SQL PROJECT
BANK DATABASE SYSTEM

MAYA BABU

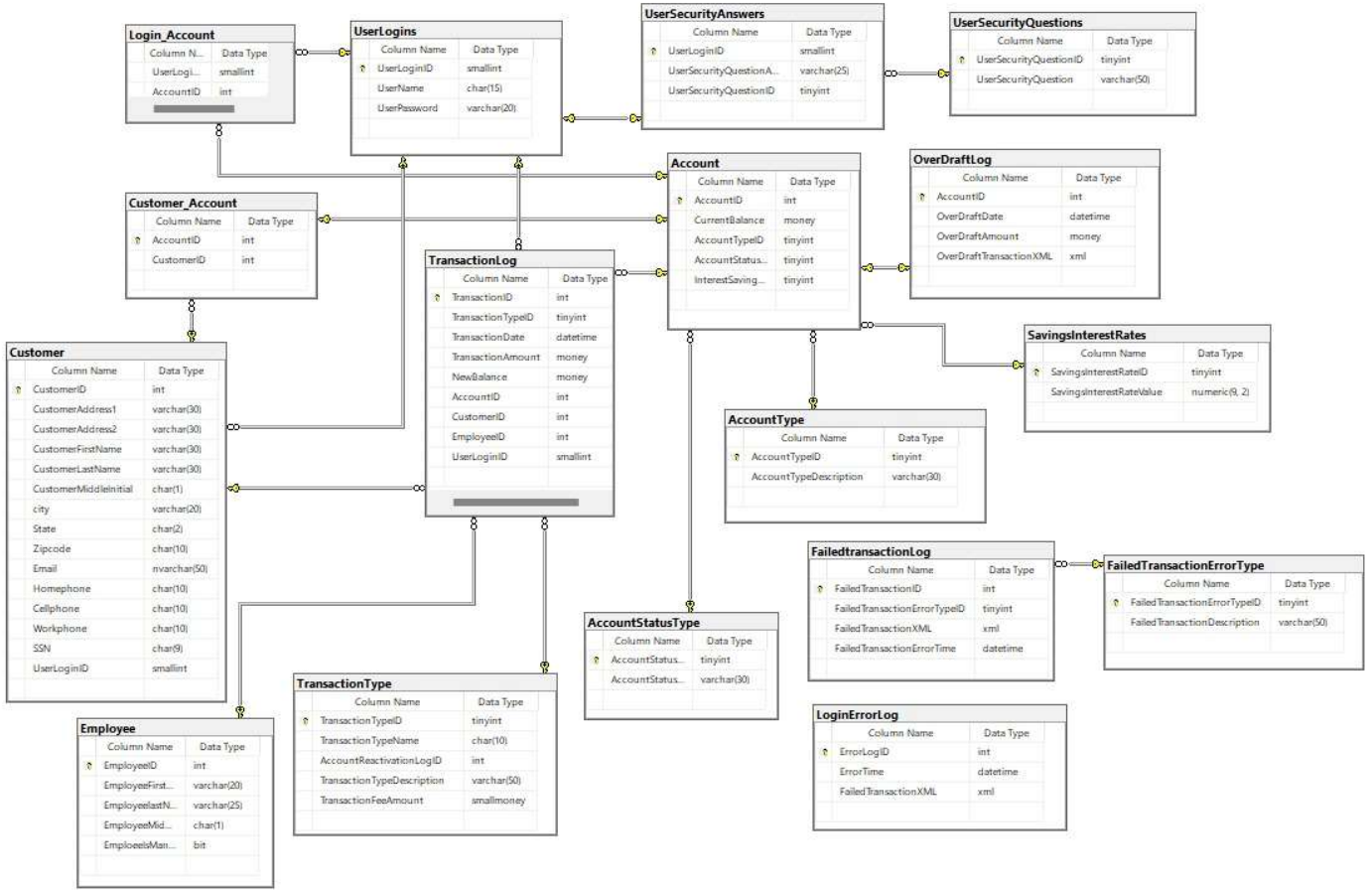
Project Objective

- Creating Database Bank
- Creating Tables
- Inserting values
- Writing Queries
- Creating Views
- Creating Store Procedures
- Retrieving Data

VIEWS

- Views are virtual tables that hold data from one or more tables.
- Views are mainly used for security purposes in databases.

Entity-Relationship Diagram



1. Create a view to get all customers with checking accounts from ON province.

```
Create view ONChequing
AS
    SELECT c.CustomerId, c.CustomerFirstName, c.CustomerLastName, c.State, at.AccountTypeDescription
FROM Customer c
JOIN Customer_Account ca
ON ca.CustomerID = c.customerid
join Account a
on a.AccountID = ca.AccountID
join AccountType at
on at.AccountTypeID = a.AccountTypeID
Where at.AccountTypeDescription = 'Chequing' and State = 'ON'

Select * from ONChequing
```

0 %

Results | Messages

CustomerId	CustomerFirstName	CustomerLastName	State	AccountTypeDescription
100001	Maya	Babu	ON	Chequing
100002	John	Samuel	ON	Chequing
100005	Sayak	Chandra	ON	Chequing

2. Create a view to get all customers with a total account balance (including interest rate) greater than 5000.

```
CREATE VIEW sp_Customers_With_Balance_Above_5000
AS
SELECT c.CustomerFirstName, SUM(a.CurrentBalance + (a.CurrentBalance * s.interestRatevalue)) AS TotalAccountBalance
From Customer c
join Customer_Account ca
ON ca.CustomerID = c.CustomerID
join Account a
ON a.AccountID = ca.AccountID
join SavingsInterestRates s
ON s.InterestSavingsRateID = a.InterestSavingsRateID
GROUP BY c.CustomerFirstName
HAVING SUM(a.CurrentBalance + (a.CurrentBalance * s.InterestRateValue)) >5000

select * from sp_Customers_With_Balance_Above_5000
```

0 %

Results | Messages

	CustomerFirstName	TotalAccountBalance
	Sam	6084.000000
	Sayak	10122.200000
	sony	8214.000000

Table- Customer_Account

	AccountID	CustomerID
1	1111	100004
2	1112	100001
3	1113	100002
4	1114	100005
5	1115	100003

Table-Account

	AccountID	CurrentBalance	AccountTypeID	AccountStatusTypeID	InterestSavingsRateID
1	1111	5200.00	1	1	102
2	1112	3400.00	2	2	101
3	1113	2860.00	2	1	104
4	1114	9460.00	2	1	105
5	1115	7400.00	2	2	101

Table-Customer Table

	CustomerID	CustomerAddress1	CustomerAddress2	CustomerFirstName	CustomerLastName	CustomerMiddleInitial	city	State	Zipcode	Email	Homephone	Cellphone	Workphone	SSN	UserLoginID
1	100001	Sunrise_Square	NULL	Maya	Babu	NULL	Scarborough	ON	M1B1R3	maya@gmail.com	NULL	1234567890	NULL	14728369	1001
2	100002	Shropshire_Drive	NULL	John	Samuel	G	Oshawa	ON	M1P1Y3	john@gmail.com	NULL	9876543210	NULL	123586974	1002
3	100003	parkway_ave	NULL	sony	sanjay	NULL	Kingston	NS	Z1YN7K	sony@gmail.com	NULL	6476476470	NULL	543256987	1003
4	100004	kitty_drive	NULL	Sam	Sung	H	Missisauga	ON	M8H7Y5	sam@gmail.com	NULL	4374374375	NULL	111222333	1004
5	100005	mersal	avenue	Sayak	Chandra	NULL	Barry	ON	J5O6G3	sai@gmail.com	NULL	6474376473	NULL	333222111	1005

3. Create a view to get counts of checking and savings accounts by customer.

```
Create view AccountsCount
as
select c.CustomerFirstName, at.AccountTypeDescription, count(*) as TotalAccountTypes
from Customer c
join Customer_Account ca
on ca.CustomerID = c.CustomerID
join Account a
on a.AccountID = ca.AccountID
join AccountType at
on at.AccountTypeID = a.AccountTypeID
group by c.CustomerFirstName, at.AccountTypeDescription

select * from AccountsCount
```

110 %

Results | Messages

	CustomerFirstName	AccountTypeDescription	TotalAccountTypes
1	John	Chequing	1
2	Maya	Chequing	1
3	Sayak	Chequing	1
4	sony	Chequing	1
5	Sam	Savings	1

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

```
create view UserLoginAndPassword
as
Select distinct ul.UserLoginID, ul.UserName, ul.UserPassword
from UserLogins ul
join Login_Account la
on ul.UserLoginID = la.UserLoginID
Join Account a
on a.AccountID = la.AccountID
where la.AccountID = '1112'

select * from UserLoginAndPassword
```

110 %

Results | Messages

	UserLoginID	UserName	UserPassword
1	1001	mayababu	maya@2000

Table- UserLogins

	UserLoginID	UserName	UserPassword
1	1001	mayababu	maya@2000
2	1002	subash123	abc1234
3	1003	kaith987	xyz123
4	1004	yaami21	sasdsdd
5	1005	xavier	123@abc

Table - Account

	AccountID	CurrentBalance	AccountTypeID	AccountStatusTypeID	InterestSavingsRateID
1	1111	5200.00	1	1	102
2	1112	3400.00	2	2	101
3	1113	2860.00	2	1	104
4	1114	9460.00	2	1	105
5	1115	7400.00	2	2	101

Table – Login_Account

	UserLoginID	AccountID
1	1003	1111
2	1002	1113
3	1001	1112
4	1004	1114
5	1005	1115

5. Create a view to get all customers' overdraft amounts.

```
create view CustomerOverDraft
as
select distinct c.CustomerFirstName, o.OverdraftAmount
from OverDraftLog o
join Customer_Account ca
on ca.AccountID = o.AccountID
join Customer c
on c.CustomerID = ca.CustomerID
go

select * from CustomerOverDraft
```

110 %

Results | Messages

	CustomerFirstName	OverdraftAmount
1	John	30.25
2	Maya	75.50
3	Sam	50.00
4	Sayak	100.75
5	sony	45.60

STORED PROCEDURE

- A stored procedure is a set of Structured Query Language (SQL) statements with an assigned name, which are stored in a relational database management system as a group, so it can be reused and shared by multiple programs

6. Create a stored procedure to add “User_” as a prefix to everyone’s login

```
create procedure AddUserPrefixtoLogin
as
begin
 update UserLogins
 set UserName ='User_' + UserName
 end

execute AddUserPrefixtoLogin

select * from UserLogins
```

110 %

Results | Messages

	UserLoginID	UserName	UserPassword
1	1001	User_mayababu	maya@2000
2	1002	User_subash123	abc1234
3	1003	User_kaith987	xyz123
4	1004	User_yaami21	sasdsdd
5	1005	User_xavier	123@abc

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

```
create proc spFullNameFromAccountId
            @AccountID int,
            @Fullname nvarchar(100) output
as
begin
  if (@AccountID in (select AccountID from Customer_Account))
        begin
                Select @FullName= c.CustomerFirstName+' '+c.CustomerMiddleInitial+' '+c.CustomerLastName
                from Customer c
                join Customer_Account ca
                on ca.CustomerID=c.CustomerID
                where ca.AccountID=@AccountID;
                set @Fullname= @FullName
        end
  else
        begin
                print 'There is no Customer with  Account Id= '+CONVERT(varchar(12), @AccountID )
        end
end

--Executing for invalid account id
Declare @FullName nvarchar(100)
exec spFullNameFromAccountId 1116, @FullName out
Print ' Full name is '+ @FullName
Declare @FullName nvarchar(100)
exec spFullNameFromAccountId 1112, @FullName out
Print ' Full name is '+ @FullName
```

```
110 %
Messages
There is no Customer with  Account Id= 1116


Completion time: 2023-10-30T00:35:15.2038550-04:00
```

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

```
CREATE PROCEDURE GetErrorLogsLast24Hours
AS
BEGIN
    SELECT ErrorLogID , ErrorTime, FailedtransactionXML
    FROM LoginErrorLog
    WHERE ErrorTime BETWEEN DATEADD(hh, -24, GETDATE()) AND GETDATE()
END;
Exec GetErrorLogsLast24Hours
```

110 %

Results | Messages

	ErrorLogID	ErrorTime	FailedtransactionXML
1	5	2023-10-29 12:10:10.000	<Error><UserLoginID>3</UserLoginID><Code>1</Code...
2	6	2023-10-29 09:10:10.000	<Error><UserLoginID>3</UserLoginID><Code>1</Code...

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

```
CREATE PROCEDURE UpdateAccountBalance
    @AccountId INT,
    @DepositAmount DECIMAL(10, 2)
AS
BEGIN
    DECLARE @CurrentBalance DECIMAL(10, 2);
        -- Get the current balance of the account
    SELECT @CurrentBalance = CurrentBalance
    FROM Account
    WHERE AccountId = @AccountId

    -- Update the current balance by adding the deposit amount
    SET @CurrentBalance = @CurrentBalance + @DepositAmount

    -- Update the CurrentBalance in the Accounts table
    UPDATE Account
    SET CurrentBalance = CurrentBalance+ @DepositAmount
    WHERE AccountId = @AccountId

    -- Return the updated CurrentBalance
    SELECT @AccountId as AccountID, @CurrentBalance AS UpdatedBalance;
END;

    declare @AccountID int
    declare @DepositAmount decimal(10,2)
    set @AccountID = 1111
    set @DepositAmount = 1000
exec UpdateAccountBalance @AccountID,@DepositAmount
```

110 %

Results | Messages

	AccountID	UpdatedBalance
1	1111	6200.00

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

```
CREATE PROCEDURE CBalanceAfterWithdrawal
    @AccountId INT,@withdrawal DECIMAL(10, 2)
AS
BEGIN
    DECLARE @CurrentBalance DECIMAL(10, 2);
        -- Get the current balance of the account
    SELECT @CurrentBalance = CurrentBalance
    FROM Account
    WHERE AccountId = @AccountId

    -- Update the current balance by deducting the withdrawal amount
    SET @CurrentBalance = @CurrentBalance - @withdrawal

    -- Update the CurrentBalance in the Accounts table
    UPDATE Account
    SET CurrentBalance = CurrentBalance - @withdrawal
    WHERE AccountId = @AccountId

    -- Return the updated CurrentBalance
    SELECT @AccountId as AccountID, @CurrentBalance AS UpdatedBalance;
END;

    declare @AccountID int
    declare @withdrawal decimal(10,2)
    set @AccountID = 1111
    set @withdrawal = 1000
exec UpdateAccountBalance @AccountID,@withdrawal
```

110 %

Results | Messages

	AccountID	UpdatedBalance
1	1111	7200.00

Thank You