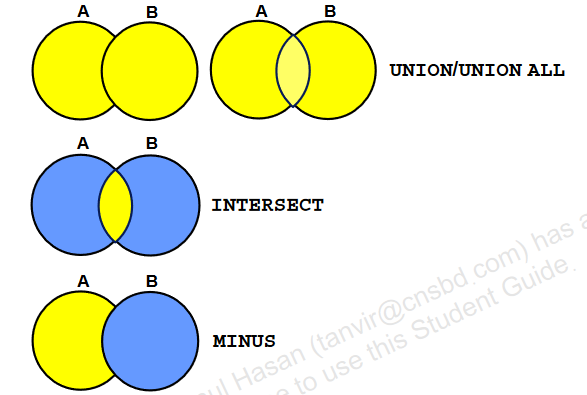
**Set Operators**

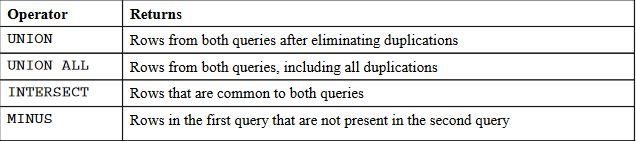
**What is set operator ?**

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

* The expressions in the SELECT lists must match in number.
* The data type of each column in the second query must match the data type of its corresponding column in the first query.
* Parentheses can be used to alter the sequence of execution.
* ORDER BY clause can appear only at the very end of the statement.

**Type of set operator :**



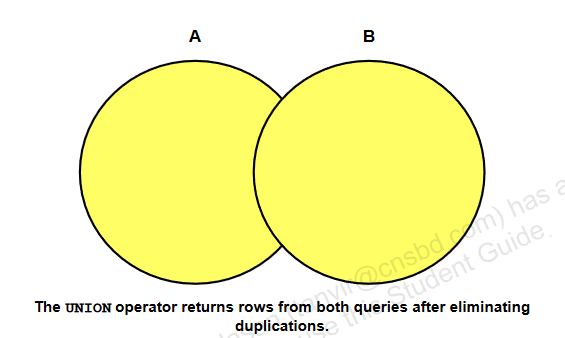


**UNION Operator**

**What is 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.

* 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(একই).
* 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.



**For Example:**

A = {a, b, c, d}

B = {x, y, z, a, c}

Now

C = A UNION B

C = {a, b, c, d} UNION {x, y, z, a, c}

= {a, b, c, d, x, y, z}

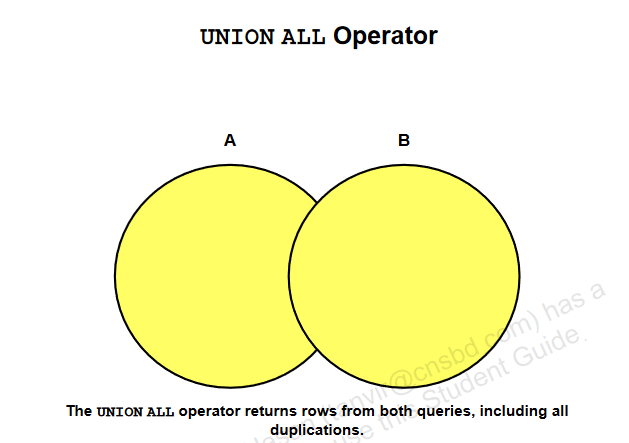
|  |
| --- |
| For Example:  **SELECT employee\_id, job\_id**  **FROM employees;**    **SELECT employee\_id, job\_id**  **FROM job\_history;**      **SELECT employee\_id, job\_id**  **FROM employees**  **UNION**  **SELECT employee\_id, job\_id**  **FROM job\_history;**    Another example:  **SELECT employee\_id, department\_id, job\_id**  **FROM employees;**    **SELECT employee\_id, department\_id, job\_id**  **FROM job\_history;**      **SELECT employee\_id, job\_id,department\_id**  **FROM employees**  **UNION**  **SELECT employee\_id, job\_id,department\_id**  **FROM job\_history;** |

|  |
| --- |
| **The HR department needs a report with the following specifications:**   * **Last name and department ID of all the employees from the EMPLOYEEStable, regardless of whether or not they belong to a department** * **Department ID and department name of all the departments from the DEPARTMENTS table, regardless of whether or not they have employees working in them.**   **Write a compound query to accomplish this.**  **SELECT last\_name,department\_id,TO\_CHAR(null)**  **FROM employees**  **UNION**  **SELECT TO\_CHAR(null),department\_id,department\_name**  **FROM departments** |

**UNION ALL Operator**

**What is UNION all operator?**

Use the UNION ALL operator to return all rows from multiple queries. The Oracle UNION ALL operator is used to combine the result sets of 2 or more SELECT statements. It returns all rows from the query and it does not remove duplicate rows between the various SELECT statements.



**For Example:**

A = {a, b, c, d}

B = {x, y, z, a, c}

Now

C = A UNION ALL B

C = {a, b, c, d} UNION ALL {x, y, z, a, c}

= {a, b, c, d, x, y, z, a, c}

|  |
| --- |
| For Example:  **SELECT employee\_id, department\_id, job\_id**  **FROM employees;**    **SELECT employee\_id, department\_id, job\_id**  **FROM job\_history;**      **SELECT employee\_id, job\_id, department\_id**  **FROM employees**  **UNION ALL**  **SELECT employee\_id, job\_id, department\_id**  **FROM job\_history**  **ORDER BY employee\_id;** |

|  |
| --- |
| **Produce a list of jobs for departments 10, 50, and 20, in that order. Display job ID and**  **department ID using the set operators.**  **SELECT distinct job\_id, department\_id**  **FROM employees**  **WHERE department\_id = 10**  **UNION ALL**  **SELECT DISTINCT job\_id, department\_id**  **FROM employees**  **WHERE department\_id = 50**  **UNION ALL**  **SELECT DISTINCT job\_id, department\_id**  **FROM employees**  **WHERE department\_id = 20 ;** |

**INTERSECT Operator**

**What is Intersect operator ?**

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

* 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.

**For Example:**

A = {a, b, c, d}

B = {x, y, z, a, c}

Now

C = A INTERSECT B

C = {a, b, c, d} INTERSECT {x, y, z, a, c}

= {a, c }

|  |
| --- |
| For Example:  **SELECT employee\_id, job\_id**  **FROM employees;**    **SELECT employee\_id, job\_id**  **FROM job\_history;**      **SELECT employee\_id, job\_id**  **FROM employees**  **INTERSECT**  **SELECT employee\_id, job\_id**  **FROM job\_history;**    Another example:  **SELECT employee\_id, department\_id, job\_id**  **FROM employees;**    **SELECT employee\_id, department\_id, job\_id**  **FROM job\_history;**      **SELECT employee\_id, job\_id, department\_id**  **FROM employees**  **INTERSECT**  **SELECT employee\_id, job\_id, department\_id**  **FROM job\_history;** |

|  |
| --- |
| **Create a report that lists the employee IDs and job IDs of those employees who**  **currently have a job title that is the same as their job title when they were initially**  **hired by the company (that is, they changed jobs, but have now gone back to doing**  **their original job).**  **SELECT employee\_id,job\_id**  **FROM employees**  **INTERSECT**  **SELECT employee\_id,job\_id**  **FROM job\_history;** |

**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.

**For Example:**

A = {a, b, c, d}

B = {x, y, z, a, c}

Now

C = A MINUS B

C = {a, b, c, d} MINUS {x, y, z, a, c}

= {b,d }

|  |
| --- |
| **Display the employee IDs of those employees who have not**  **changed their jobs even once.**      **SELECT employee\_id**  **FROM employees;**    **SELECT employee\_id**  **FROM job\_history;**    ANS :  **SELECT employee\_id**  **FROM employees**  **MINUS**  **SELECT employee\_id**  **FROM job\_history;** |

|  |
| --- |
| * **The HR department needs a list of department IDs for departments that do not contain the job ID ST\_CLERK. Use the set operators to create this report.**   **SELECT department\_id**  **FROM departments**  **MINUS**  **SELECT department\_id**  **FROM employees**  **WHERE job\_id = 'ST\_CLERK';**    **The HR department needs a list of countries that have no departments located in**  **them. Display the country ID and the name of the countries. Use the set operators to**  **create this report**  **SELECT country\_id,country\_name**  **FROM countries**  **MINUS**  **SELECT l.country\_id,c.country\_name**  **FROM locations l JOIN countries c**  **ON (l.country\_id = c.country\_id)**  **JOIN departments d**  **ON d.location\_id=l.location\_id;** |

**Matching the SELECT Statement**

|  |
| --- |
| **SELECT location\_id, department\_name "Department",**  **TO\_CHAR(NULL) "Warehouse location"**  **FROM departments;**    **SELECT location\_id, TO\_CHAR(NULL) "Department",**  **state\_province**  **FROM locations;**    **SELECT location\_id, department\_name "Department",**  **TO\_CHAR(NULL) "Warehouse location"**  **FROM departments**  **UNION**  **SELECT location\_id, TO\_CHAR(NULL) "Department",**  **state\_province**  **FROM locations;**    Another example  **SELECT employee\_id, job\_id,salary**  **FROM employees**  **UNION**  **SELECT employee\_id, job\_id,0**  **FROM job\_history;** |

**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.**

* The ORDER BY clause can appear only once at the end of the compound query.
* Component queries cannot have individual ORDER BY clauses.
* The ORDER BY clause recognizes only the columns of the first SELECT query.
* By default, the first column of the first SELECT query is used to sort the output in an ascending order.

|  |
| --- |
| **SELECT employee\_id, job\_id,salary**  **FROM employees**  **UNION**  **SELECT employee\_id, job\_id,0**  **FROM job\_history**  **ORDER BY 2;** |