# In the context of relational databases, what are relationships? Describe at least two, and provide an example of their use.

In relational databases, relationships are what tie two or more tables together and allow for the separation of duties. There are three relationship types: one-to-one (OTO), one-to-many (OTM), and many-to-many (MTM).

## One-To-One (OTO):

A one-to-one relationship is one that I do not often come across. Generally, if you have an OTO relationship, you may need to combine those tables into a single table. The one time I have seen this actually be used in real life has been when the developers hit the limit for how many columns can be put into a single table. So they created a second table and made it an OTO relationship between them. That was a mess of a system.

## One-To-Many (OTM):

One-to-Many is the most common relationship I have come across when working with databases. As the name implies, you have one row in a table that can relate to multiple rows in a different table(s).

### Example:

An example of this is a Shop Management System that keeps track of Repair Order History for customers. A Repair Order will have a customer and generally a vehicle associated with each order. Another way to look at this relationship, though, is that a customer can have multiple repair orders associated with them. However, you will typically only have a single customer related to your repair order.

## Many-To-Many (MTM):

I have seen this relationship a couple of times. This relationship is for when you need to be able to associate a single row in a table (Let's say table A) to multiple rows in another table (Let's call this one table B). Having it just like that makes it a One-To-Many relationship. To make it a Many-To-Many relationship, you will also need to have a reason to have a single row in table B be associated with multiple rows in table A.

### Example:

For example, if you are a parts company. It would be nice only to get a particular part from a single vendor. Sometimes that vendor can be out of stock. So you will, many times, have to order the same part but from different vendors. Ultimately, you will end up with a vendor table and a vendor parts table. The vendor parts table will have a vendor ID column that allows you to relate to the vendor and/or vendors that provided the parts you have in stock. Generally, to avoid duplication of data. You normally end up with an "in-between" table that provides the relationship between the tables. So, you will end up with a Vendors, VendorParts, and VendorPartsMapping tables.

# What are the advantages of relational databases? What are the benefits of NoSQL databases?

Relational databases are great for structured data patterns and data integrity. With a Relational database, you can inherently trust that if you have a One-To-Many relationship. That you will be able to find the "One" portion of the relationship. Relational databases start breaking down, however, once you get vast amounts of data within them.

NoSQL databases are great for data that is not structured, and this allows for the database to be highly scalable. As the data requirements grow, the database will function as needed. NoSQL databases work faster and more efficiently than relational databases. The issue is that your data integrity may not exist, depending on how well you coded the integrity of data inputs in your system.

# What are the disadvantages of relational databases? What are the disadvantages of NoSQL databases?

Relational databases are very rigid and harder to scale compared to NoSQL databases. This makes the cost to make changes to the structure higher and more error-prone.

NoSQL databases, on the other hand, have the exact opposite issue. At the same time, they can change the structure easily. Unless you are very disciplined with how you insert and update data. You can very easily lose track of data relations.

# Identify at least two features of MySQL and two features of MongoDB, and describe what they are and how they are used.

## MySQL features:

* MySQL allows for ACID (Atomicity, Consistency, Isolation, Durability) transactions.
  + This basically means that you can do multiple updates and/or get to the database, and if any fail. Then all fail, and nothing is actually committed to the database. The most common example given is bank transactions. You need to both debit one account while crediting another. If one fails, you really want the other to "Roll Back" any changes so you don't end up with issues.
* SQL (Structured Query Language) compatibility
  + This language is generally the same across many relational databases. You can expect to be able to use "Select", "Join", "Insert", "Delete", and "Update" statements and be able to almost exactly copy the same query from MySQL to MSSQL to Azure SQL, etc.

## MongoDB features:

* Flexible Schema
  + It is easier to describe an example than to define. Essentially, you can have just a single Schema type of TransactionLines. You can then have different transaction line types that are wildly different from each other. For instance, you could have a transaction line type that is for parts bought and used. At the same time, you have a separate Transaction line type for Labor that was done to install the parts. Both call for very different properties. Parts generally will have a quantity, PartID/Number, Description, and maybe what vendor the part came from. Labor will generally have links to the parts used, the number of hours worked, the team member who actually did the work, and sometimes, you may even sublet out the labor and need to be able to note that as well.
* The ability to index and query based on just about anything.
  + While relational databases will have issues when you start having many indexes assigned to a table, MongoDB was practically made for that. You can even nest relationships within the data and look that data up without special coding. These relationships are not rock solid, meaning that if you find the relationship too hard to manage. Then you can change your code to stop, and it won't cause the database issues. Note that doesn't stop your programs from having issues, though!