Question 1 – **30 points** – Write a query that will return all account holders with total balances of at least $1,000,000. The total balance of an account holder is the sum of all transaction amounts on all his or her accounts.

For this problem, note that the column `AccountType` on `Accounts.Account` is of data type `NCHAR(1)`.

Required Result Columns

`AccountHolderId` – The `AccountHolderID` from `Accounts.AccountHolder`.
`LastName` – The last name as it appears in `LastName` of `Accounts.AccountHolder`.
`FirstName` – The first name as it appears in `FirstName` of `Accounts.AccountHolder`.
`CheckingBalance` – The sum of `Amount` in `Accounts.Transaction` for all accounts of an account holder where `AccountType` is "C".
`SavingsBalance` – The sum of `Amount` in `Accounts.Transaction` for all accounts of an account holder where `AccountType` is "S".
`TotalBalance` – The sum of `Amount` in `Accounts.Transaction` for all accounts of an account holder, regardless of the account type.

Implementation Requirements

Your solution should contain no subqueries or window functions. The results should be sorted by `TotalBalance` in descending order, then by `AccountHolderId` in ascending order.

```
SELECT AH.AccountHolderId, AH.LastName, AH.FirstName,
   SUM(IIF(A.AccountType = N'C', T.Amount, 0.00)) AS CheckingBalance,
   SUM(IIF(A.AccountType = N'S', T.Amount, 0.00)) AS SavingsBalance,
   SUM(T.Amount) AS TotalBalance
FROM Accounts.AccountHolder AH
   INNER JOIN Accounts.Account A ON A.AccountHolderId = AH.AccountHolderId
   INNER JOIN Accounts.[Transaction] T ON T.AccountId = A.AccountId
GROUP BY AH.AccountHolderId, AH.LastName, AH.FirstName
HAVING SUM(T.Amount) >= 1000000
ORDER BY TotalBalance DESC, AH.AccountHolderId ASC;
```

Question 3 – **25 points** – Write a query that will return all accounts, along with their 2019 opening balance and YTD activity, for the account holder having the email "john.doe@jmail.com". We want all accounts included, even if there is no activity on a given account.

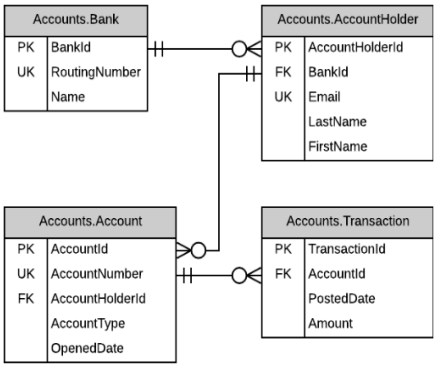
For this problem, note that the column `PostedDate` on `Accounts.Transaction` is of data type `DATE`.

Required Result Columns

`AccountNumber` – The `AccountNumber` of `Accounts.Account`.
`AccountType` – The `AccountType` of `Accounts.Account`.
`OpeningBalance` – The sum of `Amount` in `Accounts.Transaction` for all transactions posted before January 1, 2019. If an account had no activity before January 1, 2019, the value should be 0.00.
`YtdActivity` –The sum of `Amount` in `Accounts.Transaction` for all transactions posted on or after January 1, 2019. If an account has had no activity since January 1, 2019, the value should be 0.00.

Implementation Requirements

Your solution should use two scalar-valued subqueries; one to calculate `OpeningBalance` and the other to calculate `YtdActivity`. The results should be sorted by `AccountType` in ascending order, then by `AccountNumber` in ascending order.




```
SELECT A.AccountNumber, A.AccountType,
   (
      SELECT ISNULL(SUM(T.Amount), 0.00)
      FROM Accounts.[Transaction] T
      WHERE T.AccountId = A.AccountId
         AND T.PostedDate < '2019-01-01'
   ) AS OpeningBalance,
   (
      SELECT ISNULL(SUM(T.Amount), 0.00)
      FROM Accounts.[Transaction] T
      WHERE T.AccountId = A.AccountId
         AND T.PostedDate >= '2019-01-01'
   ) AS YtdActivity
FROM Accounts.AccountHolder AH
   INNER JOIN Accounts.Account A ON A.AccountHolderId = AH.AccountHolderId
WHERE AH.Email = N'john.doe@jmail.com'
ORDER BY A.AccountType ASC, A.AccountNumber ASC;
```

Question 4 – **20 points** – Write a query that will return account holders at the bank with routing number 123456789, who have had an account opened before January 1, 2015.

For this problem, note that the column `OpenedDate` on `Accounts.Account` is of data type `DATE`.

Required Result Columns

`LastName` – The last name as it appears in `LastName` of `Accounts.AccountHolder`.
`FirstName` – The first name as it appears in `FirstName` of `Accounts.AccountHolder`.
`Email` – The email address as it appears in `Email` of `Accounts.AccountHolder`.

Implementation Requirements

Your solution should use a subquery to determine whether an account holder has an account that was opened before January 1, 2015. The results should be sorted by `LastName` in ascending order, then by `FirstName` in ascending order.

```
/*****************************
 * Correlated Subquery
 *****************************/

SELECT AH.LastName, AH.FirstName, AH.Email
FROM Accounts.Bank B
   INNER JOIN Accounts.AccountHolder AH ON AH.BankId = B.BankId
WHERE B.RoutingNumber = N'123456789'
   AND EXISTS
   (
      SELECT *
      FROM Accounts.Account A
      WHERE A.AccountHolderId = AH.AccountHolderId
         AND A.OpenedDate < '2015-01-01'
   )
ORDER BY AH.LastName ASC, AH.FirstName ASC;

/*****************************
 * Self-contained Subquery
 *****************************/

SELECT AH.LastName, AH.FirstName, AH.Email
FROM Accounts.Bank B
   INNER JOIN Accounts.AccountHolder AH ON AH.BankId = B.BankId
WHERE B.RoutingNumber = N'123456789'
   AND AH.AccountHolderId IN
   (
      SELECT A.AccountHolderId
      FROM Accounts.Account A
      WHERE A.OpenedDate < '2015-01-01'
   )
ORDER BY AH.LastName ASC, AH.FirstName ASC;
```

Question 2 – **25 points** – Write a query that will return all accounts, along with their current balances and YTD activity, for the account holder having the email "john.doe@jmail.com". We want all accounts included, even if there is no activity on a given account.

For this problem, note that the column `PostedDate` on `Accounts.Transaction` is of data type DATE.

Required Result Columns

`AccountNumber` – The `AccountNumber` of `Accounts.Account`.
`AccountType` – The `AccountType` of `Accounts.Account`.
`YtdActivity` –The sum of `Amount` in `Accounts.Transaction` for all transactions posted on or after January 1, 2019. If an account has had no activity since January 1, 2019, the value should be 0.00.
`CurrentBalance` – The sum of `Amount` in `Accounts.Transaction` for all transactions posted on the account. If an account has had no activity, the value should be 0.00.

Implementation Requirements

Your solution should contain no subqueries or window functions. The results should be sorted by `AccountType` in ascending order, then by `AccountNumber` in ascending order.

```
SELECT A.AccountNumber, A.AccountType,
   SUM(IIF(T.PostedDate >= '2019-01-01', T.Amount, 0.00)) AS YtdActivity,
   ISNULL(SUM(T.Amount), 0.00) AS CurrentBalance
FROM Accounts.AccountHolder AH
   INNER JOIN Accounts.Account A ON A.AccountHolderId = AH.AccountHolderId
   LEFT JOIN Accounts.[Transaction] T ON T.AccountId = A.AccountId
WHERE AH.Email = N'john.doe@jmail.com'
GROUP BY A.AccountNumber, A.AccountType
ORDER BY A.AccountType ASC, A.AccountNumber ASC;
```

1. Write a query to return all students enrolled in either CIS560 or CIS562 for the term of Fall 2018. "Fall 2018" will appear as a `Name` in the `Term` table, and "CIS560", for example, is the `Name` in `Course`.

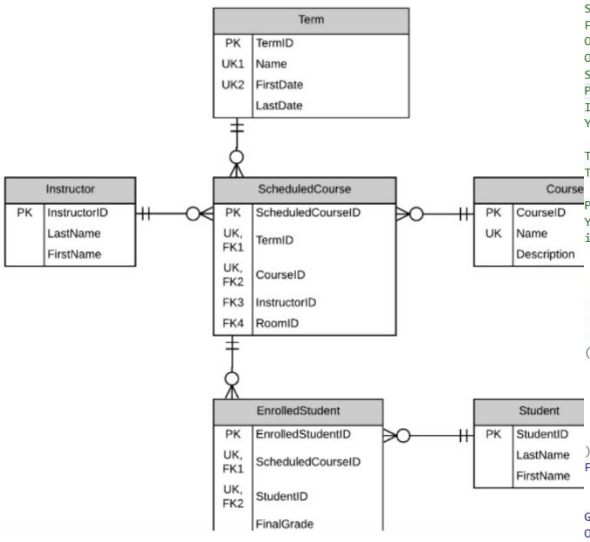
   Required Result Columns

   `LastName` – The last name of the student as it appears in `LastName` of `Student`.
   `FirstName` – The first name of the student as it appears in `FirstName` of `Student`.
   `StudentID` – The identifier of the student as it appears in `StudentID` of `Student`.
   `CourseName` – The name of the course as it appears in `Name` of `Course`.

   Implementation Requirements

   The results should be sorted by `LastName`, then `FirstName`, and finally `StudentID` all in ascending order.

```
SELECT S.LastName, S.FirstName, S.StudentID, C.Name AS CourseName
FROM Term T
   INNER JOIN ScheduledCourse SC ON SC.TermID = T.TermID
   INNER JOIN Course C ON C.CourseID = SC.CourseID
   INNER JOIN EnrolledStudent ES ON ES.ScheduledCourseID = SC.ScheduledCourseID
   INNER JOIN Student S ON S.StudentID = ES.StudentID
WHERE T.Name = N'Fall 2018'
   AND C.Name IN (N'CIS 560', N'CIS 562')
ORDER BY S.LastName ASC, S.FirstName ASC, S.StudentID ASC;
```



2. Write a query to return all instructors who are **not** teaching during the Fall 2018 term. "Fall 2018" will appear as the `Name` in the `Term` table to identify the Fall 2018 term.

   Required Result Columns

   `InstructorID` – The identifier of the instructor as it appears in `InstructorID` of `Instructor`.
   `LastName` – The last name of the instructor as it appears in `LastName` of `Instructor`.
   `FirstName` – The first name of the instructor as it appears in `FirstName` of `Instructor`.

   Implementation Requirements

   a. The solution may only use joins. That is, it should contain no subqueries.
   b. Each table in your solution should be referenced only once.
   c. The results should be sorted by `LastName`, then `FirstName`, and finally `InstructorID` all in ascending order.

```
SELECT I.InstructorID, I.LastName, I.FirstName
FROM Instructors I
   LEFT JOIN ScheduledCourse SC ON SC.InstructorID = I.InstructorID
   LEFT JOIN Term T ON T.TermID = SC.TermID
      AND T.Name = N'Fall 2018'
WHERE T.TermID IS NULL
ORDER BY I.LastName ASC, I.FirstName ASC, I.InstructorID ASC;
```

3. Write a query as an alternate solution to *Question 2*. This solution **should use a subquery**, which can either be self-contained or correlated. All other requirements are the same as *Question 2*, including that **each table should be referenced only once**.

```
-- Solution with self-contained subquery

SELECT I.InstructorID, I.LastName, I.FirstName
FROM Instructors I
WHERE I.InstructorID NOT IN
   (
      SELECT SC.InstructorID
      FROM Term T
         INNER JOIN ScheduledCourse SC ON SC.TermID = T.TermID
      WHERE T.Name = N'Fall 2018'
   )
ORDER BY I.LastName ASC, I.FirstName ASC, I.InstructorID ASC;

-- Solution with correlated subquery

SELECT I.InstructorID, I.LastName, I.FirstName
FROM Instructors I
WHERE NOT EXISTS
   (
      SELECT *
      FROM ScheduledCourse SC
         INNER JOIN Term T ON T.TermID = SC.TermID
      WHERE SC.InstructorID = I.InstructorID
         AND T.Name = N'Fall 2018'
   )
ORDER BY I.LastName ASC, I.FirstName ASC, I.InstructorID ASC;
```

```
Result Columns
SalespersonPersonID - Identifier of the sales person from Sales.Orders.
FullName - Name of the sales person.
OrderYear - The year the order was placed.
OrderMonth - The month the order was placed, containing values 1 - 12.
Sales - The total sales for the sales person and month.
PriorYearSales - The total sales for the sales person and month in the prior year. Use a correlated subquery to calculate.
Implementation Requirements
Your solution must use a correlated subquery to calculate the sales for the same month from prior year.

There is no required sort order, but for verifying your results, you may want to use sorting.
Therefore, there is no deduction if your solution includes an ORDER BY clause.

Partial Results
Your results should contain exactly 410 rows. The partial results Download partial results
include all rows for one sales person in the years 2013 and 2014.
 ****************************************/

 SELECT O.SalespersonPersonID, P.FullName,
 YEAR(O.OrderDate) AS OrderYear,
 Month(O.OrderDate) AS OrderMonth,
 SUM(OL.UnitPrice * OL.Quantity) AS Sales,
(
    SELECT SUM(OL.UnitPrice * OL.Quantity)
    FROM Sales.OrderLines OL
        INNER JOIN Sales.Orders OO ON OL.OrderID = OO.OrderID
    WHERE YEAR(OO.OrderDate) = (YEAR(O.OrderDate)-1) AND MONTH(OO.OrderDate) = MONTH(O.OrderDate) AND
    (OO.SalespersonPersonID = O.SalespersonPersonID)
) AS PriorYearSales
FROM Sales.Orders O
    INNER JOIN Application.People P ON P.PersonID = O.SalespersonPersonID
    INNER JOIN Sales.OrderLines OL ON OL.OrderID = O.OrderID
GROUP BY YEAR(O.OrderDate), MONTH(O.OrderDate), O.SalespersonPersonID, P.FullName
ORDER BY O.SalespersonPersonID ASC, YEAR(O.OrderDate) ASC, Month(O.OrderDate) ASC
```