The difference between One-to-many, Many-to-one and Many-to-Many is:

**One-to-many vs Many-to-one is a matter of perspective**. Unidirectional vs Bidirectional will not affect the mapping but will make difference on how you can access your data.

* In Many-to-one the many side will keep reference of the one side. A good example is "A State has Cities". In this case State is the one side and City is the many side. There will be a column state\_id in the table cities.

In *unidirectional*, Person class will have List<Skill> skills but Skill will not have Person person. In *bidirectional*, both properties are added and it allows you to access a Person given a skill( i.e. skill.person).

* In One-to-Many the one side will be our point of reference. For example, "A User has Addresses". In this case we might have three columns address\_1\_id, address\_2\_id and address\_3\_id or a [look up table](https://stackoverflow.com/questions/3419847/what-is-a-lookup-table) with [multi column unique constraint](https://stackoverflow.com/questions/635937/how-do-i-specify-unique-constraint-for-multiple-columns-in-mysql) on user\_id on address\_id.

In *unidirectional*, a User will have Address address. *Bidirectional* will have an additional List<User> users in the Address class.

* In Many-to-Many members of each party can hold reference to arbitrary number of members of the other party. To achieve this a [look up table](https://stackoverflow.com/questions/3419847/what-is-a-lookup-table) is used. Example for this is the relationship between doctors and patients. A doctor can have many patients and vice versa.

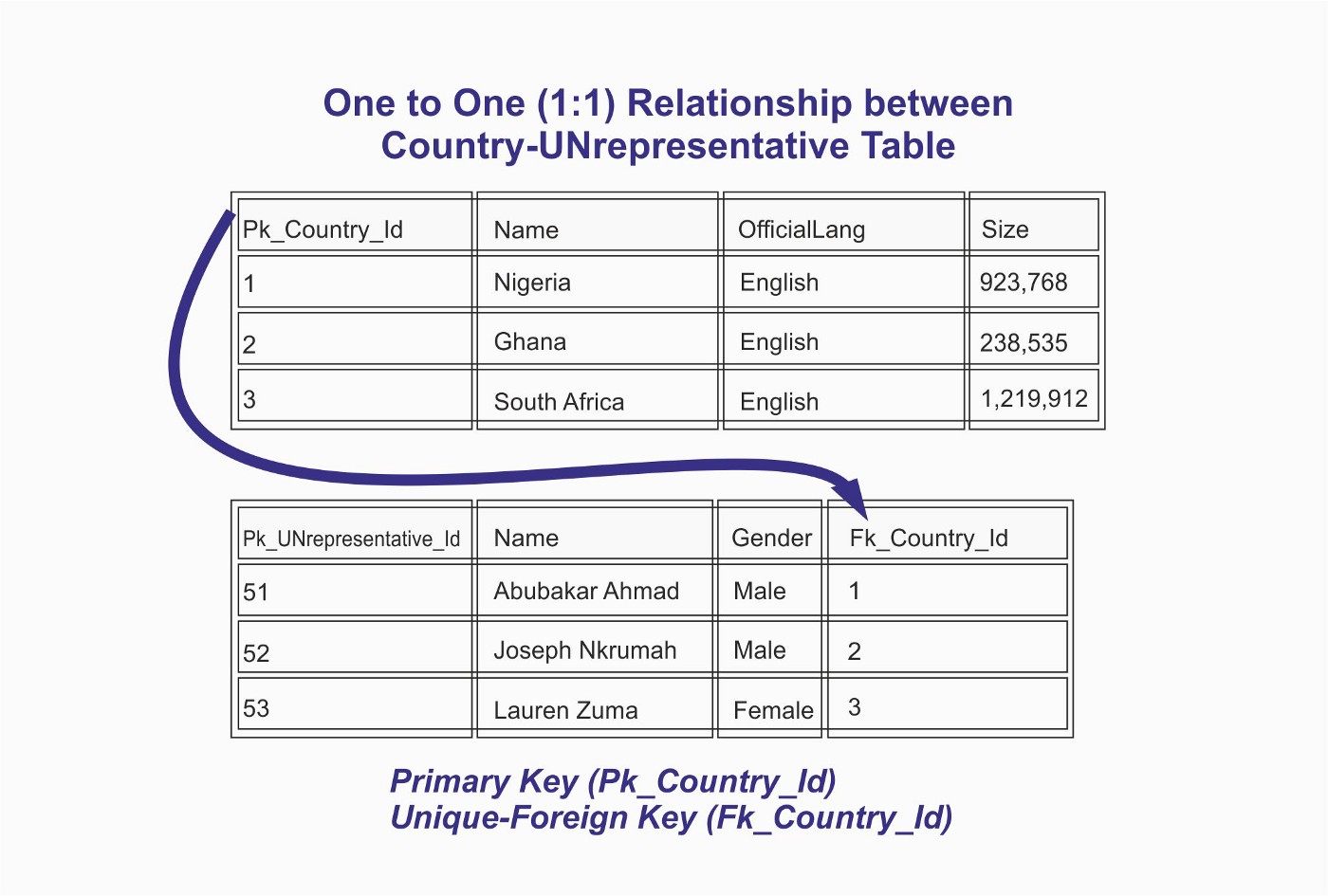
**One to One Relationship (1:1)**

When a row in a table is related to only one role in another table and vice versa,we say that is a **one to one relationship**. This relationship can be created using *Primary key-Unique foreign key constraints*.

*For instance a Country can only have one UN Representative, and also a UN Representative can only represent one Country.*

*Let’s try out the implement and see for ourselves*.

CREATE TABLE Country  
(  
Pk\_Country\_Id INT IDENTITY PRIMARY KEY,  
Name VARCHAR(100),  
Officiallang VARCHAR(100),  
Size INT(11),  
);  
CREATE TABLE UNrepresentative  
(  
Pk\_UNrepresentative\_Id INT PRIMARY KEY,  
Name VARCHAR(100),  
Gender VARCHAR(100),  
Fk\_Country\_Id INT UNIQUE FOREIGN KEY REFERENCES Country(Pk\_Country\_Id)  
);  
INSERT INTO Country ('Name','Officiallang',’Size’)  
VALUES ('Nigeria','English',923,768);INSERT INTO Country ('Name','Officiallang',’Size’)  
VALUES ('Ghana','English',238,535);INSERT INTO Country ('Name','Officiallang',’Size’)  
VALUES ('South Africa','English',1,219,912);INSERT INTO UNrepresentative ('Pk\_Unrepresentative\_Id','Name','Gender','Fk\_Country\_Id')  
VALUES (51,'Abubakar Ahmad','Male',1);INSERT INTO UNrepresentative ('Pk\_Unrepresentative\_Id','Name','Gender','Fk\_Country\_Id')  
VALUES (52,'Joseph Nkrumah','Male',2);INSERT INTO UNrepresentative ('Pk\_Unrepresentative\_Id','Name','Gender','Fk\_Country\_Id')  
VALUES (53,'Lauren Zuma,'Female',3);  
SELECT \* FROM Country  
SELECT \* FROM UNrepresentative;



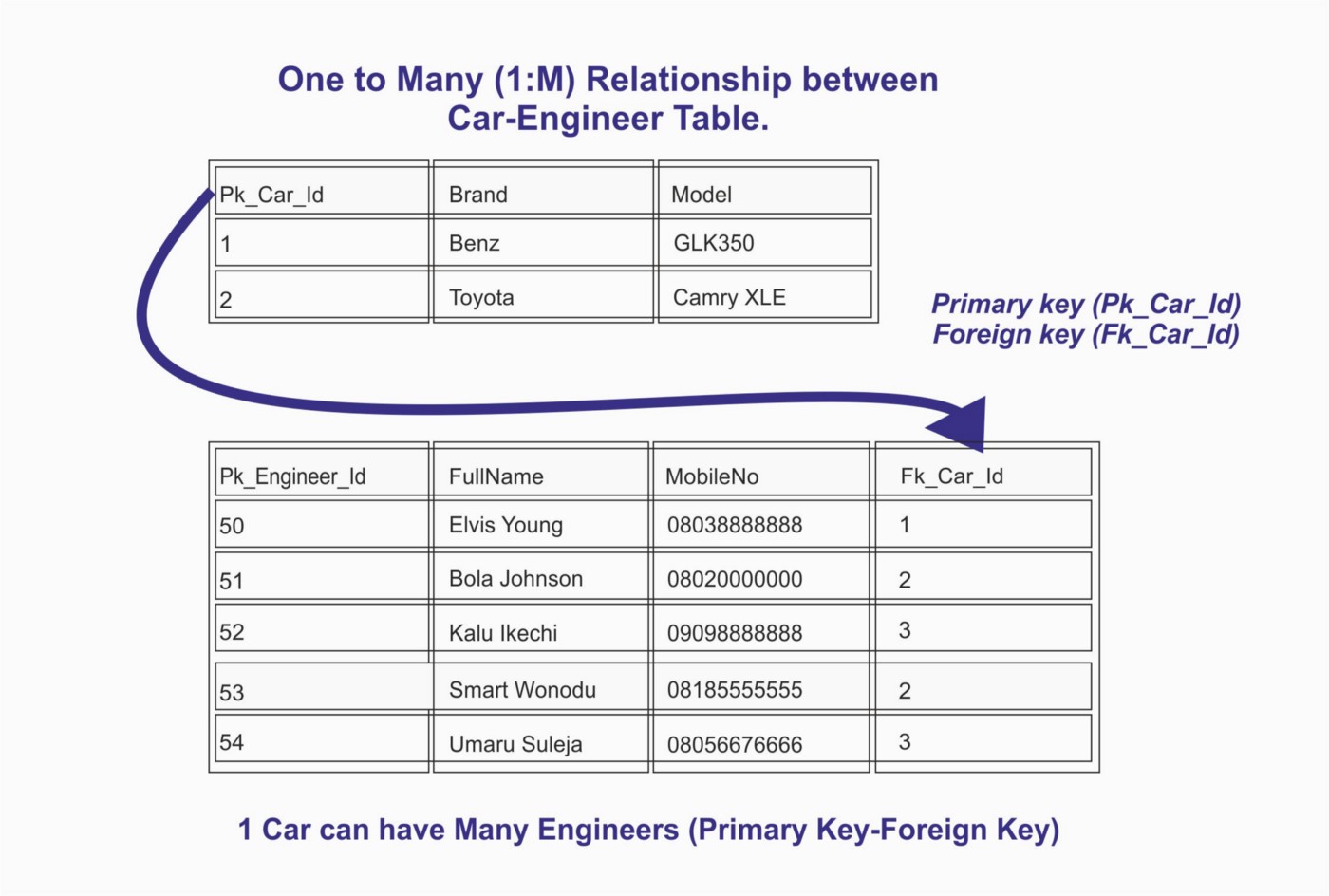
**One to Many Relationship (1:M)**

This is where a row from one table can have multiple matching rows in another table this relationship is defined as a **one to many relationship**. This type of relationship can be created using *Primary key-Foreign key relationship.*

This kind of Relationship, allows a Car to have multiple Engineers.

*Let’s try out the implementation and see for ourselves.*

CREATE TABLE Car  
(  
Pk\_Car\_Id INT PRIMARY KEY,  
Brand VARCHAR(100),  
Model VARCHAR(100)  
);CREATE TABLE Engineer  
(  
Pk\_Engineer\_Id INT PRIMARY KEY,  
FullName VARCHAR(100),  
MobileNo CHAR(11),  
Fk\_Car\_Id INT FOREIGN KEY REFERENCES Car(Pk\_Car\_Id)  
);  
INSERT INTO Car ('Brand','Model')  
VALUES ('Benz','GLK350');INSERT INTO Car ('Brand','Model')  
VALUES ('Toyota','Camry XLE');INSERT INTO Engineer ('Pk\_Engineer\_Id','FullName','MobileNo','Fk\_Car\_Id')  
VALUES(50,'Elvis Young','08038888888',2);INSERT INTO Engineer ('Pk\_Engineer\_Id','FullName','MobileNo','Fk\_Car\_Id')  
VALUES(51,'Bola Johnson','08020000000',1);INSERT INTO Engineer ('Pk\_Engineer\_Id','FullName','MobileNo','Fk\_Car\_Id')  
VALUES(52,'Kalu Ikechi','09098888888',1);INSERT INTO Engineer ('Pk\_Engineer\_Id','FullName','MobileNo','Fk\_Car\_Id')  
VALUES(53,'Smart Wonodu','08185555555',1);INSERT INTO Engineer ('Pk\_Engineer\_Id','FullName','MobileNo','Fk\_Car\_Id')  
VALUES(54,Umaru Suleja','08056676666',1);SELECT \* FROM Car;  
SELECT \* FROM Engineer;



One-to-Many (1:M) Relationship between Car-Engineer Table

*Car(Pk\_Car\_Id)*serving as*the Primary Key*and*Engineer(Fk\_Car\_Id)**as (Foreign Key)*.

Based on the Car (Pk\_Car\_Id)-Engineer(Fk\_Car\_Id) relationship, we now have a design for our database tables that consolidates the One-to-Many relationship using a foreign key!

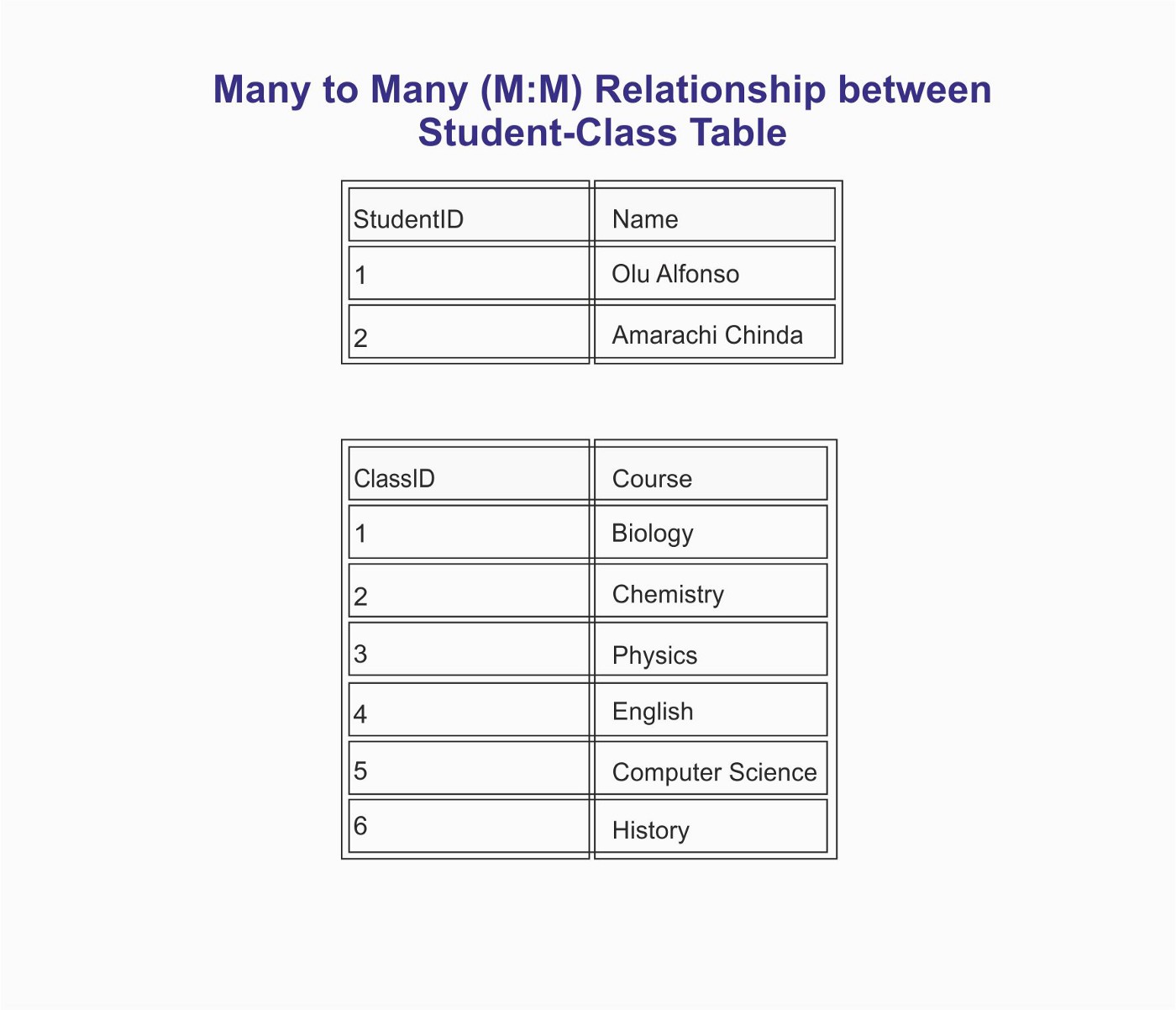
**Many to Many Relationship (M:M)**

A row from one table can have multiple matching rows in another table, and a row in the other table can also have multiple matching rows in the first table this relationship is defined as a **many to many relationship**. This type of relationship can be created using a third table called “*Junction table*” or “*Bridging table*”. This Junction or Bridging table can be assumed as a place where attributes of the relationships between two lists of entities are stored.

This kind of Relationship, allows a junction or bridging table as a connection for the two tables.

*Let’s try out the implementation and see for ourselves*.

CREATE TABLE Student(  
StudentID INT(10) PRIMARY KEY,  
Name VARCHAR(100),  
);CREATE TABLE Class(  
ClassID INT(10) PRIMARY KEY,  
Course VARCHAR(100),  
);CREATE TABLE StudentClassRelation(  
StudentID INT(15) NOT NULL,  
ClassID INT(14) NOT NULL,FOREIGN KEY (StudentID) REFERENCES Student(StudentID),  
FOREIGN KEY (ClassID) REFERENCES Class(ClassID),  
UNIQUE (StudentID, ClassID)  
);INSERT INTO Student ('Name')  
VALUES ('Olu Alfonso');INSERT INTO Student ('Name')  
VALUES ('Amarachi Chinda');  
INSERT INTO Class ('Course')  
VALUES ('Biology');INSERT INTO Class ('Course')  
VALUES ('Chemistry');INSERT INTO Class ('Type')  
VALUES ('Physics');INSERT INTO Class ('Type')  
VALUES ('English');INSERT INTO Class ('Type')  
VALUES ('Computer Science');INSERT INTO Class ('Type')  
VALUES ('History');INSERT INTO StudentClassRelation ('StudentID','ClassID')  
VALUES (1,2);INSERT INTO StudentClassRelation ('StudentID','ClassID')  
VALUES (1,4);INSERT INTO StudentClassRelation ('StudentID','ClassID')  
VALUES (1,6);INSERT INTO StudentClassRelation ('StudentID','ClassID')  
VALUES (2,3);INSERT INTO StudentClassRelation ('StudentID','ClassID')  
VALUES (2,1);INSERT INTO StudentClassRelation ('StudentID','ClassID')  
VALUES (2,6);  
INSERT INTO StudentClassRelation ('StudentID','ClassID')  
VALUES (2,1);



Many to Many (M:M) Relationship between Student-Class Table

