SQL JOINS:

To join two tables we need at least one common column.
We can use the USING or ON clause to join 2 tables.

General Syntax:

```
SELECT columns FROM table1 JOIN table2 ON (table1.column_name =
table2.column_name);
```

NATURAL JOIN: [USING keyword] [EQUI-JOIN LIKE INNER JOIN]

Automatically join 2 tables with common columns that have the same name and same data type.
If columns have different data types with common columns names, then NATURAL JOIN will return an error.

```
SELECT * FROM employees;
SELECT * from departments;

SELECT * FROM employees NATURAL JOIN departments;
```

If there are more than one common column we can use the USING clause to specify which column to use in the join operation.
Joining with USING clause is also called as EQUI-JOIN.

```
SELECT first_name, last_name, department_id, employee_id
FROM employees JOIN departments USING (department_id);
```

AMBIGUOUS COLUMNS: [Solution use table aliases when joining two tables]

After joining two tables using a certain column or columns, there might be other common columns on which join is not used, so both the columns are included in the joined table. The server does not know which column to select since both are same and results in an error.
These are called ambiguous columns.

For example:

```
SELECT first_name, last_name, department_id, manager_id FROM
employees JOIN departments USING (department_id);
```

The above will return an error because SQL developer doesn't know which manager_id column to return since it exists in both tables.
We can use table aliases to handle this situation.

[Not so important] We cannot give aliases to columns we use in the USING clause or NATURAL JOIN.

```
SELECT first_name, last_name, department_name, employee_id,
e.manager_id AS emp_man_id, d.manager_id AS dep_man_id
FROM employees e JOIN departments d
USING (department_id);
```

INNER JOINS:

Return all rows that satisfies the join condition or the expression of ON/USING clause.

Differences between USING and ON clause:

If the column names are different in joining tables but they contain the same data, we can use the ON clause. USING clause expects the joining column to be exactly the same name.

We can use aliases in the ON clause UNLIKE the USING and NATURAL JOIN

```
SELECT e.first_name, e.last_name, d.manager_id, e.manager_id,
d.department_name
FROM employees e JOIN departments d
ON (e.department_id = d.department_id AND e.manager_id =
d.manager_id);
```

(OR)

For ON clause column names doesn't need to be same just the data type and the any column from the source table can be combined with the other table, as below e.employee_id is compared with d.manager_id.

```
SELECT e.first_name, e.last_name, d.manager_id, e.manager_id,
d.department_name
```

```
FROM employees e JOIN departments d
ON (e.department_id = d.department_id AND e.employee_id =
d.manager_id);
```

MULTIPLE JOIN OPERATIONS:

We can join multiple tables depending on our needs, with the ON, USING and NATURAL JOIN to join multiple tables.

```
SELECT first_name, last_name, d.department_name, l.city,
l.postal_code, l.street_address
FROM employees e
JOIN departments d
ON (e.department_id = d.department_id)
JOIN locations l
ON (d.location_id = l.location_id);
```

RESTRICTING JOINS USING WHERE CLAUSE OR AND OPERATOR:

We can restrict rows using the WHERE clause or the AND operator
Restricting with the AND clause is only specific to the JOIN operation.

```
SELECT first_name, last_name, d.department_name, l.city,
l.postal_code, l.street_address
FROM employees e
JOIN departments d
ON (e.department_id = d.department_id)
JOIN locations l
ON (d.location_id = l.location_id)
WHERE d.department_id = 100;

SELECT first_name, last_name, d.department_name, l.city,
l.postal_code, l.street_address
FROM employees e
JOIN departments d
ON (e.department_id = d.department_id)
JOIN locations l
ON (d.location_id = l.location_id)
AND e.job_id = 'IT_PROG'
AND e.first_name = ‘David';
```

SELF JOIN:

Joining a table with itself is called as a self join.
Self joins can be used when we want to query hierarchical data or compare data in the same table.

```
SELECT worker.first_name, worker.last_name, worker.employee_id,
worker.manager_id,
manager.employee_id, manager.first_name, manager.last_name
FROM employees worker JOIN employees manager
ON(worker.manager_id = manager.employee_id);
```

NON-EQUI JOINS:

If two tables do not have common columns, we can use the non equality operators or BETWEEN AND operator to join the tables.

Some applications of a NON-EQUI JOINS is finding duplicates in a table. Advanced use case to compute the running total of a particular column. Although this is not an efficient method.

```
SELECT e.employee_id, e.first_name, e.last_name, e.job_id, e.salary,
j.min_salary, j.max_salary, j.job_id
FROM employees e JOIN jobs j
ON e.salary > j.max_salary
AND j.job_id = 'SA_REP';
```

OUTER JOIN:

Returns all the rows from at least one of the table depending on left, right, full even if there is a match or not. All the unmatched rows are returned a NULL.

There are three types of outer joins,

1) LEFT OUTER

Returns all the matched rows and all the unmatched rows from the left table. For unmatched rows the values of the other table are NULLS.

```
SELECT e.first_name, e.last_name, d.department_id, d.department_name
FROM employees e LEFT OUTER JOIN departments d
ON(e.department_id = d.department_id);
```

2) RIGHT OUTER

Returns the matched rows and unmatched rows of the right table.

3) FULL OUTER

Returns the matched rows and unmatched rows of both tables.