Lab Answer Key: Module 3: Writing SELECT Queries

Lab: Writing Basic SELECT Statements

Exercise 1: Writing Simple SELECT Statements

Task 1: Prepare the Lab Environment

Ensure that the **MT17B-WS2016-NAT**, **20761C-MIA-DC** and **20761C-MIA-SQL** virtual machines are both running, and then log on to **20761C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.

1. In the **D:\Labfiles\Lab03\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
2. In the **User Account Control** dialog box, click **Yes**.
3. When the script has finished, press Enter.

Task 2: View All the Tables in the ADVENTUREWORKS Database in Object Explorer

1. On the taskbar, click **Microsoft SQL Server Management Studio**.
2. In the **Connect to Server** dialog box, in the **Server name** box, type **MIA-SQL**, and then click **Options**.
3. Under **Connection Properties**, in the **Connect to database** list, click **<Browse server>**.
4. In the **Browse for Databases** dialog box, click **Yes**.
5. In the **Browse Server for Databases** dialog box, under **User Databases**, click **TSQL**, and then click **OK**.
6. In the **Connect to Server** dialog box, on the **Login** tab, in the **Authentication** list, click **Windows Authentication**, and then click **Connect**.
7. In Object Explorer, under **MIA-SQL**, expand **Databases**, expand **TSQL**, and then expand **Tables**.
8. Under **Tables**, notice that there are four table objects in the Sales schema:
   - Sales.Customers
   - Sales.OrderDetails
   - Sales.Orders
   - Sales.Shippers

Task 3: Write a Simple SELECT Statement That Returns All Rows and Columns from a Table

1. On the **File** menu, point to **Open**, and then click **Project/Solution**.
2. In the **Open Project** dialog box, navigate to the **D:\Labfiles\Lab03\Starter\Project** folder, and then double-click **Project.ssmssln**.
3. In Solution Explorer, expand **Queries**, and then double-click **Lab Exercise 1.sql**.
4. In the query window, highlight the statement **USE TSQL;**, and then click **Execute**.
5. In the query pane, after the **Task 2** description, type the following query:

```
SELECT *
FROM Sales.Customers;
```

6. Highlight the query you typed, and click **Execute**.
7. In the query pane, type the following code after the first query:

```
SELECT *
FROM
```

8. In Object Explorer, under **MIA-SQL**, under **Databases**, under **TSQL**, under **Tables**, click **Sales.Customers**.
9. Drag the selected table into the query pane, after the FROM clause. Add a semicolon to the end of the SELECT statement. Your finished query should look like this:

```
SELECT *FROM [Sales].[Customers];
```

10. Highlight the written query, and click **Execute**.

Task 4: Write a SELECT Statement That Returns Specific Columns

1. In Object Explorer, expand **Sales.Customers**, expand **Columns** and observe all the columns in the **Sales.Customers** table.
2. In the query pane, after the **Task 3** description, type the following query:

```
SELECT contactname, address, postalcode, city, country
FROM Sales.Customers;
```

3. Highlight the written query, and click **Execute**.
4. Observe the result. How many rows are affected by the last query? There are multiple ways to answer this question using SQL Server Management Studio. One way is to select the previous query and click **Execute**.

The total number of rows affected by the executed query is written in the Results pane under the **Messages** tab:

```
(91 row(s) affected)
```

Another way is to look at the status bar displayed below the Results pane. On the left side of the status bar, there is a message stating: “Query executed successfully.” On the right side, the total number of rows affected by the current query is displayed (91 rows).

Result: After this exercise, you should know how to create simple SELECT statements to analyze existing tables.

Exercise 2: Eliminating Duplicates Using DISTINCT

Task 1: Write a SELECT Statement That Includes a Specific Column

1. In Solution Explorer, double-click **Lab Exercise 2.sql**.
2. In the query window, highlight the statement **USE TSQL;**, and then click **Execute**.
3. In the query pane, after the **Task 1** description, type the following query:

```
SELECTcountry
FROM Sales.Customers;
```

4. Highlight the written query, and click **Execute**.
5. Observe that you have multiple rows with the same values. This occurs because the **Sales.Customers** table has multiple rows with the same value for the country column.

Task 2: Write a SELECT Statement That Uses the DISTINCT Clause

1. Highlight the previous query, and then on the **Edit** menu, click **Copy**.
2. In the query window, click the line after the **Task 2** description.
3. On the **Edit** menu, click **Paste**. You have now copied the previous query to the same query window after the task 2 description.
4. Modify the query by typing **DISTINCT** after the SELECT clause. Your query should look like this:

```
SELECT DISTINCT country
FROM Sales.Customers;
```

5. Highlight the written query, and click **Execute**.

6. Observe the result and answer these questions:

   How many rows did the query in task 1 return?

   *To answer this question, you can highlight the query written under the task 1 description, click **Execute**, and read the Results pane. (If you forgot how to access this pane, look at task 4 in exercise 1.) The number of rows affected by the query is 91.*

   How many rows did the query in task 2 return?

   *To answer this question, you can highlight the query written under the task 2 description, click **Execute**, and read the Results pane. The number of rows affected by the query is 21. This means that there are 21 distinct values for the **country** column in the **Sales.Customers** table.*

   Under which circumstances do the following queries against the **Sales.Customers** table return the same result?

```
SELECT city, region FROM Sales.Customers;
SELECT DISTINCT city, region FROM Sales.Customers;
```

   If all combinations of values in the city and region columns in the Sales.Customers table are unique, both queries would return the same number of rows. If they are not unique, the first query would return more rows than the second one with the DISTINCT clause.

   Is the DISTINCT clause applied to all columns specified in the query—or just the first column?

   The DISTINCT clause is always applied to all columns specified in the SELECT list. It is very important to remember that the DISTINCT clause does not just apply to the first column in the list.

Result: After this exercise, you should understand how to return only the different (distinct) rows in the result set of a query.

Exercise 3: Using Table and Column Aliases

Task 1: Write a SELECT Statement That Uses a Table Alias

1. In Solution Explorer, double-click **Lab Exercise 3.sql**.

2. In the query window, highlight the statement **USE TSQL;**, and click **Execute**.

3. In the query pane, after the **Task 1** description, type the following query:

```
SELECTc.contactname, c.contacttitle
FROM Sales.Customers AS c;
```

Tip: To use the IntelliSense feature when entering column names in a SELECT statement, you can use keyboard shortcuts. To enable IntelliSense, press Ctrl+Q+I. To list all the alias members, position your pointer after the alias and dot (for example, after “c.”) and press Ctrl+J.

4. Highlight the written query, and click **Execute**.

Task 2: Write a SELECT Statement That Uses Column Aliases

1. In the query pane, after the **Task 2** description, type the following query:

```
SELECTc.contactname AS Name, c.contacttitle AS Title, c.companyname AS [Company Name]
FROM Sales.Customers AS c;
```

Observe that the column alias **[Company Name]** is enclosed in square brackets. Column names and aliases with embedded spaces or reserved keywords must be delimited. This example uses square brackets as the delimiter, but you can also use the ANSI SQL standard delimiter of double quotes, as in “Company Name”.

2. Highlight the written query, and click **Execute**.

Task 3: Write a SELECT Statement That Uses Table and Column Aliases

1. In the query pane, after the **Task 3** description, type the following query:

```
SELECT p.productname AS [Product Name]
FROM Production.Products AS p;
```

2. Highlight the written query, and click **Execute**.

Task 4: Analyze and Correct the Query

1. Highlight the written query under the **Task 4** description, and click **Execute**.
2. Observe the result. Note that only one column is retrieved. The problem is that the developer forgot to add a comma after the first column name, so SQL Server treated the second word after the first column name as an

alias. For this reason, it is best practice to always use AS when specifying aliases—then it is easier to spot such errors.

3. Correct the query by adding a comma after the first column name. The corrected query should look like this:

```
SELECT city, country
FROM Sales.Customers;
```


Result: After this exercise, you will know how to use aliases for table and column names.

Exercise 4: Using a Simple CASE Expression

Task 1: Write a SELECT Statement

1. In Solution Explorer, double-click **Lab Exercise 4.sql**.
2. In the query window, highlight the statement **USE TSQL;**, and then click **Execute**.
3. In the query pane, after the **Task 1** description, type the following query:

```
SELECT p.categoryid, p.productname
FROM Production.Products AS p;
```

4. Highlight the written query, and click **Execute**.

Task 2: Write a SELECT Statement That Uses a CASE Expression

1. In the query pane, after the **Task 2** description, type the following:

```
SELECT p.categoryid, p.productname,
CASE
WHEN p.categoryid = 1 THEN 'Beverages'
WHEN p.categoryid = 2 THEN 'Condiments'
WHEN p.categoryid = 3 THEN 'Confections'
WHEN p.categoryid = 4 THEN 'Dairy Products'
WHEN p.categoryid = 5 THEN 'Grains/Cereals'
WHEN p.categoryid = 6 THEN 'Meat/Poultry'
WHEN p.categoryid = 7 THEN 'Produce'
WHEN p.categoryid = 8 THEN 'Seafood'
ELSE 'Other'
```

```
END AS categoryname
FROM Production.Products AS p;
```

This query uses a CASE expression to add a new column. Note that, when you have a dynamic list of possible values, you usually store them in a separate table. However, for this example, a static list of values is being supplied.

2. Highlight the written query, and click **Execute**.

Task 3: Write a SELECT Statement That Uses a CASE Expression to Differentiate Campaign-Focused Products

1. Highlight the previous query, and then on the **Edit** menu, click **Copy**.
2. In the query window, click the line after the **Task 3** description.
3. On the **Edit** menu, click **Paste**. You have now copied the previous query to the same query window after the task 3 description.
4. Add a new column using an additional CASE expression. Your query should look like this:

```
SELECT  p.categoryid, p.productname,
        CASE
                WHEN p.categoryid = 1 THEN 'Beverages'
                WHEN p.categoryid = 2 THEN 'Condiments'
WHEN p.categoryid = 3 THEN 'Confections'
                WHEN p.categoryid = 4 THEN 'Dairy Products'
                WHEN p.categoryid = 5 THEN 'Grains/Cereals'
                WHEN p.categoryid = 6 THEN 'Meat/Poultry'
                WHEN p.categoryid = 7 THEN 'Produce'
                WHEN p.categoryid = 8 THEN 'Seafood'
                ELSE 'Other'
        END AS categoryname,
        CASE
                WHEN p.categoryid IN (1, 7, 8) THEN 'Campaign Products'
                ELSE 'Non-Campaign Products'
        END AS iscampaign
FROM Production.Products AS p;
```

5. Highlight the written query, and click **Execute**.
6. In the result, observe that the first CASE expression uses the simple form, whereas the second uses the searched form.

Result: After this exercise, you should know how to use CASE expressions to write simple conditional logic.