



Banking Database

Iresha Samarakoon

Quan Bui

Requirement Analysis & Normalization

Requirements Analysis

The BankingDB is the database used for banking systems, where there are customers—who make transactions—and admins—who are also customers but can CRUD (Create, Read, Update, Delete) other customers. Each customer has their own credit points and credit limit, which indicate how trustworthy they are to the bank and whether they are eligible for a mortgage increase or a higher credit limit.

During the lifetime of a user in the bank, they may change their personal details frequently, from their address and phone number to their password or email. To keep track of all these changes, a table will store every modification made throughout their journey in the bank. Additionally, the user's login history will also be recorded for security and auditing purposes.

The Thunder Bay Bank currently only approves internal transactions but has the potential to operate nationally (between different banks). A user in The Thunder Bay Bank can have multiple accounts, whether it is a savings, chequing , investment, or credit account. Each type of account comes with different interest rates, which are competitive with other banks.

Every five years, if an account is not renewed, it will automatically become inactive. There are two types of transactions: Debit (DEB) and Credit (CRE).

- Debit transactions include any deposit into an account, such as transfers from another account, work payments, or returned transactions.
- Credit transactions involve deductions from the account, such as monthly payments, withdrawals, or payments for groceries.

Entities in the Database

- ❖ Account
- ❖ Account Type
- ❖ Bank
- ❖ Branch
- ❖ City
- ❖ Currency
- ❖ Login History
- ❖ Transaction
- ❖ Transaction Type
- ❖ User
- ❖ User Account Modification
- ❖ User Credit Points

Normalization

1NF

As per the following table structure, initial tables are in 1NF

- Each table cell contains a single value.
- Each record is unique.

User Table														
UserDid	FirstName	LastName	DOB	Address	City	Province	Country	Username	Password	Email	MobileNumber	SinNumber	CreateDate	UserRole

Branch Table								
BranchDid	BranchId	BranchName	Address	City	Province	Country	PhoneNumber	CityDid

Bank Table							
Bankdid	Bankid	BankName					

Transaction Table							
TransactionDid	TransationDetails	Tran_Status	TransactionAmount	CurrencyDID	TransactionTypeDid	CreatedDate	UserDid

Account type table							
AccountTypeDid	AccountTypeid	Description	InterestRate				

Account Table									
AccountDid	AccountNumber	PreviousBalance	CurrentBalance	CurrencyCode	OpenDate	EndDate	IsActive	BranchDid	UserDid

User Credit Points Table				
UserCreditPointDid	UserDid	Point	CreditLimit	CurrencyDID

User Account Modification Table					
Did	UserDid	UpdatedField	OldValue	NewValue	ModifiedDate

Login History Table			
LoginHistoryDid	LoginTime	LogOutTime	UserDid

Currency Table		
CurrencyDID	CurrencyCod	CurrencyDescription

2NF

As per the following table structure, initial tables are in 2NF

- Tables are in 1NF.
- All non key attributes are fully functional dependent on the primary key.

User Table														
UserDid	FirstName	LastName	DOB	Address	City	Province	Country	Username	Password	Email	MobileNumber	SinNumber	CreateDate	UserRole

Branch Table								
BranchDid	BranchId	BranchName	Address	City	Province	Country	PhoneNumber	CityDid

Bank Table							
Bankdid	Bankid	BankName					

Transaction Table							
TransactionDid	TransationDetails	Tran_Status	TransactionAmount	CurrencyDID	TransactionTypeDid	CreatedDate	UserDid

Account type table							
AccountTypeDid	AccountTypeid	Description	InterestRate				

Account Table									
AccountDid	AccountNumber	PreviousBalance	CurrentBalance	CurrencyCode	OpenDate	EndDate	IsActive	BranchDid	UserDid

User Credit Points Table				
UserCreditPointDid	UserDid	Point	CreditLimit	CurrencyDID

User Account Modification Table					
Did	UserDid	UpdatedField	OldValue	NewValue	ModifiedDate

Login History Table			
LoginHistoryDid	LoginTime	LogOutTime	UserDid

Currency Table		
CurrencyDID	CurrencyCod	CurrencyDescription

Transaction Type		
TransactionTypeDid	TransactionTypeId	Description

3NF

As per the following table structure, initial tables are not in 3NF

- Tables are in 2NF.
- However, in the user and branch tables province and country are dependent on the city. Therefore, to make all the table in 3NF, we have created City table with province and country and link User table and branch table using CityDid.

User Table														
UserDid	FirstName	LastName	DOB	Address	City	Province	Country	Username	Password	Email	MobileNumber	SinNumber	CreateDate	UserRole

Branch Table									
BranchDid	BranchId	BranchName	Address	City	Province	Country	PhoneNumber	CityDid	Bankdid

Transaction Table								
TransactionDid	TransactionDetails	Tran_Status	TransactionAmount	CurrencyDID	TransactionTypeDid	CreatedDate	UserDid	AccountDid

Bank Table					
Bankdid	Bankid	BankName			

Account type table					
AccountTypeDid	AccountTypeid	Description	InterestRate		

Account Table										
AccountDid	AccountNumber	PreviousBalance	CurrentBalance	CurrencyCode	OpenDate	EndDate	IsActive	BranchDid	UserDid	AccountTypeDid

User Credit Points Table				
UserCreditPointDid	UserDid	Point	CreditLimit	CurrencyDID

User Account Modification Table					
Did	UserDid	UpdatedField	OldValue	NewValue	ModifiedDate

Login History Table			
LoginHistoryDid	LoginTime	LogOutTime	UserDid

Transaction Type		
TransactionTypeID	TransactionTypeid	Description

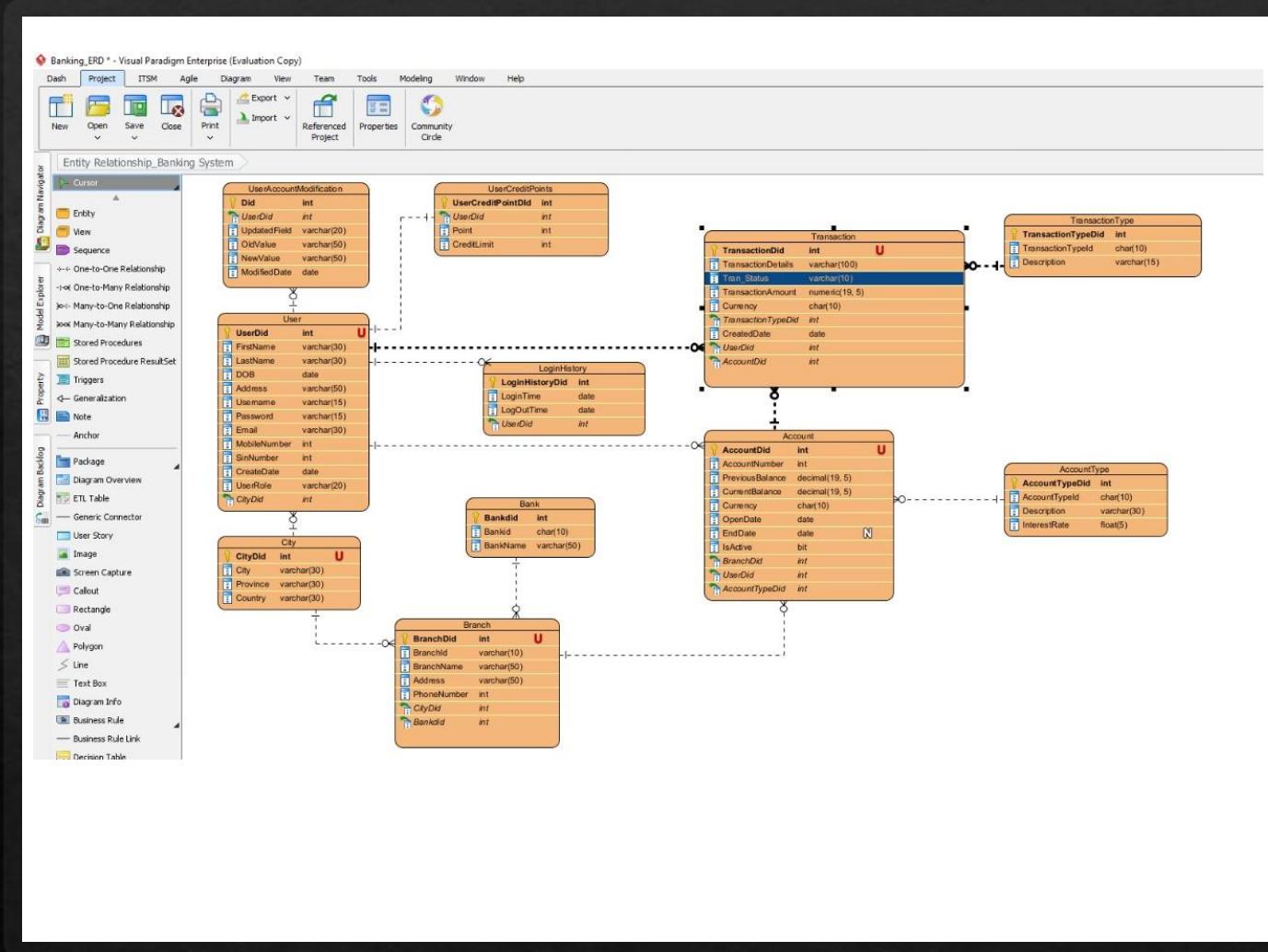
Currency Table		
CurrencyDID	CurrencyCode	CurrencyDescription

3NF Cont.

After the normalization, table are as follows.

Login History Table	Currency Table													
<table border="1"> <thead> <tr> <th><u>LoginHistoryDid</u></th><th><u>LoginTime</u></th><th><u>LogOutTime</u></th><th><u>UserDid</u></th></tr> </thead> </table>	<u>LoginHistoryDid</u>	<u>LoginTime</u>	<u>LogOutTime</u>	<u>UserDid</u>	<table border="1"> <thead> <tr> <th><u>CurrencyDID</u></th><th><u>CurrencyCode</u></th><th><u>CurrencyDescription</u></th></tr> </thead> </table>	<u>CurrencyDID</u>	<u>CurrencyCode</u>	<u>CurrencyDescription</u>						
<u>LoginHistoryDid</u>	<u>LoginTime</u>	<u>LogOutTime</u>	<u>UserDid</u>											
<u>CurrencyDID</u>	<u>CurrencyCode</u>	<u>CurrencyDescription</u>												
Bank Table														
<table border="1"> <thead> <tr> <th><u>Bankdid</u></th><th><u>Bankid</u></th><th><u>BankName</u></th></tr> </thead> </table>	<u>Bankdid</u>	<u>Bankid</u>	<u>BankName</u>	<table border="1"> <thead> <tr> <th><u>TransactionTypeDid</u></th><th><u>TransactionTypeDid</u></th><th><u>Description</u></th></tr> </thead> </table>	<u>TransactionTypeDid</u>	<u>TransactionTypeDid</u>	<u>Description</u>							
<u>Bankdid</u>	<u>Bankid</u>	<u>BankName</u>												
<u>TransactionTypeDid</u>	<u>TransactionTypeDid</u>	<u>Description</u>												
Account type table														
	<table border="1"> <thead> <tr> <th><u>AccountTypeDid</u></th><th><u>AccountTypeID</u></th><th><u>Description</u></th><th><u>InterestRate</u></th></tr> </thead> </table>	<u>AccountTypeDid</u>	<u>AccountTypeID</u>	<u>Description</u>	<u>InterestRate</u>									
<u>AccountTypeDid</u>	<u>AccountTypeID</u>	<u>Description</u>	<u>InterestRate</u>											
Transaction Table														
<table border="1"> <thead> <tr> <th><u>TransactionDid</u></th><th><u>TransactionDetails</u></th><th><u>Tran_Status</u></th><th><u>TransactionAmount</u></th><th><u>CurrencyDID</u></th><th><u>TransactionTypeDid</u></th><th><u>CreatedDate</u></th><th><u>UserDid</u></th><th><u>AccountDid</u></th></tr> </thead> </table>	<u>TransactionDid</u>	<u>TransactionDetails</u>	<u>Tran_Status</u>	<u>TransactionAmount</u>	<u>CurrencyDID</u>	<u>TransactionTypeDid</u>	<u>CreatedDate</u>	<u>UserDid</u>	<u>AccountDid</u>					
<u>TransactionDid</u>	<u>TransactionDetails</u>	<u>Tran_Status</u>	<u>TransactionAmount</u>	<u>CurrencyDID</u>	<u>TransactionTypeDid</u>	<u>CreatedDate</u>	<u>UserDid</u>	<u>AccountDid</u>						
Account Table														
<table border="1"> <thead> <tr> <th><u>AccountDid</u></th><th><u>AccountNumber</u></th><th><u>PreviousBalance</u></th><th><u>CurrentBalance</u></th><th><u>CurrencyCode</u></th><th><u>OpenDate</u></th><th><u>EndDate</u></th><th><u>IsActive</u></th><th><u>BranchDid</u></th><th><u>UserDid</u></th><th><u>AccountTypeID</u></th></tr> </thead> </table>	<u>AccountDid</u>	<u>AccountNumber</u>	<u>PreviousBalance</u>	<u>CurrentBalance</u>	<u>CurrencyCode</u>	<u>OpenDate</u>	<u>EndDate</u>	<u>IsActive</u>	<u>BranchDid</u>	<u>UserDid</u>	<u>AccountTypeID</u>			
<u>AccountDid</u>	<u>AccountNumber</u>	<u>PreviousBalance</u>	<u>CurrentBalance</u>	<u>CurrencyCode</u>	<u>OpenDate</u>	<u>EndDate</u>	<u>IsActive</u>	<u>BranchDid</u>	<u>UserDid</u>	<u>AccountTypeID</u>				
User Credit Points Table														
<table border="1"> <thead> <tr> <th><u>UserCreditPointDid</u></th><th><u>UserDid</u></th><th><u>Point</u></th><th><u>CreditLimit</u></th><th><u>CurrencyDID</u></th></tr> </thead> </table>	<u>UserCreditPointDid</u>	<u>UserDid</u>	<u>Point</u>	<u>CreditLimit</u>	<u>CurrencyDID</u>	<table border="1"> <thead> <tr> <th><u>Did</u></th><th><u>UserDid</u></th><th><u>UpdatedField</u></th><th><u>OldValue</u></th><th><u>NewValue</u></th><th><u>ModifiedDate</u></th></tr> </thead> </table>	<u>Did</u>	<u>UserDid</u>	<u>UpdatedField</u>	<u>OldValue</u>	<u>NewValue</u>	<u>ModifiedDate</u>		
<u>UserCreditPointDid</u>	<u>UserDid</u>	<u>Point</u>	<u>CreditLimit</u>	<u>CurrencyDID</u>										
<u>Did</u>	<u>UserDid</u>	<u>UpdatedField</u>	<u>OldValue</u>	<u>NewValue</u>	<u>ModifiedDate</u>									
User Table														
<table border="1"> <thead> <tr> <th><u>UserDid</u></th><th><u>FirstName</u></th><th><u>LastName</u></th><th><u>DOB</u></th><th><u>Address</u></th><th><u>CityDid</u></th><th><u>Username</u></th><th><u>Password</u></th><th><u>Email</u></th><th><u>MobileNumber</u></th><th><u>SinNumber</u></th><th><u>CreateDate</u></th><th><u>UserRole</u></th></tr> </thead> </table>	<u>UserDid</u>	<u>FirstName</u>	<u>LastName</u>	<u>DOB</u>	<u>Address</u>	<u>CityDid</u>	<u>Username</u>	<u>Password</u>	<u>Email</u>	<u>MobileNumber</u>	<u>SinNumber</u>	<u>CreateDate</u>	<u>UserRole</u>	
<u>UserDid</u>	<u>FirstName</u>	<u>LastName</u>	<u>DOB</u>	<u>Address</u>	<u>CityDid</u>	<u>Username</u>	<u>Password</u>	<u>Email</u>	<u>MobileNumber</u>	<u>SinNumber</u>	<u>CreateDate</u>	<u>UserRole</u>		
Branch Table														
<table border="1"> <thead> <tr> <th><u>BranchDid</u></th><th><u>Branchid</u></th><th><u>BranchName</u></th><th><u>Address</u></th><th><u>CityDid</u></th><th><u>PhoneNumbe</u></th><th><u>r</u></th><th><u>CityDid</u></th><th><u>Bankdid</u></th></tr> </thead> </table>	<u>BranchDid</u>	<u>Branchid</u>	<u>BranchName</u>	<u>Address</u>	<u>CityDid</u>	<u>PhoneNumbe</u>	<u>r</u>	<u>CityDid</u>	<u>Bankdid</u>					
<u>BranchDid</u>	<u>Branchid</u>	<u>BranchName</u>	<u>Address</u>	<u>CityDid</u>	<u>PhoneNumbe</u>	<u>r</u>	<u>CityDid</u>	<u>Bankdid</u>						
City Table														
<table border="1"> <thead> <tr> <th><u>CityDid</u></th><th><u>City</u></th><th><u>Province</u></th><th><u>Country</u></th></tr> </thead> </table>	<u>CityDid</u>	<u>City</u>	<u>Province</u>	<u>Country</u>										
<u>CityDid</u>	<u>City</u>	<u>Province</u>	<u>Country</u>											

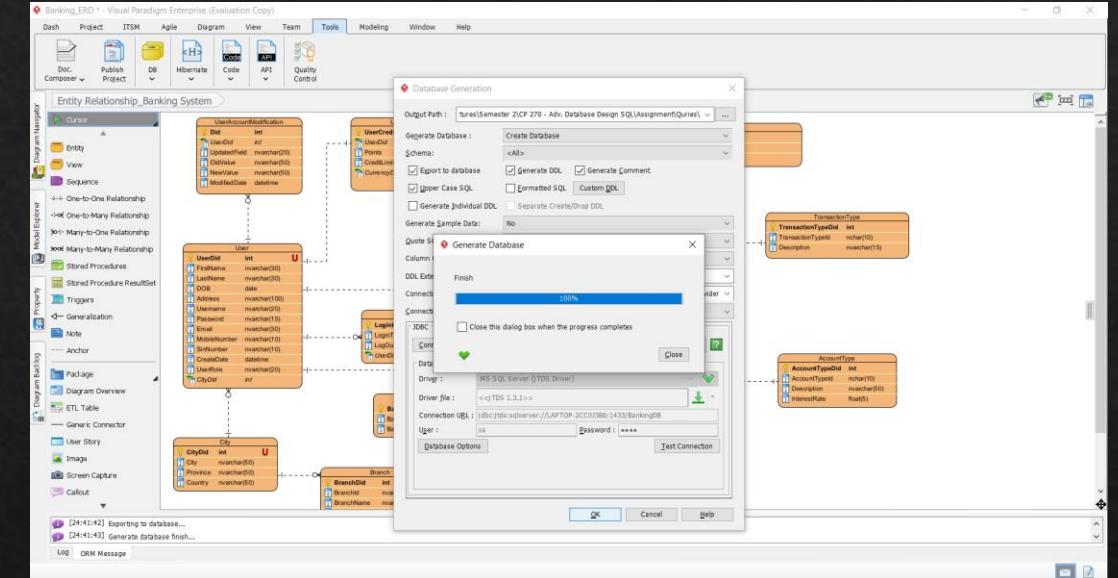
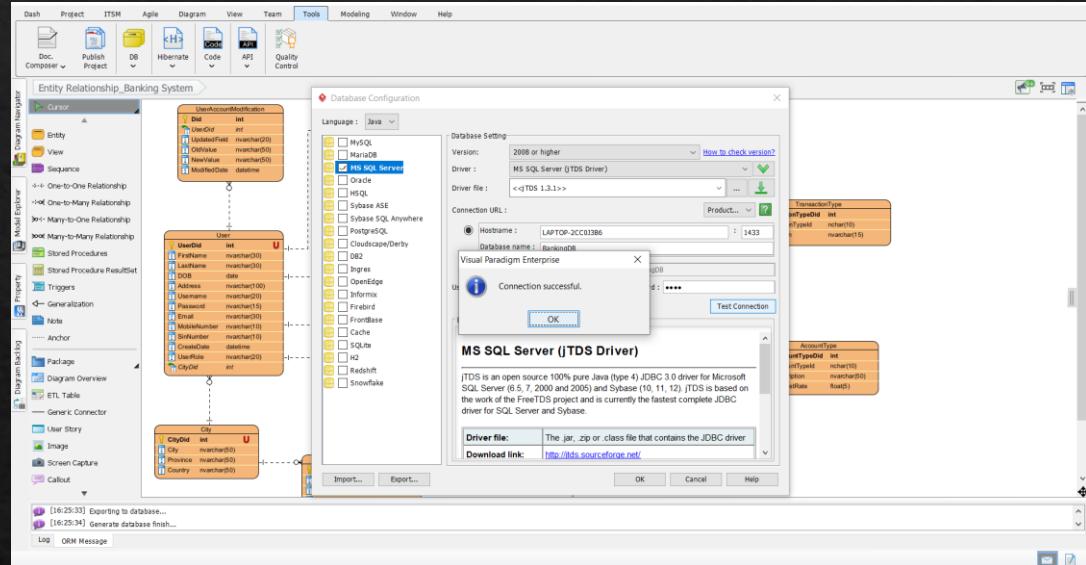
ERD and Database Conversion



ERD

The finalized ERD after
the normalization.

Database Conversion

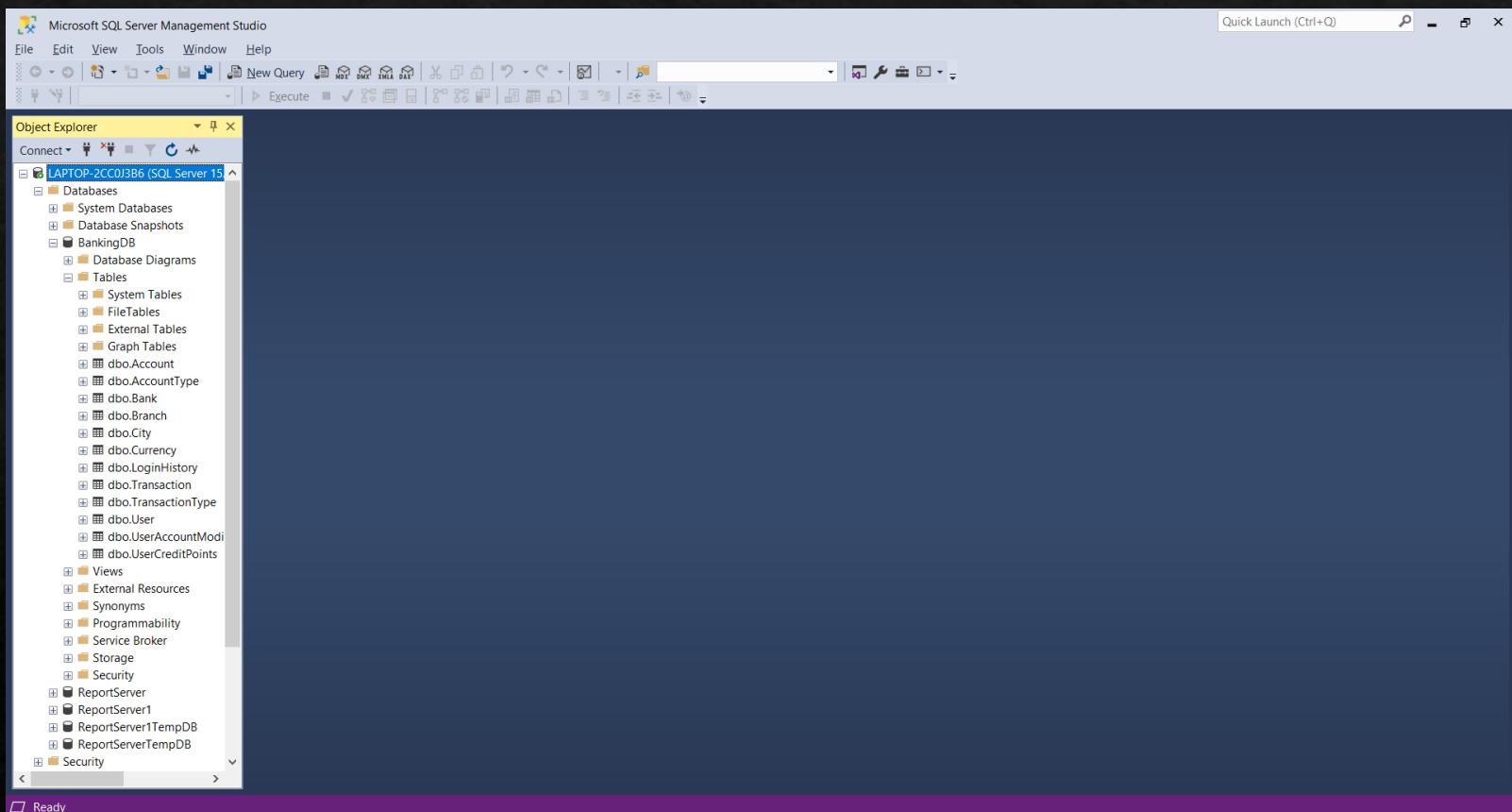


Successful database conversion

Database Creation & Reports

Database Creation

- Table structure in the database



Reports

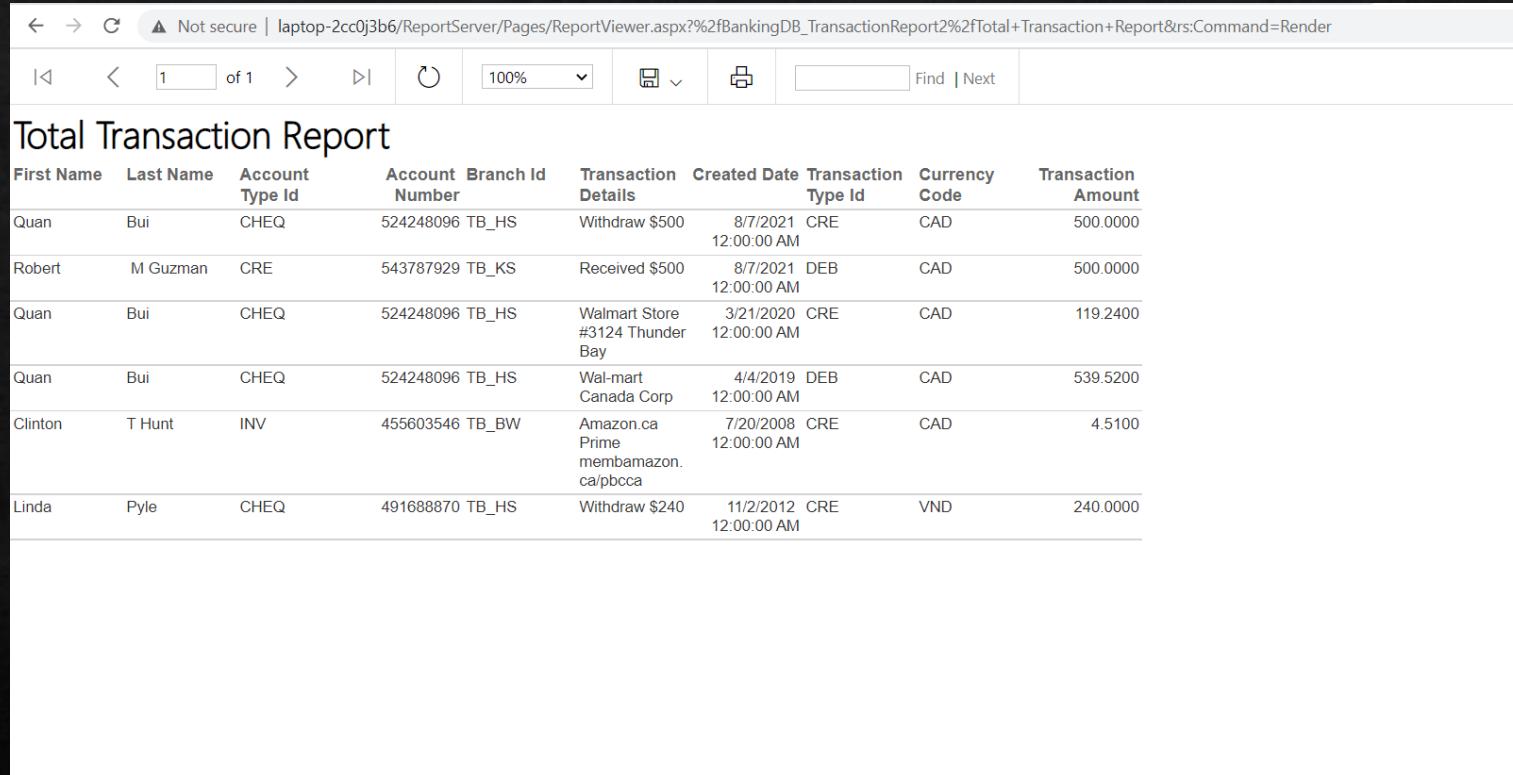
➤ Branch report by City

The screenshot shows a web browser window with a report titled "Branch Report by City". The report is displayed in a table format with the following columns: Bank Name, Branch Id, Branch Name, Address, Phone Number, City, and Province. There are eight rows of data, each representing a branch of the Bank of Thunder Bay.

Bank Name	Branch Id	Branch Name	Address	Phone Number	City	Province
Bank of Thunder Bay	TB_HS	TB Hewitson St	745 Hewitson St, Thunder Bay, ON P7B 6B5	8076235626	Thunder Bay	ON
Bank of Thunder Bay	TB_AW	TB Arthur W	1195 Arthur St W, Thunder Bay, ON P7E 6E2	8076245100	Thunder Bay	ON
Bank of Thunder Bay	TB_RR	TB Red River	225 Red River Rd, Thunder Bay, ON P7B 1A7	8073435600	Thunder Bay	ON
Bank of Thunder Bay	TB_KS	TB King St	40 King St W, Toronto, ON M5H 1H1	4168666430	Toronto	ON
Bank of Thunder Bay	TB_BS	TB Bay St	392 Bay St., Toronto, ON M5H 3K5	4168665700	Toronto	ON
Bank of Thunder Bay	TB_HS	TB Honby St	1010 Hornby St, Vancouver, BC V6Z 1V6	6046655138	Vancouver	BC
Bank of Thunder Bay	TB_BW	TB Broadway	505 W Broadway, Vancouver, BC V5Z 1E7	6046658650	Vancouver	BC
Bank of Thunder Bay	TB_HM	TB Homer	996 Homer St, Vancouver, BC V6B 2W7	6046650747	Vancouver	BC

Reports Cont.

➤ Total Transaction report



The screenshot shows a Microsoft Report Server interface with a 'Total Transaction Report' table. The table details various transactions across different accounts and branches. The columns include First Name, Last Name, Account Type Id, Account Number, Branch Id, Transaction Details, Created Date, Transaction Type Id, Currency Code, and Transaction Amount. The transactions listed are:

First Name	Last Name	Account Type Id	Account Number	Branch Id	Transaction Details	Created Date	Transaction Type Id	Currency Code	Transaction Amount
Quan	Bui	CHEQ	524248096	TB_HS	Withdraw \$500	8/7/2021 12:00:00 AM	CRE	CAD	500.0000
Robert	M Guzman	CRE	543787929	TB_KS	Received \$500	8/7/2021 12:00:00 AM	DEB	CAD	500.0000
Quan	Bui	CHEQ	524248096	TB_HS	Walmart Store #3124 Thunder Bay	3/21/2020 12:00:00 AM	CRE	CAD	119.2400
Quan	Bui	CHEQ	524248096	TB_HS	Wal-mart Canada Corp	4/4/2019 12:00:00 AM	DEB	CAD	539.5200
Clinton	T Hunt	INV	455603546	TB_BW	Amazon.ca Prime membamazon.ca/pbccca	7/20/2008 12:00:00 AM	CRE	CAD	4.5100
Linda	Pyle	CHEQ	491688870	TB_HS	Withdraw \$240	11/2/2012 12:00:00 AM	CRE	VND	240.0000

Reports Cont.

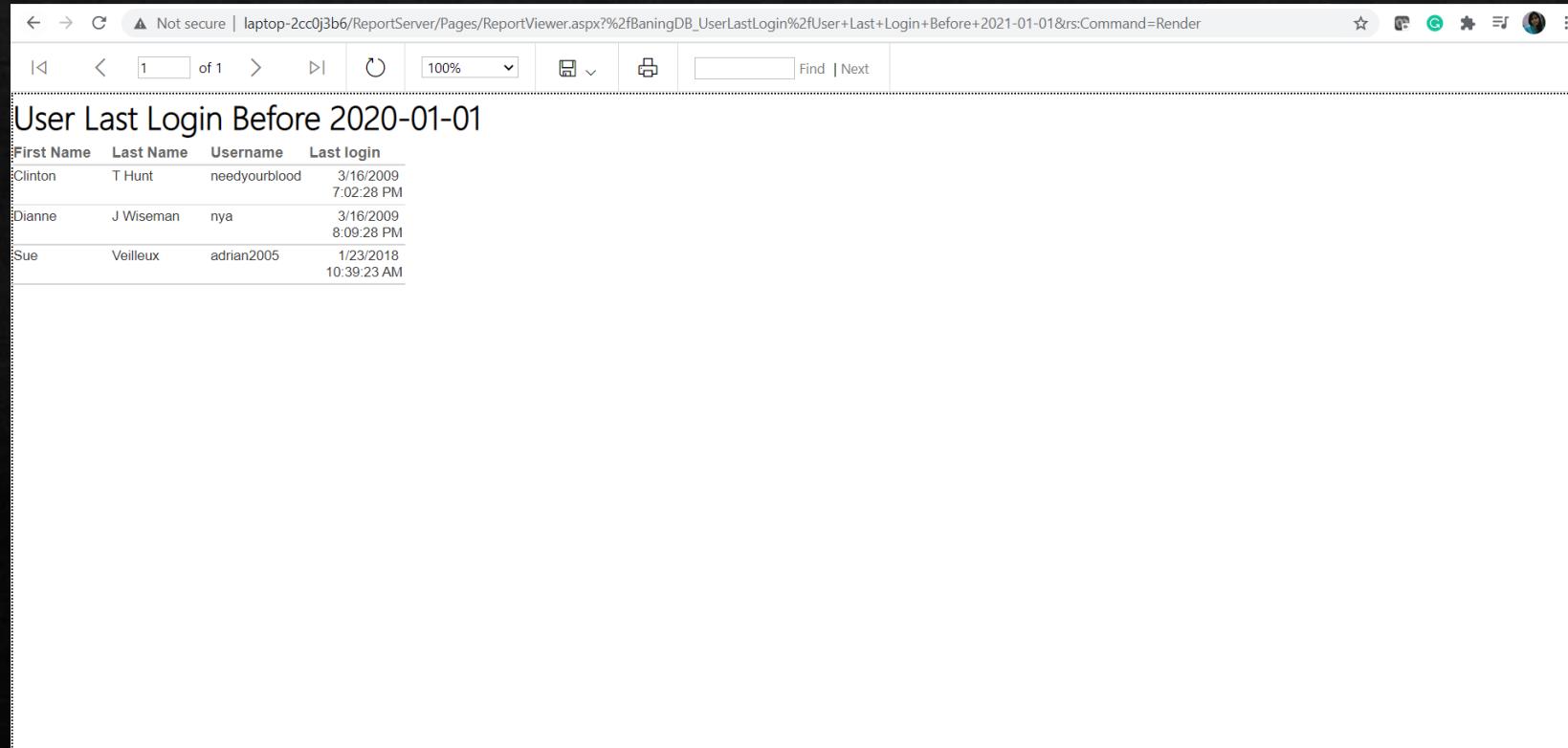
- Total Transaction By Branch

The screenshot shows a Microsoft Report Server interface. At the top, there's a navigation bar with back, forward, and search icons, followed by a message: "Not secure | laptop-2cc0j3b6/ReportServer/Pages/ReportViewer.aspx?%2fBankingDB_TotalTra". Below the navigation is a toolbar with zoom controls (100%), a refresh button, and print and download icons. The main content area has a title "Total Transaction By Branch" and a table with four columns: "Branch Id", "Branch Name", "Currency Code", and "Sum Of Transaction". The table contains four rows of data:

Branch Id	Branch Name	Currency Code	Sum Of Transaction
TB_BW	TB Broadway	CAD	4.5100
TB_HS	TB Honrby St	CAD	1158.7600
TB_HS	TB Honrby St	VND	240.0000
TB_KS	TB King St	CAD	500.0000

Reports Cont.

- User Last login before 2020-01-01



The screenshot shows a Microsoft Report Server interface with a title bar and various navigation buttons. The main content area displays a table titled "User Last Login Before 2020-01-01". The table has four columns: First Name, Last Name, Username, and Last login. The data shows three users who last logged in before January 1, 2020.

First Name	Last Name	Username	Last login
Clinton	T Hunt	needyourblood	3/16/2009 7:02:28 PM
Dianne	J Wiseman	nya	3/16/2009 8:09:28 PM
Sue	Veilleux	adrian2005	1/23/2018 10:39:23 AM

Reports Cont.

- User Credit Point Report

The screenshot shows a Microsoft Report Server report titled "User Credit Point Report". The report is displayed in a web browser window with a dark theme. At the top, there is a navigation bar with icons for back, forward, search, and refresh, followed by a status message "Not secure | laptop-2cc0j3b6/ReportServer/Pages/ReportViewer.aspx?%2fBankingDB_UserCreditP" and a zoom level of "100%". Below the title, there is a table with the following data:

First Name	Last Name	Currency Code	Points	Credit Limit
Quan	Bui	CAD	689	500.0000
Linda	Pyle	CAD	1200	2000.0000
Sue	Veilleux	USD	524	500.0000
Dianne	J Wiseman	VND	246	36000000.000 0
Michael	M Buzbee	CAD	798	1000.0000

Constraint & Indices

Database Tables

◆ Account Table

The screenshot shows the Object Explorer pane of SQL Server Management Studio. The database selected is 'BankingDB'. Under the 'Tables' node, the 'dbo.Account' table is selected and highlighted in blue. The 'Columns' node is expanded, showing the following columns:

- AccountDid (PK, int, not null)
- AccountNumber (int, not null)
- PreviousBalance (money, not null)
- CurrentBalance (money, not null)
- CurrencyCode (nchar(10), not null)
- OpenDate (datetime, not null)
- EndDate (datetime, null)
- IsActive (bit, not null)
- BranchDid (FK, int, not null)
- UserDid (FK, int, not null)
- AccountTypeDid (FK, int, not null)

Under the 'Keys' node, there are three foreign key constraints:

- PK_Account_B8A431C84CA98972
- FK_Account_AccTy
- FK_Account_Branch
- FK_Account_User

Under the 'Constraints' node, there are two default constraints:

- DF_Account_Currenc_2D27B809
- DF_Account_OpenDat_2E1BDC42

Under the 'Triggers' node, there are no triggers listed.

Under the 'Indexes' node, there is one clustered index:

- PK_Account_B8A431C84CA98972 (Clustered)

Under the 'Statistics' node, there are no statistics listed.

◆ Branch Table

The screenshot shows the Object Explorer pane of SQL Server Management Studio. The database selected is 'BankingDB'. Under the 'Tables' node, the 'dbo.Branch' table is selected and highlighted in blue. The 'Columns' node is expanded, showing the following columns:

- BranchDid (PK, int, not null)
- BranchId (nvarchar(10), not null)
- BranchName (nvarchar(50), not null)
- Address (nvarchar(50), not null)
- PhoneNumber (nvarchar(10), not null)
- CityDid (FK, int, not null)
- Bankdid (FK, int, not null)

Under the 'Keys' node, there are three foreign key constraints:

- PK_Branch_3D4D2FCDD7B21DA8
- FK_Branch_Bank
- FK_Branch_City

Under the 'Constraints' node, there are no constraints listed.

Under the 'Triggers' node, there are no triggers listed.

Under the 'Indexes' node, there is one clustered index:

- PK_Branch_3D4D2FCDD7B21DA8 (Clustered)

Under the 'Statistics' node, there are no statistics listed.

◆ City Table

The screenshot shows the Object Explorer pane of SQL Server Management Studio. The database selected is 'BankingDB'. Under the 'Tables' node, the 'dbo.City' table is selected and highlighted in blue. The 'Columns' node is expanded, showing the following columns:

- CityDid (PK, int, not null)
- City (nvarchar(50), not null)
- Province (nvarchar(50), not null)
- Country (nvarchar(50), not null)

Under the 'Keys' node, there is one primary key constraint:

- PK_City_817AA7B55F7152B3

Under the 'Constraints' node, there are no constraints listed.

Under the 'Triggers' node, there are no triggers listed.

Under the 'Indexes' node, there is one clustered index:

- PK_City_817AA7B55F7152B3 (Clustered)

Under the 'Statistics' node, there are no statistics listed.

Database Tables Cont.

❖ Account Type Table

The screenshot shows the SQL Server Object Explorer with the 'BankingDB' database selected. Under the 'Tables' node, the 'dbo.AccountType' table is highlighted with a blue selection bar. The table has the following structure:

- Columns:
 - Bankdid (PK, int, not null)
 - Bankid (nchar(10), not null)
 - BankName (nvarchar(50), not null)
- Keys:
 - PK_Bank_00828D14FF47E439
- Constraints
- Triggers
- Indexes
- Statistics

❖ Currency Table

The screenshot shows the SQL Server Object Explorer with the 'BankingDB' database selected. Under the 'Tables' node, the 'dbo.Currency' table is highlighted with a blue selection bar. The table has the following structure:

- Columns:
 - CurrencyID (PK, int, not null)
 - CurrencyCode (nchar(10), not null)
 - CurrencyDescription (nvarchar(30), not null)
- Keys:
 - PK_Currency_A6CC2BE567A153CE
- Constraints
- Triggers
- Indexes:
 - PK_Currency_A6CC2BE567A153CE (Clustered)
- Statistics

❖ Bank Table

The screenshot shows the SQL Server Object Explorer with the 'BankingDB' database selected. Under the 'Tables' node, the 'dbo.Bank' table is highlighted with a blue selection bar. The table has the following structure:

- Columns:
 - Bankdid (PK, int, not null)
 - Bankid (nchar(10), not null)
 - BankName (nvarchar(50), not null)
- Keys:
 - PK_Bank_00828D149CA314F8
- Constraints
- Triggers
- Indexes:
 - PK_Bank_00828D149CA314F8 (Clustered)
- Statistics

Database Tables Cont.

❖ Transaction Table

The screenshot shows the transaction table structure. It includes:

- Graph Tables
- dbo.Account
- dbo.AccountType
- dbo.Bank
- dbo.Branch
- dbo.City
- dbo.Currency
- dbo.LoginHistory
- dbo.Transaction** (selected)
- Columns
 - TransactionDid (PK, int, not null)
 - TransactionDetails (nvarchar(100), not null)
 - Tran_Status (nvarchar(10), not null)
 - TransactionAmount (money, not null)
 - CurrencyDid (FK, int, not null)
 - TransactionTypeDid (FK, int, not null)
 - CreatedDate (datetime, not null)
 - UserDid (FK, int, not null)
 - AccountDid (FK, int, not null)
- Keys
 - PK_Transact_8F180D8CFDB702DE
 - FK_Transactio_Currency
 - FK_Transaction_Account
 - FK_Transaction_TransacTy
 - FK_Transaction_User
- Constraints
 - DF_Transacti_Creat_30F848ED
- Triggers
- Indexes
 - PK_Transact_8F180D8CFDB702DE (Clustered)
- Statistics

❖ User Table

The screenshot shows the user table structure. It includes:

- dbo.Bank
- dbo.Branch
- dbo.City
- dbo.Currency
- dbo.LoginHistory
- dbo.Transaction
- dbo.TransactionType
- dbo.User** (selected)
- Columns
 - UserDid (PK, int, not null)
 - FirstName (nvarchar(30), not null)
 - LastName (nvarchar(30), not null)
 - DOB (date, not null)
 - Address (nvarchar(100), not null)
 - Username (nvarchar(20), not null)
 - Password (nvarchar(15), not null)
 - Email (nvarchar(30), not null)
 - MobileNumber (nvarchar(10), not null)
 - SinNumber (nvarchar(10), not null)
 - CreateDate (datetime, not null)
 - UserRole (nvarchar(20), not null)
 - CityDid (FK, int, not null)
- Keys
 - PK_User
 - FK_User_City
- Constraints
 - CHK_Email
 - DF_User_CreateDate_24927208
- Triggers
- Indexes
 - PK_User (Clustered)
- Statistics

❖ User Account Modification Table

The screenshot shows the user account modification table structure. It includes:

- System Tables
- FileTables
- External Tables
- Graph Tables
- dbo.Account
- dbo.AccountType
- dbo.Bank
- dbo.Branch
- dbo.City
- dbo.Currency
- dbo.LoginHistory
- dbo.Transaction
- dbo.TransactionType
- dbo.User**
- dbo.UserAccountModification** (selected)
- Columns
 - Did (PK, int, not null)
 - UserDid (FK, int, not null)
 - UpdatedField (nvarchar(20), not null)
 - OldValue (nvarchar(50), not null)
 - NewValue (nvarchar(50), not null)
 - ModifiedDate (datetime, not null)
- Keys
 - PK_UserAcco_C03122187B9761DF
 - FK_UserAccoun_User
- Constraints
 - DF_UserAccou_Modif_35BCFE0A
- Triggers
- Indexes
- Statistics

Database Tables Cont.

❖ User Credit Point Table

The screenshot shows the Object Explorer window in SQL Server Management Studio (SSMS) with the 'BankingDB' database selected. Under the 'Tables' node, the 'dbo.UserCreditPoints' table is highlighted. The tree view shows the following structure:

- dbo.Account
- dbo.AccountType
- dbo.Bank
- dbo.Branch
- dbo.City
- dbo.Currency
- dbo.LoginHistory
- dbo.Transaction
- dbo.TransactionType
- dbo.User
- dbo.UserAccountModification
- dbo.UserCreditPoints**
 - Columns
 - UserCreditPointDid (PK, int, not null)
 - UserDid (FK, int, not null)
 - Points (int, not null)
 - CreditLimit (money, not null)
 - CurrencyDid (FK, int, not null)
 - Keys
 - PK_UserCred_630D6573CE64D1DD
 - FK_UserCredit_Currency
 - FK_UserCredit_User
 - Constraints
 - Triggers
- dbo.Columns
- dbo.Indexes
 - PK_UserCred_630D6573CE64D1DD (Clustered)
- dbo.Statistics

❖ Login History Table

The screenshot shows the Object Explorer window in SSMS with the 'BankingDB' database selected. Under the 'Tables' node, the 'dbo.LoginHistory' table is highlighted. The tree view shows the following structure:

- BankingDB
- Database Diagrams
- Tables
 - System Tables
 - FileTables
 - External Tables
 - Graph Tables
 - dbo.Account
 - dbo.AccountType
 - dbo.Bank
 - dbo.Branch
 - dbo.City
 - dbo.Currency
 - dbo.LoginHistory**
 - Columns
 - LoginHistoryDid (PK, int, not null)
 - LoginTime (datetime, not null)
 - LogOutTime (datetime, not null)
 - UserDid (FK, int, not null)
 - Keys
 - PK_LoginHis_B314C22D175E4AB1
 - FK_LoginHisto_User
 - Constraints
 - DF_LoginHist_Login_3C69FB99
 - DF_LoginHist_LogOut_3D5E1FD2
 - Triggers
 - Indexes
 - PK_LoginHis_B314C22D175E4AB1 (Clustered)
 - Statistics

❖ Transaction type Table

The screenshot shows the Object Explorer window in SSMS with the 'BankingDB' database selected. Under the 'Tables' node, the 'dbo.TransactionType' table is highlighted. The tree view shows the following structure:

- System Tables
- FileTables
- External Tables
- Graph Tables
- dbo.Account
- dbo.AccountType
- dbo.Bank
- dbo.Branch
- dbo.City
- dbo.Currency
- dbo.LoginHistory
- dbo.Transaction
- dbo.TransactionType**
 - Columns
 - TransactionTypeID (PK, int, not null)
 - TransactionTypeDid (nchar(10), not null)
 - Description (nvarchar(15), not null)
 - Keys
 - PK_Transact_1F516DBE6477C9D5
 - Constraints
 - Triggers
- dbo.Columns
- dbo.Indexes
 - PK_Transact_1F516DBE6477C9D5 (Clustered)
- dbo.Statistics

Table Expression & Group By-Having Queries

Table Expression

- ◆ Number Of Transactions Per User

The screenshot shows the Object Explorer on the left with the database 'LAPTOP-2CC0J3B6' selected. The 'CTE.sql' query window contains the following code:

```
with Transaction_CTE(UserId, NumberOfTransactions) as (
    select UserId, count(*)
    from dbo.Transaction
    group by UserId
)
select UserId, NumberOfTransactions as "Number Of Transactions Per Person"
from Transaction_CTE
```

The results grid shows the following data:

UserId	Number Of Transactions Per Person
2	3
13	1
15	1
18	1

- ◆ Accounts created after 2020

The screenshot shows the Object Explorer on the left with the database 'LAPTOP-2CC0J3B6' selected. The 'CTE.sql' query window contains the following code:

```
with CreatedAfter20_CTE (FirstName, LastName, CreatedYear) as (
    select FirstName, LastName, YEAR(CreateDate)
    from dbo.User
    where YEAR(CreateDate) >= 2020
)
select * from CreatedAfter20_CTE;
```

The results grid shows the following data:

FirstName	LastName	CreatedYear
Quan	Bui	2021
Linda	Pyle	2021
Sue	Veilleux	2021
Clinton	T Hunt	2020

- ◆ Account balance with currency USD

The screenshot shows the Object Explorer on the left with the database 'LAPTOP-2CC0J3B6' selected. The 'CTE.sql' query window contains the following code:

```
with UsingUSD_CTE (AccountNumber, UserId, CurrentBalance, CurrencyCode) as (
    select AccountNumber, UserId, CurrentBalance, CurrencyCode
    from dbo.Account
    where CurrencyCode = 'USD'
)
select * from UsingUSD_CTE;
```

The results grid shows the following data:

AccountNumber	UserId	CurrentBalance	CurrencyCode
455603546	13	3200.72	USD
543787929	15	720.25	USD

Group By – Having Queries

- ❖ Total User Account balance whose user id = 2

The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure of 'BankingDB' with various tables like 'Account', 'Branch', 'City', etc. In the center, a query editor window titled 'CTE.sql - LAPTOP-2...C0J3B6\resha (53)' contains the following T-SQL code:

```
declare
@table nvarchar(128),
@sql nvarchar(max),
@whereClause nvarchar(50),
@groupBy nvarchar(100);

set @table = N'dbo.[Account]';
set @whereClause = 2;
set @groupBy = N'UserId';
set @sql = N'select UserId, SUM(CurrentBalance) as OverallBalance from ' + @table + ' where UserId='
+ @whereClause + ' group by ' + @groupBy;

EXEC sp_executesql @sql;
```

The results window below shows a single row of data:

UserDid	OverallBalance
2	12500.92

- ❖ User Account modification time per person

The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure of 'BankingDB' with various tables like 'Account', 'Branch', 'City', etc. In the center, a query editor window titled 'SQLQuery4.sql - LA...C0J3B6\resha (72)' contains the following T-SQL code:

```
declare
@table nvarchar(128),
@sql nvarchar(max),
@whereClause nvarchar(50),
@groupBy nvarchar(100);
set @table = N'dbo.[UserAccountModification]';
set @groupBy = N'UserId';
set @sql = N'select UserId, count(*) as ModificationTimes from '
+ @table + ' group by ' + @groupBy;
EXEC sp_executesql @sql;
```

The results window below shows the following data:

UserDid	ModificationTimes
2	3
9	1
11	1
12	2
13	1
15	1

Group By – Having Queries Cont.

- ◆ Number of branches in each city

The screenshot shows the Object Explorer on the left with the 'BankingDB' database selected. The Results pane on the right displays a query that counts the number of branches for each city. The results table shows three cities: Thunder Bay, Toronto, and Vancouver, with their respective branch counts.

```
declare
@table nvarchar(128),
@innerTable nvarchar(128),
@sql nvarchar(max),
@groupBy nvarchar(100);

set @innerTable = N'dbo.[Branch]'
set @table = N'dbo.[City]'
set @groupBy = N'CityDid';

set @sql = N'select * from ' + @table + ' right join (select CityDid, count(*) as NumberofBranches from '
+ @innerTable + ' group by ' + @groupBy + ') as GroupedBranch on ' + @table + '.CityDid=GroupedBranch.CityDid';

EXEC sp_executesql @sql;
```

CityDid	City	Province	Country	CityDid	NumberofBranches
1	Thunder Bay	ON	CAN	1	3
2	Toronto	ON	CAN	2	2
3	Vancouver	BC	CAN	3	3

- ◆ Number of bank accounts in each bank account type

The screenshot shows the Object Explorer on the left with the 'BankingDB' database selected. The Results pane on the right displays a query that counts the number of accounts for each account type. The results table shows three account types: CHEQ, INV, and CRE, with their respective account counts.

```
declare
@table nvarchar(128),
@innerTable nvarchar(128),
@innerName nvarchar(128),
@sql nvarchar(max),
@groupVariable nvarchar(100);

set @innerTable = N'dbo.[Account]'
set @table = N'dbo.[AccountType]'
set @groupVariable = N'AccountTypeID'
set @innerName = N'GroupedAccount'

set @sql = N'select * from ' + @table + ' right join (select ' + @groupVariable
+ ', count(*) as NumberofAccounts from ' + @innerTable + ' group by ' + @groupVariable + ') as '
+ @innerName + ' on ' + @table + '.' + @groupVariable + '=' + @innerName + '.' + @groupVariable;

EXEC sp_executesql @sql;
```

AccountTypeID	AccountType	Description	InterestRate	AccountTypeID	NumberofAccounts
1	CHEQ	Checking Account	0.2	2	2
2	INV	Investment Account	2	3	2
3	CRE	Credit Account	5	4	1

Group By – Having Queries Cont.

- ❖ Total credit points per currency code

The screenshot shows the SQL Server Management Studio interface. On the left is the Object Explorer pane, which lists the database structure for 'LAPTOP-2CC0J3B6' (SQL Server 15.0.2000.5). It includes Databases (System Databases, Database Snapshots), BankingDB (Database Diagrams, Tables: System Tables, FileTables, External Tables, Graph Tables, dbo.Account, dbo.AccountType, dbo.Bank, dbo.Branch, dbo.City, dbo.Currency, dbo.LoginHistory, dbo.Transaction, dbo.TransactionType, dbo.User, dbo.UserAccountModification, dbo.UserCreditPoints), Views, and External Resources. The 'Tables' node under BankingDB is expanded. In the center, there are three tabs: 'SQLQuery4.sql - LA...C0J3B6\Iresha (72)', 'CTE.sql - LAPTOP-2...C0J3B6\Iresha (53)', and 'SQLQuery3.sql - LA...C0J3B6\Iresha (69)*'. The 'SQLQuery3.sql' tab is active and contains the following T-SQL code:

```
declare
    @table nvarchar(128),
    @innerTable nvarchar(128),
    @innerName nvarchar(128),
    @sql nvarchar(max),
    @groupVariable nvarchar(100);

set @innerTable = N'dbo.[UserCreditPoints]';
set @table = N'dbo.[Currency]';
set @groupVariable = N'CurrencyDID';
set @innerName = N'GroupedCurrencies';

set @sql = N'select * from ' + @table + ' right join (select ' + @groupVariable
+ ', count(*) as NumberofAccounts, sum(Points) as TotalCreditPoint from ' + @innerTable + ' group by '
+ @groupVariable + ') as ' + @innerName + ' on ' + @table + '.' + @groupVariable +'=' + @innerName + '.';
+ @groupVariable;

EXEC sp_executesql @sql;
```

The 'Results' tab at the bottom shows the output of the query:

	CurrencyID	CurrencyCode	CurrencyDescription	CurrencyDID	NumberofAccounts	TotalCreditPoint
1	1	CAD	Canadian Dollar	1	3	2687
2	2	VND	Vietnam Dong	2	1	246
3	3	USD	US Dollar	3	1	524

Views, Transaction & Sub Queries

Views

- ❖ Accounts which is going to be expired before 2025

The screenshot shows the Object Explorer on the left with various database objects like External Tables, Graph Tables, and Views. A new view named 'dbo.Account_End' is selected. The main pane displays the T-SQL code for creating the view:

```
CREATE VIEW Account_End AS
SELECT AccountNumber, CurrentBalance, CurrencyCode, EndDate
FROM [Account]
WHERE Year(EndDate) < 2025;
```

The results pane below shows two rows of data from the 'Account' table where the 'EndDate' is before 2025.

AccountNumber	CurrentBalance	CurrencyCode	EndDate
49168870	17000.00	CAD	2024-02-12 00:00:00.000
543787929	720.25	USD	2024-11-23 00:00:00.000

The screenshot shows the Object Explorer on the left with various database objects. A new view named 'Login_History' is selected. The main pane displays the T-SQL code for creating the view:

```
CREATE VIEW User_Last_Login AS
SELECT Login_History.FirstName, Login_History.LastName, Login_History.Username,
MAX(Login_History.LoginTime) as Last_login
FROM (
    SELECT u.FirstName, u.LastName, u.Username, lh.LoginTime
    FROM dbo.[User] u, dbo.LoginHistory lh
    WHERE u.UserId = lh.UserId) as Login_History
GROUP BY Login_History.FirstName, Login_History.LastName, Login_History.Username
Having MAX(Login_History.LoginTime) <'01-01-2019'
```

The results pane below shows three rows of data from the 'User' and 'LoginHistory' tables, grouped by user and ordered by login time.

FirstName	LastName	Username	Last_login
Clinton	T Hunt	needyourblood	2009-03-16 19:02:28.000
Dianne	J Wiseman	rya	2009-03-16 20:09:28.000
Sue	Veilleux	adran2005	2018-01-23 10:39:23.000

- ❖ User details who have not login to their account after 2019

Views Cont.

❖ All user accounts details

The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer pane displays database objects like tables (dbo.Account, dbo.AccountType, etc.) and views (dbo.Views, dbo.Account_End, dbo.User_Account_Details, dbo.User_Last_Login). The main window contains a query editor with the following SQL code:

```
CREATE VIEW User_Account_Details AS
SELECT [User].FirstName, [User].LastName, AccountType.Description, Account.AccountNumber,
Branch.BranchId, Branch.BranchName, City.City
FROM Account INNER JOIN
AccountType ON Account.AccountTypeDid = AccountType.AccountTypeDid INNER JOIN
Branch ON Account.BranchDid = Branch.BranchDid INNER JOIN
City ON Branch.CityDid = City.CityDid INNER JOIN
[User] ON Account.UserId = [User].UserId
```

Below the code, the Results tab shows a grid of data:

	FirstName	LastName	Description	AccountNumber	BranchId	BranchName	City
1	Quan	Bui	Checking Account	524248096	TB_HS	TB Honrby St	Vancouver
2	Linda	Pyle	Checking Account	491688870	TB_HS	TB Honrby St	Vancouver
3	Clinton	T Hunt	Investment Account	455603546	TB_BW	TB Broadway	Vancouver
4	Quan	Bui	Investment Account	531127369	TB_HS	TB Honrby St	Vancouver
5	Robert	M Guzman	Credit Account	543787929	TB_KS	TB King St	Toronto

Transaction

- ◆ Account is getting inactive when the account end date is updated

The screenshot shows the SSMS interface. On the left, the Object Explorer displays the database structure of 'BankingDB', including tempdb, Database Snapshots, Tables (System Tables, FileTables, External Tables, Graph Tables), and severaldbo tables (Account, AccountType, Bank, Branch, City, Currency, LoginHistory, Transaction). The main window shows a T-SQL script for creating a stored procedure named 'Account_Inactivation'. The script uses BEGIN TRANSACTION, UPDATE statements to set EndDate and IsActive, and COMMIT TRANSACTION. It then EXECutes the procedure with parameters: @endDate = '2021-05-21' and @AccountNumber = 531127369. The status bar at the bottom indicates the completion time as 2021-08-12T03:19:49.6249596-04:00.

```
CREATE PROCEDURE Account_Inactivation @endDate datetime, @AccountNumber NVARCHAR(20)
AS
BEGIN
    BEGIN TRANSACTION
    Update dbo.Account SET EndDate = @endDate WHERE AccountNumber = @AccountNumber
    Update dbo.Account SET IsActive = 0 where AccountNumber = @AccountNumber
    COMMIT TRANSACTION
END
EXEC Account_Inactivation @endDate = '2021-05-21', @AccountNumber = 531127369
```

Sub Queries

- ❖ Users whose credit points are greater than 500

The screenshot shows the Object Explorer on the left with the BankingDB database selected. In the center, two query panes are open: SQLQuery5.sql and SQLQuery4.sql. The SQLQuery5.sql pane contains a query that declares a variable @CredPointAmount, sets it to 500, and then selects user information where the user's credit points are greater than the declared amount. The SQLQuery4.sql pane shows the results of this query, which return four users: Quan Bui, Linda Pyle, Sue Veilleux, and Michael M Buzbee.

```
DECLARE @CredPointAmount int;
SET @CredPointAmount = 500;
SELECT u.FirstName, u.LastName, u.DOB, u.Email, u.MobileNumber
from dbo.[User] u
WHERE u.UserId in (
    SELECT ucp.UserId
    from dbo.UserCreditPoints ucp
    where ucp.Points > @CredPointAmount);
```

FirstName	LastName	DOB	Email	MobileNumber
Quan	Bui	1997-10-27	buiquan2710@gmail.com	4375335833
Linda	Pyle	1991-06-07	6o7q2hb2oj9@temporary-mail.net	9055457304
Sue	Veilleux	1981-09-22	guf1151ag@temporary-mail.net	6478922910
Michael	M Buzbee	1980-06-05	ap21125qr1g@temporary-mail.net	9058653383

The screenshot shows the Object Explorer on the left with the BankingDB database selected. In the center, two query panes are open: SQLQuery6.sql and SQLQuery3.sql. The SQLQuery6.sql pane contains a query that declares a variable @BranchName, sets it to 'TB King st', and then selects user information where the user's city ID matches the city ID of a branch whose name is the same as the declared branch name. The SQLQuery3.sql pane shows the results of this query, which return two users: Linda Pyle and Sue Veilleux.

```
DECLARE @BranchName NVARCHAR(50)
SET @BranchName = 'TB King st'
SELECT u.FirstName, u.LastName, u.Email, u.MobileNumber
FROM dbo.[User] u
WHERE u.CityId IN (
    SELECT c.CityId
    from dbo.Branch b, dbo.City c
    WHERE b.CityId = c.CityId
    AND b.BranchName= @BranchName)
```

FirstName	LastName	Email	MobileNumber
Linda	Pyle	6o7q2hb2oj9@temporary-mail.net	9055457304
Sue	Veilleux	guf1151ag@temporary-mail.net	6478922910

- ❖ Users who are in the same city of the give branch TB King St

Sub Queries Cont.

❖ All the savings accounts details

The screenshot shows the Object Explorer on the left and a query editor on the right. The query editor contains the following T-SQL code:

```
DECLARE @AccountType VARCHAR(20)
SET @AccountType = 'Saving Account'
SELECT * from dbo.Account
where AccountDid IN(
    SELECT AccountDid FROM dbo.AccountType
    WHERE Description = @AccountType)
```

The results pane below the query shows a table with 5 rows of account details:

	AccountDid	AccountNumber	PreviousBalance	CurrentBalance	CurrencyCode	OpenDate	EndDate	IsActive	BranchDid	UserDid	AccountTypeDid
1	1	524248096	2000.00	1500.00	CAD	2021-01-11 00:00:00.000	2025-01-11 00:00:00.000	1	7	2	2
2	3	491688870	17000.00	17000.00	CAD	2020-02-12 00:00:00.000	2024-02-12 00:00:00.000	1	7	9	2
3	4	455603546	2920.72	3200.72	USD	2008-12-09 00:00:00.000	2025-12-09 00:00:00.000	0	9	13	3
4	5	531127369	10000.00	11000.92	CAD	2021-01-11 00:00:00.000	2021-05-21 00:00:00.000	0	7	2	3
5	6	543787929	729.25	720.25	USD	2019-11-23 00:00:00.000	2024-11-23 00:00:00.000	1	12	15	4

Explanation and usage of
predicate logic

Predicate Logic

- ❖ Database tables represent lots of facts and logics that exist between the tables, and database queries produce result sets based on query predicates. A predicate defines a logical condition being applied to rows in a table. The common logical conditions with two values (true, false) are extended in the SQL language by a third value (unknown). SQL Predicates are found on the tail end of clauses, functions, and SQL expressions in existing query statements.

Usage of Predicate logics in Database

- Filtering data using WHERE or HAVING clauses
- Handling the flow with conditional logic using IF or CASE expressions
- Joining tables using ON filter
- Enforcing entity, domain and referential integrity (Example: CHECK, FOREIGN KEY constraints)

Stored Procedures, Function and Triggers

Stored Procedure

- Get the user details whose role is admin

The screenshot shows the Object Explorer on the left with a connection to 'LAPTOP-2CC0J3B6'. In the center, the 'SQLQuery6.sql' window contains the following code:

```
create procedure GetAdminUsers @userRole NVARCHAR(20)
as
begin
    set nocount on
    select * from dbo.[User] where UserRole = @userRole
end
go
exec GetAdminUsers @userRole = 'admin'
```

The 'Results' tab shows the output:

UserDid	FirstName	LastName	DOB	Address	Username	Password	Email	MobileNumber	SinNumber	CreateDate	UserR
1	Quan	Bui	1997-10-27	960 William St, Thunder Bay	buiquan2710	123	buiquan2710@gmail.com	4375335833	123456789	2021-01-11 00:00:00.000	adm
2	Linda	Pyle	1991-06-07	1795 Parkdale Avenue	dale	o@6Hoh4ebie	6o7q2hb2oj9@temporary-mail.net	9055457304	437748445	2021-01-11 00:00:00.000	adm
3	Clinton	T Hunt	1971-07-23	169 Harvest Moon Dr	needyourblood	faa1Ooc4uu9	grv0wgkn5jd@temporary-mail.net	9059439653	502989528	2020-12-29 00:00:00.000	adm

The screenshot shows the Object Explorer on the left with a connection to 'LAPTOP-2CC0J3B6'. In the center, the 'SQLQuery6.sql' window contains the following code:

```
create procedure GetTransactions_WithUserId (@numRaw as int, @userDid as int )
as
begin
select TOP(@numRaw)t.TransactionDid,t.TransactionDetails,c.CurrencyCode,
t.TransactionAmount,t.CreatedDate
from dbo.[Transaction] t, dbo.Currency c
where t.CurrencyDid=c.CurrencyDID
AND UserId = @userDid
end
go
exec GetTransactions_WithUserId @numRaw =100, @userDid =2
```

The 'Results' tab shows the output:

TransactionDid	TransactionDetails	CurrencyCode	TransactionAmount	CreatedDate
1	Withdraw \$500	CAD	500.00	2021-08-07 00:00:00.000
2	Walmart Store #3124 Thunder Bay	CAD	119.24	2020-03-21 00:00:00.000
3	Wal-mart Canada Corp	CAD	539.52	2019-04-04 00:00:00.000

- Get the top 100 transaction details for user whose userdid=2

Stored Procedure Cont.

- ❖ All the branch details which are in Ontario province

The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer displays the database structure of 'BankingDB', including tables like Branch, City, and Account. In the center, the 'SQLQuery6.sql' window contains a stored procedure definition:

```
create procedure GetBranch_Province @Province NVARCHAR(20)
as
begin
set nocount on
select * from dbo.[Branch] inner join (select CityDid, City, Province, Country
from dbo.[City]) as CityTable on [Branch].CityDid=CityTable.CityDid where CityTable.Province=@Province;
end
exec GetBranch_Province 'ON'
```

The 'Results' tab on the right shows the output of the stored procedure, listing five branches located in the Ontario province:

BranchDid	BranchId	BranchName	Address	PhoneNumber	CityDid	Bankdid	CityDid	City	Province	Country
1	12	TB_KS	TB King St 40 King St W, Toronto, ON M5H 1H1	4168666430	2	11	2	Toronto	ON	CAN
2	13	TB_BS	TB Bay St 392 Bay St., Toronto, ON M5H 3K5	4168665700	2	11	2	Toronto	ON	CAN
3	14	TB_HS	TB Hewitson St 745 Hewitson St, Thunder Bay, ON P7B 6B5	8076235626	1	11	1	Thunder Bay	ON	CAN
4	15	TB_AW	TB Arthur W 1195 Arthur St W, Thunder Bay, ON P7E 6E2	8076245100	1	11	1	Thunder Bay	ON	CAN
5	16	TB_RR	TB Red River 225 Red River Rd, Thunder Bay, ON P7B 1A7	8073435600	1	11	1	Thunder Bay	ON	CAN

Functions

The screenshot shows the Object Explorer on the left with the 'BankingDB' database selected. In the center, a query window displays the creation of a function named 'dbo.Approve_Or_Disapprove'. The function takes an integer parameter '@point' and returns a character string of length 20. It uses an IF statement to set the return value based on the input point value. A SELECT statement is included to demonstrate the function's usage, selecting from the 'UserCreditPoints' table and applying the function to the 'Points' column.

```
create function dbo.Approve_Or_Disapprove(
    @point int
)
returns char(20) as
begin
    declare @returnValue char(20);
    if (@point >= 500) set @returnValue= 'Approved';
    if (@point < 500) set @returnValue= 'Disapproved';
    return @returnValue
end;

select *, dbo.Approve_Or_Disapprove(dbo.[UserCreditPoints].Points) as 'Ready To Extend Credit Limit'
from dbo.[UserCreditPoints]
```

The 'Results' tab shows the output of the query:

	UserCreditPointId	UserDid	Points	CreditLimit	CurrencyDid	Ready To Extend Credit Limit
1	1	2	689	500.00	1	Approved
2	3	9	1200	2000.00	1	Approved
3	4	10	524	500.00	3	Approved
4	5	11	246	36000000.00	2	Disapproved
5	6	12	798	1000.00	1	Approved

- ❖ check the status of accounts

The screenshot shows the Object Explorer on the left with the 'BankingDB' database selected. In the center, a query window displays the creation of a function named 'dbo.Check_Status_Account'. The function takes an integer parameter '@status' and returns a table. It uses a SELECT statement with a WHERE clause to filter accounts where the 'IsActive' column matches the input status.

```
create function dbo.Check_Status_Account (@status int)
returns table as
return
select *
from dbo.[Account]
where IsActive = @status

select * from dbo.Check_Status_Account(0);
```

The 'Results' tab shows the output of the query:

	AccountDid	AccountNumber	PreviousBalance	CurrentBalance	CurrencyCode	OpenDate	EndDate	IsActive	BranchDid	UserDid	AccountTypeDid
1	4	455603546	2920.72	3200.72	USD	2008-12-09 00:00:00.000	2025-12-09 00:00:00.000	0	9	13	3
2	5	531127369	10000.00	11000.92	CAD	2021-01-11 00:00:00.000	2021-05-21 00:00:00.000	0	7	2	3

- ❖ check if user is allowed to increase the credit limit

Functions Cont.

- ❖ To check the outstanding balance of accounts

The screenshot shows the Microsoft SQL Server Management Studio (SSMS) interface. On the left, the Object Explorer pane displays a tree view of database objects under the 'Programmability' category, including 'Stored Procedures'. In the center, a query editor window contains T-SQL code for creating a function named 'dbo.Outstanding_Balance'. The function takes two parameters: '@previous int' and '@current int', and returns a numeric value with 38 digits and 6 decimal places. It calculates the difference between the current and previous balances. Below the code, a 'Results' tab is open, showing a table with five rows of account data. The columns are: AccountDid, AccountNumber, PreviousBalance, CurrentBalance, CurrencyCode, OpenDate, EndDate, IsActive, BranchDid, UserId, AccountTypeDid, and Outstanding Balance. The data shows various account details with their corresponding previous and current balances, and the calculated outstanding balance.

	AccountDid	AccountNumber	PreviousBalance	CurrentBalance	CurrencyCode	OpenDate	EndDate	IsActive	BranchDid	UserId	AccountTypeDid	Outstanding Balance
1	1	524248096	2000.00	1500.00	CAD	2021-01-11 00:00:00.000	2025-01-11 00:00:00.000	1	7	2	2	-500.000000
2	3	491688870	17000.00	17000.00	CAD	2020-02-12 00:00:00.000	2024-02-12 00:00:00.000	1	7	9	2	0.000000
3	4	455603546	2920.72	3200.72	USD	2008-12-09 00:00:00.000	2025-12-09 00:00:00.000	0	9	13	3	280.000000
4	5	531127369	10000.00	11000.92	CAD	2021-01-11 00:00:00.000	2021-05-21 00:00:00.000	0	7	2	3	1001.000000
5	6	543787929	729.25	720.25	USD	2019-11-23 00:00:00.000	2024-11-23 00:00:00.000	1	12	15	4	-9.000000

Triggers

- ❖ If the inserted account's expired date is equal to the date, the account will be disable
- ❖ If the inserted currency doesn't come with the currency code, trigger will be called to create a currency code substring from description

The screenshot shows the SQL Server Object Explorer with two tabs open: 'SQLQuery6.sql - LA_COJ3B6\iresha (53)*' and 'SQLQuery3.sql - LA_COJ3B6\iresha (54)*'. The 'SQLQuery6.sql' tab contains the following T-SQL code:

```
drop trigger if exists t_account_insert;
go
create trigger t_account_insert on dbo.[Account] instead of insert
as begin
declare
    @accountNumber nvarchar(128),
    @previousBalance float,
    @currentBalance float,
    @currencyCode nvarchar(3),
    @openDate date,
    @endDate date,
    @isActive int,
    @branchId int,
    @userId int,
    @accountTypeDid int

select @accountNumber = AccountNumber, @previousBalance = PreviousBalance,
@currentBalance = CurrentBalance, @currencyCode = CurrencyCode, @openDate = OpenDate, @endDate = EndDate,
@isActive = IsActive, @branchId = BranchId, @userId = UserId, @accountTypeDid = AccountTypeDid from inserted;
if CONVERT(date, @endDate) = CONVERT(date, getdate()) set @isActive = 0;
insert into dbo.[Account](AccountNumber, PreviousBalance, CurrentBalance,
CurrencyCode, OpenDate, EndDate, IsActive, BranchId, UserId, AccountTypeDid)
values (@accountNumber, @previousBalance, @currentBalance, @currencyCode,
@openDate, @endDate, @isActive, @branchId, @userId, @accountTypeDid);
end
```

The 'Messages' pane at the bottom shows: 'Commands completed successfully.' and 'Completion time: 2021-08-12T10:21:56.6448197-04:00'.

The screenshot shows the SQL Server Object Explorer with two tabs open: 'SQLQuery6.sql - LA_COJ3B6\iresha (53)*' and 'SQLQuery3.sql - LA_COJ3B6\iresha (54)*'. The 'SQLQuery3.sql' tab contains the following T-SQL code:

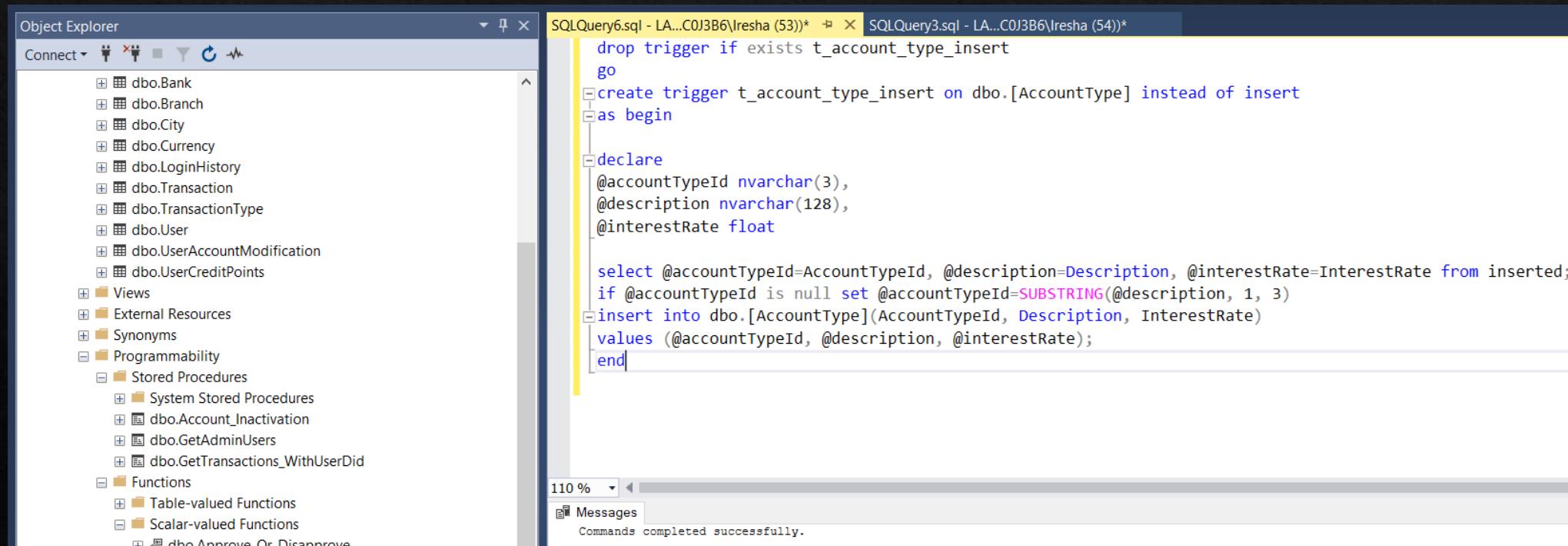
```
drop trigger if exists t_currency_insert
go
create trigger t_currency_insert on dbo.[Currency] instead of insert
as begin
declare
    @currencyCode nvarchar(3),
    @currencyDescription nvarchar(50)

select @currencyCode = CurrencyCode, @currencyDescription = CurrencyDescription from inserted;
if @currencyCode is null set @currencyCode = SUBSTRING(@currencyDescription, 1, 3)
insert into dbo.[Currency](CurrencyCode, CurrencyDescription)
values (@currencyCode, @currencyDescription);
end
```

The 'Messages' pane at the bottom shows: 'Commands completed successfully.' and 'Completion time: 2021-08-12T10:21:56.6448197-04:00'.

Triggers Cont.

- ◊ Insert a new account type without account id will trigger the trigger and automatically create an account id substring from the account description



The screenshot shows the SQL Server Management Studio interface. On the left, the Object Explorer pane displays a tree view of database objects under the 'dbo' schema, including tables like Bank, Branch, City, Currency, LoginHistory, Transaction, TransactionType, User, and various stored procedures and functions. In the center, the main window contains a T-SQL script for creating an instead-of trigger named 't_account_type_insert' on the 'AccountType' table. The script uses a cursor to iterate over the inserted rows, extracting the first three characters of the 'Description' column to set as the 'AccountTypeId'. It then inserts the row into the 'AccountType' table with the generated 'AccountTypeId' and the original 'Description' and 'InterestRate' values. The status bar at the bottom indicates 'Commands completed successfully.'

```
drop trigger if exists t_account_type_insert
go
create trigger t_account_type_insert on dbo.[AccountType] instead of insert
as begin

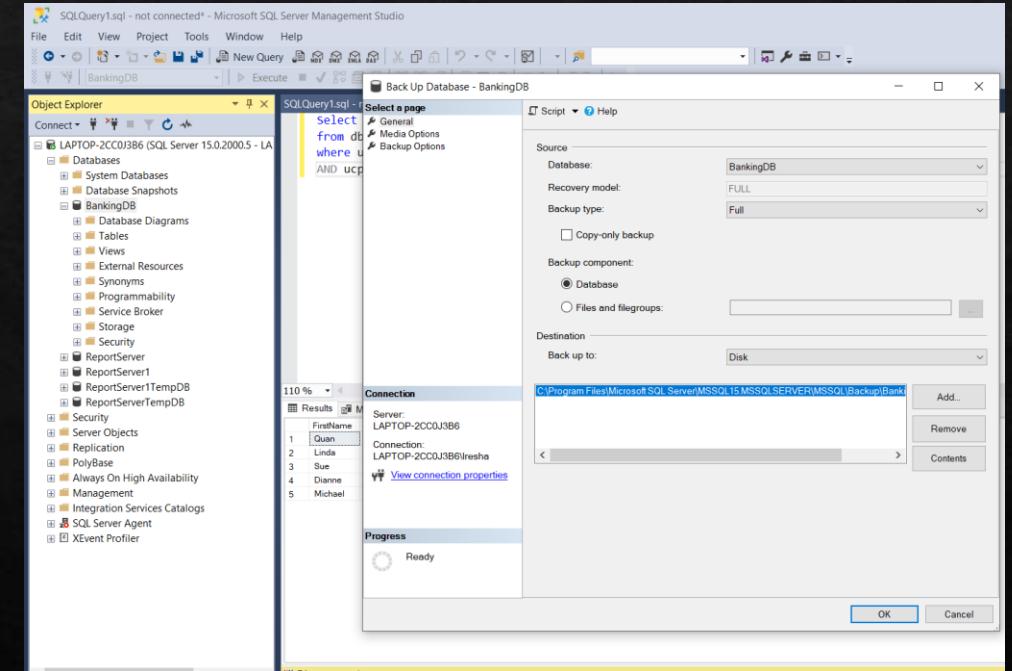
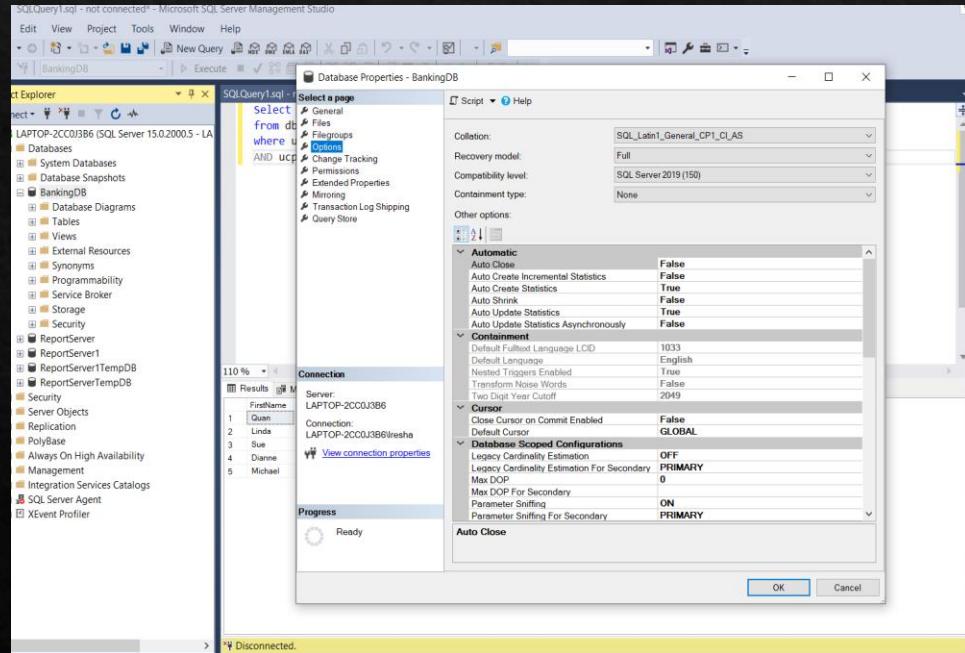
declare
@accountTypeId nvarchar(3),
@description nvarchar(128),
@interestRate float

select @accountTypeId=AccountTypeId, @description=Description, @interestRate=InterestRate from inserted;
if @accountTypeId is null set @accountTypeId=SUBSTRING(@description, 1, 3)
insert into dbo.[AccountType](AccountTypeId, Description, InterestRate)
values (@accountTypeId, @description, @interestRate);
end
```

Backup & Logs

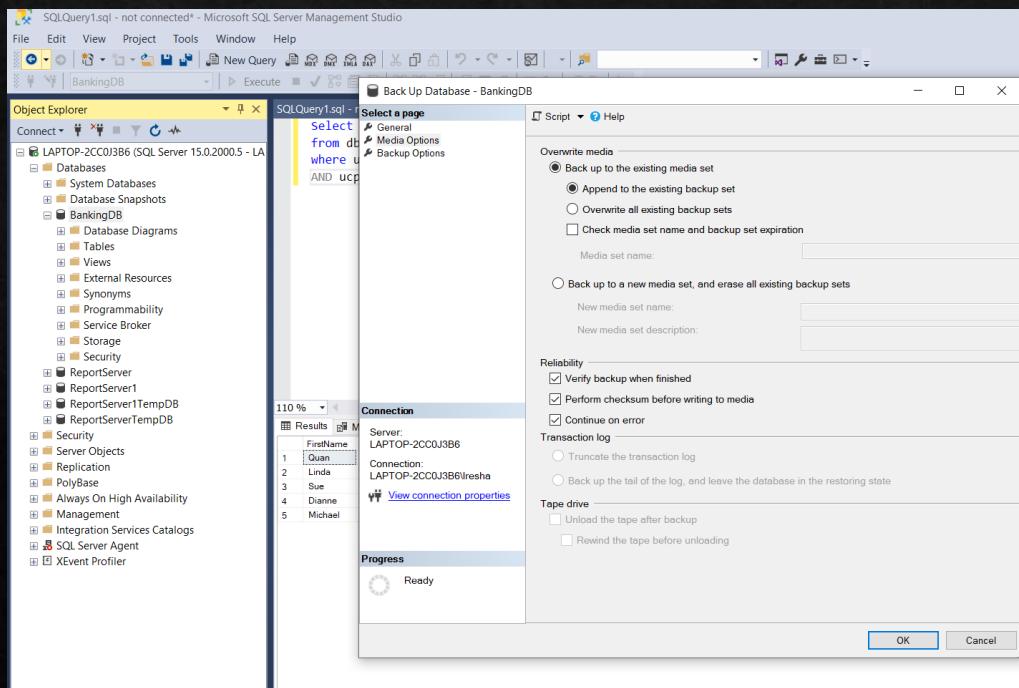
Backup & Logs

- ❖ We have selected the Recovery model as Full back up for our database
- ❖ Then we have selected back up type as Full backup and back up component as Database

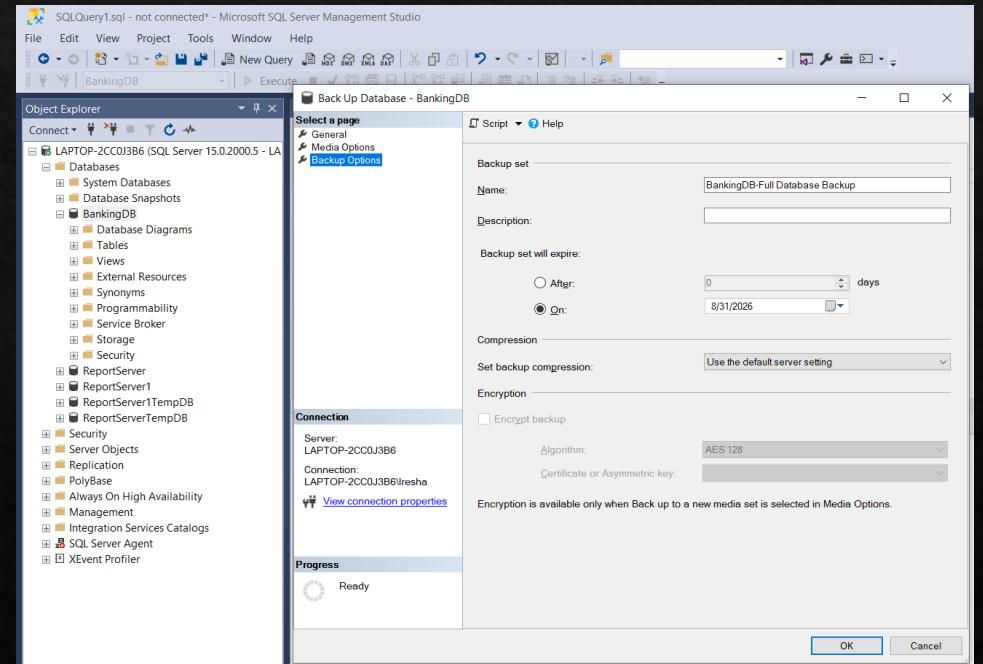


Backup & Logs

- ❖ We have selected the back up to be appended to the existing media and selected all Reliability option to ensure the reliability of the back up

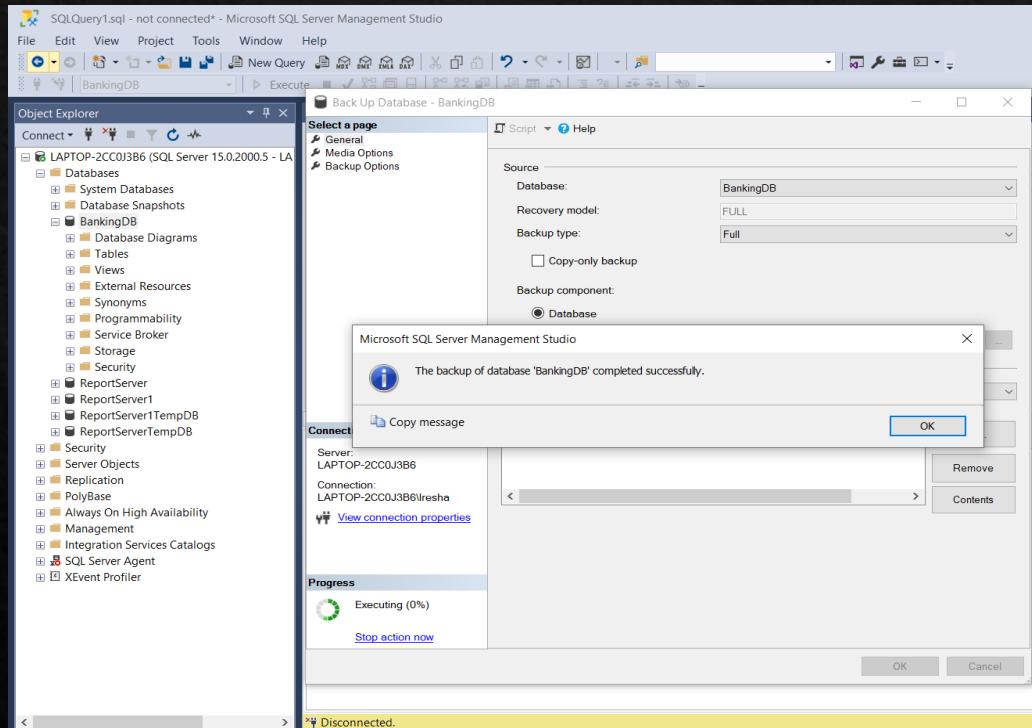


- ❖ In the back up option we have selected back up expiration date as we want to keep the back only for 5 years.



Backup & Logs

❖ Database back up completion



❖ Database Backup report

The screenshot shows the 'Backup and Restore Events' report for the 'BankingDB' database on 'LAPTOP-2CC0J3B6' at 8/11/2021 7:11:59 PM. The report provides historical data about backup and restore actions performed on the database. It includes sections for 'Average Time Taken For Backup Operations', 'Successful Backup Operations', 'Backup Operation Errors', and 'Successful Restore Operations'. The 'Successful Backup Operations' table lists one entry:

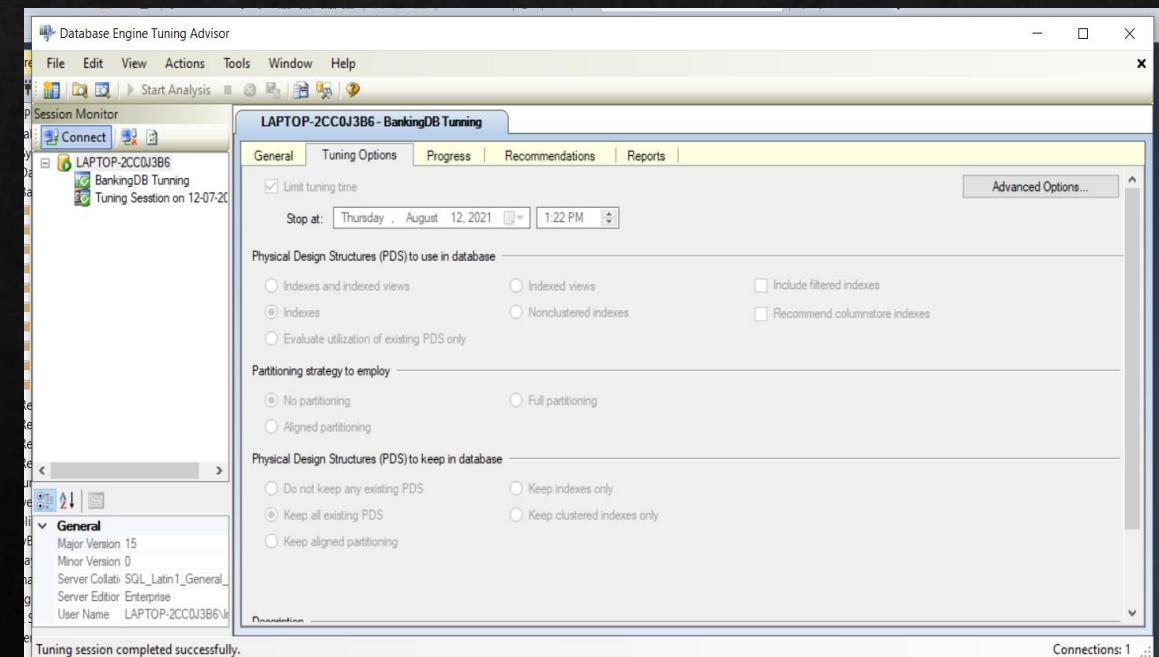
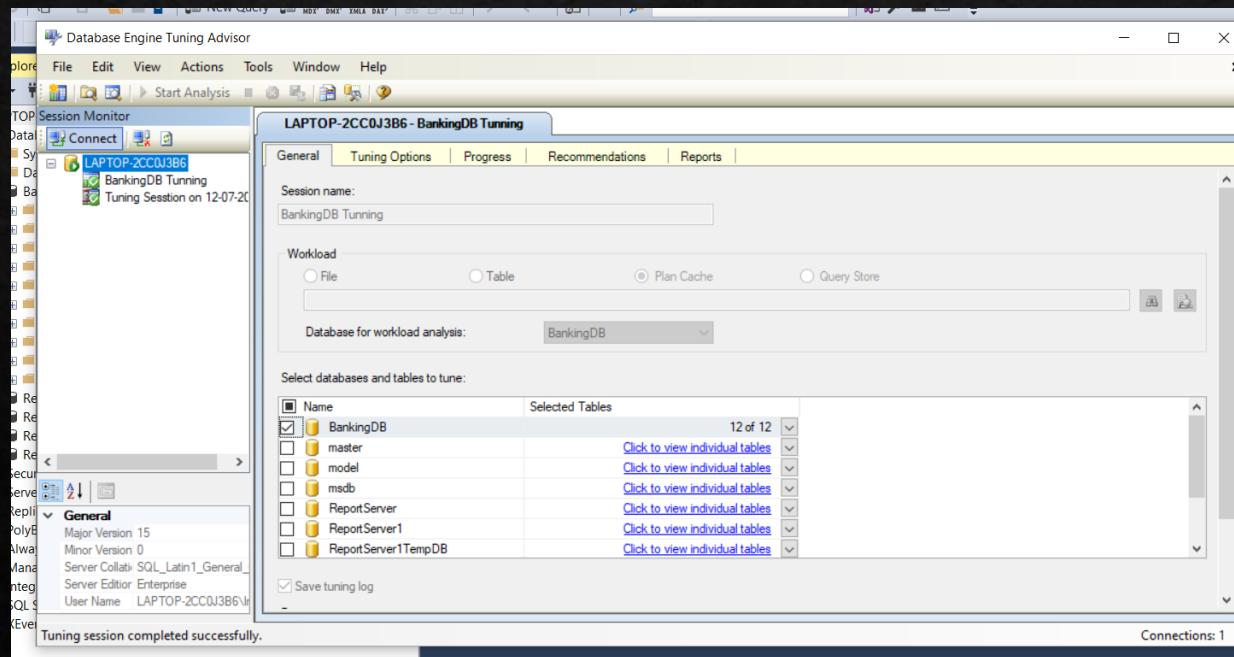
Start Time	Duration (minutes)	Backup Type	Backup Size	Backup Name	Device Type	User Name	Recovery Model	Differential Base LSN	Last LSN
8/11/2021 6:11:05 AM	0.00	Database	3.77 MB	BankingDB-Full Database Backup	Disk (temporary)	LAPTOP-2CC0J3B6\resha	FULL	Not applicable	3700000193600001

Database Tuning

- ❖ Database tuning aims to maximize use of system resources to perform work as efficiently and rapidly as possible. Most systems are designed to manage their use of system resources, but there is still much room to improve their efficiency by customizing their settings and configuration for the database and the DBMS.

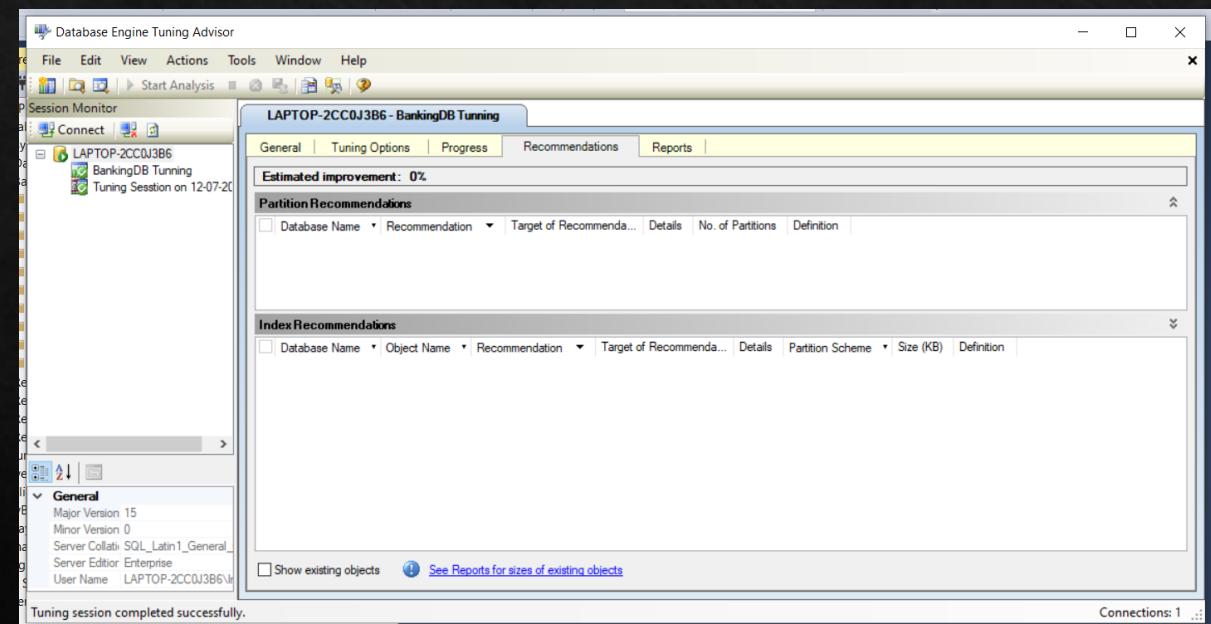
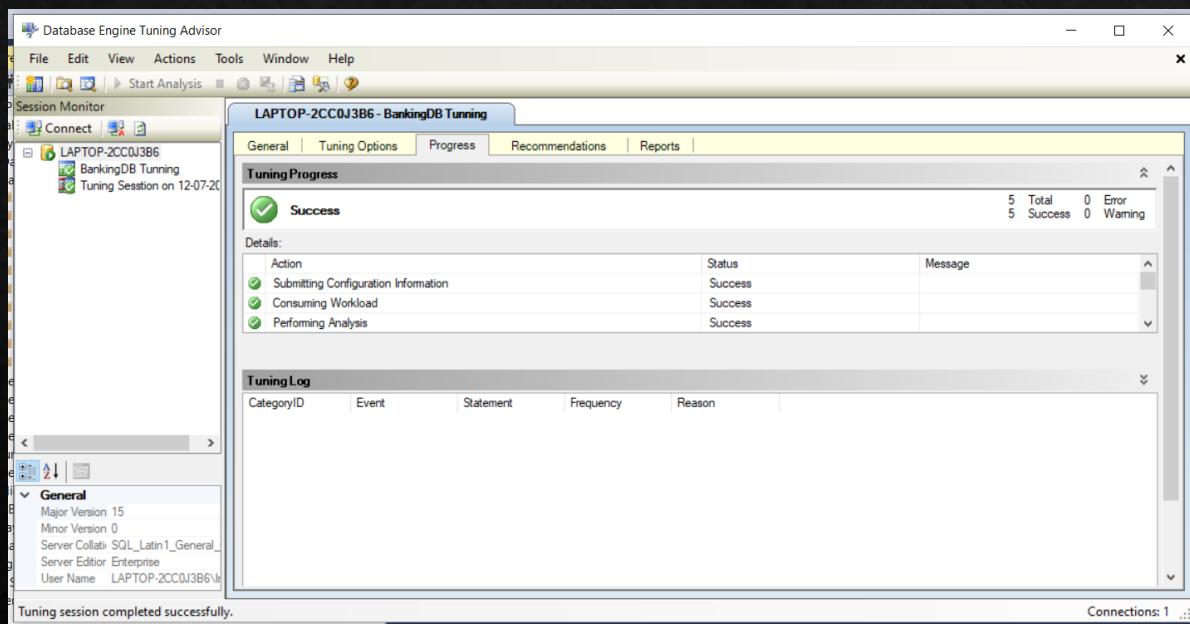
Database Tuning Cont.

- ❖ We have selected Plan Cache tuning option to tune our database.



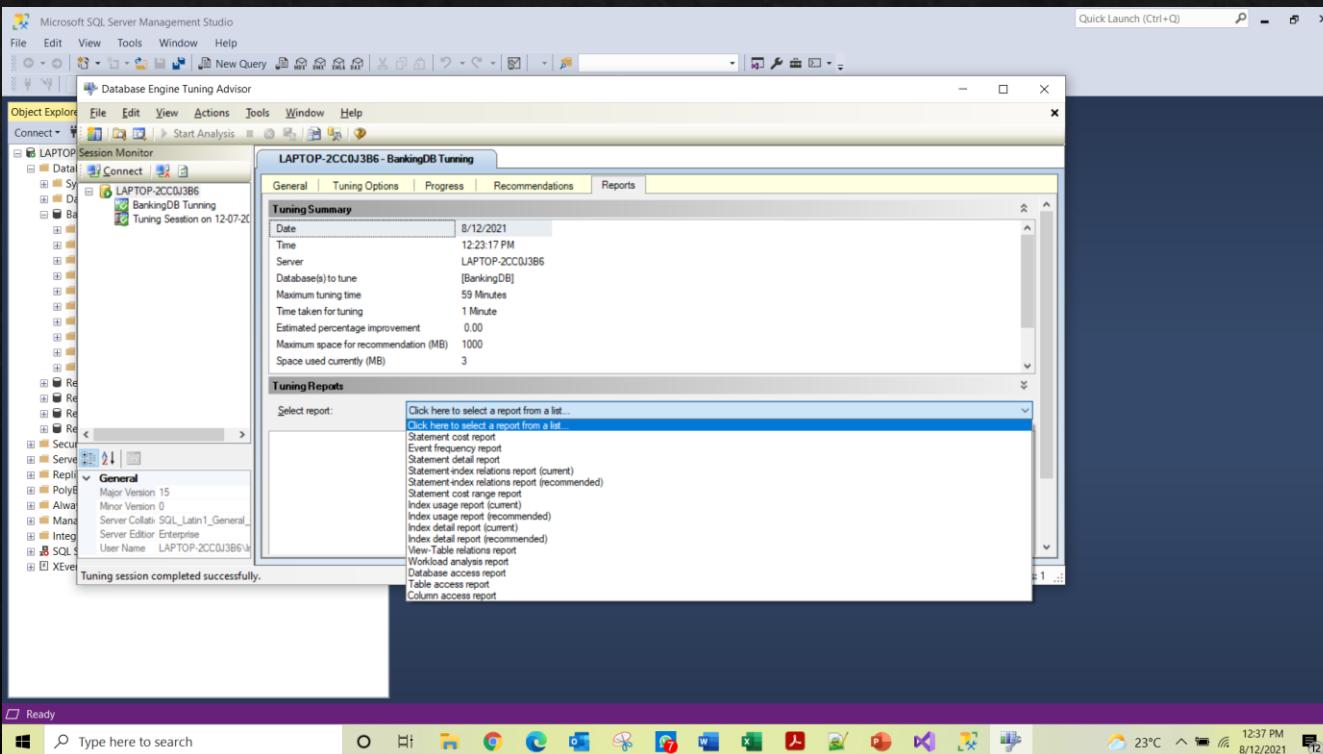
Database Tuning Cont.

- ❖ Database Tuning was successful
- ❖ There were no recommendations to be performed



Database Tuning Conti.

❖ Tunning summary



Thank You.