

**Index**

1. Technology agnostic
2. What are SQL Databases ?
3. What are NO SQL Databases ?
4. SQL VS NO SQL Databases
   1. Data model :
      1. Structure data model
      2. Semi structure data model
      3. Unstructured data model
   2. Database Language  
        
      4.2.1 Standard language   
        
      4.2.2 Complex query

4.3 Scalability

4.3.1 Vertical scalability

4.3.2 Horizontal scalability

4.4 Consistency and concurrency model  
  
 4.4.1 Strong consistency   
  
 4.4.2 Causal consistency   
  
 4.4.3 Eventual consistency   
  
 4.5 Theorem and principles   
  
 4.5.1 ACID   
  
 4.5.2 CAP

**Technology agnostic :**

**Wrong database selection :**

One of the major mistakes I see in the software industry is choosing the database based on the technology stack for example:

* If the developers team use .Net stack , the database will be automatically SQL Server.
* If the developers team use MEAN stack , the database will be automatically MongoDB.

**Good database selection :**

1. Different businesses need different databases , banking system needs database that different than social media app needs , you must choose the database that best suitable for your business domain.
2. Non-functional requirements is a very another important factor to choose your database , system of 1000 users needs database that different than system with 10,000,000 users.

**SQL Databases**

SQL databases also known as Relational Databases.

**What is SQL Database ?**

Database that store data on a tabular format   
( rows and columns )

**A close-up of a table

Description automatically generated**

**Is Excel sheets can be SQL Database because it store data on tables ( rows – columns ) ?**

A screenshot of a computer

Description automatically generated

Excel sheets is just a program to store data , **Not** any program or data store use tables, rows and columns will be SQL Database.

SQL Database has a set of standard conditions or properties that must be implemented and here list of the most important ( not all ) of them :

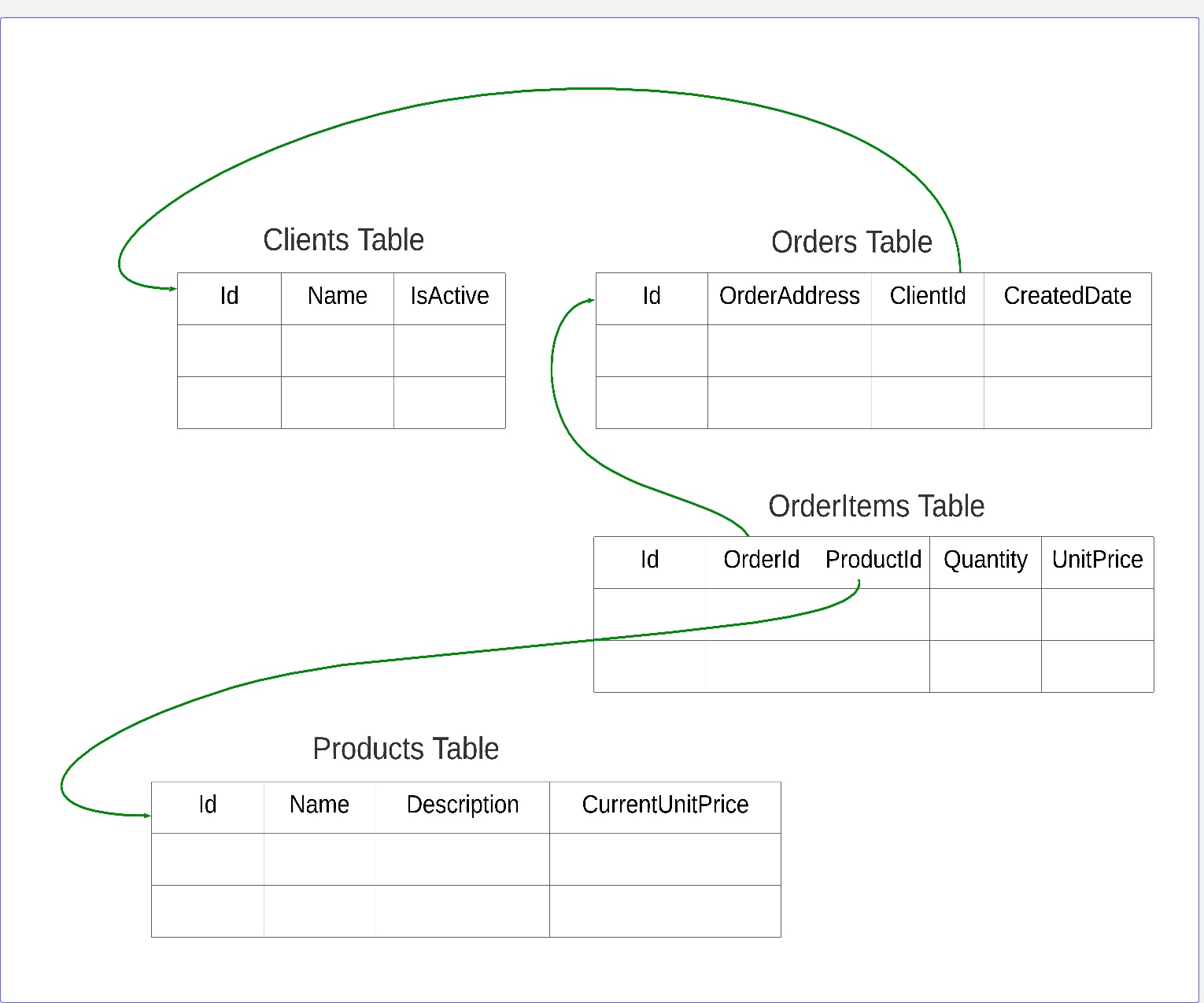
1. Relations.
2. Normalization.
3. ACID.

We will discuss them in more details in next section.

**SQL Database Properties :**

1. **Relations :**

The tables have relations between each other using foreign keys.



This relations help SQL database to achieve :

1. Data integrity for write operations ( insert – update – delete ) :   
   for example you can not delete client when he has orders and there reference to this client id in order table.

A close-up of a receipt

Description automatically generated

1. Join query for read operations :

Join query is one of the main properties on SQL databases.

using Join query : SQL Database can read and display data from different tables in single query.

A screenshot of a computer

Description automatically generated

1. **Normalization :**

Normalization in SQL Databases aims to prevent data duplications.

Normalization has a lot of forms , but we will discuss only the general concept.

In the previous section ( Relations ) we see our database consist of a set of different tables that have relations between each other , if we need data from more than one table , we join this tables.

The question now :   
why we design our database as a small tables as possible in development time and then joins them again in the runtime ?

It is better to design the database as single big table from scratch ?

The answer is summarized on two points :

1. For read operations :   
   Different queries need different data.

Example Query 1 : Get Order Details Of Id 8906

A screenshot of a computer

Description automatically generated

Example Query 2 : Get All Orders Of Client 5468

A screenshot of a computer

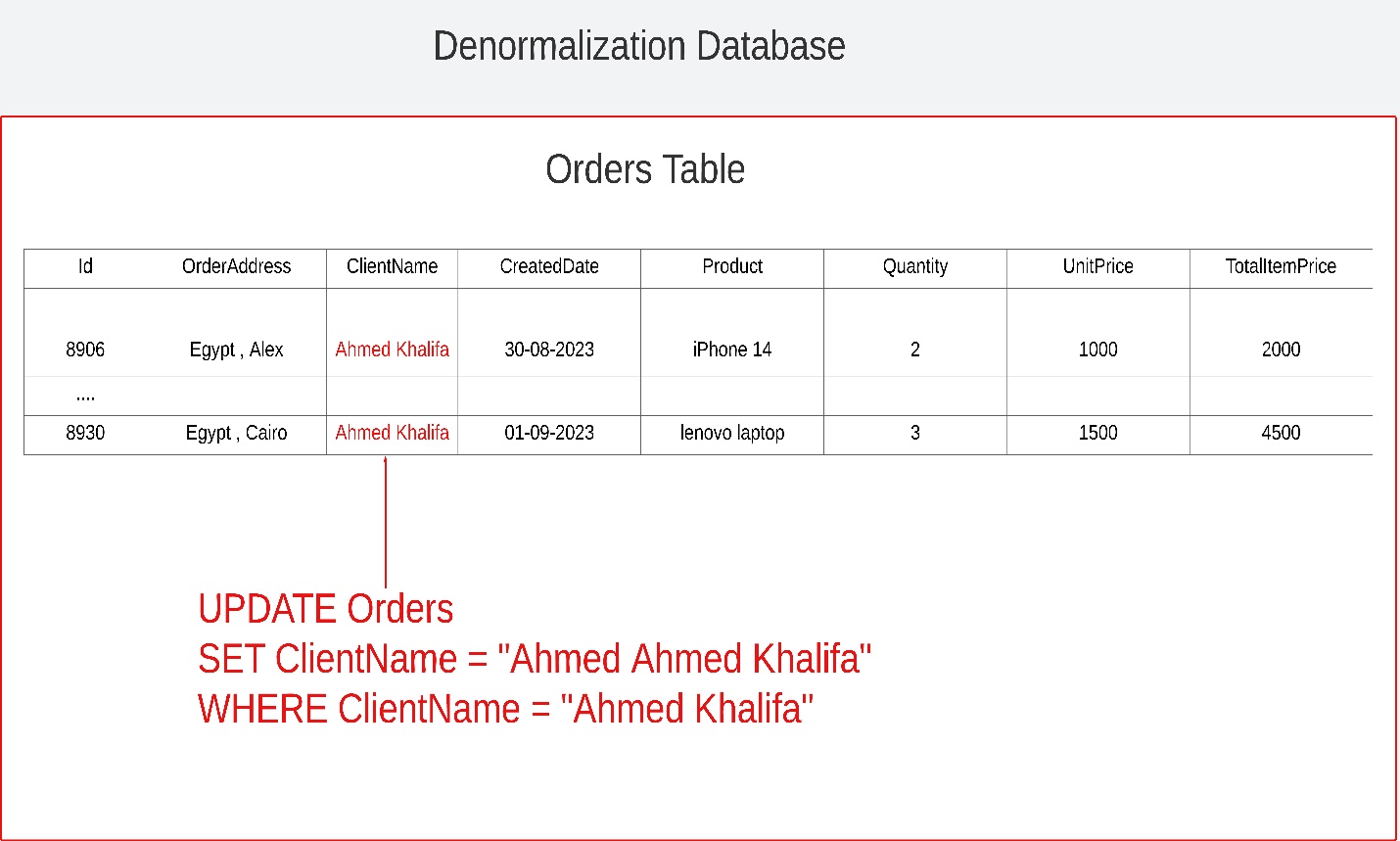
Description automatically generated

2- For write operations :

Prevent data duplication using normalization that can cause a lot of conflicts , performance issues , extra storge space for unnecessary data.

For example :   
imagine our e-commerce database consist of only single table called orders table contain every thing.

Our client want to update or change his name , in denormalization database , the update command need to loop on all rows to update our client name.



But when database apply normalization , we need only to update single row , and any join query with this table or row will reflect the new updated value immediately.

A close-up of a document

Description automatically generated

**Is normalization responsibility for database or developers ?**

Most of SQL Databases properties like ACID , Join query , data integrity are responsibility of database or RDBMS.

But normalization is responsibility o developers and their design.

**Is Denormalization always bad ?**

In some cases denormalization can be good but developers must take care of the effect sides of it.

Denormalization can good for :  
1- some business cases for historical data :  
if you note in our database design Products table has column called CurrentUnitPrice and OrderItems Table has column called UnitPrice , this because product price can change on the future , the old orders with this product must keep the old price.

2- solve performance issues when your query has a depth or a lot of joins , you can duplicate the column on one of the highest level table in join query and prevent join with that low level table , **this solution can be good when your duplicated column is rarely updated**.

**in general this solution must be used only in a very necessary cases and must not be common in your database.**