* What is One-to-One, One-to-Many and Many-to-Many relationship in sql?

**One-to-One:**

A relationship is one-to-one if and only if one record from table A is related to a maximum of one record in table B.

To establish a one-to-one relationship, the primary key of table B (with no orphan record) must be the secondary key of table A (with orphan records).

For example:

1. **CREATE** **TABLE** Gov(
2. GID number(6) **PRIMARY** **KEY**,
3. **Name** varchar2(25),
4. Address varchar2(30),
5. );
6. **CREATE** **TABLE** State(
7. SID number(3) **PRIMARY** **KEY**,
8. StateName varchar2(15),
9. SGID Number(4) **REFERENCES** Gov(GID),
10. **CONSTRAINT** GOV\_SDID **UNIQUE** (SGID)
11. );
12. **INSERT** **INTO** gov(GID, **Name**, Address)  **values**(110, 'Bob');
14. **INSERT** **INTO** STATE **values**(111, 'Virginia', 2000000);

**One-to-many (1:M)**

A relationship is one-to-many if and only if one record from table A is related to one or more records in table B. However, one record in table B cannot be related to more one record in table A.

To establish a one-to-many relationship, the primary key of table A (the "one" table) must be the secondary key of table B (the "many" table).

For example:

1. **CREATE** **TABLE** Vendor(
2. VendorNumber number(4) **PRIMARY** **KEY**,
3. **Name** varchar2(20),
4. Address varchar2(20),
5. City varchar2(15),
6. Street varchar2(2),
7. ZipCode varchar2(10),
8. Contact varchar2(16),
9. PhoneNumber varchar2(12),
10. Status varchar2(8),
11. StampDate **date**
12. );
13. **CREATE** **TABLE** Inventory(
14. Item varchar2(6) **PRIMARY** **KEY**,
15. Description varchar2(30),
16. CurrentQuantity number(4) NOT NULL,
17. VendorNumber number(2) **REFERENCES** Vendor(VendorNumber),
18. ReorderQuantity number(3) NOT NULL );

**Many-to-many (M:M)**

A relationship is many-to-many if and only if one record from table A is related to one or more records in table B and vice-versa.

To establish a many-to-many relationship, create a third table called "ClassStudentRelation" which will have the primary keys of both table A and table B.

1. **CREATE** **TABLE** Class(
2. ClassID varchar2(10) **PRIMARY** **KEY**,
3. Title varchar2(30),
4. Instructor varchar2(30),
5. Day varchar2(15),
6. **Time** varchar2(10)
7. );
8. **CREATE** **TABLE** Student(
9. StudentID varchar2(15) **PRIMARY** **KEY**,
10. **Name** varchar2(35),
11. Major varchar2(35),
12. ClassYear varchar2(10),
13. Status varchar2(10)
14. );
15. **CREATE** **TABLE** ClassStudentRelation(
16. StudentID varchar2(15) NOT NULL,
17. ClassID varchar2(14) NOT NULL,
18. **FOREIGN** **KEY** (StudentID) **REFERENCES** Student(StudentID),
19. **FOREIGN** **KEY** (ClassID) **REFERENCES** Class(ClassID),
20. **UNIQUE** (StudentID, ClassID)
21. );

* What are different types of joins in SQL?

A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are:

* INNER JOIN
* LEFT JOIN
* RIGHT JOIN
* FULL JOIN

Consider the two tables below:



Student Table



Student course Table

**JOIN/INNER JOIN**: The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be same.

1. **SELECT** table1.column1,table1.column2,table2.column1,....
2. **FROM** table1
3. **INNER** JOIN table2
4. **ON** table1.matching\_column = table2.matching\_column;

7. table1: **First** **table**.
8. table2: **Second** **table**
9. matching\_column: **Column** common **to** both the tables.

**Note**: We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.

1. **SELECT** StudentCourse.COURSE\_ID, Student.**NAME**, Student.AGE **FROM** Student
2. **INNER** JOIN StudentCourse
3. **ON** Student.ROLL\_NO = StudentCourse.ROLL\_NO;



Output:



**LEFT JOIN/ LEFT OUTER JOIN**: This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on right side, the result-set will contain null.

Syntax:

1. **SELECT** table1.column1,table1.column2,table2.column1,....
2. **FROM** table1
3. LEFT JOIN table2
4. **ON** table1.matching\_column = table2.matching\_column;

7. table1: **First** **table**.
8. table2: **Second** **table**
9. matching\_column: **Column** common **to** both the tables.



1. **SELECT** Student.**NAME**,StudentCourse.COURSE\_ID
2. **FROM** Student
3. LEFT JOIN StudentCourse
4. **ON** StudentCourse.ROLL\_NO = Student.ROLL\_NO;

Output:

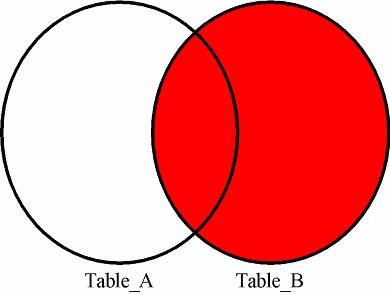


**RIGHT JOIN/RIGHT OUTER JOIN**: RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN

Syntax:

1. **SELECT** table1.column1,table1.column2,table2.column1,....
2. **FROM** table1
3. RIGHT JOIN table2
4. **ON** table1.matching\_column = table2.matching\_column;

7. table1: **First** **table**.
8. table2: **Second** **table**
9. matching\_column: **Column** common **to** both the tables.



1. **SELECT** Student.**NAME**,StudentCourse.COURSE\_ID
2. **FROM** Student
3. RIGHT JOIN StudentCourse
4. **ON** StudentCourse.ROLL\_NO = Student.ROLL\_NO;

Output:



**4. FULL JOIN:** FULL JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain *NULL* values.

1. **SELECT** table1.column1,table1.column2,table2.column1,....
2. **FROM** table1
3. **FULL** JOIN table2
4. **ON** table1.matching\_column = table2.matching\_column;

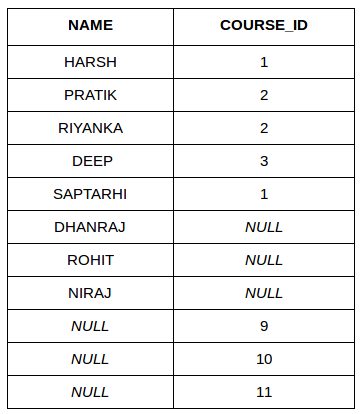
7. table1: **First** **table**.
8. table2: **Second** **table**
9. matching\_column: **Column** common **to** both the tables.



Example query:

1. **SELECT** Student.**NAME**,StudentCourse.COURSE\_ID
2. **FROM** Student
3. **FULL** JOIN StudentCourse
4. **ON** StudentCourse.ROLL\_NO = Student.ROLL\_NO;

Output:



* Find the 2nd Highest salary of Employee from employee table.

Consider below simple table:

1. Name     Salary
2. ---------------
3. abc     100000
4. bcd     1000000
5. efg     40000
6. ghi     500000

For example, in above table, “ghi” has the second highest salary as 500000.

**SELECT** **name**, **MAX**(salary) **as** salary **FROM** employee

We can nest the above query to find the second largest salary.

1. **SELECT** **name**, **MAX**(salary) **AS** salary  **FROM** employee
2. **WHERE** salary < (**SELECT** **MAX**(salary) **FROM** employee);

There are other ways also as

Method 1:

1. **SELECT** **name**, **MAX**(salary) **AS** salary **FROM** employee
2. **WHERE** salary IN (**SELECT** salary **FROM** employee MINUS **SELECT** **MAX**(salary) **FROM** employee);

Method 2:

1. **SELECT** **name**, **MAX**(salary) **AS** salary **FROM** employee
2. **WHERE** salary  (**SELECT** **MAX**(salary) **FROM** employee);

**How to find the third largest salary?**

Simple, we can do one more nesting.

1. **SELECT** **name**, **MAX**(salary) **AS** salary **FROM** employee
2. **WHERE** salary < (**SELECT** **MAX**(salary) **FROM** employee
3. **WHERE** salary < (**SELECT** **MAX**(salary) **FROM** employee));

we can find nth salary using general query like in MySQL:

Method 1:

1. **SELECT** salary **FROM** employee
2. **ORDER** **BY** salary **desc** limit n-1,1

Method 2:

1. **SELECT** **name**, salary **FROM** employee A
2. **WHERE** n-1 = (**SELECT** count(1) **FROM** employee B
3. **WHERE** B.salary>A.salary)