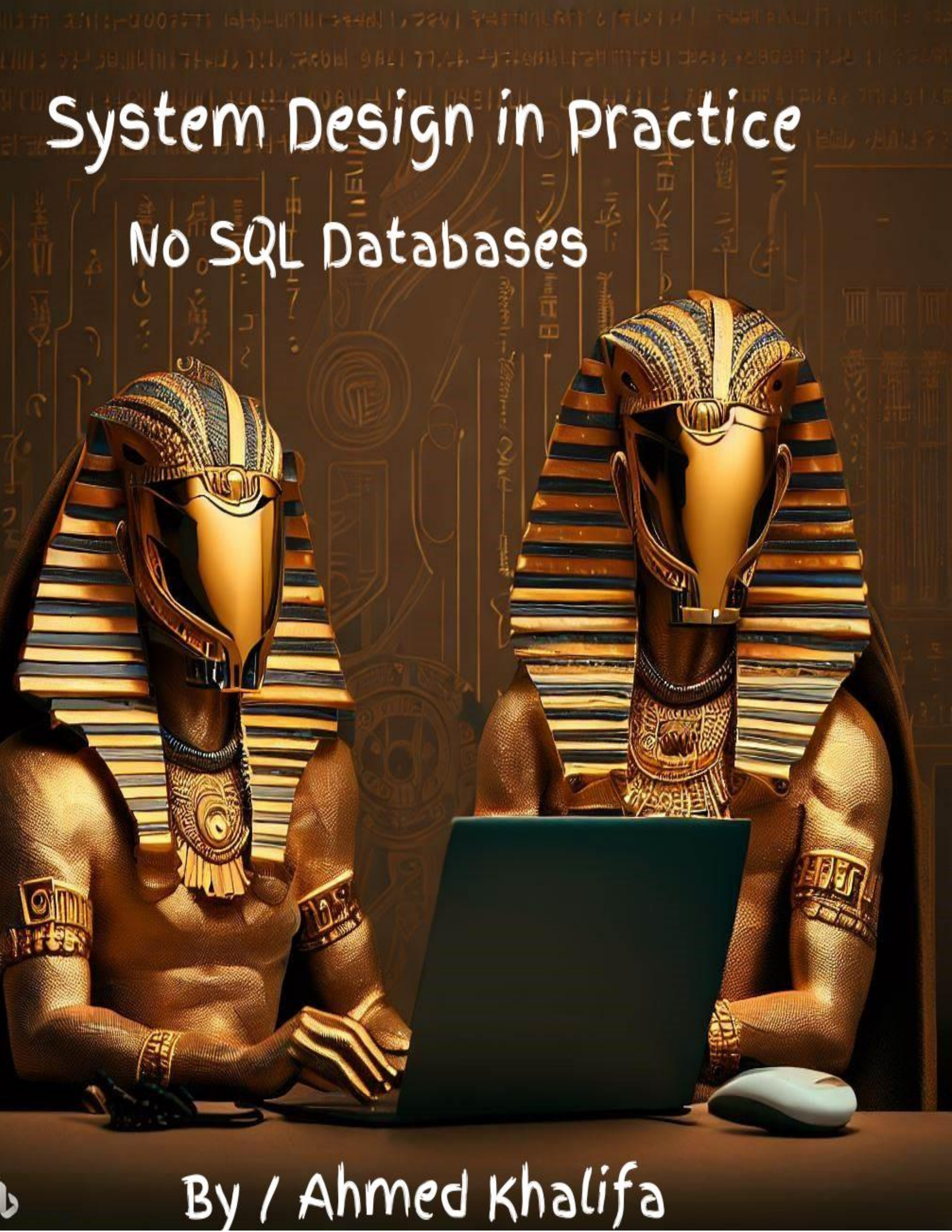
System Design in Practice

No SQL Databases



By / Ahmed Khalifa

Index

SECTION 3 : SQL VS NO SQL

3.1.Data model

- Data model types :

- How to choose best database suitable based on data model ?

3.2.Scalability

- What is Throughput , Latency , Scalability ?

- Scalability types

- How to choose best database suitable based on scalability?

3.3.Query Language and Operations

- Standard query language and operations

- Complex queries.

3.4.ACID VS CAP

- SQL Database With ACID for Sensitive Data.

- NO SQL Database With CAP for Big Data or Not- Sensitive Data.

SECTION 1
Technology agnostic :

1.1 Main reason to choose wrong database

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

1.2 How to choose the best database for your system ?

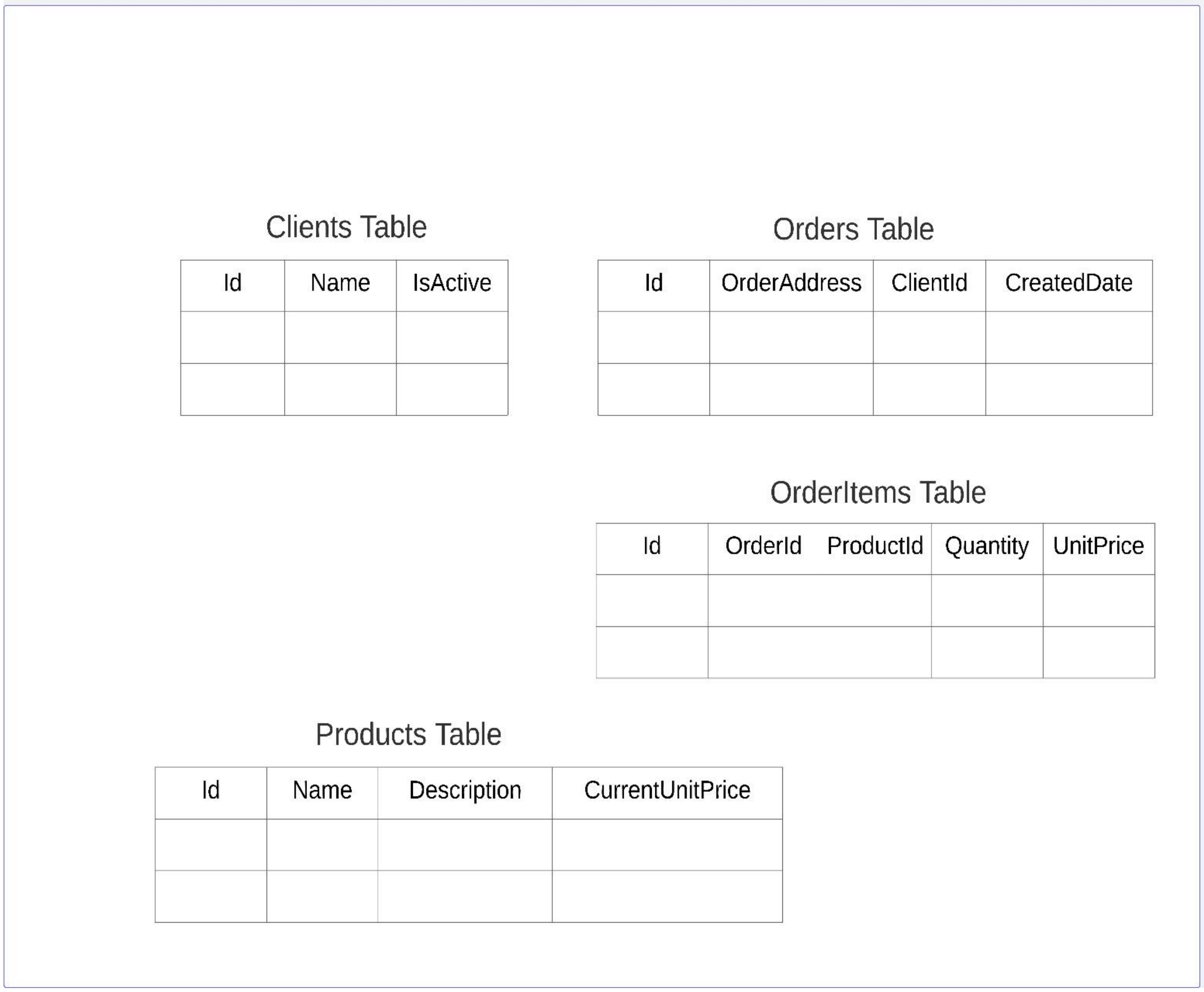
1. Different businesses need different databases, banking systems need database that different than social media app needs , you must choose the database that best suitable for your business domain.
2. Non-functional requirements are a very another important factor to choose your database, system of 1000 users' needs database that different than system with 10,000,000 users.
   we will discuss all non-functional requirements on details later.

SECTION 2
SQL Databases

SQL databases also known as Relational Databases.

2.1 What is SQL Database ?

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



Clients Table

Id	Name	IsActive

Orders Table

Id	OrderAddress	ClientId	CreatedDate

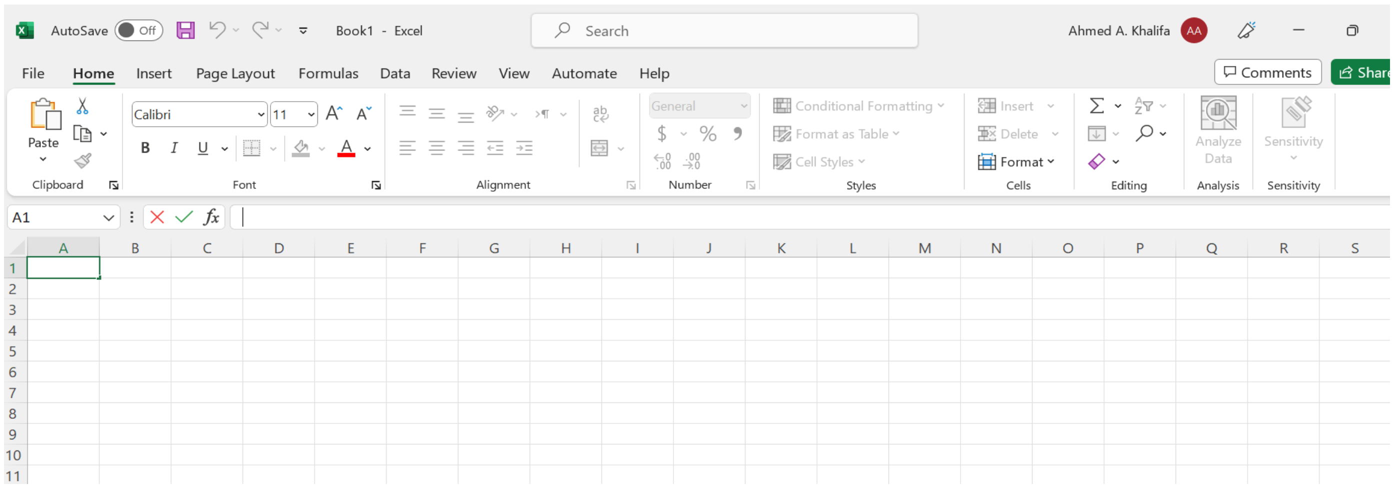
OrderItems Table

Id	OrderId ProductId	Quantity	UnitPrice

Products Table

Id	Name	Description	CurrentUnitPrice

2.2 Is Excel sheets can be SQL Database because it store data on tables ?



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.

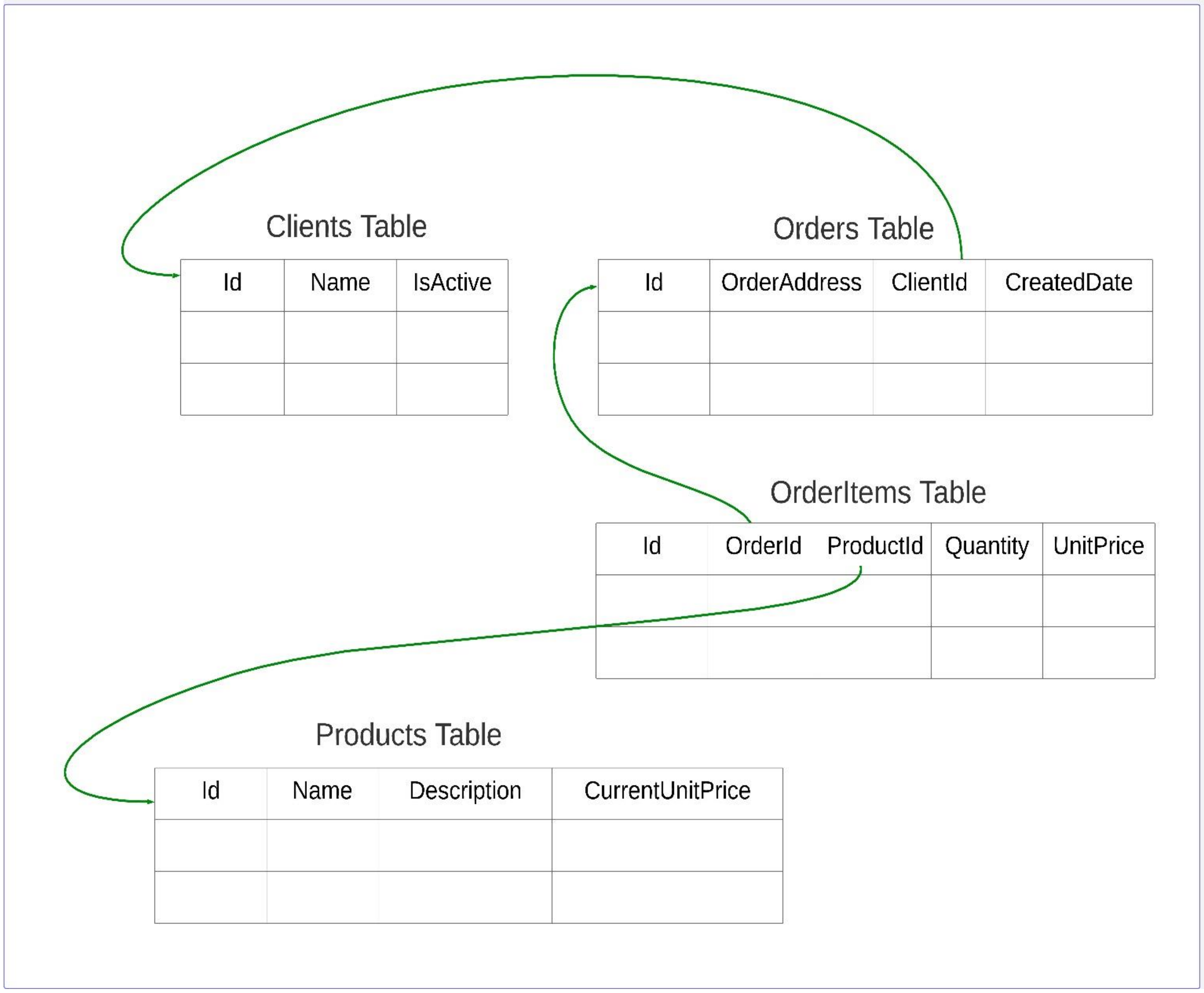
3- ACID.

We will discuss them in more details in next section.

2.3 SQL Database Properties :

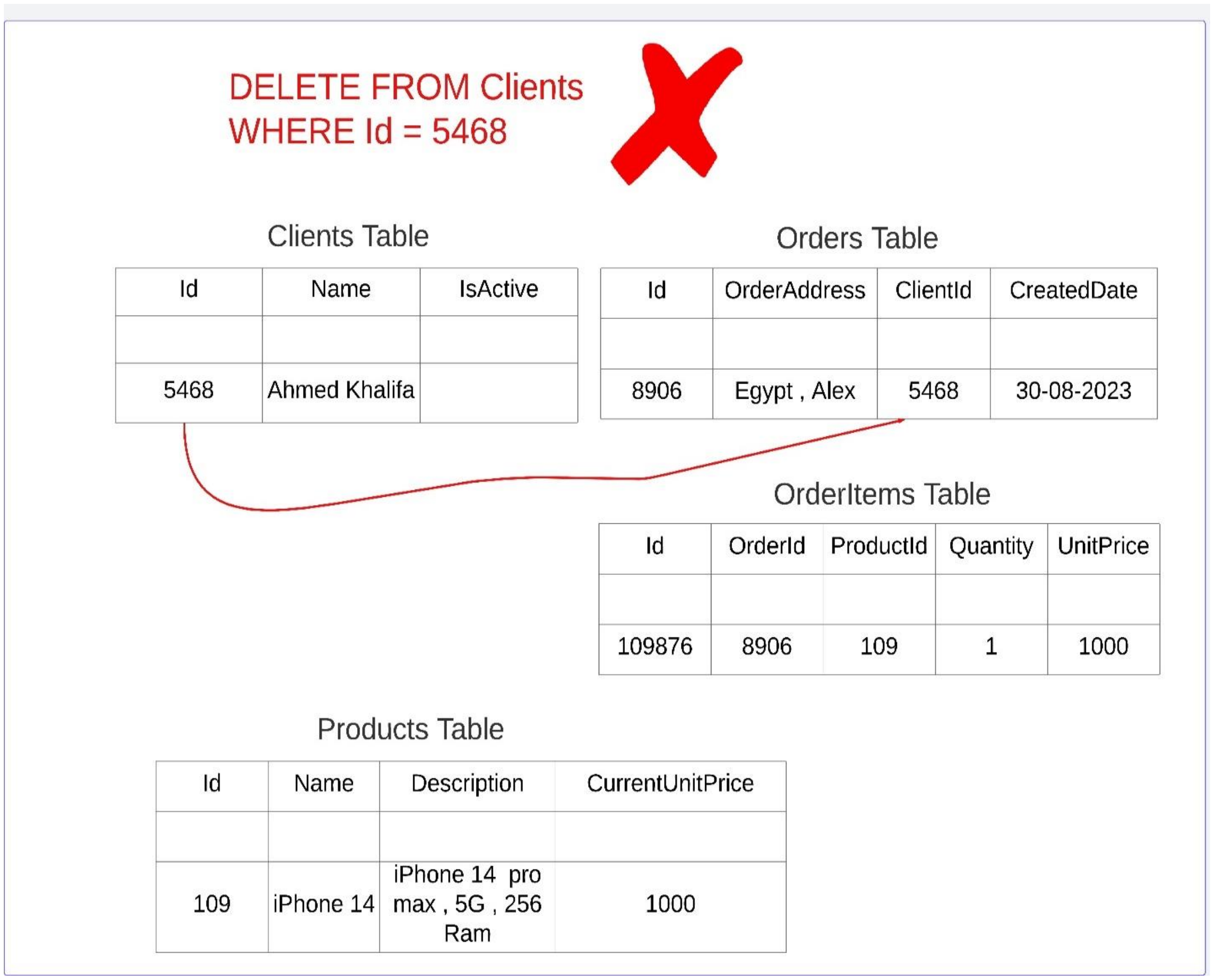
1.Relations :

The tables have relations between each other using foreign keys.



The importance of relations for write operations :

Relations achieve **data integrity** for write operations (insert – update – delete) :
example 1 : can you delete client from banking system who took a loan and did not repay it, or performed some financial transactions ?
example 2 : can you delete client has orders and there reference to this client id in order table ?



```sql
DELETE FROM Clients
WHERE Id = 5468
```

Clients Table

Id	Name	IsActive
5468	Ahmed Khalifa	

Orders Table

Id	OrderAddress	ClientId	CreatedDate
8906	Egypt , Alex	5468	30-08-2023

OrderItems Table

Id	OrderId	ProductId	Quantity	UnitPrice
109876	8906	109	1	1000

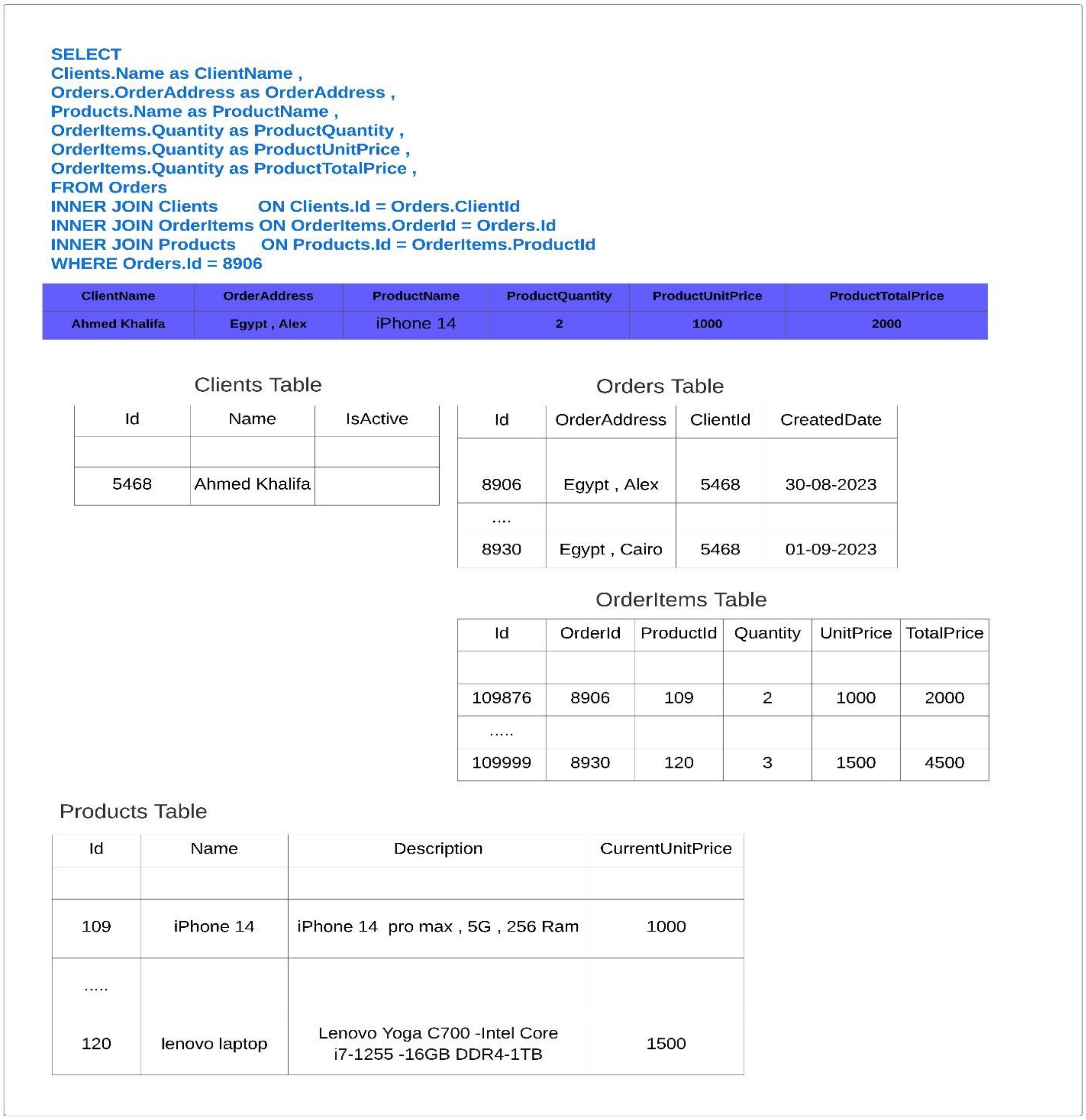
Products Table

Id	Name	Description	CurrentUnitPrice
109	iPhone 14	iPhone 14 pro max , 5G , 256 Ram	1000

The importance of relations 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.



```
SELECT
Clients.Name as ClientName ,
Orders.OrderAddress as OrderAddress ,
Products.Name as ProductName ,
OrderItems.Quantity as ProductQuantity ,
OrderItems.Quantity as ProductUnitPrice ,
OrderItems.Quantity as ProductTotalPrice ,
FROM Orders
INNER JOIN Clients      ON Clients.Id = Orders.ClientId
INNER JOIN OrderItems ON OrderItems.OrderId = Orders.Id
INNER JOIN Products     ON Products.Id = OrderItems.ProductId
WHERE Orders.Id = 8906
```

ClientName	OrderAddress	ProductName	ProductQuantity	ProductUnitPrice	ProductTotalPrice
Ahmed Khalifa	Egypt , Alex	iPhone 14	2	1000	2000

Clients Table

Id	Name	IsActive
5468	Ahmed Khalifa	

Orders Table

Id	OrderAddress	ClientId	CreatedDate
8906	Egypt , Alex	5468	30-08-2023
....			
8930	Egypt , Cairo	5468	01-09-2023

OrderItems Table

Id	OrderId	ProductId	Quantity	UnitPrice	TotalPrice
109876	8906	109	2	1000	2000
.....					
109999	8930	120	3	1500	4500

Products Table

Id	Name	Description	CurrentUnitPrice
109	iPhone 14	iPhone 14 pro max , 5G , 256 Ram	1000
..... 120	lenovo laptop	Lenovo Yoga C700 -Intel Core i7-1255 -16GB DDR4-1TB	1500

2.Normalization :

What is a normalization ?

Normalization in SQL Databases is a technique that aims to prevent data duplications and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies.

Normalization has a lot of forms , but we will discuss only the general concept.
Normalization as general concept is try to divide the database to small tables as possible and then this table will be logically communicate with each other through the relations or foreign keys.

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 importance of normalization for read operations :

Different queries need different data.

Example Query 1 : Get Order Details Of Id 8906

```
SELECT
Clients.Name as ClientName ,
Orders.OrderAddress as OrderAddress ,
Products.Name as ProductName ,
OrderItems.Quantity as ProductQuantity ,
OrderItems.Quantity as ProductUnitPrice ,
OrderItems.Quantity as ProductTotalPrice ,
FROM Orders
INNER JOIN Clients       ON Clients.Id = Orders.ClientId
INNER JOIN OrderItems ON OrderItems.OrderId = Orders.Id
INNER JOIN Products     ON Products.Id = OrderItems.ProductId
WHERE Orders.Id = 8906
```

ClientName	OrderAddress	ProductName	ProductQuantity	ProductUnitPrice	ProductTotalPrice
Ahmed Khalifa	Egypt , Alex	iPhone 14	2	1000	2000

Example Query 2 : Get All Orders Of Client 5468

```
SELECT
Clients.Name as ClientName ,
Orders.Id as OrderId ,
Orders.CreatedDate as OrderCreatedDate ,
FROM Orders
INNER JOIN Clients    ON Clients.Id = Orders.ClientId
WHERE Clients.Id = 5468
```

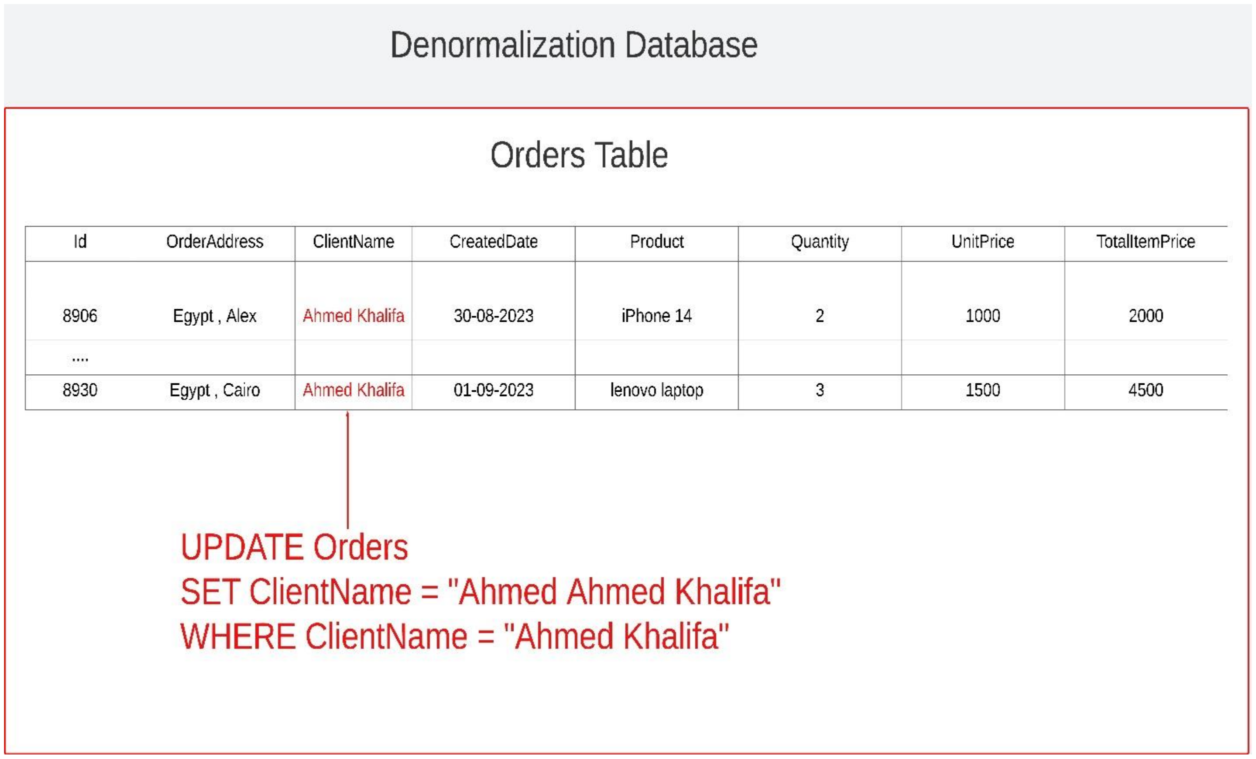
ClientName	OrderId	OrderCreatedDate
Ahmed Khalifa	8906	30-08-2023
Ahmed Khalifa	8930	01-09-2023

The importance of normalization 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.

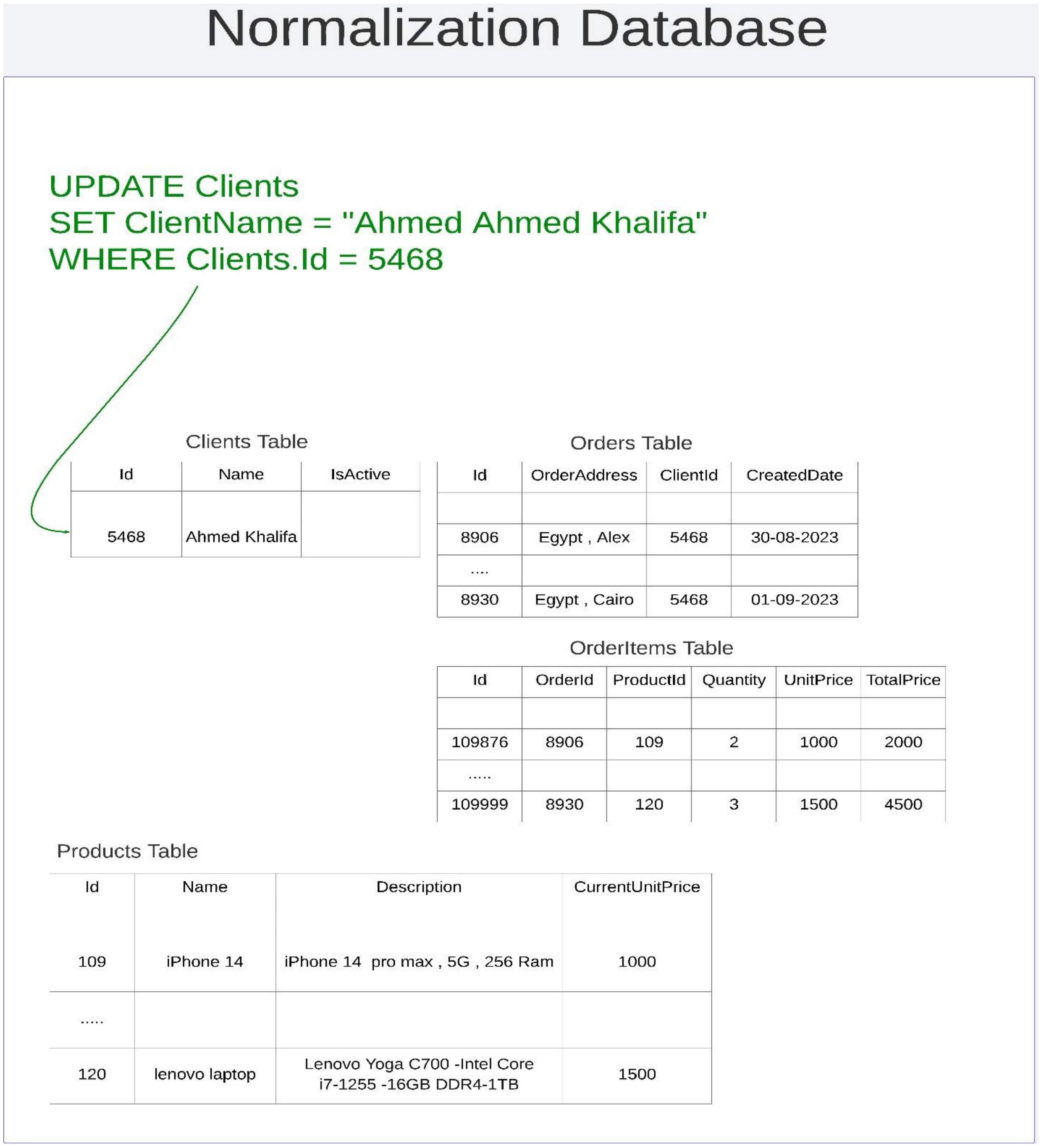
Our client want to update or change his name , in denormalization database , the update command need to loop on all rows to update our client name.



Id	OrderAddress	ClientName	CreatedDate	Product	Quantity	UnitPrice	TotalItemPrice
8906	Egypt , Alex	Ahmed Khalifa	30-08-2023	iPhone 14	2	1000	2000
....							
8930	Egypt , Cairo	Ahmed Khalifa	01-09-2023	lenovo laptop	3	1500	4500

```
UPDATE Orders
SET ClientName = "Ahmed Ahmed Khalifa"
WHERE ClientName = "Ahmed Khalifa"
```

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.



Normalization Database

```
UPDATE Clients
SET ClientName = "Ahmed Ahmed Khalifa"
WHERE Clients.Id = 5468
```

Clients Table

Id	Name	IsActive
5468	Ahmed Khalifa	

Orders Table

Id	OrderAddress	ClientId	CreatedDate
8906	Egypt , Alex	5468	30-08-2023
....			
8930	Egypt , Cairo	5468	01-09-2023

OrderItems Table

Id	OrderId	ProductId	Quantity	UnitPrice	TotalPrice
109876	8906	109	2	1000	2000
.....					
109999	8930	120	3	1500	4500

Products Table

Id	Name	Description	CurrentUnitPrice
109	iPhone 14	iPhone 14 pro max , 5G , 256 Ram	1000
.....			
120	lenovo laptop	Lenovo Yoga C700 -Intel Core i7-1255 -16GB DDR4-1TB	1500

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 of 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 :
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.

3.ACID Principles

What is ACID ?

4 Properties that define the behavior of the database transaction.

They are **Atomicity, Consistency, Isolation, Durability.**

A → (Atomicity)

What is Atomicity ?

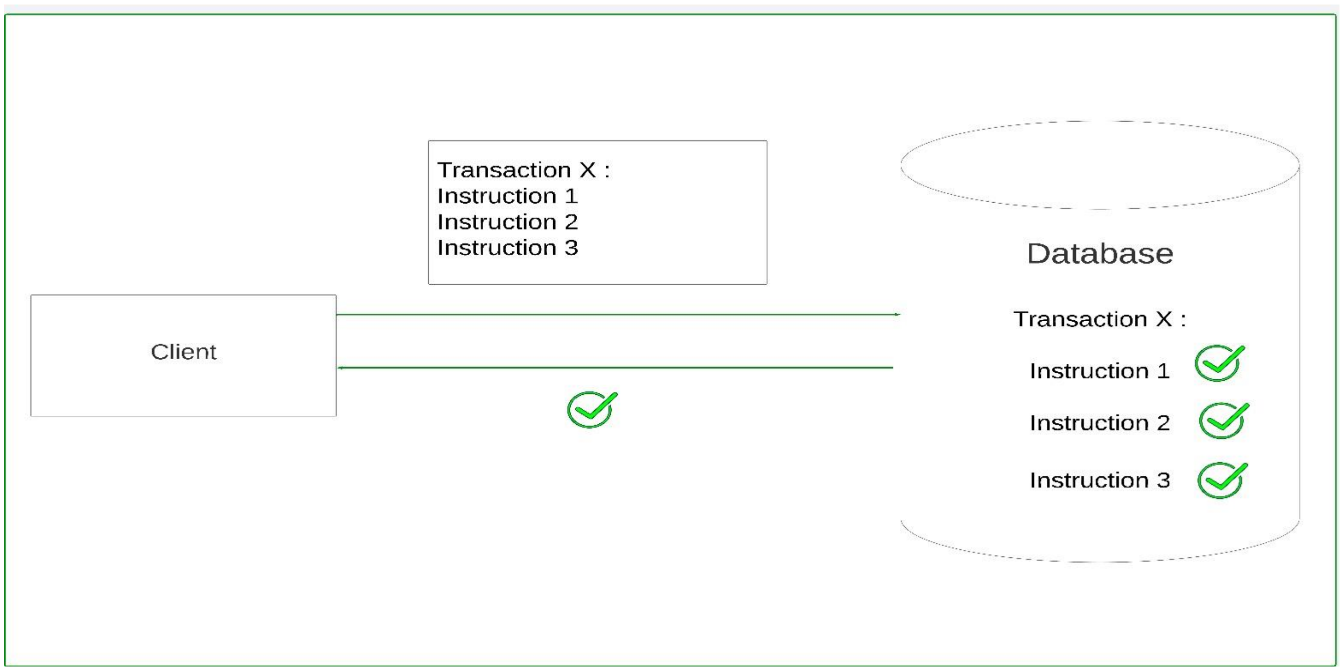
transaction work as a unit of work.

Atomicity cases :

Transaction has only two cases :

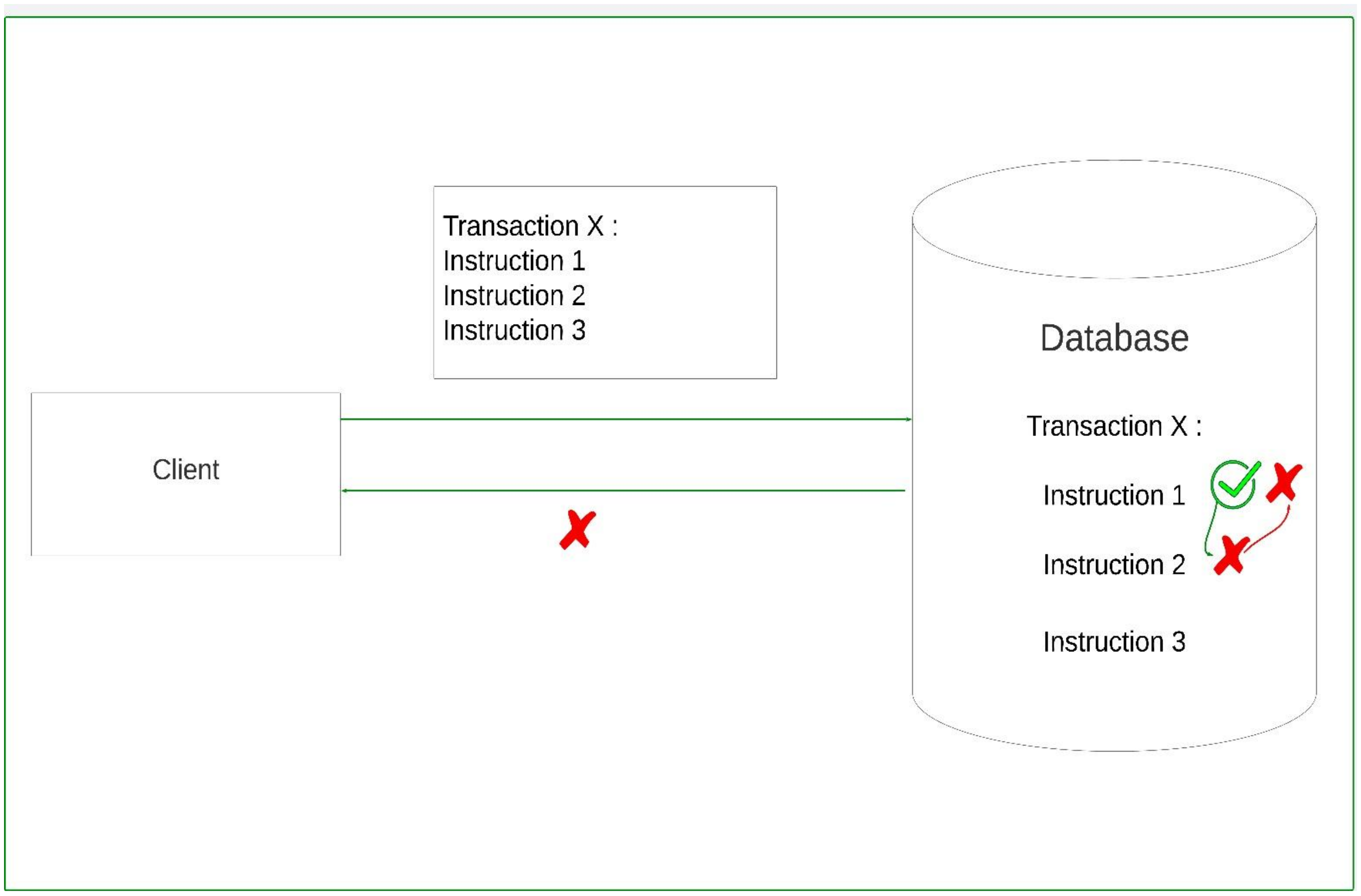
Case 1 if no failure :

All instructions or statements executed successfully.



<u>Case 2 If there are failure at any instruction :</u>

- all instructions or statements not executed.
- database should abort or rollback any instruction executed before this failure.
- then return errors to the client.

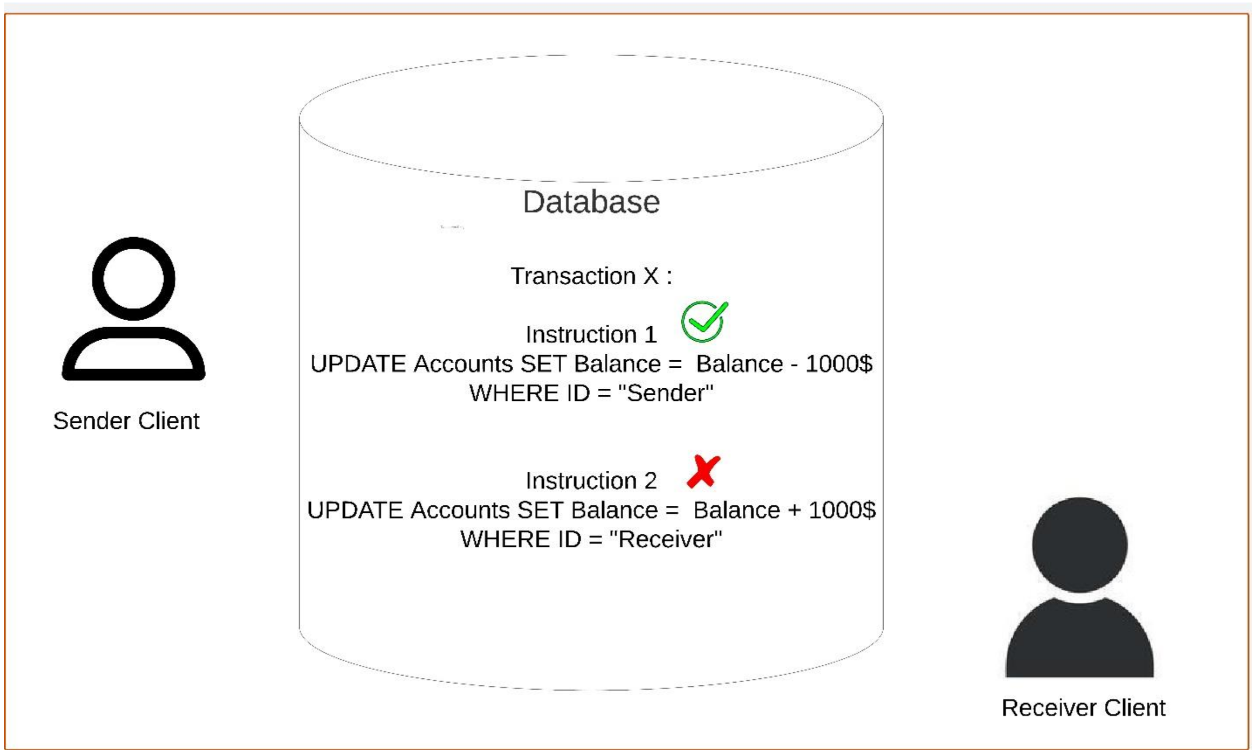


The importance of atomicity

To understand the importance of the atomicity we will discuss a very famous example :
transfer money between two bank accounts

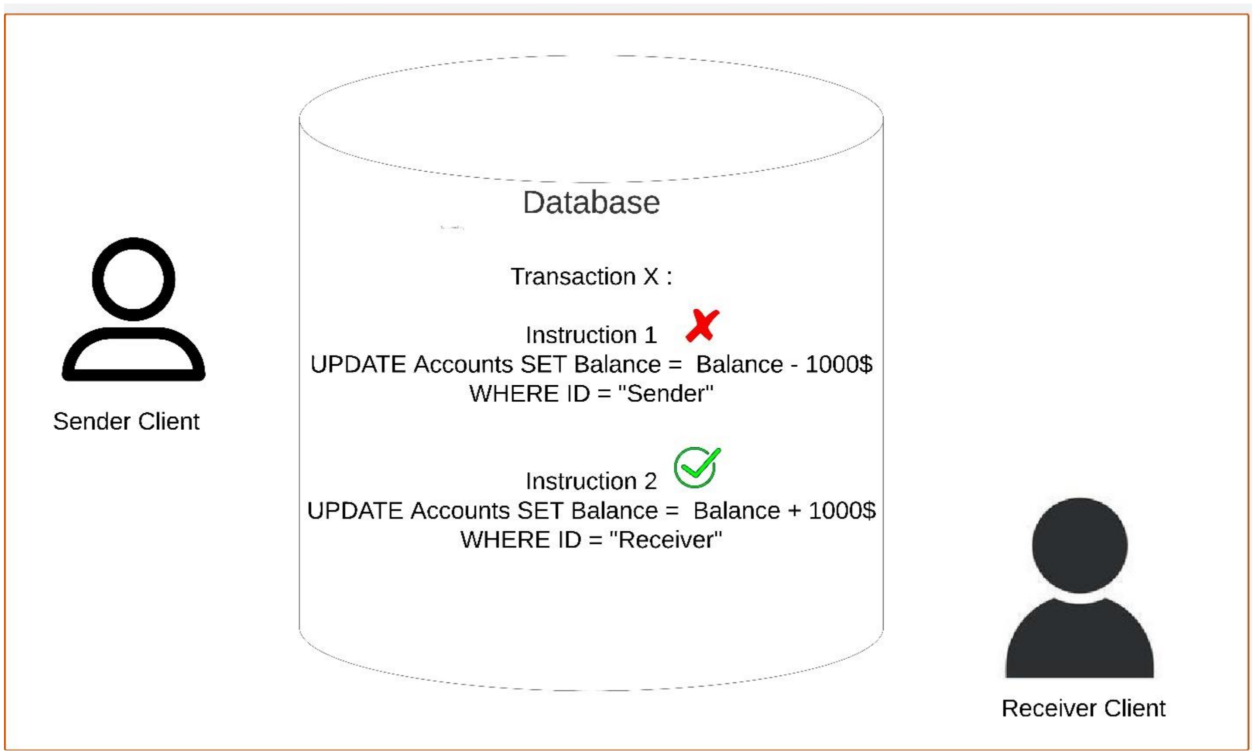
If the database does not support the atomicity :
Case 1 :

The transaction reduce the money from the client 1 account , and not sending it to the client 2 account , so client 1 loss his money.


Database
Transaction X :
Instruction 1 ✓
UPDATE Accounts SET Balance = Balance - 1000$ WHERE ID = "Sender"
Instruction 2 ✗
UPDATE Accounts SET Balance = Balance + 1000$ WHERE ID = "Receiver"
Sender Client
Receiver Client

Case 2 :

the transaction does not reduce the money from client 1 account , but send it to client 2 account , so the bank will lose its money on this case.



Transaction X :

Instruction 1 ✗
UPDATE Accounts SET Balance = Balance - 1000$ WHERE ID = "Sender"

Instruction 2 ✓
UPDATE Accounts SET Balance = Balance + 1000$ WHERE ID = "Receiver"

C → (Consistency)

Consistency in ACID Theorem has a different definition than Consistency on CAP Theorem or distributed systems , we will discuss consistency in CAP Theorem later in details.

What is Consistency in ACID ?

Transaction must move the database from valid state to another valid state.

In simplicity consistency mean all write operations or instructions in our transaction must apply all defined rules like : constraints , cascades , triggers , referential integrity and .. etc.

The importance of Consistency with example :

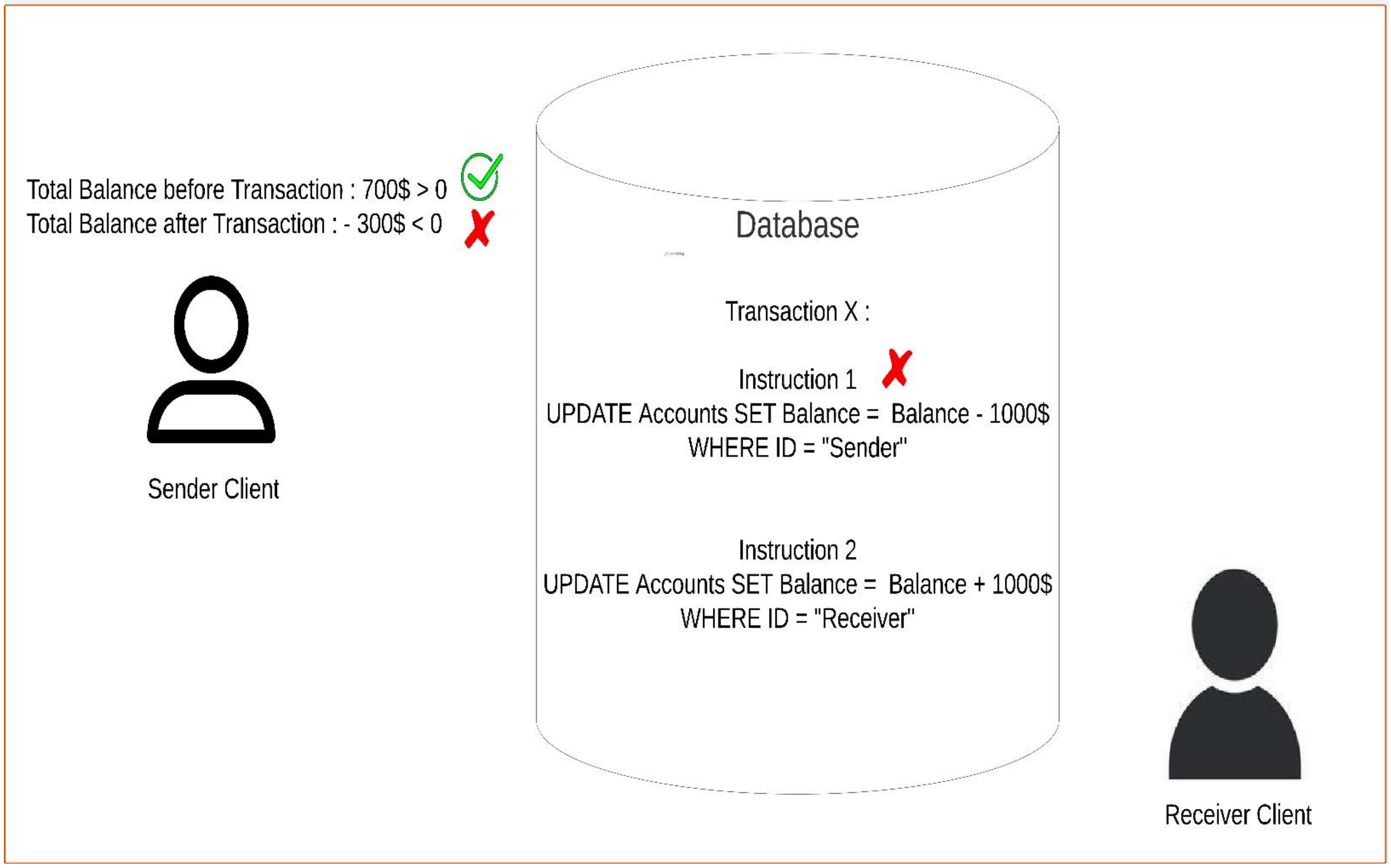
Negative balance bank account after transfer money

we have a defined rule or constraint on balance column in accounts table :

```
CREATE TABLE Accounts (
  Col 1 ,
  Col 2 ,
  Balance Decimal ,
  CONSTRAINT Balance_Positive CHECK (Balance >= 0)
);
```

Case 1 :

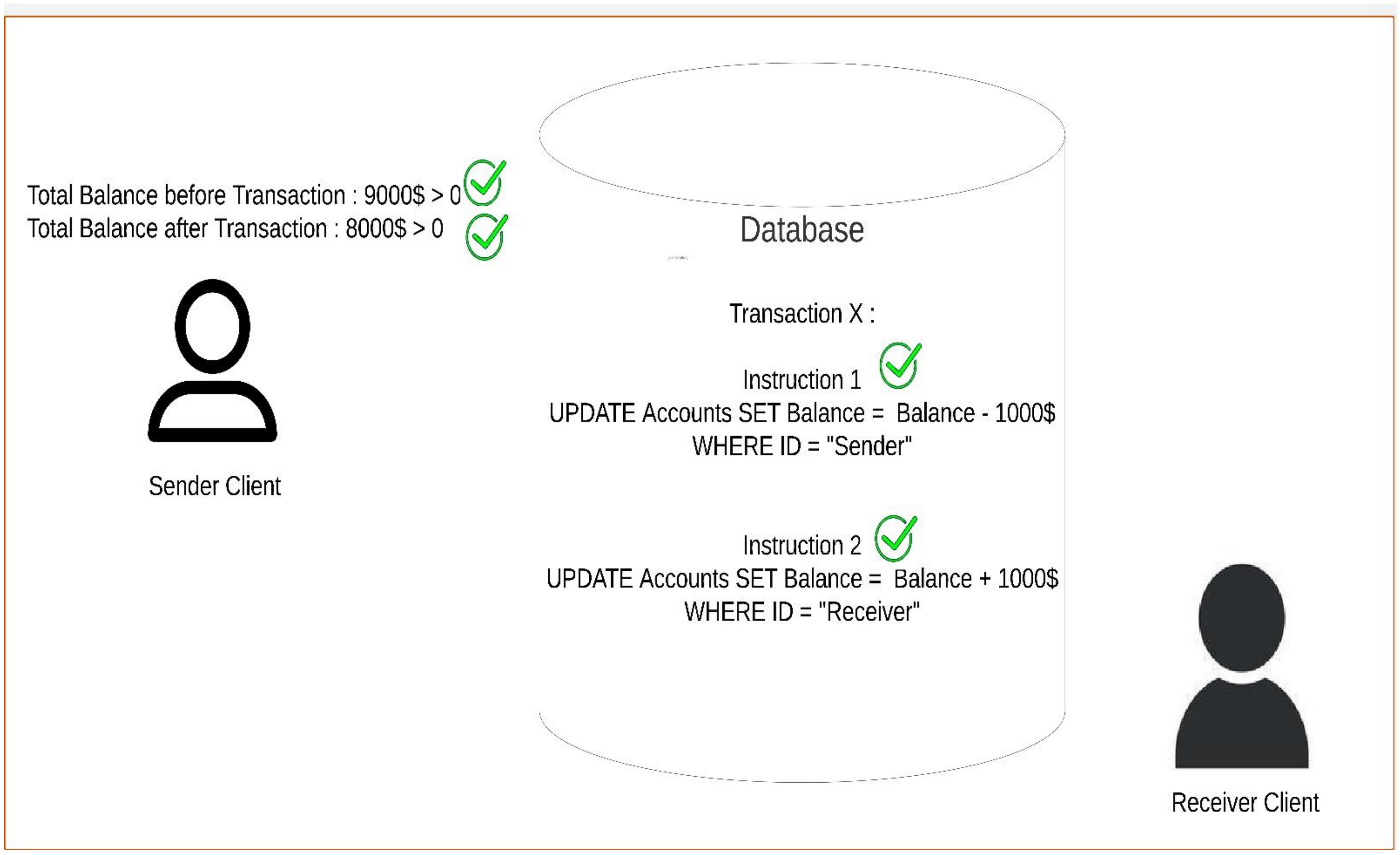
when the client try to send money that greater than his total balance this transaction must be failure.
for example client has total balance 700$ and try to send 1000$ to other client so this transaction must be failure because it will be move the database from valid consistency state to other invalid consistency state.


Total Balance before Transaction : 700$ > 0 ✓
Total Balance after Transaction : - 300$ < 0 ✗
Sender Client
Database
Transaction X :
Instruction 1 ✗
UPDATE Accounts SET Balance = Balance - 1000$
WHERE ID = "Sender"
Instruction 2
UPDATE Accounts SET Balance = Balance + 1000$
WHERE ID = "Receiver"
Receiver Client

Case 2 :

when the client try to send money that less than or equal than his total balance this transaction must be success.

for example client has total balance 9000$ and try to send 1000$ to other client so this transaction must be success because it will be move the database from valid consistency state to other valid consistency state.


Total Balance before Transaction : 9000$ > 0 ✓
Total Balance after Transaction : 8000$ > 0 ✓
Sender Client
Database
Transaction X :
Instruction 1 ✓
UPDATE Accounts SET Balance = Balance - 1000$
WHERE ID = "Sender"
Instruction 2 ✓
UPDATE Accounts SET Balance = Balance + 1000$
WHERE ID = "Receiver"
Receiver Client

Consistency vs Atomicity :

One of the misunderstanding concepts I see in the database world is consistency , what is actually mean by consistency ? what is the difference between consistency and atomicity ?

Atomicity :

Atomicity about all instructions or statements is executed successfully or rollbacked (not executed)

Consistency in ACID :

Consistency in ACID about the **results** of this instructions are correct or not.

Consistency in CAP :

Consistency in CAP about if the end user will see the last result or updated value from the replicated node in distributed systems when there are multiple copy of the same data in different nodes , there are different types of consistency in CAP That will be discuss in details in the later sections.

i → (isolation) :

What is isolation ?

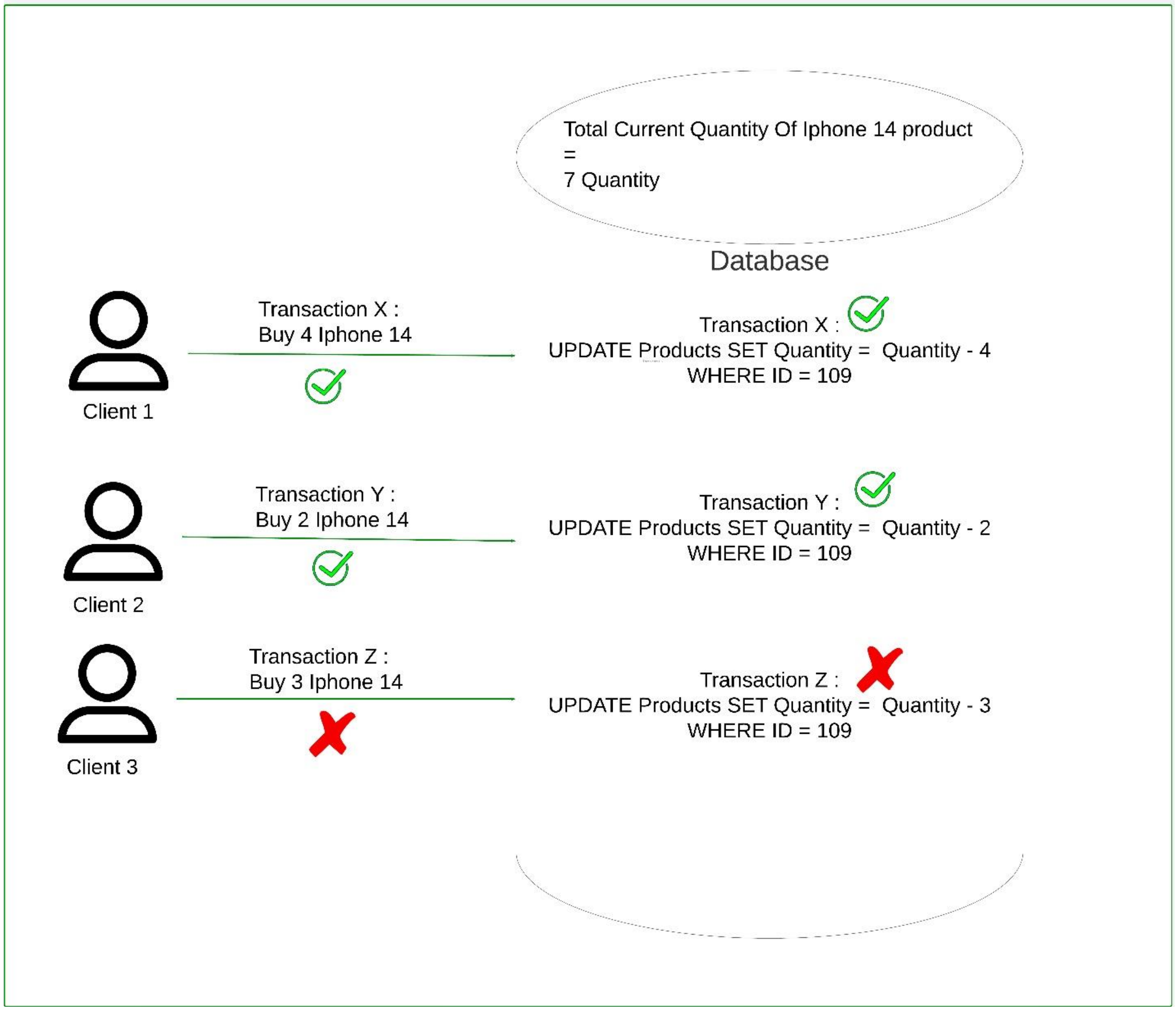
Isolation mean the ability of database to execute concurrent or parallel transactions on the same data as they executed serially without any conflicts.

There are different isolations levels where every level solve some concurrency issues , developers can choose the best one that suitable for their business needs.

We will not discuss the isolation levels here but we will just discuss the general idea of the isolation.

Isolation example :

Three clients try to buy the same product from the stock at the same time.



D → (Durability) :

What is Durability ?

Durability mean the database must be store the transactions and data on non-volatile storge like disk before it mark it as success or return success message to client .

What is the importance of durability ?

if the system or database crash due to power failures or other reason and then restart to working again , so the effects of the transactions will never lose and database continue to work from latest consistency state.

SECTION 3 No SQL Databases

3.1 What is actually mean by No SQL Database ?

- It just refers to **NOT ONLY SQL**
- It refer to any database that **does not** apply
  SQL Database standards on :

  1- Architecture level :

  does not store data on tabular format.

  2- Implementation level :

  does not implement all or some of SQL database standard properties Like Relations , ACID , SQL query language standard.

  So if **any one of the two previous condition** occur the database will be categorized as **No SQL Database**.

  Important note 1 :

  there are type of No SQL That store data on tabular format but does not implement all or some SQL database standard properties , so this type is categorized as NO SQL Database.

  Example : Column Family Database

Important note 2 :
some types of No SQL Database can support SQL Database Standard Properties but in customized or partial cases , so it categorized as NO SQL Database.
Example : some NO SQL Databases support ACID But per object not per transaction or multi-objects.

3.2 NO SQL Databases Types :

1. Documents Databases
2. Key-Value Databases
3. Column Family Databases
4. Graph Databases

We will discuss every No SQL Database type In details in the next chapters : -

- What are the business cases of every type ?
- What is the architecture of every type ?
- What is the data model of every type ?
- What are the common operations of every type ?

<u>But in the next section we will discuss the difference between SQL VS NO SQL in general regardless of every NO SQL Database type.</u>

SECTION 4 SQL VS NO SQL

There are a lot of factors to technically choose the best database suitable for your system.

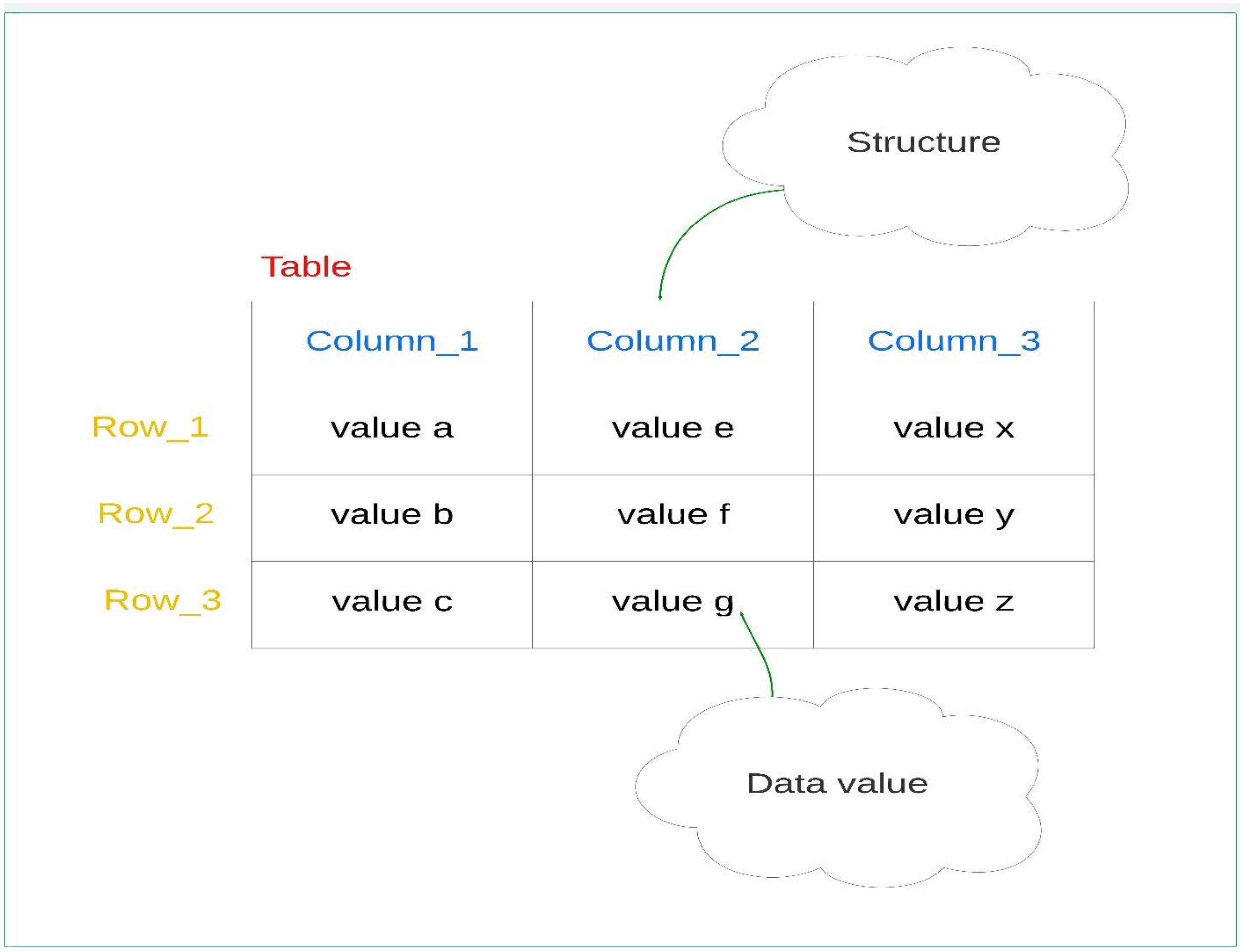
In this section we will discuss the main factors to know what if the system need SQL or NO SQL Database in general :

1. Data Model
2. Scalability
3. Query Language
4. ACID vs CAP

4.1 Data Model :

1.Structure data model :

- Every value is stored or mapped to a specific field at a specific location.
- All records of the same type should have the same fields.
- Structure and data values are decoupled or separated from each other.
- The common architecture for this data model type is tabular format (tables ,rows, columns)



	Column_1	Column_2	Column_3
Row_1	value a	value e	value x
Row_2	value b	value f	value y
Row_3	value c	value g	value z

2.Semi-structure data model :

- Semi-structure model is very similar to structure data model but there are some differences.

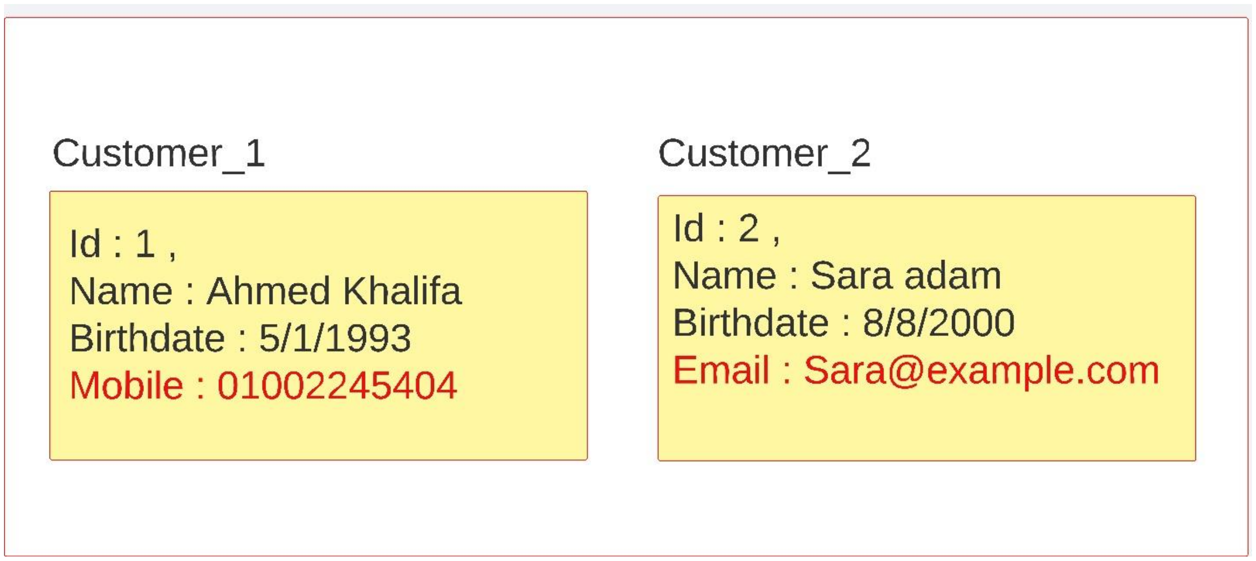
  First difference :

  Different records of the same types can have different fields.

  Second difference :

  Structure and values are coupled to each other, structure here is work as meta-data.

- Common patterns : json , xml , html.



3.Unstructured data model :

- This data model does not have any structure.

- The data is stored in native format as it received.

- Common examples of this pattern are text files, audio files, video files.

- Usually this data model has not query language but use data analysis tools , some of AI/MI algorithms to process this data and extract useful information from it.

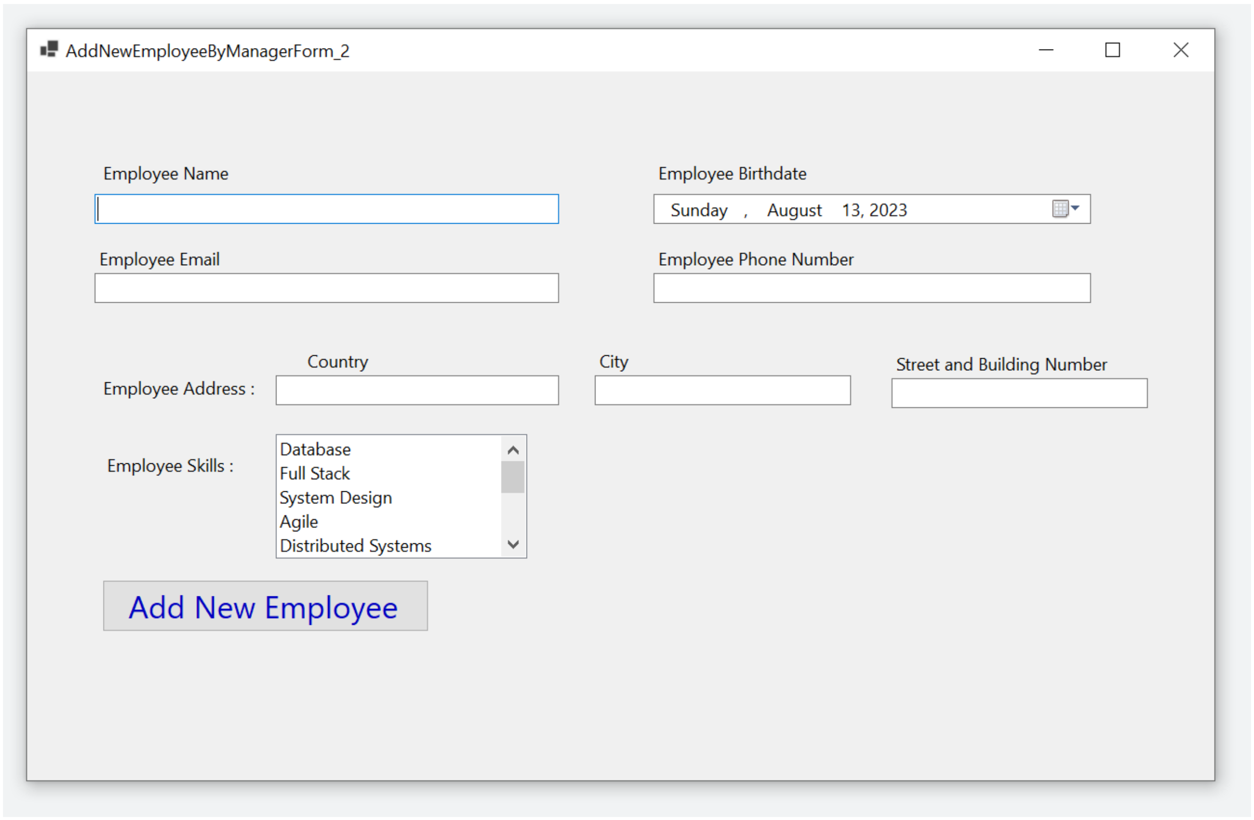
How to choose your database based on the data model ?

<u>SQL Databases :</u>

Usually, SQL Databases are suitable for structured data models where the schema is static.

The static schema means all the entities and fields of this entities are known and build by the developers on the development stage.

Business case example :
CRUD Management Systems Like HR Systems.



AddNewEmployeeByManagerForm_2

Employee Name	
Employee Birthdate	Sunday , August 13, 2023
Employee Email	
Employee Phone Number	
Employee Address : Country	
City	
Street and Building Number	
Employee Skills :	Database, Full Stack, System Design, Agile, Distributed Systems

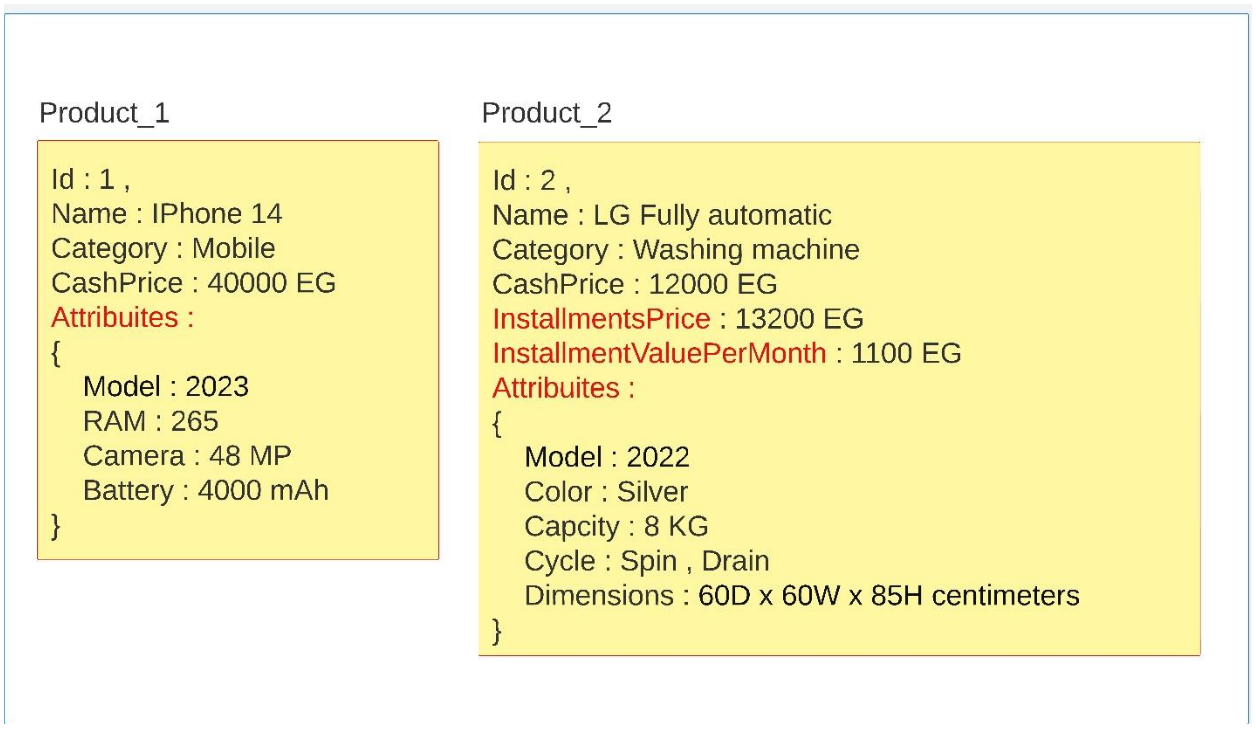
<u>NO SQL Databases :</u>

NO SQL Databases are usually suitable for :
1. Unstructured data models.
2. Semi-structured data models where the scheme is dynamic.

The dynamic scheme mean the attributes of different objects of the same entities are unknown and inserted in the run time by the end users of the systems.

Business case example :

E-commerce system where every product has different attributes, and this attributes are inserted on the run time by the seller or the admin of company , not by the developers themselves.



Product_1

```
Id : 1 ,
Name : IPhone 14
Category : Mobile
CashPrice : 40000 EG
Attribuites :
{
   Model : 2023
   RAM : 265
   Camera : 48 MP
   Battery : 4000 mAh
}
```

Product_2

```
Id : 2 ,
Name : LG Fully automatic
Category : Washing machine
CashPrice : 12000 EG
InstallmentsPrice : 13200 EG
InstallmentValuePerMonth : 1100 EG
Attribuites :
{
   Model : 2022
   Color : Silver
   Capcity : 8 KG
   Cycle : Spin , Drain
   Dimensions : 60D x 60W x 85H centimeters
}
```

4.2 Scalability :

The second major factor to determine what if your system need SQL vs NO SQL Database is the scalability.

What is mean by scalability?

Scalability has a wide range of definitions.
For good understanding of the scalability we need first to know what is the throughput and latency because the scalability is based on them.

Throughput :

 the number of operations the system can process on a specific period time.

Example :
facebook can serve 1,000,000 post per second.

Latency :

The time taken by the system to process or handle one operation , latency is also known as response time.

Example :
facebook user can publish one post and wait 300 ms until the post is processed and published successfully.

Scalability is the ability of the system to increase the throughput and keep the latency the same.

Example :
If the system has **4 nodes** , and every node can handle **2k** requests per second , so the total throughput of the system will be **8k** , and every request for example take **150 ms** as latency or response time.

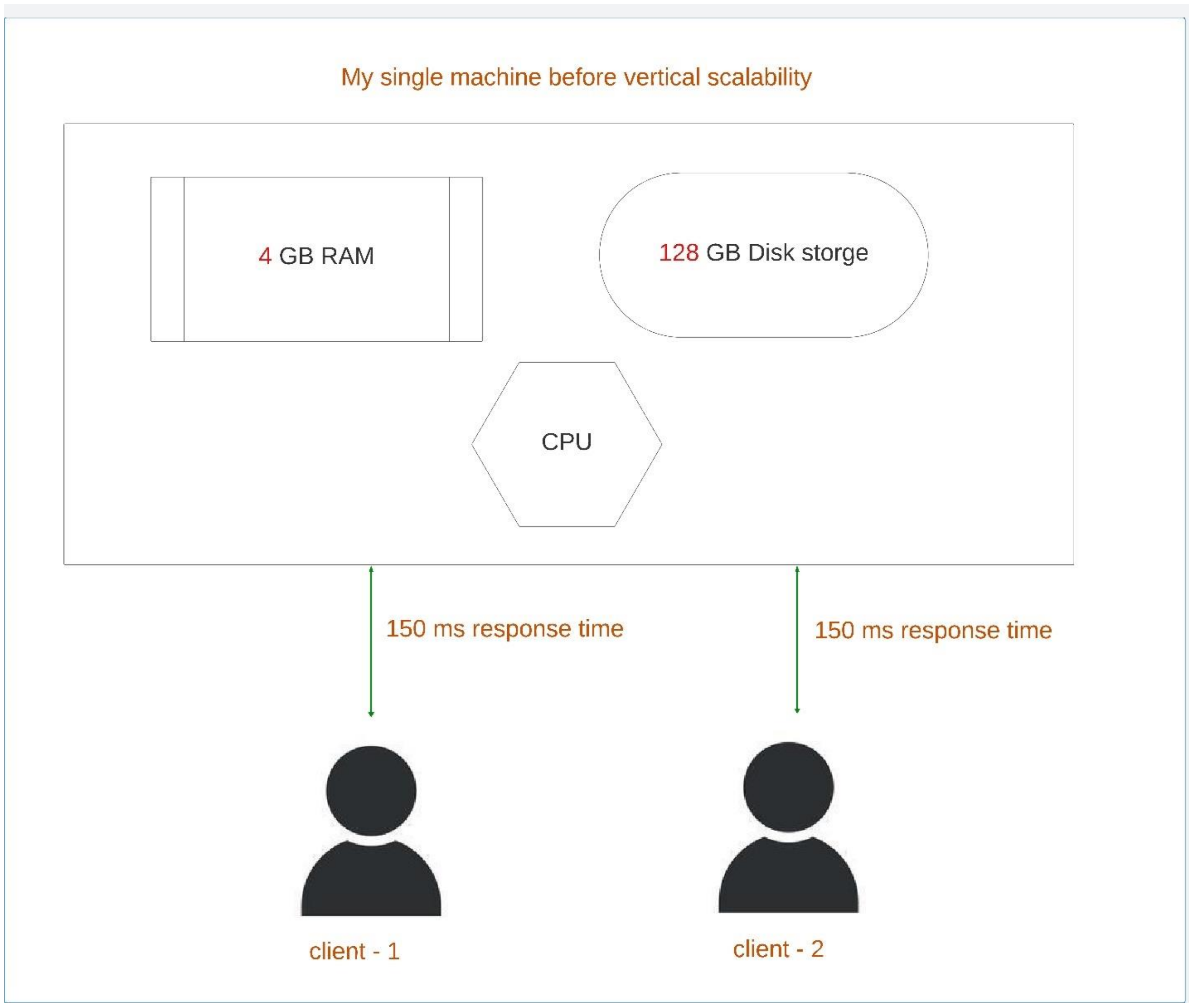
we need to increase the scalability of the system to serve **12k** per second , so we add **new 2 nodes** , so the total nodes on the system will be **6 nodes.**

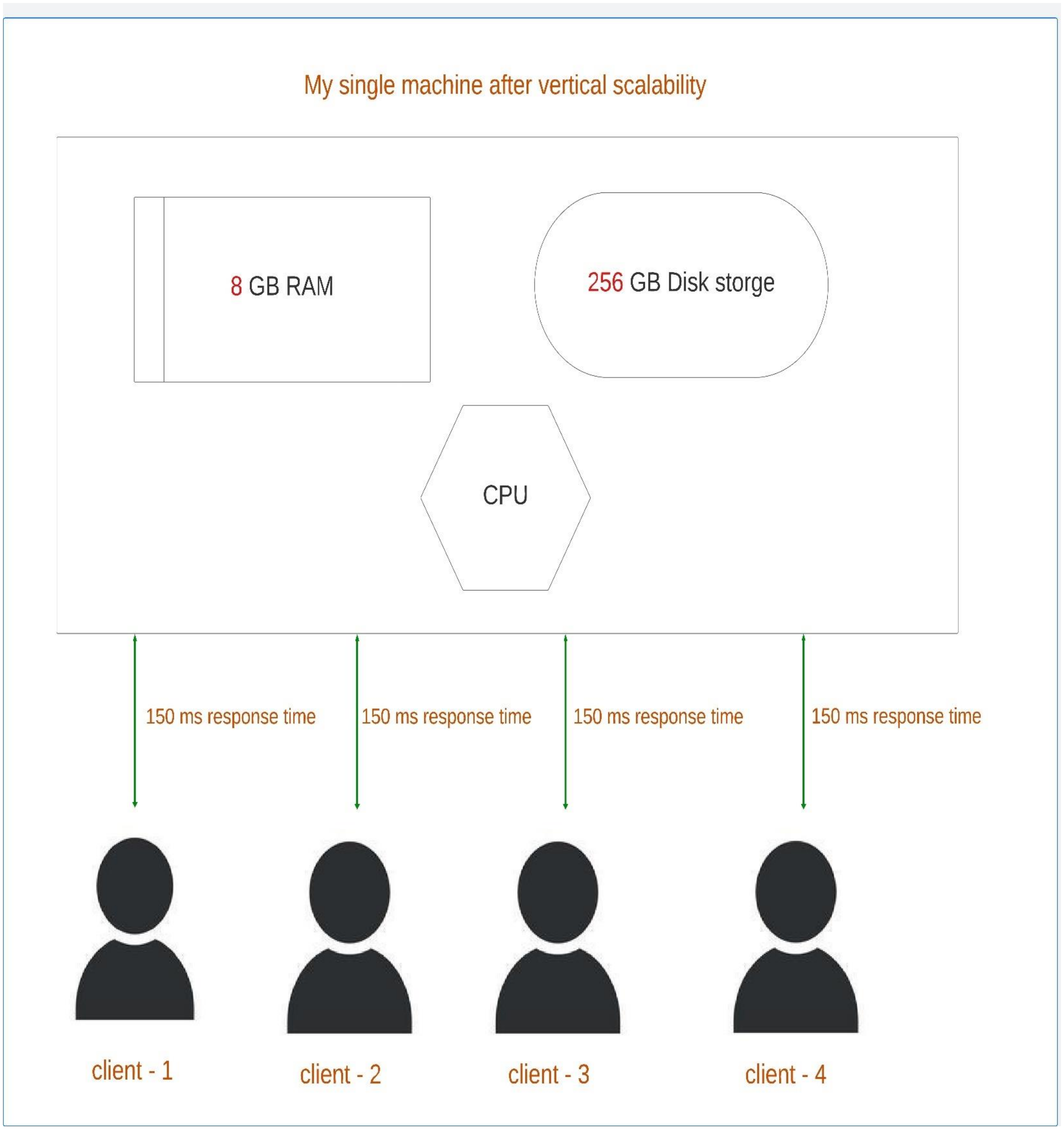
To say the system is scalable, the average of latency or the response time when the system has **4 nodes** and serve **8k requests** **must be equal** the average of latency or the response time when the system has **6 nodes** and serve **12k requests**.
So the response time must be **150 ms** when the system has **4 nodes and also 6 nodes**.

Types of scalability

1. Vertical Scalability

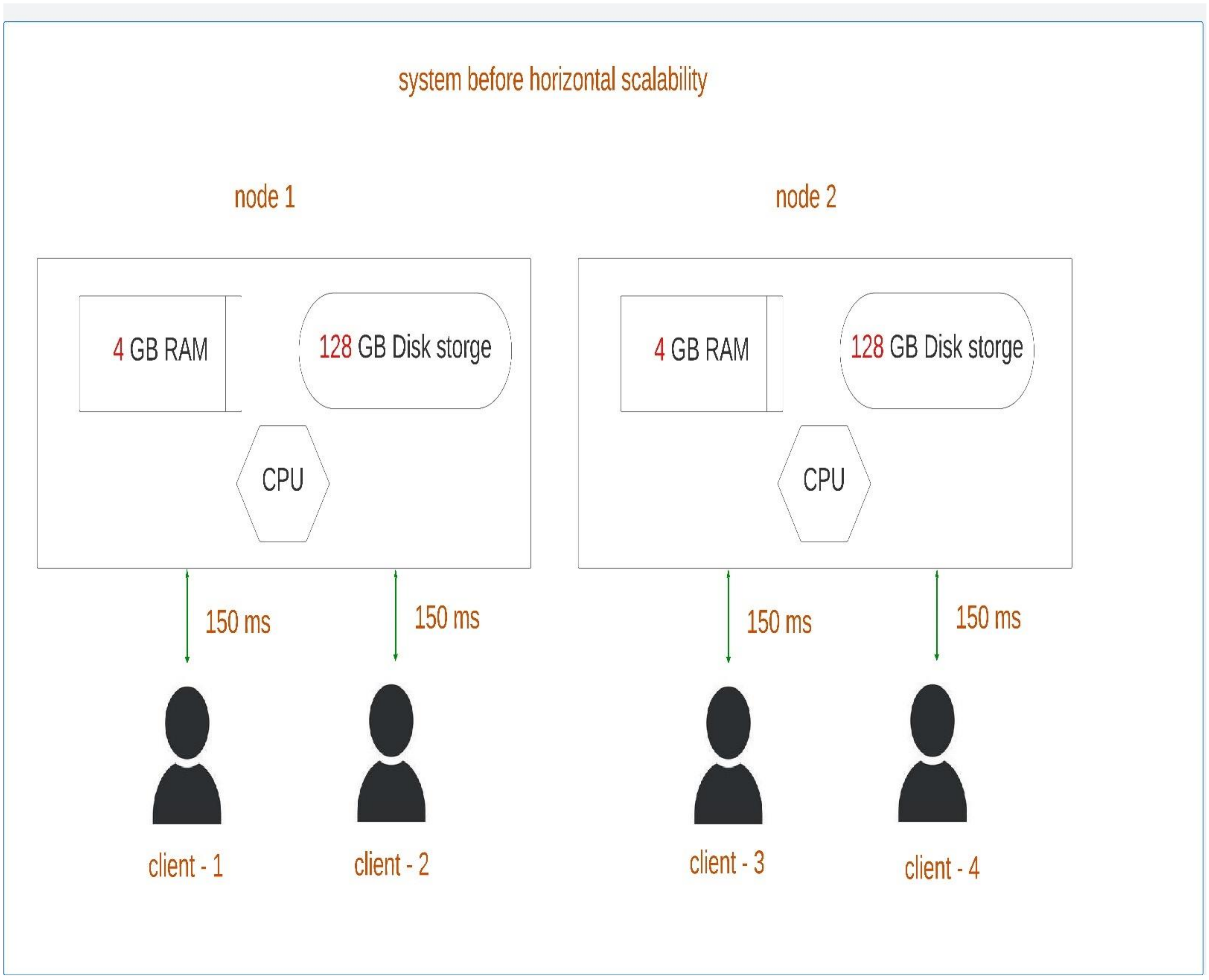
Vertical scalability mean increasing the resources of single machine like RAM , Disk storge and .. etc.

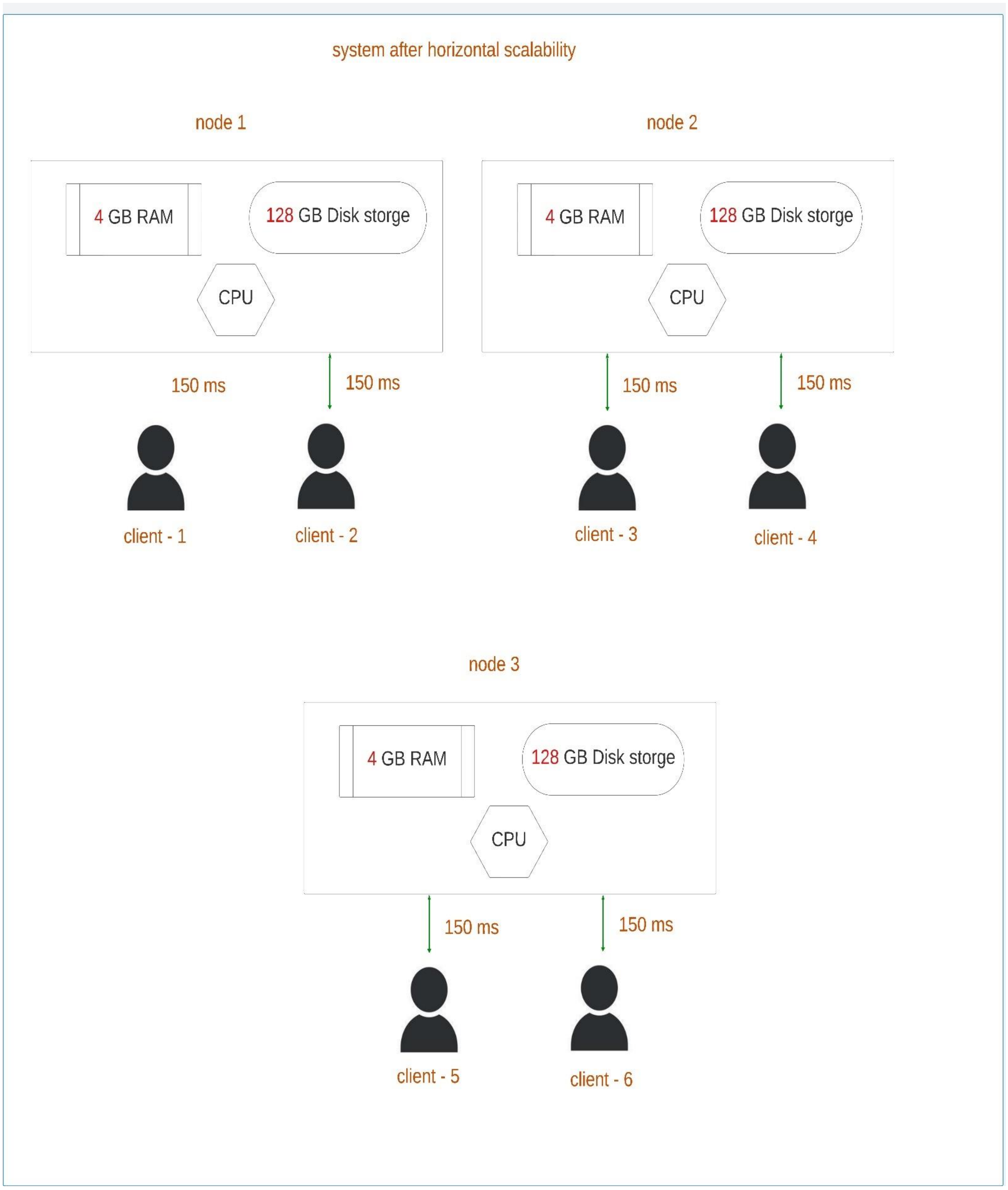



My single machine after vertical scalability
8 GB RAM
256 GB Disk storge
CPU
150 ms response time
150 ms response time
150 ms response time
150 ms response time
client - 1
client - 2
client - 3
client - 4

2. Horizontal Scalability

Horizontal scalability means increasing the number of nodes in the system.




system after horizontal scalability
node 1
4 GB RAM
128 GB Disk storge
CPU
150 ms
150 ms
client - 1
client - 2
node 2
4 GB RAM
128 GB Disk storge
CPU
150 ms
150 ms
client - 3
client - 4
node 3
4 GB RAM
128 GB Disk storge
CPU
150 ms
150 ms
client - 5
client - 6

How to choose your database based on the scalability ?

SQL Databases :

SQL Database works very good with vertical scalability.

SQL Database can also work with horizontal scalability but there will be a lot of complexity and performance issues when there are a lot of replicated database nodes in the system.

NO SQL Databases :

NO SQL Database works very good with horizontal scalability.

It is designed from scratch to work on distributed systems.

4.3 Query Language :

SQL Databases :

- Standard Query Language and Operations :

SQL Databases have standard query language which every provider implement this standard on its database :

SQL is the standard query language.

PL SQL is SQL of Oracle database.

T SQL is SQL of Microsoft SQL Server database.

So all SQL Databases have the same operations on different providers.

- Complex Queries :

SQL Databases support complex queries like :

1. Sub-Query
2. Join
3. Aggregations functions

SQL databases can work good when your system has complex business domain and needs different views of data.

NO SQL Databases :

- Standard Query Language and Operations :

NO SQL Databases does not have standard query language.

Every NO SQL Database provider has its own query language and customize the database operations to serve its goals.

- Complex Queries :

NO SQL Databases work good with simple queries and usually need straightforward way to access the data.

4.4 ACID VS CAP

SQL Databases :

SQL Databases are based on ACID Principles.

ACID Principles are very important for systems that have **Critical or Sensitive Data that need Strong Consistency Like :**

- Money transactions on banking systems
- Payments
- Stock or warehouse systems.

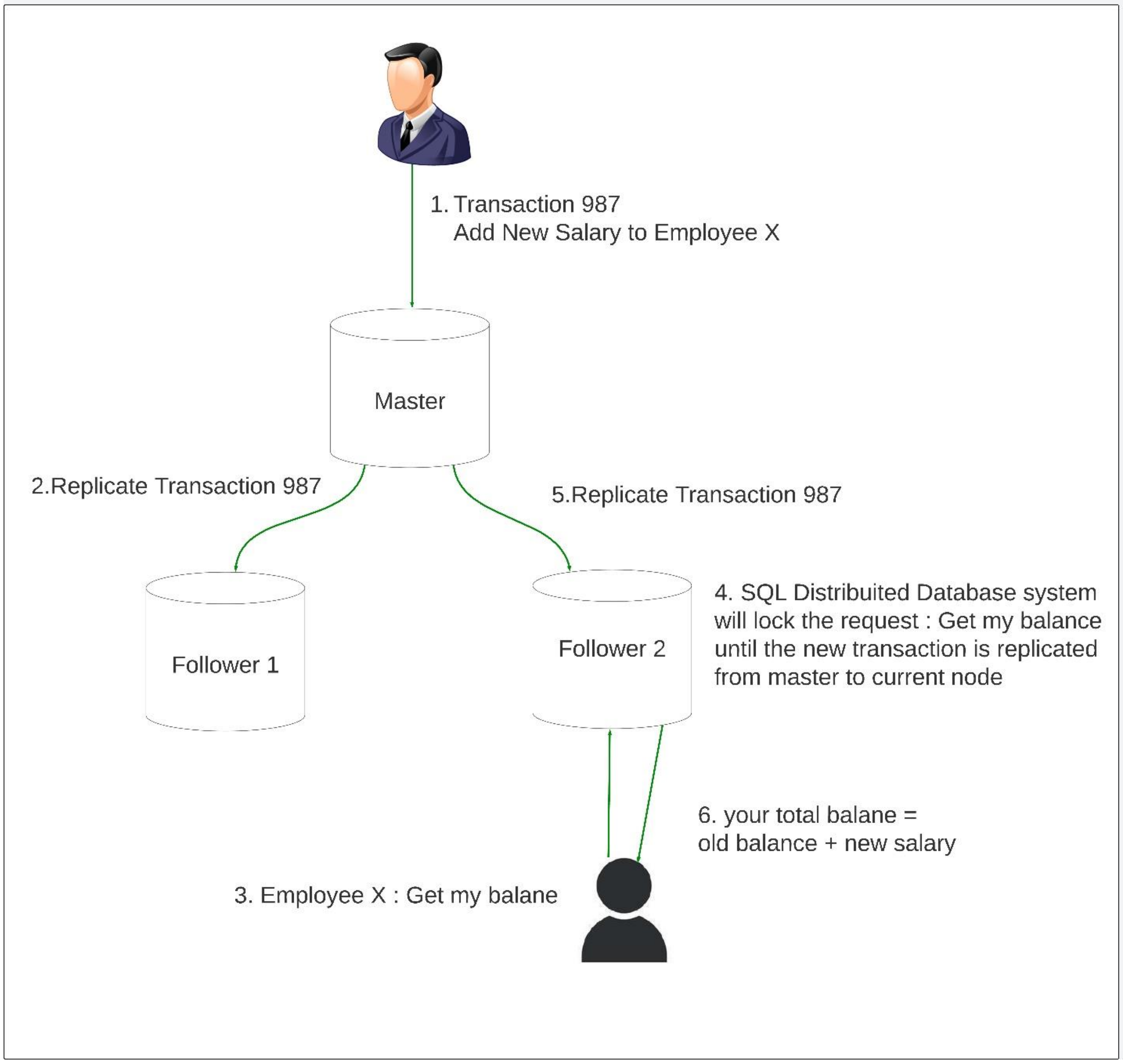
Users of this systems are more interested on strong consistency than any thing.

Users of this systems can accept some delays or high latency but can not accept to see any old view or inconsistency data.

Example :
if you open your bank account after the salary is transferred to you from your company , you need to see the total balance is old balance + new salary , you can accept the account page take 3 seconds to load , but can not accept the total balance is inconsistent or read from replicated node which has not received the transaction of your salary until now.

Strong Consistency > Availability

ACID Need Strong Consistency in distributed systems.


1. Transaction 987
Add New Salary to Employee X
Master
2.Replicate Transaction 987
5.Replicate Transaction 987
Follower 1
Follower 2
4. SQL Distribuited Database system will lock the request : Get my balance until the new transaction is replicated from master to current node
6. your total balane = old balance + new salary
3. Employee X : Get my balane

NO SQL Databases :

NO SQL Database are based on CAP Theorem.

CAP Theorem are very important for systems that have

Big Data or Not Critical Data That Can Accept Eventual Consistency Like :

- Social media apps
- News apps

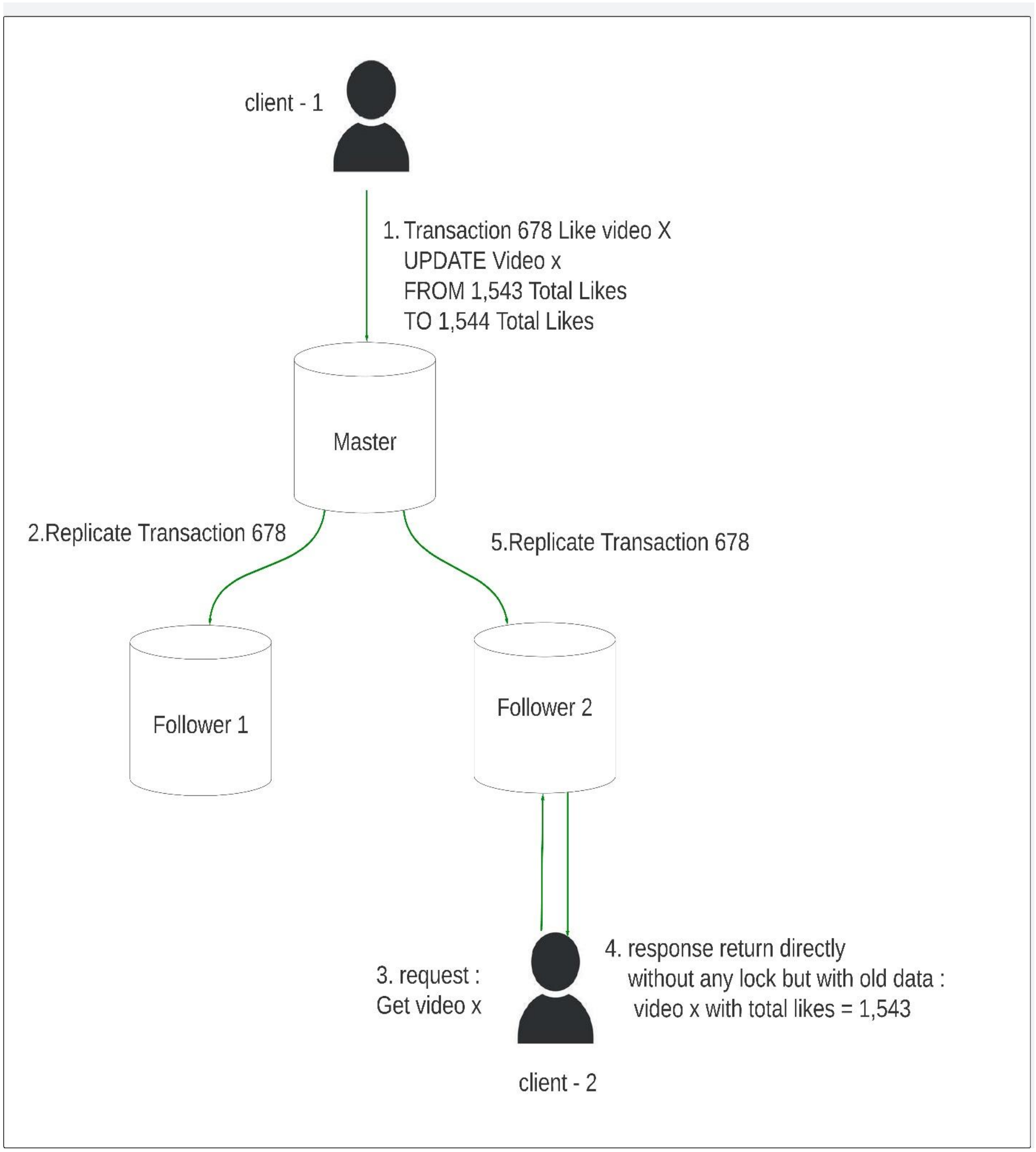
Users of this systems are more interested on high availability and low latency than any thing , they can accept eventual consistency but they can not accept slow response time.

Although some NO SQL Databases providers support strong consistency but the main benefit or perfect usage of NO SQL Databases is with Eventual consistency.

Example

If you watch video on YouTube , did you care about the accuracy of total number of likes is 1,543 or 1,544 ? or you more interested on the page and video are loaded fast ?

High Availability > Strong Consistency


client - 1
1. Transaction 678 Like video X
UPDATE Video x
FROM 1,543 Total Likes
TO 1,544 Total Likes
Master
2.Replicate Transaction 678
5.Replicate Transaction 678
Follower 1
Follower 2
3. request :
Get video x
4. response return directly
without any lock but with old data :
video x with total likes = 1,543
client - 2