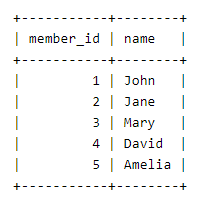
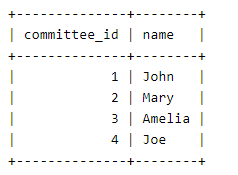
**. What are JOIN, INNER JOIN, LEFT JOIN, RIGHT JOIN? And give an example where they are useful.**



Here are two simple tables made to make some example on them to make our definitions more clear.

**Join** can be used to link multiple tables by a common column and normally it is foreign Key column. A join is a method of linking data between one ([self-join](http://www.mysqltutorial.org/mysql-self-join/)) or more tables based on values of the common column between the tables. JOIN returns all rows from tables where the key record of one table is equal to the key records of another table

## INNER JOIN

**INNER JOIN** matches each row in one table with every row in other tables and allows you to query rows that contain columns from both tables. We can also use inner join with join condition, it means if rows from both tables cause the join condition to evaluate to TRUE, the INNER JOIN creates a new row whose columns contain all columns of rows from the tables and includes this new row in the result set. Otherwise, the INNER JOIN just ignores the rows. The INNER JOIN selects all rows from both participating tables as long as there is a match between the columns.

Code :

SELECT

m.member\_id,

m.name AS member,

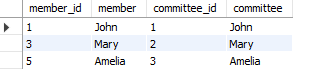
c.committee\_id,

c.name AS committee

FROM

members m

INNER JOIN committees c ON c.name = m.name;



The inner join is useful for us in general to match infos between tables. For example I want to make a table of peoples has a same talent for example we can choose the inner join to make a table containing and grouping each two persons or more have same talent.

LEFT JOIN

Deffinition : The left join selects data starting from the left table. For each row in the left table, the left join compares with every row in the right table.

If the values in the two rows satisfy the join condition, the left join clause creates a new row whose columns contain all columns of the rows in both tables and includes this row in the result set.

If the values in the two rows are not matched, the left join clause still creates a new row whose columns contain columns of the row in the left table and NULL for columns of the row in the right table.

MYSQL code:

**SELECT**

m.member\_id,

m.name AS member,

c.committee\_id,

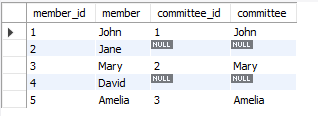
c.name AS committee

**FROM**

members m

**LEFT JOIN** committees c USING(name);

//Note here we could write **ON c.name = m.name** instead of **USING(name)**



We can see here that the table that is mentioned after ( **FROM** ) and here it is the members table, each row of it has been taken and been compared to all rows of the right table ( committees table) and when a row of the right table has the same name of the left table, a row will be created in result table and will have the first two columns same as the two columns for the selected row from the right table (members table) and for the two columns left it will be the columns from the right table ( committee table ) from the matched row found. And **IF we don’t find** a matched row (has the same name of the selected row name from the left table) the left JOIN also create a row who will has the forst two columns same for the row coming from the left table and will have another two columns where thair values will be **NULL**. Here we can see that the row ( 2 Jane ) has now matched row in the right table so in the result table the second two rows will be null.

If we want to benefit from the left Join in this example we cand simply add one more condition ligne to the code so the code will be :

SELECT

m.member\_id,

m.name AS member,

c.committee\_id,

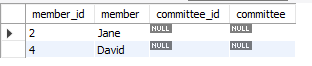
c.name AS committee

FROM

members m

LEFT JOIN committees c USING(name)

WHERE c.committee\_id IS NULL;



This code will help us to filter the result table and give us the members who are not a committee members.

## RIGHT JOIN

The [right join](https://www.mysqltutorial.org/mysql-right-join/) clause is similar to the left join clause except that the treatment of left and right tables is reversed. The right join starts selecting data from the right table instead of the left table.

The right join clause selects all rows from the right table and matches rows in the left table. If a row from the right table does not have matching rows from the left table, the column of the left table will have NULL in the final result set.

This section has the same explaination for the LEFT JOIN but in this case the selected row that will be searched and find a matched row for it is a row from ( the right table ) here in this case from the committee table.

Code :

SELECT

m.member\_id,

m.name AS member,

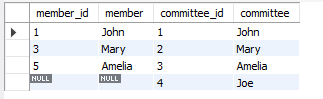
c.committee\_id,

c.name AS committee

FROM

members m

RIGHT JOIN committees c on c.name = m.name;



Offcoarse we have to make sure that will always has the same order of columns, it means the first two columns is from member table and the second two columns from the commeittee table because according to the code the first two one in from members table and the second two one is from the committee table. The procedure of making this table is :

For example if we take the first row from the right table ( committee table ) 🡺 ( 1 John ) , the right Join will start to search for a row similar to this row from the left table ( members table ) so it will find a row who has the same name and this row is the first row from the left table so it will create a result row in the result table that has order of columns same as mentioned in the code ( members columns then committees columns ) and this row will be ( 1 John 1 John ). And same thing from Lett Join if we don’t find a matched row the result row will be null value from the columns coming from the left table and same values from the right row coming from the right table.

If we want to benefit here from the right join and know the person who is a committee but not a member we can add this condition to the code :

SELECT

m.member\_id,

m.name AS member,

c.committee\_id,

c.name AS committee

FROM

members m

RIGHT JOIN committees c USING(name)

WHERE m.member\_id IS NULL;

The the result will be Joe



**MYSQL Index:**

First of all indexes make our queries super fast and double charge our application (it make it more fast and help her int fetching data).

For making the explaination more easy let’s take an example and explain his goal.

Example:

We have a website and we have a table ( authors ) for user informations. This table is too big with a big size.



Imagine that we have 23 rows for this table.

If we want to take information from this table we have to apply a certain querry on it.

so let’s apply a querry on it :

select \* from authors where first\_name = “Jonas” and last\_name = “Smith”;

if we apply this querry we can see that we have to wait at least 2 seconds for the showing result.



this time is normal for this huge table.So what if also have multiple joins and filters in this querry so it will take more time so we need **definitely** need an element for making our search faster.

So we can use Index.Index is telling MYSQ that a certain column or multiple are important for me. In thi example first\_name and last\_name are important for me. When I index them MYSQL take the values of these columns and store them in a separate location or a storage.

How to index these columns:

Let us first index only the first\_name column

Querry: ALTER TABLE authors ADD INDEX author\_index (first\_name);

This querry will take multiple seconds to be generated.

After this index if we try again to generate the main querry (fetch first\_name and last\_name) 🡺

select \* from authors where first\_name = “Jonas” and last\_name = “Smith”;

we can see that the operation didn’t take time like when we generated it without indexing the column ( first\_name ). It took only 12 milliseconds



now lets remove the index from the first\_name column and add index for the two columns (first\_name and last\_name) by applying these two queries:

**querry1**: DROP INDEX author\_index on authors;

**querry2**: ALTER TABLE authors ADD INDEX author\_index (first\_name,last\_name);

now lets appy the main querry ( select \* from authors where first\_name = “Jonas” and last\_name = “Smith”; )

we can see that this time take less time about ( 3 milliseconds )



So we can definitely see the importance of indexing for time saving. We saved time from 2 secondes to 3 millisecondes.

**It it always worth querying between two tables to fetch data?**

The table has one role and to hold data for certain informations but o be more professional and more orginize this table has to contain informations related to same domain. For example we can set one table that have all the informations of the user ( favorite color and his nationality… ) but it will too big and hard to read. We can mak two or multiple tables from this table, for example a table for user nationality and country… and a table for his talents and favorite color… Ofcoarse we can make these tables related by using constarints like foreign Key , Primery Key. So in my opinion we should querying between two tables or more for fetching data.

**Preparing table for data :**

This application has small amount of data type (not quantity). It means we are using only the data that is related to An issue Or ( PrItem ). So we can can make a table or two related by a foreign Key to save the data of PrItem ( prItem data is located in a decoration file called ‘types.d.ts’ ). We already created a generated value for the ID so we don’t have to make it ( AI : Auto Increment) soe we only have to save it and make it a foreign Key and offcoarse unique because we can’t have a same ID for two Issue. Let’s make two tables for more clarity.

**The first table will be :**

Querry:

CREATE TABLE `et3test`.`main\_table` (

`issue\_id` INT NOT NULL,

`comment` VARCHAR(150) NULL,

`link` VARCHAR(100) NULL,

`se` VARCHAR(20) NULL,

`platform` VARCHAR(45) NULL,

`size` VARCHAR(20) NULL,

`difficulty` VARCHAR(20) NULL,

`status` VARCHAR(20) NULL,

`version` VARCHAR(10) NULL,

PRIMARY KEY (`issue\_id`),

UNIQUE INDEX `idmain\_table\_UNIQUE` (`issue\_id` ASC) VISIBLE);



**The second table will be :**

Querry :

CREATE TABLE `et3test`.`second\_table` (

`issue\_id` INT NULL,

`by\_status` VARCHAR(45) NULL,

`ah\_status` VARCHAR(45) NULL,

`ht\_status` VARCHAR(45) NULL,

`by\_state` TINYINT NULL,

`ah\_state` TINYINT NULL,

`ht\_state` TINYINT NULL,

`date` DATE NULL,

`date\_string` VARCHAR(45) NULL,

INDEX `issue\_id\_idx` (`issue\_id` ASC) VISIBLE,

CONSTRAINT `issue\_id`

FOREIGN KEY (`issue\_id`)

REFERENCES `et3test`.`main\_table` (`issue\_id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION);



UML Diagram of this tables :

