

NET3000

Database Concepts and SQL

Section 3: Using the SELECT Statement

The SELECT Statement

- The SELECT statement is the most-used SQL command.
- This statement retrieves data from your tables and is composed of just six keywords that *must* be in this order:
 - a. SELECT
 - b. FROM
 - c. WHERE
 - d. GROUP BY
 - e. HAVING
 - f. ORDER BY
- All but the keyword SELECT are optional.

Understanding the Query Statements

- The SELECT clause list specifies the columns to be returned.
- The FROM clause specifies the table from which columns and rows are returned.
- The WHERE clause specifies the rows to return. The search condition is the WHERE clause restricts the rows that are retrieved by using conditional and logical operators. The conditions specified in the WHERE clause are known as the SARG --- the Search ARGument.

Group By and Having

- The GROUP BY clause specifies a selected set of rows into a set of summary rows by the values of one or more columns or expressions. One row is returned for each group. Aggregate functions in the SELECT clause <select> list provide information about each group instead of having individual rows.
- The HAVING clause specifies a search condition for a group or an aggregate. HAVING can be used only with the SELECT statement. HAVING is typically used in a GROUP BY clause. When GROUP BY is not used, HAVING behaves like a WHERE clause.

Sorting

- The ORDER BY clause specifies the sort order used on columns returned in a SELECT statement.
- The ORDER BY clause is not valid in views, inline functions, derived tables, and subqueries, unless TOP is also specified.

Partial Syntax

```
SELECT [ALL | DISTINCT] <select_list>  
FROM [[database_name.][owner_name].]{ table_name }  
[AS] alias  
WHERE <search_condition>
```

Using the SELECT Clause

- You can retrieve particular columns from a table by listing them in the select list.
- The select list contains the columns, expressions, or keywords to select.
- The select list retrieves and displays the columns in the specified order.
- Separate the column names with commas.
 - Do not place a comma after the last column name.
- Use an asterisk (*) in the select list to retrieve all columns from a table.

Syntax

```
<select_list> :: =  
{ *  
| { table_name | table_alias }. *  
| [{ table_name | table_alias }.]  
{ column_name | expression | IDENTITYCOL |  
ROWGUIDCOL }  
[ [AS] column_alias ]  
| column_alias = expression  
} [, . . . n]
```


Example

```
USE AdventureWorks ;  
GO  
SELECT *  
FROM Production.Product ;  
GO
```

Using the FROM Clause

- The FROM clause specifies the tables, views, derived tables, and joined tables used in DELETE, SELECT, and UPDATE statements.
- In the SELECT statement, the FROM clause is required except when the select list contains only constants, variables, and arithmetic expression (no column names).
- Example:
USE AdventureWorks;
GO
SELECT Title, FirstName, MiddleInitial, LastName
FROM HumanResources.Employee ;
GO

Using the WHERE Clause to Specify Rows

- The WHERE clause of the SELECT statement restricts the number of rows that are returned.
- Using the WHERE clause, you can also retrieve specific rows based on a given search condition. Only rows that match the search condition in the WHERE clause are returned by the SELECT statement. The search condition in the WHERE clause can contain an unlimited list of predicates (expressions that return a value of TRUE, FALSE, or UNKNOWN).

Rules and Advice

- Place single quotation marks around all text data.
- Use positive rather than negative predicates because negative predicates slow the speed of data retrieval by forcing table scans rather than the use of indexes.
- Whenever possible, use a WHERE clause to limit the number of rows that are returned, rather than returning all rows from a table. Returning fewer rows of data improves performance, especially when you use the SELECT * statement.

Example

```
USE AdventureWorks;  
GO  
SELECT Title, FirstName, LastName  
FROM Person.Contact  
WHERE LastName = 'Abercrombie';  
GO
```

Using String Comparisons

- You can use the LIKE operator in combination with wildcard characters to select rows by comparing character values from the rows to a pattern string specified with the LIKE operator.
- If you want to make an exact string comparison, use a comparison operator rather than the LIKE operator; for example, use `Country = 'USA'` rather than `Country LIKE 'USA'`.
- All characters in the pattern string are significant, including leading and trailing blank spaces.
- LIKE can be used only with text data of the char, nchar, varchar, nvarchar, or datetime data types.

Example

```
SELECT name  
FROM sys.objects  
WHERE name LIKE 'sys%';  
GO
```

Understanding Wildcard Characters

Wildcard	Description
% (percent)	Any string of zero or more characters
_ (underscore)	Any single character
[] (bracket)	Any single character within the specified range (for example, [s-w] or set (for example [aeiou])
[^] (caret)	Any single character not within the specified range (for example, [^s-w] or set (for example [^aeiou])

Using the BETWEEN Operator

- Use the BETWEEN operator in the WHERE clause to retrieve rows that are within a specified range of values.
- SQL Server includes the end values in the result set.
- Use the BETWEEN operator, rather than a predicate that include the AND operator, with two comparison operators ($\geq x$ AND $\leq y$).
- Use the NOT BETWEEN operator to retrieve rows outside of the specified range.
 - Be aware, however, that negative conditions slow data retrieval.
- Be careful with date and time values because midnight is the endpoint for the ending date value. No data with a time after midnight on the ending date will be returned.

Example

```
USE AdventureWorks;  
GO  
SELECT Name FROM Production.Product  
WHERE ProductID BETWEEN 319 and 342 ;  
GO
```

Using the IN Operator

- Use the IN operator in the WHERE clause to retrieve rows that match a specified list of values.
- Use either the IN operator or a series of predicates that are connected with an OR operator.
- Do not include the NULL value in the list. A NULL value in the list evaluates to the comparison, = NULL. This may return unpredictable result sets.
- Use the NOT IN operator to retrieve rows that are not in your list of values. Be aware, however, that a negative condition slows data retrieval because it forces the query optimizer to perform a table scan.

Example

```
SELECT ProductModelID, Name  
INTO dbo.Gloves  
FROM Production.ProductModel  
WHERE ProductModelID IN (3, 4);  
GO
```

Working with NULL Values

- NULL values require special handling. Essentially, NULL values have undefined values that create unexpected results when encountered in filtered columns.
- Use IS NULL, IS NOT NULL, NULLIF, and COALESCE to avoid these problems.

IS NULL Operator

- Use the IS NULL operator to retrieve rows for which information is missing from a specified column.
- Null values fail all comparisons because they do not evaluate equally with one another.
- You define whether columns allow null value in the CREATE TABLE statement.
- Use the IS NOT NULL operator to retrieve rows that have known values in the specified columns.

Example: IS NOT NULL

```
USE tempdb ;  
GO  
IF OBJECT_ID (N'#Bicycles',N'U') IS NOT NULL  
DROP TABLE #Bicycles ;  
GO
```

Example: ISNULL()

-- returns a given value if the column value is NULL

```
SELECT Description, DiscountPct, MinQTY,  
ISNULL(MaxQty, 0) AS 'Max Quantity'  
FROM Sales.SpecialOffer
```


Example: NULLIF()

-- returns NULL if both specified expressions are equal

```
SELECT ProductID, MakeFlag, FinishedGoodsFlag,  
NULLIF(MakeFlag, FinishedGoodsFlag) AS 'NULL if Equal'  
FROM Production.Product WHERE ProductID < 10
```

Example: COALESCE()

-- returns the first non-NULL expression among its arguments

```
SELECT CAST(COALESCE(Hourly_Wages * 40 * 52,  
    Salary, Commission * Num_Sales) AS Money) AS 'Total  
    Salary'  
FROM Wages
```

Using Logical Operators

- Use the logical operators AND and OR to combine a series of predicates and to refine query processing.
- Use the logical NOT operator to negate the value of a predicate.
- The results of a query may vary depending on the grouping and ordering of the predicates.
- Use the AND operator to retrieve rows that meet all of the search criteria.
- Use the OR operator to retrieve rows that meet any of the search criteria.
- Use the NOT operator to negate the expression that follows the operator.

Ordering of Predicates

- SQL Server evaluates the NOT operator first, followed by the AND operator, and then the OR operator.
- The precedence order is from left to right if all operators in an expression are of the same level.
- Group operators to control the precedence with the use of parens ().

Using Parentheses

- Use parentheses when you have two or more expressions as the search criteria.
- Using parentheses allows you to:
 - Group expressions
 - Change the order of evaluation
 - Make expression more readable

Using the ORDER BY Clause

- Use the ORDER BY clause to sort rows in the result set in ascending or descending order.
- When SQL Server is installed, a sort order is specified. The sort order is a set of rules that determine how SQL Server sorts and compares character data.
 - Execute the sp_helpsort system stored procedure or SERVERPROPERTY('Collation') to determine the sort order that is in use on your SQL Server.
- SQL Server does not guarantee an order in the result set unless the order is specified with an ORDER BY clause.
- SQL Server sorts in ascending order by default.
- Columns that are included in the ORDER BY clause do not have to appear in the select list.

ORDER BY

- The total size of the columns specified in the ORDER BY clause cannot exceed 8,060 bytes.
- You can sort by column names, computed values, or expressions.
- In the ORDER BY clause, you can refer to columns by their names, aliases, or positions in the select list.
- To sort the values of a column in descending order, specify the DESC keyword after the column reference in the column list of the ORDER BY clause.
- Do not use an ORDER BY clause on text or image columns.

Example

```
SELECT ProductModelID, Name  
FROM Production.ProductModel  
WHERE ProductModelID NOT IN (3, 4)  
UNION  
SELECT ProductModelID, Name  
FROM dbo.Gloves  
ORDER BY Name ;  
GO
```


Eliminating Duplicates

- If you require a list of unique values, use the DISTINCT clause to eliminate duplicate rows in the result set.
- All rows that meet the search condition specified in the WHERE clause of the SELECT statement are returned in the result set, unless you have specified the DISTINCT clause.
- The combination of values in the select list determines distinctiveness.
- Rows that contain any unique combination of values are retrieved and returned in the result set.
- The DISTINCT clause sorts the result set in random order unless you have included an ORDER BY clause.
- If you specify a DISTINCT clause, the ORDER by clause may include only the columns listed in the select list of the SELECT statement.

Example

```
USE AdventureWorks ;  
GO  
SELECT DISTINCT Title  
FROM HumanResources.Employee ;  
GO
```

Changing Column Names

- Create more readable column names by using the AS keyword to replace default column names with aliases in the select list.
- By default, columns that are based on expressions do not have column names. Use column aliases to give names to columns that are based on expressions.
- Place single quotation marks around column aliases that contain blank spaces or that do not conform to SQL Server object naming conventions.
- You can include up to 128 characters in a column alias.
- You can use column aliases in the ORDER BY clause of the SELECT statement but you cannot use column aliases in the WHERE, GROUP BY, or HAVING clauses of the SELECT statement.
- Partial syntax:

`{ column_name | expression } [AS] column_alias`

Example

```
USE AdventureWorks ;  
GO  
SELECT Title as 'Job Title'  
FROM HumanResources.Employee ;  
GO
```