Oracle Advanced concepts

Prepared By
Raghavendra

Working with Set Operators

Set Operators :

Set operators combine the results of two component queries into a single result. Queries containing set operators are called compound queries.

Operator	Returns
`UNION`	All distinct rows selected by either query
`UNION ALL`	All rows selected by either query, including all duplicates
`INTERSECT`	All distinct rows selected by both queries
`MINUS`	All distinct rows selected by the first query but not the second

➢You can combine multiple queries using the set operators UNION, UNION ALL, INTERSECT, and MINUS

➢All set operators have equal precedence

➢If a SQL statement contains multiple set operators, then Oracle Database evaluates them from the left to right unless parentheses explicitly specify another order

➢The corresponding expressions in the select lists of the component queries of a compound query must match in number and must be in the same data type group (such as numeric or character)

If component queries select character data, then the data type of the return values are determined as follows:

➢If both queries select values of data type CHAR of equal length, then the returned values have data type CHAR of that length
➢ If the queries select values of CHAR with different lengths, then the returned value is VARCHAR2 with the length of the larger CHAR value
➢If either or both of the queries select values of data type VARCHAR2, then the returned values have data type VARCHAR2

If component queries select numeric data, then the data type of the return values is determined by numeric precedence:

➢If any query selects values of type BINARY_DOUBLE, then the returned values have data type BINARY_DOUBLE.
➢If no query selects values of type BINARY_DOUBLE but any query selects values of type BINARY_FLOAT, then the returned values have data type BINARY_FLOAT.
➢If all queries select values of type NUMBER, then the returned values have data type NUMBER.

In queries using set operators, Oracle does not perform implicit conversion across data type groups. Therefore, if the corresponding expressions of component queries resolve to both character data and numeric data, Oracle returns an error.

Examples -The following query is valid:

```
SELECT 3 FROM DUAL
 INTERSECT
SELECT 3f FROM DUAL;
```

This is implicitly converted to the following compound query:

```
SELECT TO_BINARY_FLOAT(3) FROM DUAL
 INTERSECT
SELECT 3f FROM DUAL;
```

The following query returns an error:

```
SELECT '3' FROM DUAL
   INTERSECT
SELECT 3f FROM DUAL;
```

Restrictions on the Set Operators

The set operators are subject to the following restrictions:

- The set operators are not valid on columns of type BLOB, CLOB, BFILE, VARRAY, or nested table
- The UNION, INTERSECT, and MINUS operators are not valid on LONG columns
- If the select list preceding the set operator contains an expression, then you must provide a column alias for the expression in order to refer to it in the order_by_clause
- You cannot also specify the for_update_clause with the set operators
- You cannot specify the order_by_clause in the subquery of these operators
- You cannot use these operators in SELECT statements containing TABLE collection expressions

UNION Example :

The following statement combines the results of two queries with the UNION operator, which eliminates duplicate selected rows.

```
SQL> select deptno from emp
     union
     select deptno from dept;
```

DEPTNO
10
20
30
40

UNION ALL Example:

The UNION operator returns only distinct rows that appear in either result, while the UNION ALL operator returns all rows.
The UNION ALL operator does not eliminate duplicate selected rows:

```
SQL> select deptno from emp
     union all
     select deptno from dept;
```

DEPTNO
20
30
30
20
30
30
10
20
10
30
20
30
20
10
10
20
30
40

A deptno value that appears multiple times in either or both queries (such as '30') is returned only once by the UNION operator, but multiple times by the UNION ALL operator

INTERSECT Example :

The following statement combines the results with the INTERSECT operator, which returns only those rows returned by both queries:

```
select deptno from emp
intersect
select deptno from dept;
```

DEPTNO
10
20
30

MINUS Example :

The following statement combines results with the MINUS operator, which returns only unique rows returned by the first query but not by the second:

```
select deptno from dept
minus
select deptno from emp;
```

DEPTNO
40

ANY, SOME and ALL in Oracle

ANY, SOME and ALL – These are rarely used SQL comparison operators

ANY or SOME:

- Compares a value to each value in a list or returned by a query
- Must be preceded by =, !=, >, <, <=, >=
- Evaluates to FALSE if the query returns no rows

ALL:

- Compares a value to every value in a list or returned by a query
- Must be preceded by =, !=, >, <, <=, >=
- Evaluates to TRUE if the query returns no rows

Now some examples:

Select all employees:

SQL> **select ename, job, sal from emp;**

ENAME	JOB	SAL
SMITH	CLERK	800
ALLEN	SALESMAN	1,600
WARD	SALESMAN	1,250
JONES	MANAGER	2,975
MARTIN	SALESMAN	1,250
BLAKE	MANAGER	2,850
CLARK	MANAGER	2,450
SCOTT	ANALYST	3,000
KING	PRESIDENT	5,000
TURNER	SALESMAN	1,500
ADAMS	CLERK	1,100
JAMES	CLERK	950
FORD	ANALYST	3,000
MILLER	CLERK	1,300

Select all employees with a salary greater than 1600 or greater than 2999:

```
SQL> select ename, sal
     from emp
     where sal > any (1600, 2999);
```

ENAME	SAL
JONES	2975
BLAKE	2850
CLARK	2450
SCOTT	3000
KING	5000
FORD	3000

The optimizer expands a condition that uses the ANY or SOME comparison operator followed by a parenthesized list of values into an equivalent condition that uses equality comparison operators and OR logical operators
So, the following query returns the same result as the previous query with the ANY operator

```
SQL> select ename, sal
     from emp
     where sal > 1600 or sal > 2999;
```

ENAME	SAL
JONES	2975
BLAKE	2850
CLARK	2450
SCOTT	3000
KING	5000
FORD	3000

ANY and SOME are interchangeable
You can use either one and get the same result
Here is an example:

Select employees whose name starts with either A, W or J:

```
SQL> select ename from emp
     where substr(ename,1,1) = any ('A', 'W', 'J');
```

ENAME
ALLEN
WARD
JONES
ADAMS
JAMES

```
SQL> select ename from emp
     where substr(ename,1,1) = some ('A', 'W', 'J');
```

ENAME
ALLEN
WARD
JONES
ADAMS
JAMES

You can use a subquery instead of a parenthesized list of values after ANY or SOME:
Select employees whose salary is greater than any salesman's salary:

```
SQL> select ename
  from emp
  where sal > any (
  select sal
  from emp
  where job = 'SALESMAN');
```

ENAME
KING
FORD
SCOTT
JONES
BLAKE
CLARK
ALLEN
TURNER
MILLER

The optimizer transforms a condition that uses the ANY or SOME operator followed by a subquery into a condition containing the EXISTS operator and a correlated subquery

So, The above query is equivalent to the following (added table aliases for clarification):

```
SQL> select emp1.ename
  from emp emp1
  where exists (
  select emp2.sal
  from emp emp2
  where emp2.job = 'SALESMAN'
  and emp1.sal > emp2.sal);
```

ENAME
KING
FORD
SCOTT
JONES
BLAKE
CLARK
ALLEN
TURNER
MILLER

ALL

Select all employees with a salary greater than 1600 and greater than 2999 (This is not a logical query but it does show the usage of ALL):

```
SQL> select ename, sal
     from emp
     where sal > all (1600, 2999);
```

ENAME	SAL
SCOTT	3000
KING	5000
FORD	3000

The optimizer expands a condition that uses the ALL comparison operator followed by a parenthesized list of values into an equivalent condition that uses equality comparison operators and AND logical operators

So, the following query is equivalent to the previous one:

```
SQL> select ename, sal
     from emp
     where sal > 1600 and sal > 2999;
```

ENAME	SAL
SCOTT	3000
KING	5000
FORD	3000

You can use a subquery instead of a parenthesized list of values after ALL:

Select employees whose salary is greater than every salesman's salary:

```
SQL>  select ename
  from emp
  where sal > all (
  select sal
  from emp
  where job = 'SALESMAN');
```

ENAME
JONES
BLAKE
CLARK
SCOTT
KING
FORD

The optimizer transforms a condition that uses the ALL comparison operator followed by a subquery into an equivalent condition that uses the ANY comparison operator and a complementary comparison operator
So, the optimizer transforms the first condition (using ALL) into this one (using ANY):

```
SQL>  select emp1.ename
  from emp emp1
  where not (
  emp1.sal <= any (
  select emp2.sal
  from emp emp2
  where emp2.job = 'SALESMAN'));
```

ENAME
JONES
BLAKE
CLARK
SCOTT
KING
FORD

The optimizer then further transforms the second query into the following query using the rule for transforming conditions with the ANY comparison operator, followed by a correlated subquery:

```
SQL>  select emp1.ename
  from emp emp1
  where
  not exists (
  select emp2.sal
  from emp emp2
  where emp2.job = 'SALESMAN'
  and emp1.sal <= emp2.sal);
```

ENAME
JONES
BLAKE
CLARK
SCOTT
KING
FORD

The comparison operators ANY, SOME and ALL can be used when writing queries that answer specific questions; Whether they are the best option to use as far as performance is concerned has to be analyzed on a case by case basis