--Subquery Exercise

--Use the IQSchool database for this exercise. Each question must use a subquery in its solution.

--\*\*If the questions could also be solved without a subquery, solve it without one as well\*\*

USE [A01-School]

GO

--1. Select the Payment dates and payment amount for all payments that were Cash

SELECT PaymentDate, Amount

FROM Payment

WHERE PaymentTypeID = -- Using the = means that the RH side must be a single value

-- Assuming that every PaymentTypeDescription will be UNIQUE,

-- the following subquery will return a single column and a single row

(SELECT PaymentTypeID

FROM PaymentType

WHERE PaymentTypeDescription = 'cash')

-- Here is the Inner Join version of the above

SELECT PaymentDate, Amount

FROM Payment P

INNER JOIN PaymentType PT

ON PT.PaymentTypeID = P.PaymentTypeID

WHERE PaymentTypeDescription = 'cash'

--2. Select The Student ID's of all the students that are in the 'Association of Computing Machinery' club

-- TODO: Student Answer Here

SELECT StudentID

FROM Activity

WHERE ClubId =

(SELECT ClubId

FROM Club

WHERE ClubName='Association of Computing Machinery')

-- 2.b. Select the names of all the students in the 'Association of Computing Machinery' club. Use a subquery for your answer. When you make your answer, ensure the outmost query only uses the Student table in its FROM clause.

SELECT FirstName + ' ' + LastName AS 'Student'

FROM Student

WHERE StudentID IN

(Select DISTINCT StudentID FROM Activity)

-------------------------------------------------------------------------

--V2--

SELECT DISTINCT FirstName + ' ' + LastName AS 'Student'

FROM Student

INNER JOIN Activity

ON Student.StudentID=Activity.StudentID

-------------------------------------------------------------------------

--V3--

--3. Select All the staff full names for staff that have taught a course.

SELECT FirstName + ' ' + LastName AS 'Staff'

FROM Staff

WHERE StaffID IN -- I used IN because the subquery returns many rows

(SELECT DISTINCT StaffID FROM Registration)

-- The above can also be done as an INNER JOIN...

SELECT DISTINCT FirstName + ' ' + LastName AS 'Staff'

FROM Staff

INNER JOIN Registration

ON Staff.StaffID = Registration.StaffID

--4. Select All the staff full names that taught DMIT172.

-- TODO: Student Answer Here

SELECT FirstName + ' ' + LastName AS 'StaffID'

FROM Staff

INNER JOIN Registration

ON Staff.StaffID = Registration.StaffID

--5. Select All the staff full names of staff that have never taught a course

SELECT FirstName + ' ' + LastName AS 'Staff'

FROM Staff

WHERE StaffID NOT IN -- I used IN because the subquery returns many rows

(SELECT DISTINCT StaffID FROM Registration)

-- To do the above questions with a JOIN requires that we use an OUTER JOIN...

SELECT FirstName + ' ' + LastName AS 'Staff'

FROM Staff

LEFT OUTER JOIN Registration

ON Staff.StaffID = Registration.StaffID

WHERE Registration.StaffID IS NULL

--6. Select the Payment TypeID(s) that have the highest number of Payments made.

-- Explore the counts of payment types, before we try the subquery

SELECT PaymentTypeID, COUNT(PaymentTypeID) AS 'How many times'

FROM Payment

GROUP BY PaymentTypeID

-- To get the payment type IDs whose count is greater than or equal to all the others

-- (i.e.: whose count is the highest)

SELECT PaymentTypeID

FROM Payment

GROUP BY PaymentTypeID

HAVING COUNT(PaymentTypeID) >= ALL (SELECT COUNT(PaymentTypeID)

FROM Payment

GROUP BY PaymentTypeID)

--7. Select the Payment Type Description(s) that have the highest number of Payments made.

SELECT PaymentTypeDescription

FROM Payment

INNER JOIN PaymentType

ON Payment.PaymentTypeID = PaymentType.PaymentTypeID

GROUP BY PaymentType.PaymentTypeID, PaymentTypeDescription

HAVING COUNT(PaymentType.PaymentTypeID) >= ALL (SELECT COUNT(PaymentTypeID)

FROM Payment

GROUP BY PaymentTypeID)

-- Examining the solution:

-- - First, take a look at the results of the subquery by itself - this gives us

-- the counts and we can visually see what the highest value is

(SELECT COUNT(PaymentTypeID)

FROM Payment

GROUP BY PaymentTypeID)

-- - Second, take a look at the outer query, but leave out the filtering of aggregates.

-- Also, display the count that is used in the HAVING clause. This should give you

-- an idea of what the right answers should be.

SELECT PaymentTypeDescription

, COUNT(PaymentType.PaymentTypeID)

FROM Payment

INNER JOIN PaymentType

ON Payment.PaymentTypeID = PaymentType.PaymentTypeID

GROUP BY PaymentType.PaymentTypeID, PaymentTypeDescription

--8. What is the total avg mark for the students from Edm?

SELECT AVG(Mark) AS 'Average'

FROM Registration

WHERE StudentID IN (SELECT StudentID FROM Student WHERE City = 'Edm')

-- The above results, done as a JOIN instead of a subquery

SELECT AVG(Mark) AS 'Average'

FROM Registration

INNER JOIN Student

ON Registration.StudentID = Student.StudentID

WHERE City = 'Edm'

-- 9. What is the avg mark for each of the students from Edm? Display their StudentID and avg(mark)

-- TODO: Student Answer Here...

-- 10. Which student(s) have the highest average mark? Hint - This can only be done by a subquery.

-- TODO: Student Answer Here...

-- 11. Which course(s) allow the largest classes? Show the course id, name, and max class size.

-- TODO: Student Answer Here...

-- 12. Which course(s) are the most affordable? Show the course name and cost.

-- TODO: Student Answer Here...

-- 13. Which staff have taught the largest classes? (Be sure to group registrations by course and semester.)

-- TODO: Student Answer Here...

-- 14. Which students are most active in the clubs?

-- TODO: Student Answer Here...

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Simple Table Creation - Columns and Primary Keys

\*

\* Emergency Service & Product

\* Specification Document 1

\* Version 1.0.0

\*

\* Author: Dan Gilleland

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

-- Select the CREATE DATABASE stement below to create the demo database.

-- CREATE DATABASE [ESP-A01]

USE [ESP-A01] -- this is a statement that tells us to switch to a particular database

-- Notice in the database name above, it is "wrapped" in square brackets because

-- the name had a hypen in it.

-- For all our other objects (tables, columns, etc), we won't use hypens or spaces, so

-- the use of square brackets are optional.

GO -- this statement helps to "separate" various DDL statements in our script

-- so that they are executed as "blocks" of code.

-- To create a database table, we use the CREATE TABLE statement.

-- Note that the order in which we create/drop tables is important

-- because of how the tables are related via Foreign Keys.

/\* DROP TABLE statements (to "clean up" the database for re-creation) \*/

/\* You should drop tables in the REVERSE order in which you created them \*/

IF EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'OrderDetails')

DROP TABLE OrderDetails

IF EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'InventoryItems')

DROP TABLE InventoryItems

IF EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'Orders')

DROP TABLE Orders

IF EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'Customers')

DROP TABLE Customers

-- To create a database table, we use the CREATE TABLE statement.

-- Note that the order in which we create/drop tables is important

-- because of how the tables are related via Foreign Keys.

CREATE TABLE Customers

(

-- The body of a CREATE TABLE will identify a comma-separated list of

-- Column Declarations and Table Constraints.

CustomerNumber int

-- The following is a PRIMARY KEY constraint that has a specific name

-- Primary Key constraints ensure a row of data being added to the table

-- will have to have a unique value for the Primary Key column(s)

CONSTRAINT PK\_Customers\_CustomerNumber

PRIMARY KEY

-- IDENTITY means the database will generate a unique whole-number

-- value for this column

IDENTITY(100, 1) -- The first number is the "seed",

-- and the last number is the "increment"

NOT NULL, -- NOT NULL means the data is required

FirstName varchar(50) NOT NULL,

LastName varchar(60) NOT NULL,

[Address] varchar(40) NOT NULL,

City varchar(35) NOT NULL,

Province char(2)

-- A DEFAULT constraint will supply a default value for a column

-- whenever no value is supplied when adding a row of data

CONSTRAINT DF\_Customers\_Province

DEFAULT ('AB')

-- A CHECK constraint ensures that only the specified value(s)

-- will be accepted when adding a row of data

CONSTRAINT CK\_Customers\_Province

CHECK (Province = 'AB' OR

Province = 'BC' OR

Province = 'SK' OR

Province = 'MB' OR

Province = 'QC' OR

Province = 'ON' OR

Province = 'NT' OR

Province = 'NS' OR

Province = 'NB' OR

Province = 'NL' OR

Province = 'YK' OR

Province = 'NU' OR

Province = 'PE')

NOT NULL,

PostalCode char(6)

CONSTRAINT CK\_Customers\_PostalCode

CHECK (PostalCode LIKE '[A-Z][0-9][A-Z][0-9][A-Z][0-9]')

NOT NULL,

PhoneNumber char(13)

CONSTRAINT CK\_Customers\_PhoneNumber

CHECK (PhoneNumber LIKE '([0-9][0-9][0-9])[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]')

NULL -- NULL means the data is optional

)

CREATE TABLE Orders

(

OrderNumber int

CONSTRAINT PK\_Orders\_OrderNumber

PRIMARY KEY

IDENTITY(200, 1) NOT NULL,

CustomerNumber int

-- Foreign Key constraints ensure that when a row of data is being

-- inserted or updated, there is a row in the referenced table

-- that has the same value as its Primary Key

CONSTRAINT FK\_Orders\_CustomerNumber\_Customers\_CustomerNumber

FOREIGN KEY REFERENCES

Customers(CustomerNumber) NOT NULL,

[Date] datetime NOT NULL,

Subtotal money

CONSTRAINT CK\_Orders\_Subtotal

CHECK (Subtotal > 0) NOT NULL,

GST money

CONSTRAINT CK\_Orders\_GST

CHECK (GST >= 0) NOT NULL,

-- Total money NOT NULL, -- regular column

-- Table-level constraints are used for any constraint involving

-- two or more columns

-- CONSTRAINT CK\_Orders\_Total CHECK (Total = Subtotal + GST)

Total AS Subtotal + GST -- This is now a Computed Column

)

CREATE TABLE InventoryItems

(

ItemNumber varchar(5)

CONSTRAINT PK\_InventoryItems\_ItemNumber

PRIMARY KEY NOT NULL,

ItemDescription varchar(50) NULL,

CurrentSalePrice money

CONSTRAINT CK\_InventoryItems\_CurrentSalePrice

CHECK (CurrentSalePrice > 0) NOT NULL,

InStockCount int NOT NULL,

ReorderLevel int NOT NULL

)

CREATE TABLE OrderDetails

(

OrderNumber int

CONSTRAINT FK\_OrderDetails\_OrderNumber\_Orders\_OrderNumber

FOREIGN KEY REFERENCES

Orders(OrderNumber) NOT NULL,

ItemNumber varchar(5)

CONSTRAINT FK\_OrderDetails\_ItemNumber\_InventoryItems\_ItemNumber

FOREIGN KEY REFERENCES

InventoryItems(ItemNumber) NOT NULL,

Quantity int

CONSTRAINT DF\_OrderDetails\_Quantity

DEFAULT (1)

CONSTRAINT CK\_OrderDetails\_Quantity

CHECK (Quantity > 0) NOT NULL,

SellingPrice money

CONSTRAINT CK\_OrderDetails\_SellingPrice

CHECK (SellingPrice >= 0) NOT NULL,

-- The Amount column is a CALCULATED (or "derived") column.

-- It's value is the result of multiplying Quantity by SellingPrice.

Amount AS Quantity \* SellingPrice ,

-- The following is a Table Constraint

-- A composite primary key MUST be done as a Table Constraint

-- because it involves two or more columns

CONSTRAINT PK\_OrderDetails\_OrderNumber\_ItemNumber

PRIMARY KEY (OrderNumber, ItemNumber) -- Specify all the columns in the PK

)

-- Let's insert a few rows of data for the tables (DML Statements)

PRINT 'Inserting customer data'

INSERT INTO Customers(FirstName, LastName, [Address], City, PostalCode)

VALUES ('Clark', 'Kent', '344 Clinton Street', 'Metropolis', 'S0S0N0')

INSERT INTO Customers(FirstName, LastName, [Address], City, PostalCode)

VALUES ('Jimmy', 'Olsen', '242 River Close', 'Bakerline', 'B4K3R1')

PRINT '-- end of customer data--'

PRINT ''

PRINT 'Inserting inventory items'

INSERT INTO InventoryItems(ItemNumber, ItemDescription, CurrentSalePrice, InStockCount, ReorderLevel)

VALUES ('H8726', 'Cleaning Fan belt', 29.95, 3, 5)

INSERT INTO InventoryItems(ItemNumber, ItemDescription, CurrentSalePrice, InStockCount, ReorderLevel)

VALUES ('H8621', 'Engine Fan belt', 17.45, 10, 5)

PRINT '-- end of inventory data --'

PRINT ''

-- Let's write an SQL Query statement to view the data in the database

-- Select the customer information

SELECT CustomerNumber, FirstName, LastName,

[Address] + ' ' + City + ', ' + Province AS 'Customer Address',

PhoneNumber

FROM Customers

-- Let's do another set of DML statements to add more data to the database

PRINT 'Inserting an order'

INSERT INTO Orders(CustomerNumber, [Date], Subtotal, GST)

VALUES (100, GETDATE(), 17.45, 0.87)

INSERT INTO OrderDetails(OrderNumber, ItemNumber, Quantity, SellingPrice)

VALUES (200, 'H8726', 1, 17.45)

PRINT '-- end of order data --'

PRINT ''

GO

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Change Requests for Spec 1

\* Perform table changes through ALTER statements.

\* Syntax for ALTER TABLE can be found at

\* http://msdn.microsoft.com/en-us/library/ms190273.aspx

\* ALTER TABLE statements allow us to change an existing table without

\* having ot drop it or lose information in the table

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

-- A) Allow Address, City, Province, and Postal Code to be NULL

-- SQL requires each column to be altered SEPARATELY.

ALTER TABLE Customers

ALTER COLUMN [Address] varchar(40) NULL

GO -- this statement helps to "separate" various DDL statements in our script. It's optional.

ALTER TABLE Customers

ALTER COLUMN City varchar(35) NULL

GO

ALTER TABLE Customers

ALTER COLUMN Province char(2) NULL

GO

ALTER TABLE Customers

ALTER COLUMN PostalCode char(6) NULL

GO

-- B) Add a check constraint on the First and Last name to require at least two letters.

-- % is a wildcard for zero or more characters (letter, digit, or other character)

-- \_ is a wildcard for a single character (letter, digit, or other character)

-- [] are used to represent a range or set of characters that are allowed

IF OBJECT\_ID('CK\_Customers\_FirstName', 'C') IS NOT NULL -- 'C' specifies that I'm looking for a constraint

ALTER TABLE Customers DROP CONSTRAINT CK\_Customers\_FirstName

ALTER TABLE Customers

ADD CONSTRAINT CK\_Customers\_FirstName

CHECK (FirstName LIKE '[A-Z][A-Z]%') -- two letters plus any other chars

-- \ 1 /\ 1 /

-- Positive match for 'Fred'

-- Positive match for 'Wu'

-- Negative match for 'F'

-- Negative match for '2udor'

IF OBJECT\_ID('CK\_Customers\_LastName', 'C') IS NOT NULL

ALTER TABLE Customers DROP CONSTRAINT CK\_Customers\_LastName

ALTER TABLE Customers

ADD CONSTRAINT CK\_Customers\_LastName

CHECK (LastName LIKE '[A-Z][A-Z]%')

-- Once the ALTER TABLE changes are made for A) and B),

-- we can insert Customer information allowing certain columns to be NULL.

INSERT INTO Customers(FirstName, LastName)

VALUES ('Fred', 'Flintstone')

INSERT INTO Customers(FirstName, LastName)

VALUES ('Barney', 'Rubble')

INSERT INTO Customers(FirstName, LastName, PhoneNumber)

VALUES ('Wilma', 'Slaghoople', '(403)555-1212')

INSERT INTO Customers(FirstName, LastName, [Address], City)

VALUES ('Betty', 'Mcbricker', '103 Granite Road', 'Bedrock')

-- Select the customer information

SELECT CustomerNumber, FirstName, LastName,

[Address] + ' ' + City + ', ' + Province AS 'Customer Address',

PhoneNumber

FROM Customers

GO

/\*

You can check that the constraints work on the first/last name

by highlighting and running these scripts. They should fail.

INSERT INTO Customers(FirstName, LastName)

VALUES ('F', 'Flintstone')

INSERT INTO Customers(FirstName, LastName)

VALUES ('Fred', 'F')

\*/

-- C) Add an extra bit of information on the Customer table. The client wants to

-- start tracking customer emails, so they can send out statements for

-- outstanding payments that are due at the end of the month.

ALTER TABLE Customers

ADD Email varchar(30) NULL

-- Adding this as a nullable column, because customers already

-- exist, and we don't have emails for those customers.

GO

-- D) Add indexes to the Customer's First and Last Name columns

-- Indexes improve the performance of the database when retrieving information.

CREATE NONCLUSTERED INDEX IX\_Customers\_FirstName

ON Customers (FirstName)

CREATE NONCLUSTERED INDEX IX\_Customers\_LastName

ON Customers (LastName)

GO -- End of a batch of instructions

-- E) Add a default constraint on the Orders.Date column to use the current date.

-- GETDATE() is a global function in the SQL Server Database

-- GETDATE() will obtain the current date/time on the database server

IF OBJECT\_ID('DF\_Orders\_Date', 'C') IS NOT NULL

ALTER TABLE Orders DROP CONSTRAINT DF\_Orders\_Date

ALTER TABLE Orders

ADD CONSTRAINT DF\_Orders\_Date

DEFAULT GETDATE() FOR [Date]

-- Use \ this / for \this column/ if no value was supplied when INSERTING data

GO

-- To illustrate the default value, consider this sample row for the Orders table

INSERT INTO Orders(CustomerNumber, Subtotal, GST)

VALUES (101, 150.00, 7.50)

-- Select the current orders

SELECT OrderNumber, CustomerNumber, Total, [Date]

FROM Orders

GO

-- Prep-data for change request F) below ....

-- Some inventory data without a ItemDescription

INSERT INTO InventoryItems(ItemNumber, ItemDescription, CurrentSalePrice, InStockCount, ReorderLevel)

VALUES ('GR35A', NULL, 45.95, 8, 5)

INSERT INTO InventoryItems(ItemNumber, CurrentSalePrice, InStockCount, ReorderLevel)

VALUES ('KD5-Q', 1.45, 10, 7)

GO

SELECT ItemNumber, ItemDescription, CurrentSalePrice

FROM InventoryItems

GO

-- F) Change the InventoryItems.ItemDescription column to be NOT NULL

-- WAIT!! We have described the ItemDescription as allowing NULL values.

-- That means we might have data in the table where the

-- ItemDescription doesn't exist.

-- If we try to make that column NOT NULL, what will we do about

-- the existing data in the database where it is "empty"??

-- We can fix that by updating the data in the database

-- where that description is missing.

UPDATE InventoryItems

SET ItemDescription = '-missing-'

WHERE ItemDescription IS NULL

GO

-- Also Note: We might be asked to put a default value for the column

-- that will become required. In this case, let's use a

-- default value of '-no description-'

ALTER TABLE InventoryItems

ADD CONSTRAINT DF\_InventoryItems\_Description

DEFAULT '-no description-' FOR ItemDescription

GO

-- Here's some sample data where ItemDescription is not entered

-- and the default value comes into play. (We'll do 2 rows at once.)

INSERT INTO InventoryItems(ItemNumber, CurrentSalePrice, InStockCount, ReorderLevel)

VALUES ('B-95R', 45.95, 8, 5),

('GR47D', 92.45, 3, 3)

GO

-- Now we can change the ItemDescription to be required (NOT NULL)

ALTER TABLE InventoryItems

ALTER COLUMN ItemDescription varchar(50) NOT NULL

GO

-- Let's enter a bit more information in our database

INSERT INTO InventoryItems(ItemNumber, ItemDescription, CurrentSalePrice, InStockCount, ReorderLevel)

VALUES ('BL-92', '92mm Bolt', 3.95, 18, 6)

-- Let's see the data in the Inventory table

SELECT ItemNumber, ItemDescription, CurrentSalePrice

FROM InventoryItems

GO

-- G) Add an indes on the Item's Description column, to improve search.

CREATE NONCLUSTERED INDEX IX\_InventoryItems\_ItemDescription

ON InventoryItems (ItemDescription)

-- ------------------------------------------

-- H) Data change requests: All inventory items that are less than $5.00 have to

-- have their prices increased by 10%.

UPDATE InventoryItems

SET CurrentSalePrice = CurrentSalePrice \* 0.10

WHERE CurrentSalePrice < 5.00

-- Somebody got married....

UPDATE Customers

SET LastName = 'Flintstone'

WHERE FirstName = 'Wilma' AND LastName = 'Slaghoople'

UPDATE Customers

SET LastName = 'Rubble'

WHERE FirstName = 'Betty' AND LastName = 'Mcbricker'

UPDATE Customers

SET [Address] = '103 Granite Road',

City = 'Bedrock'

WHERE LastName = 'Rubble'

UPDATE Customers

SET [Address] = '105 Granite Road',

City = 'Bedrock'

WHERE LastName = 'Flintstone'

GO

-- Increase the current sale price for products between $10 and $30 by 2%.

UPDATE InventoryItems

SET CurrentSalePrice = CurrentSalePrice \* 0.02

WHERE CurrentSalePrice BETWEEN 10 AND 30

-- Update the prices for all belts by $5.50, because of newly introduced recyling fees

UPDATE InventoryItems

SET CurrentSalePrice = CurrentSalePrice + 5.5

WHERE ItemDescription LIKE '%Belt%'

-- And, we want to get rid of some inventory

DELETE FROM InventoryItems

WHERE ItemNumber IN ('GR47D', 'KD5-Q')

--SIMPLE SELECT EXERCISE 1

USE [A01-School]

GO

-- Simple Select, without any other clauses

SELECT 'Dan', 'Gilleland'

-- Simple Select with expressions

SELECT 'Dan' + ' ' + 'Gilleland', 18 \* 52, '5' + '10'

-- Specify a column name with some hard-code/calculated values

SELECT 'Dan' + ' ' + 'Gilleland' AS 'Instructor',

19 \* 52 AS 'Weeks at the job'

-- Let's use the SELECT statement with database tables

-- 1. Select all the information from the Club table

-- The SELECT statement produces a table of results which will consist of one or more rows of data.

SELECT ClubId, ClubName

FROM Club

-- The \* can be used to indicate all of the columns. But DO NOT DO THIS, or your instructor

-- WILL deduct marks EVERY TIME you do this.

SELECT \*

FROM Club

-- Notice that when selecting from an existing table, when we identify which columns we want to

-- show, then those column names are used as the column title for the results table.

-- The AS keyword in the SELECT clause is used to assign a title to the column in the results table

SELECT ClubId AS 'ID', ClubName

FROM Club

-- Pro-Tip: Press [ctrl] + r to toggle the results window

-- Pro-Tip: If you write the FROM clause before specifying the columns,

-- you will get Intellisense help on the column names

-- Pro-Tip: Press [ctrl] + [shift] + r to "refresh" intellisense

-- 2. Select the FirstNames and LastNames of all the students

SELECT FirstName, LastName

FROM Student

-- 2.a. Repeat the above query, but using column aliases

SELECT FirstName AS 'First Name', LastName AS 'Last Name'

FROM Student

-- 2.b. Select the student id and full name of all the students

SELECT StudentID, FirstName + ' ' + LastName AS 'Full Name'

FROM Student

-- 2.c. Select the first and last names of all the students,

-- and sort the results by the last name

SELECT FirstName, LastName

FROM Student

ORDER BY LastName

-- 2.d. Select the first and last names of all the students,

-- and sort the results by the last name, then by the first name

SELECT FirstName, LastName

FROM Student

ORDER BY LastName, FirstName

--3. Select the CourseId and CourseName of all the courses. Use the column aliases of Course ID and Course Name

SELECT CourseId AS 'Course ID', CourseName AS 'Course Name'

FROM Course

--4. Select all the course information for courseID 'DMIT101'

-- I will mark the following as a ZERO

--SELECT \* -- All columns

--FROM Course

SELECT CourseID, CourseName, CourseHours, MaxStudents, CourseCost

FROM Course

WHERE CourseID = 'DMIT101'

--5. Select the Staff names who have job positionID of 3

SELECT FirstName, LastName

--,PositionID -- Press [ctrl] + k, then [ctrl] + u to un-comment

FROM Staff

WHERE PositionID = 3

-- BTW, what is PositionID of 3 referring to?

SELECT PositionID, PositionDescription

FROM Position

--6. Select the Course Names whose course hours are less than 96

SELECT C.CourseName

FROM Course C -- I can have an alias to the table name

WHERE C.CourseHours < 96

-- Type with me the following...

SELECT ST.LastName, ST.DateHired, ST.DateReleased

FROM Staff AS ST -- The use of the AS keyword in producing table/column aliases is optional

-- but it can be a good idea for readability.

-- You can use the full table name to fully-qualify your column names

SELECT Staff.LastName, Staff.FirstName, Staff.DateHired

FROM Staff

WHERE Staff.DateReleased IS NOT NULL

-- NOTE: You can't mix the use of a table alias with the full name of the table

-- 7. Select the studentID's, CourseID and mark where the Mark is between 70 and 80

SELECT StudentID, CourseId, Mark

FROM Registration

WHERE Mark BETWEEN 70 AND 80 -- BETWEEN is inclusive

--WHERE Mark >= 70 AND Mark <= 80

-- 7.a. Select the StudentIDs where the withdrawal status is null

SELECT StudentID

--, WithdrawYN

FROM Registration

WHERE WithdrawYN IS NULL -- we use IS NULL instead of = NULL, because = NULL won't work.

-- 7.b. Select the student ids of students who have withdrawn from a course

SELECT StudentID

FROM Registration

WHERE WithdrawYN = 'Y'

--8. Select the studentID's, CourseID and mark where the Mark is

-- between 70 and 80 and the courseID is DMIT223 or DMIT168

SELECT R.StudentID, R.CourseId, R.Mark

FROM Registration R

WHERE R.Mark BETWEEN 70 AND 80

AND (R.CourseId = 'DMIT223' OR R.CourseId = 'DMIT168')

-- alternate answer to #8

SELECT R.StudentID, R.CourseId, R.Mark

FROM Registration R

WHERE R.Mark BETWEEN 70 AND 80

AND R.CourseId IN ('DMIT223', 'DMIT168') -- The IN keyword allows us to have a list of values

-- that will be checked in a OR manner.

--8.a. Select the studentIDs, CourseID and mark where the Mark is 80 and 85

SELECT R.StudentID, R.CourseId, R.Mark

FROM Registration R

WHERE R.Mark = 80 OR R.Mark = 85

-- The next two questions introduce the idea of "wildcards" and pattern matching in the WHERE clause

-- \_ is a wildcard for a single character

-- % is a wildcard for zero or more characters

-- [] is a pattern for a single character matching the pattern in the square brackets

--9. Select the students first and last names who have last names starting with S

SELECT FirstName, LastName

FROM Student

WHERE LastName LIKE 'S%'

--10. Select Coursenames whose CourseID have a 1 as the fifth character

SELECT CourseName

FROM Course

WHERE CourseID LIKE '\_\_\_\_1%' -- four underscores, 1, %

-- DMIT158

--11. Select the CourseID's and CourseNames where the CourseName contains the word 'programming'

SELECT CourseID, CourseName

FROM Course

WHERE CourseName LIKE '%Programming%'

--12. Select all the ClubNames who start with N or C.

SELECT ClubName

FROM Club

WHERE ClubName LIKE '%N'

--13. Select Student Names, Street Address and City where the lastName is only 3 letters long.

SELECT LastName, StreetAddress, City

FROM Student

WHERE LastName LIKE '\_\_%'

--14. Select all the StudentID's where the PaymentAmount < 500 OR the PaymentTypeID is 5

--View Exercise

USE [A01-School]

GO

--1. Create a view of staff full names called StaffList.

IF OBJECT\_ID('StaffList', 'V') IS NOT NULL

DROP VIEW StaffList

GO

CREATE VIEW StaffList

AS

SELECT FirstName + ' ' + LastName AS 'StaffFullName'

FROM Staff

GO

-- Now we can use the StaffList view as if it were a table

SELECT StaffFullName

FROM StaffList

-- SP\_HELP Staff

-- SP\_HELPTEXT StaffList -- Gets the text of the View

-- SP\_HELP StaffList -- Gets schema info on the View

GO

--2. Create a view of staff ID's, full names, positionID's and datehired called StaffConfidential.

IF OBJECT\_ID('StaffConfidential', 'V') IS NOT NULL

DROP VIEW StaffConfidential

GO

CREATE VIEW StaffConfidential

AS

SELECT StaffID,

FirstName + ' ' + LastName AS 'FullName',

PositionID,

DateHired

FROM Staff

GO

-- I can use it accordingly:

SELECT FullName, DateHired

FROM StaffConfidential

GO

--2a. Alter the StaffConfidential view so that it includes the position name.

ALTER VIEW StaffConfidential

AS

SELECT StaffID,

FirstName + ' ' + LastName AS 'FullName',

P.PositionID,

PositionDescription AS 'PositionName',

DateHired

FROM Staff S

INNER JOIN Position P ON S.PositionID = P.PositionID

GO

SELECT FullName, PositionName, PositionID

FROM StaffConfidential

GO

--3. Create a view called StaffExperience that returns the name of the staff members that have taught courses and the names of the courses they have taught. Sort the results by staff last name then first name, then course name.

IF OBJECT\_ID('StaffExperienceRaw', 'V') IS NOT NULL

DROP VIEW StaffExperienceRaw

GO

CREATE VIEW StaffExperienceRaw

AS

--

SELECT FirstName + ' ' + LastName as 'StaffName',

CourseName

FROM Staff AS S

INNER JOIN Registration AS R ON S.StaffID = R.StaffID

INNER JOIN Course AS C ON R.CourseId = C.CourseId

ORDER BY LastName, FirstName, CourseName

-- To learn more about OFFSET, see the following article

-- https://www.essentialsql.com/using-offset-and-fetch-with-the-order-by-clause/

-- "The OFFSET is the number of rows to skip before including them in the result."

OFFSET 0 ROWS

GO

IF OBJECT\_ID('StaffExperience', 'V') IS NOT NULL

DROP VIEW StaffExperience

GO

CREATE VIEW StaffExperience

AS

SELECT StaffName, CourseName

FROM StaffExperienceRaw

GROUP BY StaffName, CourseName

GO

SELECT StaffName, CourseName FROM StaffExperience

--4. Create a view called StudentGrades that retrieves the student ID's, full names, courseId's, course names, and marks for each student.

-- TODO: Student Answer here

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Using the Views

\* If an operation fails write a brief explanation why.

\* Do not just quote the error message generated by the server!

\*/

--5. Use the student grades view to create a grade report for studentID 199899200 that shows the students ID, full name, course names and marks.

--6. Select the same information using the student grades view for studentID 199912010.

--7. Retrieve the code for the student grades view from the database.