Class 1

Database

BY:

BRUNO ANGELO MEDEIROS
bruno.medeiros@sc.senai.br

FOREIGN KEY CONSTRAINT (FK)

To use a FOREIGN KEY you need first a PRIMARY KEY (PK).

A FOREIGN KEY in one table points to a PK in another table.

Look at the following two tables:

The “country” table:

id	name	abbreviation
1	Brazil	BR
2	Russia	RU
3	Argentina	AR

The “client” table:

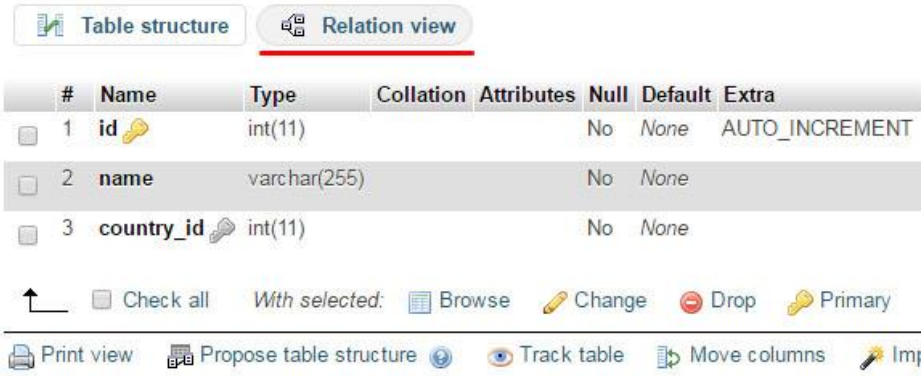
id	name	country_id
1	Bruno	1
2	Juan	3
3	Igor	2

The images above show the relationship between **country** and **client** table.

To create a FK in a table is already created, you will use the following SQL:

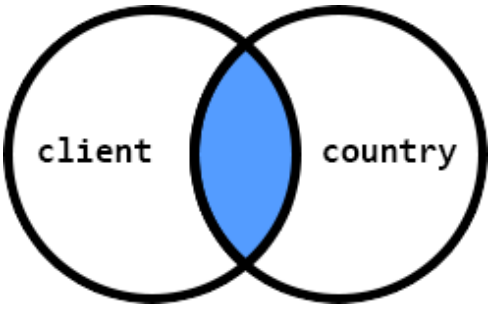
```
1. ALTER TABLE client
2. ADD FOREIGN KEY (country_id)
3. REFERENCES country(id)
```

In PHPmyAdmin you select the table, in our case **client**, and go in **Relation view** then in **Foreign key constraints,** select the id from the table **country** and then click on **save**.



JOINS

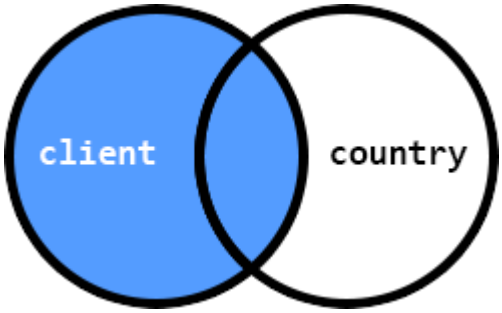
INNER JOIN



The most used clause is INNER JOIN. This join return a set of records which math in both the entities. For example:

```
1. SELECT client.name, country.name
2. FROM client
3. INNER JOIN country ON client.country_id = country.id;
```

LEFT JOIN



How do we do when we require a list of all user and their countries even if they're not enrolled in one? A LEFT JOIN produces a set of records if matches every entry in the left table (user) regardless of any matching entry in the right table (course).

```
1. SELECT client.name, country.name
2. FROM client
3. LEFT JOIN country ON client.country_id = country.id;
```

We could, for example, count the number of clients enrolled on each country:

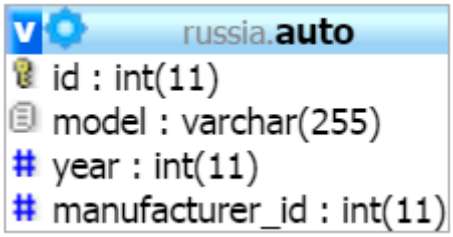
```
1. SELECT country.name, COUNT(client.name)
2. FROM country
3. LEFT JOIN client ON client.country_id = country.id
4. GROUP BY country.id;
```

RELATIONAL CONCEPTS

For this example, let's assume we're building a website that specialized in used-car sales.

We need to design a schema which will handle the searching and storage of a variety of different car models, and allow the customers to filter results based on a set of features they require in the vehicle.

The primary entity could be considered the **auto** table.



ONE-TO-MANY

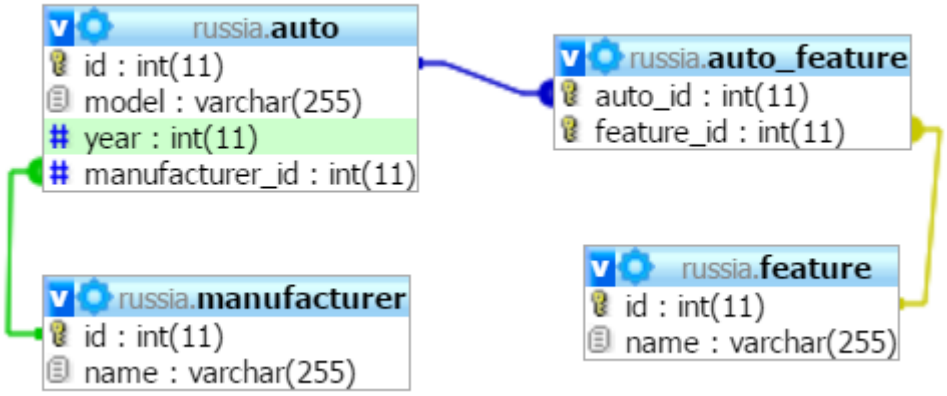
This is the most common type of relationship. Here, *one instance* of an entity can relate, or be attached to, *many instances* of another entity. In our schema the **auto** entity can have only *one* auto manufacturer. However, an auto manufacturer can produce many automobiles.

If we wanted to list the Auto's manufacturer name (not Manufacturer ID), we would use an INNER JOIN from the Auto table to the Manufacturer table along this key relationship, as shown below.

```
1. SELECT am.name
2. FROM auto a
3. INNER JOIN manufacturer am
4. ON a.manufacturer_id = am.id
5. WHERE a.id = 12;
```

MANY-TO-MANY

The type of relationship is between two entities when either entity may be associated with more than one instance of the other entity. For example, imagine the relationship between an Auto (as in car) entity and a Feature entity representing any of the many options a car may come with.



For that we need to use a third entity, which stores the **mapping** of the relationship between the two previous entities.

To select the content using the many-to-many relationship, we use like the following SQL:

```
1. SELECT f.name
2. FROM feature f
3. INNER JOIN auto_feature af
4. ON f.id = af. feature_id
5. WHERE af.auto_id = 7;
```

PATTERNS

We will use the LIKE operator and a SQL pattern to perform a pattern match rather than a literal comparison.

Patterns are strings that contain special characters. SQL pattern matching uses the LIKE and NOT LIKE operators rather than = and != to perform matching against a pattern string.

They may contain two special characters: _ matches any single character, and % matches any sequence of characters.

You can use these to create patterns that match a wide variety of values:

- Strings that begin with a particular substring:

```
1. SELECT name FROM country WHERE name LIKE 'Br%'
```

name
Brazil
Brunei

- Strings that end with a particular substring:

```
1. SELECT name FROM country WHERE name LIKE '%ia';
```

name
Russia
Australia

- Strings that contain a particular substring anywhere:

```
1. SELECT name FROM country WHERE name LIKE '%an%';
```

name
France
Finland

- Strings that contain a substring at a specific position (the pattern matches only if **a** occurs at the third position of the name column):

```
1. SELECT name FROM country WHERE name LIKE '__a%';
```

name
Brazil
France

To reverse the sense of a pattern match, use NOT LIKE.

The following query finds strings that contain no **i** characters.

```
1. SELECT name FROM country WHERE name NOT LIKE '%i%';
```

name
France