SQL Programming

Instructor: Hamid Rajaee

By: Sara Khosravi

December 2020

Project Phase 1

Question/Problem 1

1. Create a database for a banking application called “Bank”. [Basic]
2. Create all the tables mentioned in the database diagram. [Moderate]
3. Create all the constraints based on the database diagram. [Advanced]
4. Insert at least 5 rows in each table. [Basic]

Project Phase 2

Question/Problem 1

1. Create a view to getting all customers with checking accounts from ON province.[Moderate]
2. Create a view to getting all customers with a total account balance (including the interest rate) greater than 5000. [Advanced]
3. Create a view to getting counts of checking and savings accounts by customers. [Moderate]
4. Create a view to get any particular user’s login and password using AccountId. [Moderate]
5. Create a view to getting all customers’ overdraft amount. [Moderate]
6. Create a stored procedure to add “User_” as a prefix to everyone’s login (username). [Moderate]
7. Create a stored procedure that accepts AccountId as a parameter and returns the customer’s full name. [Advanced]

8. Create a stored procedure that returns error logs inserted in the last 24 hours. [Advanced]

9. Create a stored procedure that takes a deposit as a parameter and updates the CurrentBalance value for that particular account. [Advanced]

10. Create a stored procedure that takes a withdrawal amount as a parameter and updates

11. Prepare a report to describe the project. [Moderate]

12. Prepare a presentation for the project. [Moderate]

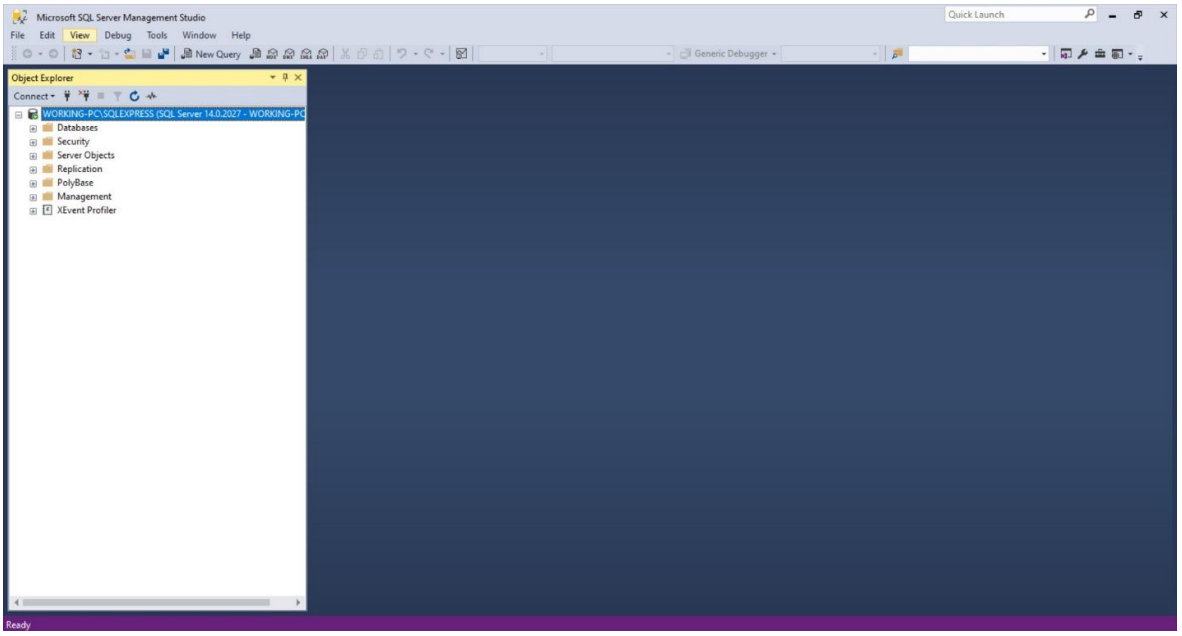
Phase One

1. Create a database for a banking application called “Bank”. [Basic]

What is a database?

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS).

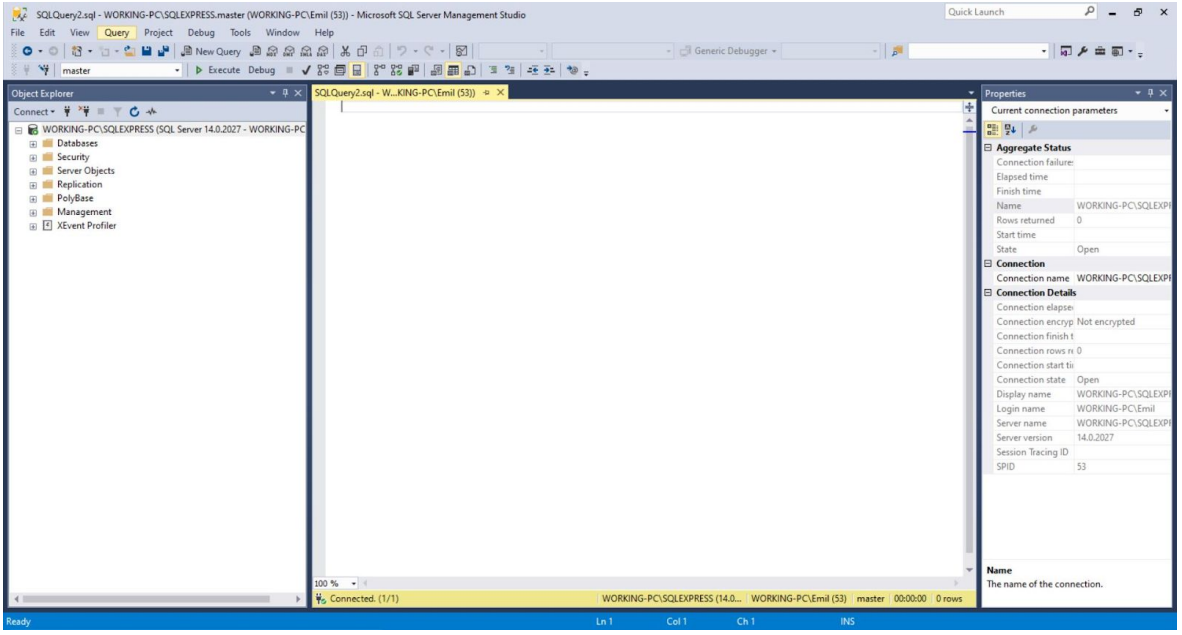
We’ll make it more fun by creating a new database. After clicking on the New Query, the new window opens and we’re able to type something in. Before typing anything, we should be sure we’re typing it in the right way. T-SQL is a language and as such it has its’ words – a set of rules on how to write different commands.



Luckily, one of these commands is the SQL Create Database command. You can see the full T-SQL Create Database syntax on Microsoft pages. I'll simplify it a lot and go only with the most basic form. In order to create a new database on our server, we need to use the following command:

Create Database BANK

Click on the + next to the folder Databases, and besides two folders, you'll also see that BANK had been created.



We'll run a command:

```
CREATE DATABASE  Bank;

Go
```

After running this command, our database is created.

2. Create all the tables mentioned in the database diagram. [Moderate]

CREATE TABLE is the keyword telling the database system what you want to do. In this case, you want to **create** a new **table**. The unique name or identifier for the **table** follows the **CREATE TABLE** statement. Then in brackets comes the list defining each column in the **table** and what sort of data type it is.

```
CREATE TABLE table_name (
 column1 datatype,
 column2 datatype,
 column3 datatype,
   ....
);
```

Most of the time we need to use Primary Key and Foreign Key in a Table. Therefore, the definition of Primary Key and Foreign Key define below:

Primary Key:

- The primary key has to be unique and not empty(not null) The PRIMARY KEY constraint uniquely identifies each record in a table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).
- The primary key is used to uniquely identify each row in a table
- The primary key does not allow nulls

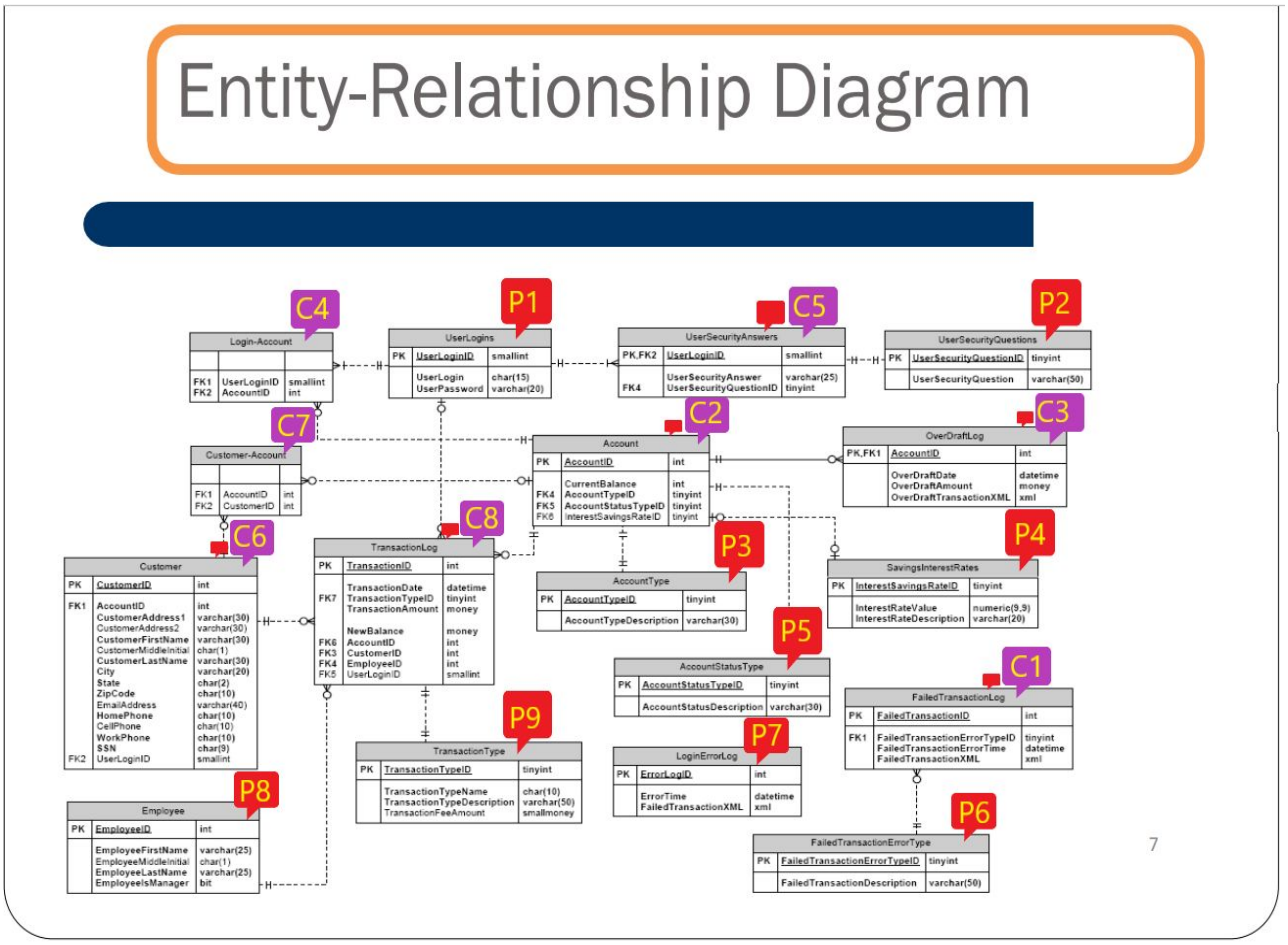
Foreign Key:

- A FOREIGN KEY is a key used to link two tables together.
- A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

- The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

The code wrote on based on the arrange table below:

1. First, the table has a primary key
2. Second, the table has less than Foreign Key



```sql
create database dbBank_SARA


go
use dbBank_SARA
go

--create a table that includes information about UserLogins.....ParentKey 1.........

create table UserLogins
(UserLoginID smallint not null primary key,
UserLogin char(15),
UserPassword varchar(20))
go

 --create a table that includes information about UserSecurityQuestions.....ParentKey 2....
 create table UserSecurityQuestions
 (UserSecurityQuestionID tinyint not null primary key,
 UserSecurityQuestion varchar(50))
 go

 --create a table that includes information about AccountTypes.....ParentKey 3.....
 create table AccountTypes
 (AccountTypeID tinyint not null primary key,
 AccountTypeDescription varchar(30))
 go

 --create a table that includes information about SavingInterestRates.....ParentKey 4.....
 create table SavingInterestRates
 (InterestSavingRateID tinyint not null primary key,
 InterestRateValue Numeric(9,9),
 InterestRateDescription varchar (20))
 go

  --create a table that includes information about AccountStatuseType.....ParentKey 5.....
 create table AccountStatuseType
 (AccountStatuseTypeID tinyint not null primary key,
  AccountStatuseDescription varchar (30))
 go

 --create a table that includes information about TransactionType.....ParentKey 6.....
```

```sql
 --create a table that includes information about TransactionType.....ParentKey 6.....
 create table TransactionType
 (TransactionTypeID tinyint not null primary key,
  TransactionTypeName varchar (10),
  TransactionTypeDescription varchar (50),
  TransactionFeeAmount smallmoney)
  go

    --create a table that includes information about LoginErrorLog.....ParentKey 7.....
 create table LoginErrorLog
 (LoginErrorID int not null primary key,
  ErrorTime datetime,
  FailedTransactionXML xml)
 go

 --create a table that includes information about Employee.....ParentKey 8.....
 create table Employee
 (EmployeeID int not null primary key,
  EmployeeFirstName varchar (25),
  EmployeeMiddleInitial char (1),
```

```
 EmployeeLastName varchar (25),
 EmployeeIsManager bit)
 go

   --create a table that includes information about FailedTransactionErrorType.....ParentKey 9.....
 create table FailedTransactionErrorType
 (FailedTransactionErrorTypeID tinyint not null primary key,
  FailedTransactionDescription varchar(50))
 go

 --create a tables that makes a relation between FailedTransactionLog and FailedTransactionErrorType.......ChildKey2 and PrimaryKey9

Create table FailedTransactionLog
(FailedTransactionID int primary key,
FailedTransactionErrorTypeID tinyint foreign key references FailedTransactionErrorType(FailedTransactionErrorTypeID),
FailedTransactionErrorTime datetime,
FailedTransactionXML xml)
go

--create a tables that makes a relation between Account, AccountType, AccountStatuseType and SavingInterestRates .......ChildKey7, PrimaryKey3, PrimaryKey5 an
```

```
--create a tables that makes a relation between Account, AccountType, AccountStatuseType and SavingInterestRat

Create table Account
(AccountID int not null primary key ,
CurrentBalance int,
AccountTypeID tinyint foreign key references AccountTypes(AccountTypeID),
AccountStatuseTypeID tinyint foreign key references AccountStatuseType(AccountStatuseTypeID),
InterestSavingRateID tinyint foreign key references SavingInterestRates(InterestSavingRateID))
go


--create a tables that makes a relation between OverDraftLog and Account.......ChildKey 1 and ChildKey7

Create table OverDraftLog
(AccountID int  primary key foreign key references Account(AccountID),
OverDraftDate datetime,
OverDraftAmount money,
OverDraftTransactionXML xml)
go
```

```sql
--create a tables that makes a relation between Login_Account, UserLogins and Account.......ChildKey3, PrimaryKey1, ChildKey7
Create table Login_Account
(UserLoginID smallint foreign key references UserLogins(UserLoginID),
AccountID int foreign key references Account(AccountID))
go

--create a tables that makes a relation between UserSecurityAnswers, UserLogins and UserSecurityQuestions.......ChildKey4, PrimaryKey1

Create table UserSecurityAnswer
(UserLoginID smallint  primary key foreign key references UserLogins(UserLoginID),
UserSecurityAnswer varchar(25),
UserSecurityQuestionID tinyint foreign key references UserSecurityQuestions(UserSecurityQuestionID))
go

--create a tables that makes a relation between Customer, Account and UserLogins.......ChildKey6, childKey7 and PrimaryKey9 ..childKey

Create table Customer
(CustomerID int not null primary key,
--AccountID int foreign key references Account(AccountID),
CustomerAddress1 varchar (30),
```

```sql
--create a tables that makes a relation between Customer, Account and UserLogins.......ChildKey6, childKey7 and PrimaryKey9 ..childKey8

Create table Customer
(CustomerID int not null primary key,
--AccountID int foreign key references Account(AccountID),
CustomerAddress1 varchar (30),
CustomerAddress2 varchar (30),
CustomerFirstName varchar (30),
CustomerMiddleInitial varchar (1),
CustomerLastName varchar (30),
City varchar (20),
State char(2),
Zipcode char (10),
EmailAddress varchar (40),
HomePhone char (10),
CellPhone char(10),
WorkPhone char (10),
SSN char (1),
UserLoginID smallint foreign key references UserLogins(UserLoginID))
go
```

```sql
--create a tables that makes a relation between Customer_Account, Customer and Account.......ChildKey5, ChildKey6 and ChildKey7
Create table Customer_Account
(CustomerID int foreign key references Customer(CustomerID),
AccountID int foreign key references Account(AccountID))
go

--create a tables that makes a relation between TransactionLog, TransactionType, Account, Customer,Employee and Userlogins
--........ChildKey8, PrimaryKey6, ChildKey7, ChildKey6, PrimaryKey8 and PrimaryKey1
Create table TransactionLog
(TransactionID int not null primary key ,
TransactionDate datetime,
TransactionTypeID tinyint foreign key references TransactionType(TransactionTypeID),
TransactionAmount money,
NewBalance money,
AccountID int foreign key references Account(AccountID),
CustomerID int foreign key references Customer(CustomerID),
EmployeeID int foreign key references Employee(EmployeeID),
UserloginID smallint foreign key references Userlogins(UserloginID))
go
/*****************************************************************/
```

The following table is the result of the code:

1. **UserLogins Table:**

Column Name	Data Type	Allow Nulls
UserLoginID	smallint	☐
UserLogin	varchar(50)	☐
UserPassword	varchar(20)	☐
		☐

2. **UserSecurityQuestions Table:**

Column Name	Data Type	Allow Nulls
UserSecurityQuestionID	tinyint	☐
UserSecurityQuestion	varchar(50)	☐
		☐

3. **AccountType Table:**

Column Name	Data Type	Allow Nulls
AccountTypeID	tinyint	☐
AccountTypeDescription	varchar(30)	☐
		☐

4. **SavingInterestsRates Table:**

Column Name	Data Type	Allow Nulls
InterestSavingRatesID	tinyint	☐
InterestRatesValue	numeric(9, 9)	☐
InterestRatesDescription	varchar(20)	☐
		☐

5. **AccountStatuseType Table:**

Column Name	Data Type	Allow Nulls
AccountStatusTypeID	tinyint	☐
AccountStatusTypeDescription	varchar(30)	☐
		☐

6. TransactionType Table:

Column Name	Data Type	Allow Nulls
TransactionTypeID	tinyint	☐
TransactionTypeName	char(10)	☐
TransactionTypeDescription	varchar(50)	☑
TransactionFeeAmount	smallmoney	☑
		☐

7. LoginErrorLog Table:

Column Name	Data Type	Allow Nulls
ErrorLogID	int	☐
ErrorTime	datetime	☐
FailedTransactionXML	xml	☑
		☐

8. Employee Table:

Column Name	Data Type	Allow Nulls
EmployeeID	int	☐
EmployeeFirstName	varchar(25)	☐
EmployeeMiddleInitial	char(1)	☑
EmployeeLastName	varchar(25)	☑
EmployeeisManager	bit	☑
		☐

9. FailedTransactionErrorType Table:

Column Name	Data Type	Allow Nulls
FailedTransactionID	int	☐
FailedTransactionErrorTypeID	tinyint	☐
FailedTransactionErrorTime	datetime	☑
FailedTransactionErrorXML	xml	☑
		☐

10. FailedTransactionLog Table:

Column Name	Data Type	Allow Nulls
FailedTransactionID	int	☐
FailedTransactionErrorTypeID	tinyint	☐
FailedTransactionErrorTime	datetime	☑
FailedTransactionErrorXML	xml	☑
		☐

11. Account Table:

Column Name	Data Type	Allow Nulls
AccountID	int	☐
CurrentBalance	int	☐
AccountTypeID	tinyint	☐
AccountStatusTypeID	tinyint	☐
InterestSavingRatesID	tinyint	☐
		☐

12. OverDraftLog Table:

Column Name	Data Type	Allow Nulls
AccountID	int	☐
OverDraftDate	datetime	☐
OverDraftAmount	money	☐
OverDraftTransactionXML	xml	☐
		☐

13. Login_Account Table:

Column Name	Data Type	Allow Nulls
UserLoginID	smallint	☐
AccountID	int	☐
		☐

14. UserSecurityAnswer Table:

Column Name	Data Type	Allow Nulls
UserLoginID	smallint	☐
UserSecurityAnswers	varchar(25)	☐
UserSecurityQuestionID	tinyint	☐
		☐

15. Customer:

Column Name	Data Type	Allow Nulls
CustomerID	int	☐
AccountID	int	☐
CustomerAddress1	varchar(30)	☐
CustomerAddress2	varchar(30)	☑
CustomerFirstName	varchar(30)	☐
CustomerMiddleInitial	char(1)	☑
CustomerLastName	varchar(30)	☐
City	varchar(20)	☐
State	char(2)	☐
ZipCode	char(10)	☐
EmailAddress	char(40)	☐
HomePhone	varchar(10)	☐
CellPhone	varchar(10)	☐
WorkPhone	varchar(10)	☐
SSN	varchar(9)	☑
UserLoginID	smallint	☐
		☐

16. CustomerAccount Table:

Column Name	Data Type	Allow Nulls
AccountID	int	☐
CustomerID	int	☐
		☐

17. TransactionLog Table

Column Name	Data Type	Allow Nulls
TransactionID	int	☐
TransactionDate	datetime	☐
TransactionTypeID	tinyint	☐
TransactionAmount	money	☐
NewBalance	money	☐
AccountID	int	☐
CustomerID	int	☐
EmployeeID	int	☐
UserLoginID	smallint	☐
		☐

3. Create all the constraints based on the database diagram. [Advanced]

You can use the keyword IDENTITY as the data type to the column along with the PRIMARY KEY constraint when creating the table. In here the first '1' means the starting value and the second '1' is the incrementing value.

SQL Create Constraints:

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

SQL NOT NULL ConstraintBy default, a column can hold NULL values. The NOT NULL constraint enforces a column to NOT accept NULL values. This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

SQL PRIMARY KEY ConstraintThe PRIMARY KEY constraint uniquely identifies each record in a table. Primary keys must contain UNIQUE values, and cannot contain NULL values. A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

SQL FOREIGN KEY ConstraintA FOREIGN KEY is a key used to link two tables together. A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table. The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a table. Primary keys must contain UNIQUE values, and cannot contain NULL values. A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

SQL FOREIGN KEY Constraint

A FOREIGN KEY is a key used to link two tables together. A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table. The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

SQL Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy

and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table-level constraints apply to the whole table.

The following constraints are commonly used in SQL:

NOT NULL - Ensures that a column cannot have a NULL value

UNIQUE - Ensures that all values in a column are different

PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

FOREIGN KEY - Uniquely identifies a row/record in another table

CHECK - Ensures that all values in a column satisfy a specific condition

DEFAULT - Sets a default value for a column when no value is specified

INDEX - Used to create and retrieve data from the database very quickly

The related code has been shown below:

```sql
/*....................SARA KHOSRAVI................*/
/*....................PHASE ONE...................*/

CREATE DATABASE Bank;

 Go

 use Bank;
 GO
 --create a table that includes information about UserLogins.....ParentKey 1.....

CREATE TABLE UserLogins
(UserLoginID SMALLINT NOT NULL IDENTITY(1,1),
    UserLogin VARCHAR(50) NOT NULL,
    UserPassword VARCHAR(20) NOT NULL,
    CONSTRAINT PKUL_UserLoginID PRIMARY KEY(UserLoginID));
 Go

  --create a table that includes information about UserSecurityQuestions.....ParentKey 2.....
 CREATE TABLE UserSecurityQuestions
(   UserSecurityQuestionID TINYINT NOT NULL IDENTITY(1,1),
    UserSecurityQuestion VARCHAR(50) NOT NULL,
    CONSTRAINT PKUSQ_UserSecurityQuestionID PRIMARY KEY(UserSecurityQuestionID));
 GO
  --create a table that includes information about AccountTypes.....ParentKey 3.....
CREATE TABLE AccountType
(   AccountTypeID TINYINT NOT NULL IDENTITY(1,1),
    AccountTypeDescription VARCHAR(30) NOT NULL,
    CONSTRAINT PKAT_AccountTypeID PRIMARY KEY(AccountTypeID));
 GO
  --create a table that includes information about SavingInterestRates.....ParentKey 4.....
 CREATE TABLE SavingsInterestRates
(   InterestSavingRatesID TINYINT NOT NULL IDENTITY(1,1),
    InterestRatesValue NUMERIC(9,9) NOT NULL,
    InterestRatesDescription VARCHAR(20) NOT NULL,
    CONSTRAINT PKSIR_InterestSavingRatesID PRIMARY KEY(InterestSavingRatesID));
 GO
```

```sql
--create a table that includes information about AccountStatuseType.....ParentKey 5.....
CREATE TABLE AccountStatusType
(   AccountStatusTypeID TINYINT NOT NULL IDENTITY(1,1),
    AccountStatusTypeDescription VARCHAR(30) NOT NULL,
    CONSTRAINT PKAST_AccountStatusTypeID PRIMARY KEY(AccountStatusTypeID));
GO


--create a table that includes information about FailedTransactionErrorType.....ParentKey 6.....
CREATE TABLE FailedTransactionErrorType
(FailedTransactionErrorTypeID TINYINT NOT NULL IDENTITY(1,1),
    FailedTransactionErrorTypeDescription VARCHAR(50) NOT NULL,
    CONSTRAINT PKFTET_FailedTransactionErrorTypeID PRIMARY KEY(FailedTransactionErrorTypeID));
GO

    --create a table that includes information about LoginErrorLog.....ParentKey 7.....
CREATE TABLE LoginErrorLog
(ErrorLogID INT NOT NULL IDENTITY(1,1),
    ErrorTime DATETIME NOT NULL,
    FailedTransactionXML XML,
    CONSTRAINT PKLEL_ErrorLogID PRIMARY KEY(ErrorLogID));
Go

--create a table that includes information about Employee.....ParentKey 8.....
CREATE TABLE Employee
(EmployeeID INT NOT NULL IDENTITY(1,1),
    EmployeeFirstName VARCHAR(25) NOT NULL,
    EmployeeMiddleInitial CHAR(1),
    EmployeeLastName VARCHAR(25),
    EmployeeisManager BIT,
    CONSTRAINT pk_E_EmployeeID PRIMARY KEY(EmployeeID))
GO

   --create a table that includes information about TransactionType.....ParentKey 9.....
CREATE TABLE TransactionType
(TransactionTypeID TINYINT NOT NULL IDENTITY(1,1),
    TransactionTypeName CHAR(10) NOT NULL,
    TransactionTypeDescription VARCHAR(50),
    TransactionFeeAmount SMALLMONEY,
    CONSTRAINT pk_TT_TransactionTypeID PRIMARY KEY(TransactionTypeID))
GO
```

```sql

 --create a tables that makes a relation between FailedTransactionLog and FailedTransactionErrorType.......ChildKey2 and PrimaryKey9

CREATE TABLE FailedTransactionLog
(   FailedTransactionID INT NOT NULL IDENTITY(1,1),
    FailedTransactionErrorTypeID TINYINT NOT NULL,
    FailedTransactionErrorTime DATETIME,
    FailedTransactionErrorXML XML,
    CONSTRAINT pkFTL_FailedTransactionID PRIMARY KEY(FailedTransactionID),
    CONSTRAINT fkFTET_FailedTransactionErrorTypeID FOREIGN KEY(FailedTransactionErrorTypeID) REFERENCES FailedTransactionErrorType(FailedTransactionErrorTypeID));
 GO

 --create a tables that makes a relation between Account, AccountType, AccountStatuseType and SavingInterestRates .......ChildKey7, PrimaryKey3, PrimaryKey5 and PrimaryKey4

CREATE TABLE Account
(   AccountID INT NOT NULL IDENTITY(1,1),
    CurrentBalance INT NOT NULL,
    AccountTypeID TINYINT NOT NULL REFERENCES AccountType (AccountTypeID),
    AccountStatusTypeID TINYINT NOT NULL,
    InterestSavingRatesID TINYINT NOT NULL,
    CONSTRAINT pk_A_AccounID PRIMARY KEY(AccountID),
    CONSTRAINT fk_AST_AccountStatusTypeID FOREIGN KEY(AccountStatusTypeID) REFERENCES AccountStatusType(AccountStatusTypeID),
    CONSTRAINT fk_SIR_InterestSavingRatesID FOREIGN KEY(InterestSavingRatesID) REFERENCES SavingsInterestRates(InterestSavingRatesID));
 GO
 --create a tables that makes a relation between OverDraftLog and Account.......ChildKey 1 and ChildKey7

CREATE TABLE OverDraftLog
(   AccountID INT NOT NULL IDENTITY(1,1),
    OverDraftDate DATETIME NOT NULL,
    OverDraftAmount Money NOT NULL,
    OverDraftTransactionXML XML NOT NULL,
    CONSTRAINT PK_ODL_AccountID PRIMARY KEY(AccountID),
    CONSTRAINT FK_A_ODL_AccountID FOREIGN KEY(AccountID) REFERENCES Account(AccountID));
 GO


 --create a tables that makes a relation between Login_Account, UserLogins and Account.......ChildKey3, PrimaryKey1, ChildKey7
CREATE TABLE Login_Account
(UserLoginID SMALLINT NOT NULL,
    AccountID INT NOT NULL,
    CONSTRAINT fk_UL_UserLogins FOREIGN KEY(UserLoginID) REFERENCES UserLogins(UserLoginID),
    CONSTRAINT fk_A_Account FOREIGN KEY(AccountID) REFERENCES Account(AccountID));
 GO



CREATE TABLE UserSecurityAnswer
(UserLoginID SMALLINT NOT NULL IDENTITY(1,1),
    UserSecurityAnswers VARCHAR(25) NOT NULL,
    UserSecurityQuestionID TINYINT NOT NULL,
    CONSTRAINT PKUSA_UserLoginID PRIMARY KEY(UserLoginID),
    CONSTRAINT FKUL_UserLoginID FOREIGN KEY(UserLoginID) REFERENCES UserLogins(UserLoginID),
    CONSTRAINT FKUSQ_UserSecurityQuestionID FOREIGN KEY(UserSecurityQuestionID) REFERENCES UserSecurityQuestions(UserSecurityQuestionID));
 GO

 --create a tables that makes a relation between Customer, TransactionLog.......ChildKey6, childKey8

CREATE TABLE Customer
(CustomerID INT NOT NULL IDENTITY(1,1),
    AccountID INT NOT NULL,
    CustomerAddress1 VARCHAR(30) NOT NULL,
    CustomerAddress2  VARCHAR(30),
    CustomerFirstName  VARCHAR(30) NOT NULL,
    CustomerMiddleInitial CHAR(1),
    CustomerLastName  VARCHAR(30) NOT NULL,
    City  VARCHAR(20) NOT NULL,
    State CHAR(2) NOT NULL,
    ZipCode CHAR(10) NOT NULL,
    EmailAddress CHAR(40) NOT NULL,
    HomePhone VARCHAR(10) NOT NULL,
    CellPhone VARCHAR(10) NOT NULL,
    WorkPhone VARCHAR(10) NOT NULL,
    SSN VARCHAR(9),
    UserLoginID SMALLINT NOT NULL,
    CONSTRAINT PKC_CustomerID PRIMARY KEY(CustomerID),
    --CONSTRAINT FKA_AccountID FOREIGN KEY(AccountID) REFERENCES Account(AccountID),
    CONSTRAINT FKUL_C_UserLoginID FOREIGN KEY(UserLoginID) REFERENCES UserLogins(UserLoginID));
 GO

```

```sql
--create a tables that makes a relation between Customer_Account, Customer and Account.......ChildKey5, ChildKey6 and ChildKey7
CREATE TABLE Customer_Account
(   AccountID INT NOT NULL ,
    CustomerID INT NOT NULL,
    CONSTRAINT PKACA_AccountID FOREIGN KEY(AccountID) REFERENCES Account(AccountID),
    CONSTRAINT PKCCA_CustomerID FOREIGN KEY(CustomerID) REFERENCES Customer(CustomerID));
GO

--create a tables that makes a relation between TransactionLog, TransactionType, Account, Customer,Employee and Userlogins
--.......ChildKey8, PrimaryKey6, ChildKey7, ChildKey6, PrimaryKey8 and PrimaryKey1

CREATE TABLE TransactionLog
(TransactionID INT NOT NULL IDENTITY(1,1),
    TransactionDate DATETIME NOT NULL,
    TransactionTypeID TINYINT NOT NULL,
    TransactionAmount Money NOT NULL,
    NewBalance Money NOT NULL,
    AccountID INT NOT NULL,
    CustomerID INT NOT NULL,
    EmployeeID INT NOT NULL,
    UserLoginID SMALLINT NOT NULL,
    CONSTRAINT PKTL_TransactionID PRIMARY KEY(TransactionID),
    CONSTRAINT FKTT_TL_TransactionTypeID FOREIGN KEY(TransactionTypeID) REFERENCES TransactionType(TransactionTypeID),
    CONSTRAINT FKA_TL_AccountID FOREIGN KEY(AccountID) REFERENCES Account(AccountID),
    CONSTRAINT FKC_TL_CustomerID FOREIGN KEY(CustomerID) REFERENCES Customer(CustomerID),
    CONSTRAINT FKE_TL_EmployeeID FOREIGN KEY(EmployeeID) REFERENCES Employee(EmployeeID),
    CONSTRAINT FKUL_TL_UserLoginID FOREIGN KEY(UserLoginID) REFERENCES UserLogins(UserLoginID));
GO
```


```sql
--create a tables that makes a relation between Customer_Account, Customer and Account.......ChildKey5, ChildKey6 and ChildKey7
CREATE TABLE Customer_Account
(   AccountID INT NOT NULL ,
    CustomerID INT NOT NULL,
    CONSTRAINT PKACA_AccountID FOREIGN KEY(AccountID) REFERENCES Account(AccountID),
    CONSTRAINT PKCCA_CustomerID FOREIGN KEY(CustomerID) REFERENCES Customer(CustomerID));
GO

--create a tables that makes a relation between TransactionLog, TransactionType, Account, Customer,Employee and Userlogins
--.......ChildKey8, PrimaryKey6, ChildKey7, ChildKey6, PrimaryKey8 and PrimaryKey1

CREATE TABLE TransactionLog
(TransactionID INT NOT NULL IDENTITY(1,1),
    TransactionDate DATETIME NOT NULL,
    TransactionTypeID TINYINT NOT NULL,
    TransactionAmount Money NOT NULL,
    NewBalance Money NOT NULL,
    AccountID INT NOT NULL,
    CustomerID INT NOT NULL,
    EmployeeID INT NOT NULL,
    UserLoginID SMALLINT NOT NULL,
    CONSTRAINT PKTL_TransactionID PRIMARY KEY(TransactionID),
    CONSTRAINT FKTT_TL_TransactionTypeID FOREIGN KEY(TransactionTypeID) REFERENCES TransactionType(TransactionTypeID),
    CONSTRAINT FKA_TL_AccountID FOREIGN KEY(AccountID) REFERENCES Account(AccountID),
    CONSTRAINT FKC_TL_CustomerID FOREIGN KEY(CustomerID) REFERENCES Customer(CustomerID),
    CONSTRAINT FKE_TL_EmployeeID FOREIGN KEY(EmployeeID) REFERENCES Employee(EmployeeID),
    CONSTRAINT FKUL_TL_UserLoginID FOREIGN KEY(UserLoginID) REFERENCES UserLogins(UserLoginID));
GO
```

The following table is the result of the code:

18. UserLogins Table:

Column Name	Data Type	Allow Nulls
UserLoginID	smallint	☐
UserLogin	varchar(50)	☐
UserPassword	varchar(20)	☐
		☐

19. UserSecurityQuestions Table:

Column Name	Data Type	Allow Nulls
UserSecurityQuestionID	tinyint	☐
UserSecurityQuestion	varchar(50)	☐
		☐

20. AccountType Table:

Column Name	Data Type	Allow Nulls
AccountTypeID	tinyint	☐
AccountTypeDescription	varchar(30)	☐
		☐

21. SavingInterestsRates Table:

Column Name	Data Type	Allow Nulls
InterestSavingRatesID	tinyint	☐
InterestRatesValue	numeric(9, 9)	☐
InterestRatesDescription	varchar(20)	☐
		☐

22. AccountStatuseType Table:

Column Name	Data Type	Allow Nulls
AccountStatusTypeID	tinyint	☐
AccountStatusTypeDescription	varchar(30)	☐
		☐

23. TransactionType Table:

Column Name	Data Type	Allow Nulls
TransactionTypeID	tinyint	☐
TransactionTypeName	char(10)	☐
TransactionTypeDescription	varchar(50)	☑
TransactionFeeAmount	smallmoney	☑
		☐

24. LoginErrorLog Table:

Column Name	Data Type	Allow Nulls
ErrorLogID	int	☐
ErrorTime	datetime	☐
FailedTransactionXML	xml	☑
		☐

25. Employee Table:

Column Name	Data Type	Allow Nulls
EmployeeID	int	☐
EmployeeFirstName	varchar(25)	☐
EmployeeMiddleInitial	char(1)	☑
EmployeeLastName	varchar(25)	☑
EmployeeisManager	bit	☑
		☐

26. FailedTransactionErrorType Table:

Column Name	Data Type	Allow Nulls
FailedTransactionID	int	☐
FailedTransactionErrorTypeID	tinyint	☐
FailedTransactionErrorTime	datetime	☑
FailedTransactionErrorXML	xml	☑
		☐

27. FailedTransactionLog Table:

Column Name	Data Type	Allow Nulls
FailedTransactionID	int	☐
FailedTransactionErrorTypeID	tinyint	☐
FailedTransactionErrorTime	datetime	☑
FailedTransactionErrorXML	xml	☑
		☐

28. Account Table:

Column Name	Data Type	Allow Nulls
AccountID	int	☐
CurrentBalance	int	☐
AccountTypeID	tinyint	☐
AccountStatusTypeID	tinyint	☐
InterestSavingRatesID	tinyint	☐
		☐

29. OverDraftLog Table:

Column Name	Data Type	Allow Nulls
AccountID	int	☐
OverDraftDate	datetime	☐
OverDraftAmount	money	☐
OverDraftTransactionXML	xml	☐
		☐

30. Login_Account Table:

Column Name	Data Type	Allow Nulls
UserLoginID	smallint	☐
AccountID	int	☐
		☐

31. UserSecurityAnswer Table:

Column Name	Data Type	Allow Nulls
UserLoginID	smallint	☐
UserSecurityAnswers	varchar(25)	☐
UserSecurityQuestionID	tinyint	☐
		☐

32. Customer:

Column Name	Data Type	Allow Nulls
CustomerID	int	☐
AccountID	int	☐
CustomerAddress1	varchar(30)	☐
CustomerAddress2	varchar(30)	☑
CustomerFirstName	varchar(30)	☐
CustomerMiddleInitial	char(1)	☑
CustomerLastName	varchar(30)	☐
City	varchar(20)	☐
State	char(2)	☐
ZipCode	char(10)	☐
EmailAddress	char(40)	☐
HomePhone	varchar(10)	☐
CellPhone	varchar(10)	☐
WorkPhone	varchar(10)	☐
SSN	varchar(9)	☑
UserLoginID	smallint	☐
		☐

33. CustomerAccount Table:

Column Name	Data Type	Allow Nulls
AccountID	int	☐
CustomerID	int	☐
		☐

34. TransactionLog Table

Column Name	Data Type	Allow Nulls
TransactionID	int	☐
TransactionDate	datetime	☐
TransactionTypeID	tinyint	☐
TransactionAmount	money	☐
NewBalance	money	☐
AccountID	int	☐
CustomerID	int	☐
EmployeeID	int	☐
UserLoginID	smallint	☐
		☐

4. Insert at least 5 rows in each table. [Basic]

Into ways for Questions 2 and 3 at least five rows have been inserted to the tables:

5 rows have been inserted to the tables in question2 illustrated below:

```sql
--...Insert at least 5 rows in each table....

--Inserting 5 records into table UserLogins

insert into UserLogins values(1,'UserLogin1', 'Password1');
insert into UserLogins values(2,'UserLogin2', 'Password2');
insert into UserLogins values(3,'UserLogin3', 'Password3');
insert into UserLogins values(4,'UserLogin4', 'Password4');
insert into UserLogins values(5,'UserLogin5', 'Password5');
select * from UserLogins;
GO

--Inserting 5 records into table UserSecurityQuestions

insert into UserSecurityQuestions values(1,'What is your favourite food?');
insert into UserSecurityQuestions values(2,'What is your favourite book?');
insert into UserSecurityQuestions values(3,'What is your name of first friend?');
insert into UserSecurityQuestions values(4,'What is your Granmother name?');
insert into UserSecurityQuestions values(5,'What is your Grandfather name?');
select * from UserSecurityQuestions;
```

```sql
--Inserting 5 records into table AccountType

Insert into AccountTypes(AccountTypeID,AccountTypeDescription) values(1,'SavingAccount1');
Insert into AccountTypes(AccountTypeID,AccountTypeDescription) values(2,'CheckingAccount1');
Insert into AccountTypes(AccountTypeID,AccountTypeDescription) values(3,'SavingAccount2');
Insert into AccountTypes(AccountTypeID,AccountTypeDescription) values(4,'CheckingAccount2');
Insert into AccountTypes(AccountTypeID,AccountTypeDescription) values(5,'SavingAccount3');
select * from AccountTypes;
GO

--Inserting 5 records into table SavingsInterestRates

insert into SavingInterestRates(InterestSavingRateID,InterestRateValue,InterestRateDescription) values(1,0.05, 'Low');
insert into SavingInterestRates(InterestSavingRateID,InterestRateValue,InterestRateDescription) values(2,0.2, 'Medium');
insert into SavingInterestRates(InterestSavingRateID,InterestRateValue,InterestRateDescription) values(3,0.3, 'High');
insert into SavingInterestRates(InterestSavingRateID,InterestRateValue,InterestRateDescription) values(4,0.4, 'Very high');
insert into SavingInterestRates(InterestSavingRateID,InterestRateValue,InterestRateDescription) values(5,0.6, 'Super high');
select * from SavingInterestRates;
GO
```

```sql
--Inserting 5 records into table AccountStatusType

insert into AccountStatuseType values(1,'Closed');
insert into AccountStatuseType values(2,'Active');
insert into AccountStatuseType values(3,'Dormant');
insert into AccountStatuseType values(4,'Passive');
insert into AccountStatuseType values(5,'Active');
select * from AccountStatuseType;
GO

--Inserting 5 records into table FailedTransactionErrorType

insert into FailedTransactionErrorType values(1,'Withdraw limit');
insert into FailedTransactionErrorType values(2,'Invalid Password');
insert into FailedTransactionErrorType values(3,'Time-out');
insert into FailedTransactionErrorType values(4,'OVERFLOW');
insert into FailedTransactionErrorType values(5,'Double detect error');
select * from FailedTransactionErrorType;
GO
```

```sql
--Inserting 5 records into table Employee

insert into Employee values(1,'Employee1', 'A', 'LN1', '0');
insert into Employee values(2,'Employee2', 'B', 'LN2', '1');
insert into Employee values(3,'Employee3', 'C', 'LN3', '0');
insert into Employee values(4,'Employee4', 'D', 'LN4', '1');
insert into Employee values(5,'Employee5', 'E', 'LN5', '1');
select * from Employee;
GO

--Inserting 5 records into table TransactionType

insert into TransactionType values(1,'Balance', 'See money', '0');
insert into TransactionType values(2,'Transfer', 'Send money', '450');
insert into TransactionType values(3,'Receive', 'Get money', '300');
insert into TransactionType values(4,'Paid', 'Paid to powerstream', '200');
insert into TransactionType values(5,'Statement', 'Checked monthly transaction', '0');
select * from TransactionType;
GO
```

```sql
--Inserting 5 records into table UserSecurityAnswers

insert into UserSecurityAnswer values(1,'kabab', 1);
insert into UserSecurityAnswer values(2,'Masnavi', 2);
insert into UserSecurityAnswer values(3,'Mahboobe', 3);
insert into UserSecurityAnswer values(4,'Maryam', 4);
insert into UserSecurityAnswer values(5,'AmirAli', 5);
select * from UserSecurityAnswer
GO

--Inserting 5 records into table Account

insert into Account values(1,22000, 1, 1, 1);
insert into Account values(2,25000, 2, 2, 2);
insert into Account values(3,17000, 1, 1, 1);
insert into Account values(4,50000, 2, 2, 2);
insert into Account values(5,35000, 2, 2, 2);
select * from Account;
GO
```

```sql
--Inserting 5 records into table LoginAccount

insert into Login_Account values(1, 1);
insert into Login_Account values(2, 2);
insert into Login_Account values(3, 3);
insert into Login_Account values(4, 4);
insert into Login_Account values(5, 5);
select * from Login_Account;
GO

--Inserting 5 records into table Customer

insert into Customer values(1, 'Address1', 'Address2', 'Customer1', 'A', 'CLastname1', 'Ottawa', 'ON', '3A5z9z', 'user1@user.com', '1416555111', '453554464', '3462325', '1', 1);
insert into Customer values(2, 'Address1', 'Address2', 'Customer2', 'B', 'CLastname2', 'Toronto', 'ON', 'fe3453', 'user2@user.com', '1647892000', '567435345', '6332423', '2', 2);
insert into Customer values(3, 'Address1', 'Address2', 'Customer3', 'C', 'CLastname3', 'Montreal', 'QC', 'fdf45', 'user3@user.com', '1438222652', '681316226', '9202521', '3', 3);
insert into Customer values(4, 'Address1', 'Address2', 'Customer4', 'D', 'CLastname4', 'Burlington', 'ON', '23ffbfs', 'user4@user.com', '1647893300', '795197107', '8674252', '4', 4)
insert into Customer values(5, 'Address1', 'Address2', 'Customer5', 'E', 'CLastname5', 'Waterloo', 'ON', 'hg4536', 'user5@user.com', '1416893666', '909077988', '9209371', '5', 5);
select * from Customer;
GO
```

```sql
--Inserting 5 records into table CustomerAccount

insert into Customer_Account values(1, 1);
insert into Customer_Account values(2, 2);
insert into Customer_Account values(3, 3);
insert into Customer_Account values(4, 4);
insert into Customer_Account values(5, 5);
select * from  Customer_Account;
GO

--Inserting 5 records into table TransactionLog

insert into TransactionLog values(1,'2020/11/4 07:30:56', 1,1000, 8000, 1, 1, 1, 1);
insert into TransactionLog values(2,'2020/11/9 12:34:57', 2,1000, 7900, 2, 2, 2, 2);
insert into TransactionLog values(3,'2020/12/5 02:14:00', 3,2000, 7700, 3, 3, 3, 3);
insert into TransactionLog values(4,'2020/12/5 05:56:59', 4,500, 7650, 4, 4, 4, 4);
insert into TransactionLog values(5,'2020/12/12 08:34:15',5,1000, 10500, 5, 5, 5, 5);
select * from TransactionLog;
GO
```

```sql
--Inserting 5 records into table OverDraftLog
--select * from OverDraftLog;
insert into OverDraftLog values(1,'2020/11/4 07:30:56', 0, 'Clear');
insert into OverDraftLog values(2,'2020/11/9 12:34:57', 5, 'Pending');
insert into OverDraftLog values(3,'2020/12/5 02:14:00', 10, 'Clear');
insert into OverDraftLog values(4,'2020/12/5 05:56:59', 15, 'Pending');
insert into OverDraftLog values(5,'2020/12/12 08:34:15', 20, 'Clear');
select * from OverDraftLog;
Go
/*************************************************************************/
```

5 rows have been inserted to the tables in question3 illustrated below:

```sql

--...Insert at least 5 rows in each table....

--Inserting 5 records into table UserLogins

insert into UserLogins values('UserLogin1', 'Password1');
insert into UserLogins values('UserLogin2', 'Password2');
insert into UserLogins values('UserLogin3', 'Password3');
insert into UserLogins values('UserLogin4', 'Password4');
insert into UserLogins values('UserLogin5', 'Password5');
select * from UserLogins;
GO

--Inserting 5 records into table UserSecurityQuestions
insert into UserSecurityQuestions values('What is your favourite food?');
insert into UserSecurityQuestions values('What is your favourite book?');
insert into UserSecurityQuestions values('What is your name of first friend?');
insert into UserSecurityQuestions values('What is your Granmother name?');
insert into UserSecurityQuestions values('What is your Grandfather name?');
select * from UserSecurityQuestions;
GO

--Inserting 5 records into table AccountType
Insert into AccountType values('SavingAccount1');
Insert into AccountType values('CheckingAccount1');
Insert into AccountType values('SavingAccount2');
Insert into AccountType values('CheckingAccount2');
Insert into AccountType values('SavingAccount3');
select * from AccountType;
GO

--Inserting 5 records into table SavingsInterestRates
insert into SavingsInterestRates values(0.05, 'Low');
insert into SavingsInterestRates values(0.2, 'Medium');
insert into SavingsInterestRates values(0.3, 'High');
insert into SavingsInterestRates values(0.4, 'Very high');
insert into SavingsInterestRates values(0.6, 'Super high');
select * from SavingsInterestRates;
GO
```

```sql
--Inserting 5 records into table AccountStatusType
insert into AccountStatusType values('Closed');
insert into AccountStatusType values('Active');
insert into AccountStatusType values('Dormant');
insert into AccountStatusType values('Passive');
insert into AccountStatusType values('Active');
select * from AccountStatusType;
GO

--Inserting 5 records into table FailedTransactionErrorType
insert into FailedTransactionErrorType values('Withdraw limit');
insert into FailedTransactionErrorType values('Invalid Password');
insert into FailedTransactionErrorType values('Time-out');
insert into FailedTransactionErrorType values('OVERFLOW');
insert into FailedTransactionErrorType values('Double detect error');
select * from FailedTransactionErrorType;
GO

--Inserting 5 records into table LoginErrorLog
insert into LoginErrorLog values(TRY_CAST('2020/12/4 07:30:56' AS DATETIME), 'Bad Connection');
insert into LoginErrorLog values(TRY_CAST('2020/12/9 12:34:57' AS DATETIME), 'Invalid User');
insert into LoginErrorLog values(TRY_CAST('2020/12/15 02:14:00' AS DATETIME), 'Wrong Password');
insert into LoginErrorLog values(TRY_CAST('2020/12/20 05:56:59' AS DATETIME), 'Server issue');
insert into LoginErrorLog values(TRY_CAST('2020/12/21 08:34:15' AS DATETIME), 'Datacenter down');
select * from LoginErrorLog;
GO

--Inserting 5 records into table Employee
insert into Employee values('Employee1', 'A', 'LN1', '0');
insert into Employee values('Employee2', 'B', 'LN2', '1');
insert into Employee values('Employee3', 'C', 'LN3', '0');
insert into Employee values('Employee4', 'D', 'LN4', '1');
insert into Employee values('Employee5', 'E', 'LN5', '1');
select * from Employee;
GO
```

```sql
--Inserting 5 records into table TransactionType
insert into TransactionType values('Balance', 'See money', '0');
insert into TransactionType values('Transfer', 'Send money', '450');
insert into TransactionType values('Receive', 'Get money', '300');
insert into TransactionType values('Paid', 'Paid to powerstream', '200');
insert into TransactionType values('Statement', 'Checked monthly transaction', '0');
select * from TransactionType;
GO

--Inserting 5 records into table FailedTransactionLog
insert into FailedTransactionLog values(1, TRY_CAST('2020/12/4 07:30:56' AS DATETIME), 'First');
insert into FailedTransactionLog values(2, TRY_CAST('2020/12/9 12:34:57' AS DATETIME), 'Second');
insert into FailedTransactionLog values(3, TRY_CAST('2020/12/15 02:14:00' AS DATETIME), 'Third');
insert into FailedTransactionLog values(4, TRY_CAST('2020/12/20 05:56:59' AS DATETIME), 'Fourth');
insert into FailedTransactionLog values(5, TRY_CAST('2020/12/21 08:34:15' AS DATETIME), 'Fifth');
select * from FailedTransactionLog;
GO

--Inserting 5 records into table UserSecurityAnswers
insert into UserSecurityAnswer values('kabab', 1);
insert into UserSecurityAnswer values('Masnavi', 2);
insert into UserSecurityAnswer values('Mahboobe', 3);
insert into UserSecurityAnswer values('Maryam', 4);
insert into UserSecurityAnswer values('AmirAli', 5);
select * from UserSecurityAnswer;
GO

--Inserting 5 records into table Account
insert into Account values(22000, 1, 1, 1);
insert into Account values(25000, 2, 2, 2);
insert into Account values(17000, 1, 1, 1);
insert into Account values(50000, 2, 2, 2);
insert into Account values(35000, 2, 2, 2);
select * from Account;
GO
```

```
insert into TransactionLog values('2020/12/12 08:34:15',5,1000, 10500, 5, 5, 5, 5);
select * from TransactionLog;
GO

--Inserting 5 records into table OverDraftLog
insert into OverDraftLog values('2020/11/4 07:30:56', 0, 'Clear');
insert into OverDraftLog values('2020/11/9 12:34:57', 5, 'Pending');
insert into OverDraftLog values('2020/12/5 02:14:00', 10, 'Clear');
insert into OverDraftLog values('2020/12/5 05:56:59', 15, 'Pending');
insert into OverDraftLog values('2020/12/12 08:34:15', 20, 'Clear');
select * from OverDraftLog;
Go
```

```

--Inserting 5 records into table LoginAccount
insert into Login_Account values(1, 1);
insert into Login_Account values(2, 2);
insert into Login_Account values(3, 3);
insert into Login_Account values(4, 4);
insert into Login_Account values(5, 5);
select * from Login_Account;
GO

--Inserting 5 records into table Customer
insert into Customer values(1, 'Address1', 'Address2', 'Customer1', 'A', 'CLastname1', 'Ottawa', 'ON', '3A5z9z', 'user1@user.com', '1416555111', '453554464', '3462325', 'A12345', 1)
insert into Customer values(2, 'Address1', 'Address2', 'Customer2', 'B', 'CLastname2', 'Toronto', 'ON', 'fe3453', 'user2@user.com', '1647892000', '567435345', '6332423', 'D34353', 2
insert into Customer values(3, 'Address1', 'Address2', 'Customer3', 'C', 'CLastname3', 'Montreal', 'QC', 'fdf45', 'user3@user.com', '1438222652', '681316226', '9202521', 'J56361', 3
insert into Customer values(4, 'Address1', 'Address2', 'Customer4', 'D', 'CLastname4', 'Burlington', 'ON', '23ffbfs', 'user4@user.com', '1647893300', '795197107', '8674252', 'I78369
insert into Customer values(5, 'Address1', 'Address2', 'Customer5', 'E', 'CLastname5', 'Waterloo', 'ON', 'hg4536', 'user5@user.com', '1416893666', '909077988', '9209371', 'K10377',
select * from Customer;
GO

--Inserting 5 records into table CustomerAccount
insert into Customer_Account values(1, 1);
insert into Customer_Account values(2, 2);
insert into Customer_Account values(3, 3);
insert into Customer_Account values(4, 4);
insert into Customer_Account values(5, 5);
select * from  Customer_Account;
GO

--Inserting 5 records into table TransactionLog
insert into TransactionLog values('2020/11/4 07:30:56', 1,1000, 8000, 1, 1, 1, 1);
insert into TransactionLog values('2020/11/9 12:34:57', 2,1000, 7900, 2, 2, 2, 2);
insert into TransactionLog values('2020/12/5 02:14:00', 3,2000, 7700, 3, 3, 3, 3);
insert into TransactionLog values('2020/12/5 05:56:59', 4,500, 7650, 4, 4, 4, 4);
insert into TransactionLog values('2020/12/12 08:34:15',5,1000, 10500, 5, 5, 5, 5);
select * from TransactionLog;
GO
```

you can insert several records up to 1000 in one insert command.

1. UserLogins Table with 5 insert rows:

	UserLoginID	UserLogin	UserPassword
1	1	UserLogin1	Password1
2	2	UserLogin2	Password2
3	3	UserLogin3	Password3
4	4	UserLogin4	Password4
5	5	UserLogin5	Password5

2. UserSecuirtyQuestions table with 5 insert rowes:

Results | Messages

	UserSecurityQuestionID	UserSecurityQuestion
1	1	What is your favourite food?
2	2	What is your favourite book?
3	3	What is your name of first friend?
4	4	What is your Granmother name?
5	5	What is your Grandfather name?

3. AccountType table with 5 insert rows:

Results | Messages

	AccountID	CurrentBalance	AccountTypeID	AccountStatusTypeID	InterestSavingRatesID
1	1	22000	1	1	1
2	2	25000	2	2	2
3	3	17000	1	1	1
4	4	50000	2	2	2
5	5	35000	2	2	2

4. SavingInterestRates table with5 insert rows

	InterestSavingRatesID	InterestRatesValue	InterestRatesDescription
1	1	0.050000000	Low
2	2	0.200000000	Medium
3	3	0.300000000	High
4	4	0.400000000	Very high
5	5	0.600000000	Super high
6	6	0.050000000	Low
7	7	0.200000000	Medium
8	8	0.300000000	High
9	9	0.400000000	Very high
10	10	0.600000000	Super high

5. AccountStatuseType table with5 insert rows:

Results | Messages

	AccountTypeID	AccountTypeDescription
1	1	SavingAccount1
2	2	CheckingAccount1
3	3	SavingAccount2
4	4	CheckingAccount2
5	5	SavingAccount3

6. TransactionType table with 5 insert rows

Results | Messages

	TransactionTypeID	TransactionTypeName	TransactionTypeDescription	TransactionFeeAmount
1	1	Balance	See money	0.00
2	2	Transfer	Send money	450.00
3	3	Receive	Get money	300.00
4	4	Paid	Paid to John	45000.00
5	5	Statement	Checked monthly transaction	0.00

7. LoginErrorLog table with 5 insert rows:

	ErrorLogID	ErrorTime	FailedTransactionXML
1	1	2020-12-04 07:30:56.000	Bad Connection
2	2	2020-12-09 12:34:57.000	Invalid User
3	3	2020-12-15 02:14:00.000	Wrong Password
4	4	2020-12-20 05:56:59.000	Server issue
5	5	2020-12-21 08:34:15.000	Datacenter down

8. Employee table with 5 insert rows:

Results | Messages

	EmployeeID	EmployeeFirstName	EmployeeMiddleInitial	EmployeeLastName	EmployeeisManager
1	1	Employee1	A	LN1	0
2	2	Employee2	B	LN2	1
3	3	Employee3	C	LN3	0
4	4	Employee4	D	LN4	1
5	5	Employee5	E	LN5	1

9. FailedTransactionErrorType table with 5 insert rows:

Results | Messages

	FailedTransactionErrorTypeID	FailedTransactionErrorTypeDescription
1	1	Withdraw limit
2	2	Invalid Password
3	3	Time-out
4	4	OVERFLOW
5	5	Double detect error

10. FailedTransactionLog table with 5 insert rows:

Results | Messages

	AccountID	CurrentBalance	AccountTypeID	AccountStatusTypeID	InterestSavingRatesID
1	1	22000	1	1	1
2	2	25000	2	2	2
3	3	17000	1	1	1
4	4	50000	2	2	2
5	5	35000	2	2	2

11. Account table with 5 insert rows:

Results | Messages

	AccountID	OverDraftDate	OverDraftAmount	OverDraftTransactionXML
1	1	2020-11-04 07:30:56.000	0.00	Clear
2	2	2020-11-09 12:34:57.000	5.00	Pending
3	3	2020-12-05 02:14:00.000	10.00	Clear
4	4	2020-12-05 05:56:59.000	15.00	Pending
5	5	2020-12-12 08:34:15.000	20.00	Clear

12. OverDraftLog table with 5 insert rows:

Results | Messages

	AccountID	OverDraftDate	OverDraftAmount	OverDraftTransactionXML
1	1	2020-11-04 07:30:56.000	0.00	Clear
2	2	2020-11-09 12:34:57.000	5.00	Pending
3	3	2020-12-05 02:14:00.000	10.00	Clear
4	4	2020-12-05 05:56:59.000	15.00	Pending
5	5	2020-12-12 08:34:15.000	20.00	Clear

13. LoginAccount table with 5 insert rows:

Results | Messages

	UserLoginID	AccountID
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5

14. UserSecurityQuestion table with 5 insert rows:

Results | Messages

	UserLoginID	UserSecurityAnswers	UserSecurityQuestionID
1	1	kabab	1
2	2	Masnavi	2
3	3	Mahboobe	3
4	4	Maryam	4
5	5	AmirAli	5

15. Customer table with 5 insert rows:

Results | Messages

	CustomerID	AccountID	CustomerAddress1	CustomerAddress2	CustomerFirstName	CustomerMiddleInitial	CustomerLastName	City	State	ZipCode	EmailAddress	HomePhone	CellPhone	WorkPhone	SSN	UserLoginID
1	1	1	Address1	Address2	Customer1	A	CLastname1	Ottawa	ON	3A5z9z	user1@user.com	1416555111	453554464	3462325	A12345	1
2	2	2	Address1	Address2	Customer2	B	CLastname2	Toronto	ON	fe3453	user2@user.com	1647892000	567435345	6332423	D34353	2
3	3	3	Address1	Address2	Customer3	C	CLastname3	Montreal	QC	fdf45	user3@user.com	1438222652	681316226	9202521	J56361	3
4	4	4	Address1	Address2	Customer4	D	CLastname4	Burlington	ON	23ffbfs	user4@user.com	1647893300	795197107	8674252	I78369	4
5	5	5	Address1	Address2	Customer5	E	CLastname5	Waterloo	ON	hg4536	user5@user.com	1416893666	909077988	9209371	K10377	5

16. CustomerAccount table with 5 insert rows:

Results | Messages

	AccountID	CustomerID
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5

17. TransactionLog table with 5 insert rows:

Results | Messages

	TransactionID	TransactionDate	TransactionTypeID	TransactionAmount	NewBalance	AccountID	CustomerID	EmployeeID	UserLoginID
1	1	2020-11-04 07:30:56.000	1	1000.00	8000.00	1	1	1	1
2	2	2020-11-09 12:34:57.000	2	1000.00	7900.00	2	2	2	2
3	3	2020-12-05 02:14:00.000	3	2000.00	7700.00	3	3	3	3
4	4	2020-12-05 05:56:59.000	4	500.00	7650.00	4	4	4	4
5	5	2020-12-12 08:34:15.000	5	1000.00	10500.00	5	5	5	5

Phase Two

To perform projects in the second phase, it is necessary to use Procedure and View. Accordingly, their definitions are given below:

View:

In **SQL**, a **view** is a virtual table based on the result-set of an **SQL** statement. ... The fields in a **view** are fields from one or more real tables in the database. You can add **SQL** functions, WHERE, and JOIN statements to a **view** and present the data as if the data were coming from one single table.

We can use **View Definition** permission in **SQL** Server to allow users to **view** the object **definitions**. We can either provide this access to a public role or an individual user. If we want to provide **view** object **definition** rights to all users with public role, execute the following query.

In a database, a **view** is the result set of a *stored* query on the data, which the database users can query just as they would in a persistent database collection object. This pre-established query command is kept in the database dictionary. Unlike ordinary *base tables* in a relational database, a view does not form part of the physical schema: as a result set, it is a virtual table computed or collated dynamically from data in the database when access to that view is requested. Changes applied to the data in a relevant *underlying table* are reflected in the data shown in subsequent invocations of the view. In some NoSQL databases, views are the only way to query data.

Views can provide advantages over tables:

- Views can represent a subset of the data contained in a table. Consequently, a view can limit the degree of exposure of the underlying tables to the outer world: a given user may have permission to query the view, while denied access to the rest of the base table.
- Views can join and simplify multiple tables into a single virtual table.
- Views can act as aggregated tables, where the database engine aggregates data (sum, average, etc.) and presents the calculated results as part of the data.

- Views can hide the complexity of data. For example, a view could appear as Sales2000 or Sales2001, transparently partitioning the actual underlying table.
- Views take very little space to store; the database contains only the definition of a view, not a copy of all the data that it presents.
- Depending on the SQL engine used, views can provide extra security.

Just as a function (in programming) can provide abstraction, so can a database view. In another parallel with functions, database users can manipulate nested views, thus one view can aggregate data from other views. Without the use of views, the normalization of databases above the second normal form would become much more difficult. Views can make it easier to create lossless join decomposition.

Just as rows in a base table lack any defined ordering, rows available through a view do not appear with any default sorting. A view is a relational table, and the relational model defines a table as a set of rows. Since sets are not ordered — by definition — neither are the rows of a view. Therefore, an ORDER BY clause in the view definition is meaningless; the SQL standard does not allow an ORDER BY clause in the subquery of a CREATE VIEW command, just as it is refused in a CREATE TABLE statement. However, sorted data can be obtained from a view, in the same way as any other table — as part of a query statement on that view. Nevertheless, some DBMS (such as Oracle Database) do not abide by this SQL standard restriction.

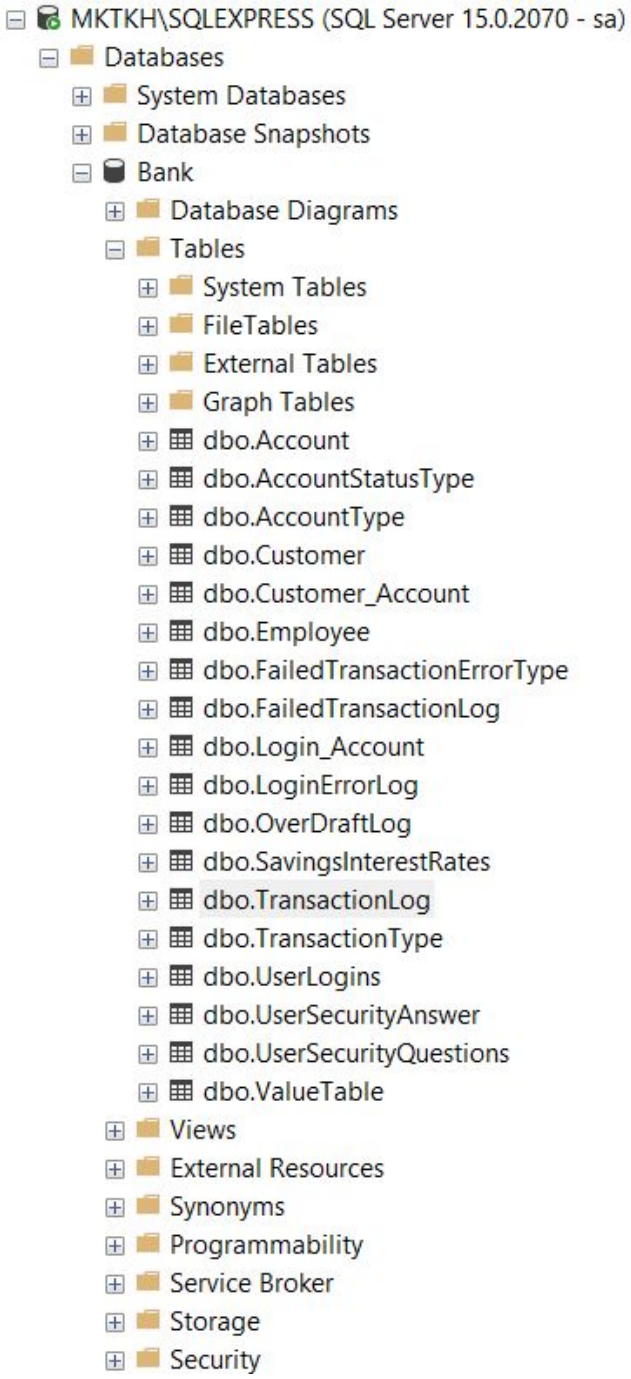
Procedure:

A stored **procedure** is a prepared **SQL** code that you can save, so the code can be reused over and over again. So if you have an **SQL** query that you write over and over again, save it as a stored **procedure**, and then just call it to execute it.

The **CREATE PROCEDURE SQL** command is used to **create a procedure**, followed by an SP name and its parameters. The BEGIN and END areas are used to define the query for the operation. This is where you will **write** a select, update, insert, or delete query.

BANK DATABASE Tables:

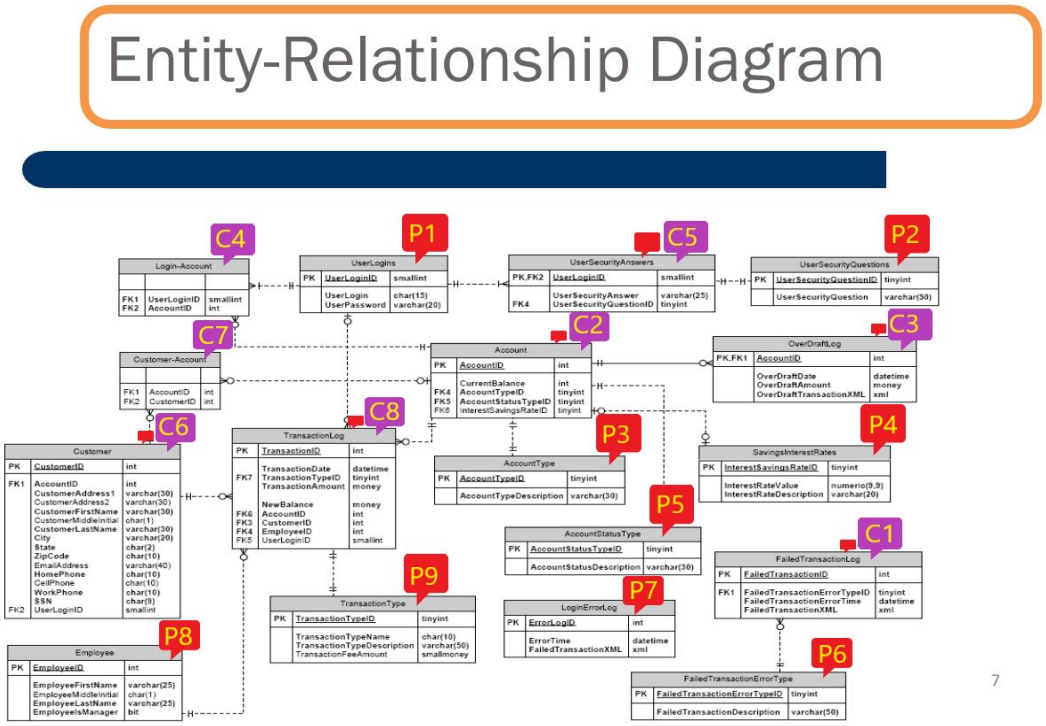
After completing the first phase of the project, the following data Bank has been created.



Bank Database Design

There are only two types of accounts at this time: Checking and Savings accounts. The provided column list should be separated into appropriate entities (tables) with relationships

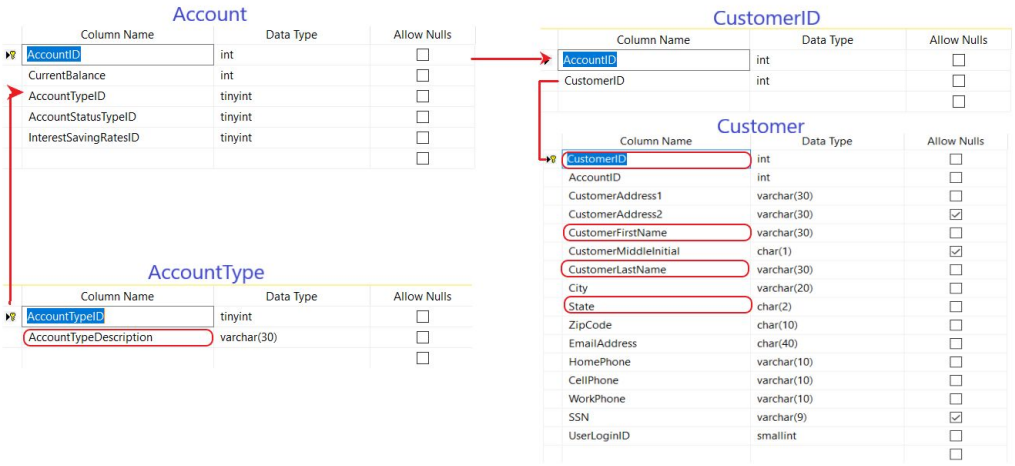
between these entities defined. The most efficient choices as far as your primary key constraints and foreign key constraints, and picked the appropriate data types for each of the columns.



1. Create a view to getting all customers with checking accounts from ON province. [Moderate]

- Created a View by the name ON_checkingA
- By selecting (select) records from Customer table who have an AccountType and show the AccountTypeDescription
- A cross reference inner join (Join) is built between (Customer, CustomerID, Account, and AccountType)

Tables required to create the view:




```
--Question1: Create a view to getting all customers with checking accounts from ON province. [Moderate]
Drop view ON_ChekingA

GO

CREATE VIEW ON_ChekingA
as
select c.CustomerID, c.CustomerFirstName+' '+c.CustomerLastName [Custome Name], c.State , at.AccountTypeDescription
from Customer c
  join Customer_Account ca
   on C.CustomerID=ca.CustomerID
   join Account a
   on ca.AccountID=a.AccountID
   join AccountType at
   on a.AccountTypeID=at.AccountTypeID
where at.AccountTypeDescription='Checkings' and c.State='ON';
Select * from ON_ChekingA
```

```
Messages
Commands completed successfully.

Completion time: 2020-12-26T14:04:09.6265972-05:00
```

```
CREATE VIEW ON_ChekingA
as
select c.CustomerID, c.CustomerFirstName+' '+c.CustomerLastName [Custome Name], c.State , at.AccountTypeDescription
from Customer c
  join Customer_Account ca
   on C.CustomerID=ca.CustomerID
   join Account a
   on ca.AccountID=a.AccountID
   join AccountType at
   on a.AccountTypeID=at.AccountTypeID
where at.AccountTypeDescription='Checkings' and c.State='ON';
Select * from ON_ChekingA
```

Results | Messages

CustomerID	Custome Name	State	AccountTypeDescription

2. Create a view to getting all customers with a total account balance (including the interest rate) greater than 5000. [Advanced]

Created a view by the name VW_Customer_ACBIR. By selecting (SELECT) records from the customer table which have an account with balances>5000 and showing the sum of their account balance and interest amount.

A cross reference inner join (Join) in built between (Customer, Account, and Saving InterestRate)

To identify account balance and interest rates.

Tables required to create the view:

SavingInterestRate

Column Name	Data Type	Allow Nulls
InterestSavingRatesID	tinyint	☐
InterestRatesValue	numeric(9, 9)	☐
InterestRatesDescription	varchar(20)	☐
		☐

Account

Column Name	Data Type	Allow Nulls
AccountID	int	☐
CurrentBalance	int	☐
AccountTypeID	tinyint	☐
AccountStatusTypeID	tinyint	☐
InterestSavingRatesID	tinyint	☐
		☐

Customer

Column Name	Data Type	Allow Nulls
CustomerID	int	☐
AccountID	int	☐
CustomerAddress1	varchar(30)	☐
CustomerAddress2	varchar(30)	☑
CustomerFirstName	varchar(30)	☐
CustomerMiddleInitial	char(1)	☑
CustomerLastName	varchar(30)	☐
City	varchar(20)	☐
State	char(2)	☐
ZipCode	char(10)	☐
EmailAddress	char(40)	☐
HomePhone	varchar(10)	☐
CellPhone	varchar(10)	☐
WorkPhone	varchar(10)	☐
SSN	varchar(9)	☑
UserLoginID	smallint	☐
		☐

```sql
/*****************************************************************************************/
/*.........................................................................................*/
--Question2. Create a view to get all customers with total account balance (including interest rate) greater than 5000. [Advanced]

DROP VIEW VW_Customer_ACBIR;
GO

CREATE VIEW VW_Customer_ACBIR AS
SELECT c.CustomerFirstName, SUM(a.CurrentBalance) AS Ac_Balance, SUM(a.CurrentBalance + (a.CurrentBalance * s.InterestSavingRatesID)/100) AS Total_Ac_Balance
FROM Customer c
JOIN Account a
ON c.AccountID = a.AccountId
JOIN SavingsInterestRates s
ON a.InterestSavingRatesID = s.InterestSavingRatesID
GROUP BY c.CustomerFirstName
HAVING SUM(a.CurrentBalance + (a.CurrentBalance * s.InterestRatesValue)/100) > 5000;
GO
```

100 %

Messages

```
Commands completed successfully.

Completion time: 2020-12-26T15:25:53.7795572-05:00
```

```sql
--Question2. Create a view to get all customers with total account balance (including interest rate) greater than 5000. [Advanced]

DROP VIEW VW_Customer_ACBIR;
GO

CREATE VIEW VW_Customer_ACBIR AS
SELECT c.CustomerFirstName, SUM(a.CurrentBalance) AS Ac_Balance, SUM(a.CurrentBalance + (a.CurrentBalance * s.InterestSavingRatesID)/100) AS Total_Ac_Balance
FROM Customer c
JOIN Account a
ON c.AccountID = a.AccountId
JOIN SavingsInterestRates s
ON a.InterestSavingRatesID = s.InterestSavingRatesID
GROUP BY c.CustomerFirstName
HAVING SUM(a.CurrentBalance + (a.CurrentBalance * s.InterestRatesValue)/100) > 5000;
SELECT * FROM VW_Customer_ACBIR
GO
```

100 %

Results | Messages

	CustomerFirstName	Ac_Balance	Total_Ac_Balance
1	Customer1	22000	22220
2	Customer2	25000	25500
3	Customer3	17000	17170
4	Customer4	45000	45900
5	Customer5	35000	35700

3. Create a view to getting counts of checking and savings accounts by customers. [Moderate]

Since we use Aggregate to find the account we need to use group by in SQL Query Joined Tables Customer, Account, and AccountType.

Tables required to create the view:

Account

Column Name	Data Type	Allow Nulls
AccountID	int	☐
CurrentBalance	int	☐
AccountTypeID	tinyint	☐
AccountStatusTypeID	tinyint	☐
InterestSavingRatesID	tinyint	☐
		☐

AccountType

Column Name	Data Type	Allow Nulls
AccountTypeID	tinyint	☐
AccountTypeDescription	varchar(30)	☐
		☐

Customer

Column Name	Data Type	Allow Nulls
CustomerID	int	☐
AccountID	int	☐
CustomerAddress1	varchar(30)	☐
CustomerAddress2	varchar(30)	☑
CustomerFirstName	varchar(30)	☐
CustomerMiddleInitial	char(1)	☑
CustomerLastName	varchar(30)	☐
City	varchar(20)	☐
State	char(2)	☐
ZipCode	char(10)	☐
EmailAddress	char(40)	☐
HomePhone	varchar(10)	☐
CellPhone	varchar(10)	☐
WorkPhone	varchar(10)	☐
SSN	varchar(9)	☑
UserLoginID	smallint	☐
		☐

```sql
--Q3. Create a view to get counts of checking and savings accounts by customer.
USE BANK
GO

--DROP VIEW VW_Customer_ACC;
--GO

CREATE VIEW VW_Customer_ACC
AS
SELECT c.CustomerFirstName, at.AccountTypeDescription, COUNT(*) AS Total_AC_Types
FROM Customer c
JOIN Account a
ON c.AccountID = a.AccountId
JOIN AccountType at
ON a.AccountTypeID = at.AccountTypeID
GROUP BY c.CustomerFirstName, at.AccountTypeDescription;
Select * from VW_Customer_ACC;
GO

/*****************************************************************************/
```

91 %

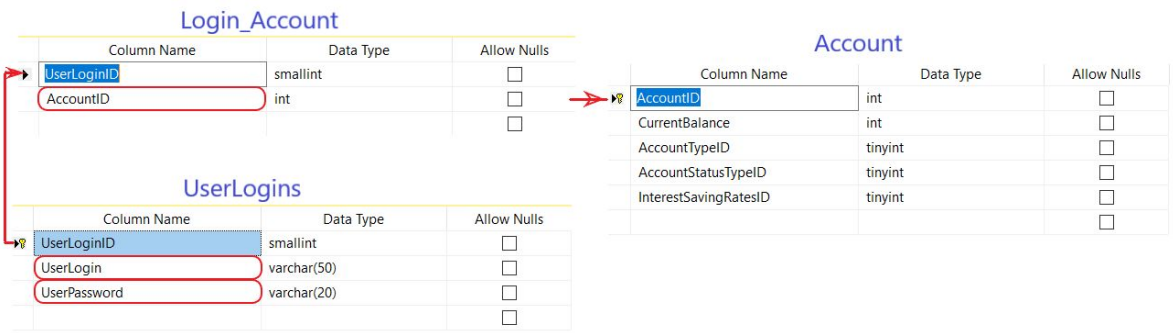
Results | Messages

	CustomerFirstName	AccountTypeDescription	Total_AC_Types
1	Customer2	CheckingAccount1	1
2	Customer4	CheckingAccount1	1
3	Customer5	CheckingAccount1	1
4	Customer1	SavingAccount1	1
5	Customer3	SavingAccount1	1

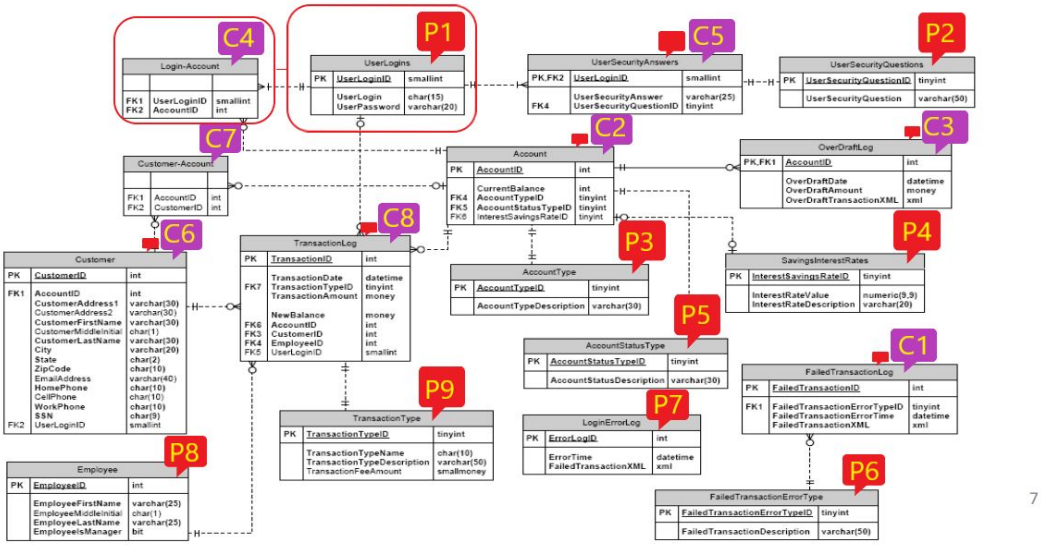
4. Create a view to get any particular user's login and password using AccountId. [Moderate]

Joined Tables UserLogins, Login_Account, and Account.

Tables required to create the view:



Entity-Relationship Diagram




```sql
/*****************************************************************************/
--Q4.Create a view to get any particular user's login and password using AccountId. [Moderate]

--DROP VIEW VW_Account_UL;
--GO

CREATE VIEW Account_UL
AS
SELECT distinct ul.UserLogin, ul.UserPassword
FROM UserLogins ul
JOIN Login_Account la
ON ul.UserLoginID = la.UserLoginID
--JOIN Account a
--ON la.AccountID = a.AccountID;
WHERE la.AccountID = '2'
select * From   Account_UL
GO
```

91 %

Results | Messages

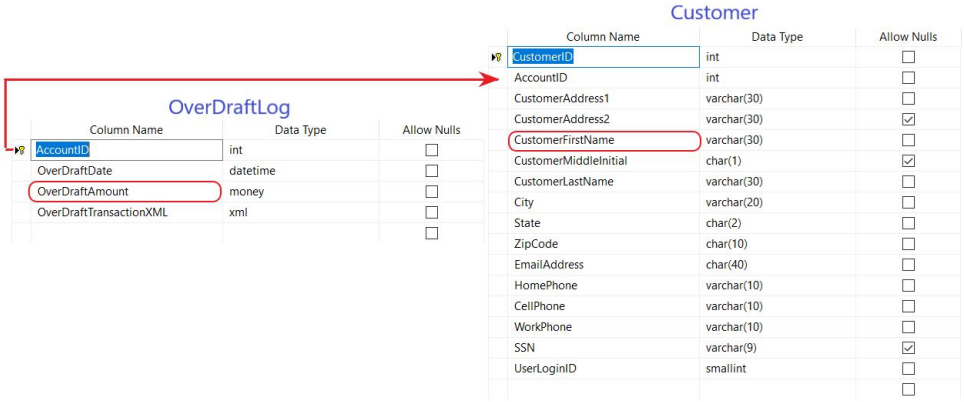
	UserLogin	UserPassword
1	User_User_User_User_User_UserLogin2	Password2

5. Create a view to getting all customers' overdraft amount. [Moderate]

- A View is a virtual table based of the select statement, also the View is stored in the database.
- Dropping View does not affect underlying table or relations
- The View is not physically stored and used for added security

1) Created a View by the name VW_Customer_OD
2) By selecting (select) records from Customer table who have an OverDraft facility and show there OF amounts
3) A cross reference inner join (Join) is built between (Customer, OverDraftLog
4) Identify OverDraft account and amounts

Tables required to create the view:



```sql
/*****************************************************************************************/
/*.....................................................................................*/
/*Question5: Create a view to get all customers' overdraft amount. [Moderate]*/

DROP VIEW VW_Customer_OD;
GO

CREATE VIEW VW_Customer_OD
AS
SELECT DISTINCT c.CustomerFirstName, o.OverDraftAmount
FROM OverDraftLog o
JOIN Customer c
ON o.AccountID = c.AccountID;
GO


/*****************************************************************************************
```

100 %

Messages

```
Commands completed successfully.

Completion time: 2020-12-26T14:55:39.0082498-05:00
```

```sql
/*****************************************************************************************/
/*.....................................................................................*/
/*Question5: Create a view to get all customers' overdraft amount. [Moderate]*/

DROP VIEW VW_Customer_OD;
GO

CREATE VIEW VW_Customer_OD
AS
SELECT DISTINCT c.CustomerFirstName, o.OverDraftAmount
FROM OverDraftLog o
JOIN Customer c
ON o.AccountID = c.AccountID;
SELECT * FROM VW_Customer_OD
GO
```

100 %

Results | Messages

	CustomerFirstName	OverDraftAmount
1	Customer1	0.00
2	Customer2	5.00
3	Customer3	10.00
4	Customer4	15.00
5	Customer5	20.00

6. Create a stored procedure to add "User_" as a prefix to everyone's login (username). [Moderate]

Benefits of using a stored procedure

It can be easily modified: We can easily modify the code inside the stored procedure without the need to restart or deploy the application. For example, If the T-SQL queries are written in the application and if we need to change the logic, we must change the code in the application and re-deploy it. SQL Server Stored procedures eliminate such challenges by storing the code

in the database. so, when we want to change the logic inside the procedure we can just do it by a simple ALTER PROCEDURE statement.

Reduced network traffic: When we use stored procedures instead of writing T-SQL queries at the application level, only the procedure name is passed over the network instead of the whole T-SQL code.

Reusable: Stored procedures can be executed by multiple users or multiple client applications without the need of writing the code again.

Security: Stored procedures reduce the threat by eliminating direct access to the tables. we can also encrypt the stored procedures while creating them so that source code inside the stored procedure is not visible.

A stored procedure is a prepared SQL code that you can save, so that code can be reused over and over again. A stored procedure can be easily modified and reduce network traffic. The stored procedures reduce the threat by eliminating direct access to the tables and it is reusable.

- assigning a name for the procedure
- We want to change any existing records in the table. Therefore we should use an update.
- Definition and usage of the CONCAT() function add two or more strings together.

1. Created a stored procedure by the name StP_Update_Login
2. Update table UserLogin and User_ to all UserLogins
3. Execute the procedure to update the records

Column Name	Data Type	Allow Nulls
UserLoginID	smallint	☐
UserLogin	varchar(50)	☐
UserPassword	varchar(20)	☐
		☐

```sql
/*****************************************************************************************/
/*****************.........................PHASE2 .......SARA.........................***************/

--Question6. Create a stored procedure to add "User_" as a prefix to everyone's login (username).[Moderate]
USE BANK;
GO

DROP PROCEDURE StP_Update_Login;
GO

CREATE PROCEDURE StP_Update_Login
AS
UPDATE UserLogins
SET UserLogin = Concat('User_', UserLogin);
GO
EXECUTE StP_Update_Login;
GO

select * from UserLogins;

/*****************************************************************************************/
```

Results | Messages

	UserLoginID	UserLogin	UserPassword
1	1	User_UserLogin1	Password1
2	2	User_UserLogin2	Password2
3	3	User_UserLogin3	Password3
4	4	User_UserLogin4	Password4
5	5	User_UserLogin5	Password5

Data in the original Table

	UserLoginID	UserLogin	UserPassword
1	1	UserLogin1	Password1
2	2	UserLogin2	Password2
3	3	UserLogin3	Password3
4	4	UserLogin4	Password4
5	5	UserLogin5	Password5

Data in the original table after add User_

Results | Messages

	UserLoginID	UserLogin	UserPassword
1	1	User_UserLogin1	Password1
2	2	User_UserLogin2	Password2
3	3	User_UserLogin3	Password3
4	4	User_UserLogin4	Password4
5	5	User_UserLogin5	Password5

7. Create a stored procedure that accepts AccountId as a parameter and returns the customer's full name. [Advanced]

- Created a stored procedure by the name spFullNameFromAccountID
- assigning a name for procedure
- Defining input parameter and it's data type
- Defining output parameter and it's data type
- Execute the procedure to update the records

```
--Question7. Create a stored procedure that accepts AccountId as a parameter and returns customer's full name.

Drop proc spFullNameFromAccountId
GO

create proc spFullNameFromAccountId        --assigning a name for procedure
            @AccountID int,                --Defining input parameter and it's data type
                                           -- Don't Forget to put "," between them
            @Fullname nvarchar(100) output --Defining output parameter and it's data type
as
begin
  if (@AccountID in (select AccountID from Customer_Account))
    begin
       select @Fullname=c.customerFirstName+' '+c.customerMiddleInitial+' '+c.customerLastName
       from customer C

    end
   else
    begin
     print 'This Account Id does not exists!'
    end
end

--Executing for valid account id
Declare @FullName nvarchar(100)
exec spFullNameFromAccountId 1009, @FullName out
Print ' Full name is '+replace (@FullName,'   ',' ')
--Executing for invalid account id
Declare @FullName nvarchar(100)
exec spFullNameFromAccountId 2999, @FullName out
Print ' Full name is '+replace (@FullName,'   ',' ')

go

/*****************************************************************************************************/
```

```
Messages
This Account Id does not exists!

Completion time: 2020-12-26T18:13:20.7118059-05:00
```

8. Create a stored procedure that returns error logs inserted in the last 24 hours. [Advanced]

- Extensible Markup Language (XML) is used to describe data.
- The XML standard is a flexible way to create information format and electronically share structured data via the public internet, as well as via corporate networks.
- XML is self defined
- Created a stored procedure by the name StoreP_Errors_24
- assigning a name for procedure
- Defining input parameter and it's data type
- Execute the procedure to update the records

```
/*****************************************************************************************/
/*.........................................................................................*/
--Question8: Create a stored procedure that returns error logs inserted in the last 24 hours. [Advanced]
DROP PROCEDURE StoreP_Errors_24;
GO

CREATE PROCEDURE StoreP_Errors_24
AS
SELECT * FROM LoginErrorLog
WHERE  ErrorTime BETWEEN DATEADD(hh, -24, GETDATE()) AND GETDATE();
GO

EXEC StoreP_Errors_24;
GO
```

Messages

```
Commands completed successfully.

Completion time: 2020-12-26T15:46:37.1331757-05:00
```

```
/*****************************************************************************************/
/*.........................................................................................*/
--Question8: Create a stored procedure that returns error logs inserted in the last 24 hours. [Advanced]
DROP PROCEDURE StoreP_Errors_24;
GO

CREATE PROCEDURE StoreP_Errors_24
AS
SELECT * FROM LoginErrorLog
WHERE  ErrorTime BETWEEN DATEADD(hh, -24, GETDATE()) AND GETDATE();
GO

EXEC StoreP_Errors_24;
GO
```

Results | Messages

ErrorLogID	ErrorTime	FailedTransactionXML

9. Create a stored procedure that takes a deposit as a parameter and updates the CurrentBalance value for that particular account. [Advanced]

Before Updating CurrentBalance, we need to insert the value into TransactionLog.

1. Created a stored procedure by the name SP_Update_cBalance_After_Deposit
2. Which takes 2 input parameters AccountID and Deposit
3. Updating table Account, CurrentBalance field with additional deposit for the input AccountID

```sql
/*****************************************************************************************/
/*.........................................................................................*/
--Question9: Create a stored procedure that takes a deposit as a parameter and updates CurrentBalance value for that particular account. [Advanced]
DROP PROCEDURE SP_Update_cBalance_After_Deposit;
GO

CREATE PROCEDURE SP_Update_cBalance_After_Deposit @AccountID INT, @Deposit INT
AS
UPDATE Account
SET CurrentBalance = CurrentBalance + @Deposit
where AccountID = @AccountID;
GO

EXEC SP_Update_cBalance_After_Deposit 3, 300;
GO


/*****************************************************************************************/
```

Messages

```
Commands completed successfully.

Completion time: 2020-12-26T17:09:25.6629788-05:00
```

```sql
/*****************************************************************************************/
/*.........................................................................................*/
--Question9: Create a stored procedure that takes a deposit as a parameter and updates CurrentBalance value for that particular account. [Advanced]
DROP PROCEDURE SP_Update_cBalance_After_Deposit;
GO

CREATE PROCEDURE SP_Update_cBalance_After_Deposit @AccountID INT, @Deposit INT
AS
UPDATE Account
SET CurrentBalance = CurrentBalance + @Deposit
where AccountID = @AccountID;
GO

EXEC SP_Update_cBalance_After_Deposit 3, 300;
GO


/*****************************************************************************************/
```

Messages

```
(1 row affected)

Completion time: 2020-12-26T17:10:44.8181258-05:00
```

```sql
/*****************************************************************************************************************************/
/*.........................................................................................................................*/
--Question9: Create a stored procedure that takes a deposit as a parameter and updates CurrentBalance value for that particular account. [Advanced]
DROP PROCEDURE SP_Update_cBalance_After_Deposit;
GO

CREATE PROCEDURE SP_Update_cBalance_After_Deposit @AccountID INT, @Deposit INT
AS
UPDATE Account
SET CurrentBalance = CurrentBalance + @Deposit
where AccountID = @AccountID;
GO

EXEC SP_Update_cBalance_After_Deposit 3, 300;
GO

SELECT * FROM Account
```

100 %

Results | Messages

	AccountID	CurrentBalance	AccountTypeID	AccountStatusTypeID	InterestSavingRatesID
1	1	22000	1	1	1
2	2	25000	2	2	2
3	3	17600	1	1	1
4	4	45000	2	2	2
5	5	35000	2	2	2

Results | Messages

	AccountID	CurrentBalance	AccountTypeID	AccountStatusTypeID	InterestSavingRatesID
1	1	22000	1	1	1
2	2	25000	2	2	2
3	3	17000	1	1	1
4	4	50000	2	2	2
5	5	35000	2	2	2

10. Create a stored procedure that takes a withdrawal amount as a parameter and updates

The SQL SELECT DISTINCT Statement

The SELECT **DISTINCT** statement is used to return only **distinct** (different) values. Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (**distinct**) values. The **SQL DISTINCT** keyword **is** used in conjunction with the SELECT statement to eliminate all the duplicate records and fetching only unique records. There may be a situation when you have multiple duplicate records in a table. The main **difference between Unique** and **Distinct** in **SQL** is that **Unique** helps to ensure that **all** the values **in a** column are different while **Distinct** helps to remove **all** the duplicate records when retrieving the records from a table.

To do this, you **use** the SELECT **DISTINCT** clause as follows: SELECT **DISTINCT** column_name FROM table_name; The query returns only **distinct** values in the specified column. In other words, it removes the duplicate values in the column from the result set.

SELECT DISTINCT *column1*, *column2, ...*

FROM *table_name*;

@ in SQL:

- The @ means it's a parameter that you will supply a value for later in your code. This is the best way of protecting against SQL injection. Create your query using parameters, rather than concatenating strings and variables.
- The reason that there is no possibility of SQL injection is that the @ is replaced with a string. When that database receives one of these variables it knows not to escape the variable for anything inside of it.

Defining input variables and their data type and separating them with comma. Note we can use soft brackets

When you are using return or output: In Execution time You must assign the result of the procedure in the Temporary variable and then print or select it, so you need declaring that temporary variable and its data type before execution i.e. :

1) you have to declare output variables (local variables)

2) for seeing the result you have to use print or select

Note: Use return just when output is an integer

```sql
/*****************************************************************************************/
/*.........................................................................................*/
--Question10.Create a stored procedure that takes a withdrawal amount as a parameter and updates

DROP PROCEDURE UpdateCBalanceWithdraw;
GO

CREATE PROCEDURE UpdateCBalanceWithdraw @AccountID INT, @Withdraw INT
AS
UPDATE Account
SET CurrentBalance = CurrentBalance - @Withdraw
WHERE AccountID = @AccountID;
GO

EXEC UpdateCBalanceWithdraw 4, 1000;
GO

SELECT * FROM Account;

/......................................................................./
```

Results | Messages

	AccountID	CurrentBalance	AccountTypeID	AccountStatusTypeID	InterestSavingRatesID
1	1	22000	1	1	1
2	2	25000	2	2	2
3	3	17000	1	1	1
4	4	50000	2	2	2
5	5	35000	2	2	2

Results | Messages

	AccountID	CurrentBalance	AccountTypeID	AccountStatusTypeID	InterestSavingRatesID
1	1	22000	1	1	1
2	2	25000	2	2	2
3	3	17000	1	1	1
4	4	50000	2	2	2
5	5	35000	2	2	2

Results | Messages

	AccountID	CurrentBalance	AccountTypeID	AccountStatusTypeID	InterestSavingRatesID
1	1	22000	1	1	1
2	2	25000	2	2	2
3	3	17000	1	1	1
4	4	49000	2	2	2
5	5	35000	2	2	2

For AccountID 4, Withdraw 1000