Exam 2

CIS560 & CIS562

You will have until the end of class to complete this exam, allowing **approximately 50 minutes**. It is a closed-book exam, except for one page of notes. You should not have any other materials next to you. **Please put away all papers, phones, and watches**.

Please write your name and eID on each page in case the pages get separated. Also indicate in which course you are enrolled by circling the appropriate course in the upper right of each sheet.

You have two options for submitting your solutions:

1. Write your solution by hand on the following pages.
2. Typing them and submitting in Canvas. *Read carefully below.*

If typing your solution on a laptop, **you still need to turn in this paper exam** with your name and eID on each sheet, and the word "Canvas" on the first page. On your laptop, you may have no more than two applications running:

1) a browser with the assignment open in Canvas
2) an application for typing your solutions, limited to a basic text editor, Word, SQL Server Management Studio, or Azure Data Studio.

You must have no other applications running, and with each application, you may have only one tab or file open.

Submission

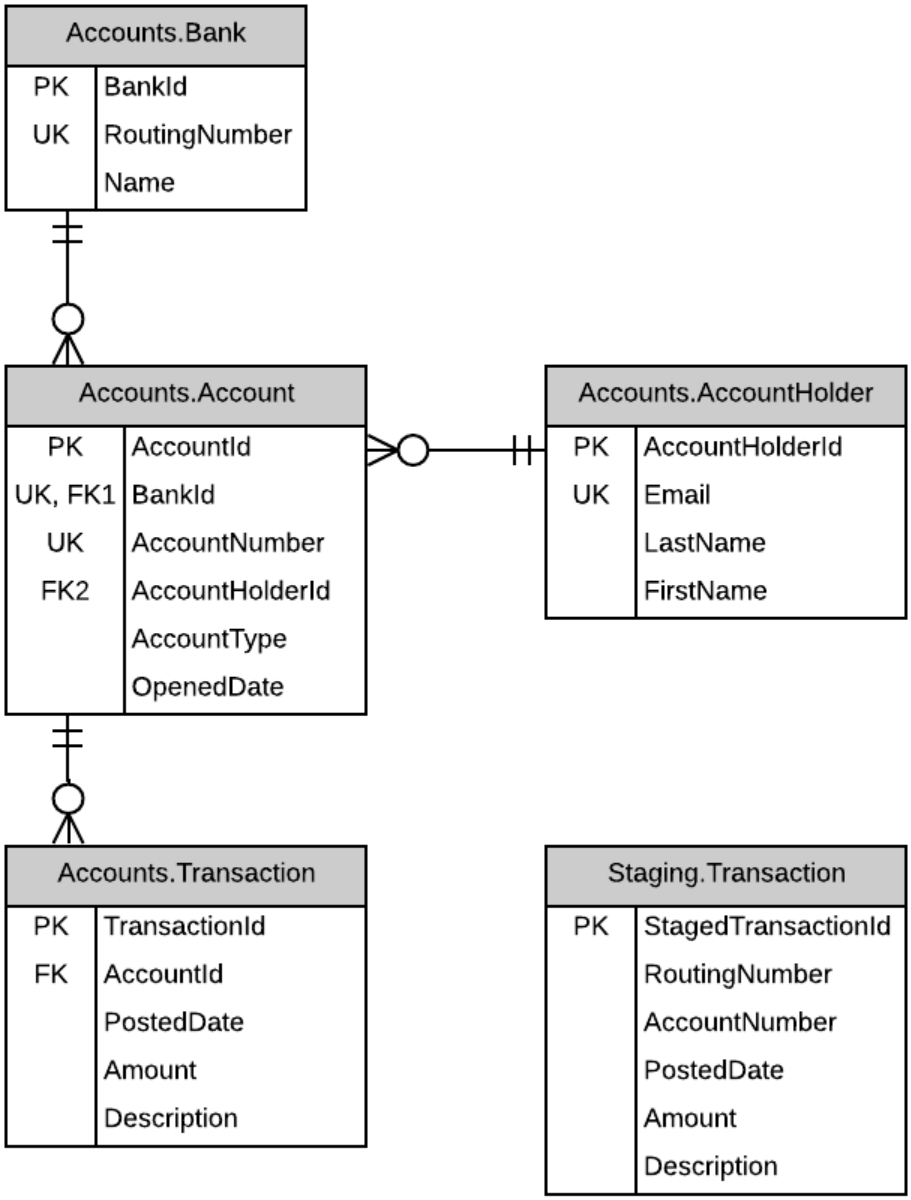
Before turning in your exam, double check the following:

- You wrote your name and eID, and circled the appropriate course, on the top of each page.
- **If using a laptop**:
    - You have all your solutions uploaded to the exam assignment in Canvas, each clearly indicating with which question it belongs using a comment or separate file. You may choose to submit your SQL solutions in one file and your other solutions in a txt or docx file.
    - You wrote "Canvas" on your first page.
    - *Regardless of whether submitting in Canvas, your paper exam must be turned in.*

NOTE: You may tear off this sheet for easier reference. You need not return this sheet.

Database Design

Some questions on this exam use tables from the below diagram. For the most part, the complete column specifications are unimportant for the purposes of this exam. All columns are defined as `NOT NULL`.



Question 1 – 25 points – Complete the implementation of the stored procedure below to provide account activity for a given account and date range. Along with the information for each transaction, we want a running balance resulting from each transaction's amount, as well as a display order to aid the caller in presenting the transactions in the same order as the balance was calculated.

Required Result Columns

PostedDate – The date as it appears in `PostedDate` of `Accounts.Transaction`.
Description – The transaction's description as it appears in `Description` of `Accounts.Transaction`.
Amount – The transaction's amount as it appears in `Amount` of `Accounts.Transaction`.
Balance – The new balance resulting from adding the transaction's amount to all prior amounts. Additionally, you will need to utilize the variable `@OpeningBalance` in the expression you use so that you can account for the activity before the beginning of the provided date range. In other words, the calculation for Balance will follow this pattern:
`@OpeningBalance + <Expression for running total of transaction amounts>.`
DisplayOrder – The order in which to display the transactions, with values `1..N`, where `N` is the number of transactions for `@AccountId` and the provided date range.

Implementation Requirements

- Both `Balance` and `DisplayOrder` should be computed using window functions.
- Both `Balance` and `DisplayOrder` columns should be computed using an order of:
  1. `PostedDate` in ascending order
  2. `Amount` such that credits (positive values) are applied and listed before debits (negative values)
  3. `TransactionId` in ascending order as a final tiebreaker.

On the next page is the starting point for the stored procedure. Complete its definition.

```
CREATE PROCEDURE Accounts.RetrieveActivity
   @AccountId INT,
   @FirstDate DATE,
   @LastDate DATE
AS

DECLARE @OpeningBalance NUMERIC(17,2) =
   (
      SELECT ISNULL(SUM(T.Amount), 0.00)
      FROM Accounts.Transaction T
      WHERE T.AccountId = @AccountId
         AND T.PostedDate < @FirstDate
   );

-- Provide the missing implementation here.

SELECT T.PostedDate, T.[Description], T.Amount,
   @OpeningBalance + SUM(T.Amount) OVER(
      ORDER BY T.PostedDate ASC, T.Amount DESC, T.TransactionId ASC
      ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS Balance,
   ROW_NUMBER() OVER(
      ORDER BY T.PostedDate ASC, T.Amount DESC, T.TransactionId ASC)
      AS DisplayOrder
FROM Accounts.[Transaction] T
WHERE T.AccountId = @AccountId
   AND T.PostedDate BETWEEN @FirstDate AND @LastDate;
```

Question 2 – 15 points – Write a SQL statement that will create the `Staging.Transaction` table. A more detailed diagram is shown here with the data types.

Staging.Transaction		
PK	StagedTransactionId	INT
	RoutingNumber	NUMERIC(9,0)
	AccountNumber	NUMERIC(12,0)
	PostedDate	DATE
	Amount	NUMERIC(17,2)
	Description	NVARCHAR(128)

Implementation Requirements

- You may assume the schema, `Staging`, for the table already exists.
- All columns are required and should not accept `NULL`.
- The column `StagedTransactionId` should automatically generate values as rows are inserted, starting with 1 and incrementing by 1. For simplicity, use a property on the `StagedTransactionId` column to achieve this desired behavior rather than using a sequence object.
- The column `Amount` should have a constraint to disallow a value of zero.

```
CREATE TABLE Staging.[Transaction]
(
   StagedTransactionId INT NOT NULL IDENTITY(1, 1) PRIMARY KEY,
   RoutingNumber NUMERIC(9,0) NOT NULL,
   AccountNumber NUMERIC(12,0) NOT NULL,
   PostedDate DATE NOT NULL,
   Amount NUMERIC(17,2) NOT NULL CHECK(Amount <> 0.00),
   [Description] NVARCHAR(128) NOT NULL
);
```

Question 3 – 20 points – Write an **INSERT** statement that will insert all rows from `Staging.Transaction` into `Accounts.Transaction`.

In order to identify the correct `AccountId` for each transaction in `Staging.Transaction`, `RoutingNumber` should be matched with `RoutingNumber` on `Accounts.Bank`, and `AccountNumber` should be matched with `AccountNumber` on `Accounts.Account`, just as the column names indicate.

Do not insert a value into the column `TransactionId` because it will automatically assign a value. This means that `StagedTransactionId` of `Staging.Transaction` will not be used, but all other columns of the staging table are necessary to properly insert into `Accounts.Transaction`.

```
INSERT Accounts.[Transaction](AccountId, PostedDate, Amount, [Description])
SELECT A.AccountId, T.PostedDate, T.[Amount], T.[Description]
FROM Staging.[Transaction] T
   INNER JOIN Accounts.Bank B ON B.RoutingNumber = T.RoutingNumber
   INNER JOIN Accounts.Account A ON A.BankId = B.BankId
      AND A.AccountNumber = T.AccountNumber;
```

Question 4 – 5 points – In writing SQL statements, which of the object types listed below can be used in the FROM element? Circle all that apply.

a) Stored Procedure

b) Inline Table-Valued Function

c) Table-Valued Function

d) Scalar-Valued Function

e) View

Question 5 – 5 points – In evaluating the functional dependencies and closures of a relation R, which of the following may be true of R if R is in Boyce-Codd Normal Form? Circle all that apply.

a) One or more of its functional dependencies may be a trivial functional dependency.

b) For one or more of its functional dependencies X → Y, X may be a superkey for R.

c) R must not have more than two attributes.

d) For one or more sets of its attributes X, the closure of X is X itself, i.e. $X^+ = X$.

e) For one or more sets of its attributes X, the closure of X contains all attributes, i.e. X^+ = [all attributes].

Question 6 – 15 points – Below is a relation and its known functional dependencies.

`Account(AccountNumber, RoutingNumber, AccountHolder, Branch, Bank, Location)`

This is the same relation with attribute abbreviations: `Account(AN, RN, AH, Br, Bk, L)`

Functional Dependencies	**Abbreviated Form**
{ Bank, Location } → { Branch }	{ Bk, L } → { Br }
{ AccountHolder } → { Bank }	{ AH } → { Bk }
{ RoutingNumber, AccountNumber } → { AccountHolder }	{ RN, AN } → { AH }
{ RoutingNumber } → { Bank }	{ RN } → { Bk }
{ Branch } → { Bank, Location }	{ Br } → { Bk, L }

Compute the closures listed below. Once complete, identify the key(s), if any, for the relation `Account`. You may use the abbreviated attributes in your solutions.

```
{ Bank, Location }+ = { Bank, Location, Branch }

{ AccountHolder }+ = { AccountHolder, Bank }

{ RoutingNumber, AccountNumber }+ =
            { RoutingNumber, AccountNumber, AccountHolder, Bank }

{ RoutingNumber }+ = { RoutingNumber, Bank }

{ Branch }+ = { Branch, Bank, Location }
```

Keys: There are no keys for relation Account.

Question 7 – 15 points – Using the relation, functional dependencies, and calculated closures from *Question 6*, decompose the Account relation into BCNF relations. Use the closure for { Branch }, which violates BCNF, to split Account for the first iteration of the decomposition. Show all your work, including intermediate relations. When complete, clearly mark the resulting BCNF relations along with the keys in each.

Again, you may use abbreviations. Below is the same relation and functional dependencies repeated for your convenience.

Account(AccountNumber, RoutingNumber, AccountHolder, Branch, Bank, Location)

With abbreviated attributes: Account(AN, RN, AH, Br, Bk, L)

Functional Dependencies	**Abbreviated Form**
{ Bank, Location } → { Branch }	{ Bk, L } → { Br }
{ AccountHolder } → { Bank }	{ AH } → { Bk }
{ RoutingNumber, AccountNumber } → { AccountHolder }	{ RN, AN } → { AH }
{ RoutingNumber } → { Bank }	{ RN } → { Bk }
{ Branch } → { Bank, Location }	{ Br } → { Bk, L }

Iteration 1

X = Branch

Branch(Branch, Bank, Location)

$Account_1$(Branch, AccountHolder, RoutingNumber, AccountNumber)

Iteration 2

Decompose: $Account_1$(Branch, AccountHolder, RoutingNumber, AccountNumber)

X = { RoutingNumber, AccountNumber }

$Account_{1.1}$(RoutingNumber, AccountNumber, AccountHolder)

$Account_{1.2}$(RoutingNumber, AccountNumber, Branch)

Solution

Branch(Branch, Bank, Location) or Branch(Branch, Bank, Location)

$Account_{1.1}$(RoutingNumber, AccountNumber, AccountHolder)

$Account_{1.2}$(RoutingNumber, AccountNumber, Branch)