SET OPERATORS

Set operators are operators that combine rows from two query result sets (or multisets). Some of the operators remove duplicates from the result, and hence return a set, whereas others don't, and hence return a multiset. T-SQL supports the following operators: **UNION**, **UNION ALL, INTERSECT**, and **EXCEPT**.

A set operator compares complete rows between the results of the two input queries involved. Whether a row will be returned in the result of the operator depends on the outcome of the comparison and the operator used. Because a set operator expects multisets as inputs, the two queries involved cannot have ORDER BY clauses. Remember that a query with an ORDER BY clause does not return a multiset—it returns a cursor. However, although the queries involved cannot have ORDER BY clauses, you can optionally add an ORDER BY clause to the result of the operator. If you're wondering how you apply a set operator to queries with TOP and OFFSET-FETCH filters

In terms of logical-query processing, each of the individual queries can have all logical-query processing phases except for a presentation ORDER BY, as I just explained. The operator is applied to the results of the two queries, and the outer ORDER BY clause (if one exists) is applied to the result of the operator.

The two input queries must produce results with the same number of columns, and corresponding columns must have compatible data types. By compatible data types, I mean that the data type that is lower in terms of data-type precedence must be implicitly convertible to the higher data type. Of course, you also can explicitly convert the data type of a column in one query to the data type of the corresponding column in the other query using the CAST or CONVERT function.

UNION operator

The UNION operator unifies the results of two input queries. If a row appears in any of the input sets, it will appear in the result of the UNION operator. T-SQL supports both the UNION ALL and UNION (implicit DISTINCT) flavors of the UNION operator.

UNION ALL operator

The UNION ALL operator The UNION ALL operator unifies the two input query results without attempting to remove duplicates from the result. Assuming that Query1 returns m rows and Query2 returns n rows, Query1 UNION ALL Query2 returns m + n rows.

For example, the following code uses the UNION ALL operator to unify employee locations and customer locations:

```
SELECT country, region, city
       FROM HR.Employees UNION ALL
       SELECT country, region, city
       FROM Sales.Customers;
```

The result has 100 rows—9 from the Employees table and 91 from the Customers table

Because UNION ALL doesn't eliminate duplicates, the result is a **multiset** and not a set. The same row can appear multiple times in the result, as is the case with (UK, NULL, London) in the result of this query.

UNION (DISTINCT) operator

The UNION (implicit DISTINCT) operator unifies the results of the two queries and eliminates duplicates. Note that if a row appears in both input sets, it will appear only once in the result; in other words, **the result is a set** and not a multiset.

For example, the following code returns distinct locations(based on country, region and city) that are either employee locations or customer locations:

```
SELECT country, region, city
       FROM HR.Employees UNION
       SELECT country, region, city
       FROM Sales.Customers;
```

This code returns 71 distinct rows (unlike the 100 rows in the result with the duplicates), as shown here in abbreviated form:

INTERSECT Operator

The INTERSECT operator returns only the rows that are common to the results of the two input queries. Figure 6-2 illustrates this operator.

INTERSECT (DISTINCT) operator

The INTERSECT operator (implied DISTINCT) returns only distinct rows that appear in both input query results. As long as a row appears at least once in both query results, it's returned only once in the operator's result.

For example, the following code returns distinct locations that are both employee locations and customer locations:

```
SELECT country, region, city
       FROM HR.Employees INTERSECT
       SELECT country, region, city
       FROM Sales.Customers;
```

UK	NULL	London
USA	WA	Kirkland
USA	WA	Seattle

I mentioned earlier that when these operators compare rows, they use an implied distinct predicate, which returns a TRUE when comparing two NULLs. For example, observe that the location (UK, NULL, London) appears in the result of the intersection. If instead of using the INTERSECT operator you use an alternative tool like an inner join or a correlated subquery, you need to add special treatment for NULLs—for example, assuming the alias E for Employees and C for Customers, using the predicate E.region = C.region OR (E.region IS NULL AND C.region IS NULL). Using the INTERSECT operator, the solution is much simpler—you don't need to explicitly compare corresponding attributes, and you don't need to add special treatment for NULLs

INTERSECT ALL operator

Standard SQL supports an ALL flavor of the INTERSECT operator, but this flavor has not been implemented in T-SQL. However, you can write your own logical equivalent with T-SQL.

Remember the meaning of the ALL keyword in the UNION ALL operator: it returns all duplicate rows. Similarly**, the keyword ALL in the INTERSECT ALL operator means that duplicate intersections will not be removed**. INTERSECT ALL returns the number of duplicate rows matching the lower of the counts in both input multisets

EXCEPT operator: (A-B)

Returns everything that's in A that's not in B

The EXCEPT operator implements set differences. It operates on the results of two input queries and returns rows that appear in the first input but not the second. Figure 6-3 illustrates this operator.

EXCEPT (DISTINCT) operator

The EXCEPT operator (implied DISTINCT) returns only distinct rows that appear in the first set but not the second. In other words, a row is returned once in the output as long as it appears at least once in the first input multiset and zero times in the second

```
SELECT country, region, city
       FROM HR.Employees EXCEPT
       SELECT country, region, city
       FROM Sales.Customers;
```

	country	region	city
1	USA	WA	Redmond
2	USA	WA	Tacoma

EXCEPT ALL operator

The EXCEPT ALL operator is similar to the EXCEPT operator, but it also takes into account the number of occurrences of each row. If a row R appears x times in the first multiset and y times in the second, and x > y, R will appear x – y times in Query1 EXCEPT ALL Query2. In other words, EXCEPT ALL returns only occurrences of a row from the first multiset that do not have a corresponding occurrence in the second.

T-SQL does not provide a built-in EXCEPT ALL operator, but you can provide an alternative of your own

PRECEDENCE

SQL defines precedence among set operators.

1) INTERSECT operator precedes UNION and EXCEPT

2) UNION and EXCEPT are evaluated in order of appearance

Using the ALL variant doesn't change the precedence. In a query that contains multiple set operators, first INTERSECT operators are evaluated, and then operators with the same precedence are evaluated based on their order of appearance.

Consider the following code:

```
SELECT country, region, city FROM Production.Suppliers
EXCEPT
SELECT country, region, city FROM HR.Employees
INTERSECT
SELECT country, region, city FROM Sales.Customers;
```

Because INTERSECT precedes EXCEPT, the INTERSECT operator is evaluated first, even though it appears second in the code. The meaning of this query is, "locations that are supplier locations, but not (locations that are both employee and customer locations)."

To control the order of evaluation of set operators, use parentheses, because they have the highest precedence. Also, using parentheses increases the readability, thus reducing the chance for errors. For example, if you want to return "(locations that are supplier locations but not employee locations) and that are also customer locations," use the following code:

```
(SELECT country, region, city FROM Production.Suppliers
EXCEPT
SELECT country, region, city FROM HR.Employees)
INTERSECT
SELECT country, region, city FROM Sales.Customers;
```