Types of Join in SQL Server

What is join??

An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.

There are many types of join.

* Inner Join

Equi-join

Natural Join

* Outer Join

Left outer Join

Right outer join

Full outer join

* Cross Join
* Self Join

Using the Code

Join is very useful to fetching records from multiple tables with reference to common column between them.

To understand join with example, we have to create two tables in SQL Server database.

1. Employee

Hide   Copy Code

create table Employee(

id int identity(1,1) primary key,

Username varchar(50),

FirstName varchar(50),

LastName varchar(50),

DepartID int

)

1. Departments

Hide   Copy Code

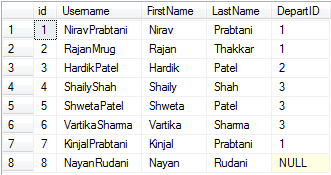
create table Departments(

id int identity(1,1) primary key,

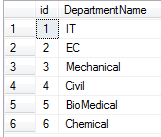
DepartmentName varchar(50)

)

Now fill Employee table with demo records like that.



Fill Department table also like this....



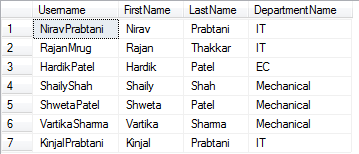
**1) Inner Join**

The join that displays only the rows that have a match in both the joined tables is known as inner join.

Hide   Copy Code

select e1.Username,e1.FirstName,e1.LastName,e2.DepartmentName \_<br />from Employee e1 inner join Departments e2 on e1.DepartID=e2.id

It gives matched rows from both tables with reference to DepartID of first table and id of second table like this.



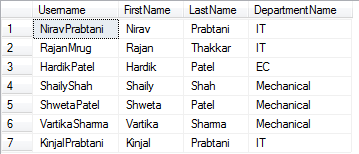
**Equi-Join**

Equi join is a special type of join in which we use only equality operator. Hence, when you make a query for join using equality operator, then that join query comes under Equi join.   
Equi join has only (=) operator in join condition.  
Equi join can be inner join, left outer join, right outer join.

Check the query for equi-join:

Hide   Copy Code

SELECT \* FROM Employee e1 JOIN Departments e2 ON e1.DepartID = e2.id



**2) Outer Join**

Outer join returns all the rows of both tables whether it has matched or not.

We have three types of outer join:

1. Left outer join
2. Right outer join
3. Full outer join

**a) Left Outer join**

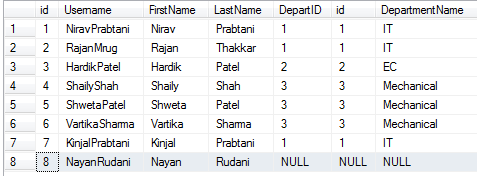
Left join displays all the rows from first table and matched rows from second table like that..

Hide   Copy Code

SELECT \* FROM Employee e1 LEFT OUTER JOIN Departments e2

ON e1.DepartID = e2.id

Result:



**b) Right outer join**

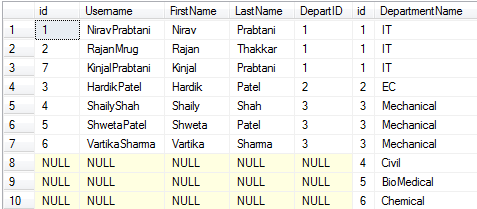
Right outer join displays all the rows of second table and matched rows from first table like that.

Hide   Copy Code

SELECT \* FROM Employee e1 RIGHT OUTER JOIN Departments e2

ON e1.DepartID = e2.id

Result:



**3) Full outer join**

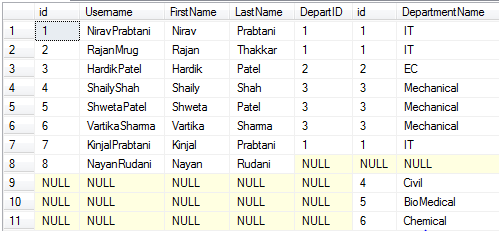
Full outer join returns all the rows from both tables whether it has been matched or not.

Hide   Copy Code

SELECT \* FROM Employee e1 FULL OUTER JOIN Departments e2

ON e1.DepartID = e2.id

Result:



**3) Cross Join**

A cross join that produces Cartesian product of the tables that are involved in the join. The size of a Cartesian product is the number of the rows in the first table multiplied by the number of rows in the second table like this.

Hide   Copy Code

SELECT \* FROM Employee cross join Departments e2

You can write a query like this also:

Hide   Copy Code

SELECT \* FROM Employee , Departments e2

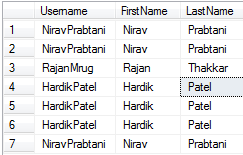
**4) Self Join**

Joining the table itself called self join. Self join is used to retrieve the records having some relation or similarity with other records in the same table. Here, we need to use aliases for the same table to set a self join between single table and retrieve records satisfying the condition in where clause.

Hide   Copy Code

SELECT e1.Username,e1.FirstName,e1.LastName from Employee e1 \_<br />inner join Employee e2 on e1.id=e2.DepartID

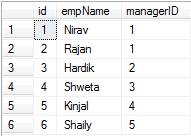
Here, I have retrieved data in which id and DepartID of employee table has been matched:



Points of Interest

Here, I have taken one example of self join in this scenario where manager name can be retrieved by manageridwith reference of employee id from one table.

Here, I have created one table employees like that:

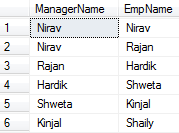


If I have to retrieve manager name from manager id, then it can be possible by Self join:

Hide   Copy Code

select e1.empName as ManagerName,e2.empName as EmpName \_<br />from employees e1 inner join employees e2 on e1.id=e2.managerid

Result:



# SQL | DDL, DML, DCL and TCL Commands

**Structured Query Language(SQL)** as we all know is the database language by the use of which we can perform certain operations on the existing database and also we can use this language to create a database. SQL uses certain commands like Create, Drop, Insert etc. to carry out the required tasks.

These SQL commands are mainly categorized into four categories as discussed below:

1. **DDL(Data Definition Language)** : DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in database.

**Examples of DDL commands:**

* + [**CREATE**](https://www.geeksforgeeks.org/sql-create/) – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
  + [**DROP**](https://www.geeksforgeeks.org/sql-drop-truncate/) – is used to delete objects from the database.
  + [**ALTER**](https://www.geeksforgeeks.org/sql-alter-add-drop-modify/)-is used to alter the structure of the database.
  + [**TRUNCATE**](https://www.geeksforgeeks.org/sql-drop-truncate/)–is used to remove all records from a table, including all spaces allocated for the records are removed.
  + [**COMMENT**](https://www.geeksforgeeks.org/sql-comments/) –is used to add comments to the data dictionary.
  + [**RENAME**](https://www.geeksforgeeks.org/sql-alter-rename/)–is used to rename an object existing in the database.

1. **DML(Data Manipulation Language)** : The SQL commands that deals with the manipulation of data present in database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

**Examples of DML:**

* + [**SELECT**](https://www.geeksforgeeks.org/sql-select-clause/) – is used to retrieve data from the a database.
  + [**INSERT**](https://www.geeksforgeeks.org/sql-insert-statement/) – is used to insert data into a table.
  + [**UPDATE**](https://www.geeksforgeeks.org/sql-update-statement/) – is used to update existing data within a table.
  + [**DELETE**](https://www.geeksforgeeks.org/sql-delete-statement/) – is used to delete records from a database table.

1. **DCL(Data Control Language)** : DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

**Examples of DCL commands:**

* + **GRANT**-gives user’s access privileges to database.
  + **REVOKE**-withdraw user’s access privileges given by using the GRANT command.

1. **TCL(transaction Control Language)** : TCL commands deals with the [transaction within the database](https://www.geeksforgeeks.org/sql-transactions/).

**Examples of TCL commands:**

* + **COMMIT**– commits a Transaction.
  + [**ROLLBACK**](https://www.geeksforgeeks.org/sql-transactions/)– rollbacks a transaction in case of any error occurs.
  + **SAVEPOINT**–sets a savepoint within a transaction.
  + **SET** **TRANSACTION**–specify characteristics for the transaction.

**Trigger:** A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

**Syntax:**

create trigger [trigger\_name]

[before | after]

{insert | update | delete}

on [table\_name]

[for each row]

[trigger\_body]

**Explanation of syntax:**

1. create trigger [trigger\_name]: Creates or replaces an existing trigger with the trigger\_name.
2. [before | after]: This specifies when the trigger will be executed.
3. {insert | update | delete}: This specifies the DML operation.
4. on [table\_name]: This specifies the name of the table associated with the trigger.
5. [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
6. [trigger\_body]: This provides the operation to be performed as trigger is fired

**BEFORE and AFTER of Trigger:**  
BEFORE triggers run the trigger action before the triggering statement is run.  
AFTER triggers run the trigger action after the triggering statement is run.

**Dirty Reads –**  
When a transaction is allowed to read a row that has been modified by an another transaction which is not committed yet that time *Dirty Reads* occurred. It is mainly occurred because of multiple transaction at a time which is not committed.

* **@@IDENTITY:** returns the last identity value generated on your SQL connection (SPID). Most of the time it will be what you want, but sometimes it isn't (like when a trigger is fired in response to an INSERT, and the trigger executes another INSERT statement).
* **SCOPE\_IDENTITY():** returns the last identity value generated in the current scope (i.e. stored procedure, trigger, function, etc).
* **IDENT\_CURRENT():** returns the last identity value for a specific table. Don't use this to get the identity value from an INSERT, it's subject to race conditions (i.e. multiple connections inserting rows on the same table).