Relational vs Non-Relational Databases Example

To show the potential differences between these 2 different database methodologies, we will now look at a more practical scenario and see how each of the would handle and store the associated data. This will allow us to see some of the differences in how both of these systems are structured and how they manage the relationships between entities.

For this example, I will be focusing on NoSQL from the perspective of a document database such as Mongo DB.

The Scenario

Let's look at this using a fairly common and understandable scenario, a customer shopping cart system. Let's assume the system needs to allow for three main parts for it to work, the customer, the shopping cart and the inventory/product items that can be added to the cart.

For this system we will require to following details to be stored in each part:

Customer Details

- Customer Name
- Address
    - Street Address
    - Suburb
    - State
    - Postcode
- Email
- Phone Number

Cart Details

- Cart/Invoice Number
- Finalised Date
- Total Price
- Payment Method
    - Type (Card, After Pay, PayPal etc)
    - Payment Confirmation
- Cart Items
    - Product Details
    - Product Quantity
    - Price

Products

- Product Name
- Description
- Price
- Stock Quantity
- Supplier Details

Supplier Details

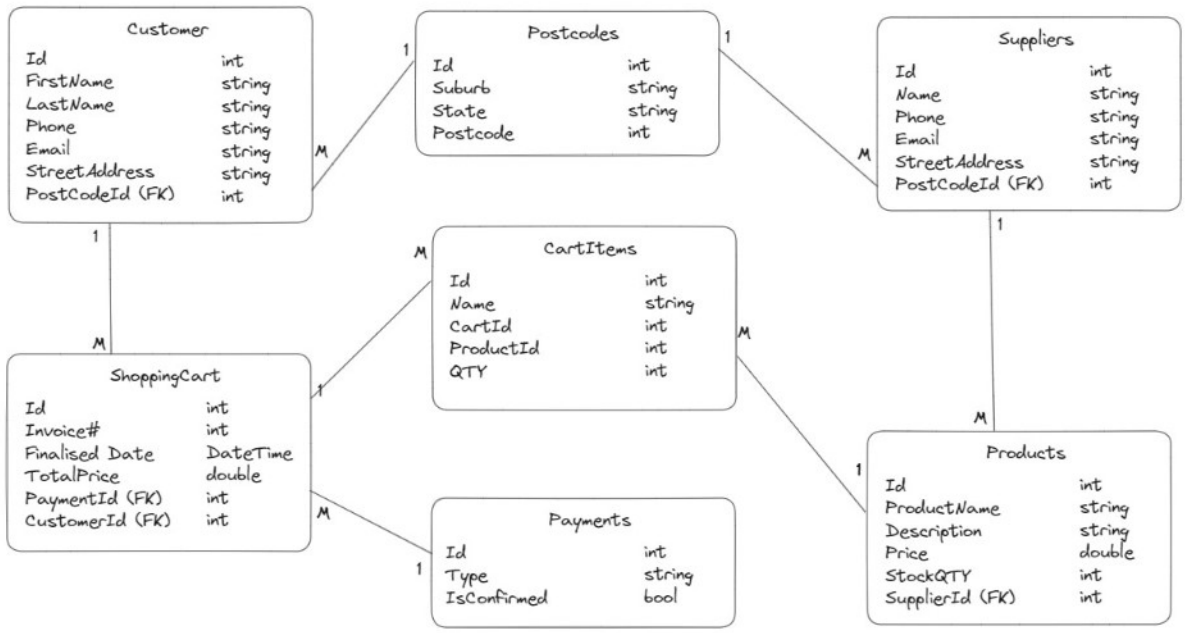
- Supplier Name
- Address
    - Street Address
    - Suburb
    - State
    - Postcode
- Email
- Phone Number

As you can see by the details we have specified, we have a few sections with sub details of their own. Additionally, the Cart Items may consist of multiple items that are in a single purchase.

Applying this in SQL (Relational Database)

If we were to do this in a traditional SQL database, we would probably start by normalising the data to separate the sub sections of the entities and remove any potential repetition of data. We would also likely need to create a linking table for the cart items because the carts and items would have a many-to-many relationship which is not allowed in traditional SQL databases.

The final result would likely be something like the following ER diagram:



As you can see, even though we really only have 3 entities in our scenario, once normalisation is finished, we now have 7 potential tables we need to build to setup this scenario in a SQL database. Not only that but we also have 7 foreign key connections we need to configure and setup for this system to work.

NoSQL Database (Mongo DB)

If we were to approach this setup in a NoSQL database, the first thing we can ignore is the need to normalise our data and create any linking tables. We will also keep the related entities such as the customer, cart and cart items in a single document. The only items we will pull into separate records are the products and suppliers, because that are standalone entities that will likely need to be accessed independently by other parts of the system.

With this in mind the first document for the customer and their purchaser history could potentially be structured something like this:

As you can see by this example, we can put a lot of associated values in a single entry. We can put all the customer's purchases and the full details of those purchases together by using array fields and object field types within the single document.

After this we would only need to create 2 more document types to represent the products and the supplier details.

These final 2 tables might look something like this:

Suppliers:

```
{
  "_id": {
    "$oid": "668b37d39236515bb3c5e86e"
  },
  "SupplierName": "GLOW Hair Appliances",
  "StreetAddress": "54 Mullet St",
  "Suburb": "Brisbane",
  "PostCode": "4301",
  "State": "QLD"
}
```

Products:

```
{
  "_id": {
    "$oid": "668b3d6a9236515bb3c5e871"
  },
  "ProductName": "Dyson Hair Dryer",
  "Description": "Innovatove hair dryer using dyson's patented engine technology.",
  "Price": 100,
  "StockQTY": 23,
  "Supplier": "GLOW Hair Appliances",
  "supplier_id": {
    "$oid": "668b37d39236515bb3c5e86e"
  }
}
```

As you can see with the 2nd example here, we can still add references to other documents in a similar way to how we would do it in SQL. But normally we would still put the relevant details of the referenced object in the object.

When we query the products, we would not retrieve the supplier details in the same query, which is why we put the supplier's name in the main record. We would only use the reference if for some reasons we needed to be able to request the supplier form the products page in our frontend which we would then process in a separate query.

Conclusion

As you can see, both of these systems handle the requirements and the data structure differently in these examples. Where SQL breaks everything into the smallest logical pieces and uses connections and references to remove any repetition of values, our NoSQL document store system prefers to keep all data that is likely top be queried together in the same place and only separates data that is non-directly related into separate documents.

The advantage to the NoSQL approach compared to traditional SQL systems is that you can request an entire customers purchase history in a single query and grab it all from one place. This makes the documents a little more complex structurally but removes the need to manage connections and relationships.

The SQL system on the other hand, requires some more work to separate and set up the different entities and their relationships but then gets the advantage of not having any repetition and allowing a change to a single field be represented every place it is used. It also allows for more variations in

queries and the option to grab subsets of data much more easily, which may not be possible in the NoSQL setup.