Introduction to SQL

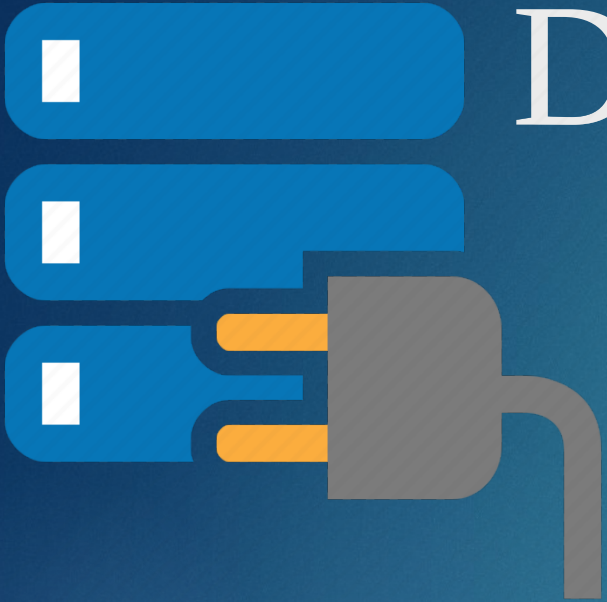
Introduction to SQL

SQL (Structured Query Language) is a programming language used to manage and manipulate data stored in a relational database management system (RDBMS).
In simpler terms, SQL is used to interact with databases, **retrieve and manipulate** data, and perform various operations on the data, such as inserting, updating, and deleting records. It provides a standard way to communicate with databases, making it possible for different software applications to work together.

Why we are using SQL?

- SQL is also a powerful language that allows for complex queries, joins, and aggregates, making it useful for a wide range of applications, from small businesses to large enterprises.
- SQL is Standard language used by many different database systems…
- SQL have the ability to manage large amount of data,easy to use,Data security,flexible and adapted and scalability(fixing the number of users)

Database Connectivity



Database Connectivity can be achieved using different methods, such as using programming languages like SQL or connecting through application programming interfaces (APIs). The use of middleware technologies like ODBC and JDBC can also facilitate database connectivity.

Database Connectivity

To connect to a database using SQL, you need to create a connection string that contains the database server name, database name, username, and password.

Here's an example of a connection string in PHP:

```
$conn = mysqli_connect("localhost", "username",
"password", "database_name");
if (!$conn) {
    die("Connection failed: "
.mysqli_connect_error());
 }
else
{
echo “Connection Successful”;
}
```

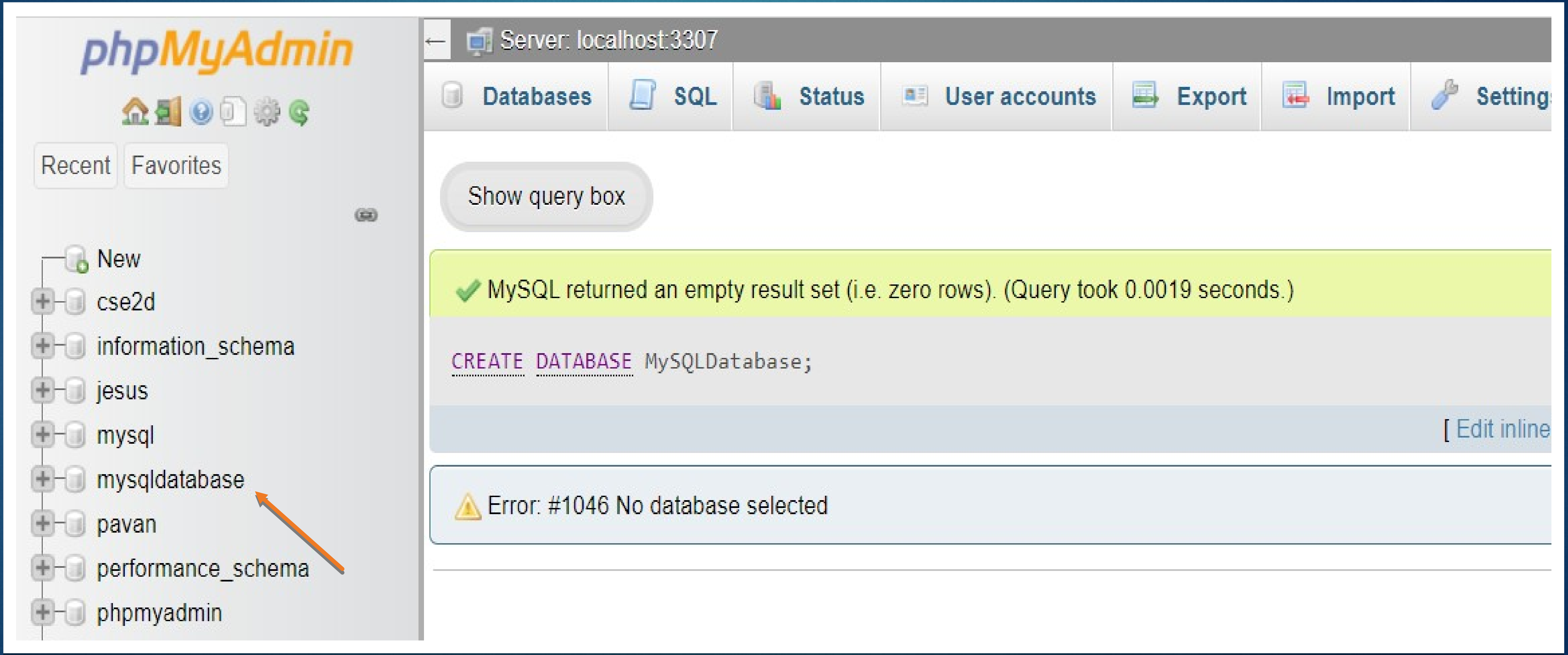
Creating Database

To create a database in SQL, you can use the **CREATE DATABASE** statement. The CREATE DATABASE statement may include additional parameters such as **character set** and **collation**. Here's an example of creating a database named "mydatabase":

```
Syntax:-  CREATE DATABASE mydatabase;

Query:-   CREATE DATABASE LakhSingh;
```

Execution:-



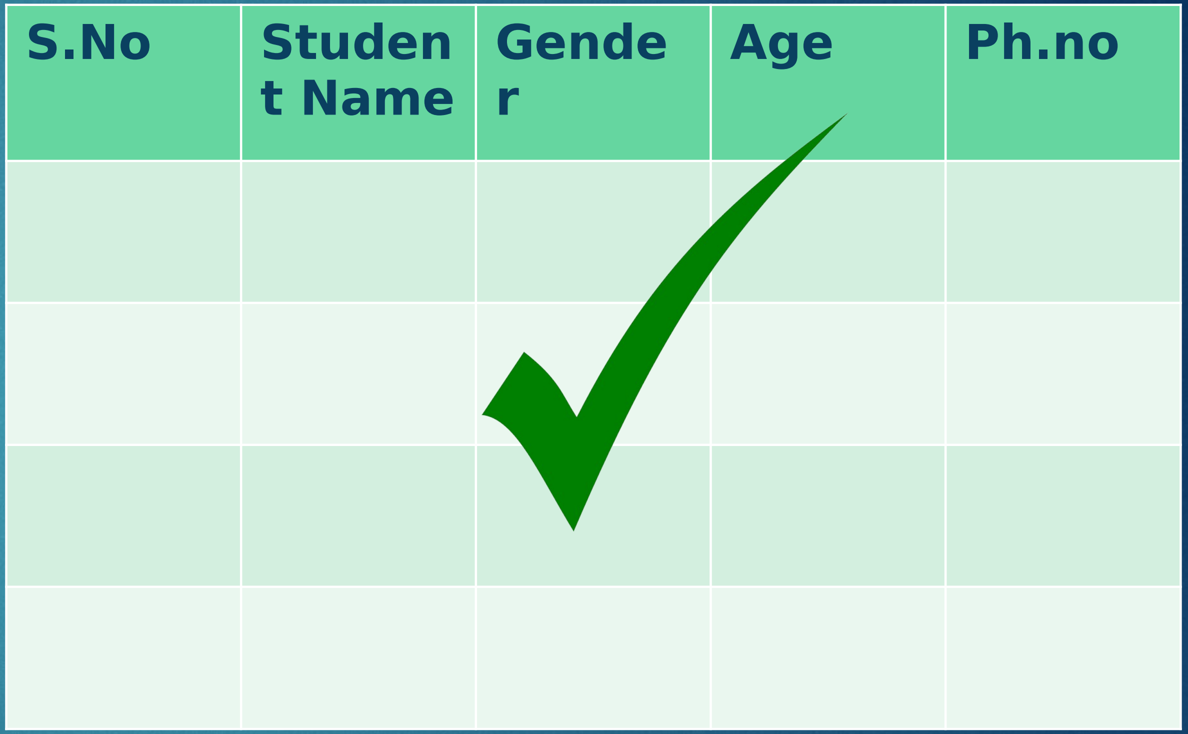
✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0019 seconds.)

```
CREATE DATABASE MySQLDatabase;
```

⚠ Error: #1046 No database selected

Creating Table

Not this table



S.No	Studen t Name	Gende r	Age	Ph.no

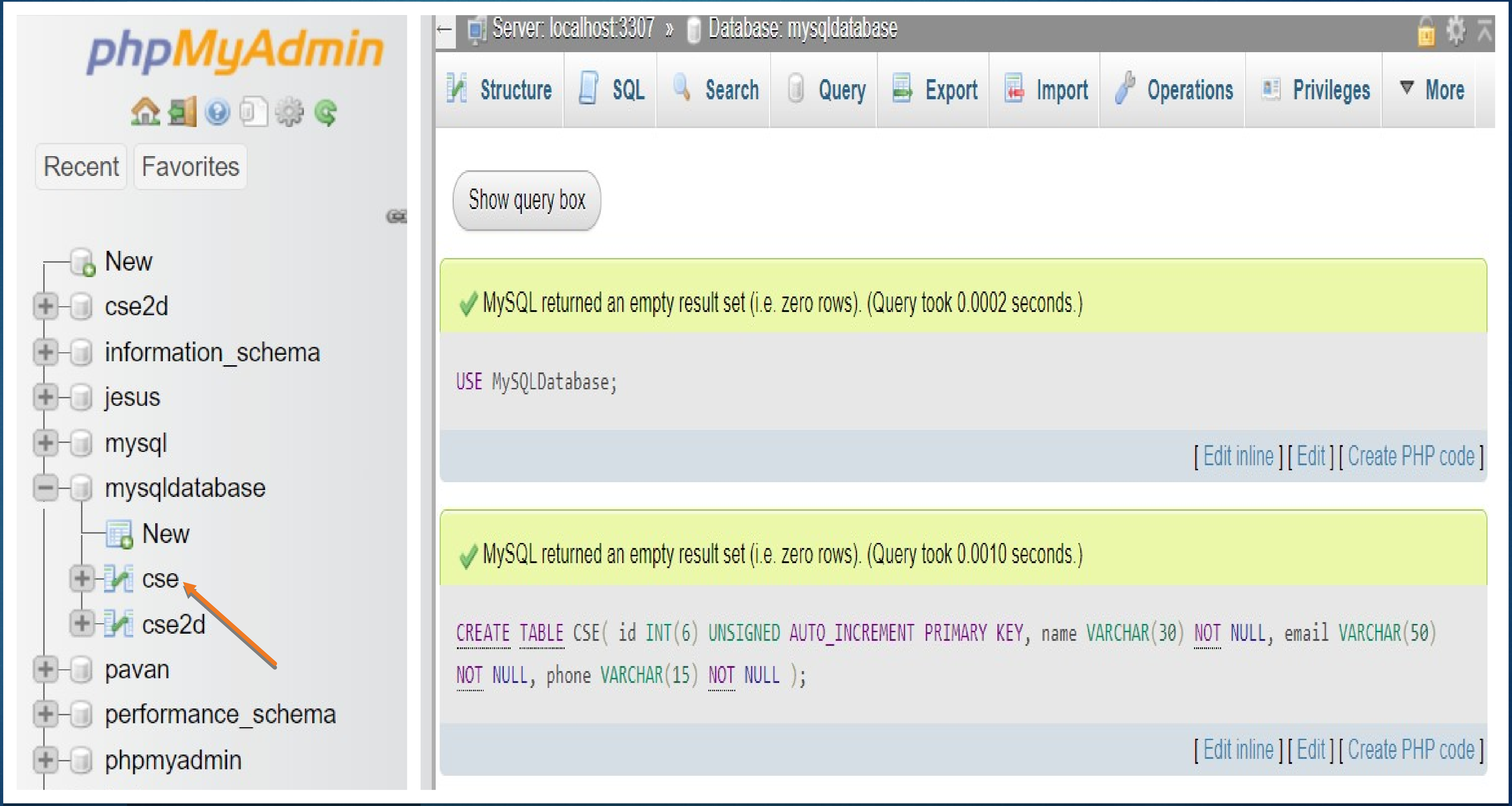
This table

Creating Table

To create a table in SQL, you can use the **CREATE TABLE** statement. Here's an example of creating a table named “CSE-2D" with columns "id", "name", "email", and "phone":

```
CREATE TABLE CSE-2D (
  id  INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(30) NOT NULL,
  email VARCHAR(50) NOT NULL,
  phone VARCHAR(15) NOT NULL
);
```

Execution:-



✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0002 seconds.)

```
USE MySQLDatabase;
```

[Edit inline] [Edit] [Create PHP code]

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0010 seconds.)

```
CREATE TABLE CSE( id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY, name VARCHAR(30) NOT NULL, email VARCHAR(50)
NOT NULL, phone VARCHAR(15) NOT NULL );
```

[Edit inline] [Edit] [Create PHP code]

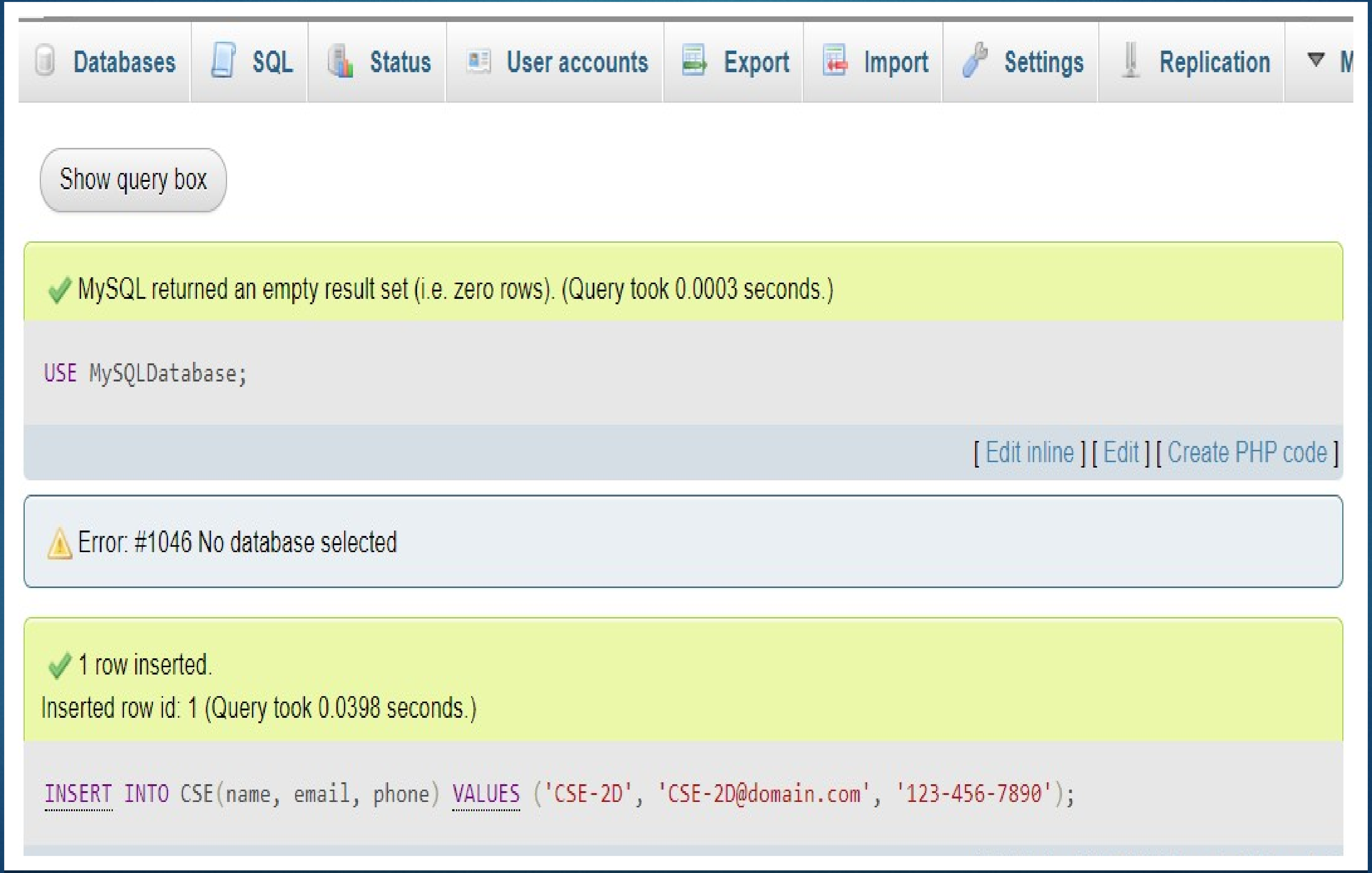
Inserting Data

To insert data into a table in SQL, you can use the **INSERT INTO** statement. Here's an example of inserting a customer named “CSE-2D" with email “CSE-2D@domain.com" and phone number “1234567890”:

Syntax:- INSERT INTO table_name (column1, column2, column3, ...) VALUES (value1, value2, value3, ...);

Query:- INSERT INTO CSE-2D (name, email, phone)
VALUES (‘CSE-2D’, ‘CSE-2D@domain.com', '123-456-7890’);

Execution1:-



✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0003 seconds.)

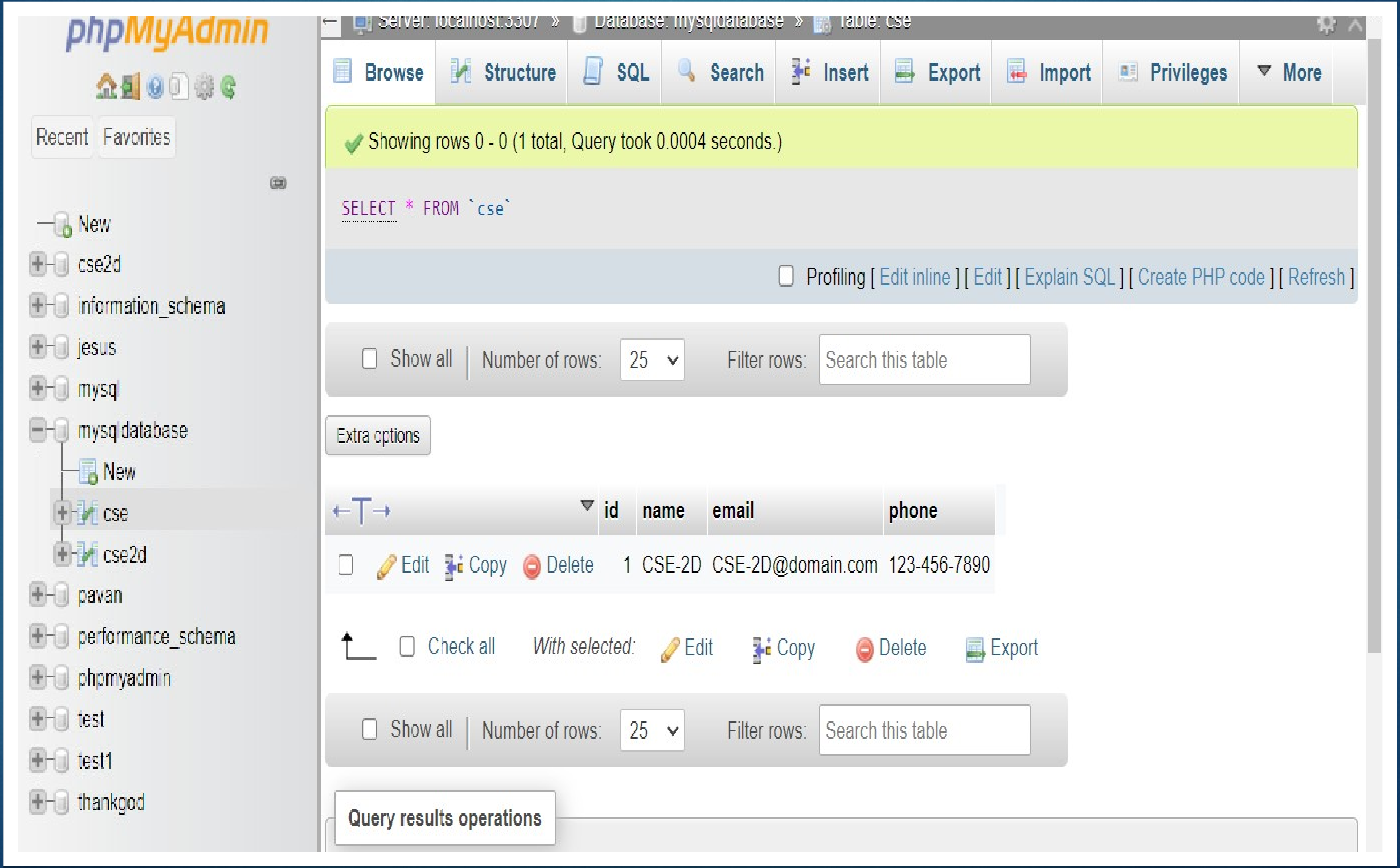
```
USE MySQLDatabase;
```

Error: #1046 No database selected

✓ 1 row inserted.
Inserted row id: 1 (Query took 0.0398 seconds.)

```
INSERT INTO CSE(name, email, phone) VALUES ('CSE-2D', 'CSE-2D@domain.com', '123-456-7890');
```

Execution2:-



Showing rows 0 - 0 (1 total, Query took 0.0004 seconds.)

```
SELECT * FROM `cse`
```

	id	name	email	phone
Edit Copy Delete	1	CSE-2D	CSE-2D@domain.com	123-456-7890

Prepared Statements

- Prepared statements in SQL (Structured Query Language) are a way to execute a query more efficiently and securely.
- In before of the MySQL version 4.1,before returning the result to the client, MySQL has fully **parse** the Query and converts the result into the String
- Our Prepared Statement was introduced in MySQL version 4.1.
- The main use of the prepared statements is to execute the same statement repeatedly with high efficiency
- Prepared statements that contain a placeholder(?)

```
SELECT * FROM student WHERE id=?;
```

Basic Work Flow

The basic work flow of the prepared statement mainly consists of two stages.

A.PREPARE:- At the prepare stage, a statement template is sent to the database server.

In shortly, **it prepares a statement for execution**

```
PREPARE statement_name FROM " Prepared_statement";
```

B.EXECUTE Stage:- In this stage, the client blinds parameter values and send them to the server.It will executes the query which already prepared by using PREPARE Statement

```
EXECUTE statement_name USING var1,var2
```

Basic Work Flow

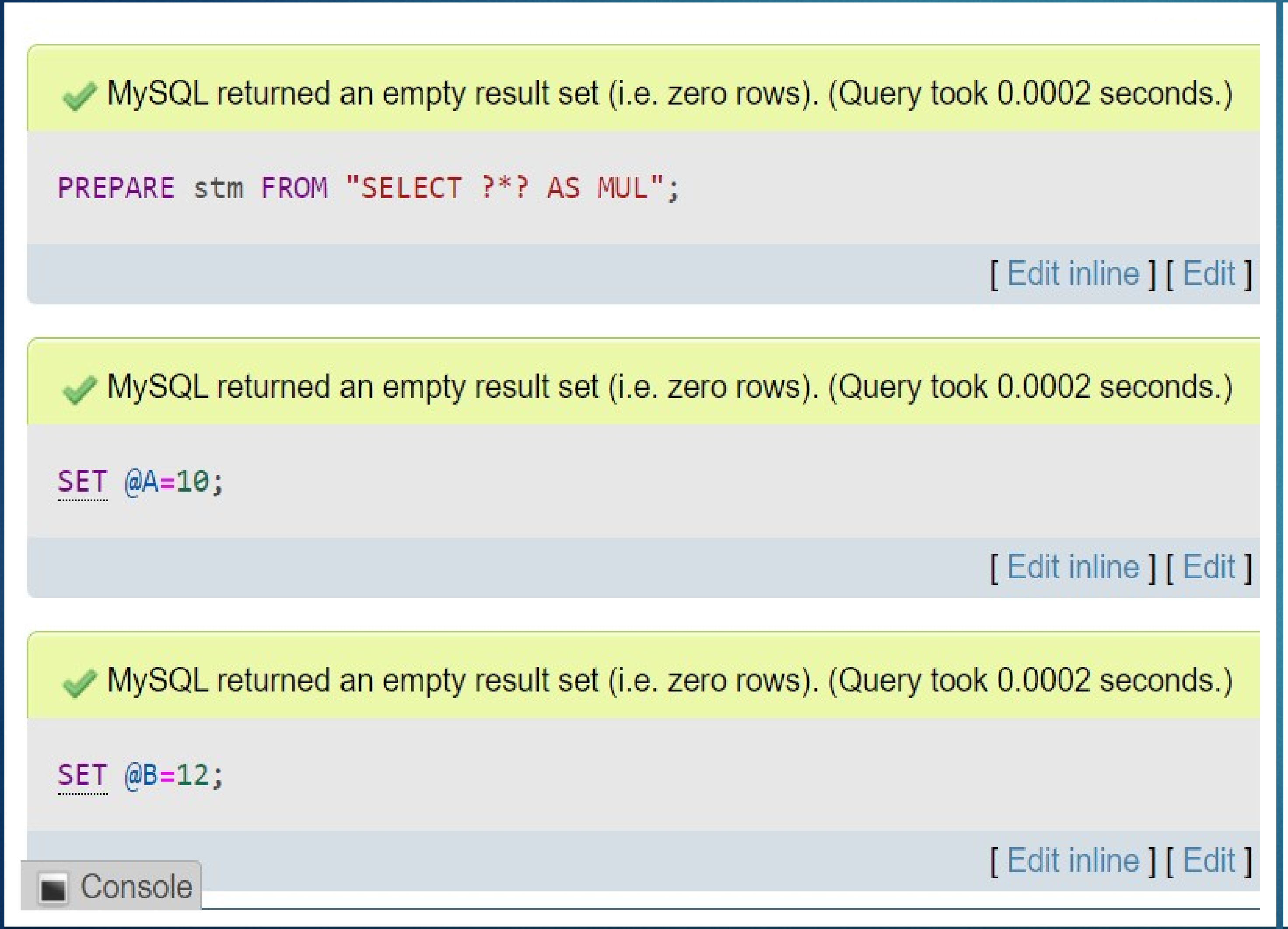
C. DEALLOCATE/DROP Stage

It is the last and optional stage, which is used to release the prepared statement.

After droping the prepare statement, we cannot use it any more

```
DEALLOCATE PREPARE statement_name;
```

Execution:-



✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0002 seconds.)

```
PREPARE stm FROM "SELECT ?*? AS MUL";
```

[Edit inline] [Edit]

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0002 seconds.)

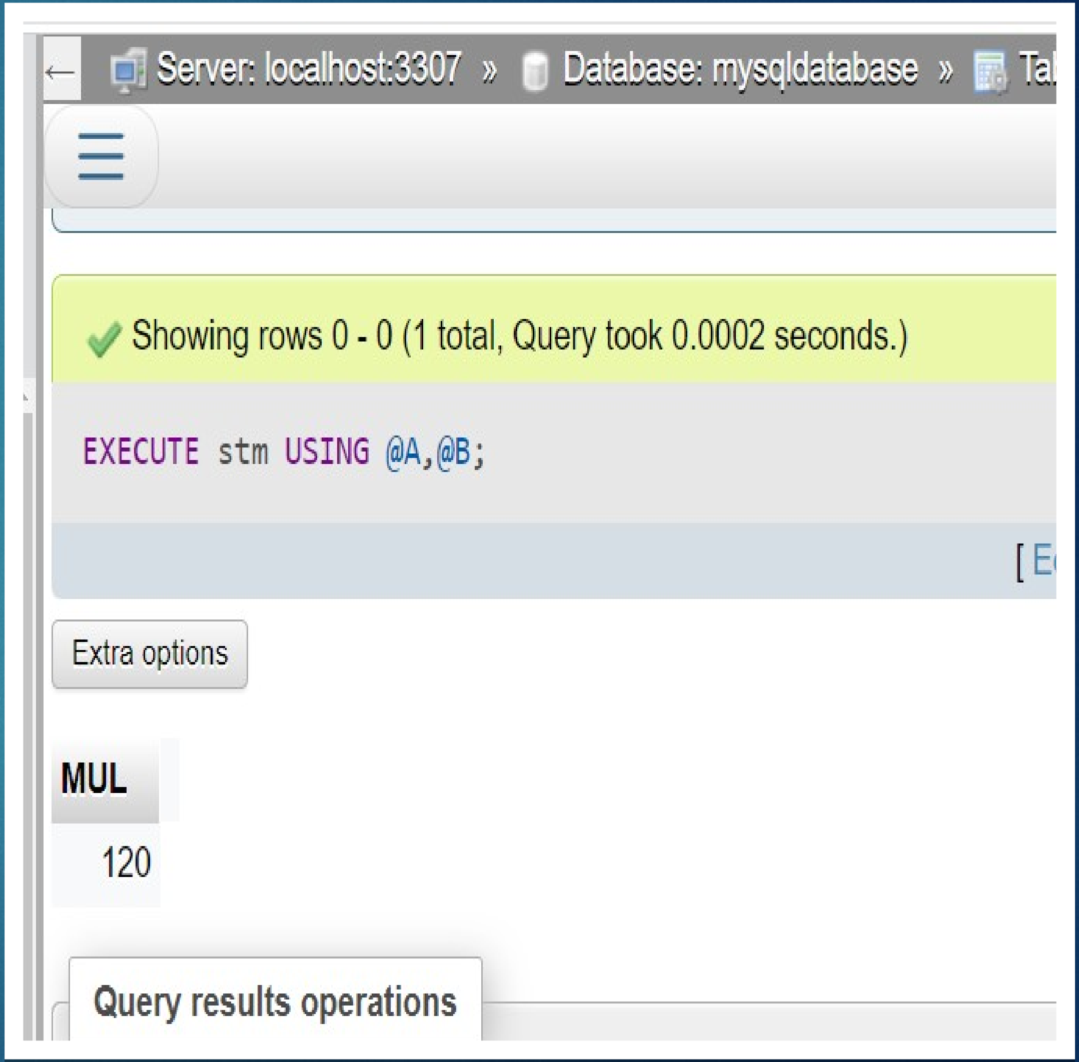
```
SET @A=10;
```

[Edit inline] [Edit]

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0002 seconds.)

```
SET @B=12;
```

[Edit inline] [Edit]



✓ Showing rows 0 - 0 (1 total, Query took 0.0002 seconds.)

```
EXECUTE stm USING @A,@B;
```

MUL
120

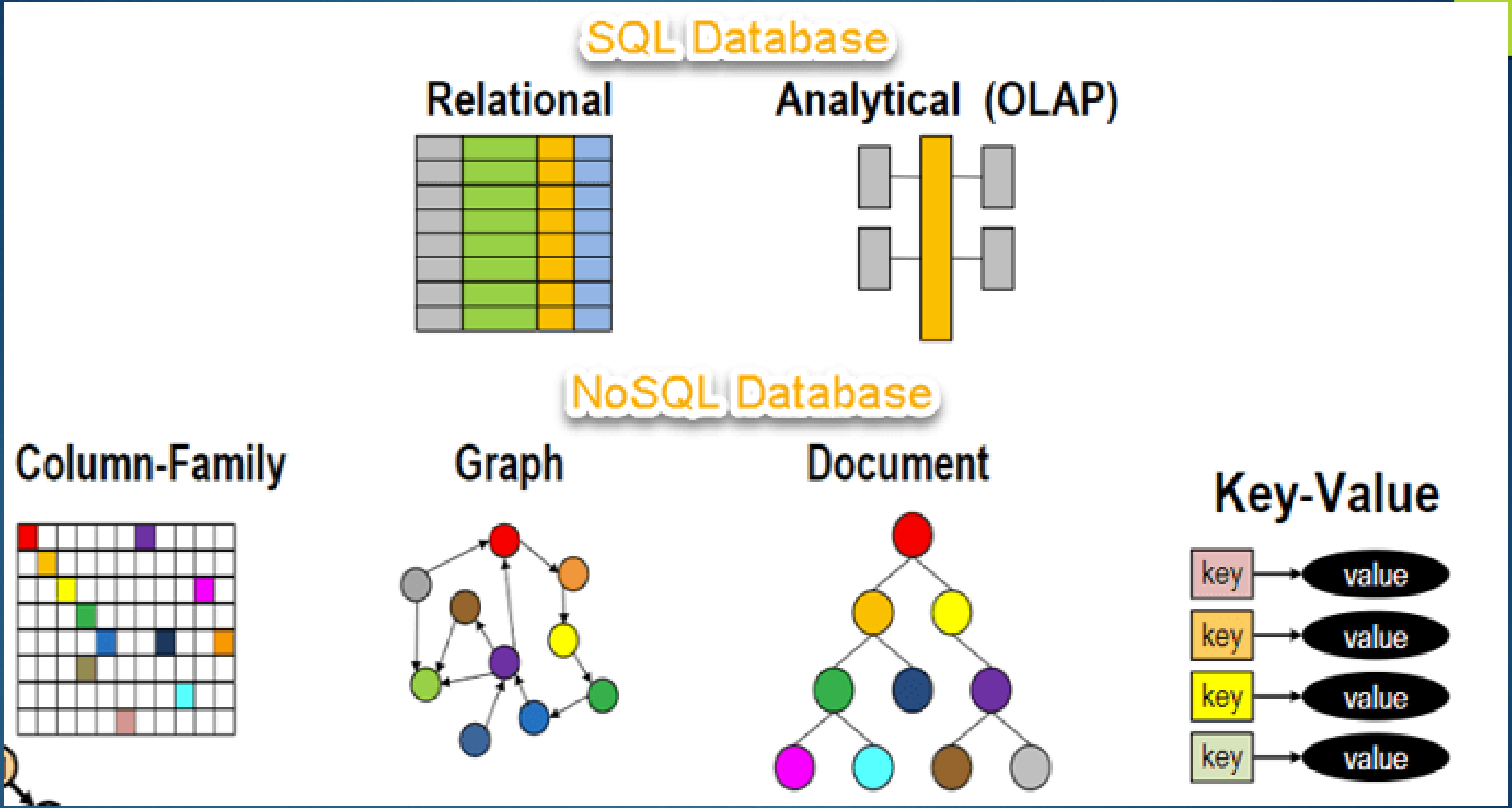
Key Points in Prepared Statements:-

- Prepare Statements created in one session are not available for other sessions. When a session ends, whether normally or abnormally, its prepared statements are no longer exists in the memory.

Statement	Prepared Statement
It is used when we want to execute the SQL query only once	It is used when we want to execute the SQL query multiple times
It is static, which means we cant pass parameters at runtime	It is dynamic,which means we can pass the parameters at runtime
The performace execution is slow	The performance execution is fast
It doesnot prevent SQL Injection	It helps to prevent SQL Injection

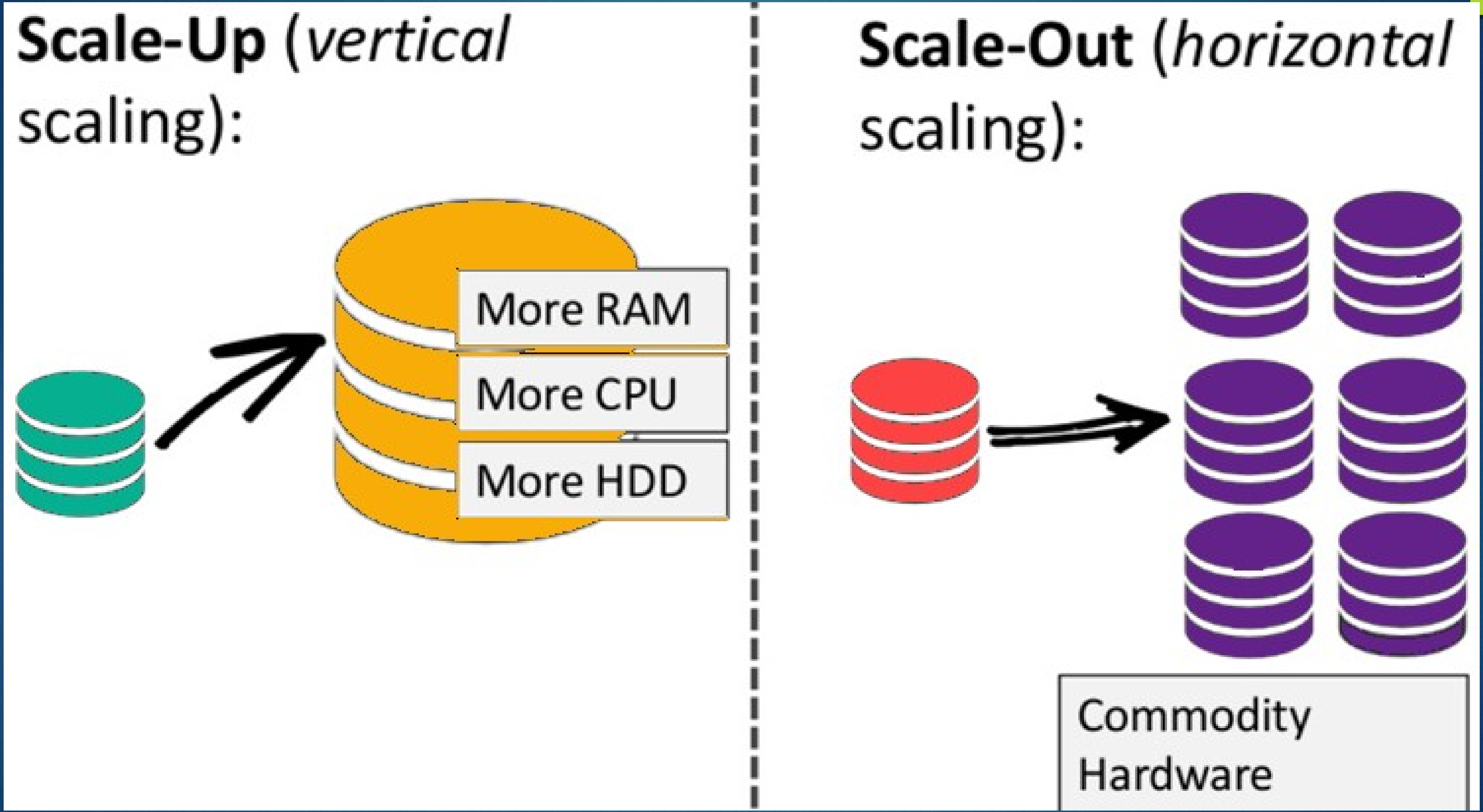
NoSQL

- **NoSQL** Database is a non-relational Data Management System, that does not require a fixed schema.
- The major purpose of using a NoSQL database is for distributed data stores with humongous data storage needs.
- NoSQL is used for Big data and real-time web apps.
- **NoSQL database** stands for “Not Only SQL” or “Not SQL.


SQL Database
Relational
Analytical (OLAP)
NoSQL Database
Column-Family
Graph
Document
Key-Value
key
value
key
value
key
value
key
value

Why NoSQL?

- The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data.
- The system response time becomes slow when you use RDBMS for massive volumes of data.
- The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as “scaling out.”


Scale-Up (*vertical* scaling):
More RAM
More CPU
More HDD
Scale-Out (*horizontal* scaling):
Commodity Hardware

Brief History of NoSQL Databases

- 1998- Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database
- 2000- Graph database Neo4j is launched
- 2004- Google BigTable is launched
- 2005- CouchDB is launched
- 2007- The research paper on Amazon Dynamo is released
- 2008- Facebooks open sources the Cassandra project
- 2009- The term NoSQL was reintroduced

Features of NoSQL

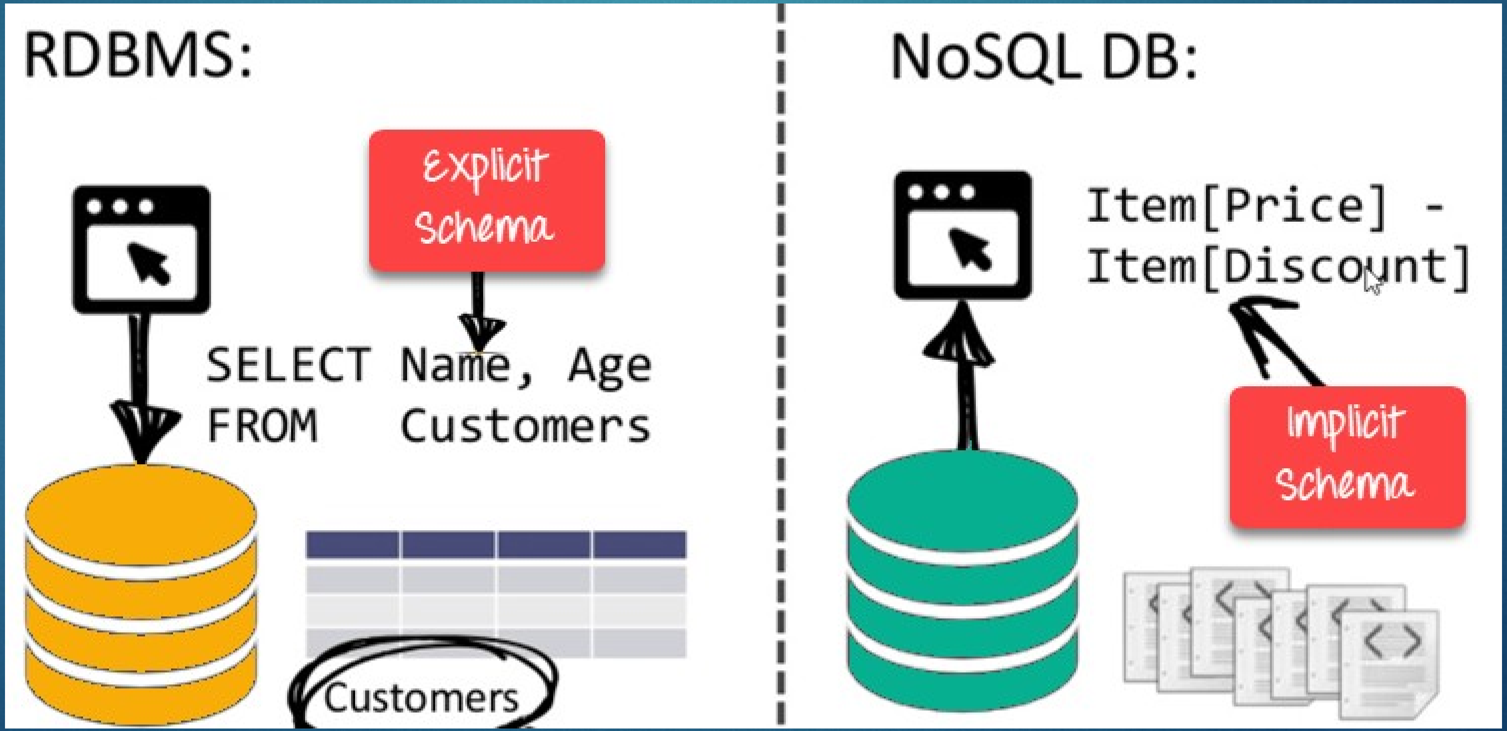
- Non Relational
- Schema free
- Simple API
- Distributed

Non-relational

- NoSQL databases never follow the relational model
- Never provide tables with flat fixed-column records
- Doesn’t require object-relational mapping and data normalization
- No complex features like query languages, query planners, referential integrity joins, ACID

Schema Free

- NoSQL databases are either schema-free or have relaxed schemas
- Do not require any sort of definition of the schema of the data
- Offers heterogeneous structures of data in the same domain

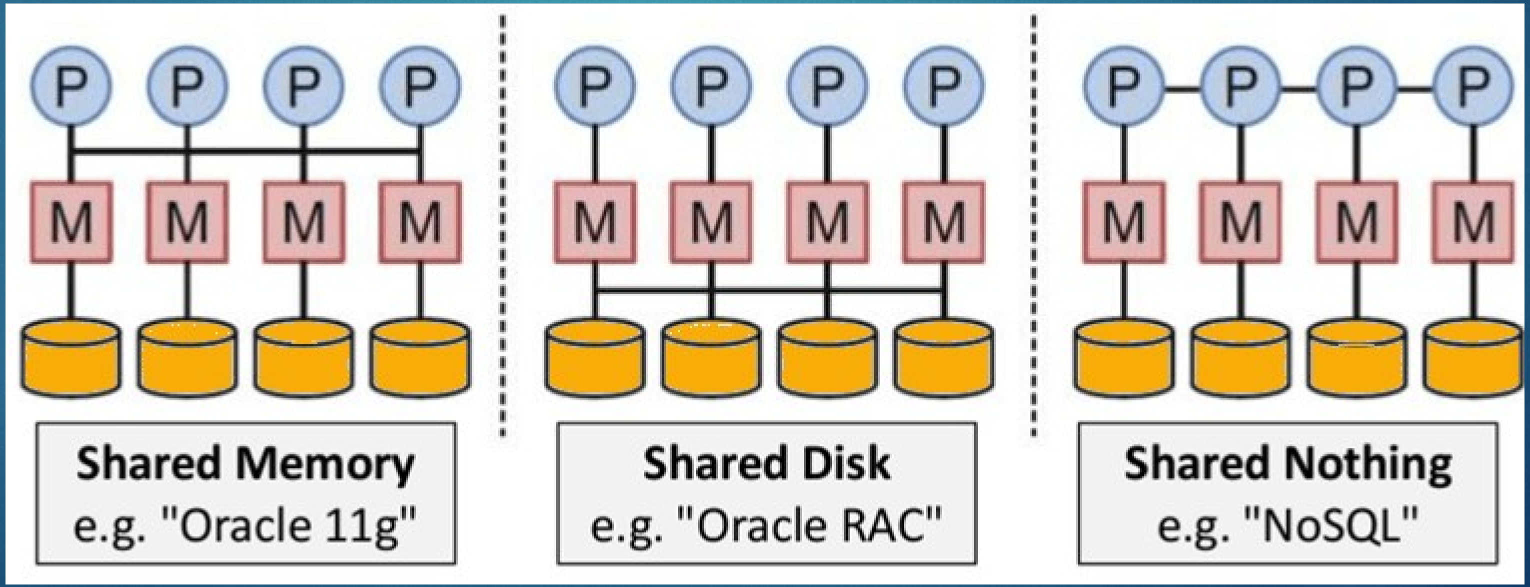


Simple API

- Offers easy to use interfaces for storage and querying data provided
- APIs allow low-level data manipulation & selection methods
- Text-based protocols mostly used with HTTP REST with JSON
- Mostly used no standard based NoSQL query language
- Web-enabled databases running as internet-facing services

Distributed

- Multiple NoSQL databases can be executed in a distributed fashion
- Offers auto-scaling and fail-over capabilities
- Often ACID concept can be sacrificed for scalability and throughput



Popular NoSQL Databases:

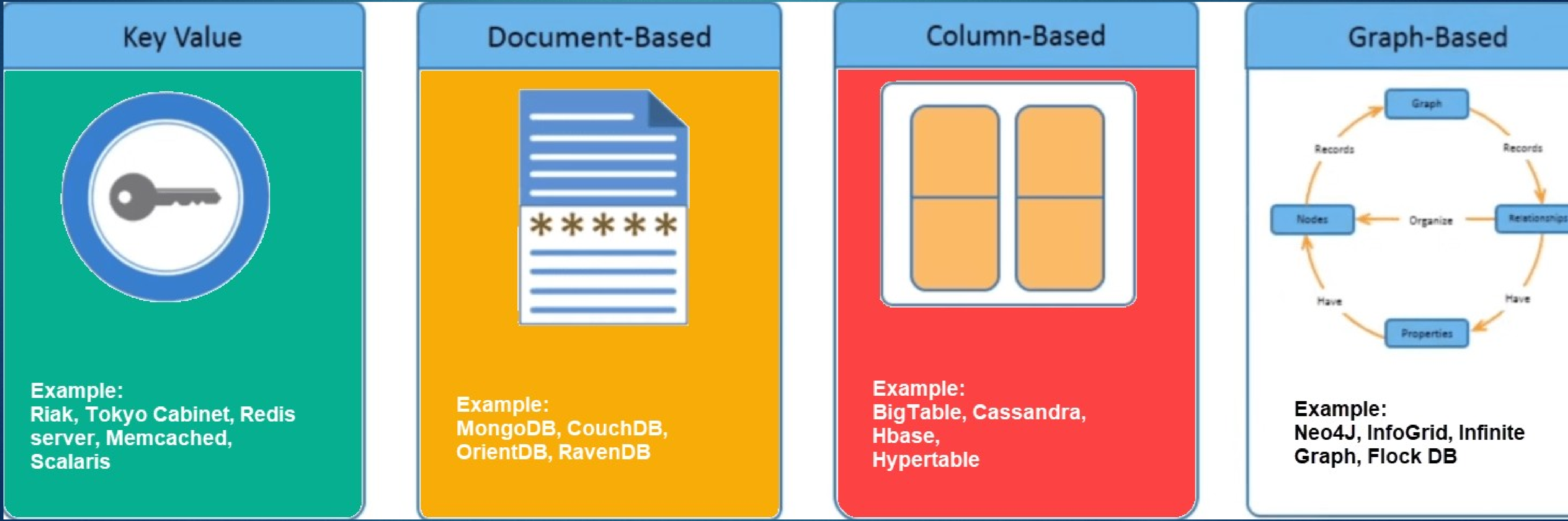
SQL vs. NoSQL Terminology

The following table compares terminology used by select NoSQL databases with terminology used by SQL databases.

SQL	MongoDB	DynamoDB	Cassandra	Couchbase
Table	Collection	Table	Table	Data bucket
Row	Document	Item	Row	Document
Column	Field	Attribute	Column	Field
Primary key	ObjectId	Primary key	Primary key	Document ID
Index	Index	Secondary index	Index	Index
View	View	Global secondary index	Materialized view	View
Nested table or object	Embedded document	Map	Map	Map
Array	Array	List	List	List

Types of NoSQL Databases

- **NoSQL Databases** are mainly categorized into four types:



Key Value

Example:
Riak, Tokyo Cabinet, Redis server, Memcached, Scalaris

Document-Based

Example:
MongoDB, CouchDB, OrientDB, RavenDB

Column-Based

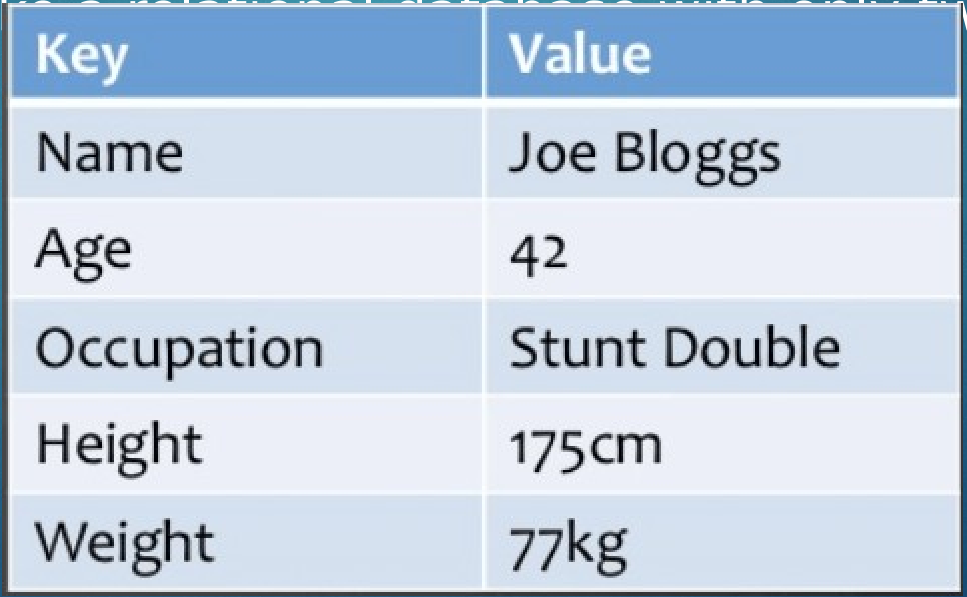
Example:
BigTable, Cassandra, Hbase, Hypertable

Graph-Based

Example:
Neo4J, InfoGrid, Infinite Graph, Flock DB

Key Value Pair Based

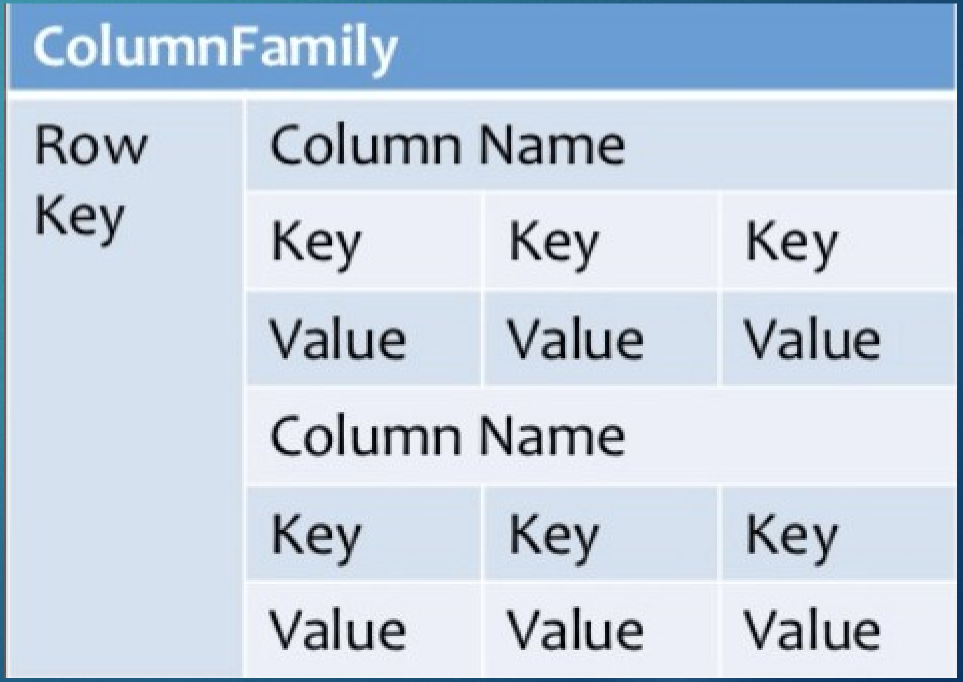
- Data is stored in key/value pairs. It is designed in such a way to handle lots of data and heavy load.
- Key-value pair storage databases store data as a hash table where each key is unique, and the value can be a JSON, BLOB(Binary Large Objects), string, etc.
- Key value stores help the developer to store schema-less data. They work best for shopping cart contents.
- A key-value store is like a relational database with only two columns which is the key and the value



Key	Value
Name	Joe Bloggs
Age	42
Occupation	Stunt Double
Height	175cm
Weight	77kg

Column-based

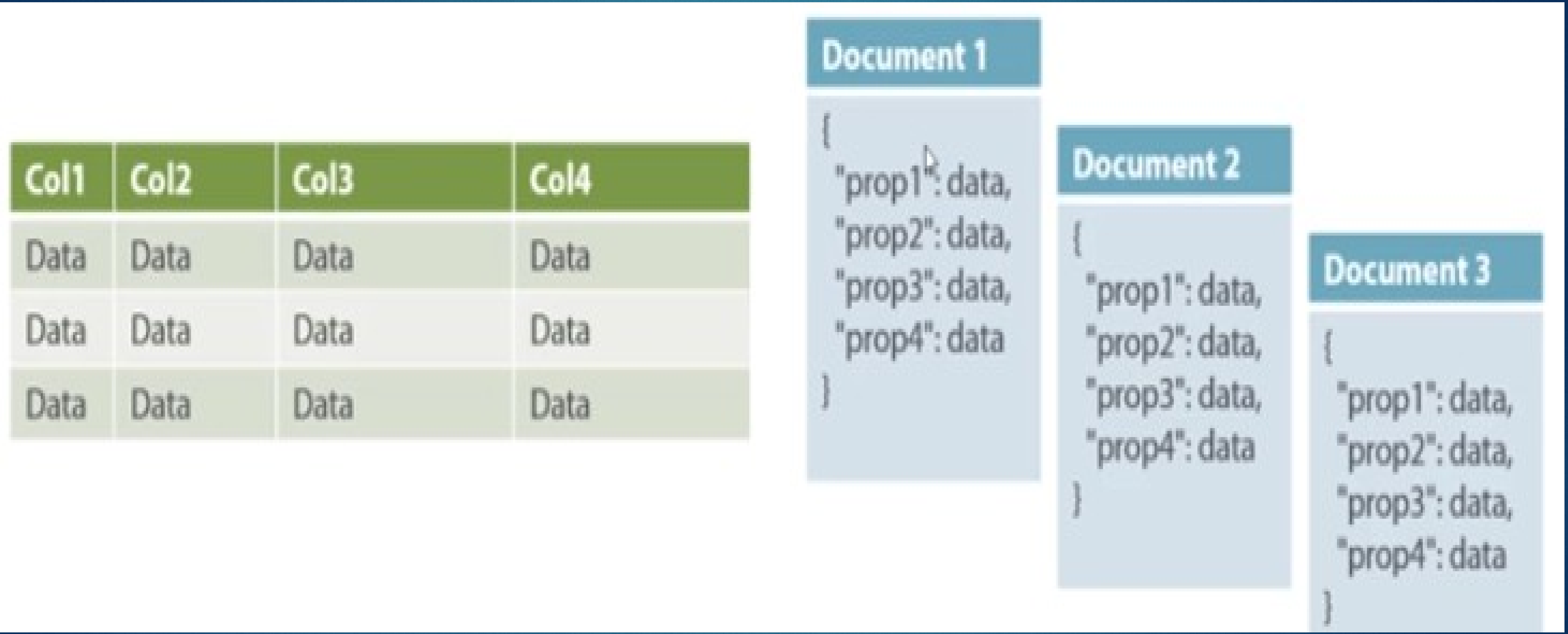
- A column-oriented database is a non-relational database that stores the data in columns instead of rows
- Columnar databases are designed to read data more efficiently and retrieve the data with greater speed. A columnar database is used to store a large amount of data.
- **Examples of Column Databases:**
  - Apache Cassandra
  - Hadoop HBase
  - HyperTable



ColumnFamily			
Row Key	Column Name		
	Key	Key	Key
	Value	Value	Value
	Column Name		
	Key	Key	Key
	Value	Value	Value

Document-Based Database:

- The document-based database is a nonrelational database. Instead of storing the data in rows and columns (tables), it uses documents to store the data in the database.
- A document database stores data in JSON, BSON, or XML documents.
- In the Document database, the particular elements can be accessed by using the index value that is assigned for faster querying.
- Documents are schema-free and therefore are flexible and easy to modify.
- **Examples of Document Databases:**
  - MongoDB
  - Couchbase
  - RavenDB



Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data

Document 1

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

Document 2

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

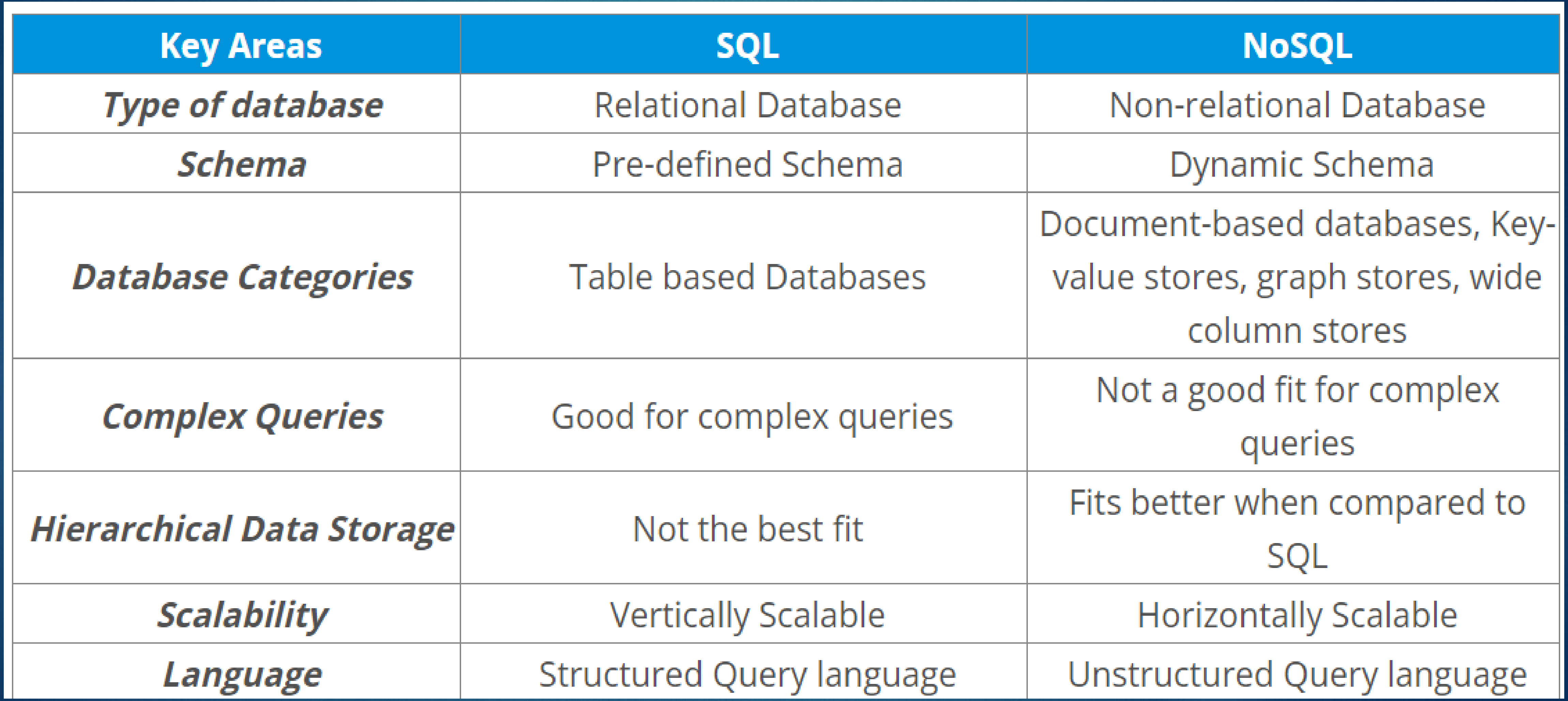
Document 3

```
{
  "prop1": data,
  "prop2": data,
  "prop3": data,
  "prop4": data
}
```

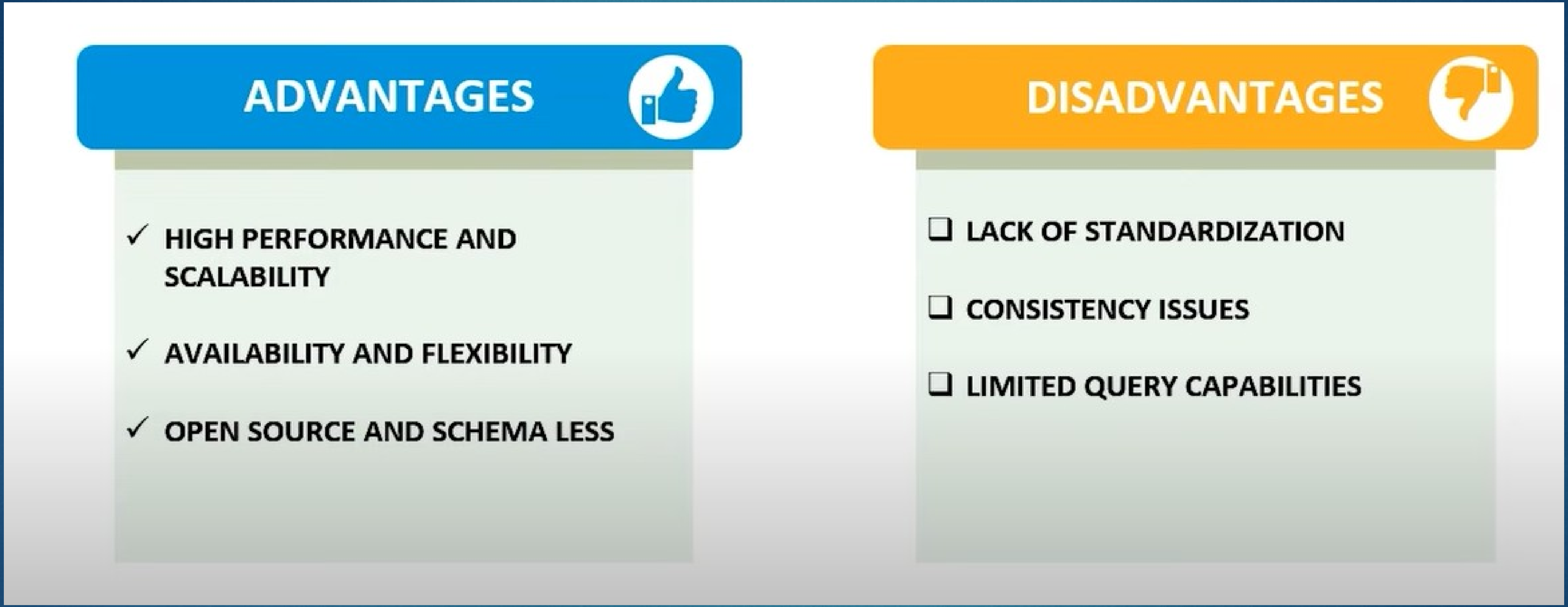
Graph-Based databases:

- Graph-based databases focus on the relationship between the elements. It stores the data in the form of nodes in the database. The connections between the nodes are called links or relationships.
- In a graph-based database, it is easy to identify the relationship between the data by using the links.
- **Example of graph databases:**
  - FlockDB
  - HyperGraphDB
  - Neo4j

SQL vs NoSQL:



Key Areas	SQL	NoSQL
Type of database	Relational Database	Non-relational Database
Schema	Pre-defined Schema	Dynamic Schema
Database Categories	Table based Databases	Document-based databases, Key-value stores, graph stores, wide column stores
Complex Queries	Good for complex queries	Not a good fit for complex queries
Hierarchical Data Storage	Not the best fit	Fits better when compared to SQL
Scalability	Vertically Scalable	Horizontally Scalable
Language	Structured Query language	Unstructured Query language



ADVANTAGES

- ✓ HIGH PERFORMANCE AND SCALABILITY
- ✓ AVAILABILITY AND FLEXIBILITY
- ✓ OPEN SOURCE AND SCHEMA LESS

DISADVANTAGES

- ❑ LACK OF STANDARDIZATION
- ❑ CONSISTENCY ISSUES
- ❑ LIMITED QUERY CAPABILITIES

Query Mechanism tool of NoSQL

- The REST-based retrieval of a value based on its key/ID with the GET resource is the most frequent data retrieval mechanism.
- Because they grasp the value in a key-value combination, document store databases allow for more challenging searches. With MapReduce, CouchDB, for example, allows you to define views.

What is the CAP Theorem?

- Brewer's theorem is another name for the CAP theorem. It claims that a distributed data store cannot provide more than two out of three guarantees.
  - 1)Consistency
  - 2)Availability
  - 3)Partition Tolerance

1)Consistency: Even after an operation has been completed, the data should stay consistent. This indicates that once data is written, it should be included in any subsequent read requests. After altering the order status, for example, all clients should be able to see the same information.

2)**Availability**: The database should be accessible and responsive at all times. There should be no downtime.

3)**Partition** Tolerance: Partition Tolerance means that the system should keep working even if the connection between the servers isn't always reliable. The servers, for example, can be divided into several groups that may or may not communicate with one another. If one portion of the database is down, the other sections are always up and running.