Lab Answer Key: Module 4: Querying Multiple Tables

Lab: Querying Multiple Tables

Exercise 1: Writing Queries That Use Inner Joins

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\Lab04\Starter** folder, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**.
4. At the command prompt, type **y**, press Enter, wait for the script to finish, and then press any key.

Task 2: Write a SELECT Statement That Uses an Inner Join

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 **Connect**.
3. On the **File** menu, point to **Open**, and then click **Project/Solution**.
4. In the **Open Project** dialog box, browse to the **D:\Labfiles\Lab04\Starter\Project** folder, and then double-click **Project.ssmssln**.
5. In Solution Explorer, expand **Queries**, and then double-click **51 - Lab Exercise 1.sql**.
6. In the query window, highlight the statement **USE TSQL;**, and click **Execute**.
7. In the query pane, after the **Task 1** description, type the following query:

```
SELECT
p.productname, c.categoryname
FROM Production.Products AS p
INNER JOIN Production.Categories AS c ON p.categoryid = c.categoryid;
```

8. Highlight the written query, and click **Execute**.
9. Observe the result and answer these questions:
   - Which column did you specify as a predicate in the ON clause of the join? Why?

In this query, the categoryid column is the predicate. By intuition, most people would say that this is the predicate because the column exists in both input tables. By the way, using the same name for columns that contain the same data but in different tables is a good practice in data modeling. Another possibility is to check for referential integrity through primary and foreign key information using SQL Server Management Studio. If there are no primary or foreign key constraints, you will have to acquire information about the data model from the developer.

- Let us say that there is a new row in the Production.Categories table and this new product category does not have any products associated with it in the Production.Products table. Would this row be included in the result of the SELECT statement written under the task 1 description?

  No, because an inner join retrieves only the matching rows based on the predicate from both input tables. Since the new value for the categoryid is not present in the categoryid column in the Production.Products table, there would be no matching rows in the result of the SELECT statement.

Result: After this exercise, you should know how to use an inner join between two tables.

Exercise 2: Writing Queries That Use Multiple-Table Inner Joins

Task 1: Execute the T-SQL Statement

1. In Solution Explorer, double-click the **61 - Lab Exercise 2.sql** query.
2. In the query window, highlight the statement **USE TSQL;**, and then click **Execute**.
3. Under the **Task 1** description, highlight the written query, and click **Execute**.
4. Observe the error message:

   Ambiguous column name 'custid'.

   This error occurred because the custid column appears in both tables; you have to specify from which table you would like to retrieve the column values.

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

1. Highlight the previous query, and on the **Edit** menu, click **Copy**.
2. In the query window, click the line after the **Task 2** description, and on the **Edit** menu, click **Paste**.
3. Add the column prefix **Customers** to the existing query so that it looks like this:

```
SELECT
Customers.custid, contactname, orderid
```

```
FROM Sales.Customers
INNER JOIN Sales.Orders ON Customers.custid = Orders.custid;
```

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

Task 3: Change the Table Aliases

1. Highlight the previous query, and on the **Edit** menu, click **Copy**.
2. In the query window, click the line after the **Task 3** description, and on the **Edit** menu, click **Paste**.
3. Modify the **T-SQL** statement to use table aliases. Your query should look like this:

```
SELECT
c.custid, c.contactname, o.orderid
FROM Sales.Customers AS c
INNER JOIN Sales.Orders AS o ON c.custid = o.custid;
```

4. Highlight the written query and click **Execute**.
5. Compare these results with the **Task 2** results.
6. Modify the **T-SQL** statement to include a full source table name as the column prefix. Your query should now look like this:

```
SELECT
Customers.custid, Customers.contactname, Orders.orderid
FROM Sales.Customers AS c
INNER JOIN Sales.Orders AS o ON c.custid = o.custid;
```

7. Highlight the written query and click **Execute**.
8. Observe the error messages:

```
Msg 4104, Level 16, State 1, Line 57
The multipart identifier "Customers.custid" could not be bound.
Msg 4104, Level 16, State 1, Line 57
The multipart identifier "Customer.contactname" could not be bound.
Msg 4104, Level 16, State 1, Line 57
The multipart identifier "Orders.orderid" could not be bound.
```

You received these error messages as, because you are using a different table alias, the full source table name you are referencing as a column prefix no longer exists. Remember that the SELECT clause is

evaluated after the FROM clause, so you must use the table aliases when specifying columns in the SELECT clause.

9. Modify the SELECT statement so that it uses the correct table aliases. Your query should look like this:

```
SELECT
c.custid, c.contactname, o.orderid
FROM Sales.Customers AS c
INNER JOIN Sales.Orders AS o ON c.custid = o.custid;
```

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

Task 4: Add an Additional Table and Columns

1. In the query pane, after the **Task 4** description, type the following query:

```
SELECT
c.custid, c.contactname, o.orderid, d.productid, d.qty, d.unitprice
FROM Sales.Customers AS c
INNER JOIN Sales.Orders AS o ON c.custid = o.custid
INNER JOIN Sales.OrderDetails AS d ON d.orderid = o.orderid;
```

2. Highlight the written query, and click **Execute**.
3. Observe the result. Remember that, when you have a multiple-table inner join, the logical query processing is different from the physical implementation. In this case, it means that you cannot guarantee the order in which the SQL Server optimizer will process the tables. For example, you cannot guarantee that the Sales.Customers table will be joined first with the Sales.Orders table, and then with the Sales.OrderDetails table.

Result: After this exercise, you should have a better understanding of why aliases are important and how to do a multiple-table join.

Exercise 3: Writing Queries That Use Self Joins

Task 1: Write a Basic SELECT Statement

1. In Solution Explorer, double-click the **71 - Lab Exercise 3.sql** query.
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
e.empid, e.lastname, e.firstname, e.title, e.mgrid
FROM HR.Employees AS e;
```

4. Highlight the written query and click **Execute**.
5. Observe that the query retrieved nine rows.

Task 2: Write a Query That Uses a Self Join

1. Highlight the previous query, and on the **Edit** menu, click **Copy**.
2. In the query window, after the **Task 2** description, click the line, and on the **Edit** menu, click **Paste**.
3. Modify the query by adding a self join to get information about the managers. The 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;
```

4. Highlight the written query and click **Execute**.
5. Observe that the query retrieved eight rows and answer these questions:
   - Is it mandatory to use table aliases when writing a statement with a self join? Can you use a full source table name as an alias?

   You must use table aliases. You cannot use the full source table name as an alias when referencing both input tables. Eventually, you could use a full source table name as an alias for one input table and another alias for the second input table.

   - Why did you get fewer rows in the result from the T-SQL statement under the task 2 description, compared to the result from the T-SQL statement under the task 1 description?

   In task 2's T-SQL statement, the inner join used an ON clause based on manager information (column mgrid). The employee who is the CEO has a missing value in the mgrid column, so this row is not included in the result.

Result: After this exercise, you should have an understanding of how to write T-SQL statements that use self joins.

Exercise 4: Writing Queries That Use Outer Joins

Task 1: Write a SELECT Statement That Uses an Outer Join

1. In Solution Explorer, double-click the **81 - Lab Exercise 4.sql** query.
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
c.custid, c.contactname, o.orderid
FROM Sales.Customers AS c
LEFT OUTER JOIN Sales.Orders AS o ON c.custid = o.custid;
```

4. Highlight the written query and click **Execute**.
5. Inspect the result. Notice that the custid 22 and custid 57 rows have a missing value in the orderid column. This is because there are no rows in the Sales.Orders table for these two values of the custid column. In business terms, this means that there are currently no orders for these two customers.

Result: After this exercise, you should have a basic understanding of how to write T-SQL statements that use outer joins.

Exercise 5: Writing Queries That Use Cross Joins

Task 1: Execute the T-SQL Statement

1. In Solution Explorer, double-click the **91 - Lab Exercise 5.sql** query.
2. In the query window, highlight the statement **USE TSQL;**, and then click **Execute**.
3. Under the **Task 1** description, highlight the T-SQL code and click **Execute**. Don't worry if you do not understand the provided T-SQL code, as it is used here to provide a more realistic example for a cross join in the next task.

Task 2: Write a SELECT Statement That Uses a Cross Join

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

```
SELECT
e.empid, e.firstname, e.lastname, c.calendardate
FROM HR.Employees AS e
CROSS JOIN HR.Calendar AS c;
```

2. Highlight the written query and click **Execute**.
3. Observe that the query retrieved 3,294 rows and that there are nine rows in the HR.Employees table. Because a cross join produces a Cartesian product of both inputs, it means that there are 366 (3,294/9) rows in the HR.Calendar table.

Task 3: Drop the HR.Calendar Table

- Under the **Task 3** description, highlight the written query and click **Execute**.

Result: After this exercise, you should have an understanding of how to write T-SQL statements that use cross joins.