# Content:

[Content: 1](#_Toc376809040)

[Requires 2](#_Toc376809041)

[Exercise 1: Design a table (75') 3](#_Toc376809042)

[Exercise 2: Querying and Filtering Data (45') 4](#_Toc376809043)

[Exercise 3: Grouping and Summarizing Data (60') 14](#_Toc376809044)

Requires:

For the following exercises:

* Print out respectively the screenshots to show the query results.
* SQL scripts for 3 exercises.
* Pack screenshots and SQL scripts into the zip file named **Assignment2\_AccountName.zip**(for instance: Assignment2\_NamNT.zip)

# Exercise 1:Design a table (75')

In the design for the Fresher Training Management, given the Trainee table with below initial attributes / fields:

* TraineeID: Trainee identifier, auto increment field
* Full\_Name: Full name of the trainee
* Birth\_Date: Trainee birth date
* Gender: Only have one of 3 values**male, female,** and **unknown**.
* ET\_IQ: Entry test point (IQ) of trainee, integer, **value range from 0 to 20**
* ET\_Gmath: Entry test point (Gmath) of trainee, integer, **value range from 0 to 20**
* ET\_English: Entry test point (English) of trainee, integer, **value range from 0 to 50**
* Training\_Class: The class code that trainee is joining
* Evaluation\_Notes: Trainee evaluation notes, free text.

**Q1**:Create the tables (with the most appropriate/economic field/column constraints & types).

**Q2**:Change the table TRAINEE to add one more field named Fsoft\_Account which is a not-null &unique field.

**Q3**: Add at least 10 records into created table.

**Q4**:Create a VIEW which includes all the ET-passed trainees. One trainee is considered as ET-passed when he/she has the entry test points satisfied below criteria:

* ET\_IQ + ET\_Gmath>=20
* ET\_IQ>=8
* ET\_Gmath>=8
* ET\_English>=18

**Q5**:Query the trainee who has the longest name, showing his/her age along with his/her basic information (as defined in the table)

# Exercise 2: Querying and Filtering Data (45')

This exercise performs on AdventureWorks2008 database that included in the same folder with the assignment.

|  |  |  |  |
| --- | --- | --- | --- |
| **Note** | A series of dots (…………) in result sets indicates that rows have been excluded for presentational reasons. | | |
| **Query 1** | Write a query that retrieves the columns ProductID, Name, Color and ListPrice from the Production.Product table, with no filter. Your result set should look something like the following.  **ProductID Name Color ListPrice**  ----------- ------------------------------ --------------- -----------  1 Adjustable Race NULL 0,00  2 Bearing Ball NULL 0,00  3 BB Ball Bearing NULL 0,00  4 Headset Ball Bearings NULL 0,00  316 Blade NULL 0,00  …………  995 ML Bottom Bracket NULL 101,24  996 HL Bottom Bracket NULL 121,49  997 Road-750 Black, 44 Black 539,99  998 Road-750 Black, 48 Black 539,99  999 Road-750 Black, 52 Black 539,99  (504 row(s) affected) | | |
|  |  | | |
| **Query 2** | Continue to work with the previous query and exclude those rows that are 0 for the column ListPrice. Your result set should look something like the following.  **ProductID Name Color ListPrice**  ----------- ------------------------------ --------------- ----------  514 LL Mountain Seat Assembly NULL 133,34  515 ML Mountain Seat Assembly NULL 147,14  516 HL Mountain Seat Assembly NULL 196,92  517 LL Road Seat Assembly NULL 133,34  518 ML Road Seat Assembly NULL 147,14  …………  997 Road-750 Black, 44 Black 539,99  998 Road-750 Black, 48 Black 539,99  999 Road-750 Black, 52 Black 539,99  (304 row(s) affected) | | |
|  |  | | |
| **Query 3** | Use the same query, but this time you just want to see the rows that are NULL for the Color column. Your result set should look something like the following.  **ProductID Name Color ListPrice**  ----------- ------------------------------ - -------------- -----------  1 Adjustable Race NULL 0,00  2 Bearing Ball NULL 0,00  3 BB Ball Bearing NULL 0,00  4 Headset Ball Bearings NULL 0,00  316 Blade NULL 0,00  ………  947 HL Touring Handlebars NULL 91,57  994 LL Bottom Bracket NULL 53,99  995 ML Bottom Bracket NULL 101,24  996 HL Bottom Bracket NULL 121,49  (248 row(s) affected) | | |
|  |  | | |
| **Query 4** | Use the same query, but this time you just want to see the rows that *are not* NULL for the Color column. Your result set should look something like the following.  **ProductID Name Color ListPrice**  ----------- ------------------------------ --------------- -----------  317 LL Crankarm Black 0,00  318 ML Crankarm Black 0,00  319 HL Crankarm Black 0,00  320 Chainring Bolts Silver 0,00  ………  992 Mountain-500 Black, 48 Black 539,99  993 Mountain-500 Black, 52 Black 539,99  997 Road-750 Black, 44 Black 539,99  998 Road-750 Black, 48 Black 539,99  999 Road-750 Black, 52 Black 539,99  (256 row(s) affected) | | |
|  |  | | |
| **Query 5** | Now, combine two search arguments in the query you have been working with. You just want to see the rows that *are not* NULL for the column Color, and the column ListPricehas a value greater than zero. Your result set should look something like the following.  **ProductID Name Color ListPrice**  ----------- ------------------------------ --------------- ----------  680 HL Road Frame - Black, 58 Black 1431,50  706 HL Road Frame - Red, 58 Red 1431,50  707 Sport-100 Helmet, Red Red 34,99  708 Sport-100 Helmet, Black Black 34,99  ………  993 Mountain-500 Black, 52 Black 539,99  997 Road-750 Black, 44 Black 539,99  998 Road-750 Black, 48 Black 539,99  999 Road-750 Black, 52 Black 539,99  (245 row(s) affected) | | |
|  |  | | |
| **Query 6** | Now we want a report that concatenates the columns Name and Color from the Production.Product table. Your result set should look something like the following. Make sure you exclude rows that are NULL for the column Color. Also notice the column name.  **Name And Color**  ------------------------------------------  LL Crankarm : Black  ML Crankarm : Black  HL Crankarm : Black  Chainring Bolts : Silver  Chainring Nut : Silver  Chainring : Black  ………  Mountain-500 Black, 44 : Black  Mountain-500 Black, 48 : Black  Mountain-500 Black, 52 : Black  Road-750 Black, 44 : Black  Road-750 Black, 48 : Black  Road-750 Black, 52 : Black  (256 row(s) affected) | | |
|  |  | | |
| **Query 7** | Customize the previous query so the answer looks like the following.  **Name And Color**  --------------------------------------------------  NAME: LL Crankarm -- COLOR: Black  NAME: ML Crankarm -- COLOR: Black  NAME: HL Crankarm -- COLOR: Black  NAME: Chainring Bolts -- COLOR: Silver  NAME: Chainring Nut -- COLOR: Silver  NAME: Chainring -- COLOR: Black  ………  NAME: Mountain-500 Black, 48 -- COLOR: Black  NAME: Mountain-500 Black, 52 -- COLOR: Black  NAME: Road-750 Black, 44 -- COLOR: Black  NAME: Road-750 Black, 48 -- COLOR: Black  NAME: Road-750 Black, 52 -- COLOR: Black  (256 row(s) affected) | | |
|  |  | | |
| **Query 8** | Now we would like to see the columns ProductID and Name from the Production.Product table filtered by ProductID from 400 to 500. Write a query that makes your result set look something like the following. Try to make your WHERE clause as simple and readable as possible.  ProductID Name  ----------- -----------------------------  400 LL Hub  401 HL Hub  402 Keyed Washer  403 External Lock Washer 3  404 External Lock Washer 4  405 External Lock Washer 9  406 External Lock Washer 5  ………  494 Paint - Silver  495 Paint - Blue  496 Paint - Yellow  497 Pinch Bolt  (98 row(s) affected) | | |
|  |  | | |
| **Query 9** | We would like to see the columns ProductID, Name and color from the Production.Product table restricted to the colors black and blue. Write a query that makes your result set look something like the following. Try to make your WHERE clause as simple and readable as possible.  ProductID Name Color  ----------- ----------------------------------- --------  317 LL Crankarm Black  318 ML Crankarm Black  319 HL Crankarm Black  322 Chainring Black  680 HL Road Frame - Black, 58 Black  708 Sport-100 Helmet, Black Black  711 Sport-100 Helmet, Blue Blue  722 LL Road Frame - Black, 58 Black  ………  992 Mountain-500 Black, 48 Black  993 Mountain-500 Black, 52 Black  997 Road-750 Black, 44 Black  998 Road-750 Black, 48 Black  999 Road-750 Black, 52 Black  (119 row(s) affected) | | |
|  | | | |
| **Query 10** | | **Wildcards**  This exercise and the next three following will make use of wildcards in Transact-SQL. To begin with, we would like a report on products that begins with the letter S.  Write a query that retrieves the columns Name and ListPrice from the Production.Product table. Your result set should look something like the following. Order the result set by the Name column.  Name ListPrice  -------------------------------------------------- -----------  Seat Lug 0,00  Seat Post 0,00  Seat Stays 0,00  Seat Tube 0,00  Short-Sleeve Classic Jersey, L 53,99  Short-Sleeve Classic Jersey, M 53,99  ………  Sport-100 Helmet, Blue 34,99  Sport-100 Helmet, Red 34,99  Steerer 0,00  Stem 0,00  (14 row(s) affected) | |
|  | | | |
| **Query 11** | | Now we would like a report on products that begins with the letters S or A. Write a query that retrieves the columns Name and ListPrice from the Production.Product table. Your result set should look something like the following. Order the result set by the Name column.  Name ListPrice  -------------------------------------------------- ----------  Adjustable Race 0,00  All-Purpose Bike Stand 159,00  AWC Logo Cap 8,99  Seat Lug 0,00  Seat Post 0,00  ………  Sport-100 Helmet, Red 34,99  Steerer 0,00  Stem 0,00  (17 row(s) affected) | |
|  | | | |
| **Query 12** | | Adjust your query so you retrieve rows that have a Name that begins with the letters SPO, but is then *not* followed by the letter K. After this zero or more letters can exists. Order the result set by the *Name* column.  **Name ListPrice**  -------------------------------------------------- -----------  Sport-100 Helmet, Black 34,99  Sport-100 Helmet, Blue 34,99  Sport-100 Helmet, Red 34,99  (3 row(s) affected) | |
|  | | | |
| **Query 13** | | Write a query that retrieves *unique* colors from the table Production.Product. We do not want to see all the rows, just what colors that exist in the column Color. Your result set should look something like the following.  **Color**  ---------------  NULL  Black  Blue  Grey  Multi  Red  Silver  Silver/Black  White  Yellow  (10 row(s) affected) | |
|  | | | |
| **Query 14** | | Write a query that retrieves the unique combination of columns ProductSubcategoryID and Color from the Production.Product table. Format and sort so the result set accordingly to the following. We do not want any rows that are NULL.in any of the two columns in the result.  ProductSubcategoryID Color  -------------------- ---------------  1 Silver  1 Black  2 Yellow  2 Red  2 Black  3 Yellow  …………  31 Red  31 Blue  31 Black  32 Silver  35 Grey  (34 row(s) affected) | |
| **Query 15** | | Something is “wrong” with the WHERE clause in the following query.  We do not want any Red or Black products from any SubCategory than those with the value of 1 in column ProductSubCategoryID, unless they cost between 1000 and 2000.  SELECT ProductSubCategoryID  , LEFT([Name],35) AS [Name]  , Color, ListPrice  FROM Production.Product  WHERE Color IN ('Red','Black')  OR ListPrice BETWEEN 1000 AND 2000  AND ProductSubCategoryID = 1  ORDER BY ProductID  Write the query in the editor and execute it. Take a look at the result set and then adjust the query so it delivers the following result set.   * **Tip:**   Operator precedence is often a source of confusion. | |
| **ProductSubCategoryID Name Color ListPrice**  -------------------- ----------------------------------- --------------- ---------  14 HL Road Frame - Black, 58 Black 1431,50  14 HL Road Frame - Red, 58 Red 1431,50  14 HL Road Frame - Red, 62 Red 1431,50  14 HL Road Frame - Red, 44 Red 1431,50  14 HL Road Frame - Red, 48 Red 1431,50  14 HL Road Frame - Red, 52 Red 1431,50  14 HL Road Frame - Red, 56 Red 1431,50  12 HL Mountain Frame - Silver, 42 Silver 1364,50  12 HL Mountain Frame - Silver, 44 Silver 1364,50  12 HL Mountain Frame - Silver, 48 Silver 1364,50  ......  2 Road-350-W Yellow, 44 Yellow 1700,99  2 Road-350-W Yellow, 48 Yellow 1700,99  1 Mountain-500 Black, 40 Black 539,99  1 Mountain-500 Black, 42 Black 539,99  1 Mountain-500 Black, 44 Black 539,99  1 Mountain-500 Black, 48 Black 539,99  1 Mountain-500 Black, 52 Black 539,99  (63 row(s) affected) | | | |
|  | | | |
| **Exercise 16** | | | Use the Production.Product table to return product name, color and list price for each product. For the color column, where there is NULL, replace it with the string *Unknown*.  Name Color ListPrice  ------------------------------ --------------- ----------  Adjustable Race Unknown 0,00  Bearing Ball Unknown 0,00  BB Ball Bearing Unknown 0,00  Headset Ball Bearings Unknown 0,00  .....  HL Bottom Bracket Unknown 121,49  Road-750 Black, 44 Black 539,99  Road-750 Black, 48 Black 539,99  Road-750 Black, 52 Black 539,99  (504 row(s) affected) |

# Exercise 3: Grouping and Summarizing Data (60')

This exercise performs on AdventureWorks2008 database that included in the same folder with the assignment

|  |  |
| --- | --- |
| **Query 1** | How many products can you find in the Production.Product table? Your result set should look like the following.  -----------  504  (1 row(s) affected) |
|  |  |
| **Query 2** | Write a query that retrieves the number of products in the Production.Product table that are included in a subcategory. The rows that have NULL in column ProductSubcategoryID are considered to not be a part of any subcategory.  **HasSubCategoryID**  ----------------  295  (1 row(s) affected)  Notice that the result has a column name. Also take a look at the message you receive from SQL Server |
|  |  |
| **Query 3** | How many Products reside in each SubCategory?  The answer to this is retrievable if you write a query that use the COUNT aggregate function combined with a GROUP BY clause.  The column ProductSubcategoryID is a candidate for building groups of rows when querying the Production.Product table. Your result set should look something like the result below.  Notice the column alias for the second column.  ProductSubcategoryID CountedProducts  -------------------- ---------------  NULL 209  1 32  2 43  3 22  4 8  ………  34 1  35 1  36 2  37 11  (38 row(s) affected) |
|  |  |
| **Query 4** | Try to write two different queries to find out how many products that do not have a product subcategory. One query without the WHERE clause and one query using a WHERE clause. The rows that have NULL in column ProductSubcategoryID are considered to not be a part of any subcategory.  **Tip**: In the first we can use a difference between expressions to calculate the figure.  -----------  209  (1 row(s) affected)  The other way using a WHERE clause. Observe the column alias.  **NoSubCat**  -----------  209  (1 row(s) affected) |
|  |  |
| **Query 5** | A report is needed, the summary of products in stock. Write a query against another table this time, the Production.ProductInventory table.  ProductID TheSum  ----------- -----------  1 1085  2 1109  3 1352  4 1322  316 1361  ………  997 153  998 155  999 194  (432 row(s) affected) |
| **Query 6** | Continue to write on the query in previous exercise. Add a WHERE clause that extracts the rows that have the column LocationID set to 40 and limit the result to include just summarized quantities less then 100.  Tip: now is the time to see the HAVING clause in action.  ProductID TheSum  ----------- -----------  492 16  493 28  494 4  495 25  496 44  814 96  943 88  (7 row(s) affected) |
|  |  |
| **Query 7** | In this query we also want to see what shelf the product is to be delivered from. Add code to the previous query.  **Shelf ProductID TheSum**  ---------- ----------- -----------  B 492 16  B 493 28  B 494 4  B 495 25  B 496 44  N/A 814 96  N/A 943 88  (7 row(s) affected) |
|  |  |
| **Query 8** | We would like to see the average quantity for products where column LocationID has the value of 10. The table Production.ProductInventory has the answer.  **TheAvg**  -----------  295  (1 row(s) affected) |
|  |  |
| **Query 9** | To continue to write on the previous query, we would like to see the result by shelf excluding rows that has the value of N/A in the column Shelf. We also want to see a total average based on shelf only and also for all products (“grand total”).  **ProductID Shelf TheAvg**  ----------- ---------- -----------  NULL NULL 328  476 A 404  477 A 353  478 A 622  NULL A 459  316 B 388  398 B 404  …………  813 E 251  946 E 248  947 E 244  NULL E 255  327 F 443  NULL F 443  482 L 176  483 L 459  484 L 196  485 L 176  486 L 457  487 L 324  488 L 305  NULL L 299  (47 row(s) affected) |
|  |  |
| **Query 10** | We want to know number of members (rows) and average list price in the Production.Product table. This should be grouped independently over the Color and the Class column. We are not interested in any rows where Color nor Class are null (WHERE Class IS NOT NULL AND Color IS NOT NULL).  Hint: think GROUPING SETS.  **Color Class TheCountAvgPrice**  --------------- ----- ----------- ---------------------  NULL H 71 1925,146  NULL L 76 461,2811  NULL M 52 809,6705  Black NULL 72 917,7209  Blue NULL 22 1081,3713  Red NULL 37 1438,8948  Silver NULL 30 1202,065  Silver/Black NULL 6 61,19  Yellow NULL 32 1072,229  (9 row(s) affected) |
|  |  |
| **Query 11** | We now want to examine the function GROUPING. The following query generates the result below the query itself. Take a look and complete the query so it results to the second result set.  SELECT ProductSubcategoryID  , COUNT(Name) as Counted  FROM Production.Product  GROUP BY ROLLUP (ProductSubcategoryID)  **ProductSubcategoryID Counted**  -------------------- -----------  NULL 209  1 32  2 43  3 22  4 8  …………  32 1  33 3  34 1  35 1  36 2  37 11  NULL 504  (39 row(s) affected)  Below you find the second result set, continue to write a complete query so it gives the following answer. The added column is for clarity regarding NULL.  **ProductSubcategoryID Counted IsGrandTotal**  -------------------- ----------- -----------  NULL 209 0  NULL 504 1  1 32 0  2 43 0  3 22 0  ……  34 1 0  35 1 0  36 2 0  37 11 0  (39 row(s) affected) |