Entity Relationships

Different tables in our databases represent different entities. Customers, departments, employees, products, invoices and so forth. When we design our database we need to know how these entities relate to each other.

Sometimes the term cardinality is used to describe the different types of relationships that can occur between these entities of tables. So cardinality refers to how the occurrences in one entity are linked to the number occurrences in another entity.

There are 3 types of relationships. One-to-one, one-to-many, and many-to-many.

On an Entity Relationship Diagram, that uses the crow's feet notation, you may see the relationships represented by these symbols; one relationship, a many relationship, a one and only one, zero or one, one or many, and zero or many. We will see how these work between entities here in a minute.

Let's look at the one-to-one relationship. Say we have an entity representing an employee.

And we have another entity representing a company car.

We know that each employee is allocated a company car.

And each company car is assigned to one employee.

If we look at the crows feet notation representing this relationship we see the one and only one employee has a company car. And one and only one company car is assigned to one employee.

Perhaps we have some employees who don't need a company car.

Therefore, we might see something like this where an employee might have a car allocated to them but some employees might not have any car at all. But the left side of the notation remains because if they do have a car assigned to them they will have one and only one car. And we are also assuming here that, if there is a company car, it will be assigned to an employee.

So here we have another one-to-one example. This is a view of the ERD or Entity Relationship Diagram. Each library item can either be a book, a video, or a magazine. So we have a one-to-one relationship between library_item and video, also between library_item and book and between library_item and magazine. Take a look at how the primary key and foreign keys are related here. The primary key of library_item is a foreign key of the other tables. So each library_item will belong to one and only one of the three categories. It will be a video, a book or a magazine.

Now if we look at structure of the entities as they would look in a database with data entered into the tables we can see that same primary and foreign key structure at work. The video table is left off but we can see that each primary key in the library_items table is related to a foreign key in the books, magazine or video tables. The foreign key won't be repeated in those tables because there will only be one book or one magazine or one video for each library item. And visa versa each library item is either a book, magazine or video, just one of those.

Here's another example of why you might want a one-to-one relationship. A one-to-one relationship is not as common as the other two relationships we will talk about but there are certain situations where you will use it. Say you have a employee table that anyone in the company can have access to but there are some attributes of the employee you want kept private like their salary or the social security number or their home address. For security and privacy reasons you might want that information stored in a separate table of the database. There would be a one-to-one relationship here because each employee would have one and only one private information about them. And the private information about each employee belongs to one and only one employee.

Let's look at the one-to-many relationship.

A customer,

and orders.

A customer makes orders.

An order is made by a customer. For example my one customer account with Amazon has many orders that I've made in the past.

So with the crows feet notation we can see that we have a one-to-many relationship here. A customer can make one to many orders. And an order is made by one and only one customer. Now it could be argued that one order could be purchased by many customers but if you think about a transaction, usually it boils down to one customer who uses their one account to order, regardless of how many people use what was ordered in the end. So for our purposes here we are assuming one customer and only one customer makes an order. But over time, that one customer can make many purchases. But each order belongs to one customer. This is a one-to-many relationship.

Here is an example of two entities the departments of a company and the employees who work for that company. Each department has employees and each employee belongs to a department within that company. Here we see that a department has to have at least one employee but can have many employees. And each employee belongs to one and only one department.

Here are the same entities in the database with data for each entity. We have two departments representing the Accounting department and the IT department. The primary keys here are abbreviated characters for each department. But in the employees table you can see that many people can belong to that one department. Robert and Letty both belong to the accounting department. The primary key is related to the foreign key of each employee. That foreign key value can be repeated many times in the table for each employee that belongs to that department.

Lastly, we have the many-to-many relationship between entities.

Say we have a student entity

and a class entity.

There is a many-to-many relationship here.

For example here at this university students can take classes and

classes can have students in them. In our example here, a class can have one or more students in the class and students can take one or more classes.

Here is another many-to-many example. A person can subscribe to many magazines and magazines can have many subscribers.

Relational database systems don't allow you to implement a direct many-to-many relationship. A many-to-many relationship always has to be resolved or broken down into 2 one-to-many relationships. This provides a much better data structure for the database. It is simple to do and usually makes a lot of sense. This table that is created is referred to as a linking table, or a joining or a bridge table. But here we see that is makes a logical table called subscriptions. One subscriber can have many subscriptions. Each subscription belongs to one subscriber. One magazine can many subscriptions. Each subscription belongs to one magazine. An easy way to remember how to resolve a many-to-many in the ERD is to have the linking table with the many part of the relationship. And therefore the linking table will contain the foreign keys.

Here with the table data,

we can see that Sally Smith has two subscriptions. Her primary key of 234 is listed twice in the subscription table. Her two subscriptions belong to the Ensign and the Friend.

In the other relationship of magazines to Subscriptions we see that one magazine for example the Ensign can have many subscriptions. So Sally and Mark Miller both show a subscription to the Ensign here.

The linking table has the two foreign keys. In this case the two foreign keys could make up the composite primary key of the linking table. They are both needed together to uniquely identify each row of the table. But we do have a primary key column called subscription_id

So back to our university example. The students and classes many-to-many relationship

can be resolved in a logical linking table of enrollment. One student can enroll in many times and each class has many students enrolled in it.

So there we have the three entity relationships that you will work with while designing your database. one-to-one, one-to-many and many-to-many.