

*Advanced SQL*

Training Assignments

|  |  |
| --- | --- |
| Program Code | ASQL |
| Version | 3.1 |
| Effective Date | 01/11/2016 |

**Hanoi, 11/2016**

RECORD OF CHANGES

\*A - Added M - Modified D - Deleted

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Changes | A\* M, D | Contents | Version |
| 14-Oct-2016 | Create | A | Add the new assignments. | v1.0 |
| 14-Oct-2018 | Update | M | Template. | v1.1 |
| 01-Jun-2019 | Update | M | Update Objective | v1.2 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Contents

[For the following assignments: 4](#_Toc17276067)

[Day 1. Unit 1: Advanced DML Statements 4](#_Toc17276068)

[Assignment 1\_Opt5: AdventureWorks2008 4](#_Toc17276069)

[Exercise1: Working with Subqueries (90') 4](#_Toc17276070)

[Exercise 2: Joining Data from multiple tables (90') 13](#_Toc17276071)

|  |  |
| --- | --- |
|  | **CODE: ASQL\_Assignment1\_Opt5**  **TYPE: Medium**  **LOC: n/a**  **DURATION: 180 MINUTES** |

# For the following assignments:

* Print out respectively the screenshots to show the query results.
* Pack screenshots and SQL scripts or your answers into the zip file named ASQL\_Assignment1\_AccountName.zip (for instance: ASQL\_Assignment1\_NamNT.zip) then handle to the evaluator via email ([XYZ@fsoft.com.vn](mailto:XYZ@fsoft.com.vn) ) or follow the guidance of the class admin.

# Day 1. Unit 1: Advanced DML Statements

## Assignment 1\_Opt5: AdventureWorks2008

**Barems**: Exercise 1 - 40%, Exercise 2 - 60%

**Objectives**: H5SD - SQL skills

**Problem Descriptions**:

Exercise1: Working with Subqueries (90')

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

|  |  |
| --- | --- |
| **Query 1** | Write a query that filters data and return the column “Name” from table Production.Product. The filtering of rows is achieved by a WHERE clause that compares a single value from a subquery.  The inner subquery shall return a specific ProductSubcategoryID that the outer query uses as a filter of products to include in the report. The inner query will use its own WHERE clause to deliver its value, the ProductSubcategoryID, by retrieving it where the column “Name” in table Production.ProductSubcategory have the value of ‘Saddles’.  The result set should look like the following.  Name  -----------------------------  LL Mountain Seat/Saddle  ML Mountain Seat/Saddle  HL Mountain Seat/Saddle  LL Road Seat/Saddle  ML Road Seat/Saddle  HL Road Seat/Saddle  LL Touring Seat/Saddle  ML Touring Seat/Saddle  HL Touring Seat/Saddle  (9 row(s) affected) |
|  |  |
| **Query 2** | In this exercise you can change the previous query to deliver the following result set. The WHERE clause in the subquery will now use the wildcard string ‘Bo%’ for a comparison.  The result set should look like the following.  Name  ----------------------------  Water Bottle - 30 oz.  Mountain Bottle Cage  Road Bottle Cage  LL Bottom Bracket  ML Bottom Bracket  HL Bottom Bracket  (6 row(s) affected) |
|  |  |
| **Query 3** | Write a query that return all products that has the same price as the cheapest (lowest ListPrice) Touring Bike (ProductSubcategoryID = 3). Use the MIN() aggregate function in the subquery to return the lowest ListPrice to the outer query.  The result set should look like the following.  Name  --------------------------  Touring-3000 Blue, 54  Touring-3000 Blue, 58  Touring-3000 Blue, 62  ………  Touring-3000 Yellow, 62  Touring-3000 Blue, 44  Touring-3000 Blue, 50  (10 row(s) affected) |
|  |  |
| **Query 4** | **Part 1:**  A list of countries (table: Person.CountryRegion, column: Name) that hosts less than ten instances of StateProvince in table Person.StateProvince is what your boss wants. Write a query that satisfies your boss.  **Tip**: a subquery using HAVING clause and aggregate function COUNT() can do the job.  The result set should look like the following.  Name  ---------------------------  American Samoa  Australia  Germany  Micronesia  United Kingdom  Marshall Islands  Northern Mariana Islands  Palau  Virgin Islands, U.S.  (9 row(s) affected)  **Part 2:**  Rewrite the query as a JOIN, the same result set should be retrieved. Here HAVING and COUNT() are as useful as in the previous query. |
|  |  |
| **Query 5** | In this query we will put a subquery in an expression in the SELECT list. We would like to see a report on how the corporate salesmen are doing in their history of business (although some parameters in this exercise might be missing for a correct result).  Aggregate the average from column SubTotal in table Sales.SalesOrderHeader as a subquery (pay attention to the NULL values in column SalesPersonID), then substractthe grouped averages by SalesPersonID in the outer query. Also, in the outer query NULL values can cause confusing results. Remember that NULL in the SalesPersonID column means Internet sales, and those sales are of no interest for this query.  The result set should look like the following.  SalesPersonID SalesDiff  ------------- ---------------------  284 2871,794  281 -7021,975  278 7022,1684  ………  286 -5717,4173  289 8503,1384  283 1659,0548  (17 row(s) affected) |
|  |  |
| **Query 6** | In this exercise we will build the final query in three steps. The final result will show which bicycles that costs 400 to 800 less than the average bike. The final query will make use of a derived table to give us the answer.  **Step 1:**  Find out the average ListPrice value in table Production.Product. Restrict the rows you work on to values 1, 2 and 3 in the column ProductSubcategoryID.  The intermediate result set should be:  ---------------------  1586,737  (1 row(s) affected)  **Step 2:**  Incorporate the entire previous query as a part of an expression in the new outer SELECT list you will write. Take column ListPrice from table Production.Product and subtract the previous query as the expression, give the new column the column alias ‘Diff’. Negative values indicate a cheaper bike.  The intermediate result set should look something like the following.  Name Diff  -------------------------------------------------- ----------  Road-150 Red, 62 1991,533  Road-150 Red, 44 1991,533  Road-150 Red, 48 1991,533  Road-150 Red, 52 1991,533  Road-150 Red, 56 1991,533  Road-450 Red, 58 -128,747  Road-450 Red, 60 -128,747  Road-450 Red, 44 -128,747  Road-450 Red, 48 -128,747  ………  Mountain-500 Black, 48 -1046,747  Mountain-500 Black, 52 -1046,747  Road-750 Black, 44 -1046,747  Road-750 Black, 48 -1046,747  Road-750 Black, 52 -1046,747  (97 row(s) affected)  **Step 3:**  Now, the final query.  Wrap parentheses around the previous query and turn it into a derived table, give it the alias ‘X’. Query the derived table for all its columns and use BETWEEN to extract bicycles that are in the interval 400 to 800, remember, negative values indicates cheaper. Give the BETWEEN a reflection of how it works, from the smallest to the largest value.  The final result set should look like the following.  Name Diff  -------------------------------------------------- -----------  Mountain-300 Black, 38 -506,747  Mountain-300 Black, 40 -506,747  Mountain-300 Black, 44 -506,747  Mountain-300 Black, 48 -506,747  Road-550-W Yellow, 38 -466,247  Road-550-W Yellow, 40 -466,247  Road-550-W Yellow, 42 -466,247  Road-550-W Yellow, 44 -466,247  Road-550-W Yellow, 48 -466,247  (9 row(s) affected)  You might also want to try writing this using a Common Table Expression (CTE, using WITH), instead of a derived table. |
|  |  |
| **Query 7** | In this exercise we will examine the correlated subquery. First you use a join and then a correlated subquery to do the same work. As a final part of this exercise you will compare execution plans between the two queries. Remember if not ORDER BY is present in the query; any arbitrary order will be chosen by SQL Server.  **Part 1:**  The following query reports the salesmen that had more than 5000 in bonus. Write and execute the query, save it for further use.  SELECT P.FirstName + ' ' + P.LastName  FROM Sales.SalesPerson SP  JOIN HumanResources.Employee E  ON E.BusinessEntityID = SP.BusinessEntityID  JOIN Person.Person AS P  ON E.BusinessEntityID = P.BusinessEntityID  WHERE Bonus > 5000  The result:  ---------------------  Tsvi Reiter  Jae Pak  Lynn Tsoflias  (3 row(s) affected)  **Part 2:**  Rewrite the query and use a correlated subquery instead. When you are finished, save the query for part 3 of this exercise.  **Part 3:**  Compare the both queries execution plans. What differs?  To do this, click the button “Include Actual Execution Plan”, and then the “Execute” button. When the queries are executed, you will see an extra tab in the result window. |
|  |  |
| **Query 8** | In this exercise you will write a correlated subquery using EXISTS as the first exercise. Then, as the second, the join equivalent query.  We would like a report on the salesmen that are not assigned to a store to cover. The table Sales.SalesPerson can be used as the outer query table, the table Sales.Store is a candidate for the inner query. In the table Sales.Store you have a column by the name SalesPersonID, a sales person encountered in the Sales.SalesPerson but not in the Sales.Store table (BusinessEntityID column) is of interest in this report.  The result should be:  SalesPersonID  -------------  268  284  287  288  (4 row(s) affected)  **Part 1:**  Write the correlated subquery using EXISTS.  **Part 2:**  Write the JOIN equivalent query |
|  |  |
| **Query 9** | You will write a report on counting products and their belongings in product categories and product subcategories. As a tool you will use the Common Table Expression. The exercise is divided into two parts before the final query is complete.  **Part 1:**  First you write a query that gather the result set following. Table Production.Product and the aggregate function COUNT() will be used.  ProductSubcategoryID  -------------------- -----------  NULL 209  1 32  2 43  3 22  ………  34 1  35 1  36 2  37 11  (38 row(s) affected)  **Part 2:**  Turn the previous query into a CTE by wrapping parentheses around it, name it “TempSet” and specify the column names as the syntax of CTE specifies. The CTE columns should have the names “ProdSubID” and “CountedProds”. Create the CTE and issue a SELECT \* FROM TempSet to check functionality.  It should look like  ProdSubID CountedProds  ----------- ------------  NULL 211  1 32  2 43  ………  35 1  36 2  37 11  (38 row(s) affected)  Then comment out the SELECT \* FROM TempSettestquery.  Now join the table Production.ProductSubcategory and the CTE by appropriate columns and use SUM() aggregate function to summarize the CTE column “CountedProds”. Make the Join an outer join to catch the CTE column ProdSubIDs value of NULL.  The final result set should look something like the following.  ProductCategoryID SubCat SumProds  ----------------- ----------- -----------  NULL 0 211  1 3 97  2 14 134  3 8 35  4 12 29  Warning: Null value is eliminated by an  aggregate or other SET operation.  (5 row(s) affected) |

Exercise 2: Joining Data from multiple tables (90')

|  |  |
| --- | --- |
| **Query 1** | Write a query that lists the country and province names stored in AdventureWorks2008sample database. In the Person schema you will find the CountryRegion and StateProvince tables. Join them and produce a result set similar to the following. Notice that there is no particular sort order in the result set.  Country Province  ------------------------------ ------------------------  CanadaAlberta  United StatesAlaska  United StatesAlabama  United StatesArkansas  American SamoaAmerican Samoa  ………  France Belford (Territoire de)  France Essonne  France Hauts de Seine  FranceSeine Saint Denis  France Val de Marne  France Val d'Oise  (181 row(s) affected) |
|  |  |
| **Query 2** | Continue to work with the previous query and add a filter to only list the countries Germany and Canada. Also notice the sort order and column headings of the result set. Your result set should look similar to the following.  Country Province  ------------------------------ ------------------------  Canada Alberta  Canada British Columbia  Canada Brunswick  CanadaLabrador  CanadaManitoba  CanadaNewfoundland  ………  GermanyBrandenburg  GermanyHamburg  Germany Hessen  Germany Nordrhein-Westfalen  GermanySaarland  GermanySaxony  (20 row(s) affected |
|  |  |
| **Query 3** | We want information about orders. From the Sales.SalesOrderHeader table we want the SalesOrderID, OrderDate and SalesPersonIDcolums. From the Sales.SalesPerson table we want the BusinessEntityID (which identifies the sales person), Bonus and the SalesYTD (how much this person sold for yet this year) columns.  (As an aside, note that joining SalesOrderHeader to SalesPerson will restrict the result to non-Internet orders (order processed on the Internet has 1 in the OnlineOrderFlag, and has NULL for the SalesPersonID column.)  Note that the time portion below has been removed from the OrderDate column for presentation purposes. |
| SalesOrderIDOrderDateSalesPersonIDBusinessEntityID Bonus SalesYTD  ------------ ----------- ------------- ---------------- ---------- ---------------------  43659 2001-07-01 279 279 6700,00 2811012,7151  43660 2001-07-01 279 279 6700,00 2811012,7151  43661 2001-07-01 282 282 5000,00 3189356,2465  43662 2001-07-01 282 282 5000,00 3189356,2465  43663 2001-07-01 276 276 2000,00 5200475,2313  43664 2001-07-01 280 280 5000,00 0,00  .....  71949 2004-06-01 277 277 2500,00 3857163,6332  71950 2004-06-01 279 279 6700,00 2811012,7151  71951 2004-06-01 279 279 6700,00 2811012,7151  71952 2004-06-01 275 275 4100,00 4557045,0459  (3806 row(s) affected) | |
|  |  |
| **Query 4** | Use above query, add JobTitle and remove the SalesPersonID and the BusinessEntityID columns. You need to join to the HumanResources.Employee table. |
| SalesOrderIDOrderDateJobtitle Bonus SalesYTD  ------------ ----------- -------------------- -------- -----------------  43659 2001-07-01 Sales Representative 6700.00 2811012,7151  43660 2001-07-01 Sales Representative 6700.00 2811012,7151  43661 2001-07-01 Sales Representative 5000.00 3189356,2465  43662 2001-07-01 Sales Representative 5000.00 3189356,2465  ......  71947 2004-06-01 Sales Representative 2500.00 3857163,6332  71948 2004-06-01 Sales Representative 6700.00 2811012,7151  71949 2004-06-01 Sales Representative 2500.00 3857163,6332  71950 2004-06-01 Sales Representative 6700.00 2811012,7151  71951 2004-06-01 Sales Representative 6700.00 2811012,7151  71952 2004-06-01 Sales Representative 4100.00 4557045,0459  (3806 row(s) affected) | |
|  |  |
| **Query 5** | Now use above query and join to the Person.Person table. Add the FirstName and LastName column and remove the JobTitle, and SalesYTD columns.  If you study the foreign key relationships between the tables, you might notice that there is no direct foreign key relationship between the Employee and the Person table. But there is an indirect relationship through the BusinessEntity table, since shince this is one-to-one relationship to both Person and Employee, we actually don’t need this table in our query.  SalesOrderIDOrderDateFirstNameLastName Bonus  ------------ ----------- ---------- --------------- --------  43659 2001-07-01 Tsvi Reiter 6700.00  43660 2001-07-01 Tsvi Reiter 6700.00  43661 2001-07-01 José Saraiva 5000.00  43662 2001-07-01 José Saraiva 5000.00  43663 2001-07-01 Linda Mitchell 2000.00  .........  71946 2004-06-01 José Saraiva 5000.00  71947 2004-06-01 Jillian Carson 2500.00  71948 2004-06-01 Tsvi Reiter 6700.00  71949 2004-06-01 Jillian Carson 2500.00  71950 2004-06-01 Tsvi Reiter 6700.00  71951 2004-06-01 Tsvi Reiter 6700.00  71952 2004-06-01 Michael Blythe 4100.00  (3806 row(s) affected) |
|  |  |
| **Query 6** | Since we don’t return any columns from the Employee table, and the relationships for the BusinessEntityID column are one-to-one, we actually don’t need the Employee table in the query. Re-write above query to that you remove the Employee table from the query, and make sure that the result is the same as from above query. |
|  |  |
| **Query 7** | Now we don’t want the Bonus column anymore. Again, we can remove table from the query, because we have one-to-one relationships. Remove the Bonus column and the references to the SalesPerson table.  SalesOrderIDOrderDateFirstNameLastName  ------------ ----------- ---------- ---------------  43659 2001-07-01 Tsvi Reiter  43660 2001-07-01 Tsvi Reiter  43661 2001-07-01 José Saraiva  43662 2001-07-01 José Saraiva  ......  71948 2004-06-01 Tsvi Reiter  71949 2004-06-01 Jillian Carson  71950 2004-06-01 Tsvi Reiter  71951 2004-06-01 Tsvi Reiter  71952 2004-06-01 Michael Blythe  (3806 row(s) affected) |
|  |  |
| **Query 8** | We also want to see order details information. Use above query and join it to the Sales.SalesOrderDetail table, from which you return the ProductIDandOrderQty column. Also, concatenate the FirstName and LastName columns into one column named SalesPerson. Order the rows by the OrderDate, SalesOrderID columns.  SalesOrderIDOrderDateSalesPersonProductIDOrderQty  ------------ ----------- ----------------- ----------- --------  43659 2001-07-01 Tsvi Reiter 776 1  43659 2001-07-01 Tsvi Reiter 777 3  43659 2001-07-01 Tsvi Reiter 778 1  43659 2001-07-01 Tsvi Reiter 771 1  43659 2001-07-01 Tsvi Reiter 772 1  43659 2001-07-01 Tsvi Reiter 773 2  ......  71952 2004-06-01 Michael Blythe 910 4  71952 2004-06-01 Michael Blythe 924 4  71952 2004-06-01 Michael Blythe 926 5  71952 2004-06-01 Michael Blythe 920 2  71952 2004-06-01 Michael Blythe 743 1  71952 2004-06-01 Michael Blythe 742 4  71952 2004-06-01 Michael Blythe 994 3  71952 2004-06-01 Michael Blythe 985 3  (60919 row(s) affected) |
|  |  |
| **Query 9** | Now we want the name of the product instead of the ProductID column. You can get this by joining to the Production.Product table. |
| SalesOrderIDOrderDateSalesPersonProductNameOrderQty  ------------ ----------- ---------------- ------------------------------- --------  43659 2001-07-01 Tsvi Reiter Mountain-100 Black, 42 1  43659 2001-07-01 Tsvi Reiter Mountain-100 Black, 44 3  43659 2001-07-01 Tsvi Reiter Mountain-100 Black, 48 1  43659 2001-07-01 Tsvi Reiter Mountain-100 Silver, 38 1  43659 2001-07-01 Tsvi Reiter Mountain-100 Silver, 42 1  ......  71952 2004-06-01 Michael Blythe LL Mountain Frame - Black, 48 5  71952 2004-06-01 Michael Blythe LL Mountain Frame - Silver, 52 2  71952 2004-06-01 Michael Blythe HL Mountain Frame - Black, 42 1  71952 2004-06-01 Michael Blythe HL Mountain Frame - Silver, 46 4  71952 2004-06-01 Michael Blythe LL Bottom Bracket 3  71952 2004-06-01 Michael Blythe Mountain-500 Silver, 42 3  (60919 row(s) affected) | |
|  |  |
| **Query 10** | Now you want to use above and limit so you:  1. Only see order with order value over 100 000 (SubTotal column in order header table)  2. Only see orders with order date of year 2004. You can either limit this using the DATEPART function to return the year, or you can use a range in the WHERE clause, as described in <http://www.karaszi.com/SQLServer/info_datetime.asp>, the “Searching for datetime values” section. |
| SalesOrderIDOrderDateSalesPersonProductNameOrderQty  ------------ ----------- ---------------- ------------------------------- --------  61184 2004-01-01 Shu Ito Short-Sleeve Classic Jersey, S 5  61184 2004-01-01 Shu Ito Touring-2000 Blue, 46 3  61184 2004-01-01 Shu Ito Touring-3000 Blue, 50 11  61184 2004-01-01 Shu Ito LL Touring Frame - Blue, 54 1  61184 2004-01-01 Shu Ito Touring-3000 Yellow, 54 5  61184 2004-01-01 Shu Ito Touring-2000 Blue, 60 4  61184 2004-01-01 Shu Ito Touring-1000 Blue, 60 8  ......  71847 2004-06-01 Jae Pak Touring-2000 Blue, 60 6  71847 2004-06-01 Jae Pak HL Touring Frame - Blue, 60 6  71847 2004-06-01 Jae Pak HL Touring Handlebars 3  71847 2004-06-01 Jae Pak Touring-2000 Blue, 46 4  71847 2004-06-01 Jae Pak Touring-1000 Yellow, 50 2  71847 2004-06-01 Jae Pak HL Touring Frame - Yellow, 60 4  (695 row(s) affected) | |
| **Query 11** | We want to see information about countries and provinces. Join the CountryRegion and the StateProvince tables (both in the Person schema). Note that we want to keep the countries for which there are no provinces! Sort the result on country name and province name.  CountryName ProvinceName  ------------------------- --------------------  Afghanistan NULL  Albania NULL  Algeria NULL  American Samoa American Samoa  Andorra NULL  Angola NULL  ......  Armenia NULL  Aruba NULL  Australia New South Wales  Australia Queensland  Australia South Australia  Australia Tasmania  Australia Victoria  ......  Virgin Islands, British NULL  Virgin Islands, U.S. Virgin Islands  Wallis and Futuna NULL  Yemen NULL  Zambia NULL  Zimbabwe NULL  (407 row(s) affected) |
|  |  |
| **Query 12** | Write a query that retrieves customers that have not yet placed an order. This can be done using an outer join because the customer exists in the Sales.Customer table but not in the Sales.SalesOrderHeader table. Your result set should look similar to the following.  CustomerID SalesOrderID  ----------- ------------  1 NULL  2 NULL  3 NULL  4 NULL  5 NULL  6 NULL  ......  697 NULL  698 NULL  699 NULL  700 NULL  701 NULL  (701 row(s) affected) |
|  |  |
| **Query 13** | Using a full join we can retrieve a result set listing products that have no product model name and product model names that is not assigned to a specific product. Write a query that delivers the following result set by using the tables Production.Product and Production.ProductModel  ProductName ProductModelName  ------------------------------ ---------------------  Adjustable Race NULL  Bearing Ball NULL  BB Ball Bearing NULL  Headset Ball Bearings NULL  Blade NULL  LL Crankarm NULL  ML Crankarm NULL  ………  NULL ML Road Seat/Saddle 1  NULL Road-350  NULL HL Mountain Seat/Saddle 1  NULL HL Road Seat/Saddle 1  NULL Mountain-400  NULL LL Mountain Seat/Saddle 1  NULL Road-550  NULL LL Road Seat/Saddle 2  (218 row(s) affected) |

.

**-- THE END --**