IT-234 – database concepts

UNIT 6 – USING SQL COMMANDS TO QUERY EXISTING DATA

overview

This unit further examines the use of data manipulation language (DML) to query data in an existing database.

You will establish and work with a new database called Northwind in this unit.

overview

A database deployment script for Northwind is provided, which you will execute within the Microsoft SQL Server Management Studio (SSMS) application.

This deployment script contains both the database structures and data required for the unit assignment.

overview

Once you have data in the tables, what can you do with it?

This leads to the fundamental purpose of a database, which is to effectively facilitate data retrieval by users.

You will examine different SQL syntax for selecting data into a result set.

There are many ways to limit and format the result set into exactly what you require.

overview

The quandary you must overcome is to determine if the result set has returned correct results or not.

A well-formed SQL query should return something.

But you will need to analyze the results to determine if what was returned is what you asked for and if what you asked for is really what you wanted.

Sounds confusing at first, but do not worry; a little practice is all you will need.

overview

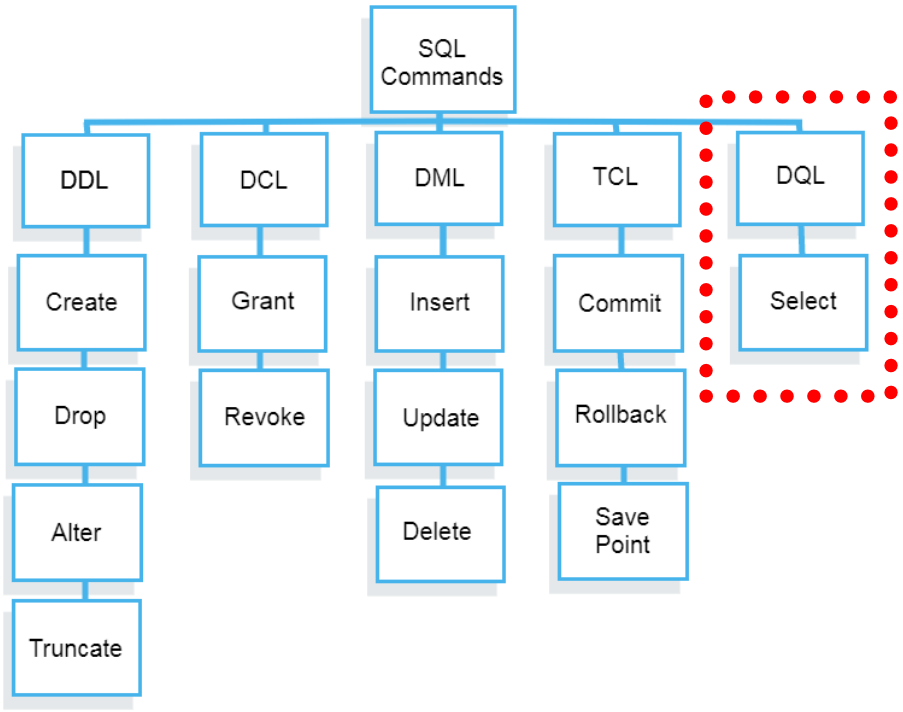
After completing this unit, you should be able to:

- Use DML commands to query existing tables.

Types of SQL

- Here are five types of widely used SQL queries.
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)
  - Data Control Language(DCL)
  - Transaction Control Language(TCL)
  - Data Query Language (DQL)

TYPES OF SQL


SQL Commands
DDL
Create
Drop
Alter
Truncate
DCL
Grant
Revoke
DML
Insert
Update
Delete
TCL
Commit
Rollback
Save Point
DQL
Select

Data Query Language (DQL)

DQL commands are basically SELECT statements.

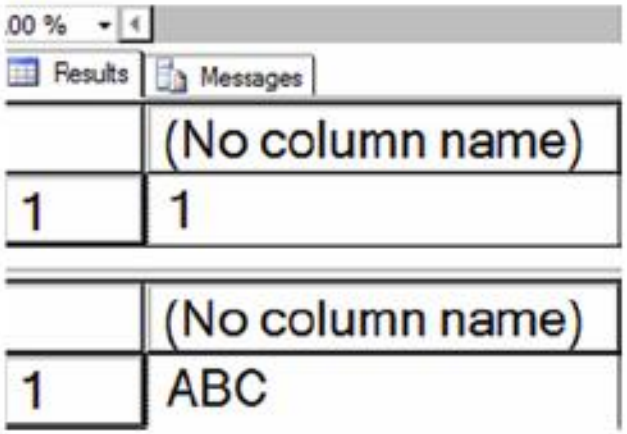
SELECT statements let you query the database to find information in one or more tables and return the query as a result set.

A result set is an array structure; or more precisely, a result set is a two-dimensional array.

Data Query Language (DQL)

- Clauses of the SELECT statement:
  - **SELECT**
    - List the columns (and expressions) to be returned from the query
  - **FROM**
    - Indicate the table(s) or view(s) from which data will be obtained
  - **WHERE**
    - Indicate the conditions under which a row will be included in the result
  - **GROUP BY**
    - Indicate categorization of results
  - **HAVING**
    - Indicate the conditions under which a category (group) will be included
  - **ORDER BY**
    - Sorts the result according to specified criteria

```
SELECT 1;
SELECT 'ABC';
```



00 %

Results | Messages

	(No column name)
1	1

	(No column name)
1	ABC

SELECT Statements

Selecting a Literal Value

Perhaps the simplest form of a `SELECT` statement is that used to return a literal value. A *literal value* is one that you specify exactly. It is not data that come from the database.

Using the SELECT Statement

You use the `SELECT` statement to retrieve data from SQL Server. T-SQL requires only the word `SELECT` followed by at least one item in what is called a *select-list*.

SELECT Statements

Retrieving from a Table

You will usually want to retrieve data from a table instead of literal values. After all, if you already know what value you want, you probably don't need to execute a query to get that value.

```
SELECT <column1>, <column2> FROM <schema>.<table>;
```

```
USE AdventureWorks;
GO
SELECT BusinessEntityID, JobTitle
FROM HumanResources.Employee;
```

	BusinessEntityID	JobTitle
1	1	Chief Executive Officer
2	2	Vice President of Engineering
3	3	Engineering Manager
4	4	Senior Tool Designer
5	5	Design Engineer
6	6	Design Engineer
7	7	Research and Development Manager
8	8	Research and Development Engineer
9	9	Research and Development Engineer
10	10	Research and Development Manager

EXAMPLE

Product		
ProductID	Name	ListPrice
1	Widget	2.99
2	Gizmo	1.79
3	Thingybob	3.49

Select All Columns

```
SELECT * FROM Product;
```

ProductID	Name	ListPrice
1	Widget	2.99
2	Gizmo	1.79
3	Thingybob	3.49

EXAMPLE

Product		
ProductID	Name	ListPrice
1	Widget	2.99
2	Gizmo	1.79
3	Thingybob	3.49

Select Specific Columns

```
SELECT Name, ListPrice
FROM Product;
```

Name	ListPrice
Widget	2.99
Gizmo	1.79
Thingybob	3.49

EXAMPLE

Product		
ProductID	Name	ListPrice
1	Widget	2.99
2	Gizmo	1.79
3	Thingybob	3.49

```
SELECT Name AS Product, ListPrice * 0.9 AS SalePrice
FROM Product;
```

Expressions & Aliases

Product	SalePrice
Widget	2.69
Gizmo	1.61
Thingybob	3.14

SELECT Statements

Adding a WHERE Clause

To filter the rows returned from a query, you will add a `WHERE` clause to your `SELECT` statement. The database engine processes the `WHERE` clause second, right after the `FROM` clause. The `WHERE` clause will contain expressions, called *predicates*, that can be evaluated to `TRUE`, `FALSE`, or `UNKNOWN`.

Filtering Data

Usually an application requires only a fraction of the rows from a table at any given time. For example, an order-entry application that shows the order history will often need to display the orders for only one customer at a time. There might be millions of orders in the database, but the operator of the software will view only a handful of rows instead of the entire table. Filtering data is a very important part of T-SQL.

```
SELECT <column1>,<column2>
FROM <schema>.<table>
WHERE <column> = <value>;
```

EXAMPLE

Product		
ProductID	Name	ListPrice
1	Widget	2.99
2	Gizmo	1.79
3	Thingybob	3.49

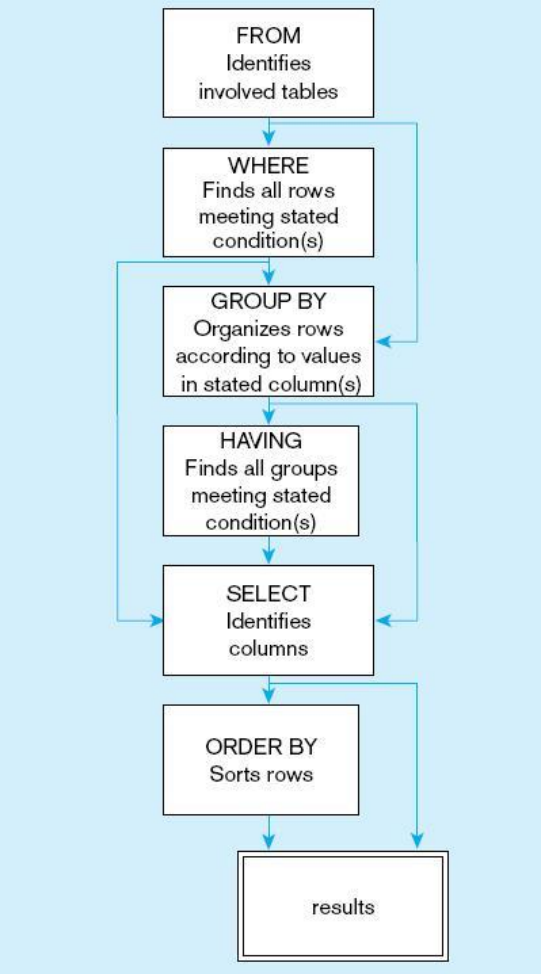
Find products with list price less than $2.75

```
SELECT Name, ListPrice
FROM Product
WHERE ListPrice < 2.75;
```

Name	ListPrice
Gizmo	1.79

Comparison Operators in SQL

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
!=	Not equal to



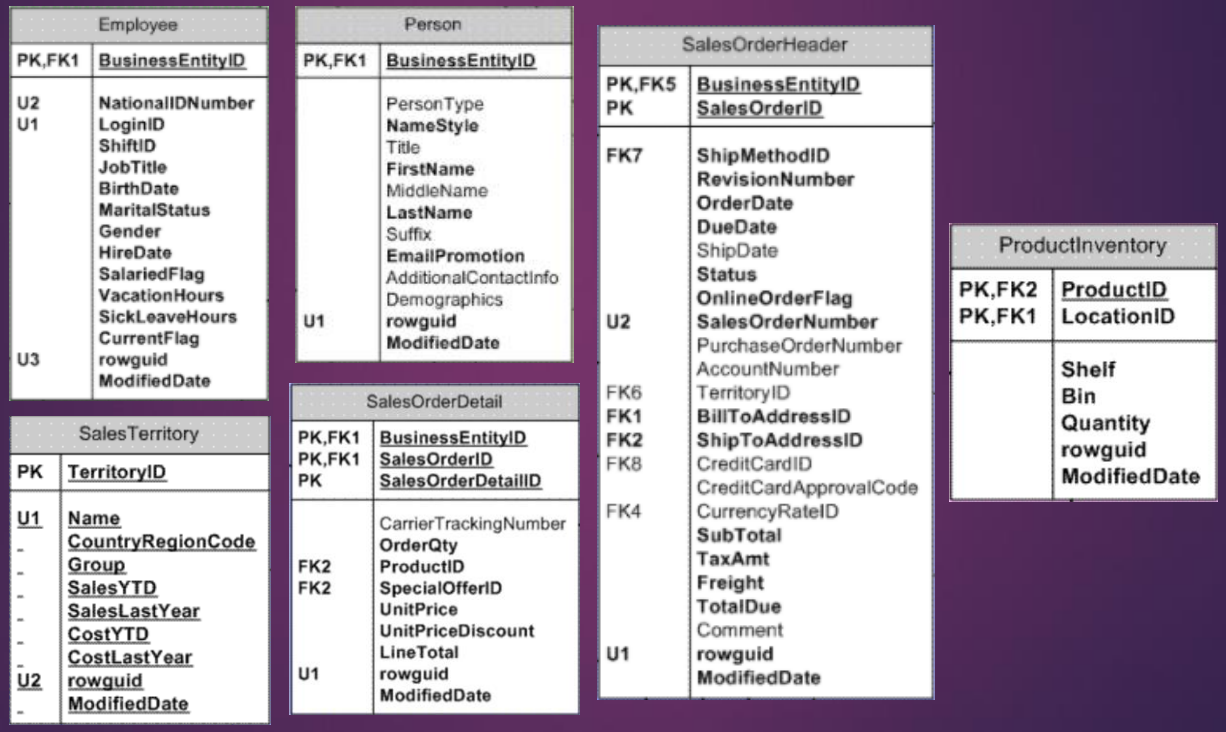
Data Query Language (DQL)

SQL STATEMENT PROCESSING ORDER

Working with Data Types
Transact-SQL Data Types

Exact Numeric	Approximate Numeric	Character	Date/Time	Binary	Other
tinyint	float	char	date	binary	cursor
smallint	real	varchar	time	varbinary	hierarchyid
int		text	datetime	image	sql_variant
bigint		nchar	datetime2		table
bit		nvarchar	smalldatetime		timestamp
decimal/numeric		ntext	datetimeoffset		uniqueidentifier
numeric					xml
money					geography
smallmoney					geometry

SELECT statement example tables



SELECT Statement Examples

	CustomerID	SalesOrderID
1	11000	43793
2	11000	51522
3	11000	57418

	CustomerID	SalesOrderID
1	11000	43793

	CustomerID	SalesOrderID	OrderDate
1	27645	43702	2005-07-02 00:00:00.000
2	16624	43703	2005-07-02 00:00:00.000
3	11005	43704	2005-07-02 00:00:00.000
4	11011	43705	2005-07-02 00:00:00.000

	BusinessEntityID	LoginID	JobTitle
1	1	adventure-works\ken0	Chief Executive Officer

```
--1
SELECT CustomerID, SalesOrderID
FROM Sales.SalesOrderHeader
WHERE CustomerID = 11000;

--2
SELECT CustomerID, SalesOrderID
FROM Sales.SalesOrderHeader
WHERE SalesOrderID = 43793;

--3
SELECT CustomerID, SalesOrderID, OrderDate
FROM Sales.SalesOrderHeader
WHERE OrderDate = '2005-07-02';

--4
SELECT BusinessEntityID, LoginID, JobTitle
FROM HumanResources.Employee
WHERE JobTitle = 'Chief Executive Officer';
```

Using WHERE Clauses with Alternate Operators

Within `WHERE` clause expressions, you can use many comparison operators, not just the equals sign. Books Online lists the following operators:

> *(greater than)*

< *(less than)*

= *(equals)*

<= *(less than or equal to)*

>= *(greater than or equal to)*

!= *(not equal to)*

<> *(not equal to)*

!< *(not less than)*

!> *(not greater than)*

```
--Using a DateTime column
--1
SELECT CustomerID, SalesOrderID, OrderDate
FROM Sales.SalesOrderHeader
WHERE OrderDate > '2005-07-05';

--2
SELECT CustomerID, SalesOrderID, OrderDate
FROM Sales.SalesOrderHeader
WHERE OrderDate < '2005-07-05';

--3
SELECT CustomerID, SalesOrderID, OrderDate
FROM Sales.SalesOrderHeader
WHERE OrderDate >= '2005-07-05';

--4
SELECT CustomerID, SalesOrderID, OrderDate
FROM Sales.SalesOrderHeader
WHERE OrderDate <> '2005-07-05';
```

SELECT Statement examples

```
--Using a string column
--10
SELECT BusinessEntityID, FirstName
FROM Person.Person
WHERE FirstName <> 'Catherine';

--11
SELECT BusinessEntityID, FirstName
FROM Person.Person
WHERE FirstName != 'Catherine';

--12
SELECT BusinessEntityID,
FROM Person.Person
WHERE FirstName > 'M';

--13
SELECT BusinessEntityID,
FROM Person.Person
WHERE FirstName !> 'M';
```

```
--5
SELECT CustomerID, SalesOrderID, OrderDate
FROM Sales.SalesOrderHeader
WHERE OrderDate != '2005-07-05';

--Using a number column
--6
SELECT SalesOrderID, SalesOrderDetailID, OrderQty
FROM Sales.SalesOrderDetail
WHERE OrderQty > 10;

--7
SELECT SalesOrderID, SalesOrderDetailID, OrderQty
FROM Sales.SalesOrderDetail
WHERE OrderQty <= 10;

--8
SELECT SalesOrderID, SalesOrderDetailID, OrderQty
FROM Sales.SalesOrderDetail
WHERE OrderQty <> 10;

--9
SELECT SalesOrderID, SalesOrderDetailID, OrderQty
FROM Sales.SalesOrderDetail
WHERE OrderQty != 10;
```

SELECT Statement Examples

Using BETWEEN

`BETWEEN` is another useful operator you can use in the `WHERE` clause to specify an inclusive range of values. It is frequently used with dates but can be used with string and numeric data as well. Here is the syntax for `BETWEEN`:

```
SELECT <column1>,<column2>
FROM <schema>.<table>
WHERE <column> BETWEEN <value1> AND <value2>;
```

	CustomerID	SalesOrderID	OrderDate
1	27645	43702	2005-07-02 00:00:00.000
2	16624	43703	2005-07-02 00:00:00.000
3	11005	43704	2005-07-02 00:00:00.000
4	11011	43705	2005-07-02 00:00:00.000

	CustomerID	SalesOrderID	OrderDate
1	25000	73018	2008-06-15 00:00:00.000
2	25001	61662	2008-01-08 00:00:00.000
3	25002	61397	2008-01-03 00:00:00.000
4	25003	60269	2007-12-18 00:00:00.000

	BusinessEntityID	JobTitle
1	1	Chief Executive Officer
2	5	Design Engineer
3	6	Design Engineer
4	15	Design Engineer

```
--1
SELECT CustomerID, SalesOrderID, OrderDate
FROM Sales.SalesOrderHeader
WHERE OrderDate BETWEEN '2005-07-02' AND '2005-07-04';

--2
SELECT CustomerID, SalesOrderID, OrderDate
FROM Sales.SalesOrderHeader
WHERE CustomerID BETWEEN 25000 AND 25005;

--3
SELECT BusinessEntityID, JobTitle
FROM HumanResources.Employee
WHERE JobTitle BETWEEN 'C' and 'E';
```

SELECT Statements

Using BETWEEN with NOT

To find values outside a particular range of values, you write the `WHERE` clause expression using `BETWEEN` along with the `NOT` keyword. In this case, the query returns any rows outside the range.

	CustomerID	SalesOrderID	OrderDate
1	27645	43702	2005-07-02 00:00:00.000
2	16624	43703	2005-07-02 00:00:00.000
3	11005	43704	2005-07-02 00:00:00.000
4	11011	43705	2005-07-02 00:00:00.000

	CustomerID	SalesOrderID	OrderDate
1	25000	73018	2008-06-15 00:00:00.000
2	25001	61662	2008-01-08 00:00:00.000
3	25002	61397	2008-01-03 00:00:00.000
4	25003	60269	2007-12-18 00:00:00.000

	BusinessEntityID	JobTitle
1	1	Chief Executive Officer
2	5	Design Engineer
3	6	Design Engineer
4	15	Design Engineer

```
--1
SELECT CustomerID, SalesOrderID, OrderDate
FROM Sales.SalesOrderHeader
WHERE OrderDate NOT BETWEEN '2005-07-02' AND '2005-07-04';

--2
SELECT CustomerID, SalesOrderID, OrderDate
FROM Sales.SalesOrderHeader
WHERE CustomerID NOT BETWEEN 25000 AND 25005;

--3
SELECT BusinessEntityID, JobTitle
FROM HumanResources.Employee
WHERE JobTitle NOT BETWEEN 'C' and 'E';
```

SELECT Statements

THE RECORDS BELOW WILL NOT BE IN THE RESULT SETS!!!!

SELECT Statements

Filtering on Date and Time

Some temporal data columns store the time as well as the date. If you attempt to filter on such a column specifying only the date, you may retrieve incomplete results.

```
--1
SELECT ID, MyDate, MyValue
FROM #DateTimeExample
WHERE MyDate = '2009-01-03';

--2
SELECT ID, MyDate, MyValue
FROM #DateTimeExample
WHERE MyDate BETWEEN '2009-01-03 00:00:00' AND '2009-01-03 23:59:59';
```

	ID	MyDate	MyValue

	ID	MyDate	MyValue
1	2	2009-01-03 13:00:00	Trike
2	3	2009-01-03 13:10:00	Bell
3	4	2009-01-03 17:35:00	Seat

SELECT Statements

Using WHERE Clauses with Two Predicates

So far, the examples have shown only one condition or predicate in the `WHERE` clause, but the `WHERE` clause can be much more complex. They can have multiple predicates by using the logical operators AND and OR.

	BusinessEntityID	FirstName	MiddleName	LastName
1	1525	Ken	NULL	Myer
2	203	Ken	L	Myer

	BusinessEntityID	FirstName	MiddleName	LastName
1	1459	Deanna	NULL	Meyer
2	1455	Eric	B.	Meyer
3	1457	Helen	M.	Meyer
4	2140	Ken	NULL	Meyer
5	1523	Dorothy	J.	Myer
6	1525	Ken	NULL	Myer
7	203	Ken	L	Myer
8	2319	Linda	NULL	Myer

```
--1
SELECT BusinessEntityID, FirstName, MiddleName, LastName
FROM Person.Person
WHERE FirstName = 'Ken' AND LastName = 'Myer';

--2
SELECT BusinessEntityID, FirstName, MiddleName, LastName
FROM Person.Person
WHERE LastName = 'Myer' OR LastName = 'Meyer';
```

SELECT Statements

Using the IN Operator

The `IN` operator is very useful when a set of multiple values must be compared to the same column. Follow the `IN` operator with a list of possible values for a column within parentheses. Here is the syntax:

```
SELECT <column1>,<column2>
FROM <schema>.<table>
WHERE <column> IN (<value1>,<value2>);
```

```
--1
SELECT BusinessEntityID,FirstName,MiddleName,LastName
FROM Person.Person
WHERE FirstName = 'Ken' AND
    LastName IN ('Myer','Meyer');

--2
SELECT TerritoryID, Name
FROM Sales.SalesTerritory
WHERE TerritoryID IN (2,2,1,4,5);

--3
SELECT TerritoryID, Name
FROM Sales.SalesTerritory
WHERE TerritoryID NOT IN (2,1,4,5);
```

	BusinessEntityID	FirstName	MiddleName	LastName
1	2140	Ken	NULL	Meyer
2	1525	Ken	NULL	Myer
3	203	Ken	L	Myer

	TerritoryID	Name
1	1	Northwest
2	2	Northeast
3	4	Southwest
4	5	Southeast

	TerritoryID	Name
1	9	Australia
2	6	Canada
3	3	Central
4	7	France
5	8	Germany
6	10	United Kingdom

Using LIKE

Pattern matching is possible by using the `LIKE` operator in the expression instead of equal to (=) or one of the other operators. Most of the time, the percent sign (`%`) is used as a wildcard along with `LIKE` to represent zero or more characters. You will also see the underscore (_) used as a wildcard to replace exactly one character, but it's not used as often.

```
--1
SELECT DISTINCT LastName
FROM Person.Person
WHERE LastName LIKE 'Sand%';
```

	LastName
1	Sandberg
2	Sanders
3	Sandidge
4	Sandoval

```
--2
SELECT DISTINCT LastName
FROM Person.Person
WHERE LastName NOT LIKE 'Sand%';
```

	LastName
1	Abbas
2	Abel
3	Abercrombie
4	Abolrous

```
--3
SELECT DISTINCT LastName
FROM Person.Person
WHERE LastName LIKE '%Z%';
```

	LastName
1	Alvarez
2	Arbelaez
3	Bacalzo
4	Baltazar

```
--4
SELECT DISTINCT LastName
FROM Person.Person
WHERE LastName LIKE 'Bec_';
```

	LastName
1	Beck

SELECT Statements

NULL and three-valued logic

In the database world, `NULL` is used to indicate the absence of any data value. For example, at the time of recording the customer information, the email may be unknown, so it is recorded as `NULL` in the database.

Normally, the result of a logical expression is `TRUE` or `FALSE`. However, when `NULL` is involved in the logical evaluation, the result is `UNKNOWN`. This is called a three-valued logic: `TRUE`, `FALSE`, and `UNKNOWN`.

The results of the following comparisons are `UNKNOWN`:

```
NULL = 0
NULL <> 0
NULL > 0
NULL = NULL
```

The syntax for the IS NULL condition in SQL Server (Transact-SQL) is: `expression IS NULL`

- If *expression* is a NULL value, the condition evaluates to TRUE.
- If *expression* is not a NULL value, the condition evaluates to FALSE.

The syntax for the IS NOT NULL condition in SQL Server (Transact-SQL) is: `expression IS NOT NULL`

- If *expression* is NOT a NULL value, the condition evaluates to TRUE.
- If *expression* is a NULL value, the condition evaluates to FALSE.

SELECT Statements

An Example Illustrating NULL

```
--1 Returns 19,972 rows
SELECT MiddleName
FROM Person.Person;

--2 Returns 291 rows
SELECT MiddleName
FROM Person.Person
WHERE MiddleName = 'B';
```

```
--3 Returns 11,182 but 19,681 were expected
SELECT MiddleName
FROM Person.Person
WHERE MiddleName != 'B';

--4 Returns 19,681
SELECT MiddleName
FROM Person.Person
WHERE MiddleName IS NULL
    OR MiddleName !='B';
```

SELECT Statements

SELECT Statements

Sorting Data

You can specify one or more columns in the `ORDER BY` clause separated by commas. The sort order is ascending by default, but you can specify descending order by using the keyword `DESCENDING` or `DESC` after the column name. You can also specify `ASCENDING` or `ASC` if you wish, but the sort order is ascending by default. Here is the syntax for `ORDER BY`:

```
SELECT <column1>,<column2>
FROM <schema>.<tablename>
ORDER BY <column1>[<sort direction>],<column2> [<sort direction>]
```

```
--1
SELECT ProductID, LocationID
FROM Production.ProductInventory
ORDER BY LocationID;

--2
SELECT ProductID, LocationID
FROM Production.ProductInventory
ORDER BY ProductID, LocationID DESC;
```

	ProductID	LocationID
1	1	1
2	2	1
3	3	1
4	4	1
5	317	1
6	318	1
7	319	1

	ProductID	LocationID
1	1	50
2	1	6
3	1	1
4	2	50
5	2	6
6	2	1
7	3	50
8	3	6
9	3	1

Distinct clause

Sometimes, you may want to get only distinct values in a specified column of a table.

To do this, you use the SELECT DISTINCT clause as follows:

```
SELECT DISTINCT
    column_name
FROM
    table_name;
```

Distinct clause

The query returns only distinct values in the specified column.

In other words, it removes the duplicate values in the column from the result set.

Distinct clause

- If you use multiple columns as follows:

- The query uses the combination of values in all specified columns in the SELECT list to evaluate the uniqueness.

```
SELECT DISTINCT
        column_name1,
        column_name2 ,
        ...
FROM
        table_name;
```

Distinct clause

If you apply the DISTINCT clause to a column that has NULL, the DISTINCT clause will keep only one NULL and eliminates the other.

In other words, the DISTINCT clause treats all NULL “values” as the same value.

Distinct clause examples

sales.customers
* customer_id
first_name
last_name
phone
email
street
city
state
zip_code

```
SELECT
    city
FROM
    sales.customers
ORDER BY
    city;
```

city
Albany
Albany
Albany
Amarillo
Amarillo
Amarillo
Amarillo
Amarillo

DUPLICATES IN RESULTS

DISTINCT one column example

```
SELECT DISTINCT
    city
FROM
    sales.customers
ORDER BY
    city;
```

city
Albany
Amarillo
Amityville
Amsterdam
Anaheim
Apple Valley
Astoria
Atwater

NO DUPLICATES IN RESULTS

Distinct clause examples

sales.customers

- * customer_id
- first_name
- last_name
- phone
- email
- street
- city
- state
- zip_code

```
SELECT
    city,
    state
FROM
    sales.customers
```

city	state
Albany	NY
Albany	NY
Albany	NY
Amarillo	TX
Amarillo	TX
Amarillo	TX
Amarillo	TX
Amarillo	TX
Amityville	NY
Amityville	NY
Amityville	NY
Amityville	NY
Amityville	NY

DUPLICATES IN RESULTS

DISTINCT multiple columns example

```
SELECT DISTINCT
    city,
    state
FROM
    sales.customers
```

city	state
Hopewell Junction	NY
Houston	TX
Howard Beach	NY
Huntington	NY
Huntington Station	NY
Ithaca	NY
Jackson Heights	NY
Jamaica	NY

NO DUPLICATES IN RESULTS

Distinct clause examples

DISTINCT with null values example

sales.customers

- * customer_id
- first_name
- last_name
- phone
- email
- street
- city
- state
- zip_code

```
SELECT DISTINCT
    phone
FROM
    sales.customers
ORDER BY
    phone;
```

phone
NULL
(210) 436-8676
(210) 851-3122
(212) 152-6381
(212) 171-1335
(212) 211-7621
(212) 325-9145
(212) 578-2912

Aggregate functions

An aggregate function performs a calculation one or more values and returns a single value.

The aggregate function is often used with the GROUP BY clause and HAVING clause of the SELECT statement.

Aggregate functions

- The most commonly used aggregate functions are:
  - **COUNT** – counts the number of elements in the group defined
  - **SUM** – calculates the sum of the given attribute/expression in the group defined
  - **AVG** – calculates the average value of the given attribute/expression in the group defined
  - **MIN** – finds the minimum in the group defined
  - **MAX** – finds the maximum in the group defined

production.products

- * product_id
- product_name
- brand_id
- category_id
- model_year
- list_price

```
SELECT
    AVG(list_price) avg_product_price
FROM
    production.products;
```

avg_product_price
1520.591401

```
SELECT
    COUNT(*) product_count
FROM
    production.products
WHERE
    list_price > 500;
```

product_count
213

```
SELECT
    MAX(list_price) max_list_price
FROM
    production.products;
```

max_list_price
11999.99

```
SELECT
    MIN(list_price) min_list_price
FROM
    production.products;
```

min_list_price
89.99

Aggregate function examples

Aggregate function examples

production.stocks

* store_id
* product_id
  quantity

```
SELECT
    product_id,
    SUM(quantity) stock_count
FROM
    production.stocks
GROUP BY
    product_id
ORDER BY
    stock_count DESC;
```

product_id	stock_count
188	86
64	82
109	79
196	79
61	78
182	77
166	77
219	75
142	75
252	75

UNIT 6 ASSIGNMENT

UNIT 6 ASSIGNMENT

- Purpose:
  - You are asked to develop SQL DML statements to address the questions presented below.
  - You must use the Northwind database for this assignment.

UNIT 6 ASSIGNMENT

- Assignment Instructions:

- Deploy the Northwind database into your Microsoft SQL Server instance by executing the provided SQL script in a Microsoft SQL Server Management Studio (SSMS) query window.

UNIT 6 ASSIGNMENT

Assignment Instructions:

- Once the Northwind database is successfully deployed, generate SQL statements to address the assignment problems.
- Use the provided database design diagram as a guide in forming your statements.

UNIT 6 ASSIGNMENT

Assignment Instructions:

- Problem 1: Display the product records with unit prices equal to or greater than $100.

	ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
1	29	Thüringer Rostbratwurst	12	6	50 bags x 30 sausgs.	123.79	0	0	0	1
2	38	Côte de Blaye	18	1	12 - 75 cl bottles	263.50	17	0	15	0

UNIT 6 ASSIGNMENT

Assignment Instructions:

- Problem 2: Present the unit value minimum, maximum, average, and standard deviation for current and discontinued products. Group the results based on the "Discontinued"

	Discontinued	Minimum Unit Price	Maximum Unit Price	Average Unit Price	Unit Price Standard Deviation
1	0	2.50	263.50	26.7343	32.464315951983
2	1	4.50	123.79	47.255	41.7765422900733

UNIT 6 ASSIGNMENT

Assignment Instructions:

- Problem 3: List suppliers with names starting with "G" in alphabetical order.

	CompanyName
1	Gai pâturage
2	G'day, Mate
3	Grandma Kelly's Homestead

UNIT 6 ASSIGNMENT

- Assignment Instructions:
- Problem 4: Display the full names of all Employees in FirstName, “ ” , LastName format.

	Employee Name
1	Nancy Davolio
2	Andrew Fuller
3	Janet Leverling
4	Margaret Peacock
5	Steven Buchanan
6	Michael Suyama
7	Robert King
8	Laura Callahan
9	Anne Dodsworth

UNIT 6 ASSIGNMENT

- Assignment Instructions:

- Problem 5: Display all of the supplier names and HomePage values for NULL HomePage values. Present the results in alphabetical order by supplier name.

	CompanyName	age
1	Aux joyeux ecclésiastiques	
2	Bigfoot Breweries	NULL
3	Cooperativa de Quesos 'Las Cabras'	NULL
4	Escargots Nouveaux	NULL
5	Exotic Liquids	NULL
6	Forêts d'érables	NULL
7	Gai pâturage	NULL
8	Grandma Kelly's Homestead	NULL
9	Heli Süßwaren GmbH & Co. KG	NULL
10	Karkki Oy	NULL
11	Leka Trading	NULL
12	Lyngbysild	NULL
13	Ma Maison	NULL
14	New England Seafood Cannery	NULL
15	Nord-Ost-Fisch Handelsgesellschaft mbH	NULL
16	Norske Meierier	NULL
17	Pasta Buttini s.r.l.	NULL
18	Pavlova, Ltd.	NULL
19	PB Knäckebröd AB	NULL
20	Refrescos Americanas LTDA	NULL
21	Specialty Biscuits, Ltd.	NULL
22	Svensk Sjöföda AB	NULL
23	Tokyo Traders	NULL
24	Zaanse Snoepfabriek	NULL

UNIT 6 ASSIGNMENT

	Country	Number of Customers
1	Brazil	9
2	France	11
3	Germany	11
4	Mexico	5
5	Spain	5
6	UK	7
7	USA	13

- Assignment Instructions:

- Problem 6: Get the number of Northwind customers in each country that has five or more customers. Present the results in alphabetical order by country name.

UNIT 6 ASSIGNMENT

- Assignment Instructions:

- Problem 7: Get the total product costs (i.e., multiplying the unit price by quantity) for ProductID = 20 along with the associated orders IDs from the OrderDetails table. Provide the results in ascending order based on the order ID.

	OrderID	ProductID	Prod...st
1	10252	20	2592.00
2	10272	20	388.80
3	10292	20	1296.00
4	10372	20	777.60
5	10384	20	1814.40
6	10514	20	3159.00
7	10523	20	1215.00
8	10593	20	1701.00
9	10660	20	1701.00
10	10828	20	405.00
11	10878	20	1620.00
12	10927	20	405.00
13	10953	20	4050.00
14	10986	20	1215.00
15	11021	20	1215.00
16	11077	20	81.00

UNIT 6 ASSIGNMENT

Assignment Instructions:

- Problem 8: Select all of the cities where there are employees with no duplicates values displayed. Show the cities in alphabetical order.

	City
1	Kirkland
2	London
3	Redmond
4	Seattle
5	Tacoma

UNIT 6 ASSIGNMENT

- Assignment Instructions:

- Problem 9: List first and last names for employees who live in London, Seattle, or Redmond. Show the employee's city in the output.

	First Name	Last Name	City
1	Nancy	Davolio	Seattle
2	Margaret	Peacock	Redmond
3	Steven	Buchanan	London
4	Michael	Suyama	London
5	Robert	King	London
6	Laura	Callahan	Seattle
7	Anne	Dodsworth	London

UNIT 6 ASSIGNMENT

Assignment Instructions:

- Problem 10: Display all product IDs and names for out-of-stock products having a unit price in the range of $20 to $25. Show the units-in-stock value in the results.

	ProductID	ProductName	UnitPrice	UnitsInStock
1	5	Chef Anton's Gumbo Mix	21.35	0

UNIT 6 ASSIGNMENT

Assignment Requirements:

Microsoft SQL Server Express and SQL Server Management Studio (SSMS) MUST be installed to complete this Assignment.

Compose your Assignment in a Word document.

UNIT 6
ASSIGNMENT

Assignment Requirements:

- Embed the screenshots of your SQL statements and confirmatory output (e.g., query results) into the Word document.
- The assignment is due by the final day of the Unit 6 week.

UNIT 6 ASSIGNMENT

Directions for Submitting Your Assignment:

- Name your assignment document according to this convention: **IT234_<*YourName*>_Unit6.docx** (replace **<*YourName*>** with your full name).
- Submit your completed assignment to the Unit 6 Assignment Dropbox by the final day of the Unit 6 week.
- Review the Unit 6 Assignment Rubric before beginning this activity.

Any Questions?