**Set Operators**

Set operators combine the results of two or more component queries into one result. Queries containing set operators are called *compound* *queries*.

All set operators have equal precedence. If a SQL statement contains multiple set operators, the Oracle server evaluates them from left (top) to right (bottom)—if no parentheses explicitly specify another order. You should use parentheses to specify the order of evaluation explicitly in queries that use the INTERSECT operator with other set operators.

**Set Operator Guidelines**

* + - The expressions in the SELECT lists of the queries must match in number and data type. Queries that use UNION, UNION ALL, INTERSECT, and MINUS operators in their WHERE clause must have the same number and data type of columns in their SELECT list. The data type of the columns in the SELECT list of the queries in the compound query may not be exactly the same. The column in the second query must be in the same data type group (such as numeric or character) as the corresponding column in the first query.
    - Set operators can be used in subqueries.
    - You should use parentheses to specify the order of evaluation in queries that use the INTERSECT operator with other set operators. This ensures compliance with emerging SQL standards that will give the INTERSECT operator greater precedence than the other set operators.

**Oracle Server and Set Operators**

When a query uses set operators, the Oracle server eliminates duplicate rows automatically except in the case of the UNION ALL operator. The column names in the output are decided by the column list in the first SELECT statement. By default, the output is sorted in ascending order of the first column of the SELECT clause.

The corresponding expressions in the SELECT lists of the component queries of a compound query must match in number and data type. If component queries select character data, the data type of the return values is determined as follows:

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

If component queries select numeric data, the data type of the return values is determined by numeric precedence. If all queries select values of the NUMBER type, the returned values have the NUMBER data type. In queries using set operators, the Oracle server 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, the Oracle server returns an error.

**UNION Operator**

The UNION operator returns all rows that are selected by either query. Use the UNION operator to return all rows from multiple tables and eliminate any duplicate rows.

**Guidelines**

* + - The number of columns being selected must be the same.
    - The data types of the columns being selected must be in the same data type group (such as numeric or character).
    - The names of the columns need not be identical.
    - UNION operates over all of the columns being selected.
    - NULL values are not ignored during duplicate checking.
    - By default, the output is sorted in ascending order of the columns of the SELECT clause.
* **Using the UNION Operator**

The UNION operator eliminates any duplicate records. If records that occur in both the EMPLOYEES and the JOB\_HISTORY tables are identical, the records are displayed only once. Observe in the output shown in the slide that the record for the employee with the EMPLOYEE\_ID 200 appears twice because the JOB\_ID is different in each row.

Consider the following example:

SELECT employee\_id, job\_id, department\_id

FROM employees

UNION

SELECT employee\_id, job\_id, department\_id

FROM job\_history;

* **Using the UNION Operator (continued)**

In the preceding output, employee 200 appears three times. Why? Note the DEPARTMENT\_ID values for employee 200. One row has a DEPARTMENT\_ID of 90, another 10, and the third 90. Because of these unique combinations of job IDs and department IDs, each row for employee 200 is unique and, therefore, not considered to be a duplicate. Observe that the output is sorted in ascending order of the first column of the SELECT clause (in this case, EMPLOYEE\_ID).

* **Using the UNION Operator (continued)**

In the preceding output, employee 200 appears three times. Why? Note the DEPARTMENT\_ID values for employee 200. One row has a DEPARTMENT\_ID of 90, another 10, and the third 90. Because of these unique combinations of job IDs and department IDs, each row for employee 200 is unique and, therefore, not considered to be a duplicate. Observe that the output is sorted in ascending order of the first column of the SELECT clause (in this case, EMPLOYEE\_ID).

* **UNION ALL Operator**

Use the UNION ALL operator to return all rows from multiple queries.

* **Guidelines**

The guidelines for UNION and UNION ALL are the same, with the following two exceptions that pertain to UNION ALL: Unlike UNION, duplicate rows are not eliminated and the output is not sorted by default.

**Using the UNION ALL Operator**

In the example, 30 rows are selected. The combination of the two tables totals to 30 rows. The UNION ALL operator does not eliminate duplicate rows. UNION returns all distinct rows selected by either query. UNION ALL returns all rows selected by either query, including all duplicates. Consider the query in the slide, now written with the UNION clause:

SELECT employee\_id, job\_id,department\_id  
 FROM employees  
 UNION  
 SELECT employee\_id, job\_id,department\_id  
 FROM job\_history  
 ORDER BY employee\_id;

The preceding query returns 29 rows. This is because it eliminates the following row (because it is a duplicate):

**INTERSECT Operator**

Use the INTERSECT operator to return all rows that are common to multiple queries.

**Guidelines**

* + - The number of columns and the data types of the columns being selected by the SELECT statements in the queries must be identical in all the SELECT statements used in the query. The names of the columns, however, need not be identical.
    - Reversing the order of the intersected tables does not alter the result.
    - INTERSECT does not ignore NULL values.
* **Using the INTERSECT Operator**
* In the example in this slide, the query returns only those records that have the same values in the selected columns in both tables.
* What will be the results if you add the DEPARTMENT\_ID column to the SELECT statement from the EMPLOYEES table and add the DEPARTMENT\_ID column to the SELECT statement from the JOB\_HISTORY table, and run this query? The results may be different because of the introduction of another column whose values may or may not be duplicates.
* **Example:**
* SELECT employee\_id, job\_id, department\_id
* FROM employees
* INTERSECT
* SELECT employee\_id, job\_id, department\_id
* FROM job\_history;
* Employee 200 is no longer part of the results because the EMPLOYEES.DEPARTMENT\_ID value is different from the JOB\_HISTORY.DEPARTMENT\_ID value.
* **MINUS Operator**
* Use the MINUS operator to return all distinct rows selected by the first query, but not present in the second query result set (the first SELECT statement MINUS the second SELECT statement).
* **Note:** The number of columns must be the same and the data types of the columns being selected by the SELECT statements in the queries must belong to the same data type group in all the SELECT statements used in the query. The names of the columns, however, need not be identical.
* **Using the MINUS Operator**
* In the example in the slide, the employee IDs in the JOB\_HISTORY table are subtracted from those in the EMPLOYEES table. The results set displays the employees remaining after the subtraction; they are represented by rows that exist in the EMPLOYEES table, but do not exist in the JOB\_HISTORY table. These are the records of the employees who have not changed their jobs even once.
* **Matching the SELECT Statements**
* Because the expressions in the SELECT lists of the queries must match in number, you can use the dummy columns and the data type conversion functions to comply with this rule. In the slide, the name, Warehouse location, is given as the dummy column heading. The TO\_CHAR function is used in the first query to match the VARCHAR2 data type of the state\_province column that is retrieved by the second query. Similarly, the TO\_CHAR function in the second query is used to match the VARCHAR2 data type of the department\_name column that is retrieved by the first query.
* The output of the query is shown:
* **Matching the SELECT Statement: Example**
* The EMPLOYEES and JOB\_HISTORY tables have several columns in common (for example, EMPLOYEE\_ID, JOB\_ID, and DEPARTMENT\_ID). But what if you want the query to display the employee ID, job ID, and salary using the UNION operator, knowing that the salary exists only in the EMPLOYEES table?
* The code example in the slide matches the EMPLOYEE\_ID and JOB\_ID columns in the EMPLOYEES and JOB\_HISTORY tables. A literal value of 0 is added to the JOB\_HISTORY SELECT statement to match the numeric SALARY column in the EMPLOYEES SELECT statement.
* In the results shown in the slide, each row in the output that corresponds to a record from the JOB\_HISTORY table contains a 0 in the SALARY column.
* **Using the ORDER BY Clause in Set Operations**
* The ORDER BY clause can be used only once in a compound query. If used, the ORDER BY clause must be placed at the end of the query. The ORDER BY clause accepts the column name or an alias. By default, the output is sorted in ascending order in the first column of the first SELECT query.
* **Note:** The ORDER BY clause does not recognize the column names of the second SELECT query. To avoid confusion over column names, it is a common practice to ORDER BY column positions.
* For example, in the following statement, the output will be shown in ascending order of job\_id.
* SELECT employee\_id, job\_id,salary
* FROM employees
* UNION
* SELECT employee\_id, job\_id,0
* FROM job\_history
* ORDER BY 2;
* If you omit ORDER BY, by default, the output will be sorted in ascending order of employee\_id. You cannot use the columns from the second query to sort the output.