Logical Query Processing

- The order in which a query is written is not the order in which it is evaluated by SQL Server.

```
5: SELECT      <select list>
1: FROM        <table source>
2: WHERE       <search condition>
3: GROUP BY    <group by list>
4: HAVING      <search condition>
6: ORDER BY    <order by list>
```

Using the ORDER BY Clause

```
SELECT <select_list>
FROM   <table_source>
ORDER BY <order_by_list> ASC|DESC;
```

- ORDER BY sorts rows in results for presentation purposes
  - No guaranteed order of rows without ORDER BY
  - Use of ORDER BY guarantees the sort order of the result
  - Last clause to be logically processed

```
5: SELECT      <select list>
1: FROM        <table source>
2: WHERE       <search condition>
3: GROUP BY <group by list>
4: HAVING <search condition>
6: ORDER BY <order by list>
```

Using the ORDER BY Clause

```
SELECT <select_list>
FROM   <table_source>
ORDER BY <order_by_list> ASC|DESC;
```

- ORDER BY sorts rows in results for presentation purposes
  - No guaranteed order of rows without ORDER BY
  - Use of ORDER BY guarantees the sort order of the result
  - Last clause to be logically processed
  - Sorts all NULLs together

```
SELECT empid, lastname, region
FROM HR.Employees
ORDER BY region ;
```

	empid	lastname	region
1	5	Buck	NULL
2	6	Suurs	NULL
3	7	King	NULL
4	9	Dolgopyatova	NULL
5	8	Cameron	WA
6	1	Davis	WA
7	2	Funk	WA
8	3	Lew	WA
9	4	Peled	WA

Using the ORDER BY Clause

```
SELECT <select_list>
FROM   <table_source>
ORDER BY <order_by_list> ASC|DESC;
```

- ORDER BY sorts rows in results for presentation purposes
  - No guaranteed order of rows without ORDER BY
  - Use of ORDER BY guarantees the sort order of the result
  - Last clause to be logically processed
  - Sorts all NULLs together
- ORDER BY can refer to:
  - Columns by name, alias or ordinal position (not recommended)

ORDER BY Clause Examples

- ORDER BY with column names:

```
SELECT orderid, custid, orderdate
FROM Sales.Orders
ORDER BY orderdate;
```

- ORDER BY with column alias:

```
SELECT orderid, custid, YEAR(orderdate)
   AS orderyear
FROM Sales.Orders
ORDER BY orderyear;
```

- ORDER BY with descending order:

```
SELECT orderid, custid, orderdate
FROM Sales.Orders
ORDER BY orderdate DESC;
```

Using the ORDER BY Clause

```
SELECT <select_list>
FROM   <table_source>
ORDER BY <order_by_list> ASC|DESC;
```

- ORDER BY sorts rows in results for presentation purposes
  - No guaranteed order of rows without ORDER BY
  - Use of ORDER BY guarantees the sort order of the result
  - Last clause to be logically processed
  - Sorts all NULLs together

- ORDER BY can refer to:
  - Columns by name, alias or ordinal position (not recommended)

```
SELECT orderid, custid, orderdate
FROM Sales.Orders
ORDER BY 3 DESC;
```

```
SELECT orderid, custid, orderdate
FROM Sales.Orders
ORDER BY orderdate DESC;
```

Using the ORDER BY Clause

```
SELECT <select_list>
FROM   <table_source>
ORDER BY <order_by_list> ASC|DESC;
```

- ORDER BY sorts rows in results for presentation purposes
  - No guaranteed order of rows without ORDER BY
  - Use of ORDER BY guarantees the sort order of the result
  - Last clause to be logically processed
  - Sorts all NULLs together

- ORDER BY can refer to:
  - Columns by name, alias or ordinal position (not recommended)
  - Columns not part of SELECT list
    - Unless DISTINCT specified → Any columns in the ORDER BY list must be found in the SELECT list
- Declare sort order with ASC or DESC

```
SELECT orderid, custid, orderdate
FROM Sales.Orders
ORDER BY orderdate DESC;
```

ORDER BY Clause Examples

ORDER BY <order_by_expression> [ASC | DESC] [,...n]

```
SELECT orderid, custid,
    YEAR(orderdate) AS orderyear
FROM Sales.Orders
ORDER BY orderyear DESC;
```

orderid	custid	orderyear
11072	20	2008
11073	58	2008
11074	73	2008
11075	68	2008
11076	9	2008
11077	65	2008
10400	19	2007
10401	65	2007
10402	20	2007
10403	20	2007
10404	49	2007
10405	47	2007

```
SELECT orderid, custid,
    YEAR(orderdate)
FROM Sales.Orders
ORDER BY YEAR(orderdate) DESC;
```

orderid	custid	(No column name)
11072	20	2008
11073	58	2008
11074	73	2008
11075	68	2008
11076	9	2008
11077	65	2008
10400	19	2007
10401	65	2007
10402	20	2007
10403	20	2007
10404	49	2007
10405	47	2007

ORDER BY Clause Examples

ORDER BY <order_by_expression> [ASC | DESC] [,...n]

```
SELECT orderid, custid,
    YEAR(orderdate) AS orderyear
FROM Sales.Orders
ORDER BY orderyear DESC;
```

orderid	custid	orderyear
11072	20	2008
11073	58	2008
11074	73	2008
11075	68	2008
11076	9	2008
11077	65	2008
10400	19	2007
10401	65	2007
10402	20	2007
10403	20	2007
10404	49	2007
10405	47	2007

```
SELECT hiredate, firstname, lastname
FROM HR.Employees
ORDER BY hiredate DESC, lastname ASC;
```

	hiredate	firstname	lastname
1	2004-11-15 00:00:00.000	Zoya	Dolgopyatova
2	2004-03-05 00:00:00.000	Maria	Cameron
3	2004-01-02 00:00:00.000	Russell	King
4	2003-10-17 00:00:00.000	Sven	Buck
5	2003-10-17 00:00:00.000	Paul	Suurs
6	2003-05-03 00:00:00.000	Yael	Peled
7	2002-08-14 00:00:00.000	Don	Funk
8	2002-05-01 00:00:00.000	Sara	Davis
9	2002-04-01 00:00:00.000	Judy	Lew

Filtering Data with Predicates

- Filtering Data in the WHERE Clause with Predicates
- WHERE Clause Syntax

```
WHERE <column> <operator> <expression>
```

```
SELECT contactname, country
FROM Sales.Customers
WHERE country = 'Spain';
```

Comparison Operators	=, >, <, >=, <=, <>, !=, !>, !<

Filtering Data in the WHERE Clause With Predicates

- WHERE clauses use predicates
  - Must be expressed as logical conditions
  - Only rows for which predicate evaluates to TRUE are accepted
  - Values of FALSE or UNKNOWN filtered out

```
SELECT contactname, country
FROM Sales.Customers
WHERE country = 'Spain';
```

Filtering Data in the WHERE Clause With Predicates

- WHERE clauses use predicates
  - Must be expressed as logical conditions
  - Only rows for which predicate evaluates to TRUE are accepted
  - Values of FALSE or UNKNOWN filtered out
- WHERE clause follows FROM, precedes other clauses
  - Can’t see aliases declared in SELECT clause

Filtering Data in the WHERE Clause With Predicates

a simple calculated expression in the SELECT list,

```
SELECT orderid, custid, YEAR(orderdate) AS ordyear
FROM Sales.Orders
WHERE YEAR(orderdate) = 2006;
```

- WHERE clause follows FROM, precedes other clauses
  - Can't see aliases declared in SELECT clause

```
SELECT orderid, custid, YEAR(orderdate) AS ordyear
FROM Sales.Orders
WHERE ordyear = 2006;
```

```
Msg 207, Level 16, State 1, Line 3
Invalid column name 'ordyear'.
```

T-SQL Language Elements: Predicates and Operators

Elements:	Predicates and Operators:
Predicates	IN, BETWEEN, LIKE
Comparison Operators	=, >, <, >=, <=, <>, !=, !>, !<
Logical Operators	AND, OR, NOT
Arithmetic Operators	+, -, *, /, %
Concatenation	+

T-SQL enforces operator precedence

Order of Evaluation	Operators
1	( ) Parentheses
2	*, /, % (Multiply, Divide, Modulo)
3	+, - (Add/Positive/Concatenate, Subtract/Negative)
4	=, <, >, >=, <=, !=, !>, !< (Comparison)
5	NOT
6	AND
7	BETWEEN, IN, LIKE, OR
8	= (Assignment)

T-SQL enforces operator precedence

Predicates and Operators	Description
OR	Combines two Boolean expressions and returns TRUE if either expression is TRUE.

```
SELECT custid, companyname, country
FROM Sales.Customers
WHERE country = N'UK' OR country = N'Spain';
```

	custid	companyname	country
1	4	Customer HFBZG	UK
2	8	Customer QUHWH	Spain
3	11	Customer UBHAU	UK
4	16	Customer GYBBY	UK
5	19	Customer RFNQC	UK
6	22	Customer DTDMN	Spain
7	29	Customer MDLWA	Spain
8	30	Customer KSLQF	Spain
9	38	Customer LJUCA	UK
10	53	Customer GCJSG	UK
11	69	Customer SIUIH	Spain
12	72	Customer AHPOP	UK

Predicates and Operators	Description
IN	Determines whether a specified value matches any value in a subquery or a list.
OR	Combines two Boolean expressions and returns TRUE if either expression is TRUE.

```
SELECT custid, companyname, country
FROM Sales.Customers
WHERE country = N'UK' OR country = N'Spain';
```

The following example modifies the previous query to use the IN operator for the same results:

```
SELECT custid, companyname, country
FROM Sales.Customers
WHERE country IN (N'UK',N'Spain');
```

	custid	companyname	country
1	4	Customer HFBZG	UK
2	8	Customer QUHWH	Spain
3	11	Customer UBHAU	UK
4	16	Customer GYBBY	UK
5	19	Customer RFNQC	UK
6	22	Customer DTDMN	Spain
7	29	Customer MDLWA	Spain
8	30	Customer KSLQF	Spain
9	38	Customer LJUCA	UK
10	53	Customer GCJSG	UK
11	69	Customer SIUIH	Spain
12	72	Customer AHPOP	UK

Predicates and Operators	Description
IN	Determines whether a specified value matches any value in a subquery or a list.
OR	Combines two Boolean expressions and returns TRUE if either expression is TRUE.
NOT	Reverses the result of a search condition.

```
SELECT custid, companyname, country
FROM Sales.Customers
WHERE country = N'UK' OR country = N'Spain';
```

The following example modifies the previous query to use the IN operator for the same results:

```
SELECT custid, companyname, country
FROM Sales.Customers
WHERE country IN (N'UK',N'Spain');
```

The following example uses the NOT operator to reverse the previous condition:

```
SELECT custid, companyname, country
FROM Sales.Customers
WHERE country NOT IN (N'UK',N'Spain');
```

	custid	companyname	country
1	1	Customer NRZBB	Germany
2	2	Customer MLTDN	Mexico
3	3	Customer KBUDE	Mexico
4	5	Customer HGVLZ	Sweden
5	6	Customer XHXJV	Germany
6	7	Customer QXVLA	France
7	9	Customer RTXGC	France
8	10	Customer EEALV	Canada
9	12	Customer PSNMQ	Argentina
10	13	Customer VMLOG	Mexico

Predicates and Operators	Description
IN	Determines whether a specified value matches any value in a subquery or a list.
AND	Combines two Boolean expressions and returns TRUE only when both expressions are TRUE.
OR	Combines two Boolean expressions and returns TRUE if either expression is TRUE.
NOT	Reverses the result of a search condition.

```
SELECT orderid, custid, orderdate
FROM Sales.Orders
WHERE orderdate > '20061231' AND orderdate < '20080101';
```

	orderid	custid	orderdate
1	10400	19	2007-01-01 00:00:00.000
2	10401	65	2007-01-01 00:00:00.000
3	10402	20	2007-01-02 00:00:00.000
4	10403	20	2007-01-03 00:00:00.000
5	10404	49	2007-01-03 00:00:00.000
6	10405	47	2007-01-06 00:00:00.000
7	10406	62	2007-01-07 00:00:00.000
8	10407	56	2007-01-07 00:00:00.000
9	10408	23	2007-01-08 00:00:00.000
10	10409	54	2007-01-09 00:00:00.000
11	10410	10	2007-01-10 00:00:00.000
12	10411	10	2007-01-10 00:00:00.000
13	10412	87	2007-01-13 00:00:00.000
14	10413	41	2007-01-14 00:00:00.000

Predicates and Operators	Description
IN	Determines whether a specified value matches any value in a subquery or a list.
BETWEEN	Specifies an inclusive range to test.
AND	Combines two Boolean expressions and returns TRUE only when both expressions are TRUE.
OR	Combines two Boolean expressions and returns TRUE if either expression is TRUE.
NOT	Reverses the result of a search condition.

```
SELECT orderid, custid, orderdate
FROM Sales.Orders
WHERE orderdate > '20061231' AND orderdate < '20080101';
```

	orderid	custid	orderdate
1	10400	19	2007-01-01 00:00:00.000
2	10401	65	2007-01-01 00:00:00.000
3	10402	20	2007-01-02 00:00:00.000
4	10403	20	2007-01-03 00:00:00.000
5	10404	49	2007-01-03 00:00:00.000
6	10405	47	2007-01-06 00:00:00.000
7	10406	62	2007-01-07 00:00:00.000
8	10407	56	2007-01-07 00:00:00.000
9	10408	23	2007-01-08 00:00:00.000
10	10409	54	2007-01-09 00:00:00.000
11	10410	10	2007-01-10 00:00:00.000
12	10411	10	2007-01-10 00:00:00.000
13	10412	87	2007-01-13 00:00:00.000
14	10413	41	2007-01-14 00:00:00.000

	orderid	custid	orderdate
1	10399	83	2006-12-31 00:00:00.000
2	10400	19	2007-01-01 00:00:00.000
3	10401	65	2007-01-01 00:00:00.000
4	10402	20	2007-01-02 00:00:00.000
5	10403	20	2007-01-03 00:00:00.000
6	10404	49	2007-01-03 00:00:00.000
7	10405	47	2007-01-06 00:00:00.000
8	10406	62	2007-01-07 00:00:00.000
9	10407	56	2007-01-07 00:00:00.000
10	10408	23	2007-01-08 00:00:00.000
11	10409	54	2007-01-09 00:00:00.000
12	10410	10	2007-01-10 00:00:00.000
13	10411	10	2007-01-10 00:00:00.000
14	10412	87	2007-01-13 00:00:00.000

```
SELECT orderid, custid, orderdate
FROM Sales.Orders
WHERE orderdate BETWEEN '20061231' AND '20080101';
```

Predicates and Operators	Description
IN	Determines whether a specified value matches any value in a subquery or a list.
BETWEEN	Specifies an inclusive range to test.
LIKE	Determines whether a specific character string matches a specified pattern.
AND	Combines two Boolean expressions and returns TRUE only when both expressions are TRUE.
OR	Combines two Boolean expressions and returns TRUE if either expression is TRUE.
NOT	Reverses the result of a search condition.

```
SELECT
    custid, companyname, contactname
FROM Sales.Customers
WHERE
    contactname LIKE N'A%';
```

	custid	companyname	contactname
1	1	Customer NRZBB	Allen, Michael
2	4	Customer HFBZG	Arndt, Torsten

Filtering in the SELECT Clause Using the TOP Option

The simplified syntax of the TOP option is as follows:

```
SELECT TOP (N) <column_list>
FROM  <table_source>
WHERE <search_condition>;
```

to retrieve only the five most recent orders

```
SELECT TOP (5) orderid, custid, orderdate
FROM Sales.Orders
ORDER BY orderdate DESC;
```

The TOP operator depends on an ORDER BY clause to provide meaningful precedence

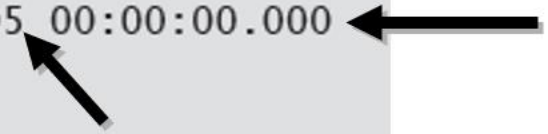
Filtering in the SELECT Clause Using the TOP Option

to retrieve only the five most recent orders

```
SELECT TOP (5) orderid, custid, orderdate
FROM Sales.Orders
ORDER BY orderdate DESC;
```

```
orderid     custid      orderdate
----------- ----------- -----------------------
11077       65          2008-05-06 00:00:00.000
11076       9           2008-05-06 00:00:00.000
11075       68          2008-05-06 00:00:00.000
11074       73          2008-05-06 00:00:00.000
11073       58          2008-05-05 00:00:00.000

(5 row(s) affected)
```



Filtering in the SELECT Clause Using the TOP Option

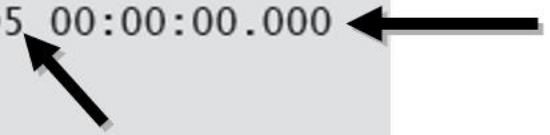
Whether to include WITH TIES will depend on your knowledge of the source data and its potential for unique values.

More rows qualify for the second-oldest order date.

```
SELECT TOP (5) WITH TIES orderid, custid, orderdate
FROM Sales.Orders
ORDER BY orderdate DESC;
```

```
orderid     custid      orderdate
----------- ----------- -----------------------
11077       65          2008-05-06 00:00:00.000
11076       9           2008-05-06 00:00:00.000
11075       68          2008-05-06 00:00:00.000
11074       73          2008-05-06 00:00:00.000
11073       58          2008-05-05 00:00:00.000

(5 row(s) affected)
```



Filtering in the SELECT Clause Using the TOP Option

Whether to include WITH TIES will depend on your knowledge of the source data and its potential for unique values.

More rows qualify for the second-oldest order date.

```
SELECT TOP (5) WITH TIES orderid, custid, orderdate
FROM Sales.Orders
ORDER BY orderdate DESC;
```

```
orderid     custid      orderdate
----------- ----------- -----------------------
11077       65          2008-05-06 00:00:00.000
11076       9           2008-05-06 00:00:00.000
11075       68          2008-05-06 00:00:00.000
11074       73          2008-05-06 00:00:00.000
11073       58          2008-05-05 00:00:00.000
11072       20          2008-05-05 00:00:00.000
11071       46          2008-05-05 00:00:00.000
11070       44          2008-05-05 00:00:00.000

(8 row(s) affected)
```

Filtering in the SELECT Clause Using the TOP Option

if the Sales.Orders table contains 830 orders, the following query will return 83 rows:

```
SELECT TOP (10) PERCENT orderid, custid, orderdate
FROM Sales.Orders
ORDER BY orderdate DESC;
```

TOP (N) PERCENT may also be used with the WITH TIES option.

TOP (N) PERCENT will round up to the nearest integer for purposes of row counts.

Filtering in the ORDER BY Clause Using OFFSET-FETCH

```
ORDER BY order_by_expression  [ ASC | DESC ]  [  , . . . n  ]
```

```
ORDER BY order_by_expression
                              [ ASC | DESC ]
                              [  , . . . n  ]
                              [  <offset_fetch>  ]
```

Filtering in the ORDER BY Clause Using OFFSET-FETCH

To return only a range of the rows (like TOP)

Extra: to supply a starting point (an offset) and a value to specify how many rows you would like to return (a fetch value).

- Retrieve first 50 rows only (alternative to TOP):

```
SELECT orderid, custid, empid, orderdate
FROM Sales.Orders
ORDER BY orderdate, orderid DESC
OFFSET 0 ROWS FETCH FIRST 50 ROWS ONLY;
```

OFFSET-FETCH Syntax

- Retrieve first 50 rows only (alternative to TOP):

```
SELECT orderid, custid, empid, orderdate
FROM Sales.Orders
ORDER BY orderdate, orderid DESC
OFFSET 0 ROWS FETCH FIRST 50 ROWS ONLY;
```

- Retrieve rows 51-100

```
SELECT orderid, custid, empid, orderdate
FROM Sales.Orders
ORDER BY orderdate, orderid DESC
OFFSET 50 ROWS FETCH NEXT 50 ROWS ONLY;
```

```
ORDER BY <order_by_list>
OFFSET <offset_value> ROW|ROWS FETCH FIRST|NEXT <fetch_value> ROW|ROWS ONLY
```

Filtering in the ORDER BY Clause Using OFFSET-FETCH

OFFSET-FETCH is an extension to the ORDER BY clause:

- Allows filtering a requested range of rows
  - Dependent on ORDER BY clause
- Provides a mechanism for paging through results
- Specify number of rows to skip, number of rows to retrieve:

```
ORDER BY <order_by_list>
OFFSET <offset_value> ROW(S)
FETCH FIRST|NEXT <fetch_value> ROW(S) ONLY
```

```
ORDER BY <order_by_list>
OFFSET <offset_value> ROW|ROWS FETCH FIRST|NEXT <fetch_value> ROW|ROWS ONLY
```

OFFSET-FETCH Syntax

- OFFSET value must be supplied
  - May be zero if no skipping is required
- FETCH clause is optional, allows all rows following OFFSET value to be returned
- Natural Language approach to code:
  - ROW and ROWS interchangeable
  - FIRST and NEXT interchangeable
- OFFSET value and FETCH value may be constants or expressions, including variables and parameters

```
OFFSET <offset_value> ROW|ROWS
FETCH FIRST|NEXT <fetch_value> ROW|ROWS ONLY
```

The syntax for the OFFSET-FETCH clause is as follows:

```
OFFSET { integer_constant | offset_row_count_expression } { ROW | ROWS }
    [FETCH { FIRST | NEXT } {integer_constant | fetch_row_count_expression } { ROW |
ROWS } ONLY]
```

Three-Valued Logic

- SQL Server uses NULLs to mark missing values
  - NULL can be "missing but applicable" or "missing but inapplicable"
    - Customer middle name: not supplied, or doesn’t have one?

Three-Valued Logic

- With no missing values, predicate outputs are TRUE or FALSE only (5 >2, 1=1)
- With missing values, outputs can be TRUE, FALSE or UNKNOWN (NULL > 99, NULL = NULL)
- Predicates return UNKNOWN when comparing missing value to another value, including another missing value

a NULL is neither TRUE nor FALSE; it is a mark for UNKNOWN, which represents the third value in three-valued logic (three possible outcomes).

Handling NULL in Queries

- Different components of SQL Server handle NULL differently
  - Query filters (ON, WHERE, HAVING) filter out UNKNOWNs
  - CHECK constraints accept UNKNOWNS

Treat NULL like a FALSE result

  - ORDER BY, DISTINCT treat NULLs as equals

```
SELECT empid, lastname, region
FROM HR.Employees
ORDER BY region ASC;
```

This returns the following
with all employees whose region
is missing (marked as NULL) sorted first:

empid	lastname	region
5	Buck	NULL
6	Suurs	NULL
7	King	NULL
9	Dolgopyatova	NULL
8	Cameron	WA
1	Davis	WA
2	Funk	WA
3	Lew	WA
4	Peled	WA

Handling NULL in Queries

- Different components of SQL Server handle NULL differently
  - Query filters (ON, WHERE, HAVING) filter out UNKNOWNs
  - CHECK constraints accept UNKNOWNS
  - ORDER BY, DISTINCT treat NULLs as equals

Treat NULL like a FALSE result

- Testing for NULL

```
SELECT custid, city, region, country
FROM Sales.Customers
WHERE region IS     NULL;
```

This returns correct results:

```
empid       lastname             region
----------- -------------------- ---------
5           Buck                 NULL
6           Suurs                NULL
7           King                 NULL
9           Dolgopyatova         NULL

(4 row(s) affected)
```

```
SELECT empid, lastname, region
FROM HR.Employees
WHERE region = NULL;
```

This returns inaccurate results:

```
empid       lastname             region
----------- -------------------- -------

(0 row(s) affected)
```