/\* Orysya Stus - Assignment 4 \*/

/\* Lab 11.1: Self-Join Hierarchies \*/

/\*

Skill Check 1: In JProCo join the Employee table to itself and just show each employee’s EmpID, FirstName and LastName. Show each boss’s first and last name as an expression field called BossFullName. The expression field should have a space in the middle. Also make sure to use the right type of outer join so that Sally Smith appears, even though she has no boss. When you are done, your result should resemble the figure below.

\*/

USE JProCo

SELECT

E1.EmpID,

E1.FirstName,

E1.LastName,

BL.FirstName + ' ' + BL.LastName AS BossFullName

FROM Employee BL

RIGHT JOIN Employee E1

ON E1.ManagerID = BL.EmpID

/\* Lab 11.2: Range Hierarchies \*/

/\*

Skill Check 1: From the Employee table in JProCo write a query to see employee # 3 (Lee Osako) and all the people who were hired after him. Show just the FirstName , LastName and HireDate from both tables in the join. Make the fourth field an expression field that says “Was hired before”. When you’re done, your result will look like the figure you see in the figure below.

\*/

SELECT

E1.FirstName,

E1.LastName,

E1.HireDate,

'Was hired before' AS Note,

BL.FirstName,

BL.LastName,

BL.HireDate

FROM Employee E1

INNER JOIN Employee BL

ON E1.HireDate < BL.HireDate

WHERE E1.EmpID = 3

/\*

Skill Check 2: In dbBasics, use the Military table to write a range hierarchy showing the highest Army GradeRank and all the GradeRanks which are below Colonel. When you’re done, your result will look like the figure below.

\*/

Use dbBasics

SELECT

M1.GradeRank,

M1.GradeName,

'OUTRANKS' AS Note,

M2.GradeRank,

M2.GradeName

FROM Military M1

INNER JOIN Military M2

ON M1.GradeRank > M2.GradeRank

WHERE M1.GradeRank = 8

/\* Lab 12.1 Basic Subqueries \*/

/\*

Skill Check 1: Some invoices are large orders with many products on them. In the SalesInvoice table , 10 of the 1877 sales invoices contain more than 30 products . Run an aggregated subquery using the SalesInvoiceDetail table in order to find the InvoiceIDs for the 10 sales invoices containing more than 30 products. Feed those 10 InvoiceIDs into the criteria of the outer query. When you’re done, your result will resemble the figure below.

\*/

SELECT \*

FROM SalesInvoice

WHERE InvoiceID IN (

SELECT InvoiceID

FROM SalesInvoiceDetail

GROUP BY InvoiceID

HAVING Count(\*) > 30)

/\*

Skill Check 2: Write a subquery which will feed EmpIDs into an outer query of the Employee table. Show the records for just those employees who have found grants. When you’re done, your result should resemble the figure below.

\*/

SELECT \*

FROM Employee

WHERE EmpID IN (SELECT DISTINCT EmpID

FROM [Grant])

/\*

Skill Check 3: Query the Customer table using a subquery, which shows all the customers who have purchased (Hint: everyone appearing in the SalesInvoice table has bought something from JProCo). The query should show all customers who have ordered at least once from JProCo. If a customer has ordered multiple times, make sure they only show once in the result. When you’re done, your result will resemble the figure below.

\*/

SELECT \*

FROM Customer

WHERE CustomerID IN (SELECT DISTINCT CustomerID

FROM SalesInvoice)

/\* Lab 12.2: Correlated Subqueries \*/

/\*

Skill Check 1: Query the Supplier table and use a subquery from the CurrentProducts table to create a ProductCount expression field. The ProductCount should show the number of products for each supplier. When you are done, your result should resemble the figure below.

\*/

SELECT

\*,

(SELECT Count(\*)

FROM CurrentProducts AS CP

WHERE CP.SupplierID = S.SupplierID) AS ProductCount

FROM Supplier AS S

/\*

Skill Check 2: Query the Employee table to find the 7 newest employees by hire date. Create an expression field called GrantCount that shows the number of grants found by each of those newest employees. Only the FirstName, LastName and HireDate columns and the expression field are to be shown. When you are done, your result should resemble the figure

\*/

SELECT TOP 7

E.FirstName,

E.LastName,

E.HireDate,

(SELECT Count(\*)

FROM [Grant] AS G

WHERE G.EmpID = E.EmpID) AS GrantCount

FROM Employee AS E

ORDER BY E.HireDate DESC

/\* Lab 12.3: Subquery Extensions \*/

/\*

Skill Check 1: We want to compare the RetailPrice values of products supplied by JProCo’s three external suppliers. Show the supplier who, if it were to introduce a $ 1000 product, that would represent the highest RetailPrice they offer. Find all suppliers (from the Supplier table) whose current products are less than $ 1000 (from the CurrentProducts table). Run the comparison using a correlated subquery and use ANY, ALL, or SOME to modify your comparison operator. Use the CurrentProducts and Supplier tables in JProCo for your correlated subquery. Your results should resemble the figure below.

\*/

SELECT \*

FROM Supplier AS S

WHERE 1000 > ALL (SELECT RetailPrice

FROM CurrentProducts AS CP

WHERE CP.SupplierID = S.SupplierID)

/\*

Skill Check 2: Show all employees who, if they were to find a $ 5000 grant, it would represent the highest grant they have ever found. Use the ALL operator with your correlated subquery. When you’re done, your result will resemble the figure you see below.

\*/

SELECT \*

FROM Employee AS E

WHERE 5000 > ALL (SELECT Amount

FROM [Grant] AS G

WHERE G.EmpID = E.EmpID)

/\* Lab 13.1: Updates and Subqueries \*/

/\*

Skill Check 1: Review the data shown here in the PriceIncrease table. Turn the records of this table into a subquery which will increment prices in the CurrentProducts table. See figure below.

\*/

UPDATE CP SET RetailPrice = RetailPrice + (SELECT SUM(Change)

FROM PriceIncrease AS PR

WHERE PR.ProductID = CP.ProductID)

FROM CurrentProducts CP

WHERE ProductID IN (SELECT ProductID FROM PriceIncrease)

SELECT \*

FROM CurrentProducts

/\* Lab 13.2: Existence Subqueries \*/

/\*

Skill Check 1: Use an EXISTS subquery to find all products that have been sold according to the SalesInvoiceDetail table as shown in the table below.

SELECT \*

FROM CurrentProducts AS cp

WHERE EXISTS (--Remaining Code Here)

\*/

SELECT \*

FROM CurrentProducts AS CP

WHERE EXISTS (SELECT ProductID

FROM SalesInvoiceDetail AS SD

WHERE SD.ProductID = CP.ProductID)

/\*

Skill Check 2: Use an EXISTS subquery to find all customers who have made a purchase according to the SalesInvoice table. See the two figures below.

\*/

SELECT \*

FROM Customer AS CU

WHERE EXISTS (SELECT CustomerID

FROM SalesInvoice AS SI

WHERE CU.CustomerID = SI.CustomerID)

/\*

Skill Check 3: Use an EXISTS subquery to find all employees who have found more than two grants. See the two figures below.

\*/

SELECT \*

FROM Employee AS EM

WHERE EXISTS (SELECT EmpID

FROM [Grant] AS GR

WHERE EM.EmpID = GR.EmpID

GROUP BY EmpID

HAVING Count(\*) > 2)

/\* Lab 14.1: Using MERGE \*/

/\*

Skill Check 1: Create a stored procedure called UpsertLocation that accepts four parameters – one for each field in the Location table. This stored procedure should make updates to existing records and insert any new records. After creating the stored procedure, call on it with the following codes:

\*/

CREATE PROC UpsertLocation @LocID int, @Street varchar(50), @City varchar(25), @State varchar(25)

AS

BEGIN

MERGE Location AS L

USING (SELECT @LocID,@Street,@City,@State)

as LocScr (LocationID, street, city, [state])

ON L.LocationID = LocScr.LocationID

WHEN MATCHED THEN UPDATE SET L.street = @Street,

L.City = @City, L.[State] = @State

WHEN NOT MATCHED THEN INSERT (LocationID, street,

city, [state])

VALUES(@LocID,@Street,@City,@State);

END

GO

EXEC UpsertLocation 1,'545 Pike','Seattle','WA'

EXEC UpsertLocation 5,'1595 Main','Philadelphia','PA'

GO

SELECT \* FROM Location

/\*

Skill Check 2: You have a table named PayRatesFeed with the updated pay information that needs to be fed into the PayRates table shown in the figure below.

\*/

MERGE dbo.PayRates AS PR

USING dbo.PayRatesFeed AS F

ON PR.EmpID = F.EmpID

WHEN MATCHED THEN UPDATE

SET PR.YearlySalary = F.YearlySalary,

PR.MonthlySalary = F.MonthlySalary,

PR.HourlyRate = F.HourlyRate

WHEN NOT MATCHED THEN INSERT

(EmpID,YearlySalary, MonthlySalary, HourlyRate)

VALUES (F.EmpID,

F.YearlySalary,

F.MonthlySalary,

F.HourlyRate);

GO

SELECT \*

FROM PayRatesFeed

/\*

For Skill Check2

Write a MERGE statement that will update employee 1 to a YearlySalary of $ 97500 and insert a new pay record for EmpID 14 with year salary of $ 52000. When you are done, your result should resemble the figure below.

\*/

MERGE dbo.PayRates AS PR

USING dbo.PayRatesFeed AS F

ON PR.EmpID = F.EmpID

WHEN MATCHED AND NOT (PR.YearlySalary = F.YearlySalary) THEN UPDATE

SET PR.YearlySalary = F.YearlySalary,

PR.MonthlySalary = F.MonthlySalary,

PR.HourlyRate = F.HourlyRate

WHEN NOT MATCHED THEN INSERT

(EmpID,YearlySalary, MonthlySalary, HourlyRate)

VALUES (F.EmpID,

F.YearlySalary,

F.MonthlySalary,

F.HourlyRate);

GO

SELECT \*

FROM PayRatesFeed

SELECT \*

FROM PayRates

/\* Lab 15.1: The OUTPUT Clause \*/

/\*

Skill Check 1: The Contractor table has four records: three records from Location 1 and one from Location 2. JProCo is closing down Location 2 next year and thus must delete one record from the Contractor table. As this record is being deleted, we want to see the affected record displayed in our result pane. Write this query and execute the deletion with the appropriate output statement. When you’re done, the output will show on your screen as shown below.

\*/

DELETE FROM Contractor

OUTPUT deleted.\*

WHERE LocationID = 2

/\*

Skill Check 2: A new contractor named Vern Anderson is coming onboard to work in Location 1. Write the code to execute this insert, and use the GETDATE() function to populate the current date and time in the HireDate field of the Contractor table. Use the OUTPUT clause to show the results of your insertion as you run it. When you’re done, your result should resemble the figure below.

\*/

INSERT INTO Contractor

OUTPUT inserted.\*

VALUES ('Anderson','Vern',GetDate(),1)

/\*

Skill Check 3: All six yearly salaried employees are getting a raise of $ 1500 per year. Run the appropriate UPDATE statement on the PayRates table and show the results on screen. When you are done, your result should resemble the figure below.

\*/

UPDATE dbo.PayRates SET YearlySalary = YearlySalary + 1500

OUTPUT inserted.\*, deleted.\*

WHERE YearlySalary IS NOT NULL AND Selector IS NOT NULL