When normalizing a database or adding tables to an existing database, we need to be able to relate tables to each other.

There are three ways in which a table can be related to another table:

1. **One-to-one**: A record in one table is related to one record in another table.
2. **One-to-many**: A record in one table is related to many records in another table.
3. **Many-to-many**: Multiple records in one table are related to multiple records in another table.

Handling a one-to-one relationship or a one-or-many relationship can be done by adding the *primary key* of one table into the other table as a [*foreign key*](https://dzone.com/articles/how-to-create-a-foreign-key-on-a-table-in-oracle-s).

However, for many-to-many relationships, it’s a bit different. Let’s have a look at an example.

**Many-to-Many Relationships: An Example**

Let’s say we are creating a database for a university (which is an example I’ve used often). We capture details about students who attend classes, among other things. The rules are:

* A student can be enrolled in multiple classes at a time (for example, they may have three or four classes per semester).
* A class can have many students (for example, there may be 20 students in one class).

This means a student has many classes, and a class has many students.

We can’t add the primary key of one table into the other, or both, because this only stores a *single* relationship, and we need *many*.

We couldn’t do this:

|  |  |  |
| --- | --- | --- |
| **Student ID** | **Class ID** | **Student Name** |
| 1 | 3, 5, 9 | John |
| 2 | 1, 4, 5, 9 | Debbie |

This would mean we have one column for storing multiple values, which is very hard for maintenance and querying.

We also couldn’t have many columns for class ID values, as this would get messy and create a limit on the number of relationships.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Student ID** | **Class ID 1** | **Class ID 2** | **Class ID 3** | **Student Name** |
| 1 | 3 | 5 | 9 | John |
| 2 | 1 | 4 | 5 | Debbie |

**Many-to-Many Relationships**

So how do we capture this?

We use a concept called a *joining table* or a *bridging table*.

A *joining table* is a table that sits between the two other tables of a many-to-many relationship. Its purpose is to store a record for each of the combinations of these other two tables. It might seem like a bit of work to create, but it’s simple to do and provides a much better data structure.

To create one for this example, we can create a new table called class\_enrolment.

Now, the name of the table is important. It’s good to be descriptive with the table. I’ve seen joining tables named with the two names of the other two tables together (such as student\_class). I think this is valid and will get the job done, but having a more descriptive name is helpful, as it tells you more about what the table is.

So, we have a new table called class\_enrollment. It stores two columns: one for each of the primary keys from the other table.

Our table would look like this:

|  |  |
| --- | --- |
| **Student ID** | **Class ID** |
| 1 | 3 |
| 1 | 5 |
| 1 | 9 |
| 2 | 1 |
| 2 | 4 |
| 2 | 5 |
| 2 | 9 |

This stores separate records for each combination of student and class. Our student and class tables remain the same:

**Student**:

|  |  |
| --- | --- |
| **Student ID** | **Student name** |
| 1 | John |
| 2 | Debbie |

**Class**:

|  |  |
| --- | --- |
| **Class ID** | **Class name** |
| 1 | English |
| 2 | Maths |
| 3 | Spanish |
| 4 | Biology |
| 5 | Science |
| 6 | Programming |
| 7 | Law |
| 8 | Commerce |
| 9 | Physical Education |

Having our data structure in this way makes it easier to add more relationships between tables and to update our students and classes without impacting the relationships between them.