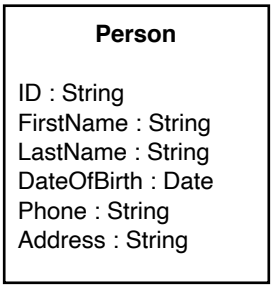
Designing Databases

Class Notes

Databases can get complicated and need to be planned and documented. A database with a single table, however, is very straight forward. You simply name the table and list every field within it, like so…

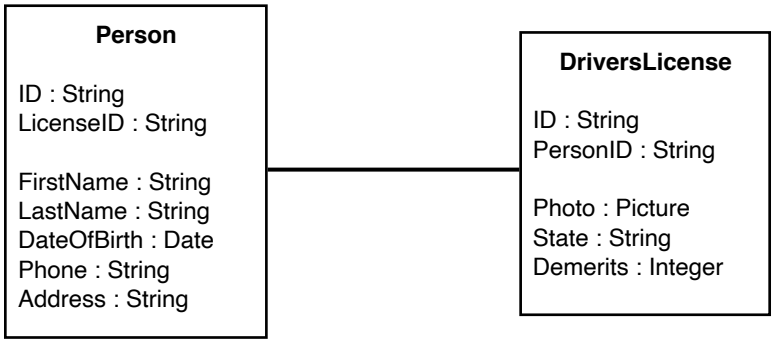


Relational Databases

Databases get more complex when there is more than one table. In this case, tables have relationships to each other, which can be one-to-one, one-to-many or many-to-many.

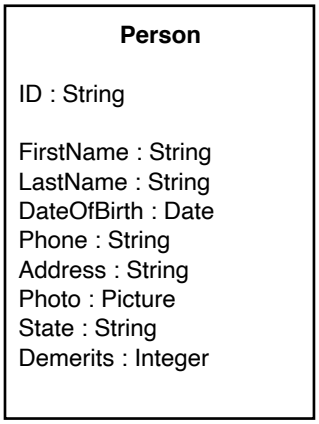
One-to-One Relationship

For example, if you have a table of people and a table of driver's licenses belonging to those people, that would be a one-to-one relationship. Each person has one driver's license and each driver's license belongs to one person. We depict this like so…



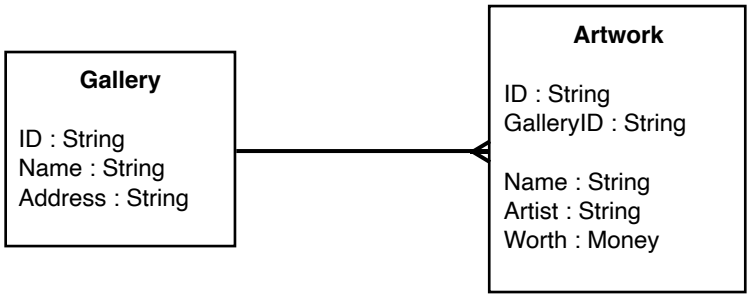
Note that we put the primary key first (the ID in both cases). Underneath that, we have what is known as a foreign key. A foreign key is a key into a different table and is the basis of the relationship between them. The Person table has a LicenseID field that allows you to search the DriversLicense table for the license belonging to that person. Similarly, the DriversLicense table has a PersonID so you can find the detail of the person if you have only their license details.

If you have a one-to-one relationship between tables, you should think about merging them. This is not always appropriate but is worth at least considering. In this case, it would give us the following…



One-to-Many Relationship

An art gallery's relationship to the artwork displayed there would be a one-to-many relationship. Each gallery contains many pieces of art but each piece of art can only belong to one museum.

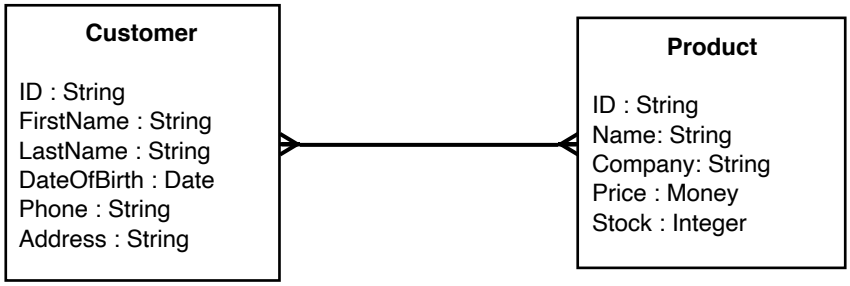


There are a few things to note here…

- The crow's foot on the right of the line indicates which side is "many".
- The artwork has a gallery ID to indicate where it's housed but the gallery does not have any artwork IDs. The reason is simple: How many artwork IDs do you need? A hundred? Two hundred? We can't make a thousand artwork ID fields in the gallery table to make sure we have enough room - and it's unnecessary. We can just look in the artwork table to get a count of everything at a particular gallery.
- We are not necessarily using datatypes from SQL (the above example has a "money" type). We are not designing an SQL database here but rather just a database - one that might be implemented in SQL but could also be implemented in something else. In designing the tables, we use human-understandable terms. So, "picture" or "music track" instead of "blob" and so on.

Many-to-Many Relationship

The third type of relationship is many-to-many. A customer table would have a many-to-many relationship with a products table. So, each customer can buy many products and each product can be bought by many customers.

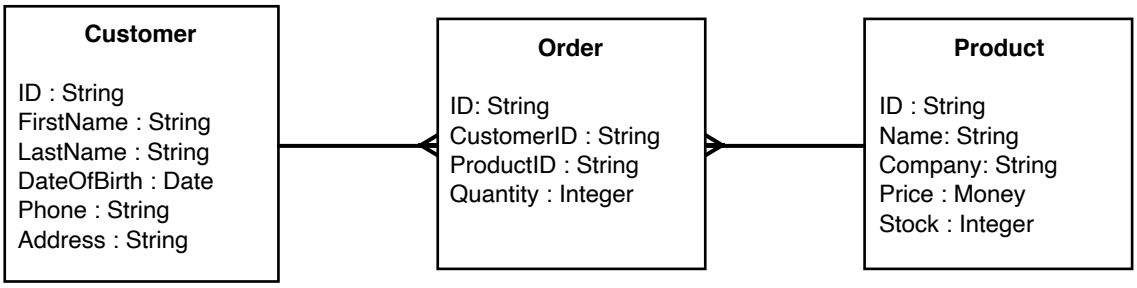


Note that neither database has a foreign key to the other. This is because many-to-many relationships cannot be modelled by databases. After all, if a customer can order many products, then the customer table needs fields for multiple product IDs - but how many? And if a product can be ordered by many customers, then the product table needs fields for multiple customer IDs - and, again how many?

If this was a programming language, we could add a dynamic array of foreign keys in each table, but this is not supported in databases for speed reasons. Remember that a database needs to potentially hold, search and sort millions of records so speed is of paramount importance. We need another solution.

Join Tables

To make a functional many-to-many relationship in a database, we need to add a table in between the two we already have. This is called a join table, bridging table or junction table. Using the previous example of the customer and the products, adding a join table would give us this…



We now have a customer that can make many orders, but each order only belongs to one customer. Similarly, each order is only for one product but each product can be in multiple orders.

For example, a customer named Jane with ID 12345 comes into the shop and buys some toilet rolls (ID 123), canned soup (ID 234) and lemons (ID 456). That would create three records for one shop visit in the order table, like so…

ID	CustomerID	ProductID	Quantity
1	12345	123	2
2	12345	234	3
3	12345	456	4

Note that sometimes the join table will have an obvious name - in this case, the thing between customers and products makes sense to be called “orders” - but sometimes there is no handy word and the join table is just called something like “Customer-Product”.