CUSTOMER SIDE DATABASE FOR BANK

150116841 - HALE ŞAHİN

150114856 - ÖZGE YILDIRIM

ABOUT

All of the banks have a big data for storing and controlling. Their database are stored in a big inventory which is a department called ‘ Data Warehouse’ . There are customers, employees and other entities in a bank. But when they are complex inside, we choose the only the customer side problem for a bank.

The problem is that how to design a database for customer side of a bank efficiently and applicable. The database should be easy to use, it should allow to make four main operations; add, delete, update and select.

Our first aim was try to get what our customer wants to solve this problem. We have the real problem statement and we completely understand what is the problem. Then we think about the solution approaches.

Secondly, we determined our entity names and their attributes. We drew the ER diagram for this project. Then, we indicated the relations of the entities. When we have our first design, we extended and improved that design and we made sure that this new design doesn’t have any conflicts, error or missing points.

Lastly, we created the database according to the our design. Then we tested our design to get any errors and fix them before we present our project.

INFORMATION ABOUT THE TABLES AND ATTRIBUTES

The database will consist of tables bank, bank account, branch, customer, bank card, transactions,debit card, credit card tables.

Customer is the center entity for this project. Identifier for customer is customer id attribute, which is the primary key and can not be null, and should be unique for each entry. Customer has name composite attribute which includes name and surname. Customer entity has the phone number and the composite address information of customer. Bank Account entity is identified by account number information and account also has other attributes as follows: balance and account type. Balance attribute will keep the current amount of money in the account which is determined by account number and account type can be differ as deposit account or checking account etc. Bank entity is for the relation between customer and branches of bank, it makes the allocation for them. Branches identified by branch id and includes other attributes as branch name and branch city which indicates where this branch located. Bank Card entity is identified by unique card number. Expire date and card type are the attributes for card entity. Debit card is a sub type of Bank Card it has computed columns of card number and it has withdraw limit for the with drawing some amount of money for a day. Credit card is also a disjoint type of Bank Card. It has loan limit, current debt and minimum monthly payment amount as attributes. Transactions are identified by transaction number. Transaction type like payment, loan is an attribute for transaction. Another attribute is amount of the transaction.

RELATIONSHIPS

A customer can possess many accounts, and one or many accounts are possessed by one customer. A bank has many customers, one or many customer are customers of a bank. A bank may have many branches or not, but branches are included by a specific bank. A customer may have many cards, and one or many cards are owned by one customer. Credit Card and Debit Card are sub-types of card entity. They have all entities that card entity has. Customer cannot have both of these cards, have to choose one of them.

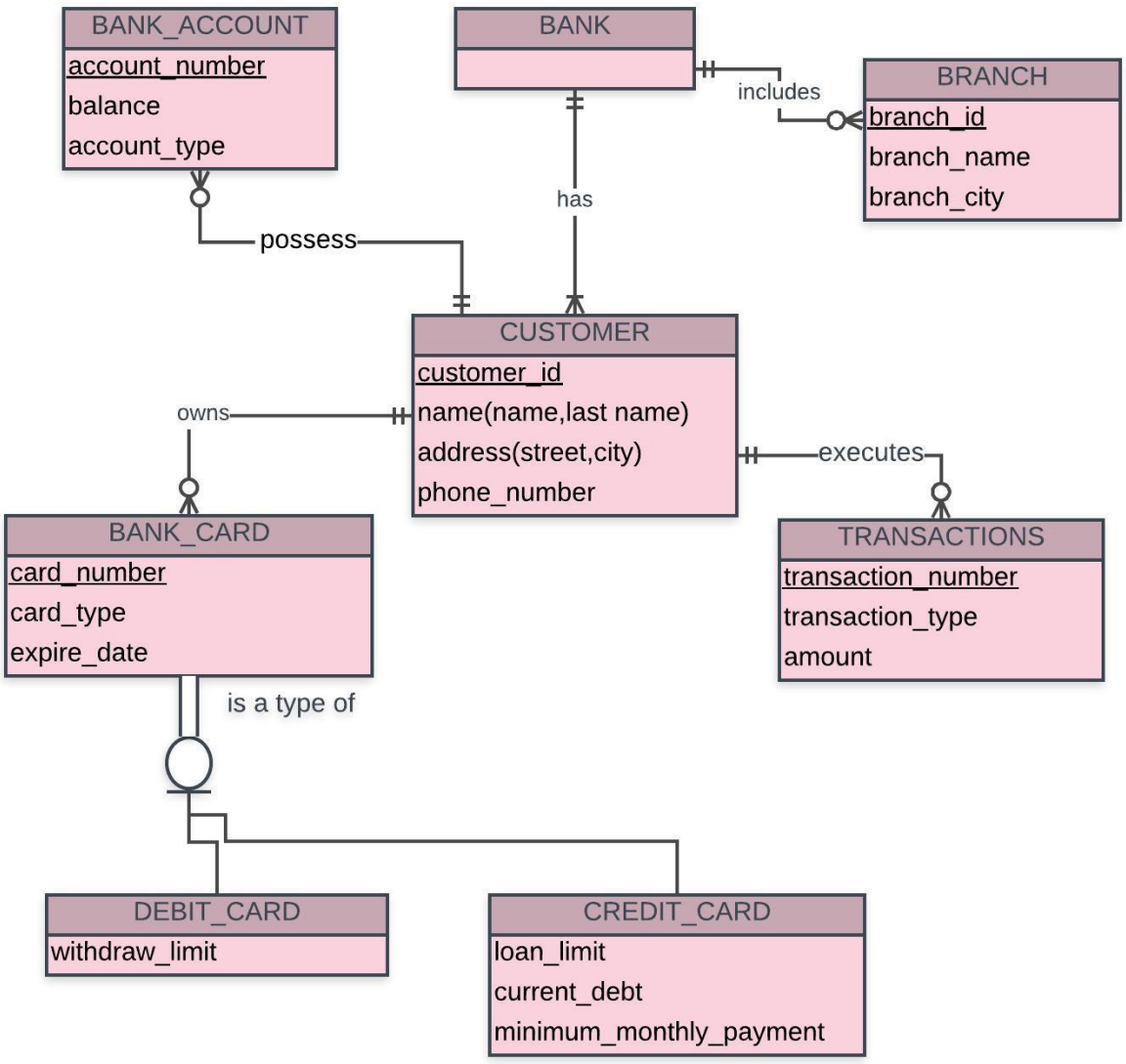
DATA AND REQUIREMENT ANALYSIS

Banks do not share information about their company. That is why data set for this project is created randomly by the team members. We will implement an interface for our database, that will have buttons like add a new customer, update an information, delete a data or call some data about customer or bank to see. This program will be easy to use for bank worker who does not know anything about database systems. They will click on the 'Add new customer' button and a screen will appear then they will enter the requested information which are customer name, address, phone number and customer id will be assigned automatically behind. During bank worker do this process, in the background actually insert function will be called with that button, and the entered information will be the values for insert queue. Same process will be done for other operations like delete or update.

When the loan limit information of a customer asked, then this data will be accessed by in order of customer id, then card number, then credit card table's loan limit attribute. All tables are connected and have relationships.

The last user which is in this case the workers of bank, cannot see the actual database tables. That is only available for Data Warehouse engineers. However, they can call some data by giving the identifier data values. Identifier is the way to access whole table for database systems. Updating and deleting a value should be done with considering that they can be copied or using in another tables like super/sub types. In that case the operation will be done with all the branches. Again it will be done without last user realizing.

E-R DIAGRAM



CONSTRAINTS -- INDICES -- DEFAULTS

Indices

We created indices for both uniqueness and primary keys : branch_id, customer_id. They are counting by the rule giving them in the constraint, and they makes easy to search in data.

Constraints

Customer table's primary key is customer_id, that' s constraint is customer_pk. Identity starts with 1 and increase 1 by 1 for each entry. Bank_account table's primary key is account_number and that's constraint is acc_pk. Bank_account also has foreign key (cust_id) and that's constraint is cust_id_fk which is referenced customer id from Customer table. Identity start the 100900001111145896 and it is increased by 2 for each entry . Branch table primary key is branch_id and that's constraint is branch_pk, it represents city code for branches uniquely. Bank table is a relation so it consists of foreign keys to bond them together. Foreign key cust_id, that' s constraint is bank_cust references customer id from customer table. Bank table's another foreign key bran_id, that's constraint is bank_cust2 which references branch id attribute column of the branch table. Bank_card table' s primary key card_number and that's constraint is card_pk. Identity start the 1111222233334444 and it is increased by 1 for each new account. Debit card table' s foreign keys are card_number, card_type and credit card table' s foreign keys also are card_number and card_type. They are both have same foreign keys that referenced from their supertype bank card. Transactions table' s primary key transaction_number, that's constraint transaction_pk and this primary key is also calculated automatically instead of giving by hand it starts fro this number 11112465244 and counts plus 4 for each new entry. Foreign key is cust_id, that's foreign key is transaction_fk which referenced customer id from customer table. All these identity counters helped to make easy to insert new entries to the tables.

Defaults

We used default in only one place which was in the Bank Account table, balance attribute. We defined this attribute as default value with #0, while we were creating the table. So that, for each new record of account, all accounts will be opened with #0 balance.

Functions

We created one unction for the assigning city id for each city name entry into the branch. It used for computed column.

```
CREATE OR ALTER FUNCTION BranchCity
(@city VARCHAR(15))
returns INT
AS
BEGIN
    DECLARE @c_id int;
    SELECT @c_id = CASE @city
         WHEN 'Central City' THEN 1
         WHEN 'Chicago' THEN 2
         WHEN 'Columbia' THEN 3
         WHEN 'Krypton' THEN 4
         WHEN 'Minnesota' THEN 5
         WHEN 'Moscow' THEN 6
         WHEN 'Mumbai' THEN 7
         WHEN 'National City' THEN 8
         WHEN 'New Islands' THEN 9
         WHEN 'New Jersey' THEN 10
         WHEN 'New Orleans' THEN 11
         WHEN 'New York' THEN 12
         WHEN 'Ohio' THEN 13
         WHEN 'Oklahoma' THEN 14
         WHEN 'Paris' THEN 15
         WHEN 'San Francisco' THEN 16
         WHEN 'Star City' THEN 17
         WHEN 'Texas' THEN 18
         END;
RETURN (@c_id);
END;
GO
```

Stored Procedures

We created procedures to get service for manipulation of data easy and more convenient to use. Our procedures are for insert into bank table, insert into bank account table, insert into bank card table, insert into branch table, insert into credit card table, insert into debit card table, insert into customer table, insert new transaction, update bank account table.

Here are some examples;

```
--7 INSERT CREDIT BANK CARD
CREATE PROCEDURE InsertCreditCard
@cid INT,
@loan INT,
@dept NUMERIC(20,5)
AS
DECLARE @cardno NUMERIC(16,0);
DECLARE @minimum_payment NUMERIC(20,5);
SELECT @cardno=card_number FROM BANK_CARD
WHERE @cid=BANK_CARD.c_id
set @minimum_payment=(@dept*30)/100;
INSERT INTO CREDIT_CARD
VALUES(@cardno,@loan,@dept,@minimum_payment)
GO
```

Here, this procedure is for the adding new record to the credit card entity easily. We take customer id, loan amount, and dept amount information from the user. Then we use customer id information to get card number of that person because card number knowledge is required in order to insert field into credit table. Then we calculated minimum payment value by getting the 30 % of current dept value. Then we called insert function.

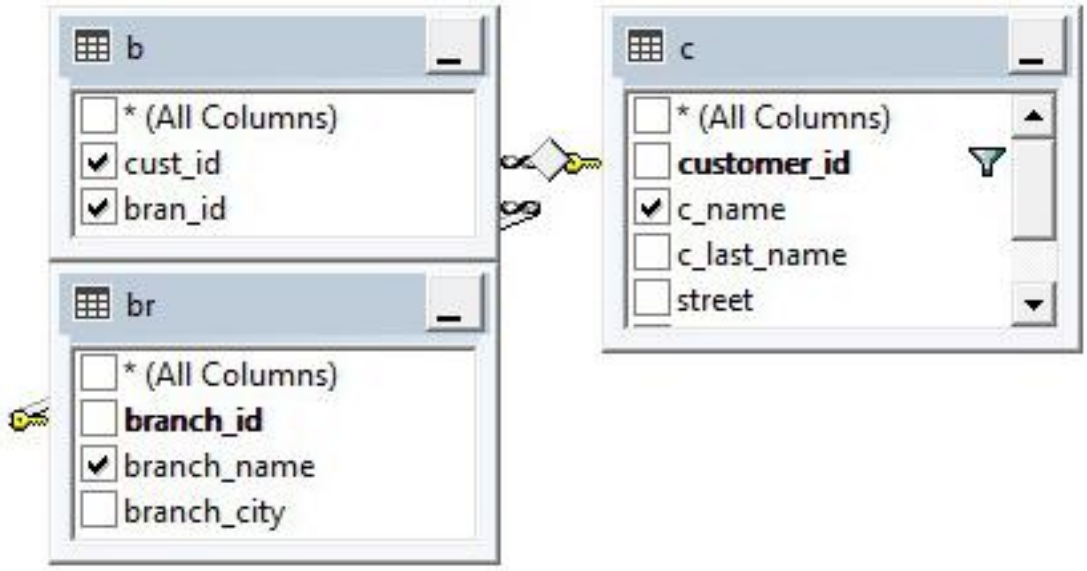
```
--8 UPDATE BALANCE IN BANK ACCOUNT
CREATE PROCEDURE UpdateBankAccount
@id INT,
@type VARCHAR(15),
@amount INT
AS
DECLARE @balance INT;
SELECT @balance=balance FROM BANK_ACCOUNT
WHERE @id=BANK_ACCOUNT.cust_id
IF @type='Add'
SET @balance=@balance + @amount;
ELSE
SET @balance=@balance - @amount;
UPDATE BANK_ACCOUNT SET balance = @balance
WHERE @id=BANK_ACCOUNT.cust_id
GO
```

This one was for updating the balance column of the bank account table. Customer id, transaction type and amount are taken from the user. Then if transaction is addition make the current balance value increased by the given amount and opposite for otherwise.

Views

We created some views for bank_info, card_info, customer_branch, debit, top15_customer_branch, transact. We used chained select statements, joins and conditions.

Here are some examples for views:



Above is the diagram for below query of view we created. It indicates which columns are included in the view. This was for the understanding of view. We selected from first 15 entry of the customer table which joined for 2 other tables columns to see which person is registered for which branch in which city.

```
CREATE VIEW TOP15_CUSTOMER_BRANCH AS
SELECT cust_id,bran_id,c_name,branch_name
FROM BANK b, CUSTOMER c,BRANCH br
WHERE c.customer_id<15
AND b.cust_id=c.customer_id
AND b.bran_id=br.branch_id
```

—

In the next view, we used chained select statements to get the branch names which are not ‘National Bank’ specifically.

```
CREATE VIEW CUSTOMER_BRANCH AS
SELECT c_name,c_last_name,branch_name FROM CUSTOMER c, BRANCH b,BANK
WHERE BANK.cust_id=c.customer_id
AND BANK.bran_id=b.branch_id
AND bran_id IN(
SELECT branch_id FROM BRANCH
WHERE branch_name<>'National Bank')
```

For the third one we used join annd conditional statements again. We bring some information of people who has withdraw limit more than 1000.

```
CREATE VIEW DEBIT AS
SELECT dc.card_number,dc.card_type,withdraw_limit,c_id
FROM DEBIT_CARD dc LEFT JOIN BANK_CARD bc
ON dc.card_number=bc.card_number
WHERE withdraw_limit >1000
```

Triggers

We created three triggers. First and second is for the recording log for each insertion and deletion into customer table.

```
CREATE TRIGGER insCUSTOMERLOG ON CUSTOMER
 AFTER INSERT AS
BEGIN

    INSERT INTO CUSTOMER_LOG(cid,ACTIONTYPE,ACTIONDATE)
    select i.customer_id, 'I', GETDATE()   FROM inserted i;
 END

CREATE TRIGGER delete_CUSTOMER_LOG ON CUSTOMER
 AFTER DELETE AS
BEGIN
    DECLARE @hasRecords int;
    select @hasRecords=COUNT(*) from BANK_ACCOUNT
    where BANK_ACCOUNT.cust_id=(select customer_id from deleted)
      IF (@hasRecords) > 0
    BEGIN
   RAISERROR('You cannot delete corresponding customer it has records in bank account table.',16, 1)
    ROLLBACK TRANSACTION --rollback delete operation
    END
     IF (@hasRecords) = 0
    BEGIN
   INSERT INTO CUSTOMER_LOG(cid,ACTIONTYPE,ACTIONDATE)
    select i.customer_id, 'D', GETDATE()   FROM deleted i;
    END
 END;
```

Third one is for the balance changes log file. When some change occurred in balance, whether is addition or extraction it kept and the new value of the balance is recorded.

```sql
CREATE TABLE BALANCE_LOG
(
    cid INT NOT NULL,
    ACTIONTYPE varchar(15),
    last_status int
);


CREATE TRIGGER insert_balanceLOG ON TRANSACTIONS
AFTER INSERT AS
BEGIN
    DECLARE @balance int;
    SELECT @balance=balance FROM BANK_ACCOUNT
    WHERE BANK_ACCOUNT.cust_id=(SELECT i.cust_id FROM inserted i)

    INSERT INTO BALANCE_LOG(cid,ACTIONTYPE,last_status)
    select i.cust_id, i.transaction_type, @balance FROM inserted i;
END;
```