, we can use them to develop a formal methodology for testing and improving relation schemas, let continue with normalization process for our relational schema design.

There are two approaches often used for relational design projects:

- Performing a conceptual schema design using a conceptual model such as ER or EER and map the conceptual design into a set of relations.
- Designing the relations based on external knowledge derived from an existing implementation of files or forms or reports.

These approaches are useful evaluate for the goodness and decompose relations and need to achieve higher normal forms.

9.2.3 First Normal Form(1NF)

- The first normal form states that:

  1. There is only one value at the intersection of each row and column of a relation - no set valued attributes in 1NF. Disallows composite attributes, multivalued attributes, and **nested relations**.
  2. To be part of the formal definition of a relation in the basic (flat) relational model.
  3. The only attribute values permitted by 1NF are single **atomic** (or **indivisible) values**.

9.2.4 Second Normal Form(2NF)

- The second normal form states that:

  1. Satisfying first normal form.
  2. All attributes must be fully functionally dependent on the primary key.
  3. Don't violate partial dependency problem.

- **Definition.** A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the primary key of R.

9.2.5 Third Normal Form(3NF)

- The third normal form states that:

  1. Satisfying second normal form.
  2. No non-prime attribute in a relation is transitively dependent on the primary key.

- **Definition.** A relation schema R is in third normal form (3NF) if, whenever a nontrivial functional dependency X $\rightarrow$ A holds in R, either (a) X is a superkey of R, or (b) A is a prime attribute of R.

9.2.6 Boyce-Codd normal form(BCNF)

- Boyce-Codd normal form states that:

  1. Satisfying third normal form.

2. A relation R is in Boyce-Codd Normal Form (BCNF) if whenever an functional dependence A -> B holds in R, then B is a superkey of R.

- **Definition.** A relation schema R is in BCNF if whenever a nontrivial functional dependency X → A holds in R, then X is a superkey of R.
- Every relation in BCNF is also in 3NF; however, a relation in 3NF is not necessarily in BCNF.

9.2.7 Fourth Normal Form

- The fourth normal form must obey the following point:
  1. It should be in the Boyce-Codd Normal Form (BCNF).
  2. In addition to a database meeting the requirements of BCNF, it must not contain more than one multivalued dependency.
- If a table violates the normalization standard of Fourth Normal Form, it creates unnecessary redundancies and can contribute to inconsistent data. To solve this problem, it is neccessary to break this information into two table.

9.2.8 Fifth Normal Form

- A relation R is in 5NF if and only if it satisfies following conditions
  1. It should be in should be already in 4NF (4NF).
  2. It cannot be further non loss decomposed (join dependency).
- However, such a dependency is very difficult to detect in practice and therefore, normalization into 5NF is considered very rarely in practice.

9.3 Normal form of our relations

9.3.1 Dependencies

- **Customer Relation**

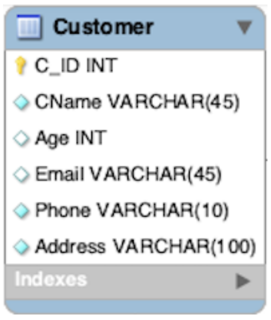


Figure 15: Customer Relation

C_ID ⟶ {CName, Age, Email, Phone, Address}

- **BankStaff Relation**

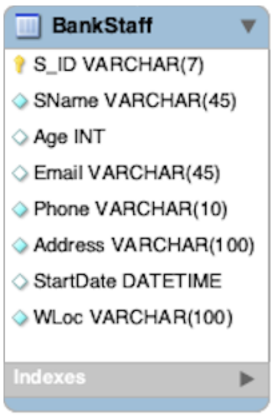


Figure 16: BankStaff Relation

S_ID ⟶ {SName, Age, Email, Phone, Address, StartDate, WLoc}

- **BankAccount Relation**

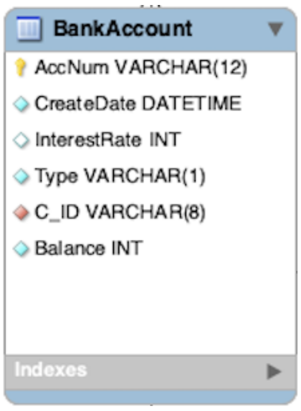


Figure 17: BankAccount Relation

AccNum $\longrightarrow$ {CreateDate, InterestRate, Type, C_ID, Balance}

- **BankCard Relation**

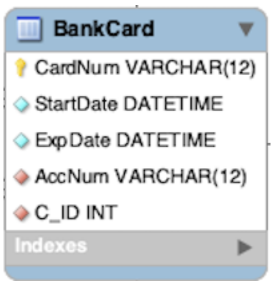


Figure 18: BankCard Relation

CardNum $\longrightarrow$ {StartDate, ExpDate, AccNum, C_ID}

- **CreditCard Relation**

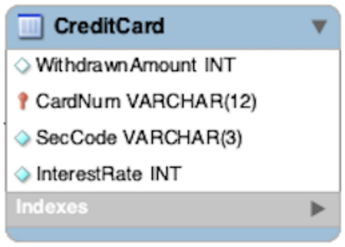


Figure 19: CreditCard Relation

CardNum $\longrightarrow$ {WithdrawnAmount, SecCode, InterestRate}

- **DebitCard Relation**

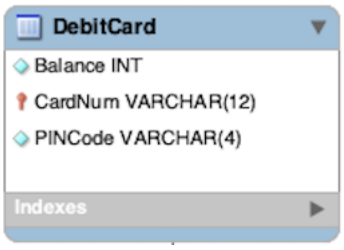


Figure 20: DebitCard Relation

CardNum $\longrightarrow$ {Balance, PINCode}

- **CardTransaction Relation**

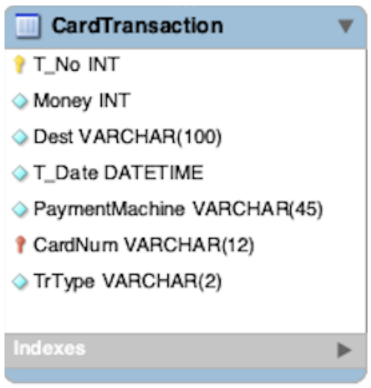


Figure 21: CardTransaction Relation

{T_no, CardNum} $\longrightarrow$ {Money, Dest, T_Date, PaymentMachine, TrType}

- **OnlAccTransaction Relation**

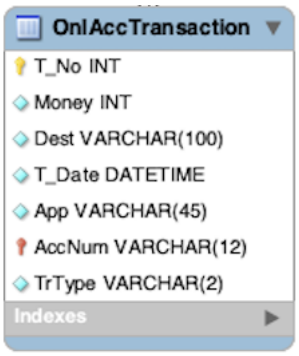


Figure 22: OnlAccTransaction Relation

{T_no, CardNum} ⟶ {Money, Dest, T_Date, App, TrType}

- **OffAccTransaction Relation**

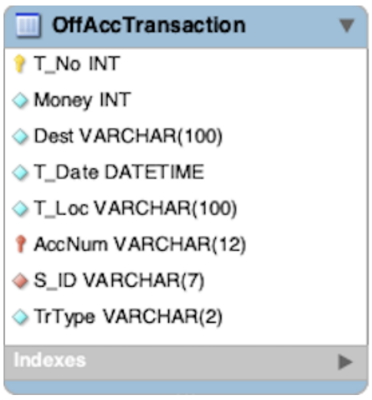


Figure 23: OffAccTransaction Relation

{T_no, CardNum} ⟶ {Money, Dest, T_Date, T_Loc, S_ID, TrType}

9.3.2 First Normal Form

All of our relation have all attribute containing atomic value and there is no nested relation. Therefore, all of our database's relations are in 1NF

9.3.3 Second Normal Form

From all dependencies listed from our relations, non-prime attributes are fully functionally dependent on the primary keys of its relation. Therefore we can conclude that our database's relations are in 2NF

9.3.4 Third Normal Form

Consider dependencies listed from our relations, all attribute are dependent on superkey of its relation. Therefore we can conclude that our database's relations are in 3NF

9.3.5 Boyce-Codd Normal Form

Finally, among all dependencies, superkey is attributes that determine other attribute and the attributes that is dependent on superkey are not prime attribute, which satisfy the condition in definition of BCNF.

9.3.6 Normal Form Conclusion

Our database's relation satisfy Boyce-Codd Normal Form. We did not modify the original relational databse. Therefore, we can conclude that we minimized redundancy and the insertion, deletion, and update anomalies of our database.

10 Database security and Solution

10.1 Definition of database security

Database security is the technique that protects and secures the database against intentional or accidental threats.

Type of security related to this area, for example:

- Regarding the right to access certain ethical and legal information - ex. Limit access of data or prevent inappropriate access to all or part of unauthorized organizations or persons.
- The kind of information should not be made publicly available about policy issues at the governmental, institutional, or corporate level (credit rating or personal medical record).

- The need in some organizations to identify multiple security levels and to categorize the data and users based on these classifications.

Database security refers to the range of tools, controls, and measures designed to establish and preserve database confidentiality, integrity, and availability which are all threads thread to databases.

To protect database against from the confidentiality, integrity, and availability threads. Implementing four kind of control measures is neccessary that is **Access control**, **Inference control**, **Flow control**, **Data encryption**.

1. Access control is handled by creating user accounts and passwords to control the login process by the DBMS. The user must log in to the DBMS by entering the account number and password whenever database access is needed.
2. Statistical databases are used to provide statistical information or summaries of values based on various criteria (ex. age groups, income levels education levels of population statistics). Security for this databases must ensure that information about individuals cannot be accessed. Thus, the corresponding control measures are called inference control measures.
3. Flow control prevents information from flowing in such a way that it reaches unauthorized users.
4. Data encryption is used to protect sensitive data (such as credit or debit card numbers, account number) that is transmitted via some type of communications network. Because the data is encoded using coding algorithm, therefore encryption can be used to provide additional protection for sensitive portions of a database.

10.2 Demonstration and Solution for Bank_Transaction database

- Firstly, we create three accounts, namely **A1, A2, A3**.

```
create user 'A1'@'localhost','A2'@'localhost',
'A3'@'localhost';
```

- And we wants to grant **A1** the privilege to insert and delete tuples in both of relations **BankStaff** and **Customer**, but we does not want **A1** to be able to propagate these privileges to additional accounts, so we use the following codes:

```
grant select, delete on Customer to 'A1'@'localhost';
grant select, delete on BankStaff to 'A1'@'localhost';
```

- Suppose that we wants to allow **A2** to retrieve information from either of the two tables and also to be able to propagate the **SELECT** privilege to other accounts so we use the following codes:

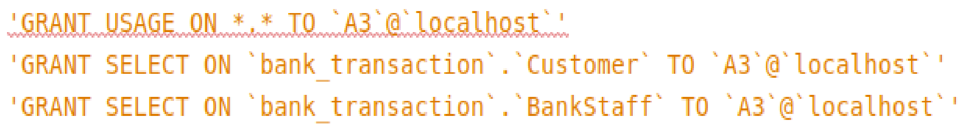
```
grant select on Customer to 'A2'@'localhost'
with grant option;
grant select on BankStaff to 'A2'@'localhost'
with grant option;
```

- User **A2** can grant the **SELECT** privilege on the **Customer** and **BankStaff** relations to **A3** by issuing:

```
use bank_transaction;
grant select on Customer to 'A3'@'localhost';
grant select on BankStaff to 'A3'@'localhost';
```

- To check all privileges of user **A3**, we conduct the following codes:

```
show grants;
```



```
'GRANT USAGE ON *.* TO `A3`@`localhost`'
'GRANT SELECT ON `bank_transaction`.`Customer` TO `A3`@`localhost`'
'GRANT SELECT ON `bank_transaction`.`BankStaff` TO `A3`@`localhost`'
```

Figure 24: Privileges of user **A3**.

- Now to revoke **SELECT** privilege of user **A2** on Customer's table, we can issue:

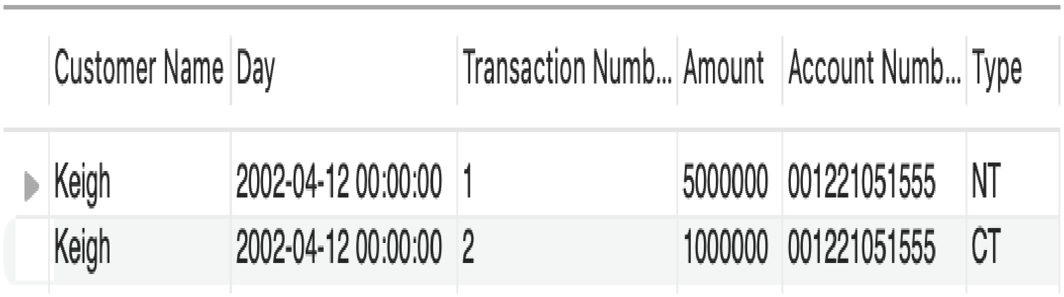
```
revoke select on Customer from 'A2'@'localhost';
```

11 Tuning and Manipulating Bank_Transaction database

11.1 Retrieve data from our database

1. Find information (T_No, T_Date, Money, AccNum, TrType) about the 10 most recent online transaction of Customer whose ID is 100000

```
select
    C.CName as 'Customer Name',
    OT.T_Date as 'Day',
    OT.T_No as 'Transaction Number',
    OT.Money as 'Amount',
    OT.AccNum as 'Account Number',
    OT.TrType as 'Type'
from OnlAccTransaction as OT, Customer as C
where C.C_ID = '100000'
and OT.AccNum = (select AccNum from BankAccount where C_ID = '100000')
order by T_Date desc limit 10;
```

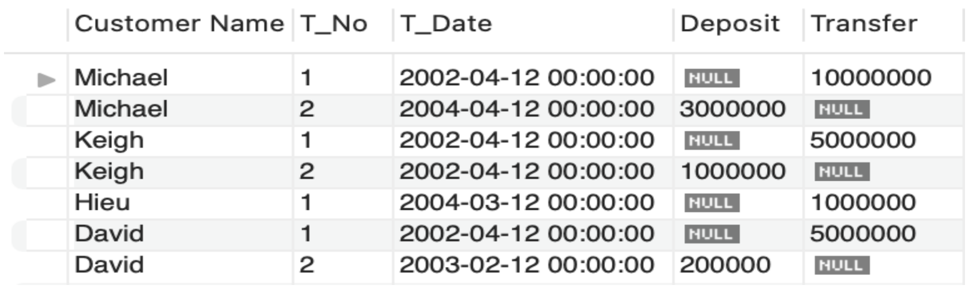


Customer Name	Day	Transaction Numb...	Amount	Account Numb...	Type
Keigh	2002-04-12 00:00:00	1	5000000	001221051555	NT
Keigh	2002-04-12 00:00:00	2	1000000	001221051555	CT

Figure 25: Result of 3 most recent onl-transaction.

2. Find the online transaction history of all customer and classify their amount of money involve into deposit or transfer column base on the transaction type.

```
select
        C.CName as 'Customer Name',
        OT.T_No, OT.T_Date,
case when OT.TrType = 'CT' then Money end as Deposit,
case when OT.TrType = 'NT' then Money end as Transfer
from OnlAccTransaction as OT, Customer as C, BankAccount as AC
where OT.AccNum = AC.AccNum and C.C_ID = AC.C_ID
order by C.CName desc;
```

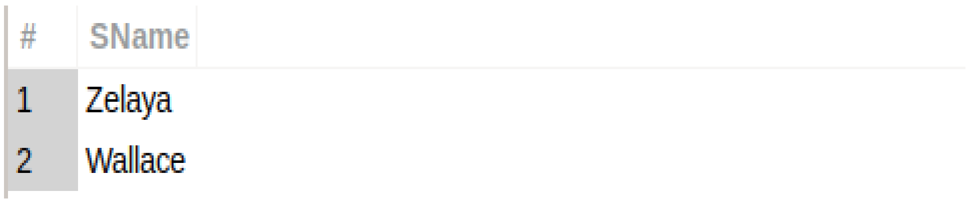


Customer Name	T_No	T_Date	Deposit	Transfer
Michael	1	2002-04-12 00:00:00	NULL	10000000
Michael	2	2004-04-12 00:00:00	3000000	NULL
Keigh	1	2002-04-12 00:00:00	NULL	5000000
Keigh	2	2002-04-12 00:00:00	1000000	NULL
Hieu	1	2004-03-12 00:00:00	NULL	1000000
David	1	2002-04-12 00:00:00	NULL	5000000
David	2	2003-02-12 00:00:00	200000	NULL

Figure 26: Customer's money when charging or withdrawing.

3. Find name of bank staffs who are responsible for accounts having created date <= 2011-08-01.

```
select BS.SName
from BankStaff as BS,
     OffAccTransaction as O,
     BankAccount as BA
where O.S_ID = BS.S_ID
and   O.AccNum = BA.AccNum
and   BA.CreateDate <= '2011-08-01';
```



#	SName
1	Zelaya
2	Wallace

Figure 27: Staff's name for query **3**.

11.2 Trigger & Stored Procedure

In this section, our team creates two trigger to validate data even before they are inserted or updated:

11.2.1 Trigger for Online Transaction:

Our flow is to check if the account have enough money to transfer before transaction (We do not check if the transaction is deposit). After insertion, we will change the balance of the Bank Account and its relating Debit Cards to make sure that the Balance is up-to-date and consistence

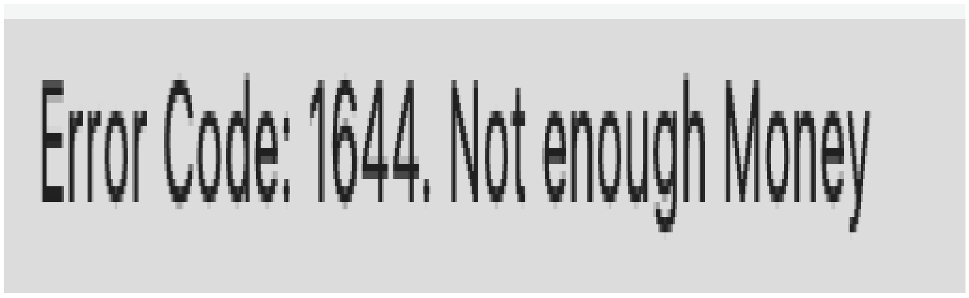
Trigger for Online Transaction to make sure that they have enough money to transfer money

```
drop trigger if exists check_balance_CT;
delimiter //
create trigger check_balance_CT
before insert on OnlAccTransaction
for each row
begin
if new.TrType = 'CT'
and new.Money > (select Balance from BankAccount where AccNum = new.AccNum)
then signal sqlstate '45000'
set message_text = 'Not enough Money';
        end if;
end //
delimiter ;
```

And we test with the following insertion: Based on the trigger the insertion cannot be done since the account does not have enough balance to transfer

```
insert into OnlAccTransaction values (3, 50000000, null, '2002-04-12',
'Viettin', '001221051555', 'CT');
```

Result:



```
Error Code: 1644. Not enough Money
```

Figure 28: Error return from trigger 1.

Then we add trigger after insertion to make things up-to-date and consistent. 'CT' means you transfer your money to another account and 'NT' means you deposit amount of money to your account

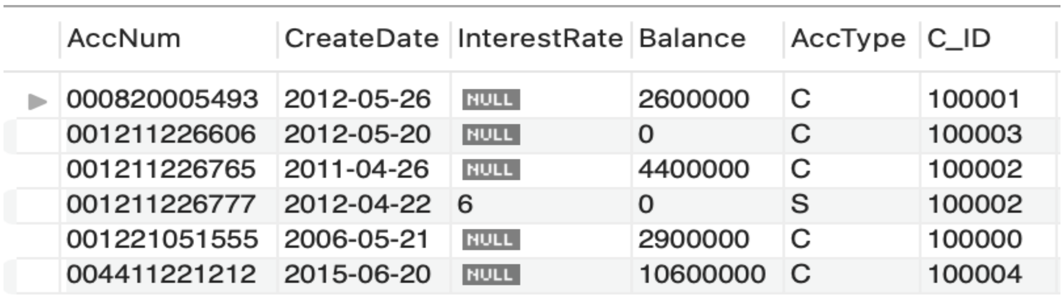
```
delimiter //
create trigger Onl_Transaction_After
after insert on OnlAccTransaction
for each row
begin
if new.TrType = 'CT' then
-- Minus the balance of sender
        update BankAccount as A
        set A.balance = A.balance - new.Money
    where A.AccNum = new.AccNum;
-- Add the balance of receiver
    update BankAccount as A
        set A.balance = A.balance + new.Money
    where A.AccNum = new.Dest;
-- Make Debit Card consistent
    update DebitCard as DC, BankCard as BC
        set DC.balance = DC.balance - new.Money
    where BC.AccNum = new.AccNum and DC.CardNum = BC.CardNum;
    update DebitCard as DC, BankCard as BC
        set DC.balance = DC.balance + new.Money
    where BC.AccNum = new.Dest and DC.CardNum = BC.CardNum;
end if;
if new.TrType = 'NT' then
-- Add the balance of sender
        update BankAccount as A
        set A.balance = A.balance + new.Money
    where A.AccNum = new.AccNum;
-- Make Debit Card consistent
    update DebitCard as DC, BankCard as BC
        set DC.balance = DC.balance + new.Money
    where BC.AccNum = new.AccNum and DC.CardNum = BC.CardNum;
end if;
end //
delimiter ;
```

We test with the following context:

- Account 001221051555 transfer 50000 to account 004411221212
- Account 000820005493 deposits 500000

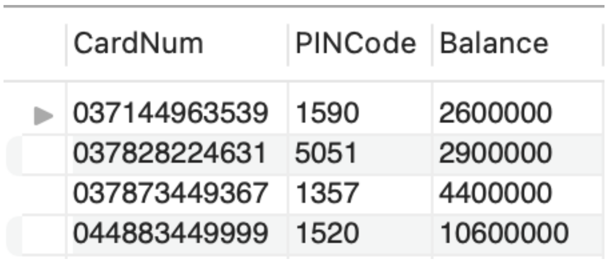
```
insert into OnlAccTransaction values (3, 50000, '004411221212',
'2002-04-12', 'Viettin', '001221051555', 'CT');
insert into OnlAccTransaction values (3, 500000, null, '2002-04-12',
'Viettin', '000820005493', 'NT');
select * from OnlAccTransaction;
select * from BankAccount;
select * from DebitCard
```

Result:
Before:



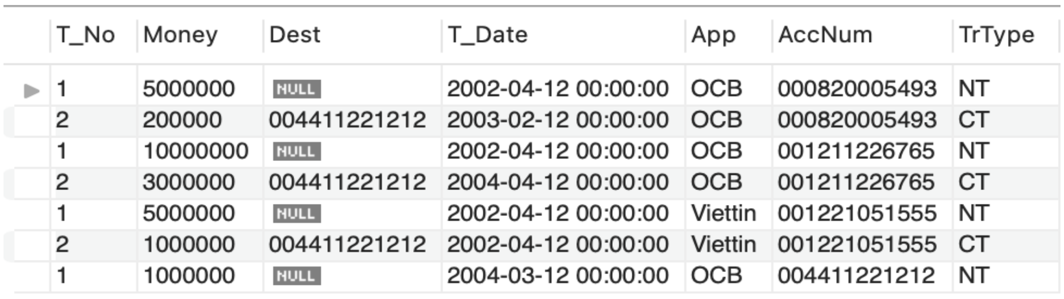
AccNum	CreateDate	InterestRate	Balance	AccType	C_ID
000820005493	2012-05-26	NULL	2600000	C	100001
001211226606	2012-05-20	NULL	0	C	100003
001211226765	2011-04-26	NULL	4400000	C	100002
001211226777	2012-04-22	6	0	S	100002
001221051555	2006-05-21	NULL	2900000	C	100000
004411221212	2015-06-20	NULL	10600000	C	100004

Figure 29: Initial Bank Account Data



CardNum	PINCode	Balance
037144963539	1590	2600000
037828224631	5051	2900000
037873449367	1357	4400000
044883449999	1520	10600000

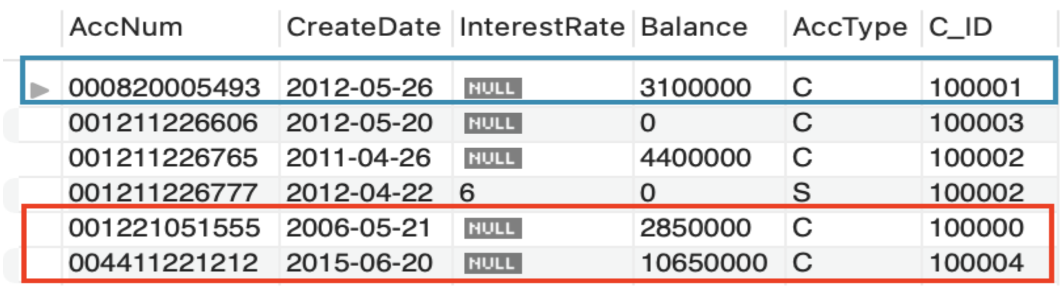
Figure 30: Initial Debit Card Data



T_No	Money	Dest	T_Date	App	AccNum	TrType
1	5000000	NULL	2002-04-12 00:00:00	OCB	000820005493	NT
2	200000	004411221212	2003-02-12 00:00:00	OCB	000820005493	CT
1	10000000	NULL	2002-04-12 00:00:00	OCB	001211226765	NT
2	3000000	004411221212	2004-04-12 00:00:00	OCB	001211226765	CT
1	5000000	NULL	2002-04-12 00:00:00	Viettin	001221051555	NT
2	1000000	004411221212	2002-04-12 00:00:00	Viettin	001221051555	CT
1	1000000	NULL	2004-03-12 00:00:00	OCB	004411221212	NT

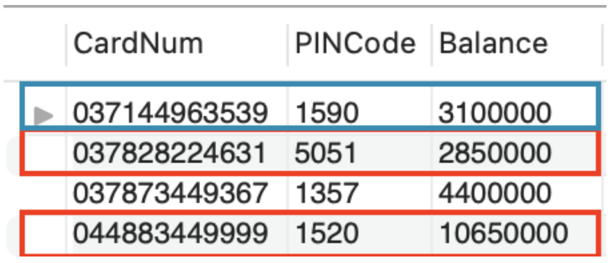
Figure 31: Initial Online Account Transaction Data

After:



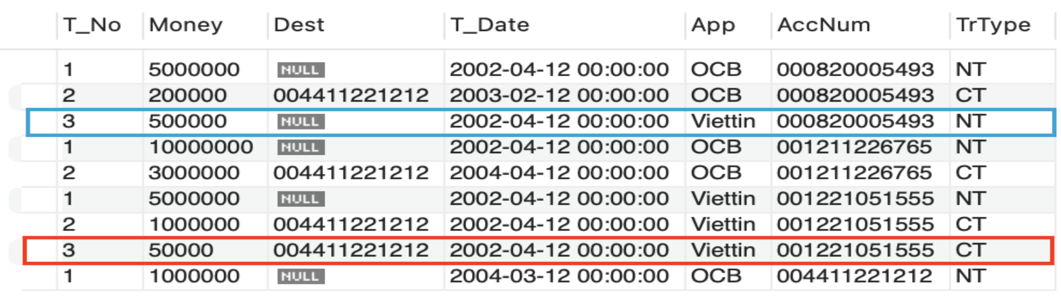
AccNum	CreateDate	InterestRate	Balance	AccType	C_ID
000820005493	2012-05-26	NULL	3100000	C	100001
001211226606	2012-05-20	NULL	0	C	100003
001211226765	2011-04-26	NULL	4400000	C	100002
001211226777	2012-04-22	6	0	S	100002
001221051555	2006-05-21	NULL	2850000	C	100000
004411221212	2015-06-20	NULL	10650000	C	100004

Figure 32: Updated Bank Account Data



CardNum	PINCode	Balance
037144963539	1590	3100000
037828224631	5051	2850000
037873449367	1357	4400000
044883449999	1520	10650000

Figure 33: Updated Debit Card Data



T_No	Money	Dest	T_Date	App	AccNum	TrType
1	5000000	NULL	2002-04-12 00:00:00	OCB	000820005493	NT
2	200000	004411221212	2003-02-12 00:00:00	OCB	000820005493	CT
3	500000	NULL	2002-04-12 00:00:00	Viettin	000820005493	NT
1	10000000	NULL	2002-04-12 00:00:00	OCB	001211226765	NT
2	3000000	004411221212	2004-04-12 00:00:00	OCB	001211226765	CT
1	5000000	NULL	2002-04-12 00:00:00	Viettin	001221051555	NT
2	1000000	004411221212	2002-04-12 00:00:00	Viettin	001221051555	CT
3	50000	004411221212	2002-04-12 00:00:00	Viettin	001221051555	CT
1	1000000	NULL	2004-03-12 00:00:00	OCB	004411221212	NT

Figure 34: Updated Online Transaction Data

Result Explanation: In the after results, we add blue bounding box for the change made by deposit transaction and the red one for the change made by transfer money transaction. As you can see, the amount of money in deposit transaction is 500000 and the balance of both is added 500000 to be 3100000 and both balance value is consistent too. Similar things happen with the transfer money transaction.

11.2.2 Trigger procedure for adding staff:

1. Trigger for Customer to detect whether age is less than 0.

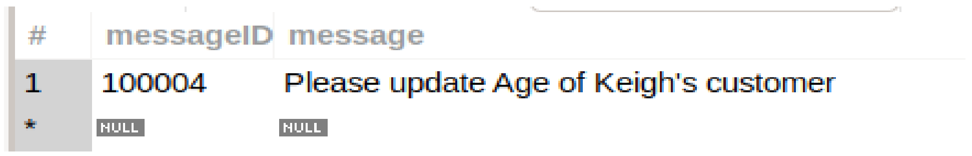
```
create table message
(messageID      int          not null,
 message        varchar(100),
 primary key (messageID));
```

message table will annouce when inserting a customer's age < 0 and need to update. Therefore, trigger should like this:

```
delimiter //
create trigger check_age_email before insert on Customer
for each row
begin
if new.Age < 0 then insert into message (messageID, message)
values (new.C_ID, concat('Please update Age of ', new.CName,
'\'s customer'));
end if;
if new.Email not regexp '^[A-Z0-9][A-Z0-9.]*@[A-Z0-9][A-Z0-9.]*
\.[A-Z]{2,4}'
then insert into message (messageID, message)
values (new.C_ID, concat('Please update Email of ',
new.CName,'\'s customer'));
end if;
end //
delimiter ;
```

Let check whether our trigger is still working

```
insert into Customer values (100004, 'Keigh', -1,
'keigh@gmail.com', '0923451131', '731 Fondren, Houston,TX');
select * from message;
```



#	messageID	message
1	100004	Please update Age of Keigh's customer
*	NULL	NULL

Figure 35: Trigger before inserting

You can drop this trigger if you want.

```
drop trigger if exists check_age_email;
```

Procedure:

Note that in trigger above, it's not prevent new value from inserting into Customer's table. Now we will create another trigger and procedure for BankStaff's table in order to check Age of staffs who must be greater than 18 years old and valid emails for them. The insertion will not be allowed unless it meet the requirements.

```
delimiter //
create procedure validate_staff(
    in age int,
    in email varchar(45)
)
begin
    if age <= 18 then signal sqlstate '45000'
set message_text = 'Age must be greater than 18';
    end if;
    if not (select email regexp '^[A-Z0-9]
    [A-Z0-9.]*@[A-Z0-9][A-Z0-9.]*\.[A-Z]{2,4}') then
```

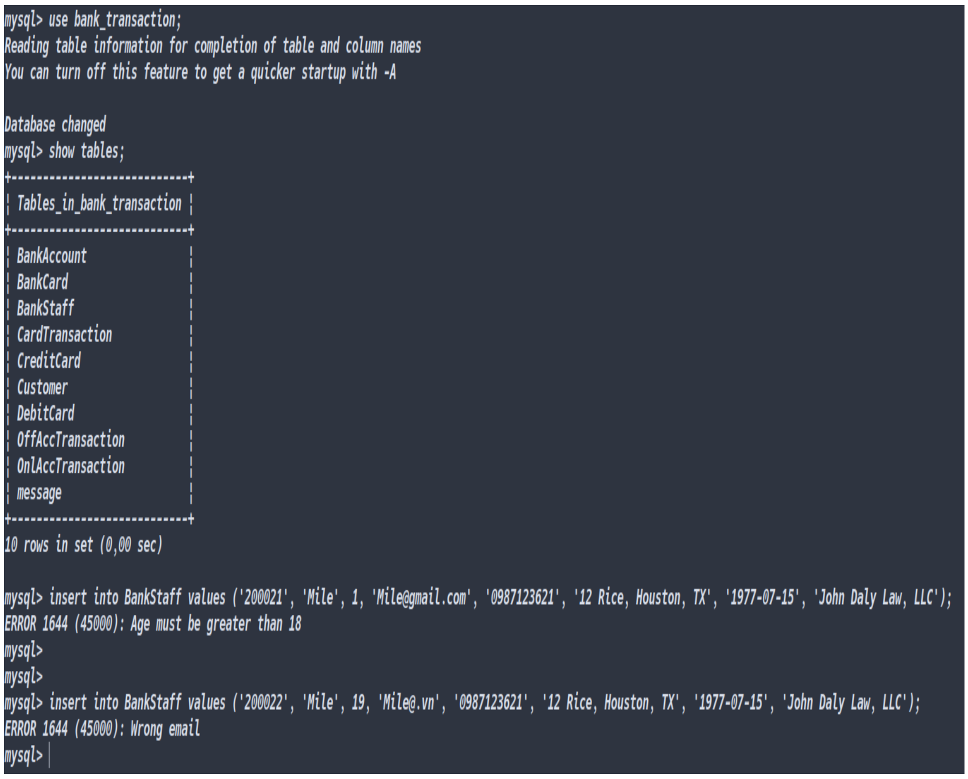
```
signal sqlstate '45000' set message_text = 'Wrong email';
    end if;
end//
delimiter ;

delimiter //
create trigger validate_staff_insert
before insert on BankStaff for each row
begin
    call validate_staff(new.Age, new.Email);
end//
delimiter ;
```

Let check whether our procedure and trigger are still working.

```
insert into BankStaff values ('200021', 'Mile', 1,
'Mile@gmail.com', '0987123621', '12 Rice, Houston, TX',
'1977-07-15', 'John Daly Law, LLC');
insert into BankStaff values ('200022', 'Mile', 19, 'Mile@.vn',
'0987123621', '12 Rice, Houston, TX', '1977-07-15',
'John Daly Law, LLC');
```

Here is the result of that procedure.



```
mysql> use bank_transaction;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+--------------------------+
| Tables_in_bank_transaction |
+--------------------------+
| BankAccount              |
| BankCard                 |
| BankStaff                |
| CardTransaction          |
| CreditCard               |
| Customer                 |
| DebitCard                |
| OffAccTransaction        |
| OnlAccTransaction        |
| message                  |
+--------------------------+
10 rows in set (0,00 sec)

mysql> insert into BankStaff values ('200021', 'Mile', 1, 'Mile@gmail.com', '0987123621', '12 Rice, Houston, TX', '1977-07-15', 'John Daly Law, LLC');
ERROR 1644 (45000): Age must be greater than 18
mysql>
mysql>
mysql> insert into BankStaff values ('200022', 'Mile', 19, 'Mile@.vn', '0987123621', '12 Rice, Houston, TX', '1977-07-15', 'John Daly Law, LLC');
ERROR 1644 (45000): Wrong email
mysql>
```

Figure 36: Error when inserting staff member.

11.3 Indexing

11.3.1 Advantages of using indexes

Indexes are used to find rows with specific column values quickly. Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows. The larger the table, the more this costs. If the table has an index for the columns in question, MySQL can quickly determine the position to seek to in the middle of the data file without having to look at all the data.

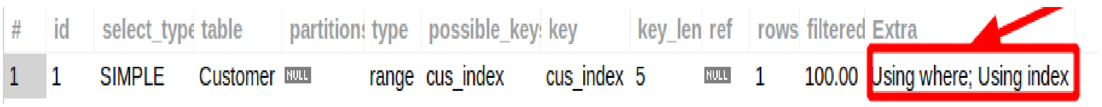
11.3.2 Demonstration for using indexes in Bank_Transaction database

- Firstly, we create index, namely cus_index on Age attribute of Customer's table and use the following code:

```
create index cus_index on Customer(Age);
```

- Then we check how fast it is when using index and **EXPLAIN** clause to find customers who are greater than 30 years old:

```
explain select C_ID from Customer where Age > 30;
```

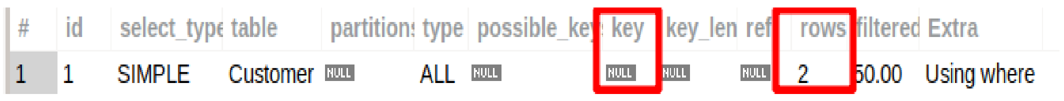


#	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	1	SIMPLE	Customer	NULL	range	cus_index	cus_index	5	NULL	1	100.00	Using where; Using index

Figure 37: The result of using cus_index.

- Now we drop this index and use **EXPLAIN** clause again to compare the fast without using index.

```
drop index cus_index on Customer;
explain select C_ID from Customer where Age > 30;
```



#	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	1	SIMPLE	Customer	NULL	ALL	NULL	NULL	NULL	NULL	2	50.00	Using where

Figure 38: The result of without using cus_index.

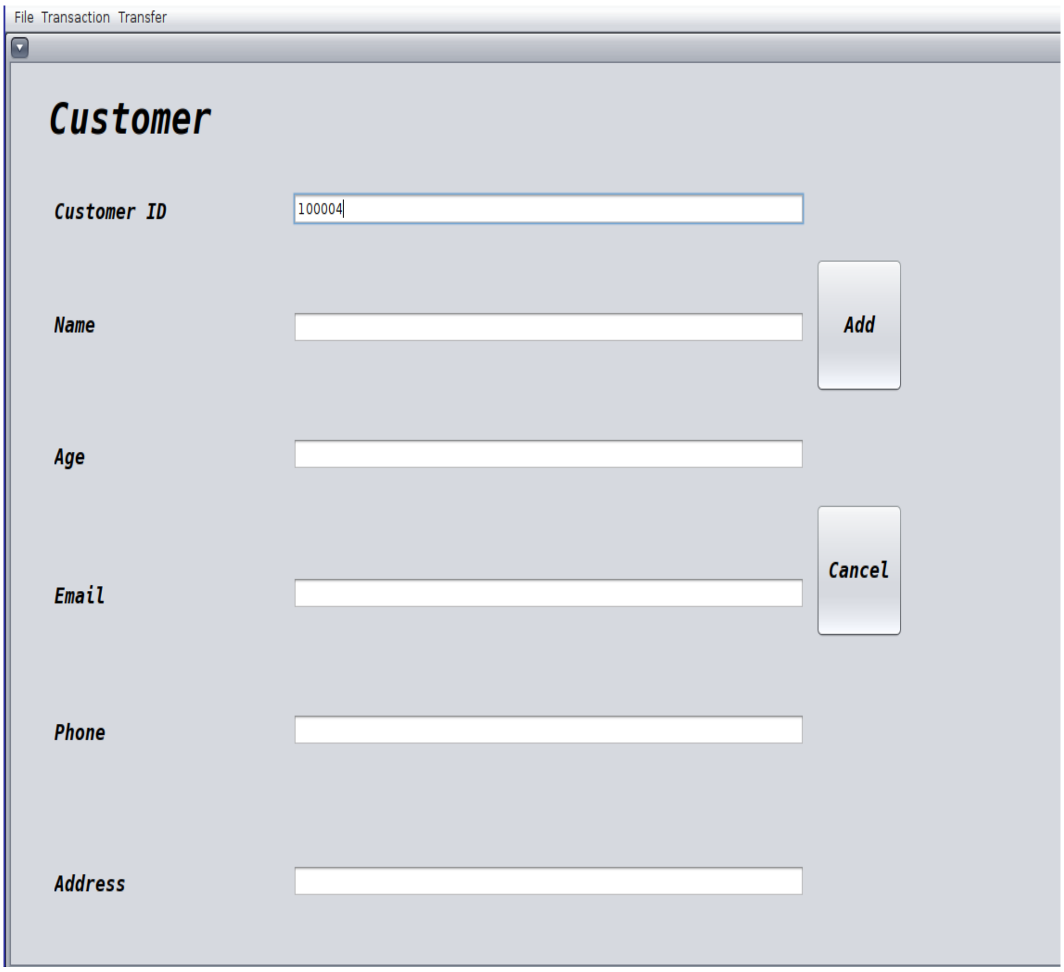
- As you can see from the **figure 23**, there are no index used by DBMS to find the Age attribute and there are 2 filtered rows. In the contrast, in the **figure 22** there is only 1 filtered row. Because we only inserted a little bit data to our database so we don't see the differences between them. If there are lots of data in relations, then using indexes is always better than without using them.

12 An application for our Bank_Transaction database

Firstly, we create a GUI(graphical user interface) and connect it to Mysql in Java (Netbeans) and using **mysql-connector-java** like this.

Figure 39: GUI for application.

Next, let move on to file in menu bar, there are three table to tackle data that is **Customer**, **BankCard**, **Account** residing in **file**. Begin with **Customer**

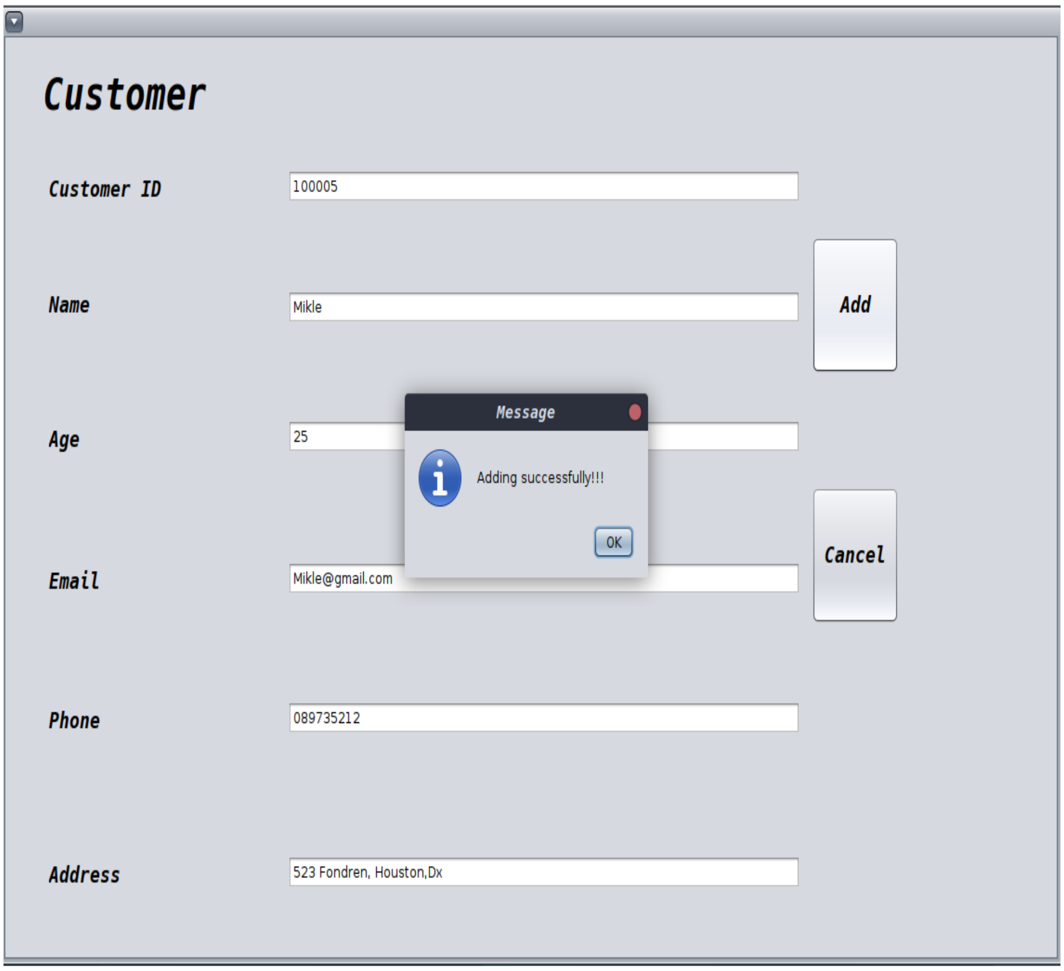


Customer

Customer ID	100004
Name	
Age	
Email	
Phone	
Address	

Figure 40: Customer interface.

Now, let insert one value to our database.



Customer

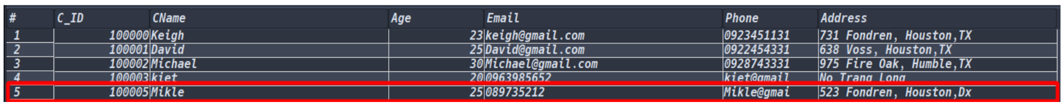
Customer ID	100005
Name	Mikle
Age	25
Email	Mikle@gmail.com
Phone	089735212
Address	523 Fondren, Houston,Dx

Message

Adding successfully!!!

Figure 41: Inserting data to Customer's table.

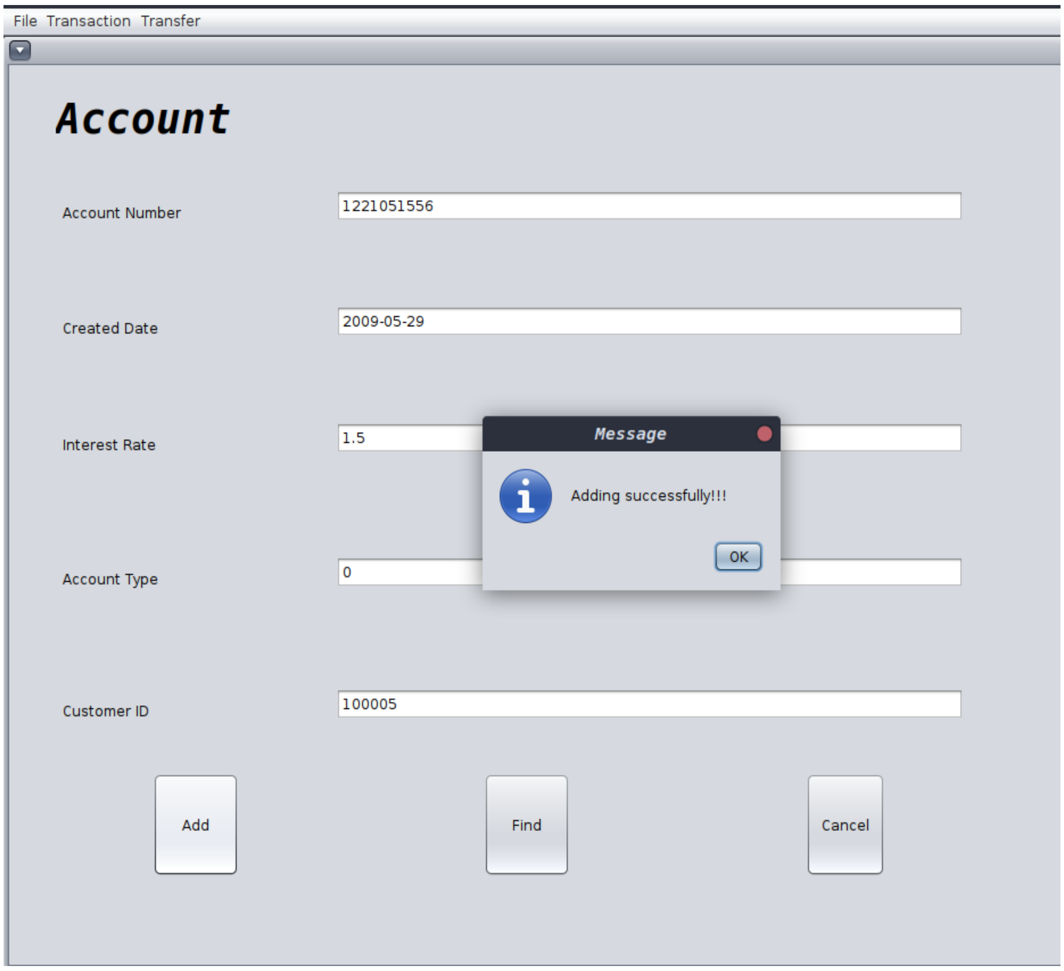
It's said that **Adding successfully!!!**



#	C_ID	CName	Age	Email	Phone	Address
1	100000	Keigh	23	keigh@gmail.com	0923451131	731 Fondren, Houston,TX
2	100001	David	25	David@gmail.com	0922454331	638 Voss, Houston,TX
3	100002	Michael	30	Michael@gmail.com	0928743331	975 Fire Oak, Humble,TX
4	100003	kiet	20	0963985652	kiet@gmail	No Trang Long
5	100005	Mikle	25	089735212	Mikle@gmai	523 Fondren, Houston,Dx

Figure 42: Data in Customer's table.

Continue to **Account** interface(abbreviate for BankAccount). When inserting data into **BankAccount**, result look like this.



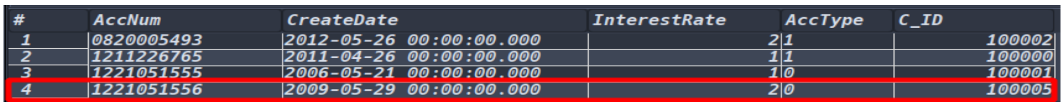
Account

Account Number	1221051556
Created Date	2009-05-29
Interest Rate	1.5
Account Type	0
Customer ID	100005

Message

Adding successfully!!!

Figure 43: Inserting data to BankAccount's table.



#	AccNum	CreateDate	InterestRate	AccType	C_ID
1	0820005493	2012-05-26 00:00:00.000	2	1	100002
2	1211226765	2011-04-26 00:00:00.000	1	1	100000
3	1221051555	2006-05-21 00:00:00.000	1	0	100001
4	1221051556	2009-05-29 00:00:00.000	2	0	100005

Figure 44: Data in BankAccount's table.

You can also find data with **Find** button just typing Customer ID and the result will show similar with figure 28.

And for BankCard interface which shows the same result.

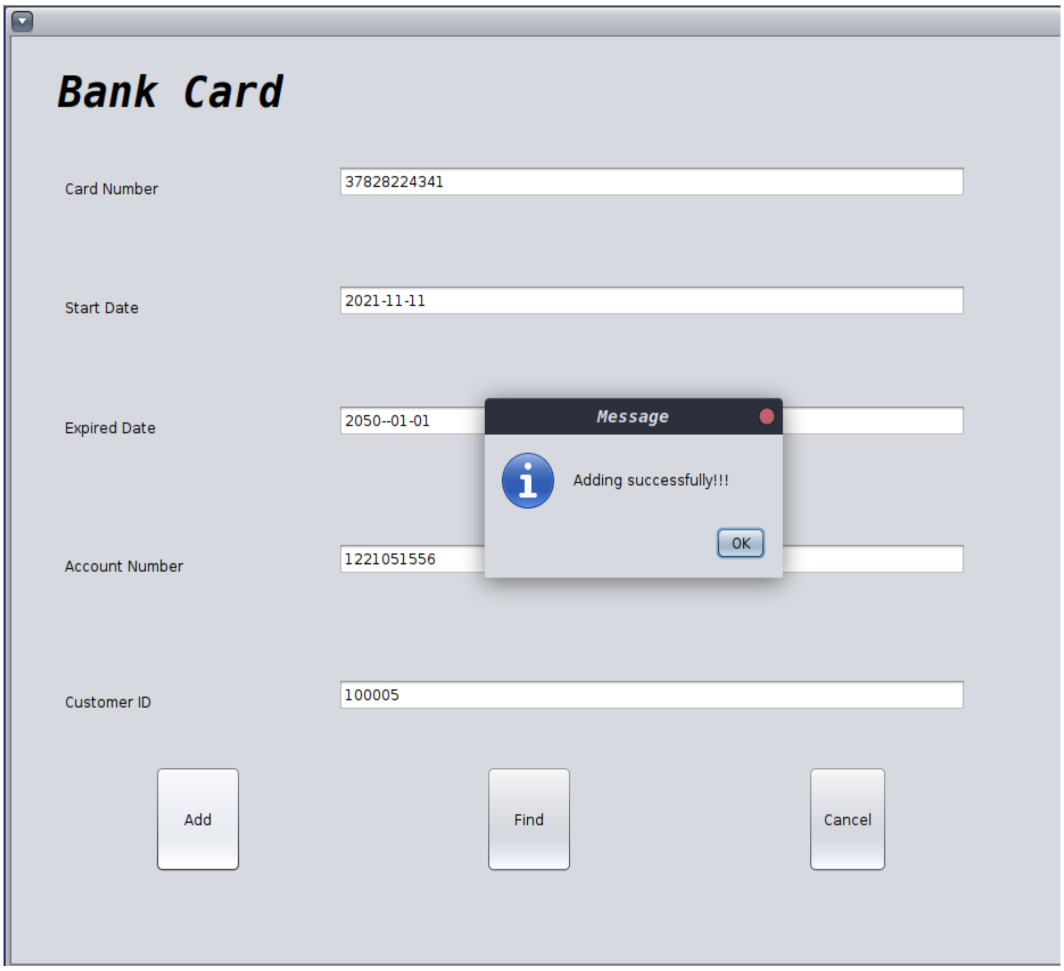
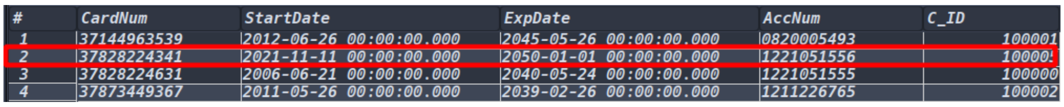


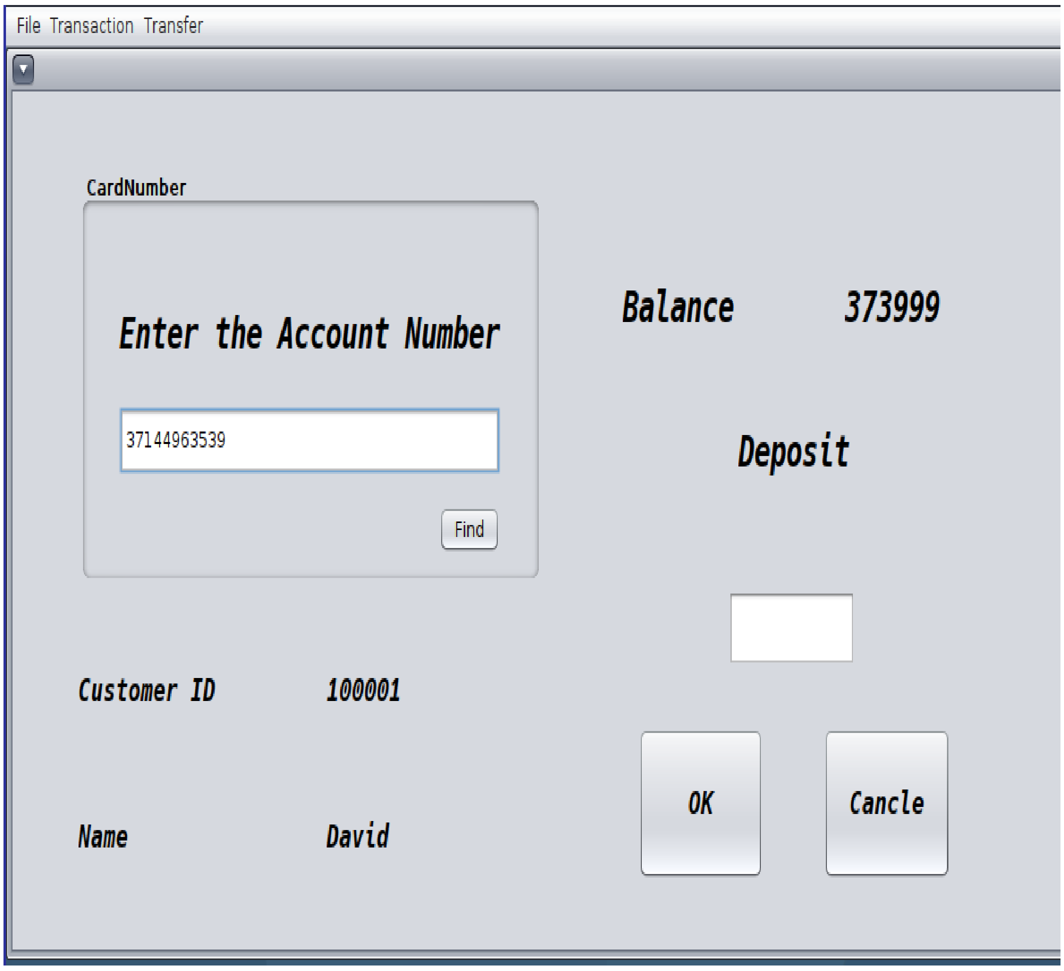
Figure 45: Inserting data to BankCard's table.



#	CardNum	StartDate	ExpDate	AccNum	C_ID
1	37144963539	2012-06-26 00:00:00.000	2045-05-26 00:00:00.000	0820005493	100001
2	37828224341	2021-11-11 00:00:00.000	2050-01-01 00:00:00.000	1221051556	100005
3	37828224631	2006-06-21 00:00:00.000	2040-05-24 00:00:00.000	1221051555	100000
4	37873449367	2011-05-26 00:00:00.000	2039-02-26 00:00:00.000	1211226765	100002

Figure 46: Data in BankCard's table.

Now, go to transact money in **Transaction** menu.



CardNumber

Enter the Account Number

37144963539

Balance 373999

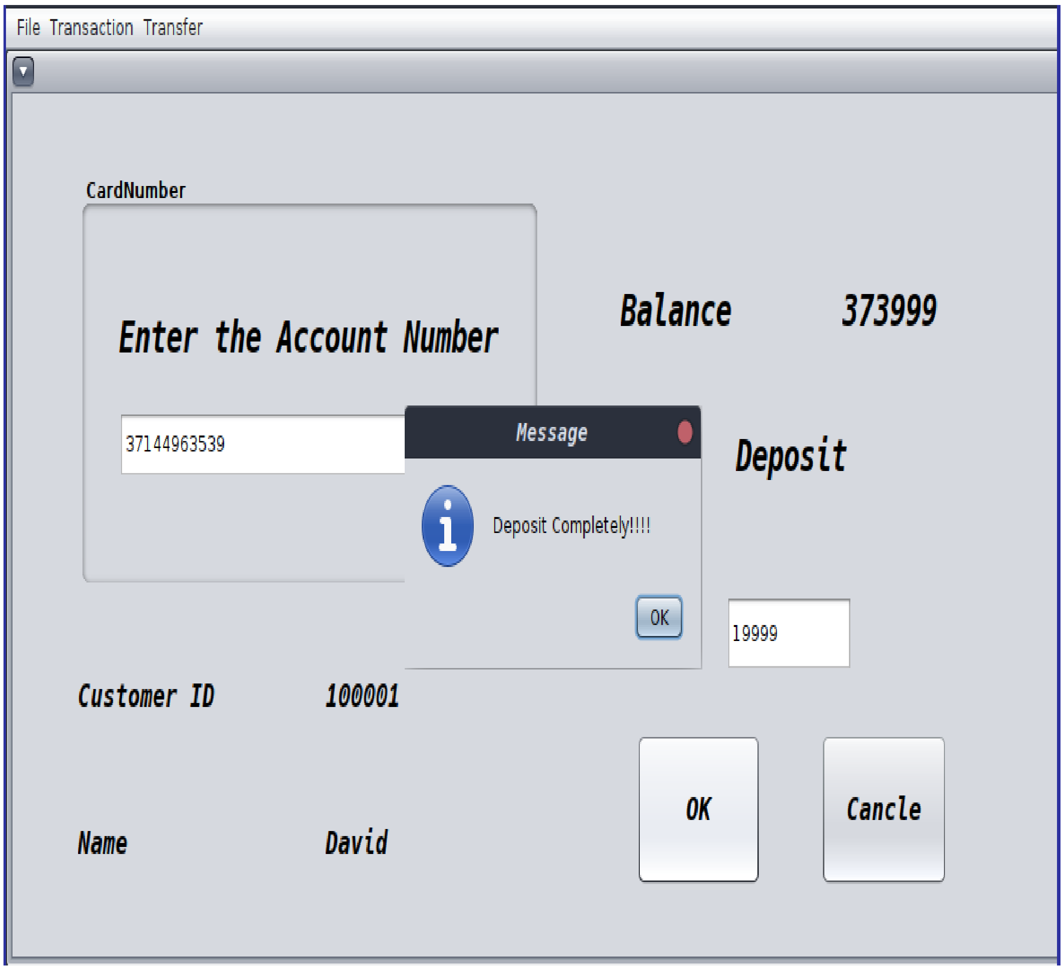
Deposit

Customer ID 100001

Name David

Figure 47: Finding Account Number.

Let's enter money in deposit field.



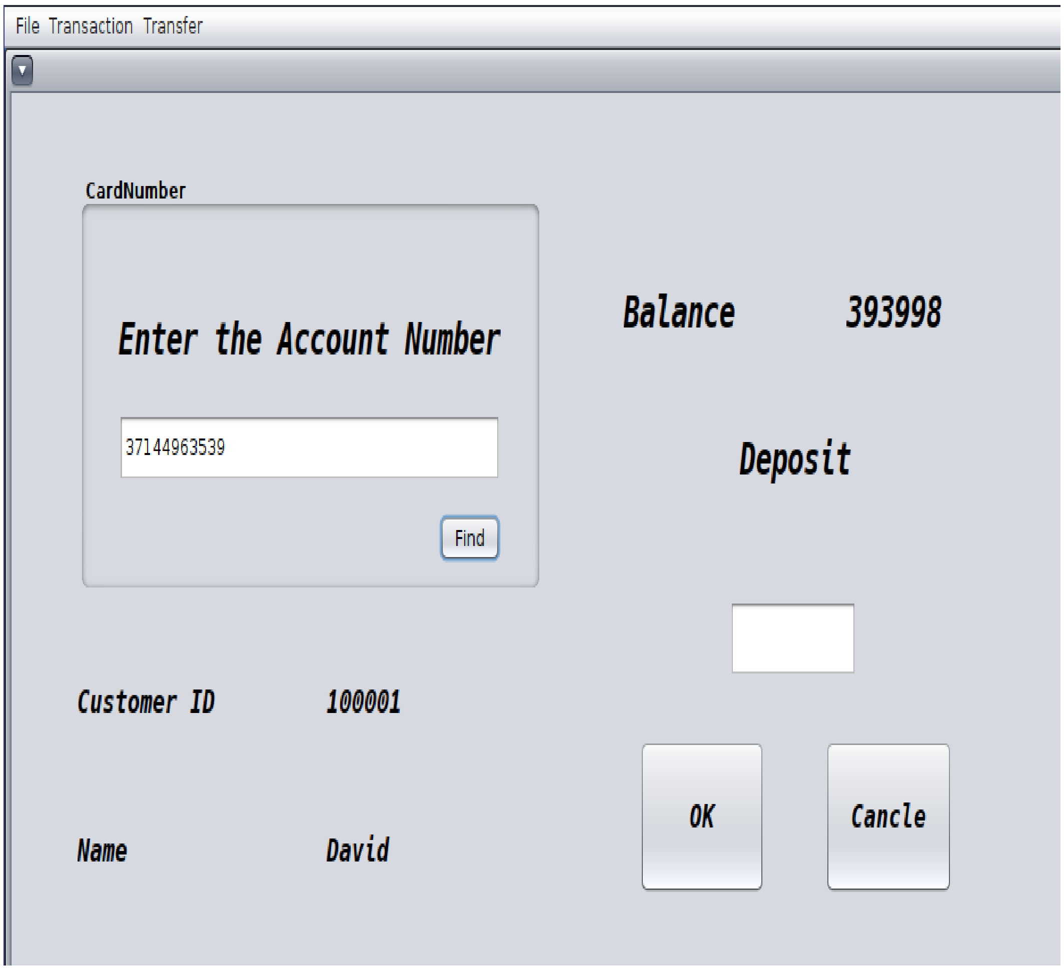
37144963539

Deposit Completely!!!!

19999

Figure 48: When entering **OK** button.

If you find this **Account Number** again, the result looks like this



37144963539

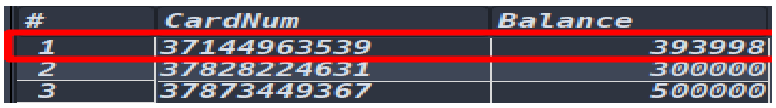
Balance 393998

Deposit

Customer ID 100001

Name David

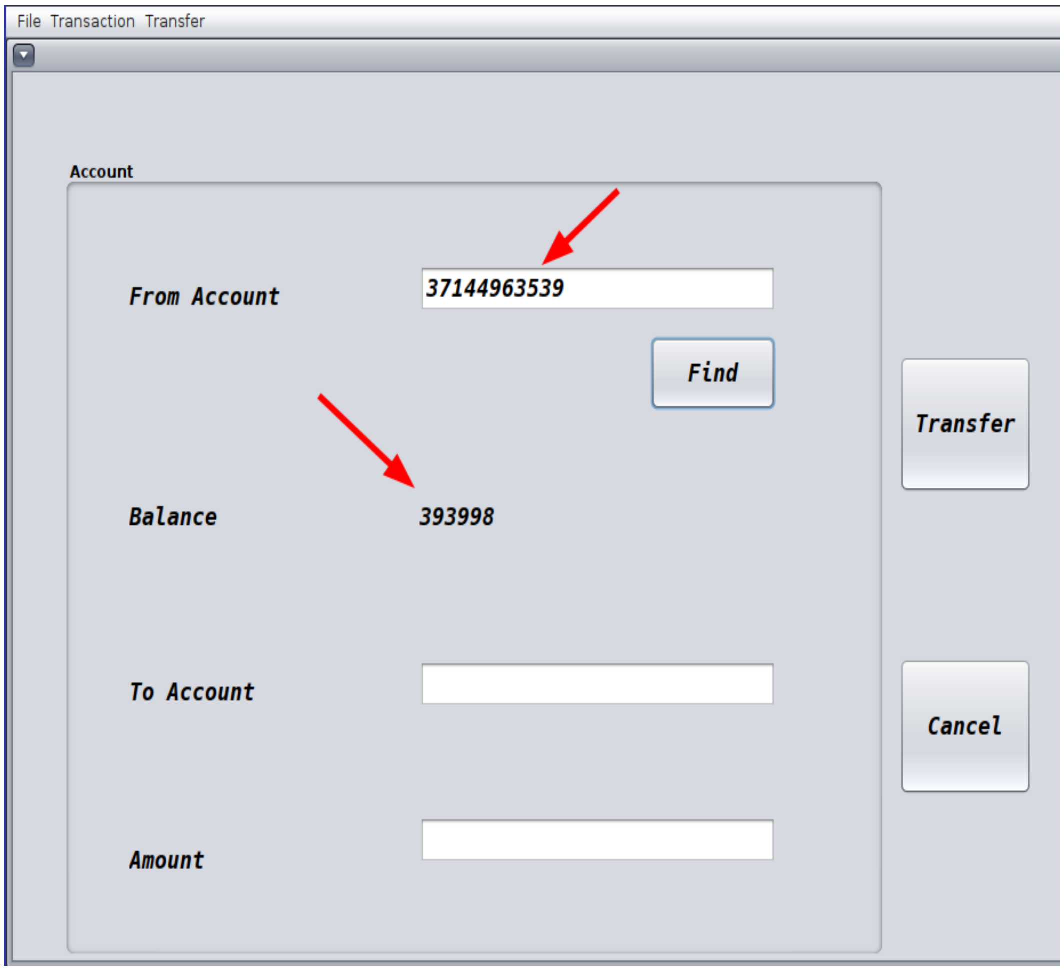
Figure 49: Result when finding again.



#	CardNum	Balance
1	37144963539	393998
2	37828224631	300000
3	37873449367	500000

Figure 50: Data in DebitCard's table.

Our application also help to transfer money between Account's number. Now, let see what's in **Transfer** interface.



From Account 37144963539

Balance 393998

To Account

Amount

Figure 51: Finding Account Number 37144963539.

Now we want to transfer money to Account Number 37873449367 with amount 59999 transferred.

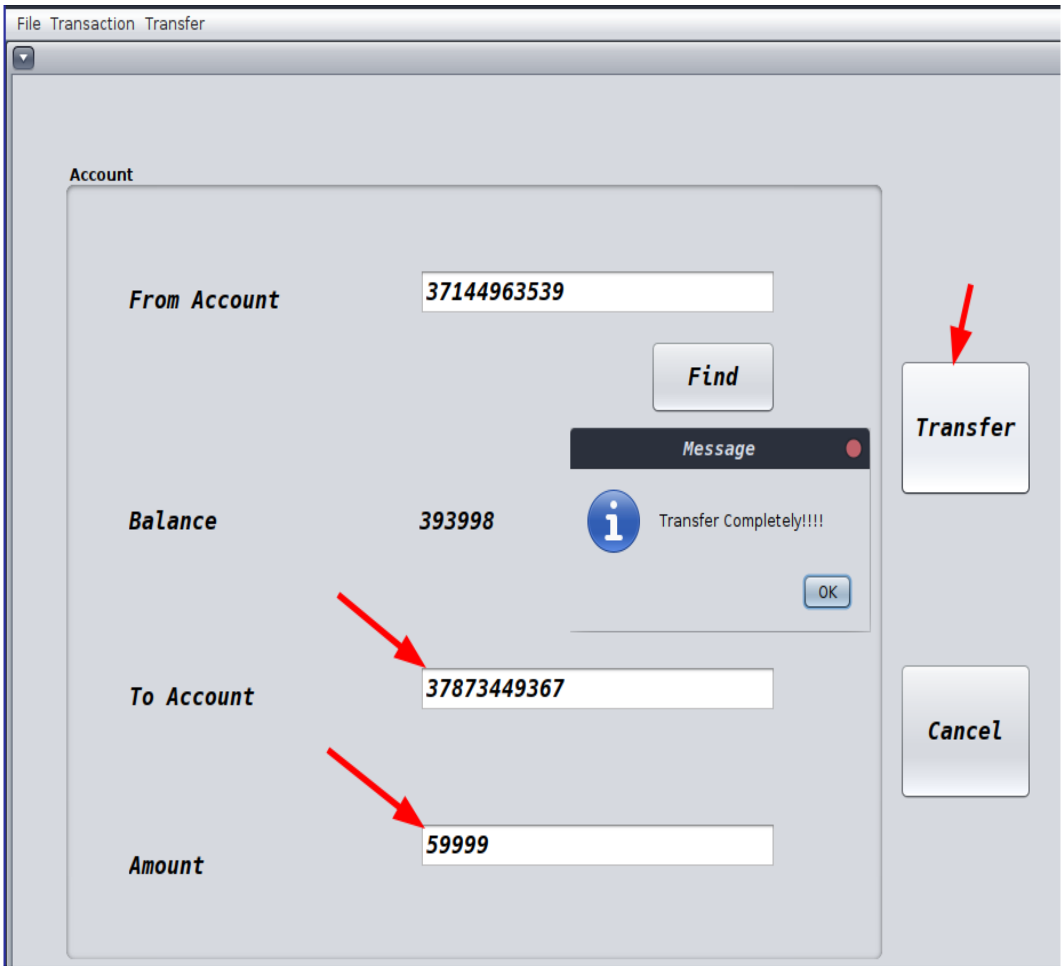
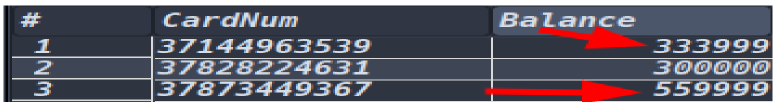


Figure 52: Transferring completely.

It will update DebitCard's table in our database.



#	CardNum	Balance
1	37144963539	333999
2	37828224631	300000
3	37873449367	559999

Figure 53: Result of updating DebitCard's table.

So, that is all for our application. And here is reference for demonstration. Demonstration for our application

13 Conclusion for assignment 2:

Firstly, we completed insertion initial data to the DBMS, we tried to make it consistent. The data will then be used for query. Secondly, we discussed about Normal Form and Functional Dependency. Following with its theory, we analyze our database based on that and we concluded that all relations in Bank Transaction database is in Boy-Codd Normal Form. In the next part, we discuss about security including access control, interface control, flow control, data encryption. We also have a demonstration about security on granting privilege.

The most interesting section to us is the two next sections. The first one is about tuning data. We do querying data, creating trigger and procedure for relations in our database. Trigger is very important in our database because it make our database consistent. We have demonstrated triggers for making insertion of online account transaction consistent and one for validate staff insertion. Further work is still need to make our whole database consistent. For other transaction relations, the step will be similar to the Online Transaction. The next one is about create a top application for our database. Although it is not totally completed, it provided basic functions for user to work with our database. Moreover, the application will be updated in the near future to make it work better with more functions.

14 References

- https://www.youtube.com/watch?v=JRN8I9C7XbE
- Ramez, Elmasri, Shamkant, B.Navathe-Fundamentals of Database Systems Pearson 2015.
- Here is our source code in github
- https://dev.mysql.com/doc/refman/8.0/en/regexp.html
- https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html
- https://dev.mysql.com/doc/index-enterprise.html
- https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html