wawiwa
Tech Training

Data Analysis SQL Class Booklet

Lesson 13 - SQL in the analyst cycle

All class exercises are in the class presentation

Lesson 14 - DML (Data Manipulation Language) and Import/Export to csv file

Part 1 – Select into, Insert into

1. Write a query that creates a new table called MarketingContacts with the following columns from the PersonPerson and Person.EmailAddress tables:
   BusinessEntityID, First name, Middle name, Last name, Email address.

   Display only people who are classified in PersonType as 'IN' and for whom the type of business promotion specified in their EmailPromotion details is 1.

2. Display all the data from the new table, MarketingContacts.

3. The cashier made an error while typing the data and a line was omitted. Therefore, add the following data as one more row in the MarketingContacts table:

   BusinessEntityID = 30000
   First name = Noam
   Last name = Morchi
   Email = noam811@adventure-works.com

4. Continuing from the previous question, in order to check that the data was input to the MarketingContacts table correctly, write a query that displays only the input row.

Results | Messages

	BusinessEntityID	FirstName	MiddleName	LastName	EmailAddress
1	30000	Noam	NULL	Morchi	noam811@adventure-works.com

5. Write a query that creates a new table called NewProductTable with the following columns from the tables we are working with:

   a. Product ID

   b. Category code

   c. Category name

   d. Sub-category code

   e. Sub-category name

f. List price

g. Item cost

Note: Consider from which table each field should be taken.

6. Continuing from the previous question, display the table you created.

Results Messages

	ProductID	ProductCategoryID	CategoryName	ProductSubcategoryID	SubCategoryName	ListPrice	StandardCost
231	726	2	Components	14	Road Frames	337.22	187.1571
232	727	2	Components	14	Road Frames	337.22	187.1571
233	728	2	Components	14	Road Frames	337.22	187.1571
234	729	2	Components	14	Road Frames	337.22	187.1571

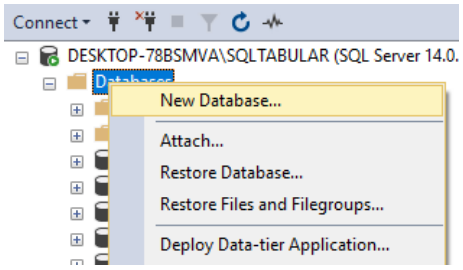
ProductID	Product CategoryID	Category Name	ProductSub categoryID	SubCategory Name	ListPrice	Standard Cost
100	2	Components	12	Mountain Frames	500	280
101	3	Clothing	20	Gloves	490	110

7. Write a query that adds the following records:

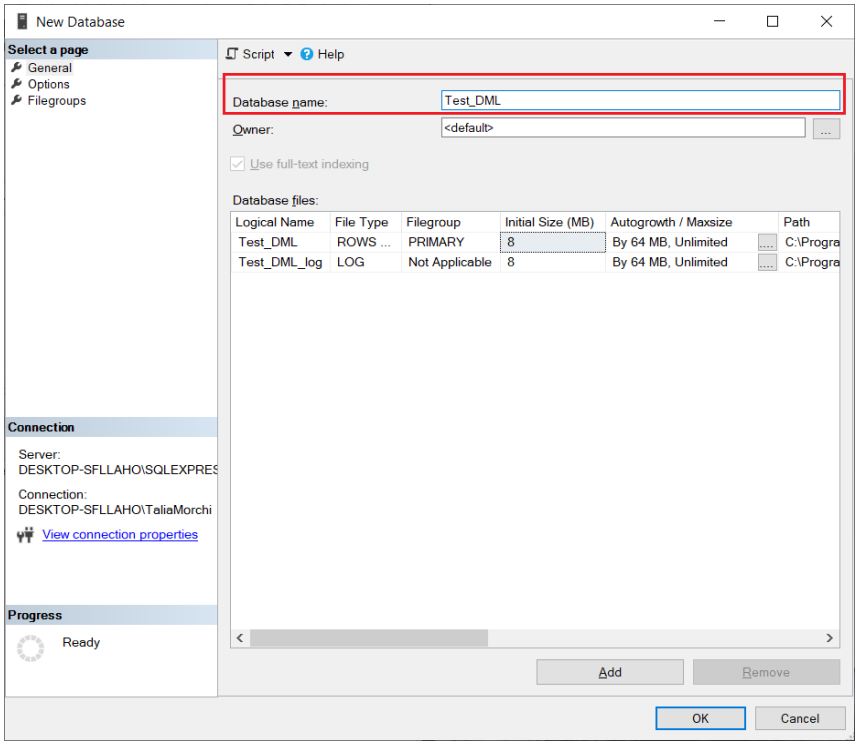
8. Continuing from the previous question, write a query that displays only the two rows that were added to the table.

9. To prepare for the next section, create a new database . Choose from one of the two following methods:

a. Right-click on the DataBases folder NewDatabase

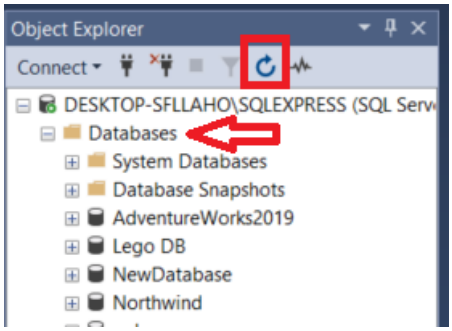


Then, choose a new name for your database, and click OK.



b. Run the following command to create a new database called Test_DML:
create database Test_DML

Once you have created the new database, go to the Databases folder and click Refresh (rounded arrow in blue)



10. Write a query that creates a new table called SalesPerCustomer2012 in the new database (Test_DML) that displays the SubTotal of each customer for the orders they placed in 2012.

    The table should consist of the following columns:
    Customer ID, First name, Last name and SubTotal of the orders from 2012.

    Instruction:
    Write a query that displays the total payment for all orders in 2012 for each customer, and the other columns listed.

    Once the query displays the desired data, add a line of code to the query to make it a Select Into query to add the results to another database.

    If you do not remember how, consult the lesson presentation.
    Refer to the following question to see a preview of the resulting table.

11. Write a query that displays the resulting table. Pay attention which database you are running the query on.

    Following is a preview of the results:

Results | Messages

	CustomerID	FirstName	LastName	TotalSales
1	11090	Trevor	Bryant	3578.27
2	11129	Julia	Wright	3578.27
3	11171	Jonathan	Hill	3578.27
4	11175	Luis	Wang	3578.27
5	11189	Lawrence	Blanco	3578.27
6	11191	Kristi	Perez	3578.27
7	11216	Jasmine	Torres	3578.27

Part 2 – Select into, Insert into, Update, Delete

1. In the NewProductTable, delete the rows with a ListPrice equal to 0. How many lines were deleted?

2. Continuing from the previous question, display the data from the NewProductTable table, and check that there are no rows with a list price equal to 0.

3. Update the SubCategoryName of product number 709 in the NewProductTable to read: Blue Socks.
   What was the value before the change?
   Check your answer.

4. Update the SubCategoryName of all the products with the ProductSubcategoryID 24 in the NewProductTable table to read: Long tights
   What was the value before the change?
   Check your answer.

5. Preparation for the next question:
   Write a query that displays the Product ID and ListPrice columns from the NewProductTable only for the items with ProductIDs 100 and 101.

   What is the list price of these two items?

6. The list prices of products number 100 and 101 in the NewProductTable table, increased by 10%. Write a query that will update the new prices of these products in the NewProductTable.

   What is the list price after the price raise?
   (Use the query from the previous question.)

7. In the NewProductTable, delete all the rows with a product code between 700 and 850 (inclusive) and a ProductCategoryID of 2 or 3.

   How many rows were deleted?

8. Delete all the rows in the NewProductTable.
   Check your answer

Part 3 – Import and Export data to csv file

1. Write a query that displays the following columns from the Production.Product table: ProductID, Name, ProductNumber, ListPrice, StandardCost.

2. Select the query results and copy them (including titles).
   Open an Excel sheet, paste the results, save it as a CSV file, and close it.

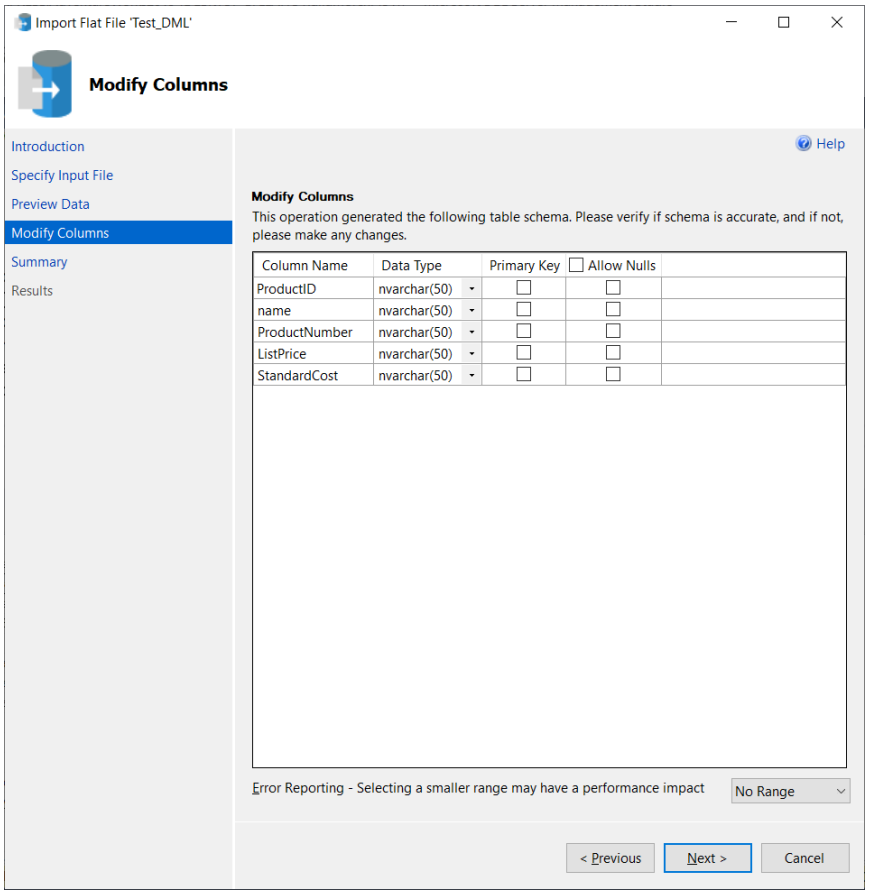
3. Import the csv file created from the query results into the database Test_DML:
   (This question has 'e' sub-sections)

a. Check that the defined data type setting of the fields suits the content. If not, change it to one of the data types learned. (Consult the appropriate slide from the presentation.)

b. If the column is a primary key of the table, indicate it in the column definition (Primary Key).

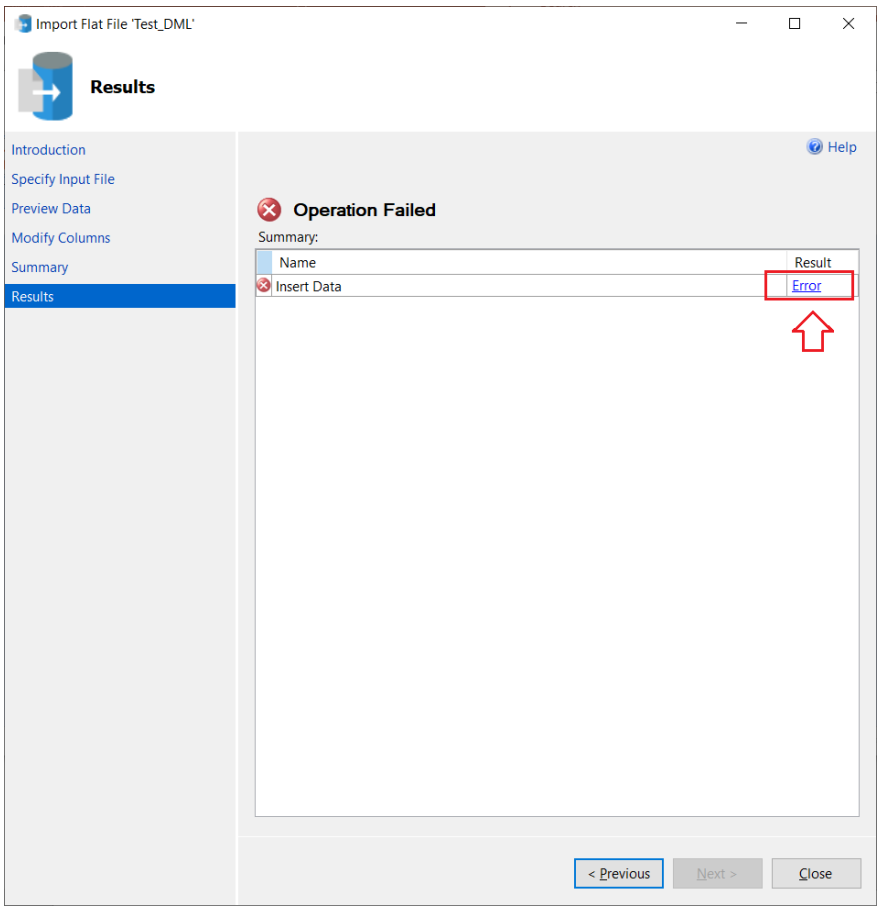
c. If the values in the column can contain NULL, indicate it in the column definition (Allow Nulls).

d. The three previous sections will be performed in the following screen in data import:



e. If the data import is unsuccessful, click on the Error to open the error message. (See illustration below.)

It is possible to go back by clicking the Previous button and to correct the fields that were not set correctly.



4. Verify that the table was imported correctly by writing a simple query that displays all the data from this table.

5. Write a query that displays the BusinessEntityID, FirstName and LastName from the person.person table.

6. Save the results as a CSV file by right-clicking on the query results Save Results As…

7. Open the CSV file and verify that the data was saved properly.

Lesson 15 – DDL (Data Definition Language) and View

Part 1 – Create, Constraints, Index

1. Create a new database called Test_DDL. From now on, in this practice, this is the only database that will be used.
2. Add a new table called Departments to the database.
   The table will contain the following columns:
   a. DepartmentCode: an integer, Primary key
   b. Name: A string up to 10 characters long
3. Create another index for the Departments table that sorts the data according to the value in the Name column.
4. Add a new table called Employees to the database.
   The table will contain the following columns:
   a. EmployeeNo : an integer, Primary key
   b. FirstName: a tring up to 20 characters long
   c. LastName: a string up to 20 characters long
   d. PhoneNo : a string up to 15 characters long
   e. Department: an integer, Foreign key to the DepartmentCode column in the Departments table
   f. Country: a string up to 20 characters long, default is "USA"
5. Create another index for the Employees table that sorts the data first according to the value in the Department column, in ascending order, and second according to the EmployeeNo column, in descending order.

Part 2 – Alter, Delete, Truncate

1. Add a column called Location, a 50 character string, to the Departments table. Set the default value in the column as: Main Office

2. Add a column called BirthDate, which will contain date or date and time data, to the Employees table.

3. After review, it appears that there is no need for the PhoneNo column in the Employees table. Delete this column from the table.

4. After further review, it appears that there is no need for an Employees table. Delete this table.

5. Write a query based on the data in the AdventureWorks database, and insert the query results into a new table named Orders in the Test_DDL database.

   The query will display the following columns from the order data:
   SalesOrderID, OrderDate, CustomerID, ProductID, OrderQty and LineTotal.

   Hint: Select into

6. Continuing from the previous section, add a column called SpecialSale to the Orders table. The data type in this column will be an integer (int) and the default value 0.

   (Explanation: The reason for this field is to designate the special sales. These will be defined according to criteria that will be specified in the following questions.)

7. Display the data in the Orders table.

   a. What are the values in the SpecialSale column?

   b. Why does the column not contain 0 values, even though the default value was set as 0?

8. Reset (update the value to 0) the SpecialSale field for all the rows in the Orders table.
   Check your answer

9. Now, it has been decided that a special order record is any order row with a LineTotal over $ 10,000.

   Update the SpecialSale field to 1 only for the sales with a LineTotal over 10,000.

10. Write a query that displays all the columns from the Orders table. Sort the data by LineTotal in descending order.

    Use this query to check the results of the previous query.

11. Delete all the rows in the Orders table that have an order date from 2012 and the value 1 in the SpecialSale column.

12. Delete all the data in the Orders table.

13. Delete the Orders table.

Part 3 – View

Instructions:

Be sure to check after each section that the operation ran successfully.

You will work as an analyst. Assessing your performance is necessary and critical for your professional integrity.

1. Create a new VIEW called vSaleFullDetails that will display data from an Order details table (to be defined in next paragraph), together with important fields from an Order header table.

   The fields to be displayed:
   Order number, Order date, Customer number, Quantity ordered, Item price after discount (calculated field – give it a name), total to be paid per row.

   Note : Examine the columns, recognize the meaning of the values in the columns, and calculate the price after discount. There are several ways to calculate the value in this column. Be sure that you are calculating correctly. It is best to take one line as an example and calculate it manually to check the result.

Results | Messages

	SalesOrderID	orderdate	CustomerID	OrderQty	ProductID	Unit price after discount	LineTotal
1	43659	2011-05-31 00:00:00.000	29825	1	776	2024.994000	2024.994000
2	43659	2011-05-31 00:00:00.000	29825	3	777	2024.994000	6074.982000
3	43659	2011-05-31 00:00:00.000	29825	1	778	2024.994000	2024.994000
4	43659	2011-05-31 00:00:00.000	29825	1	771	2039.994000	2039.994000
5	43659	2011-05-31 00:00:00.000	29825	1	772	2039.994000	2039.994000
6	43659	2011-05-31 00:00:00.000	29825	2	773	2039.994000	4079.988000
7	43659	2011-05-31 00:00:00.000	29825	1	774	2039.994000	2039.994000
8	43659	2011-05-31 00:00:00.000	29825	3	714	28.840400	86.521200
9	43659	2011-05-31 00:00:00.000	29825	1	716	28.840400	28.840400
10	43659	2011-05-31 00:00:00.000	29825	6	709	5.700000	34.200000
11	43659	2011-05-31 00:00:00.000	29825	2	712	5.186500	10.373000

2. Continuing from the previous question, display the VIEW that was created (vSaleFullDetails).

3. Continuing from the previous question, add the following data to the VIEW in the place that seems to be most correct column order:
   First name of the customer, Last name of the customer.

   In addition, limit sales details to sales in 2013 only.

4. Continuing from the previous question, run the query from question 2 again and check that the changes were successfully done.

5. Create a new VIEW called vSalePerYearSeller that will display the total quantity and value of sales for each SalesPersonID each year (i.e., record for seller 1 for 2011, record for seller 1 year 2012 ... record for seller 2 year 2011, etc.)

   The fields that to be displayed: Year, SalesPersonID, Total quantity of items sold and Total sales price, grouped by year and seller.

Results | Messages

	Year	SalesPersonID	TotalQty	TotalPrice
1	2011	NULL	1201	3863120.213400
2	2012	NULL	2743	6390599.947300
3	2013	NULL	28959	10732127.330000
4	2014	NULL	27495	8372829.730000
5	2011	274	30	28926.246500
6	2012	274	1061	453524.523283
7	2013	274	1366	431088.723659
8	2014	274	638	178584.362508
9	2011	275	1394	875823.831840
10	2012	275	8456	3375456.894536

6. Display the records in the query sorted according to year and salesperson, to create an "Annual sales report by seller":

   a. Try to find a way to sort the records as requested. Only after you have tried, move on to the next section – whether you succeeded and especially if you did not.

   b. The ORDER BY phrase cannot be written into VIEW, so in order to sort data, a query must be written to retrieve the data and sort it. Do this.

Results | Messages

	Year	SalesPersonID	TotalQty	TotalPrice
1	2011	NULL	1201	3863120.213400
2	2011	274	30	28926.246500
3	2011	275	1394	875823.831840
4	2011	276	1289	1149715.325300
5	2011	277	2053	1311627.291846
6	2011	278	746	500091.820200
7	2011	279	1681	1521289.188068
8	2011	280	709	648485.586200
9	2011	281	1293	967597.289926
10	2011	282	1896	1175007.475324

Lesson 16 – User-Defined Scalar Functions

Part 1 – Basic scalar functions: create, alter, drop, and call the function

1. Create a function called fnFuncLearning that does not take parameters and returns integer values (type int).
   Set the function to always return the number 2.
   Call the function and check that it returns the correct value.

2. Modify the fnFuncLearning function so that it returns the result of the following formula (Do not calculate the result yourself. Let the function do the calculation.):
   6 + 2 * (4-2 * 3)

   Call the function, and check the accuracy.

3. Modify the fnFuncLearning function so that it takes a parameter of type int. Decide on a parameter name.
   Modify the function's operation so that the result it returns is the parameter (the value sent to the function in the parameter) multiplied (*) by 10.

   Call the function and send a parameter. Check that the result is correct.

   Reminder: When calling a function with a parameter, write the parameter inside parentheses. For example:
   selectdbo.fnFuncLearning(5)as Result

4. Modify the fnFuncLearning function so it takes two parameters of type int, and returns the product of the first parameter multiplied by the second.

   Call the function and send a parameter. Check that the result is correct

   Reminder: To call a function with more than one parameter, separate the parameters with a comma. For example:
   selectdbo.fnFuncLearning(12, 5)as Result

5. The preliminary practice is complete, so there is no further need for the fnFuncLearning function. Delete the function.

6. Create a new function called fnGetProfit that takes 3 parameters: price, cost and quantity (more details below), and calculates the profit for all items.

a. Parameter Details:
   Parameter Name: @Price; Data Type: decimal (8,2)
   Parameter Name: @Cost; Data Type: decimal (8,2)
   Parameter name: @Qty; Data type: int

b. Think how to calculate profit.
   Hint: (Price - Cost) * Qty

c. Check your answer

7. Continuing from the previous section, the function will be used in 3 different queries.
   Write a query for each of the following sections:

   a. In order to check the accuracy of the function, run it with fixed parameters and, at the same time, calculate the expected result manually, to make sure that the results are identical.

      Send the following values to the function:
      Price = 100, Cost = 30, Qty = 10

      Calculate the answer manually, and check to be sure that the query returns the correct answer.

   b. Explore the theoretical profit from each of the products in the product table.
      Send the function the following values:
      list price, item cost, and quantity = 1.
      Run the query only on products that have a value for price (price higher than 0).

   c. In order to check the real profit from the Order details table, write a query that displays the following columns:

   d. Order number, item number, price after discount (calculated column. There are two ways to calculate this value.), item cost (from the Product table), profit per sales record (by calling the function and sending the appropriate parameters).

      Examine the results. Are there any strange results? If so, what might be the reason for this?

Part 2–Scalar functions that use variables

1. For the purpose of learning, create a function called fnFuncLearning and follow the instructions in the following sections:
   (If the function was not deleted in the previous part of the exercise, update the function using "alter").

   a. The function will take two parameters of data type int.

   b. Within the function, define a variable of type integer (int) and give it a name of your choice.

   c. The function will insert the result of the product of the two received parameters into the variable.

   d. The function will returnsthe variable that has been defined as the returned value.

   Call the function and send a parameter to it. Check that the result is correct.

2. Create a function called fnGetOrderCustomer that takes a parameter, Order number (int data type), and returns the customer number from that order (int data type). Think of a way to check that the function is working properly, and verify that the result is accurate.

3. Continue on from the previous question. In order to check the accuracy of the function, run the two queries below and compare the results.

   One query uses the function
   The other query performs the same calculation as the function.

   When the two queries are run, the answers obtained in both should be identical. (If the answer is not the same, there is an error in one of the queries)

   a.
```
Select dbo.fnGetOrderCustomer(43767) as CustID
```

   b.
```
Select SalesOrderID,
       CustomerID
from sales.SalesOrderHeader
where SalesOrderID = 43767
```

4. Create a new function called fnGetProductOrderAmount, which takes a ProductID and a year, and returns the SubTotal for that product in that year.

   Think what data type the function returns.

   Call the function and send a parameter. Check that the result is correct.

5. in the same way as in the previous question, create a new function called fnGetProductOrderQty that takes a ProductID and a year, and returns the OrderQty for that product in that year.

   Think what data type the function returns.

   Call the function and send a parameter. Check that the result is correct.

6. Write a query based on the product table that uses the functions from the previous two questions and displays the product code, product name, SubTotal in 2012 and OrderQty in 2012 for each product in the product table.

   Do you think it is necessary to filter the results so that only products that were actually ordered in 2012 are displayed, or is there also significance to the information in the unsold rows?

   Preview of the query results:

Results | Messages

	ProductID	Name	OrderAmount2012	OrderQty2012
1	1	Adjustable Race	NULL	NULL
2	879	All-Purpose Bike Stand	NULL	NULL
3	712	AWC Logo Cap	16273391.16	2048
4	3	BB Ball Bearing	NULL	NULL
5	2	Bearing Ball	NULL	NULL
6	877	Bike Wash - Dissolver	NULL	NULL
7	316	Blade	NULL	NULL
8	843	Cable Lock	9188995.33	773

7. Create a new function called fnSalesPerYear that takes a year as a parameter (e.g., 2013, 2014 ... What type of variable is this?), and returns the SubTotal in that year from the Order header file.

   Call the function and send it a parameter. Check that the result is correct

   Instructions and Hints:

a. Parameter: an integer (int) that represents a year

b. Value returned from the function: a decimal number: decimal (10,2), i.e., a total of 10 digits, two of which are after the decimal point.

c. Hint 1: A variable should be used.

d. Hint 2: Think how you could write a query that does this.

8. Following on from the previous question, modify the fnSalesPerYear function so it takes two parameters: year and BusinessEntityID.
The function will return the order amount for that BusinessEntityID in the specified year.

   Detailed instructions:

   a. The function will take two parameters of data type int (for entity number and year).

   b. Recall the relationships between the Sales.SalesOrderHeader table and the Person.Person table, and how to filter orders by BusinessEntityID. Refer to the ERD file.

9. Continuing on from the previous question, write a query based on the Persons table, that displays the BusinessEntityID, first name, last name and total of orders by that person in 2013.

   Filter the results to display only the people who ordered products in 2013.

   A point to consider:
   Note that running a function from a query requires system resources, so there is a price to pays in run time.
   It is, therefore, important to decide when there is an advantage to using a function and when it is worthwhile to write a standard query.

Lesson 17– Temporary Tables & Stored Procedures

Part 1 – Temporary tables

1. Set up a temporary table named #tmpProduct, and input the data of all the products with a ProductNumber that begins with BK.

2. Check that a temporary table has been created within the temporary database (tempdb), and write a query that displays all the data from this table.

3. Open a new query window and try to run the query from the previous question. Did it succeed? Why?

4. Input the following data from the Products table into the new temporary table, #tmpNewProduct:
   Product ID, catalog number, color, product name, weight and list price.

5. Use the temporary table to calculate how many items there are of each color. Sort in ascending order.

Part 2 – Stored procedures

1. Create a procedure named spProductList that runs the following query:

```
select ProductID,
       ProductNumber,
       [Name]
from Production.Product
```

2. Continue on from the previous question. In order to check the accuracy of the procedure, check that the procedure was created in the "stored procedures" folder. (If you do not remember the full path, refer to the presentation from the lesson.)

   In addition, run the procedure, and check that the desired result was obtained.

3. Following on from the previous question, delete the spProductList procedure.

4. Create a procedure called spRankYears that displays the following data for each year: year, order quantity, total order amount (SubTotal) for that year.

   Call the procedure to check its integrity.

5. Create a procedure called spBestSeller that displays the following data for each item ordered: the item code, the total quantity of the item ordered and total amount for ordering this item.

   Filter the query results to show only 2013 data.
   Sort the results by the total amount for the item in descending order.

   Call the procedure and check that it is correct.

   A preview of the procedure results:

Results | Messages

	ProductID	OrderedQty	TotalAmount
1	782	1470	2212974.782652
2	783	1262	1932388.290685
3	779	1164	1815673.093232
4	784	1036	1666660.023576
5	781	1033	1657616.282084
6	780	1004	1596847.227451
7	793	797	1262950.162500
8	794	721	1154069.879040

6. Continuing on from the previous question, modify the spBestSeller procedure to add a column in which the product is ranked according to the value in the Total amount column.

   Display only the 20 best-selling items, and sort the results by the ranking.

   Call the procedure and check that it is correct.

   A preview of the procedure results:

Results | Messages

	ProductID	OrderedQty	TotalAmount	RankByAmount
1	782	1470	2212974.782652	1
2	783	1262	1932388.290685	2
3	779	1164	1815673.093232	3
4	784	1036	1666660.023576	4
5	781	1033	1657616.282084	5
6	780	1004	1596847.227451	6
7	793	797	1262950.162500	7

7. Erase the spBestSeller procedure.

Part 3 – Stored procedure

1. Create a procedure named spSalesPerYear that takes a year number (data type: int) as a parameter and displays all the data from that year's Order header table.

   Call the procedure and check that it is correct by running the following code:

```
exec spSalesPerYear 2013
exec spSalesPerYear 2014
```

2. Continuing on from the previous question, modify the spSalesPerYear procedure so it takes two parameters: year and customer number (data type: int) and returns all the data from the Order header table for the specified year and customer.

   Call the procedure and check that it is correct by running the following code:

```
exec spSalesPerYear 2013, 17767
exec spSalesPerYear 2014, 27386
```

3. Create a procedure called spProductList that takes the parameters detailed below and returns all products from the Product table that meet the requirements.

   The purpose of the parameters is to specify the desired data ranges for the types of products and their list prices.

   a. The parameters that the procedure takes:

      - A 2-character string: Check against the two left-hand characters in the ProductNumber column
      - From-list price: the minimum list price (inclusive) that will be included in the query results
      - Until- list price: the maximum list price (inclusive) that will be included in the query results

   b. The procedure will display the following columns:
   ProductID, ProductNumber, Name, ListPrice

   c. Call the procedure and check that it is correct by running the following code:

```
Exec spProductList 'bk', 200, 1500
```

Lesson 18–Stored Procedures & Triggers

Part 1 – Stored procedure using variables

1. Create a procedure called spCustSalesPerYear that takes a customer number and year as parameters, and returns as output variables: the amount of orders placed and the total orders for that year

2. To check if the procedure is correct, write the instructions defined in the following sections, and then mark and run all the code you wrote:

   a. Define two variables:

      - @vTotalOrders (data type: int)
      - @vTotalAmount (data type: decimal (10,2))

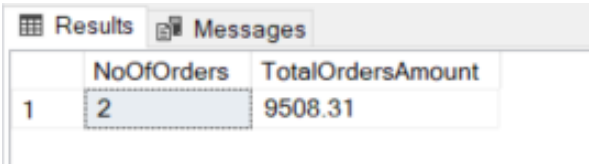
   b. Call the procedure and send it the following parameters:

      - Customer number – 29890
      - Year – 2011
      - The two parameters you defined in the previous section.

      Note that the last two parameters will function as output variables, so when sending them to the procedure you must add the code word: out

   c. Write a query showing the value of the two variables returned as output variables from the procedure

   d. Preview query results:



3. Create a procedure called spOrdersRangeAmount that takes parameters (detailed below) that produce a data range, and returns as an output variable the total amount of all incoming orders in the data range

   e. Parameters for creating the data range:

- @inFromDate - The start date (inclusive) of the data range
- @inToDate - The final (inclusive) date of the data range
- @inFromSubCategory - The minimum (including) subcategory code in the data range
- @inToSubCategory - The maximum (including) subcategory code in the data range
- @inColor - The color of the items that will fit into the data range

4. To check if the procedure is correct, write the instructions defined in the following sections, and then mark and run all the code you wrote:
   a. Define a variable named @vTotalAmount (given type: decimal (10,2))
   b. Call the procedure and send it the appropriate parameters so that it will run over the following ranges:
      - Date: 01/01/2012 to 31/12/2012
      - Subcategory code: 1 to 5
      - Product color: Yellow
   c. Write a query that shows the value of the variable returned as an output variable from the procedure
   d. Preview query results: