Module 3: Writing SELECT Queries

Contents:

This document belongs to Sebastian Amihaesie. sebastian.amihaesie@gmail.com No unauthorized copies allowed!

Module Overview

You can use the SELECT statement to query tables and views. It is likely that you will use the SELECT statement more than any other single statement in T-SQL. You can manipulate the data with SELECT to customize how SQL Server returns the results. This module introduces you to the fundamentals of the SELECT statement, focusing on queries against a single table.

Objectives

After completing this module, you will be able to:

- Write simple SELECT statements.
- Eliminate duplicates using the DISTINCT clause.
- Use table and column aliases.

- Write simple CASE expressions.

Lesson 1: Writing Simple SELECT Statements

In this lesson, you will learn the structure and format of the SELECT statement, in addition to enhancements that will add functionality and readability to your queries.

Lesson Objectives

At the end of this lesson, you will be able to:

- Understand the elements of the SELECT statement.
- Write simple SELECT queries against a single table.
- Eliminate duplicate rows using the DISTINCT clause.
- Add calculated columns to a SELECT statement.

Elements of the SELECT Statement

Clause	Expression
SELECT	<select list>
FROM	<table or view>
WHERE	<search condition>
GROUP BY	<group by list>
ORDER BY	<order by list>

The SELECT and FROM clauses are the primary focus of this module. You will learn about the other clauses in later modules of this course. You have already learned the order of operations in logical query processing; this will help you to understand how to form your SELECT statements correctly.

Remember that the FROM, WHERE, GROUP BY and HAVING clauses are evaluated by the query engine before the contents of the SELECT clause. Therefore, elements you write in the SELECT clause, particularly calculated columns and aliases, will not be visible to the other clauses.

For more information on the SELECT elements, see:

SELECT (Transact-SQL)

http://aka.ms/vvwmme

Retrieving Columns from a Table or View

- Use SELECT with column list to show columns
- Use FROM to specify the source table or view
  - Specify both schema and object names
- Delimit names if necessary
- End all statements with a semicolon

Keyword	Expression
SELECT	<select list>
FROM	<table or view>

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

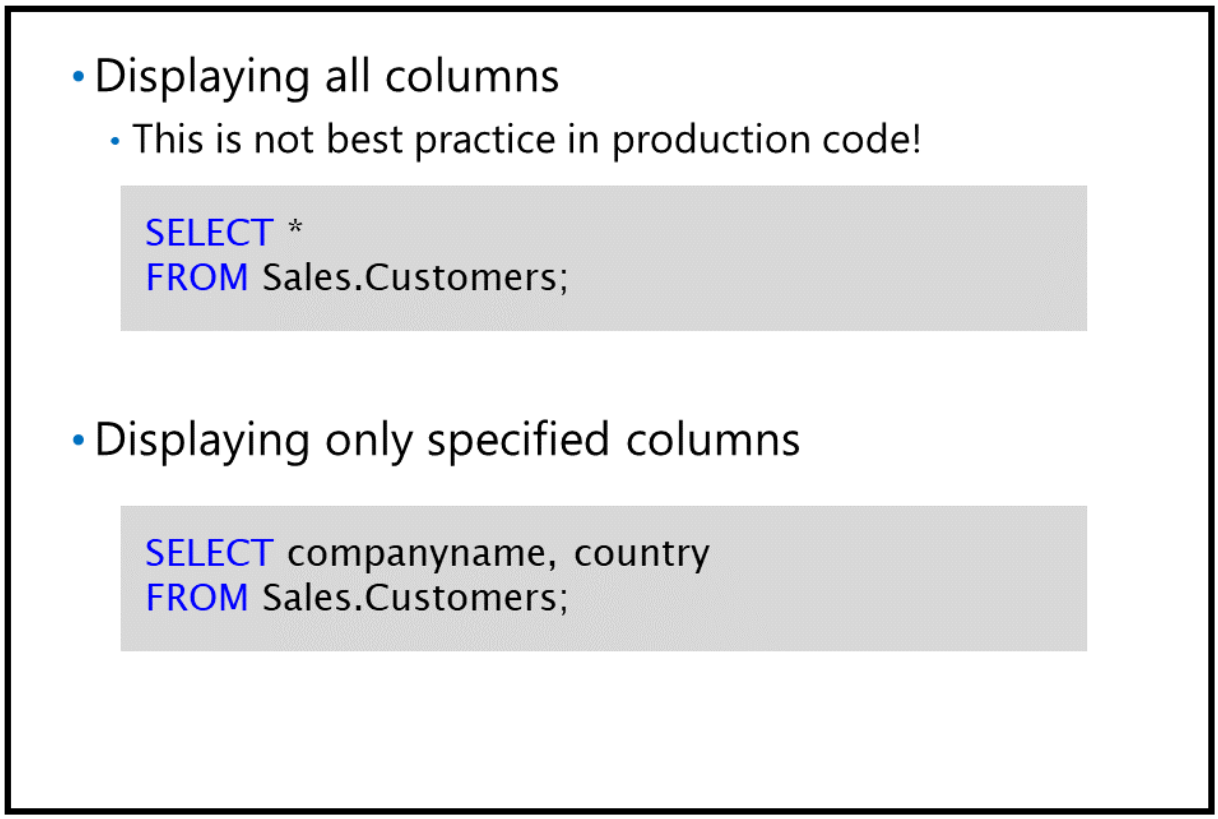
The SELECT clause specifies the columns from the source table(s) or view(s) that you want to return as the result set of the query. In addition to columns from the source table, you can add others in the form of calculated expressions.

The FROM clause specifies the name of the table or view that is the source of the columns in the SELECT clause. To avoid errors in table or view name resolution, it is best to include the schema and object name, in the format SCHEMA.OBJECT—for example Sales.Customer.

If the table or view name contains irregular characters, such as spaces or other special characters, you need to delimit, or enclose, the name. T-SQL supports the use of the ANSI standard double quotes “Sales Order Details”, and the SQL Server specific square brackets [Sales Order Details].

End all statements with a semicolon (;) character. In SQL Server, semicolons are an optional terminator for most statements. However, future versions will require its use. For current usages when a semicolon is required, such as some common table expressions (CTEs) and some Service Broker statements, the error messages returned for a missing semicolon are often cryptic. Therefore, you should adopt the practice of terminating all statements with a semicolon.

Displaying Columns



To display columns in a query, you need to create a comma-delimited column list. The order of the columns in your list will determine their display in the output, regardless of the order in which you have defined them in the source table. This

gives your queries the ability to absorb changes that others may make to the structure of the table, such as adding or reordering the columns.

T-SQL supports the use of the asterisk, or “star” character (*) to substitute for an explicit column list. This will retrieve all columns from the source table. While the asterisk is suitable for a quick test, avoid using it in production work, as changes made to the table will cause the query to retrieve all current columns in the table’s current defined order. This could cause bugs or other failures in reports or applications expecting a known number of columns returned in a defined order. Furthermore, returning data that is not needed can slow down your queries and cause performance issues if the source table contains a large number of rows.

By using an explicit column list in your SELECT clause, you will always achieve the desired results, providing the columns exist in the table. If a column is dropped, you will receive an error that will help identify the problem and fix your query.

Using Calculations in the SELECT Clause

- Calculations are scalar, returning one value per row

Operator	Description
+	Add or concatenate
-	Subtract
*	Multiply
/	Divide
%	Modulo

- Using scalar expressions in the SELECT clause

```
SELECT unitprice, qty, (qty * unitprice)
FROM Sales.OrderDetails;
```

In addition to retrieving columns stored in the source table, a SELECT statement can perform calculations and manipulations. Calculations and manipulations can change the source column data, and use built-in T-SQL functions, which you will learn about later in this course.

As the results will appear in a new column, repeated once per row of the result set, calculated expressions in a SELECT clause must be scalar—they must return only a single value.

Calculated Expression

```
SELECT unitprice, qty, (unitprice * qty)
FROM Sales.OrderDetails;
```

The results appear as follows:

```
unitprice       qty
-----------     ----------      ---------
14.00           12              168.00
9.80            10              98.00
34.80           5               174.00
18.60           9               167.40
```

Note that the new calculated column does not have a name returned with the results. To provide a name, you use a column alias, which you will learn about later in this module.

Create a Calculated Column

```
SELECT empid, lastname, hiredate, YEAR(hiredate)
FROM HR.Employees;
```

The results:

```
empid   lastname        hiredate
------  ---------       ----------------------   ---------
```

```
1                Davis           2002-05-01 00:00:00.000    2002
2                Funk            2002-08-14 00:00:00.000    2002
3                Lew                     2002-04-01 00:00:00.000
2002
```

You will learn more about date and other functions later in this course. The use of YEAR in this example is provided only to illustrate calculated columns.

> **Note:** Not all calculations will be recalculated for each row. SQL Server may calculate a function's result just once at the time of query execution, and reuse the value for each row. This will be discussed later in the course.

Demonstration: Writing Simple SELECT Statements

In this demonstration, you will see how to use simple SELECT queries.

Demonstration Steps

Use Simple SELECT Queries

1. 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**.
2. Run **D:\Demofiles\Mod03\Setup.cmd** as an administrator.
3. In the **User Account Control** dialog box, click **Yes**.
4. At the command prompt, type **y**, and press Enter. When the script has completed, press any key.
5. Start SQL Server Management Studio and connect to the **Azure SQL** database engine instance using SQL Server authentication.
6. Open the **Demo.ssmssln** solution in the **D:\Demofiles\Mod03\Demo** folder.
7. In Solution Explorer, expand **Queries**, and open the **Demonstration A.sql** script file. You may need to enter your password to connect to the **Azure SQL** database engine.

8. In the **Available Databases** list, click **AdventureWorksLT**.
9. Select the code under the comment **Step 2**, and then click **Execute**.
10. Select the code under the comment **Step 3**, and then click **Execute**.
11. Select the code under the comment **Step 4**, and then click **Execute**.
12. Select the code under the comment **Step 5**, and then click **Execute**.
13. Select the code under the comment **Step 6**, and then click **Execute**.
14. Select the code under the comment **Step 7**, and then click **Execute**.
15. On the **File** menu, click **Close**.
16. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Discovery

You have a table named Sales with the following columns: Country, NumberOfReps, TotalSales.You want to find out the average amount of sales a sales representative makes in each country. What SELECT query could you use?

Show solution Reset

Lesson 2: Eliminating Duplicates with DISTINCT

T-SQL queries may display duplicate rows, even if the source table has a key column enforcing uniqueness. Typically, this is the case when you retrieve only a few of the columns in a table. In this lesson, you will learn how to eliminate duplicates using the DISTINCT clause.

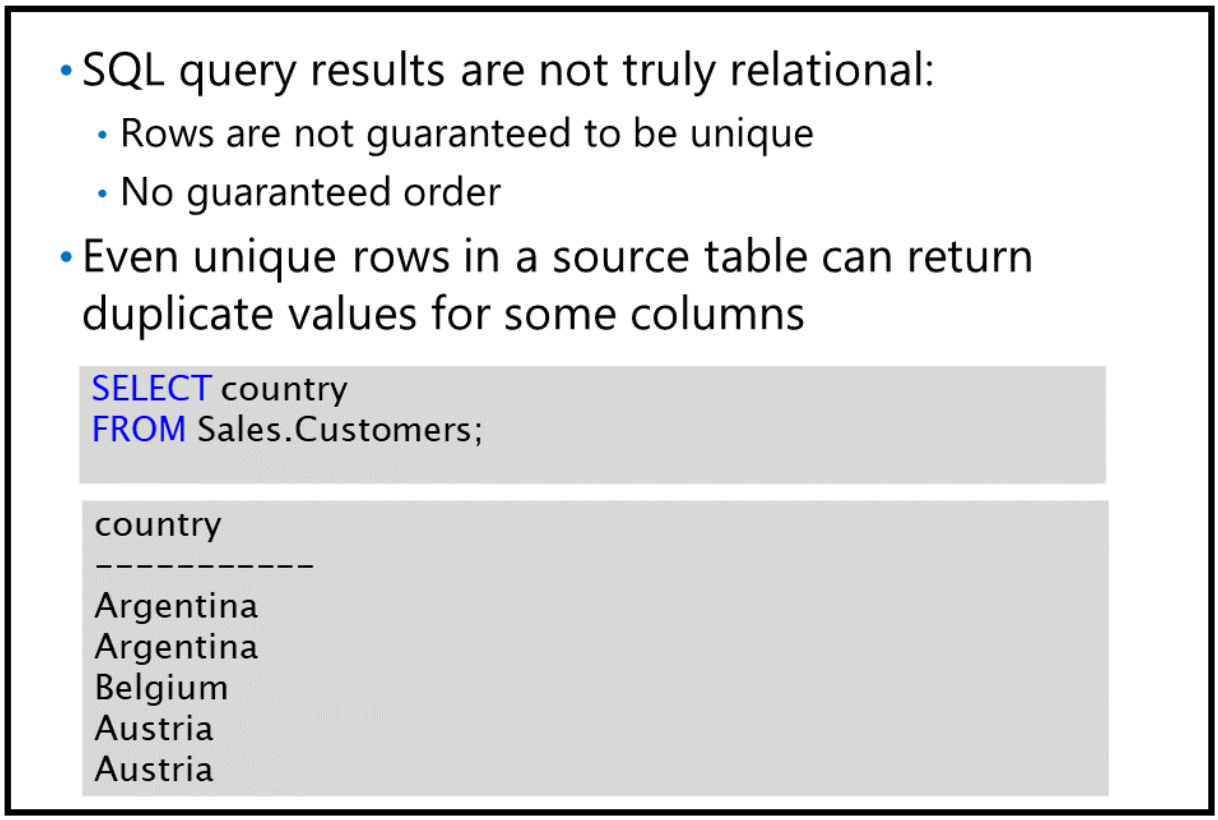
Lesson Objectives

In this lesson, you will learn how to:

- Understand how T-SQL query results are not true sets and may include duplicates.
- Understand how DISTINCT may be used to remove duplicate rows from the SELECT results.
- Write SELECT DISTINCT clauses.

This document belongs to Sebastian Amiaesie@or... Not authorized

SQL Sets and Duplicate Rows



While the theory of relational databases calls for unique rows in a table, in practice, T-SQL query results are not true sets. The rows retrieved by a query are not guaranteed to be unique, even when they come from a source table that uses a primary key to differentiate each row. The rows are not returned in any particular order. You will learn how to address this with ORDER BY later in this course.

Add to this the fact that the default behavior of a SELECT statement is to include the keyword ALL, and you can begin to see why duplicate values might be returned by a query—especially when you include only some of the columns in a table (and omit the unique columns).

This document belongs to Sebastian Amiaesie@gmail.com Not authorized copies allowed!

SELECT Query

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

This document belongs to Sebastian Amihaesie. sebastian.amihaesie@gmail.com No unauthorized copies allowed

A partial result shows many duplicate country names, which at best is too long to easily interpret. At worst, it gives a wrong answer to the question: “How many countries are represented among our customers?”

```
country
--------------------
Germany
Mexico
Mexico
UK
Sweden
Germany
Germany
France
UK
Austria
Brazil
Spain
France
Sweden

…
Germany
France
Finland
Poland

(91 rows(s) affected)
```

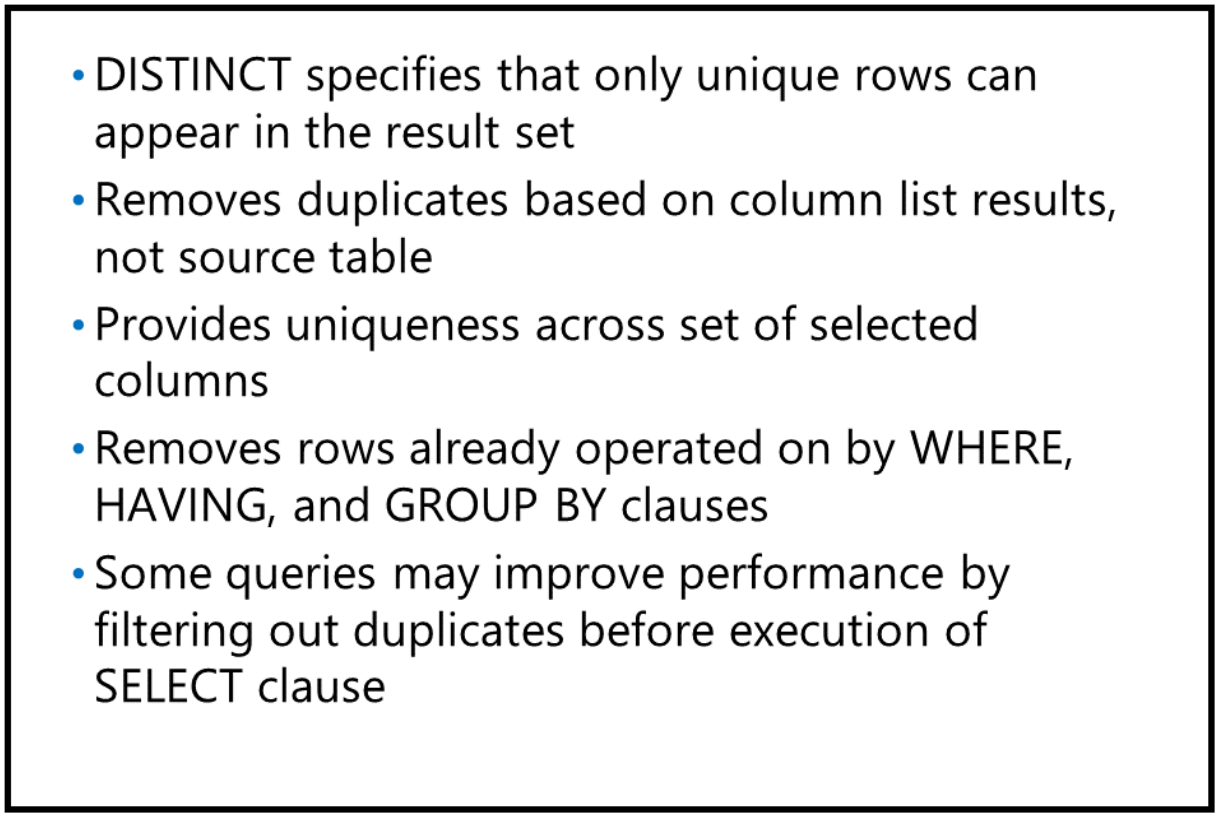
This document

All Statement

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

Without further instruction, the query will return one result for each row in the Sales.Customers table; however, as only the country column is specified, you will see just this column for all 91 rows.

Understanding DISTINCT



Replacing the default SELECT ALL clause with SELECT DISTINCT will filter out duplicates in the result set. SELECT DISTINCT specifies that the result set must contain only unique rows. However, it is important to understand that the DISTINCT option operates only on the set of columns returned by the SELECT clause. It does not take into account any other unique columns in the source table. DISTINCT also operates on all the columns in the SELECT list, not just the first one.

The logical order of operations also ensures that the DISTINCT operator will remove rows that may have already been processed by WHERE, HAVING, and GROUP BY

clauses.

DISTINCT Statement

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

No unauthorized access allowed.

This will return the desired results. Note that, while the results appear to be sorted, this is not guaranteed by SQL Server. The result set now contains only one instance of each unique output row:

```
country
--------
Argentina
Austria
Belgium
Brazil
Canada
Denmark
Finland
France
Germany
Ireland
Italy
Mexico
Norway
Poland
Portugal
Spain
Sweden
Switzerland
UK
USA
Venezuela
```

```
(21 row(s) affected)
```

> **Note:** You will learn additional methods for filtering out duplicate values later in this course. After you have learned them, you could consider the relative performance costs of filtering with SELECT DISTINCT, compared to those other methods.

Sebastian.amihaesie@gmail.com belongs to Sebastian Amihaesie. No unauthorized copies allowed!

SELECT DISTINCT Syntax

```
SELECT DISTINCT <column list>

FROM <table or view>
```

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

```
companyname          country
--------------       -------
Customer AHPOP       UK
Customer AHXHT       Mexico
Customer AZJED       Germany
Customer BSVAR       France
Customer CCFIZ       Poland
```

Remember that DISTINCT looks at rows in the output set, created by the SELECT clause. Therefore, only unique column values will be returned by a SELECT DISTINCT clause.

This document belongs to Sebastian Amihaesie. Sebastian.amihaesie@gmail.com No unauthorized copies allowed!

SELECT Statement

```
SELECT firstname, lastname
FROM Sales.Customers;
```

This document

The results:

```
firstname               lastname
------------    -----------------
Sara            Davis
Don                     Funk
Sara            Lew
Don                     Davis
Judy            Lew
Judy            Funk
Yael            Peled
```

However, a SELECT DISTINCT query against both columns will retrieve all unique combinations of the two columns which, in this case, is the same seven employees.

This document belongs to Sebastian Amihaesie
sebastian.amihaesie@gmail.com
No unauthorized copies

DISTINCT Syntax

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

The results:

This document

```
firstname
--------------------
Don
Judy
Sara
Yael

(4 row(s) affected)
```

Document

A challenge in designing such queries is that, while you may need to retrieve a distinct list of values from one column, you might want to see additional attributes (columns) from others. Later in this course, you will see how to combine DISTINCT with the GROUP BY clause as a way of further processing and displaying information about distinct lists of values.

This document belongs to Sebastian Amihaesie. sebastian.amihaesie@gmail.com No unauthorized copies allowed!

Demonstration: Eliminating Duplicates with DISTINCT

In this demonstration, you will see how to eliminate duplicate rows.

Demonstration Steps

Eliminate Duplicate Rows

1. In Solution Explorer, open the **Demonstration B.sql** script file. You may need to enter your password to connect to the **Azure SQL** database engine.
2. In the **Available Databases** list, click **AdventureWorksLT**.
3. Select the code under the comment **Step 2**, and then click **Execute**.
4. Select the code under the comment **Step 3**, and then click **Execute**.
5. Select the code under the comment **Step 4**, and then click **Execute**.
6. On the **File** menu, click **Close**.
7. Keep SQL Server Management Studio open for the next demonstration.

This document belongs to Sebastian Amihaesie. sebastian.amihaesie@gmail.com No unauthorized copies allowed!

Check Your Knowledge

Discovery

You have company departments in five countries. You have the following query for the Human Resources database:

```
SELECT DeptName, Country
FROM HumanResources.Departments
```

This returns:

```
DeptName  Country
--------- --------
```

Sales UK
Sales USA
Sales France
Sales Japan
Marketing USA
Marketing Japan
Research USA
How many rows are returned?

Show solution Reset

Lesson 3: Using Column and Table Aliases

When retrieving data from a table or view, a T-SQL query will name each column after its source. You can relabel columns by using aliases in the SELECT clause. However, columns created with expressions will not be named automatically. Column aliases can be used to provide custom column headers. At the table level, you can use aliases in the FROM clause to provide a convenient way of referring to a table elsewhere in the query, enhancing readability.

Lesson Objectives

In this lesson, you will learn how to:

- Use aliases to refer to columns in a SELECT list.
- Use aliases to refer to columns in a FROM clause.
- Understand the impact of the logical order of query processing on aliases.

Use Aliases to Refer to Columns

```
SELECT DISTINCT <column list>

FROM <table or view>
```

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

```
companyname          country
---------------      -------
Customer AHPOP       UK
Customer AHXHT       Mexico
Customer AZJED       Germany
Customer BSVAR       France
Customer CCFIZ       Poland
```

Column aliases can be used to relabel columns when returning the results of a query. For example, cryptic names of columns in a table such as "qty" can be replaced with "quantity".

Expressions that are not based on a source column in the table will not have a name provided in the result set. This includes calculated expressions and function calls. While T-SQL doesn't require that a column in a result set have a name, it's a good idea to provide one.

In T-SQL, there are multiple methods of creating a column alias, with identical output results.

AS Keyword

```
SELECT orderid, unitprice, qty AS quantity
FROM Sales.OrderDetails;
```

Alias with Equals Sign

```
SELECT orderid, unitprice, quantity = qty
FROM Sales.OrderDetails;
```

Alias Following Column Name

This document belongs to Sebastian Amihaesie.
sebastian.amihaesie@gmail.com
No unauthorized copies allowed!

```
SELECT orderid, unitprice, qty quantity
FROM Sales.OrderDetails;
```

While there is no difference in performance or execution, a variance in readability may cause you to choose one or the other as a convention.

Warning: Column aliases can also be accidentally created, by omitting a comma between two column names in the SELECT list.

Accidental Alias

This document belongs to Sebastian Amihaesie.
sebastian.amihaesie@gmail.com
No unauthorized copies allowed!

```
SELECT orderid, unitprice quantity
FROM Sales.OrderDetails;
```

The results:

This document belongs to Sebastian Amihaesie.
sebastian.amihaesie@gmail.com

```
orderid         quantity
---------       ---------------------
10248           14.00
10248           9.80
10248           34.80
10248           18.60
```

This document belongs to Sebastian Amihaesie.

As you can see, this could be difficult to identify and fix in a client application. The only way to avoid this problem is to list columns carefully, separating them with commas and adopting the AS style of aliases, to make it easier to spot mistakes.

Check Your Knowledge

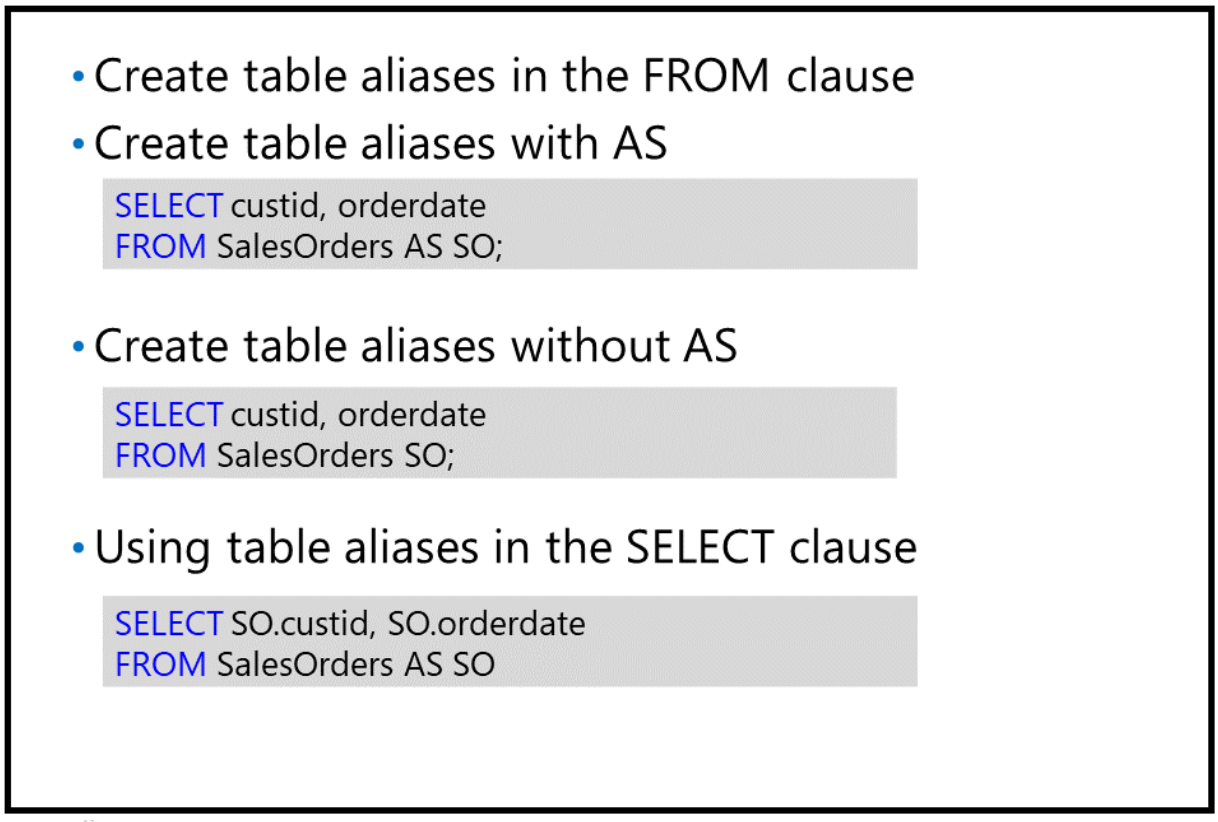
Discovery

Which of the following statements use correct column aliases?

1. SELECT Name AS ProductName FROM Production.Product
2. SELECT Name = ProductName FROM Production.Product
3. SELECT ProductName == Name FROM Production.Product
4. SELECT ProductName = Name FROM Production.Product
5. SELECT Name AS Product Name FROM Production.Product

Show solution Reset

Use Aliases to Refer to Tables



Aliases can also be used in the FROM clause to refer to a table; this can improve readability and save redundancy when referencing the table elsewhere in the query. While this module has focused on single-table queries, which don't necessarily benefit from table aliases, this technique will prove useful as you learn more complex queries in subsequent modules.

To create a table alias in a FROM clause, you will use syntax similar to several of the column alias techniques.

Table Alias using AS

```
SELECT orderid, unitprice, qty
FROM Sales.OrderDetails AS OD;
```

Table Alias Without AS

```
SELECT orderid, unitprice, qty
FROM Sales.OrderDetails OD;
```

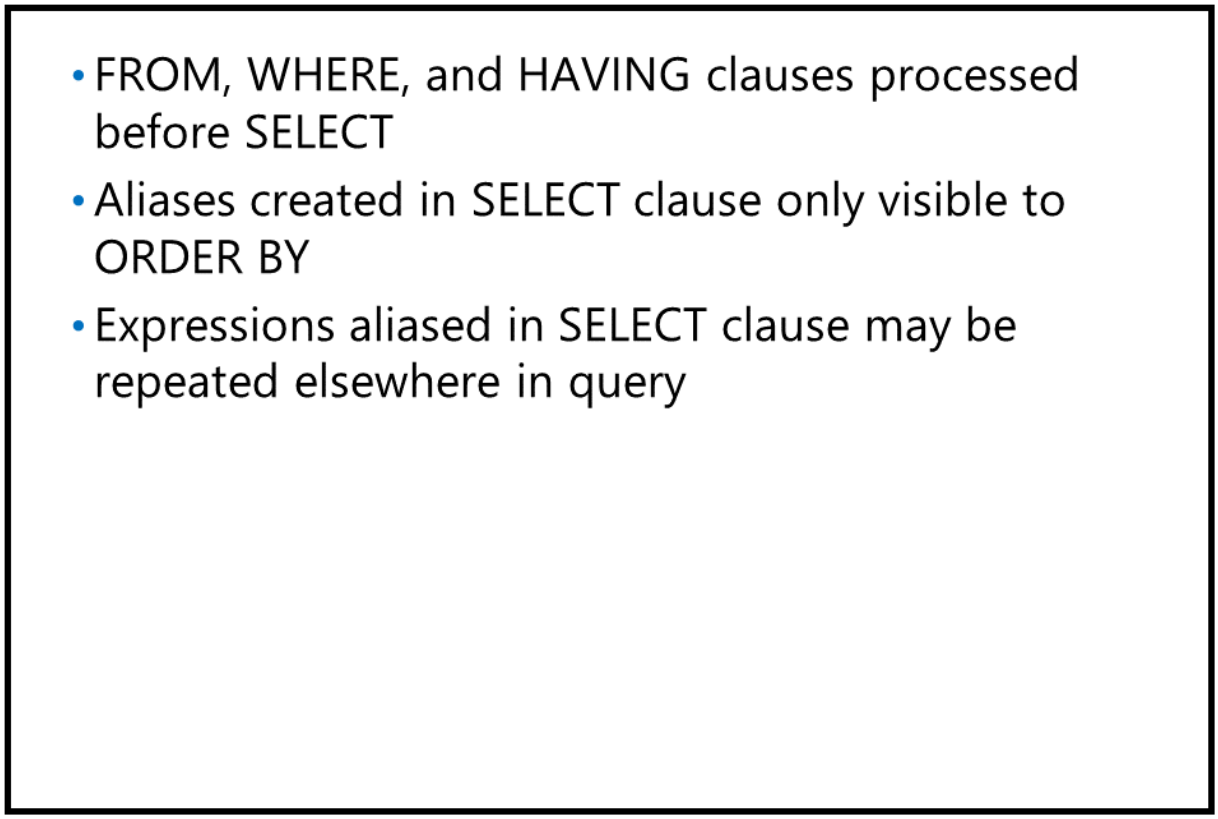
Table and Column Aliases Combined

```
SELECT OD.orderid, OD.unitprice, OD.qty AS Quantity
FROM Sales.OrderDetails AS OD;
```

> **Note:** There is no table alias equivalent to the use of the equals sign (=) in a column alias.

As this module focuses on single-table queries, you might not yet see a benefit to using table aliases. In the next module, you will learn how to retrieve data from multiple tables in a single SELECT statement. In those queries, the use of table aliases to represent table names will be useful.

The Impact of Logical Processing Order on Aliases



When using column aliases, an issue can arise. Aliases created in the SELECT clause may not be referred to in others in the query—such as a WHERE or HAVING clause. This is due to the logical order query processing. The WHERE and HAVING clauses are processed before the SELECT clause and its aliases are evaluated (HAVING and WHERE clauses will be covered in a separate module). An exception to this is the ORDER BY clause.

ORDER BY with Alias

```
SELECT orderid, unitprice, qty AS quantity
FROM Sales.OrderDetails
ORDER BY quantity;
```

Incorrect WHERE with Alias

```
SELECT orderid, unitprice, qty AS quantity
FROM Sales.OrderDetails
WHERE quantity > 10;
```

```
WHERE quantity > 20;
```

The resulting error message is:

```
Msg 207, Level 16, State 1, Line 1
Invalid column name 'quantity'.
```

Correct WHERE with Alias

```
SELECT orderid, YEAR(orderdate) AS orderyear
FROM Sales.Orders
WHERE YEAR(orderdate) = '2008'
```

Additionally, within the SELECT clause, you might not refer to a column alias that was defined in the same SELECT statement, regardless of column order.

Column Alias Used in SELECT Clause

```
SELECT productid, unitprice AS price, price * qty AS total
FROM Sales.OrderDetails;
```

The resulting error:

```
Msg 207, Level 16, State 1, Line 1
Invalid column name 'price'.
```

Demonstration: Using Column and Table Aliases

In this demonstration, you will see how to use column and table aliases.

Demonstration Steps

Use Column and Table Aliases

1. In Solution Explorer, open the **Demonstration C.sql** script file. You may need to enter your password to connect to the **Azure SQL** database engine.
2. In the **Available Databases** list, click **AdventureWorksLT**.
3. Select the code under the comment **Step 2**, and then click **Execute**.
4. Select the code under the comment **Step 3**, and then click **Execute**.
5. Select the code under the comment **Step 4**, and then click **Execute**.
6. Select the code under the comment **Step 5**, and then click **Execute**.
7. On the **File** menu, click **Close**.
8. Keep SQL Server Management Studio open for the next demonstration.

This document belongs to Sebastian Amihaesie. sebastian.amihaesie@gmail.com No unauthorized copies allowed!

Check Your Knowledge

Discovery

You have the following query:
SELECT FirstName LastName
FROM HumanResources.Employees;
You are surprised to find that the query returns the following:
LastName

Fred
Rosalind
Anil
Linda
What error have you made in the SELECT query?

Show solution Reset

Lesson 4: Writing Simple CASE Expressions

A CASE expression extends the ability of a SELECT clause to manipulate data as it is retrieved. Often when writing a query, you need to substitute a value of a column with another value. While you will learn how to perform this kind of lookup from another table later in this course, you can also perform basic substitutions using simple CASE expressions in the SELECT clause. In real-world environments, CASE is often used to help make cryptic data that is held in a column more meaningful.

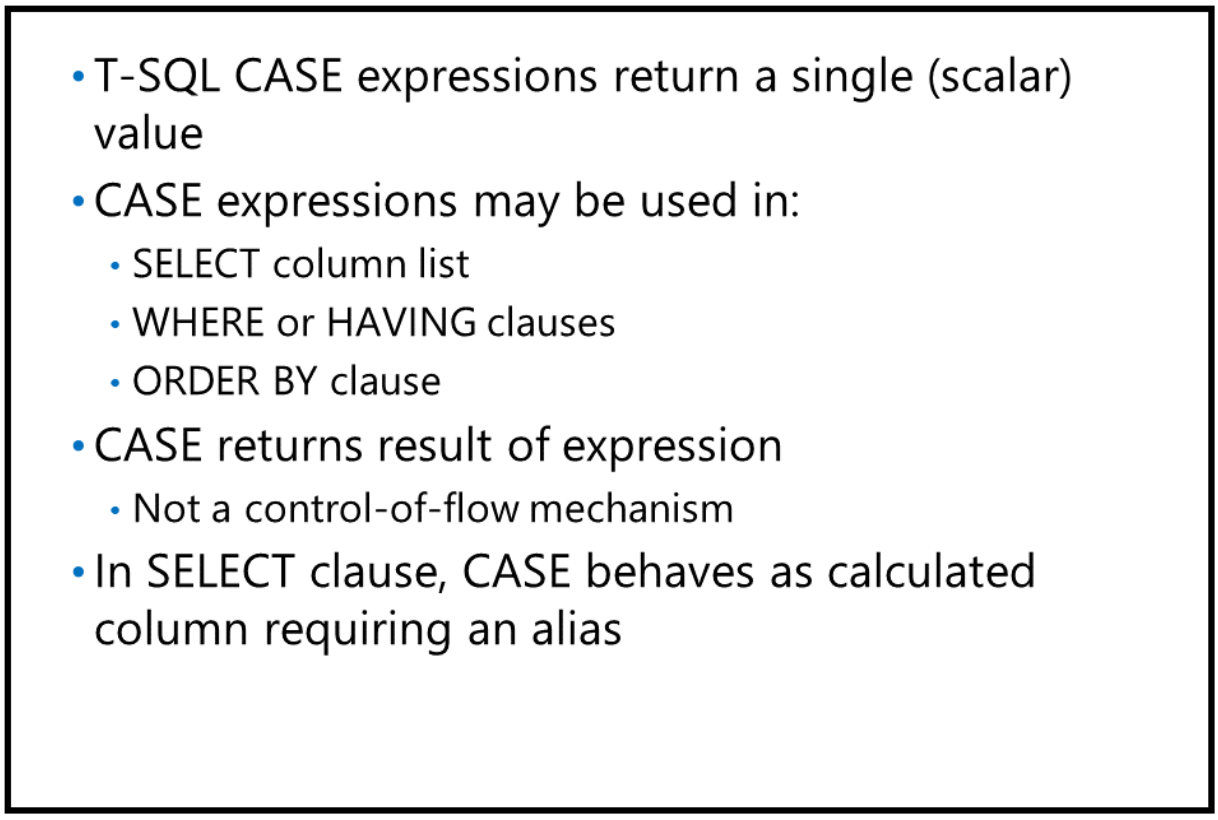
A CASE expression returns a scalar (single-valued) value based on conditional logic, often with multiple conditions. As a scalar value, it may be used wherever single values can be used. Besides the SELECT statement, CASE expressions can be used in WHERE, HAVING, and ORDER BY clauses.

Lesson Objectives

In this lesson, you will learn how to:

- Understand the use of CASE expressions in SELECT clauses.
- Understand the simple form of a CASE expression.

Using CASE Expressions in SELECT Clauses

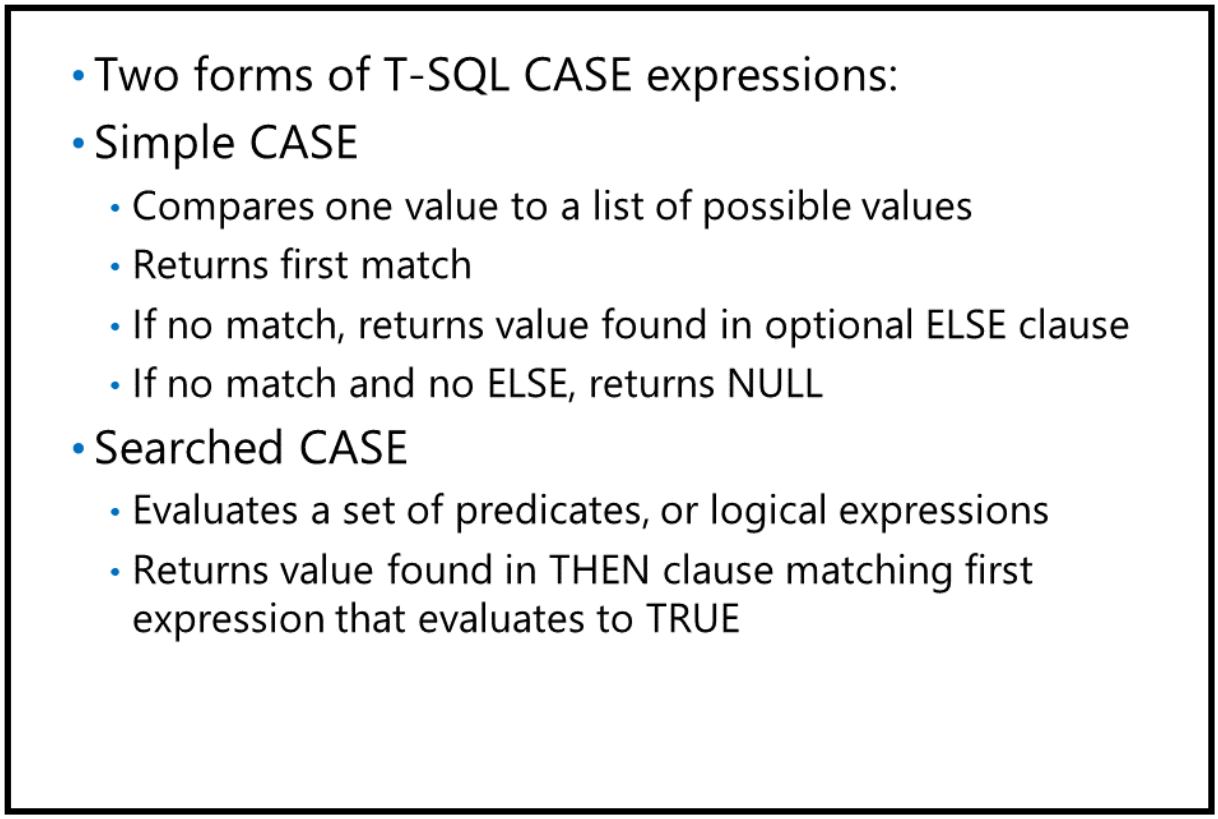


In T-SQL, CASE expressions return a single, or scalar, value. Unlike some other programming languages, in T-SQL, CASE expressions are not statements, nor do they specify the control of programmatic flow. Instead, they are used in SELECT (and other) clauses to return the result of an expression. The results appear as a calculated column and, for clarity, should be aliased.

This document belongs to Sebastian Amihaesie@gmail.com
No unauthorized copies allowed!

In T-SQL queries, CASE expressions are often used to provide an alternative value for one stored in the source table. For example, a CASE expression might be used to provide a friendly text name for something stored as a compact numeric code.

Forms of CASE Expressions



In T-SQL, CASE expressions may take one of two forms—simple CASE, or searched (Boolean) CASE.

Simple CASE expressions, the subject of this lesson, compare an input value to a list of possible matching values:

- If a match is found, the first matching value is returned as the result of the CASE expression. Multiple matches are not permitted.
- If no match is found, a CASE expression returns the value found in an ELSE

clause, if one exists.

- If no match is found and no ELSE clause is present, the CASE expression returns a NULL.

CASE Expression

```
SELECT productid, productname, categoryid,
        CASE categoryid
          WHEN 1 THEN 'Beverages'
          WHEN 2 THEN 'Condiments'
          WHEN 3 THEN 'Confections'
          ELSE 'Unknown Category'
       END AS categoryname
FROM Production.Categories
```

The results:

```
productid productname  categoryid  categoryname
--------- ------------ ---------- --------------------
101       Tea          1          Beverages
102       Mustard      2          Condiments
103       Dinner Rolls 9          Unknown Category
```

> **Note:** The preceding example is presented for illustration only and will not run against the sample databases provided with the course.

Searched (Boolean) CASE expressions compare an input value to a set of logical predicates or expressions. The expression can contain a range of values to match against. Like a simple CASE expression, the return value is found in the THEN clause of the matching value.

Due to their dependence on predicate expressions, which will not be covered until later in this course, further discussion of searched CASE expressions is beyond the scope of this lesson.

See CASE (Transact-SQL) in Microsoft Docs:

CASE (Transact-SQL)

http://aka.ms/ims4v6

This document belongs to Sebastian Amihaesie
sebastian.amihaesie@gmail.com
No unauthorized copies allowed!

Demonstration: Simple CASE Expressions

In this demonstration, you will see how to use a simple CASE expression.

Demonstration Steps

Use a Simple CASE Expression

1. In Solution Explorer, open the **Demonstration D.sql** script file. You may need to enter your password to connect to the **Azure SQL** database engine.
2. In the **Available Databases** list, click **AdventureWorksLT**.
3. Select the code under the comment **Step 2**, and then click **Execute**.
4. Select the code under the comment **Step 3**, and then click **Execute**.
5. Close SQL Server Management Studio, without saving changes.

This document belongs to Sebastian Amihaesie
sebastian.amihaesie@gmail.com
No unauthorized copies allowed!

Check Your Knowledge

Discovery

You have the following SELECT query:
SELECT FirstName, LastName, Sex
FROM HumanResources.Employees;
This returns:
FirstName LastName Sex
---------- --------- ----

Maya Steele 1
Adam Brookes 0
Naomi Sharp 1
Pedro Fielder 0
Zachary Parsons 0
How could you make these results clearer?

Show solution Reset

This document belongs to Sebastian Amihaesie. sebastian.amihaesie@gmail.com No unauthorized copies allowed!

Lab: Writing Basic SELECT Statements

Scenario

As a business analyst for Adventure Works, you will be writing reports using corporate databases stored in SQL Server. You can use your set of business requirements for data to write basic T-SQL queries to retrieve the specified data from the databases.

Objectives

After completing this lab, you will be able to:

- Write simple SELECT statements.
- Eliminate duplicate rows by using the DISTINCT keyword.
- Use table and column aliases.
- Use a simple CASE expression.

Lab Setup

Estimated Time: 40 minutes

Virtual machine: **20761C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Writing Simple SELECT Statements

Scenario

As a business analyst, you want a better understanding of your corporate data. Usually, the best approach for an initial project is to get an overview of the main tables and columns, so you can better understand different business requirements. After an initial overview, you will provide a report for the marketing department, whose staff want to send invitation letters for a new campaign. You will use the TSQL sample database.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. View All the Tables in the ADVENTUREWORKS Database in Object Explorer
3. Write a Simple SELECT Statement That Returns All Rows and Columns from a Table
4. Write a SELECT Statement That Returns Specific Columns

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

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

Exercise 2: Eliminating Duplicates Using DISTINCT

Scenario

After supplying the marketing department with a list of all customers for a new campaign, you are asked to provide a list of all the countries that the customers come from.

The main tasks for this exercise are as follows:

1. Write a SELECT Statement That Includes a Specific Column
2. Write a SELECT Statement That Uses the DISTINCT Clause

This document belongs to Sebastian Amihaesie. sebastian.amihaesie@gmail.com No unauthorized copies allowed!

Detailed Steps ▼

Detailed Steps ▼

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

This document belongs to Sebastian Amihaesie. sebastian.amihaesie@gmail.com No unauthorized copies allowed!

Exercise 3: Using Table and Column Aliases

Scenario

After receiving the initial list of customers, the marketing department would like to have column titles that are more readable and a list of all products in the TSQL database.

The main tasks for this exercise are as follows:

This document belongs to Sebastian Amihaesie. sebastian.amihaesie@gmail.com No unauthorized copies allowed!

1. Write a SELECT Statement That Uses a Table Alias
2. Write a SELECT Statement That Uses Column Aliases
3. Write a SELECT Statement That Uses Table and Column Aliases
4. Analyze and Correct the Query

This document belongs to Sebastian Amihaesie. sebastian.amihaesie@gmail.com No unauthorized copies allowed!

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

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

This document belongs to Sebastian Amihaesie. sebastian.amihaesie@gmail.com No unauthorized copies allowed!

Exercise 4: Using a Simple CASE Expression

Scenario

Your company has a long list of products and the members of the marketing department would like to have product category information in their reports. They have supplied you with a document containing the following mapping between the product category IDs and their names:

categoryid	categoryname
1	Beverages
2	Condiments
3	Confections
4	Dairy Products
5	Grains/Cereals
6	Meat/Poultry
7	Produce
8	Seafood

They have an active marketing campaign, and would like to include product category information in their reports.

The main tasks for this exercise are as follows:

1. Write a SELECT Statement
2. Write a SELECT Statement That Uses a CASE Expression
3. Write a SELECT Statement That Uses a CASE Expression to Differentiate Campaign-Focused Products

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

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

Module Review and Takeaways

In this module, you have learned how to:

- Write simple SELECT statements.
- Eliminate duplicates using the DISTINCT clause.
- Use table and column aliases.
- Write simple CASE expressions.

Best Practice: Terminate all T-SQL statements with a semicolon. This will make your code more readable, avoid certain parsing errors, and protect your code against changes in future versions of SQL Server.
Consider standardizing your code on the AS keyword for labeling column and table aliases. This will make it easier to read and avoids accidental aliases.

Review Question(s)

Check Your Knowledge

Discovery

Why is the use of SELECT * not a recommended practice?

Show solution Reset

Real-world Issues and Scenarios

You can create a column alias without using the AS keyword, something you are likely to see in code samples online, or written by developers you work with. While the T-SQL engine will parse this without issue, there is a problem when a comma is omitted between column names—the first column will take the name of the second column as its alias. Not only will the column have a misleading name, but you will also have one column too few in your result set. Always use the AS keyword to avoid this problem.

This document belongs to Sebastian Amihaesie.
sebastian.amihaesie@gmail.com
No unauthorized copies allowed!