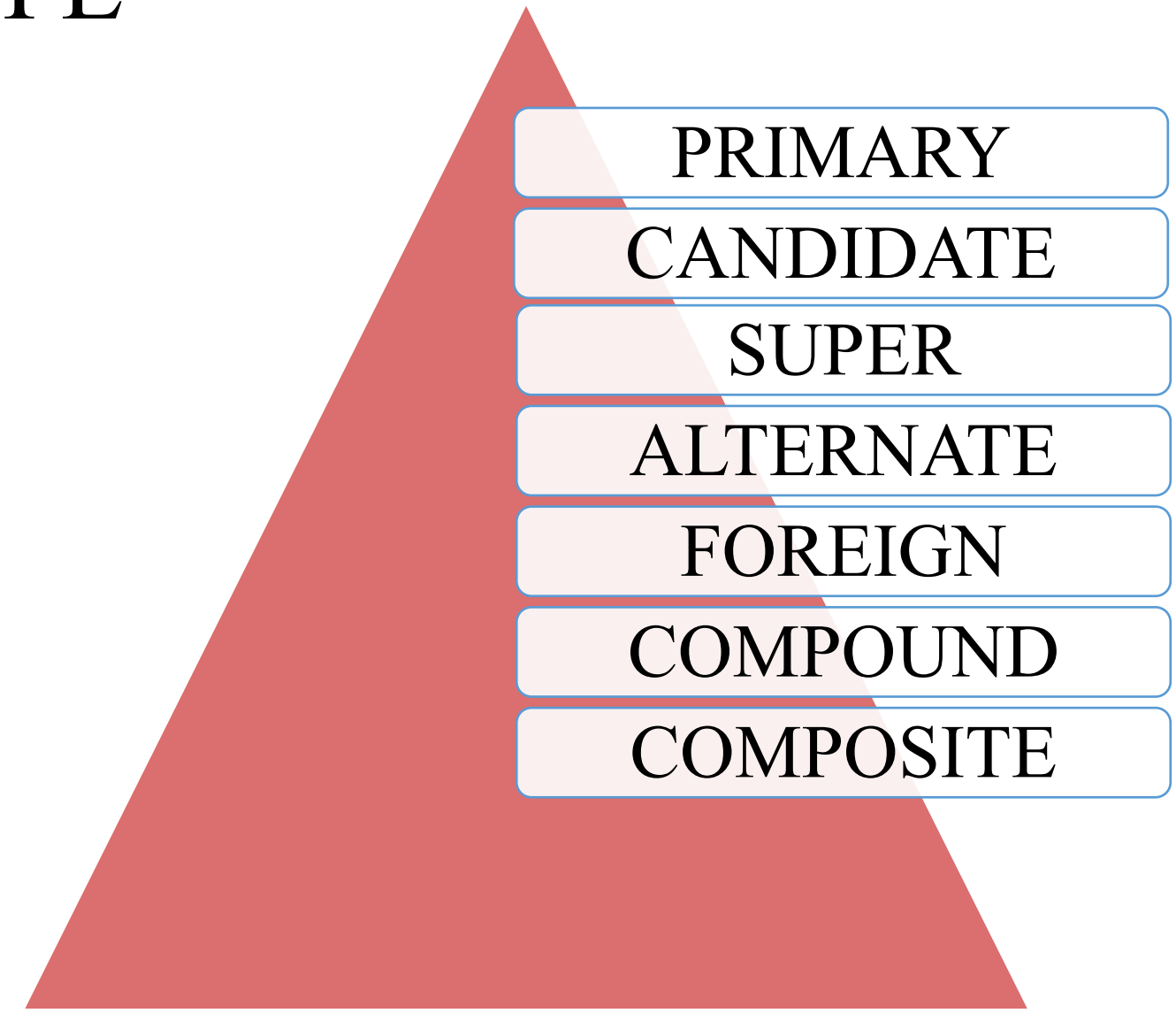
KEYS and Join in SQL

KEYS

- Play an important role in the relational database.
- It is an attribute or set of attributes
  - which helps you to identify a row(tuple) in a relation(table).
- It allow to find the relation between two tables.
- It helps
  - uniquely identify a row in a table by a combination of one or more columns in that table.

KEY: TYPE


PRIMARY
CANDIDATE
SUPER
ALTERNATE
FOREIGN
COMPOUND
COMPOSITE

Primary Key

- A single field or combination of fields that uniquely defines a record.
- None of the fields that are part of the primary key can contain a NULL value
- A table can have only one primary key(unique).
  - Ex-Student_id or <Student_id, Student name> are primary key

Student_id	Student_name	Student_branch
coe19d001	Venketesh	CSE
coe19d002	Shree Prakash	CSE

Candidate key

- Candidate key is a primary key
- Definition:
  - Subset of primary key is not a primary key
  - Student_id is a candidate key
  - <Student_id, Student name> is not a candidate key

Student_id	Student_name	Student_branch
coe19d001	Venketesh	CSE
coe19d002	Shree Prakash	CSE

primary key in MySQL

- A primary key is created using
  - either a CREATE TABLE statement

    or
  - an ALTER TABLE statement.

- ALTER TABLE statement in MySQL
  - to drop, disable or enable a primary key.

Primary Key - Using CREATE TABLE statement

```
CREATE TABLE table_name ( column1 column_definition,
column2 column_definition, ...
CONSTRAINT [constraint_name]
PRIMARY KEY
(column1, column2, ... column_n) );
```

table_name: The name of the table that you wish to create.

column1, column2

The columns that you wish to create in the table.

constraint_name

The name of the primary key.

column1, column2, ... column_n

The columns that make up the primary key.

Example1

```
CREATE TABLE contact ( contact_id INT(11) NOT NULL,
 last_name VARCHAR(30) NOT NULL, first_name
VARCHAR(25),
CONSTRAINT contacts_pk PRIMARY KEY (contact_id) );
```

In this example, we've created a primary key on the ***contacts*** table called ***contacts_pk***. It consists of only one column - **the *contact_id* column**.

```
mysql> create database if not exists mykeys;
Query OK, 1 row affected (2.67 sec)
```

```
mysql> use mykeys;
Database changed
```

```
mysql> CREATE TABLE contact ( contact_id INT(11) NOT NULL,
    ->  last_name VARCHAR(30) NOT NULL, first_name
    -> VARCHAR(25),
    -> CONSTRAINT contacts_pk PRIMARY KEY (contact_id) );
Query OK, 0 rows affected, 1 warning (1.04 sec)
```

```
mysql> insert into contacts(contact_id,last_name,first_name)values(006,'shree','prakash');
Query OK, 1 row affected (0.13 sec)

mysql> insert into contacts(contact_id,last_name,first_name)values(006,'shree','Ramesh');
ERROR 1062 (23000): Duplicate entry '6' for key 'contacts.PRIMARY'
mysql> insert into contacts(contact_id,last_name,first_name)values(007,'shree','Ramesh');
Query OK, 1 row affected (0.10 sec)
```

```
mysql> select *from contact;
+------------+-----------+------------+
| contact_id | last_name | first_name |
+------------+-----------+------------+
|          6 | shree     | prakash    |
|          7 | shree     | Ramesh     |
|          8 | shree     | Ramesh     |
+------------+-----------+------------+
3 rows in set (0.00 sec)
```

Example2

```
CREATE TABLE contacts ( last_name VARCHAR(30) NOT NULL,
 first_name VARCHAR(25) NOT NULL,
 birthday DATE, CONSTRAINT contacts_pk
PRIMARY KEY (last_name, first_name) );
```

This example creates a **primary key** called ***contacts_pk*** that is made up of a combination of the ***last_name*** and ***first_name*** columns. So each combination of ***last_name*** and ***first_name*** must be **unique** in the *contacts* table.

Primary Key - Using ALTER TABLE statement

```
ALTER TABLE table_name ADD CONSTRAINT [ constraint_name ]
 PRIMARY KEY  (column1, column2,... column_n) ;
```

table_name
The name of the table to modify.
constraint_name
The name of the primary key.
column1, column2, ... column_n
The columns that make up the primary key.

Example1

```
ALTER TABLE contacts ADD CONSTRAINT contacts_pk
PRIMARY KEY (contact_id);
```

In this example, we've created a primary key on the existing *contacts* table called *contacts_pk*. It consists of the *contact_id* column.

Example2

```
ALTER TABLE contacts ADD CONSTRAINT contacts_pk
 PRIMARY KEY (last_name, first_name);
```

a primary key called contacts_pk that is made up of a combination of the *last_name* and *first_name* columns.

Drop Primary Key

```
ALTER TABLE table_name DROP PRIMARY KEY;
```

Example:

```
ALTER TABLE contacts DROP PRIMARY KEY;
```

In this example, we've dropped the primary key on the *contacts* table.

NOTE: We do not need to specify the name of the primary key as there can only be one on a table.

FOREIGN KEY

- A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
- The table containing the foreign key is called the child table
- The table containing the candidate key is called the referenced or parent table.
- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

Example

"Persons" table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

"Orders" table:

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

"PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

FOREIGN KEY on CREATE TABLE

- **CREATE TABLE Orders (**
  OrderID int NOT NULL,
  OrderNumber int NOT NULL,
  PersonID int,
  PRIMARY KEY (OrderID),
  FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
  );

- SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

FOREIGN KEY on multiple columns

```
•CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (Person
 ID)
    REFERENCES Persons(PersonID)
 );
```

FOREIGN KEY on ALTER TABLE

- To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:
  - ALTER TABLE Orders
    ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
- FOREIGN KEY constraint on multiple columns, SQL syntax:
  - ALTER TABLE Orders
    ADD CONSTRAINT FK_PersonOrder
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

DROP a FOREIGN KEY

- ALTER TABLE Orders
  DROP FOREIGN KEY FK_PersonOrder;

- Example:
  - ALTER TABLE Orders
    DROP CONSTRAINT FK_PersonOrder;

SUPER KEY

- a superset of a candidate key.
- a set of an attribute which can uniquely identify a tuple

```
mysql> select *from contact;
+------------+-----------+------------+
| contact_id | last_name | first_name |
+------------+-----------+------------+
|          6 | shree     | prakash    |
|          7 | shree     | Ramesh     |
```

(contact_id, last_name) is a super key

COMPOUND KEY

- combination of two or more columns that uniquely identify rows in a table.
- Field may or may not a primary key
- there combination must be unique

(last_name, first_name)

```
mysql> select *from contact;
+------------+-----------+------------+
| contact_id | last_name | first_name |
+------------+-----------+------------+
|          6 | shree     | prakash    |
|          7 | shree     | Ramesh     |
```

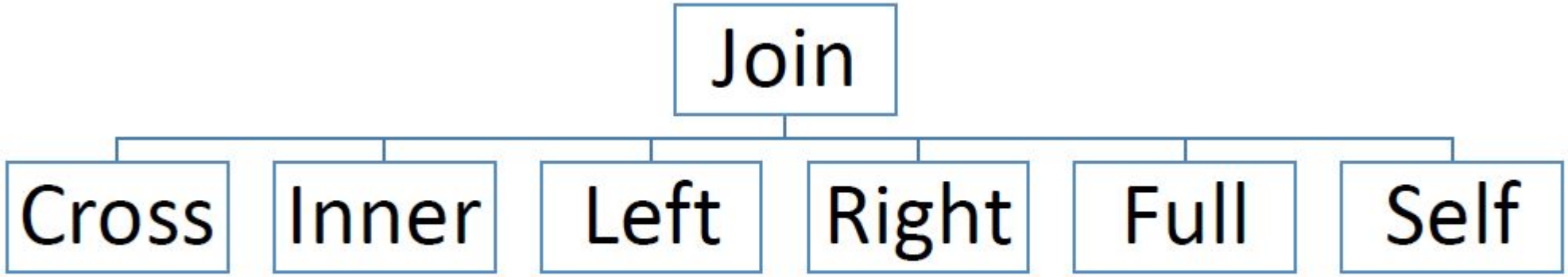

JOIN

JOIN

Table1: committees	Table2: member
committee_id	member_id
Name	Name

Find the name in committee who is a member.
Both the tables need to be queried

- Method of linking data between one or more tables based on values of the common column between the tables.



CREATION OF TABLE

```
mysql> create database onlyjoin;
Query OK, 1 row affected (2.17 sec)
```

```
mysql> use onlyjoin;
Database changed
```

```
Database changed
mysql> CREATE TABLE members (member_id INT AUTO_INCREMENT,name VARCHAR(100),PRIMARY KEY (member_id));
Query OK, 0 rows affected (4.19 sec)
```

```
mysql> INSERT INTO members(name)VALUES('John'),('Jane'),('Mary'),('David'),('Amelia');
Query OK, 5 rows affected (0.43 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

```
mysql> CREATE TABLE committees (committee_id INT AUTO_INCREMENT, name VARCHAR(100),PRIMARY KEY (committee_id));
Query OK, 0 rows affected (1.00 sec)

mysql> INSERT INTO committees(name)VALUES('John'),('Mary'),('Amelia'),('Joe');
Query OK, 4 rows affected (0.25 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

TABLE 1: members

```
mysql> select * from members;
+-----------+--------+
| member_id | name   |
+-----------+--------+
|         1 | John   |
|         2 | Jane   |
|         3 | Mary   |
|         4 | David  |
|         5 | Amelia |
+-----------+--------+
5 rows in set (0.12 sec)
```

TABLE 2: committees

```
mysql> select *from committees;
+--------------+--------+
| committee_id | name   |
+--------------+--------+
|            1 | John   |
|            2 | Mary   |
|            3 | Amelia |
|            4 | Joe    |
+--------------+--------+
4 rows in set (0.00 sec)
```

CROSS JOIN

Cartesian product of rows from the joined tables (NO CONDITION).

Combines each row from the first table with every row from the right table.

SELECT select_list FROM table_1 CROSS JOIN table_2;

```
mysql> select * from committees cross join members;
+--------------+--------+-----------+--------+
| committee_id | name   | member_id | name   |
+--------------+--------+-----------+--------+
|            1 | John   |         1 | John   |
|            2 | Mary   |         1 | John   |
|            3 | Amelia |         1 | John   |
|            4 | Joe    |         1 | John   |
|            1 | John   |         2 | Jane   |
|            2 | Mary   |         2 | Jane   |
|            3 | Amelia |         2 | Jane   |
|            4 | Joe    |         2 | Jane   |
|            1 | John   |         3 | Mary   |
|            2 | Mary   |         3 | Mary   |
|            3 | Amelia |         3 | Mary   |
|            4 | Joe    |         3 | Mary   |
|            1 | John   |         4 | David  |
|            2 | Mary   |         4 | David  |
|            3 | Amelia |         4 | David  |
|            4 | Joe    |         4 | David  |
|            1 | John   |         5 | Amelia |
|            2 | Mary   |         5 | Amelia |
|            3 | Amelia |         5 | Amelia |
|            4 | Joe    |         5 | Amelia |
+--------------+--------+-----------+--------+
20 rows in set (0.00 sec)
```

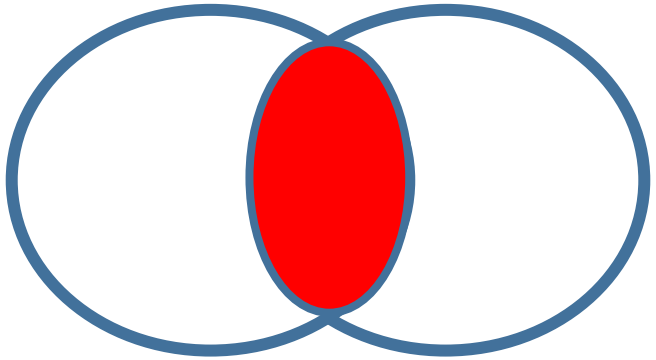
```
mysql> select committee_id from committees cross join members;
+--------------+
| committee_id |
+--------------+
|            1 |
|            2 |
|            3 |
|            4 |
|            1 |
|            2 |
|            3 |
|            4 |
|            1 |
|            2 |
|            3 |
|            4 |
|            1 |
|            2 |
|            3 |
|            4 |
|            1 |
|            2 |
|            3 |
|            4 |
+--------------+
20 rows in set (0.00 sec)
```

INNER JOIN

SELECT column_list **FROM** table_1 **INNER JOIN** table_2 **ON** join_condition;

compares each row from the first table with every row from the second table

- If values in both rows cause the join condition evaluates to true,
- the inner join clause creates a new row whose column
- contains all columns of the two rows from both tables and include this new row in the final result set.



the inner join clause includes only rows whose values match.

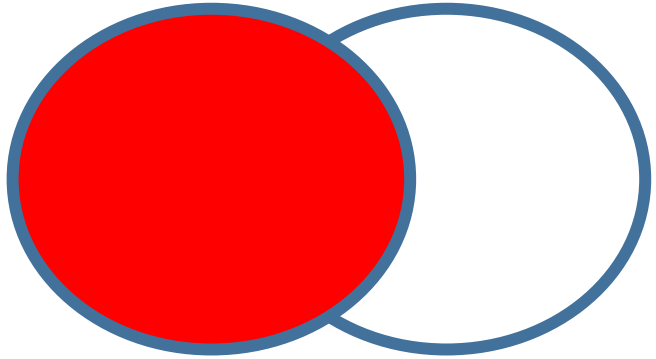
```
mysql> select * from members;
+-----------+--------+
| member_id | name   |
+-----------+--------+
|         1 | John   |
|         2 | Jane   |
|         3 | Mary   |
|         4 | David  |
|         5 | Amelia |
+-----------+--------+
5 rows in set (0.12 sec)
```

```
mysql> select *from committees;
+--------------+--------+
| committee_id | name   |
+--------------+--------+
|            1 | John   |
|            2 | Mary   |
|            3 | Amelia |
|            4 | Joe    |
+--------------+--------+
4 rows in set (0.00 sec)
```

```
mysql> select * from committees inner join members on committees.name=members.name;
+--------------+--------+-----------+--------+
| committee_id | name   | member_id | name   |
+--------------+--------+-----------+--------+
|            1 | John   |         1 | John   |
|            2 | Mary   |         3 | Mary   |
|            3 | Amelia |         5 | Amelia |
+--------------+--------+-----------+--------+
3 rows in set (0.06 sec)
```

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 cause the join condition evaluates to true
  - the left join creates a new row whose columns contain all columns of the rows in both tables and includes this row in the result set.
  - In case there is no matching rows from the right table found, NULLs are used for columns of the row from the right table in the final result set



SELECT column_list **FROM** table_1 **LEFT JOIN** table_2 **ON** join_condition;

```
mysql> select * from committees left join members on committees.name=members.name;
+--------------+--------+-----------+--------+
| committee_id | name   | member_id | name   |
+--------------+--------+-----------+--------+
|            1 | John   |         1 | John   |
|            2 | Mary   |         3 | Mary   |
|            3 | Amelia |         5 | Amelia |
|            4 | Joe    |      NULL | NULL   |
+--------------+--------+-----------+--------+
4 rows in set (0.06 sec)
```

```
mysql> select * from committees left join members using (name);
+--------+--------------+-----------+
| name   | committee_id | member_id |
+--------+--------------+-----------+
| John   |            1 |         1 |
| Mary   |            2 |         3 |
| Amelia |            3 |         5 |
| Joe    |            4 |      NULL |
+--------+--------------+-----------+
4 rows in set (0.00 sec)
```

If column name is same

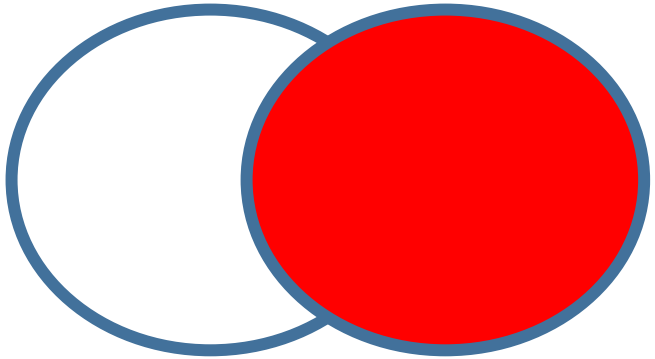
Right Join

- 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.

```
SELECT column_list FROM table_1 RIGHT JOIN table_2 ON join_condition;
```

If column name is same:

```
SELECT column_list FROM table_1 RIGHT JOIN table_2 USING (column_name);
```



```
mysql> select * from committees right join members on committees.name=members.name;
+--------------+--------+-----------+--------+
| committee_id | name   | member_id | name   |
+--------------+--------+-----------+--------+
|            1 | John   |         1 | John   |
|         NULL | NULL   |         2 | Jane   |
|            2 | Mary   |         3 | Mary   |
|         NULL | NULL   |         4 | David  |
|            3 | Amelia |         5 | Amelia |
+--------------+--------+-----------+--------+
5 rows in set (0.00 sec)
```

```
mysql> select * from members;
+-----------+--------+
| member_id | name   |
+-----------+--------+
|         1 | John   |
|         2 | Jane   |
|         3 | Mary   |
|         4 | David  |
|         5 | Amelia |
+-----------+--------+
5 rows in set (0.12 sec)
```

```
mysql> select * from committees right join members using(name);
+--------+-----------+--------------+
| name   | member_id | committee_id |
+--------+-----------+--------------+
| John   |         1 |            1 |
| Jane   |         2 |         NULL |
| Mary   |         3 |            2 |
| David  |         4 |         NULL |
| Amelia |         5 |            3 |
+--------+-----------+--------------+
5 rows in set (0.00 sec)
```

```
mysql> select *from committees;
+--------------+--------+
| committee_id | name   |
+--------------+--------+
|            1 | John   |
|            2 | Mary   |
|            3 | Amelia |
|            4 | Joe    |
+--------------+--------+
4 rows in set (0.00 sec)
```

Full Join

```
SELECT table1.column1,
table2.column2... FROM table1 FULL
JOIN table2 ON table1.common_field =
table2.common_field;
```

combines the results of both left and right outer joins

table will contain all records from both the tables and fill in NULLs for missing matches on either side

```
mysql> select * from members full join committees on  committee_id=member_id;
+-----------+-------+--------------+--------+
| member_id | name  | committee_id | name   |
+-----------+-------+--------------+--------+
|         1 | John  |            1 | John   |
|         2 | Jane  |            2 | Mary   |
|         3 | Mary  |            3 | Amelia |
|         4 | David |            4 | Joe    |
+-----------+-------+--------------+--------+
4 rows in set (0.00 sec)
```

```
mysql> select committee_id,member_id from committees full join members using(name);
+--------------+-----------+
| committee_id | member_id |
+--------------+-----------+
|            1 |         1 |
|            2 |         3 |
|            3 |         5 |
+--------------+-----------+
3 rows in set (0.00 sec)
```

Self Join

- Membership operator allows to qualify an SQL identifier with another SQL identifier of which it is a component.
- Aliases
  - give a table, or a column in a table, a temporary name.
  - only exists for the duration of the query.
  - often used to make column names more readable
  - SELECT *column_name* AS *alias_name* FROM *table_name;*

```
SELECT a.column_name, b.column_name...
 FROM table1 a, table1 b WHERE a.common_field = b.common_field;
```

Self Join

SELECT a.column_name, b.column_name...
FROM table1 a, table1 b WHERE a.common_field = b.common_field;

```
mysql> select * from committees a, committees b where a.name=b.name;
+--------------+--------+--------------+--------+
| committee_id | name   | committee_id | name   |
+--------------+--------+--------------+--------+
|            1 | John   |            1 | John   |
|            2 | Mary   |            2 | Mary   |
|            3 | Amelia |            3 | Amelia |
|            4 | Joe    |            4 | Joe    |
+--------------+--------+--------------+--------+
4 rows in set (0.00 sec)
```

```
mysql> select a.committee_id, b.name from committees a, committees b where a.name=b.name;
+--------------+--------+
| committee_id | name   |
+--------------+--------+
|            1 | John   |
|            2 | Mary   |
|            3 | Amelia |
|            4 | Joe    |
+--------------+--------+
4 rows in set (0.06 sec)
```

LAB Exercise

- Consider a Table name IIITDM(Faculty name, Student_Id, Building name),Faculty(Faculty_id,Faculty_name,department,Subject),Student(Student_Id, Student_name, department, course, Building_name), Building(Building_name, Room_no,Floor), Course(department, subject, course_id).
  1. Find the student_id who is learning course from a particular faculty and living in room no 30.
  2. Check the course of a student living in room no 140 in a given building.
  3. Find faculty who is not teaching subject to a particular student.
  4. Find the course taught by the faculty to the student lives in third floor of a particular building.