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

-- LAB 11

--

-- Exercise 1

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

USE TSQL;

GO

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

-- Task 1

--

-- Write a SELECT statement to return the productid, productname, supplierid, unitprice, and discontinued columns from the Production.Products table. Filter the results to include only products that belong to the category Beverages (categoryid equals 1).

--

-- Observe and compare the results that you got with the desired results shown in the file 52 - Lab Exercise 1 - Task 1 Result.txt.

--

-- Modify the T-SQL code to include the following supplied T-SQL statement. Put this statement before the SELECT clause:

--

-- Execute the complete T-SQL statement. This will create an object view named ProductBeverages under the Production schema.

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

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

-- Task 2

--

-- Write a SELECT statement to return the productid and productname columns from the Production.ProductsBeverages view. Filter the results to include only products where supplierid equals 1.

--

-- Execute the written statement and compare the results that you got with the desired results shown in the file 53 - Lab Exercise 1 - Task 2 Result.txt.

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

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

-- Task 3

--

-- The IT department has written a T-SQL statement that adds an ORDER BY clause to the view created in task 1.

--

-- Execute the provided code. What happened? What is the error message? Why did the query fail?

--

-- Modify the supplied T-SQL statement by including the TOP (100) PERCENT option. The query should look like this:

--

-- Execute the modified T-SQL statement. By applying the needed changes, you have altered the existing view. Notice that you are still using still use the ORDER BY clause.

--

-- If you write a query against the modified Production.ProductsBeverages view, will it be guaranteed that the retrieved rows will be sorted by productname? Please explain.

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

ALTER VIEW Production.ProductsBeverages AS

SELECT

productid, productname, supplierid, unitprice, discontinued

FROM Production.Products

WHERE categoryid = 1

ORDER BY productname;

GO

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

-- Task 4

--

-- The IT department has written a T-SQL statement that adds an additional calculated column to the view created in task 1.

--

-- Execute the provided query. What happened? What is the error message? Why did the query fail?

--

-- Apply the changes needed to get the T-SQL statement to execute properly.

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

ALTER VIEW Production.ProductsBeverages AS

SELECT

productid, productname, supplierid, unitprice, discontinued,

CASE WHEN unitprice > 100. THEN N'high' ELSE N'normal' END

FROM Production.Products

WHERE categoryid = 1;

GO

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

-- Task 5

--

-- Remove the created view by executing the provided T-SQL statement. Execute this code exactly as written inside a query window.

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

IF OBJECT\_ID(N'Production.ProductsBeverages', N'V') IS NOT NULL

DROP VIEW Production.ProductsBeverages;

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

-- LAB 11

--

-- Exercise 2

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

USE TSQL;

GO

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

-- Task 1

--

-- Write a SELECT statement against a derived table and retrieve the productid and productname columns. Filter the results to include only the rows in which the pricetype column value is equal to high. Use the SELECT statement from exercise 1, task 4 as the inner query that defines the derived table. Do not forget to use an alias for the derived table. (You can use the alias p.)

--

-- Execute the written statement and compare the results that you got with the desired results shown in the file 62 - Lab Exercise 2 - Task 1 Result.txt.

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

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

-- Task 2

--

-- Write a SELECT statement to retrieve the custid column and two calculated columns: totalsalesamount, which returns the total sales amount per customer, and avgsalesamount, which returns the average sales amount of orders per customer. To correctly calculate the average sales amount of orders per customer, you will first have to calculate the total sales amount per order. You can do so by defining a derived table based on a query that joins the Sales.Orders and Sales.OrderDetails tables. You can use the custid and orderid columns from the Sales.Orders table and the qty and unitprice columns from the Sales.OrderDetails table.

--

-- Execute the written statement and compare the results that you got with the recommended result shown in the file 63 - Lab Exercise 2 - Task 2 Result.txt.

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

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

-- Task 3

--

-- Write a SELECT statement to retrieve the following columns:

-- orderyear, representing the year of the order date

-- curtotalsales, representing the total sales amount for the current order year

-- prevtotalsales, representing the total sales amount for the previous order year

-- percentgrowth, representing the percentage of sales growth in the current order year compared to the previous order year

-- You will have to write a T-SQL statement using two derived tables. To get the order year and total sales columns for each SELECT statement, you can query an already existing view named Sales.OrderValues. The val column represents the sales amount.

-- Do not forget that the order year 2006 does not have a previous order year in the database, but it should still be retrieved by the query.

--

-- Execute the T-SQL code and compare the results that you got with the recommended result shown in the file 64 - Lab Exercise 2 - Task 3 Result.txt.

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

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

-- LAB 11

--

-- Exercise 3

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

USE TSQL;

GO

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

-- Task 1

--

-- Write a SELECT statement like that in exercise 2, task 1, but use a CTE instead of a derived table. Use inline column aliasing in the CTE query and name the CTE ProductBeverages.

--

-- Execute the T-SQL code and compare the results that you got with the recommended result shown in the file 72 - Lab Exercise 3 - Task 1 Result.txt.

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

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

-- Task 2

--

-- Write a SELECT statement against Sales.OrderValues to retrieve each customer’s ID and total sales amount for the year 2008. Define a CTE named c2008 based on this query using the external aliasing form to name the CTE columns custid and salesamt2008. Join the Sales.Customers table and the c2008 CTE, returning the custid and contactname columns from the Sales.Customers table and the salesamt2008 column from the c2008 CTE.

--

-- Execute the T-SQL code and compare the results that you got with the recommended result shown in the file 73 - Lab Exercise 3 - Task 2 Result.txt.

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

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

-- Task 3

--

-- Write a SELECT statement to retrieve the custid and contactname columns from the Sales.Customers table. Also retrieve the following calculated columns:

-- salesamt2008, representing the total sales amount for the year 2008

-- salesamt2007, representing the total sales amount for the year 2007

-- percentgrowth, representing the percentage of sales growth between the year 2007 and 2008

-- If percentgrowth is NULL, then display the value 0.

--

-- You can use the CTE from the previous task and add another CTE for the year 2007. Then join both of them with the Sales.Customers table. Order the result by the percentgrowth column.

--

-- Execute the T-SQL code and compare the results that you got with the recommended result shown in the file 74 - Lab Exercise 3 - Task 3 Result.txt.

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

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

-- LAB 11

--

-- Exercise 4

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

USE TSQL;

GO

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

-- Task 1

--

-- Write a SELECT statement against the Sales.OrderValues view and retrieve the custid and totalsalesamount columns as a total of the val column. Filter the results to include orders only for the order year 2007.

--

-- Execute the written statement and compare the results that you got with the recommended result shown in the file 82 - Lab Exercise 4 - Task 1 Result.txt.

--

-- Define an inline table-valued function using the following function header and add your previous query after the RETURN clause.

--

-- Modify the query by replacing the constant year value 2007 in the WHERE clause with the parameter @orderyear.

--

-- Highlight the complete code and execute it. This will create an inline table-valued function named dbo.fnGetSalesByCustomer.

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

-- initial SQL statement

CREATE FUNCTION dbo.fnGetSalesByCustomer

(@orderyear AS INT) RETURNS TABLE

AS

RETURN

-- copy here the SQL statement

GO

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

-- Task 2

--

-- Write a SELECT statement to retrieve the custid and totalsalesamount columns from the dbo.fnGetSalesByCustomer inline table-valued function. Use the value 2007 for the needed parameter.

--

-- Execute the written statement and compare the results that you got with the recommended result shown in the file 83 - Lab Exercise 4 - Task 2 Result.txt.

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

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

-- Task 3

--

-- In this task, you will query the Production.Products and Sales.OrderDetails tables. Write a SELECT statement that retrieves the top three sold products based on the total sales value for the customer with ID 1. Return the productid and productname columns from the Production.Products table. Use the qty and unitprice columns from the Sales.OrderDetails table to compute each order line’s value, and return the sum of all values per product, naming the resulting column totalsalesamount. Filter the results to include only the rows where the custid value is equal to 1.

--

-- Execute the T-SQL code and compare the results that you got with the recommended result shown in the file 84 - Lab Exercise 4 - Task 3\_1 Result.txt.

--

-- Create an inline table-valued function based on the following function header, using the previous SELECT statement. Replace the constant custid value 1 in the query with the function’s input parameter @custid:

--

-- Highlight the complete code and execute it. This will create an inline table-valued function named dbo.fnGetTop3ProductsForCustomer that excepts a parameter for the customer id.

--

-- Test the created inline table-valued function by writing a SELECT statement against it and use the value 1 for the customer id parameter. Retrieve the productid, productname, and totalsalesamount columns, and use the alias p for the inline table-valued function.

--

-- Execute the T-SQL code and compare the results that you got with the recommended result shown in the file 85 - Lab Exercise 4 - Task 3\_2 Result.txt.

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

-- initial SQL statement

GO

CREATE FUNCTION dbo.fnGetTop3ProductsForCustomer

(@custid AS INT) RETURNS TABLE

AS

RETURN

-- copy here the SQL statement

GO

-- write here the SQL statement against the created function

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

-- Task 4

--

-- Write a SELECT statement to retrieve the same result as in exercise 3, task 3, but use the created inline table-valued function in task 2 (dbo.fnGetSalesByCustomer).

--

-- Execute the written statement and compare the results that you got with the recommended result shown in the file 86 - Lab Exercise 4 - Task 4 Result.txt.

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

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

-- Task 5

--

-- Remove the created inline table-valued functions by executing the provided T-SQL statement. Execute this code exactly as written inside a query window.

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

IF OBJECT\_ID('dbo.fnGetSalesByCustomer') IS NOT NULL

DROP FUNCTION dbo.fnGetSalesByCustomer;

IF OBJECT\_ID('dbo.fnGetTop3ProductsForCustomer') IS NOT NULL

DROP FUNCTION dbo.fnGetTop3ProductsForCustomer;

GO