Lab Answer Key: Module 5: Sorting and Filtering Data

Lab: Sorting and Filtering Data

Exercise 1: Write Queries that Filter Data Using a WHERE Clause

Task 1: Prepare the Lab Environment

1. Ensure that the **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**.
2. In the **D:\Labfiles\Lab05\Starter** folder, right-click **Setup.cmd** and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**, and then wait for the script to finish.
4. At the command prompt, when prompted, press any key.

Task 2: Write a SELECT Statement Using a WHERE Clause

1. Start SQL Server Management Studio and connect to the MIA-SQL database engine instance using Windows authentication.
2. On the **File** menu, point to **Open**, and then click **Project/Solution**.
3. In the **Open Project** dialog box, navigate to the **D:\Labfiles\Lab05\Starter\Project** folder, and then double-click **Project.ssmssln**.
4. In Solution Explorer, expand **Queries**, and then double-click **51 - Lab Exercise 1.sql**.
5. In the query pane, highlight the statement **USE TSQL;**, and then click **Execute**.
6. In the query pane, type the following query after the **Task 1** description:

```
SELECT
custid, companyname, contactname, address, city, country, phone
FROM Sales.Customers
WHERE
country = N'Brazil';
```

7. Highlight the query and click **Execute**.

   Note the use of the “N” prefix for the character literal ‘Brazil’. This prefix is used because the country column is a Unicode data type. When expressing a Unicode character literal, you need to specify the character “N” (for National) as a prefix. If the “N” is omitted, then the query may still run successfully. However, the safest

way is to include the “N” every time, to ensure the results are predictable. You will learn more about data types in the next module.

Task 3: Write a SELECT Statement Using an IN Predicate in the WHERE Clause

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

```
SELECT
custid, companyname, contactname, address, city, country, phone
FROM Sales.Customers
WHERE
country IN (N'Brazil', N'UK', N'USA');
```

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

Task 4: Write a SELECT Statement Using a LIKE Predicate in the WHERE Clause

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

```
SELECT
custid, companyname, contactname, address, city, country, phone
FROM Sales.Customers
WHERE
contactname LIKE N'A%';
```

2. Remember that the percent sign (%) wildcard represents a string of any size (including an empty string), whereas the underscore (_) wildcard represents a single character.
3. Highlight the written query and click **Execute**.

Task 5: Observe the T-SQL Statement Provided by the IT Department

1. Highlight the T-SQL statement provided under the **Task 4a** description, and click **Execute**.
2. Highlight the provided T-SQL statement, and on the toolbar, click **Edit**, and then click **Copy**.
3. In the query window, click the line after the **Task 4b** description, and on the toolbar, click **Edit**, and then click **Paste**.

4. Modify the query so that it looks like this:

```
SELECT
c.custid, c.companyname, o.orderid
FROM Sales.Customers AS c
LEFT OUTER JOIN Sales.Orders AS o ON c.custid = o.custid
WHERE
c.city = N'Paris';
```

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

6. Observe the result. Is it the same as that of the first SQL statement?

   The result is not the same. When you specify the predicate in the ON clause, the left outer join preserves all the rows from the left table (**Sales.Customers**) and adds only the matching rows from the right table (**Sales.Orders**), based on the predicate in the ON clause. This means that all the customers will show up in the output, but only the ones from Paris will have matching orders. When you specify the predicate in the WHERE clause, the query will filter only the Paris customers. So be aware that, when you use an outer join, the result of a query where the predicate is specified in the ON clause can differ from the result of a query in which the predicate is specified in the WHERE clause. (When using an INNER JOIN, the results are always the same.) This is because the ON predicate is matching—it defines which rows from the nonpreserved side to match to those from the preserved side. The WHERE predicate is a filtering predicate—if a row from either side doesn't satisfy the WHERE predicate, the row is filtered out.

Task 6: Write a SELECT Statement to Retrieve Customers Without Orders

1. In the query pane, type the following query after the **Task 5** description:

```
SELECT
c.custid, c.companyname
FROM Sales.Customers AS c
LEFT OUTER JOIN Sales.Orders AS o ON c.custid = o.custid
WHERE o.custid IS NULL;
```

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

It is important to note that, when you are looking for a NULL, you should use the IS NULL operator, not the equality operator. The equality operator will always return UNKNOWN when you compare something to a NULL. It will even return UNKNOWN when you compare two NULLs.

The choice of which attribute to filter from the nonpreserved side of the join is also important. You should choose an attribute that can have a NULL only when the row is an outer row (for example, a NULL originating from the base table). For this purpose, three cases are safe to consider:

- A primary key column. A primary key column cannot be NULL. Therefore, a NULL in such a column can only mean that the row is an outer row.
- A join column. If a row has a NULL in the join column, it is filtered out by the second phase of the join. So a NULL in such a column can only mean that it is an outer row.
- A column defined as NOT NULL. A NULL in a column that is defined as NOT NULL can only mean that the row is an outer row.

Result: After this exercise, you should be able to filter rows of data from one or more tables by using WHERE predicates with logical operators.

Exercise 2: Write Queries that Sort Data Using an ORDER BY Clause

Task 1: Write a SELECT Statement Using an ORDER BY Clause

1. In Solution Explorer, double-click **61 - Lab Exercise 2.sql**.
2. In the query pane, highlight the statement **USE TSQL;**, and click **Execute**.
3. In the query pane, type the following query after the **Task 1** description:

```
SELECT
c.custid, c.contactname, o.orderid, o.orderdate
FROM Sales.Customers AS c
INNER JOIN Sales.Orders AS o ON c.custid = o.custid
WHERE
o.orderdate >= '20080401'
ORDER BY
o.orderdate DESC, c.custid ASC;
```

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

Notice the date filter. It uses a literal (constant) of a date. SQL Server recognizes “20080401” as a character string literal, and not as a date and time literal. However, because the expression involves two operands of different types, one needs to be implicitly converted to the other’s type. In this example, the character string literal is converted to the column’s data type (DATETIME) because character strings are considered lower in terms of data type precedence—with respect to date and time data types. Data type precedence and working with date values are covered in detail in the next module.

Also notice that the character string literal follows the format “yyyymmdd”. Using this format is a best practice because SQL Server knows how to convert it to the correct date, regardless of the language settings.

Task 2: Apply the Needed Changes and Execute the T-SQL Statement

1. Highlight the written query under the **Task 2** description, and click **Execute**.
2. Observe the error message:

```
Invalid column name 'mgrlastname'.
```

3. This error occurred because the WHERE clause is evaluated before the SELECT clause and, at that time, the column did not have an alias. To fix this problem, you must use the source column name with the appropriate table alias. Modify the T-SQL statement to look like this:

```
SELECT
e.empid, e.lastname, e.firstname, e.title, e.mgrid,
m.lastname AS mgrlastname, m.firstname AS mgrfirstname
FROM HR.Employees AS e
INNER JOIN HR.Employees AS m ON e.mgrid = m.empid
WHERE
m.lastname = N'Buck';
```

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

Task 3: Order the Result by the firstname Column

1. Highlight the previous query, and on the **Edit** menu, click **Copy**.
2. In the query window, click the line after the **Task 3a** description, and on the **Edit** menu, click **Paste**.
3. Modify the T-SQL statement to remove the WHERE clause, and add an ORDER BY clause that uses the source column name of m.firstname. Your query should look like this:

```
SELECT
e.empid, e.lastname, e.firstname, e.title, e.mgrid,
m.lastname AS mgrlastname, m.firstname AS mgrfirstname
FROM HR.Employees AS e
INNER JOIN HR.Employees AS m ON e.mgrid = m.empid
ORDER BY
m.firstname;
```

4. Highlight the written query and click **Execute**.
5. Highlight the previous query, and on the **Edit** menu, click **Copy**.

6. In the query window, click the line after the **Task 3b** description, and on the **Edit** menu, click **Paste**.
7. Modify the ORDER BY clause so that it uses the alias for the same column (mgrfirstname). Your query should look like this:

```
SELECT
e.empid, e.lastname, e.firstname, e.title, e.mgrid,
m.lastname AS mgrlastname, m.firstname AS mgrfirstname
FROM HR.Employees AS e
INNER JOIN HR.Employees AS m ON e.mgrid = m.empid
ORDER BY
mgrfirstname;
```

8. Highlight the written query and click **Execute**.
9. Compare the results for Task 3a and 3b.
10. Why were you equally able to use a source column name or an alias column name?

Result: After this exercise, you should know how to use an ORDER BY clause.

Exercise 3: Write Queries that Filter Data Using the TOP Option

Task 1: Writing Queries That Filter Data Using the TOP Clause

1. In Solution Explorer, double-click **71 - Lab Exercise 3.sql**.
2. In the query pane, highlight the statement **USE TSQL;**, and then click **Execute**.
3. In the query pane, type the following query after the **Task 1** description:

```
SELECT TOP (20)
orderid, orderdate
FROM Sales.Orders
ORDER BY orderdate DESC;
```

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

Task 2: Use the OFFSET-FETCH Clause to Implement the Same Task

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

```
SELECT
orderid, orderdate
FROM Sales.Orders
ORDER BY orderdate DESC
OFFSET 0 ROWS FETCH FIRST 20 ROWS ONLY;
```

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

Remember that the OFFSET-FETCH clause was a new functionality in SQL Server 2012 and will not work in earlier versions. Unlike the TOP clause, the OFFSET-FETCH clause must be used with the ORDER BY clause.

Task 3: Write a SELECT Statement to Retrieve the Most Expensive Products

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

```
SELECT TOP (10) PERCENT
productname, unitprice
FROM Production.Products
ORDER BY unitprice DESC;
```

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

Implementing this task with the OFFSET-FETCH clause is possible but not easy because, unlike TOP, OFFSET-FETCH does not support a PERCENT option.

Result: After this exercise, you should have an understanding of how to apply the TOP option in the SELECT clause of a T-SQL statement.

Exercise 4: Write Queries that Filter Data Using the OFFSET-FETCH Clause

Task 1: OFFSET-FETCH Clause to Fetch the First 20 Rows

1. In Solution Explorer, double-click **81 - Lab Exercise 4.sql**.
2. In the query pane, highlight the statement **USE TSQL;**, and then click **Execute**.
3. In the query pane, type the following query after the **Task 1** description:

```
SELECT
custid, orderid, orderdate
FROM Sales.Orders
ORDER BY orderdate, orderid
OFFSET 0 ROWS FETCH FIRST 20 ROWS ONLY;
```

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

Task 2: Use the OFFSET-FETCH Clause to Skip the First 20 Rows

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

```
SELECT
custid, orderid, orderdate
FROM Sales.Orders
ORDER BY orderdate, orderid
OFFSET 20 ROWS FETCH NEXT 20 ROWS ONLY;
```

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

Task 3: Write a Generic Form of the OFFSET-FETCH Clause for Paging

1. The correct code is:

```
OFFSET (@pagenum - 1) * @pagesize ROWS FETCH NEXT @pagesize ROWS ONLY
```

2. To test the above expression, type the following query after the **Task 3** description:

```
DECLARE @pagenum int,
                @pagesize int;

SET @pagenum = 3;
SET @pagesize = 10;

SELECT
custid, orderid, orderdate
FROM Sales.Orders
ORDER BY orderdate, orderid
OFFSET (@pagenum - 1) * @pagesize ROWS FETCH NEXT @pagesize ROWS ONLY;
```

3. Highlight the query and click **Execute**.

4. Compare your results with the recommended results in file **D:\Labfiles\Lab05\Solution\84 - Lab Exercise 4-Task 3 Result**. Try changing the values for **@pagenum** and/or **@pagesize**, highlight the whole query (including the DECLARE and SET statements) and then click **Execute**.

Result: After this exercise, you will be able to use OFFSET-FETCH to work page-by-page through a result set returned by a SELECT statement.