Instructor Notes:

Instructor Notes:

Explain the lesson coverage

Lesson Objectives

- In this lesson, you will learn:
  - Transact-SQL Programming Language
  - Types of Transact-SQL Statements
  - Transact-SQL Syntax Elements
  - Restricting rows
  - Operators
  - Functions – String, Date, Mathematical, System, Others
  - Grouping and summarizing data

Instructor Notes:

Transact-SQL Programming Language

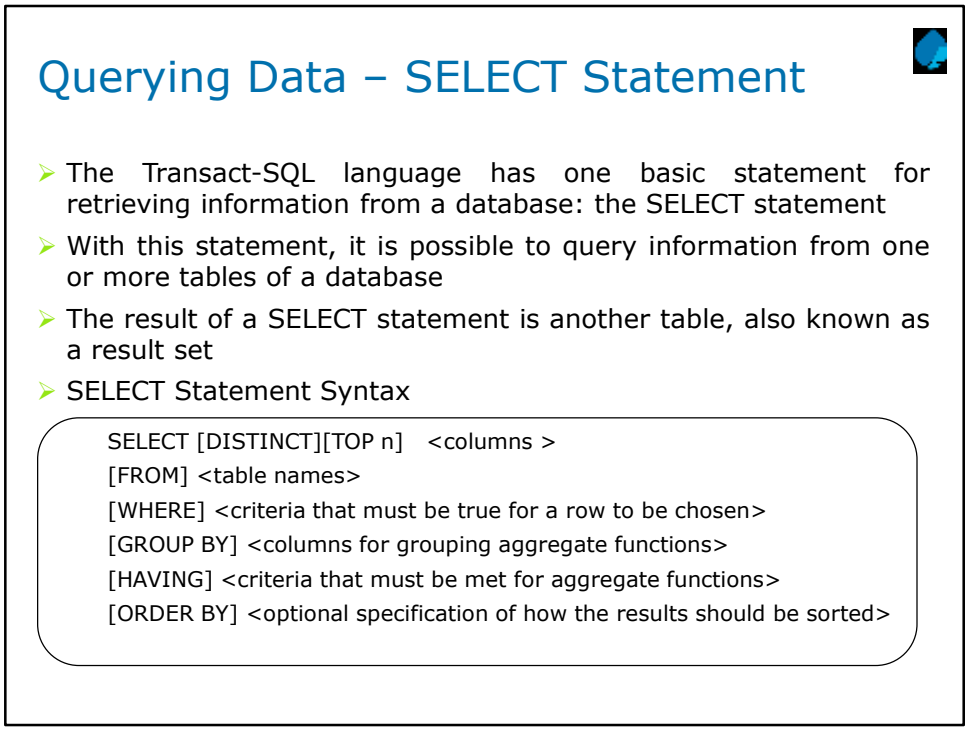
- Transact SQL, also called T-SQL, is Microsoft's extension to the ANSI SQL language
- Structured Query Language(SQL), is a standardized computer language that was originally developed by IBM
- T-SQL expands on the SQL standard to include procedural programming, local variables, various support functions for string processing, date processing, mathematics, etc.
- Transact-SQL is central to using SQL Server
- All applications that communicate with SQL Server do so by sending Transact-SQL statements to the server, regardless of the user interface of the application

Transact-SQL Programming Language

T-SQL (Transact SQL) is a proprietary SQL extension used by Microsoft SQL Server. T-SQL adds extensions for procedural programming.

Transact-SQL is central to using SQL Server. All applications that communicate with an instance of SQL Server do so by sending Transact-SQL statements to the server, regardless of the user interface of the application.

Instructor Notes:



Querying Data – SELECT Statement

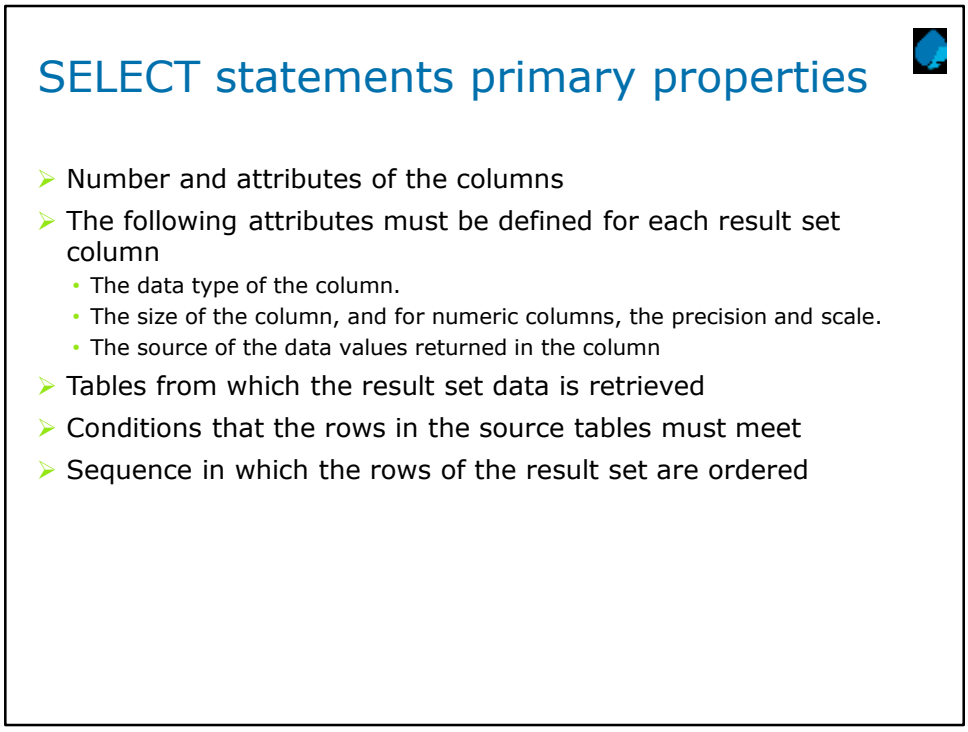
You wouldn't have any reason to store data if you didn't want to retrieve it. Relational databases gained wide acceptance principally because they enabled users to query, or access, data easily, without using predefined, rigid navigational paths. Indeed, the acronym SQL stands for Structured Query Language. Microsoft SQL Server provides rich and powerful query capabilities.

The SELECT statement is the most frequently used SQL command and is the fundamental way to query data. The SQL syntax for the SELECT statement is intuitive—at least in its simplest forms—and resembles how you might state a request in English. As its name implies, it is also structured and precise. However, a SELECT statement can be obscure or even tricky. As long as you write a query that's syntactically correct, you get a result.

The basic form of SELECT, which uses brackets ([]) to identify optional items, is shown in the slide.

Notice that the only clause that must always be present is the verb SELECT; the other clauses are optional. For example, if the entire table is needed, you don't need to restrict data using certain criteria, so you can omit the WHERE clause.

Instructor Notes:

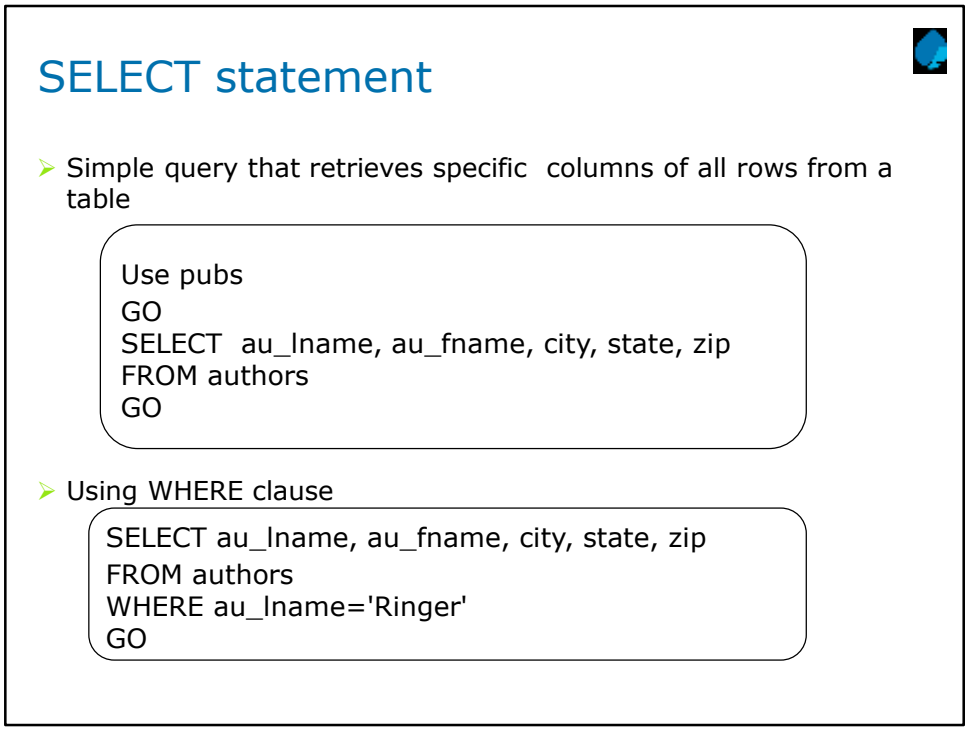


SELECT statements primary properties

The full syntax of the SELECT statement is complex, but most SELECT statements describe four primary properties of a result set:

- The number and attributes of the columns in the result set.
- The following attributes must be defined for each result set column:
  - The data type of the column.
  - The size of the column, and for numeric columns, the precision and scale.
  - The source of the data values returned in the column.
- The tables from which the result set data is retrieved, and any logical relationships between the tables.
- The conditions that the rows in the source tables must meet to qualify for the SELECT. Rows that do not meet the conditions are ignored.
- The sequence in which the rows of the result set are ordered.

Instructor Notes:



SELECT Statement

Here's a simple query that retrieves all columns of all rows from the *authors* table in the *pubs* sample database:

```
SELECT  au_lname, au_fname, city, state, zip
FROM authors
```

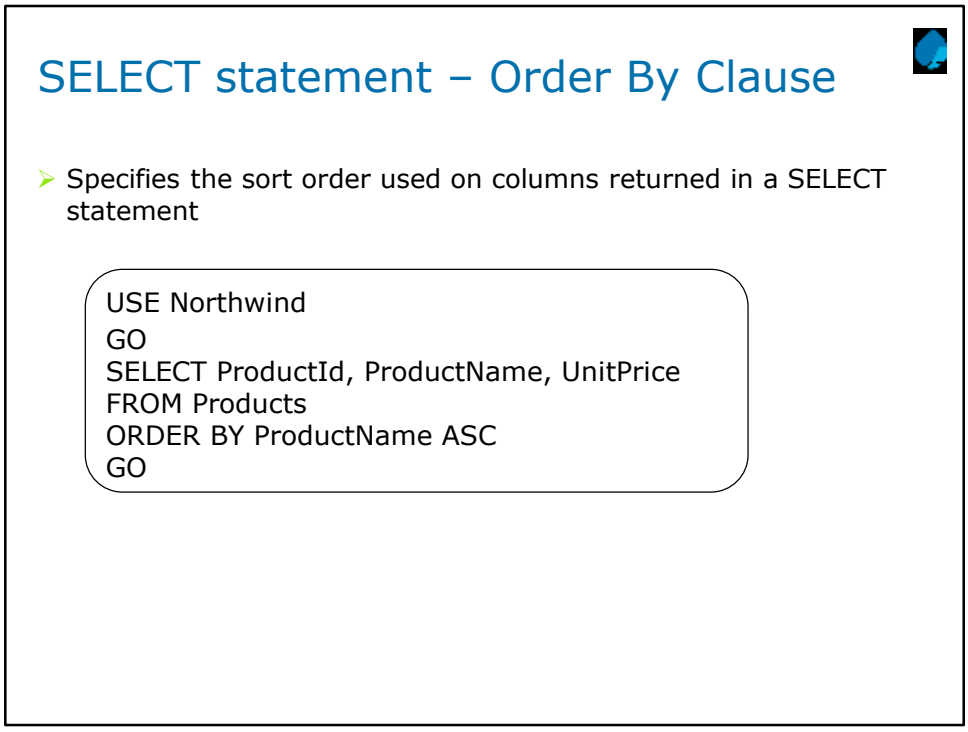
You can specify a query from one table return only certain columns or rows that meet your stated criteria. In the select list, you specify the exact columns you want, and then in the WHERE clause, you specify the criteria that determine whether a row should be included in the answer set. Still using the *pubs* sample database, suppose we want to find the first name and the city, state, and zip code of residence for authors whose last name is *Ringer*:

```
SELECT au_lname, au_fname, city, state, zip
FROM authors
WHERE au_lname='Ringer'
```

Retrieving only Anne Ringer's data requires an additional expression that is combined (AND'ed) with the original. In addition, we'd like the output to have more intuitive names for some columns, so the query would be:

```
SELECT 'Last Name'=au_lname, 'First'=au_fname, city, stat
e, zip
FROM authors
WHERE au_lname='Ringer' AND au_fname='Anne‘
```

Instructor Notes:



SELECT Statement – Order By Clause

Specifies the sort order used on columns returned in a SELECT statement. Default is ASCending sort, DESC is used to sort in a descending manner

```
SELECT ProductId, ProductName, UnitPrice
FROM Products
ORDER BY ProductName DESC
```

ORDER BY is the last operation to be done, before the result set is submitted to the client, therefore ORDER BY cannot be used in certain places like subquery, SELECT INTO etc.

Instructor Notes:

Use of DISTINCT

- DISTINCT is used to eliminate duplicate rows
- Precedes the list of columns to be selected from the table(s)
- The DISTINCT considers the values of all the columns as a single unit and evaluates on a row-by-row basis to eliminate any redundant rows

- Example

```
SELECT DISTINCT Region
FROM  Northwind.dbo.Employees
```

Instructor Notes:

Demo

- Using SELECT Statement

Instructor Notes:

Use of Operators

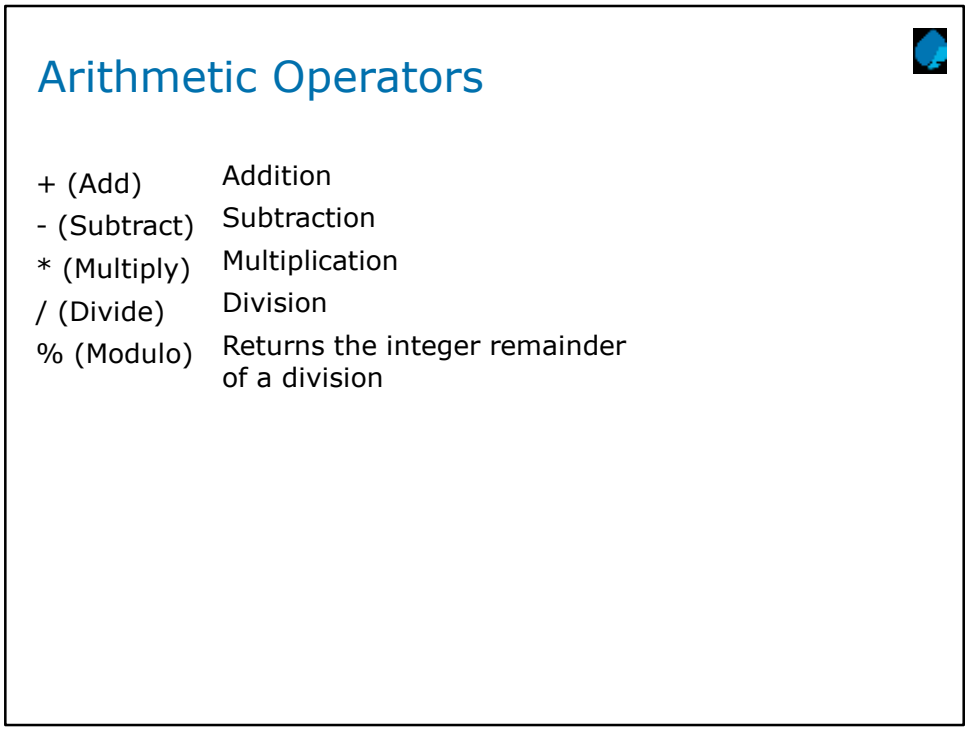
- An operator is a symbol specifying an action that is performed on one or more expressions.
- Arithmetic Operators
- Logical Operators
- Assignment Operator
- String Concatenation Operator
- Comparison Operators
- Compound Assignment Operator

Use of Operators

An operator is a symbol specifying an action that is performed on one or more expressions. The following tables lists the operator categories that SQL Server uses.

- Arithmetic Operators
- Logical Operators
- Assignment Operator
- String Concatenation Operator
- Comparison Operators
- Compound Assignment Operator
- Bitwise Operators
- Unary Operators

Instructor Notes:



Arithmetic operators perform mathematical operations on two expressions of one or more of the data types of the numeric data type category.

Operator	**Meaning**
+ (Add)	Addition
- (Subtract)	Subtraction
* (Multiply)	Multiplication
/ (Divide)	Division
% (Modulo)	Returns the integer remainder of a division.

For example, 12 % 5 = 2 because the remainder of 12 divided by 5 is 2.The plus (+) and minus (-) operators can also be used to perform arithmetic operations on **datetime** and **smalldatetime** values.

Instructor Notes:

Logical Operators

Operator	Meaning
ALL	TRUE if all of a set of comparisons are TRUE
AND	TRUE if both Boolean expressions are TRUE
ANY	TRUE if any one of a set of comparisons are TRUE
BETWEEN	TRUE if the operand is within a range
EXISTS	TRUE if a subquery contains any rows
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern.
NOT	Reverses the value of any other Boolean operator
OR	TRUE if either Boolean expression is TRUE.

Logical operators test for the truth of some condition. Logical operators, like comparison operators, return a **Boolean** data type with a value of TRUE, FALSE, or UNKNOWN.

Operator	**Meaning**
ALL	TRUE if all of a set of comparisons are TRUE.
AND	TRUE if both Boolean expressions are TRUE.
ANY	TRUE if any one of a set of comparisons are TRUE.
BETWEEN	TRUE if the operand is within a range.
EXISTS	TRUE if a subquery contains any rows.
IN	TRUE if the operand is equal to one of a list of expr.
LIKE	TRUE if the operand matches a pattern.
NOT	Reverses the value of any other Boolean operator.
OR	TRUE if either Boolean expression is TRUE.
SOME	TRUE if some of a set of comparisons are TRUE.

Example for IN:

```
USE Northwind
SELECT FirstName, LastName,  Title
FROM  Employees
WHERE     Title  IN  ('Design  Engineer',  'Tool  Designer',
'Marketing Assistant');
GO
```

Instructor Notes:

Like Operators

- Pattern: The pattern to search for in match_expression.

Wildcard character	Example
% Any string of zero or more characters.	WHERE title LIKE '%computer%' finds all book titles with the word 'computer' anywhere in the book title
_ (underscore)Any single character.	WHERE au_fname LIKE '_ean' finds all four-letter first names that end with ean, such as Dean or Sean.

Determines whether a given character string matches a specified pattern. A pattern can include regular characters and wildcard characters. During pattern matching, regular characters must match exactly the characters specified in the character string. Wildcard characters, however, can be matched with arbitrary fragments of the character string. Using wildcard characters makes the LIKE operator more flexible than using the = and != string comparison operators.

For example to list out all titles which starts with a The , the query would be

```
SELECT title, price
FROM pubs.dbo.titles
WHERE title LIKE 'The%'
```

The output will not be the same if the pattern is given as

```
SELECT title, price
FROM pubs.dbo.titles
WHERE title LIKE '%The%'
```

This would list all titles have the somewhere - beginning , middle or end

Instructor Notes:

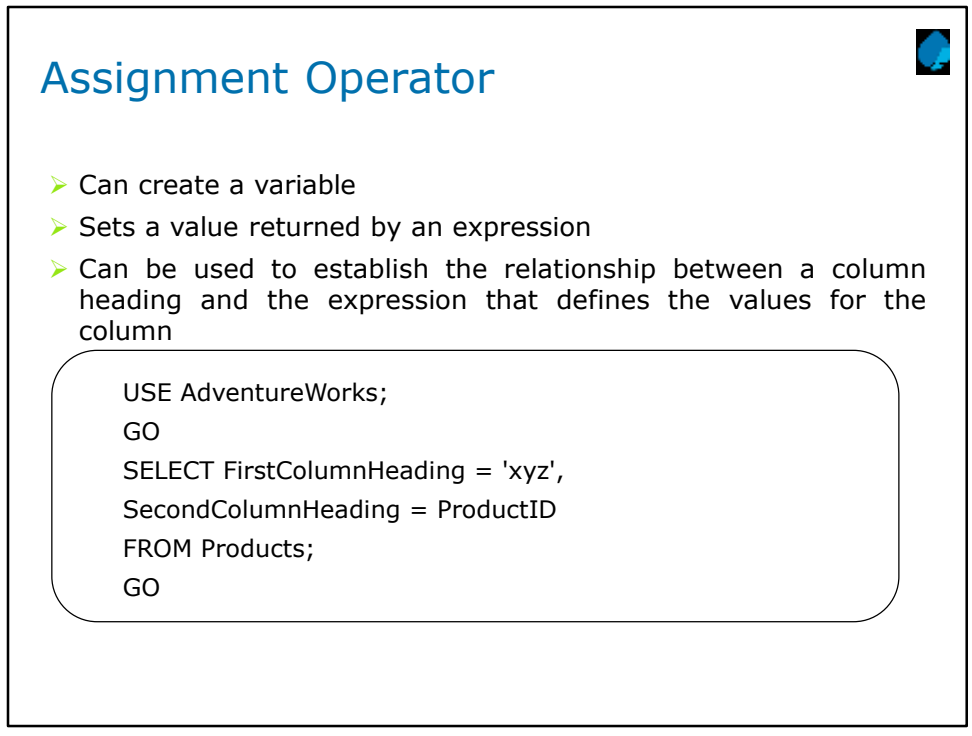
Working with NULL Values

- NULL values are treated differently from other values
- NULL is used as a placeholder for unknown or inapplicable values
- We have to use the IS NULL and IS NOT NULL operators to test for NULL Values

```
SELECT LastName, FirstName, Address
    FROM Persons
WHERE Address IS NULL
```

```
SELECT LastName, FirstName, Address
    FROM Persons
WHERE Address IS NOT NULL
```

Instructor Notes:



The equal sign (=) is the only Transact-SQL assignment operator. In the following example, the @MyCounter variable is created, and then the assignment operator sets @MyCounter to a value returned by an expression.

```
DECLARE @MyCounter INT;
SET @MyCounter = 1;
```

The assignment operator can also be used to establish the relationship between a column heading and the expression that defines the values for the column. The following example displays the column headings FirstColumnHeading and SecondColumnHeading. The string xyz is displayed in the FirstColumnHeading column heading for all rows. Then, each product ID from the Product table is listed in the SecondColumnHeading column heading.

```
USE AdventureWorks;
SELECT FirstColumnHeading = 'xyz',
SecondColumnHeading = ProductID
FROM Products;
```

Instructor Notes:

Comparison Operator

=	(Equals) Equal to
>	(Greater Than) Greater than
<	(Less Than) Less than
>=	(Greater Than or Equal To) Greater than or equal to
<=	(Less Than or Equal To) Less than or equal to
<>	(Not Equal To) Not equal to
!=	(Not Equal To) Not equal to
!<	(Not Less Than) Not less than
!>	(Not Greater Than) Not greater than

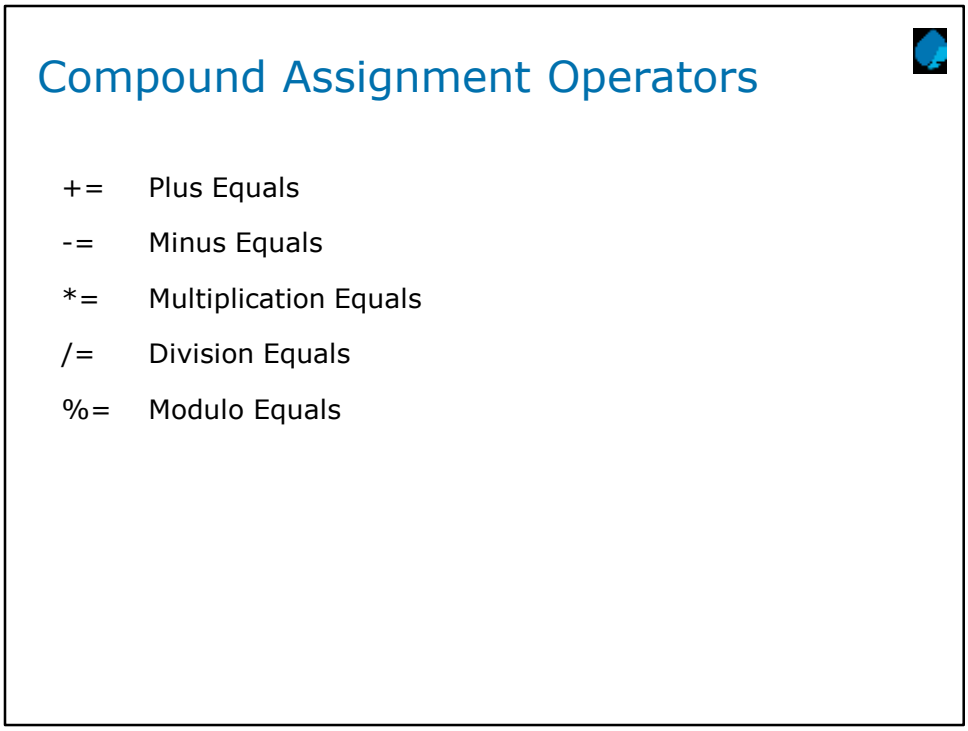
Comparison operators test whether two expressions are the same. Comparison operators can be used on all expressions except expressions of the **text**, **ntext**, or **image** data types. The following table lists the Transact-SQL comparison operators.

Operator	**Meaning**
=	(Equals) Equal to
>	(Greater Than) Greater than
<	(Less Than) Less than
>=	(Greater Than or Equal To) Greater than or equal to
<=	(Less Than or Equal To) Less than or equal to
<>	(Not Equal To) Not equal to
!=	(Not Equal To) Not equal to (not SQL-92 standard)
!<	(Not Less Than) Not less than (not SQL-92 standard)
!>	(Not Greater Than) Not greater than (not SQL-92 standard)

Example:

```
use Northwind;
SELECT ProductID, ProductName, UnitPrice AS Price
FROM Products
WHERE SupplierID = 7 AND UnitsInStock < 25
ORDER BY ProductName ASC ;
```

Instructor Notes:



Compound Assignment Operators

Compound assignment operators help abbreviate code that assigns a value to a column or a variable. The new operators are:

- += (plus equals)
- -= (minus equals)
- *= (multiplication equals)
- /= (division equals)
- %= (modulo equals)

You can use these operators wherever assignment is normally allowed—for example, in the SET clause of an UPDATE statement or in a SET statement that assigns values to variables. The following code example demonstrates the use of the += operator:

```
DECLARE @price AS MONEY = 10.00;
SET @price += 2.00;
SELECT @price;
```

This code sets the variable @price to its current value, 10.00, plus 2.00, resulting in 12.00.

Instructor Notes:

Demo

- Using LIKE operator
- Working with commonly used operators

Instructor Notes:

Using System Functions

- String Functions
- Date and Time Functions
- Mathematical Functions
- Aggregate Functions
- System Functions

Using System Functions

1. **String Functions -** Perform operations on a string (**char** or **varchar**) input value and return a string or numeric value.
2. **Date and Time Functions -** Perform operations on a date and time input values and return string, numeric, or date and time values.
3. **Mathematical Functions -** Perform calculations based on input values provided as parameters to the functions, and return numeric values.
4. **Aggregate Functions -** You can use aggregate functions to calculate and summarize data.
5. **System Functions -** Perform operations and return information about values, objects, and settings in an instance of SQL Server.

Instructor Notes:

Using System Functions – String Functions

- STR - Returns character data converted from numeric data
- REPLACE - Replaces all occurrences of a specified string value with another string value
- LEFT - Returns the left part of a character string with the specified number of characters
- RIGHT - Returns the right part of a character string with the specified number of characters

Examples

The following example replaces the string code in abcdefghi with xxx.

```
SELECT REPLACE('abcdefghicde','cde','xxx');
```

Here is the result set.

abxxxfghixxx

The following example converts an expression that is made up of five digits and a decimal point to a six-position character string. The fractional part of the number is rounded to one decimal place.

```
SELECT STR(123.45, 6, 1)
```

Here is the result set.

123.5 (1 row(s) affected)

The following example returns the five leftmost characters of each product name.

```
USE Northwind;
SELECT LEFT(ProductName, 5)
FROM Products
ORDER BY ProductID;
```

Instructor Notes:

Using System Functions – String Functions

- SUBSTRING - Returns part of a character, binary, text, or image expression
- LEN - Returns the number of characters of the specified string expression, excluding trailing blanks
- REVERSE - Returns the reverse of a character expression
- LOWER - Returns a character expression after converting uppercase character data to lowercase
- UPPER - Returns a character expression with lowercase character data converted to uppercase
- + -- used for concatenating strings

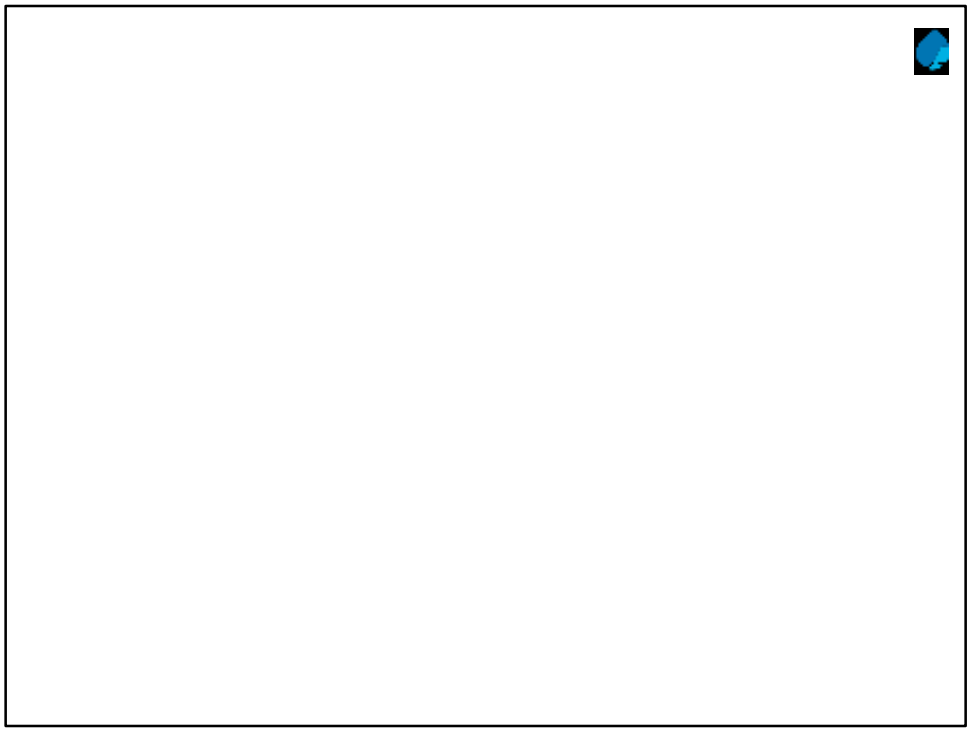
The following example returns the five rightmost characters of the first name for each contact

```
USE AdventureWorks
GO
SELECT RIGHT(FirstName, 5) AS 'First Name'
FROM Person.Contact
WHERE ContactID < 5
ORDER BY FirstName; GO
```

Here is the result set.

```
First Name
----------
erine
stavo
berto
Kim
```

Instructor Notes:



use Northwind;
Go

SELECT LastName, SUBSTRING(FirstName, 1, 1) AS Initial FROM Employees WHERE LastName like ‘S%'

SELECT LEN(CompanyName) AS Length,CompanyName,city
FROM Suppliers
WHERE Country ='USA‘

Some more string functions are:

ASCII, NCHAR, SOUNDEX, CHAR, PATINDEX, SPACE, CHARINDEX, QUOTENAME, DIFFERENCE, STUFF, REPLICATE, UNICODE, LTRIM, RTRIM.
All built-in string functions are deterministic. This means they return the same value any time they are called with a specific set of input values

Example of string concatenation
SELECT TitleOfCourtesy+’ ‘+LastName+’,’+FirstName
FROM northwind.dbo.Employees
ORDER BY TITLE
.

Instructor Notes:

Using System Functions – Date Functions

- GETDATE - Returns the current database system timestamp as a datetime value without the database time zone offset
- GETUTCDATE - Returns the current database system timestamp as a datetime value. The database time zone offset is not included
- CURRENT_TIMESTAMP - Returns the current database system timestamp as a datetime value without the database time zone offset
- SYSDATETIME - Returns a datetime2(7) value that contains the date and time of the computer on which the instance of SQL Server is running. The time zone offset is not included

Date Functions Examples

GETDATE

SELECT GETDATE()

Result - 2012-03-10 13:56:36.620

GETUTCDATE

SELECT GETUTCDATE()

Result - 2012-03-10 13:56:36.620

CURRENT_TIMESTAMP

SELECT CURRENT_TIMESTAMP

Result - 2012-03-10 13:56:36.620

SYSDATETIME

Result - 2012-04-10 14:04:46.50

Instructor Notes:

Using System Functions – Date Functions

- SYSDATETIMEOFFSET - Returns a datetimeoffset(7) value that contains the date and time of the computer on which the instance of SQL Server is running. The time zone offset is included.
- SYSUTCDATETIME - Returns a datetime2(7) value that contains the date and time of the computer on which the instance of SQL Server is running. The date and time is returned as UTC time (Coordinated Universal Time).

SYSDATETIMEOFFSET – DATETIMEOFFSET Defines a date that is combined with a time of a day that has time zone awareness and is based on a 24-hour clock

SELECT SYSDATETIMEOFFSET()

Result - 2012-03-10 14:06:56.7418989 +05:30

SYSUTCDATETIME

SELECT SYSUTCDATETIME()

Result - 2012-03-10 08:39:06.94

Instructor Notes:

Date Functions – To retrieve Date & Time Parts

- DATENAME - Returns a character string that represents the specified datepart of the specified date
- DATEPART - Returns an integer that represents the specified datepart of the specified date

datepart	Return value – DATEPART	Return Value - DATENAME
year, yyyy, yy	2007	2007
quarter, qq, q	4	4
month, mm, m	9	October
dayofyear, dy, y	303	303
day, dd, d	30	30
week, wk, ww	44	44
weekday, dw	3	Tuesday
hour, hh	12	12
minute, n	15	15
second, ss, s	32	32
millisecond, ms	123	123
microsecond, mcs	123456	123456

Examples

The following example extracts the month name from the date returned by GETDATE.

```
SELECT DATENAME(month, GETDATE()) AS 'Month Name';
```

Result

```
Month Name
------------------------------
February
```

The following example extracts the month name from a column.

```
USE AdventureWorks
GO
SELECT StartDate, DATENAME(month, StartDate) AS StartMonth FROM
Production.WorkOrder WHERE WorkOrderID = 1; GO
```

Result

```
StartDate StartMonth
------------------------------ ------------
2001-07-04 00:00:00.000 July
```

Try this :

```
SELECT DATENAME(YEAR,GETDATE())
SELECT DATEPART(YEAR,GETDATE())
SELECT DATENAME(MONTH,GETDATE())
SELECT DATEPART(MONTH,GETDATE())
```

Instructor Notes:

Using System Functions – Date Functions

- DATEDIFF - Returns the count (signed integer) of the specified datepart boundaries crossed between the specified startdate and enddate.
- DATEADD- Returns a specified date with the specified number interval (signed integer) added to a specified datepart of that date.

Examples

DATEDIFF

```
CREATE TABLE dbo.Duration
(
startDate datetime2 ,endDate datetime2
)
INSERT INTO dbo.Duration(startDate,endDate)
VALUES('2007-05-06 12:10:09','2007-05-07 12:10:09')

SELECT DATEDIFF(day,startDate,endDate) AS 'Duration' FROM dbo.Duration;
-- Returns: 1
```

DATEADD-

The following example adds 2 days to each OrderDate to calculate a new PromisedShipDate.

```
USE AdventureWorks;
GO
SELECT SalesOrderID
    ,OrderDate
    ,DATEADD(day,2,OrderDate) AS PromisedShipDate
FROM Sales.SalesOrderHeader;
```

Instructor Notes:

Using System Functions – Mathematical Functions

- ABS - A mathematical function that returns the absolute (positive) value of the specified numeric expression
- RAND - Returns a random float value from 0 through 1
- ROUND - Returns a numeric value, rounded to the specified length or precision
- SQRT - Returns the square root of the specified float value

The following scalar functions perform a calculation, usually based on input values that are provided as arguments, and return a numeric value:
ABS - A mathematical function that returns the absolute (positive) value of the specified numeric expression.
RAND - Returns a random **float** value from 0 through 1.
RAND **(** [*seed*] **)**
ROUND - Returns a numeric value, rounded to the specified length or precision.
ROUND **(** *numeric_expression* **,** *length* [**,***function*] **)**
SQRT - Returns the square root of the specified float value.
SQRT **(** *float_expression* **)**

Examples:
SELECT ABS(-1.0), ABS(0.0), ABS(1.0);
Here is the result set.
1.0 .0 1.0

SELECT ROUND(123.4545, 2); GO (Output: 123.4500)
SELECT ROUND(123.45, -2);GO (Output: 100.00)

Some more mathematical functions are:
DEGREES, ACOS, EXP, ASIN, FLOOR, SIGN, ATAN, LOG, SIN, ATN2, LOG10, CEILING, PI, SQUARE, COS, POWER, TAN, COT, RADIANS

Instructor Notes:

Using System Functions – Aggregate Functions

- The aggregate functions are: sum, avg, count, min, max, and count(*)
- Aggregate functions are used to calculate and summarize data

```
USE pubs
GO

SELECT AVG(price * 2)
FROM titles
GO

SELECT  MAX(price) as Maxprice,MIN(price) as Minprice
FROM  titles
GO
```

Using aggregate functions

The aggregate functions are: **sum, avg, count, min, max,** and **count(*)**. You can use aggregate functions to calculate and summarize data.

Examples:

```
use pubs;
select sum(ytd_sales)
from titles;
```

Note that there is no column heading for the aggregate column.

An aggregate function take as an argument the column name on whose values it will operate. You can apply aggregate functions to all the rows in a table, to a subset of the table specified by a **where** clause, or to one or more groups of rows in the table. From each set of rows to which an aggregate function is applied, Adaptive Server generates a single value.

Here is the syntax of the aggregate function:

aggregate_function ([all | distinct] *expression*)

Expression is usually a column name. However, it can also be a constant, a function, or any combination of column names, constants, and functions connected by arithmetic or bitwise operators. You can also use a **case** expression or subquery in an expression.

Instructor Notes:

System Functions – To retrieve System Information

- CURRENT_TIMESTAMP - Returns the current date and time, equivalent to GETDATE.
- CURRENT_USER - Returns the name of the current user, equivalent to USER_NAME().
- HOST_ID & HOST_NAME - Returns the workstation identification number and name.

The following functions perform operations on and return information about values, objects, and settings in SQL Server

CURRENT_TIMESTAMP - Returns the current date and time. This function is the ANSI SQL equivalent to GETDATE.

CURRENT_USER - Returns the name of the current user. This function is equivalent to USER_NAME().

HOST_ID - Returns the workstation identification number. The workstation identification number is the process ID (PID) of the application on the client computer that is connecting to SQL Server.

HOST_NAME - Returns the workstation name.

Instructor Notes:

System Functions – To retrieve System Information

- CAST and CONVERT - Explicitly converts an expression of one data type to another
- CAST (expression AS data_type [(length)])
- CONVERT (data_type [(length)] , expression [, style])

Example

```
Select CONVERT(char,100)     --converts 100 to '100'
Select CAST(100 as char)
```

CAST and CONVERT - Explicitly converts an expression of one data type to another. CAST and CONVERT provide similar functionality.
Syntax
Syntax for CAST: CAST **(** expression AS data_type [**(**length **)**]**)**
Syntax for CONVERT: CONVERT **(** data_type [**(** length **)**] **,** expression [**,** style] **)**

Instructor Notes:

Demo

- Working with commonly used Functions

Instructor Notes:

Organizing Query result into Groups

- Using group by clause
- Group by clause divides the output of a query into groups
- Can group by one or more column names

Example

lists no of employees in each region

```
SELECT  Region, count(EmployeeID)
FROM Northwind.dbo.Employees
GROUP by REGION
GO
```

Organizing query results into groups: the *group by* clause

The **group by** clause divides the output of a query into groups. You can group by one or more column names, or by the results of computed columns using numeric datatypes in an expression. When used with aggregates, **group by** retrieves the calculations in each subgroup, and may return multiple rows. The maximum number of columns or expressions you can use in a **group by** clause is 16.

You cannot **group by** columns of text or image datatypes.

While you can use **group by** without aggregates, such a construction has limited functionality and may produce confusing results. The following example groups the results by title type:

select type, advance from titles group by type,advance

type	advance
business	5,000.00
business	5,000.00
business	10,125.00
business	5,000.00
mod_cook	0.00
mod_cook	15,000.00
UNDECIDED	NULL
popular_comp	7,000.00
popular_comp	8,000.00
popular_comp	NULL

……

Instructor Notes:

Groups By

- SQL standards for group by are more restrictive
- SQL standard requires that:
  - Columns in a select list must be in the group by expression or they must be arguments of aggregate functions
  - A group by expression can only contain column names in the select list

```
USE pubs;
GO
select pub_id, type, avg(price), sum(ytd_sales)
from titles
group by pub_id, type
GO
```

Instructor Notes:

Using Aggregation with Groups

```
USE pubs;
GO
select type, sum(advance)
from titles
group by type;
GO
```

With an aggregate for the *advance* column, the query returns the sum for each group:

select type, sum(advance) from titles group by type

type	
UNDECIDED	NULL
business	25,125.00
mod_cook	15,000.00
popular_comp	15,000.00
psychology	21,275.00
trad_cook	19,000.00

The summary values in a **group by** clause using aggregates are called vector aggregates, as opposed to scalar aggregates, which result when only one row is returned.

Instructor Notes:

Selecting Group

- Use the having clause to display or reject rows defined by the group by clause

  Example

```
USE pubs;
GO
select type
from titles
group by type
having count(*) > 1
GO
```

```
USE pubs;
GO
select type
from titles
where count(*)> 1
GO
```

Selecting groups of data: the *having* clause

Use the **having** clause to display or reject rows defined by the **group by** clause. The **having** clause sets conditions for the **group by** clause in the same way **where** sets conditions for the **select** clause, except **where** cannot include aggregates, while **having** often does.

How the *having*, *group by*, and *where* clauses interact

When you include the **having**, **group by**, and **where** clauses in a query, the sequence in which each clause affects the rows determines the final results:

The **where** clause excludes rows that do not meet its search conditions.

The **group by** clause collects the remaining rows into one group for each unique value in the **group by** expression.

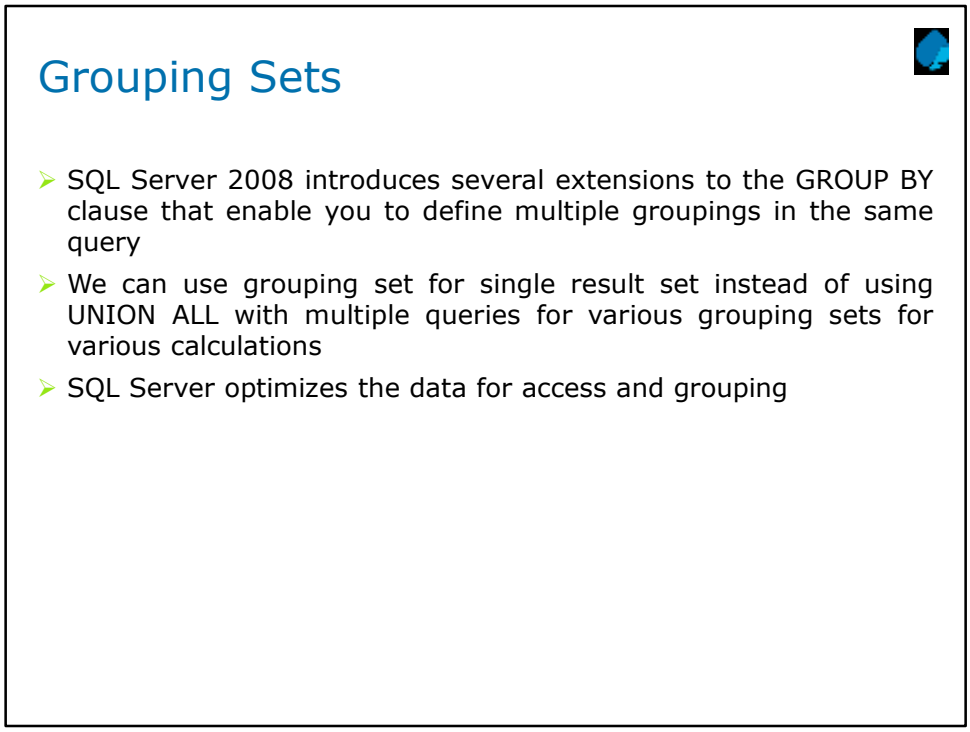
Aggregate functions specified in the select list calculate summary values for each group.

The **having** clause excludes rows from the final results that do not meet its search conditions.

The following query illustrates the use of **where**, **group by**, and **having** clauses in one **select** statement containing aggregates:

```
select stor_id, title_id, sum(qty)
from salesdetail
where title_id like 'PS%'
group by stor_id, title_id having sum(qty) > 200
```

Instructor Notes:



Grouping Sets

SQL Server 2008 introduces several extensions to the GROUP BY clause that enable you to define multiple groupings in the same query. We can use grouping set for single result set instead of using UNION ALL with multiple queries for various grouping sets for various calculations. SQL Server optimizes the data for access and grouping.

Without the extensions, a single query normally defines one “grouping set” (a set of attributes to group by) in the GROUP BY clause. If you want to calculate aggregates for multiple grouping sets, you usually need multiple queries. If you want to unify the result sets of multiple GROUP BY queries, each with a different grouping set, you must use the UNION ALL set operation between the queries.

You might need to calculate and store aggregates for various grouping sets in a table. By pre-processing and materializing the aggregates, you can support applications that require fast response time for aggregate requests. However, the aforementioned approach, in which you have a separate query for each grouping set, is very inefficient. This approach requires a separate scan of the data for each grouping set and expensive calculation of aggregates.

Instructor Notes:

Grouping Sets

- Example – Grouping Sets equivalent to UNION ALL

```
SELECT customer, NULL as year, SUM(sales) FROM T
GROUP BY customer
UNION ALL SELECT NULL as customer, year, SUM(sales) FROM T GROUP BY year
```

```
SELECT customer, year, SUM(sales) FROM T
GROUP BY GROUPING SETS ((customer), (year))
```

With the new GROUPING SETS subclause, you simply list all grouping sets that you need. Logically, you get the same result set as you would by unifying the result sets of multiple queries.
However, with the GROUPING SETS subclause, which requires much less code, SQL Server optimizes data access and the calculation of aggregates. SQL Server will not necessarily need to scan data once for each grouping set; plus, in some cases it calculates higher-level aggregates based on lower-level aggregates instead of re-aggregating base data.

Instructor Notes:

Demo

- Working with Group By

Instructor Notes:

Introduction

- SET operators are mainly used to combine the same type of data from two or more tables into a single result
- SET Operators supported in SQL Server are
  - UNION /UNION ALL
  - INTERSECT
  - EXCEPT

UNION
Combine two or more result sets into a single set, without duplicates.

UNION ALL
Combine two or more result sets into a single set, including all duplicates.

INTERSECT
Takes the data from both result sets which are in common.

EXCEPT
Takes the data from first result set, but not the second (i.e. no matching to each other)

Instructor Notes:

Explain the UNION and UNION ALL with a demo .

Make the students think how else they can get the same result (using IN)

Introduction (Contd…)

- UNION /UNION ALL
  - Combine two or more result sets into a single set, without duplicates.
  - The number of columns have to match
  - UNION ALL works exactly like UNION except that duplicates are NOT removed

```
SELECT ProductID,ProductName FROM Products
WHERE categoryID=1234
UNION
SELECT ProductID,ProductName FROM Products
WHERE categoryID=5678
GO
```

The above query will list all products belonging to category 1234 and 5678

The number of columns in the SELECT clauses must be same . In case the result has to be sorted on any order , the last query can only have Order by clause

Instructor Notes:

Explain the same with a demo . The instructor should have an understanding of what the queries do

Introduction (Contd…)

- INTERSECT
  - Takes the data from both result sets which are in common.
  - All the other conditions remain same

```
SELECT CustomerID FROM Customers
INTERSECT
SELECT CustomerID FROM Orders
```

- EXCEPT
  - Takes the data from first result set, which is not available in the second
  - It is like a complement operation

```
SELECT CustomerID FROM Customers
EXCEPT
SELECT CustomerID FROM Orders
```

Here in this example using an INTERSECT operation I can find out all Customers who have placed orders

Using an EXCEPT operation , I can find out all Customers who haven't placed any Orders

Instructor Notes:

Rules of Set Operation

- The result sets of all queries must have the same number of columns.
- In every result set the data type of each column must match the data type of its corresponding column in the first result set.
- In order to sort the result, an ORDER BY clause should be part of the last statement.
- The records from the top query must match the positional ordering of the records from the bottom query.
- The column names or aliases of the result set are given in the first select statement

Instructor Notes:

Summary

- Transact-SQL is central to using SQL Server
- SELECT is the basic & commonly used data retrieval statement used in SQL Server
- SQL Server provides variety of System functions which can help us in performing our day to day activities
- For example, String Functions, Date Functions, Mathematical functions, Aggregate Functions and so on
- We can make use of Group By to divide the SQL query result into groups
- We can use grouping set for single result set instead of using UNION ALL with multiple queries for various grouping sets for various calculations

Instructor Notes:

Review Question

- Question 1: _____ is central to using SQL Server
- Question 2: _____ returns the current database system timestamp as a datetime value without the database time zone offset
- Question 3: We can use _____ _____ for single result set instead of using UNION ALL with multiple queries

Instructor Notes:

Review Question

- Question 1: The Set operation that will show all the rows from both the resultsets including duplicates is _____
  - Option 1: Union All
  - Option 2: Union
  - Option 3: Intersect
  - Option 4: Minus

- Question 2: The Except operator returns _____