###### WEB TECHNOLOGIES

###### UNIT-3

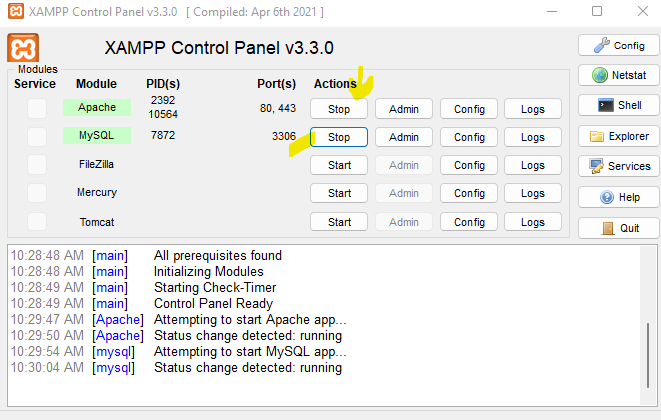
###### UNIT-III

**Database Connectivity**: Introduction to SQL: Connect, create database, create table, insert, prepared statements. Use of NoSQL: Introduction to NoSQL, Difference between SQL and NoSQL, Types of NoSQL Databases, Query mechanism tools for NoSQL.

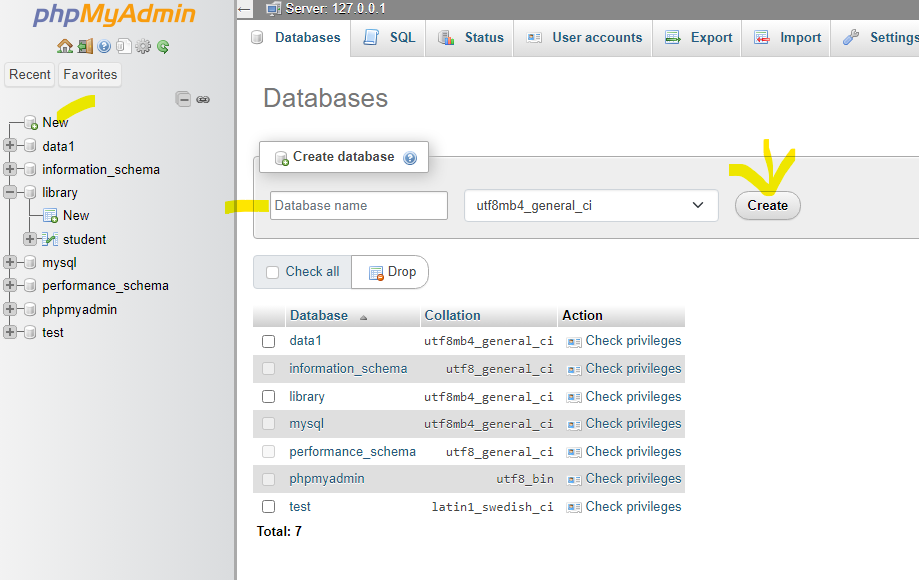
**Authentication:** Google OAuth: Basic Steps. Access to Google APIs: For Server-side Web apps, for Java Script Web apps, for Mobile & Desktop apps

**PHP and MYSQL Connectivity**

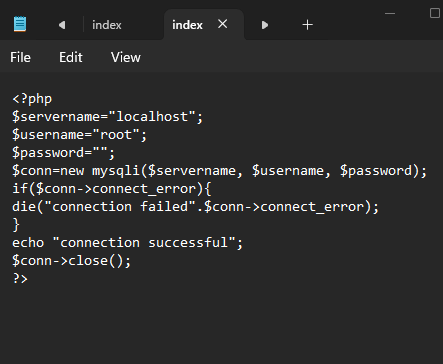
1. Open the XAMPP control panel. Start the Apache and Mysql.



1. Go to default browser and type in the url section as **localhost/phpmyadmin/**



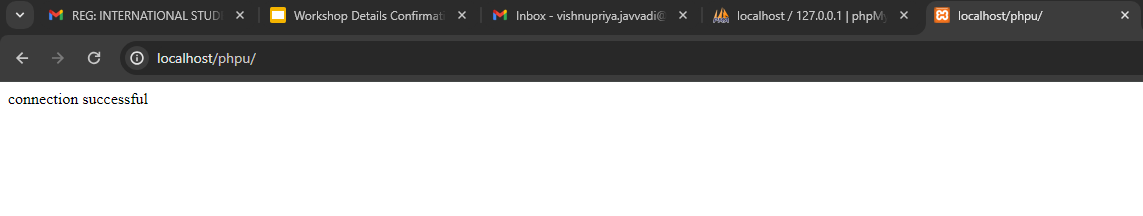
1. Create new database.
2. Go to any text editor and create the following file.



Save this file as “index.php”

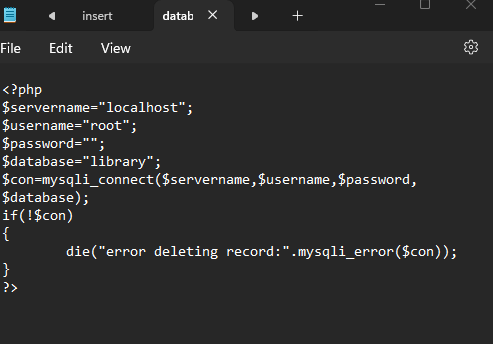
C:/xampp/htdocs/phpu/index.php

1. Go to browser> localhost/phpu/



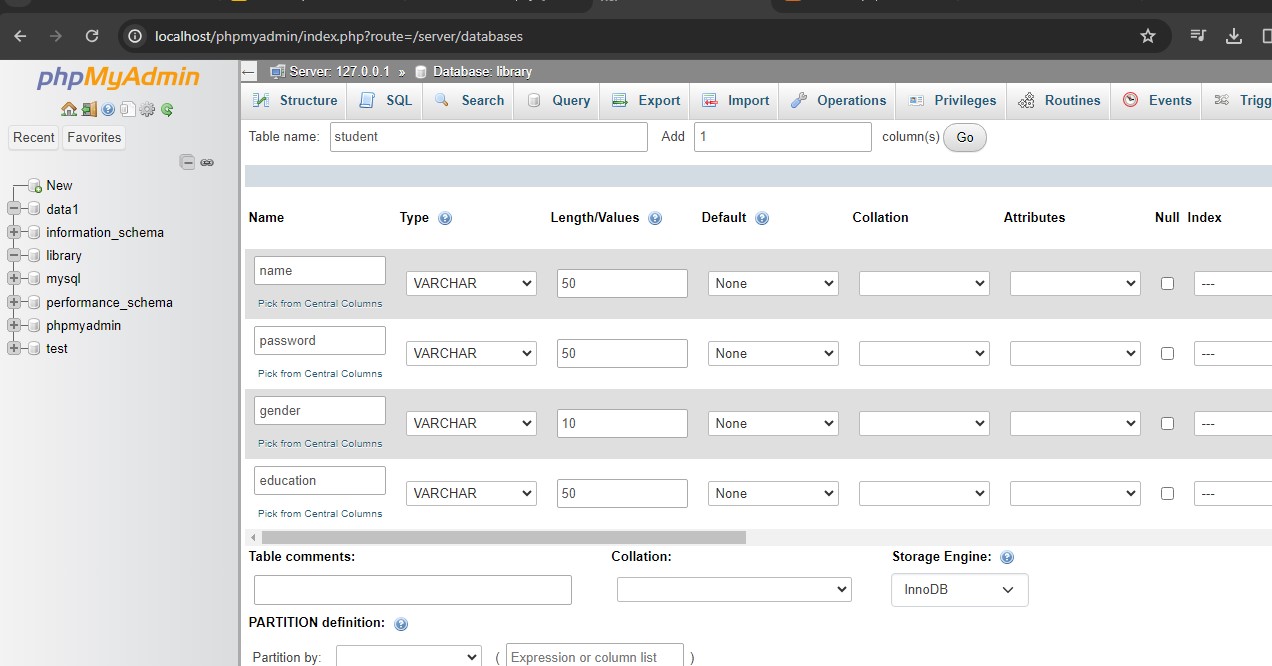
**HOW TO STORE THE DATA IN THE DATABASE?**

1. Xampp>htdocs>phpu>database.php

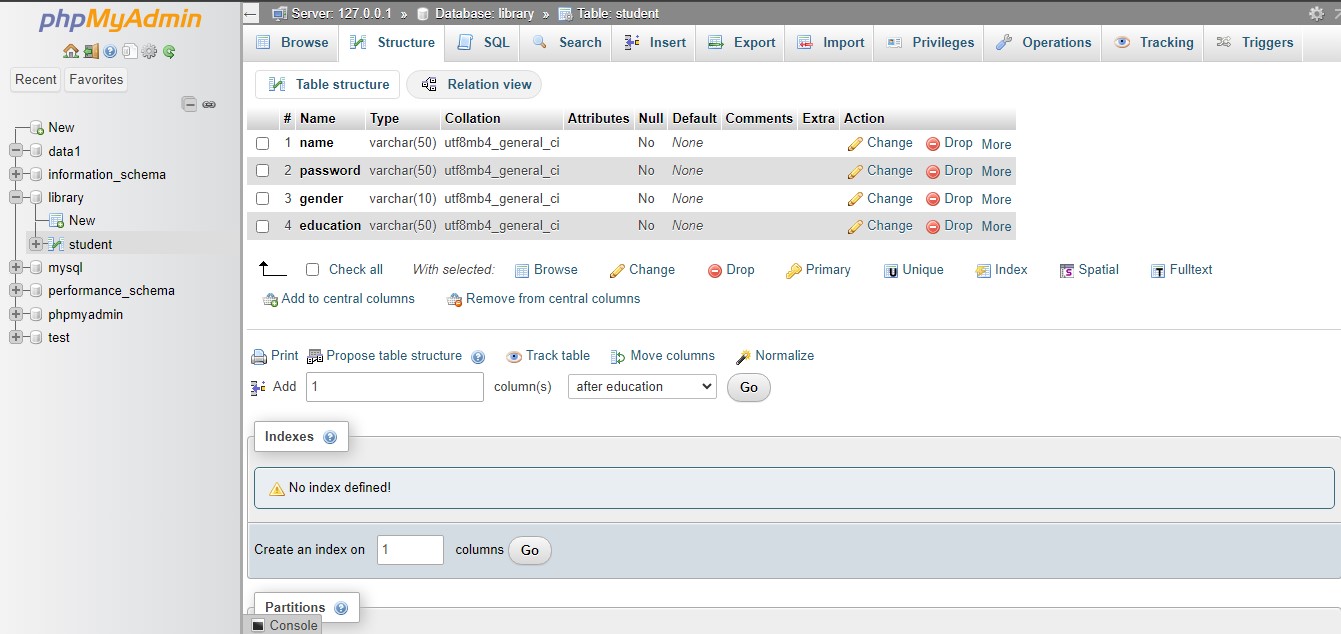
****

1. Go to localhost/phpmyadmin/

As we have created a **database** called **library** in that we need to create **table** named **student** and insert data



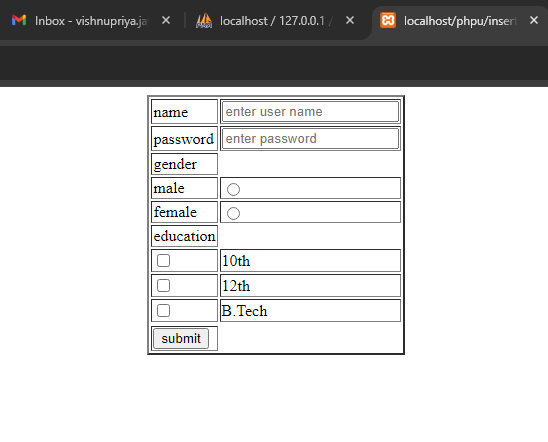
1. And save the data then



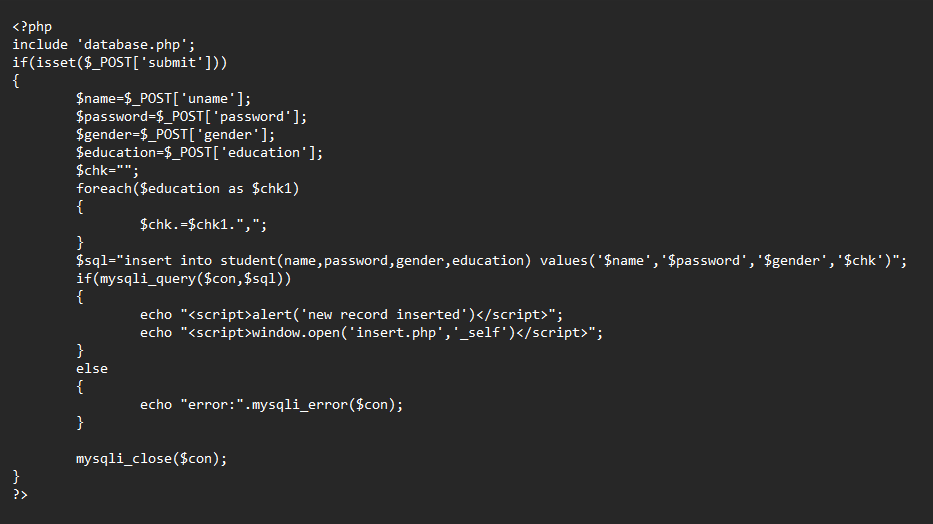
1. Now we need to create a form name it as “insert.php”



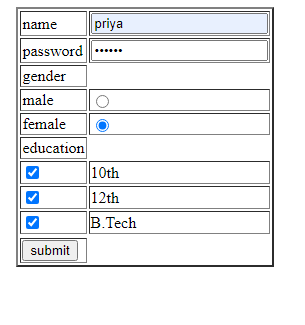
After executing the above code we will get the output as:



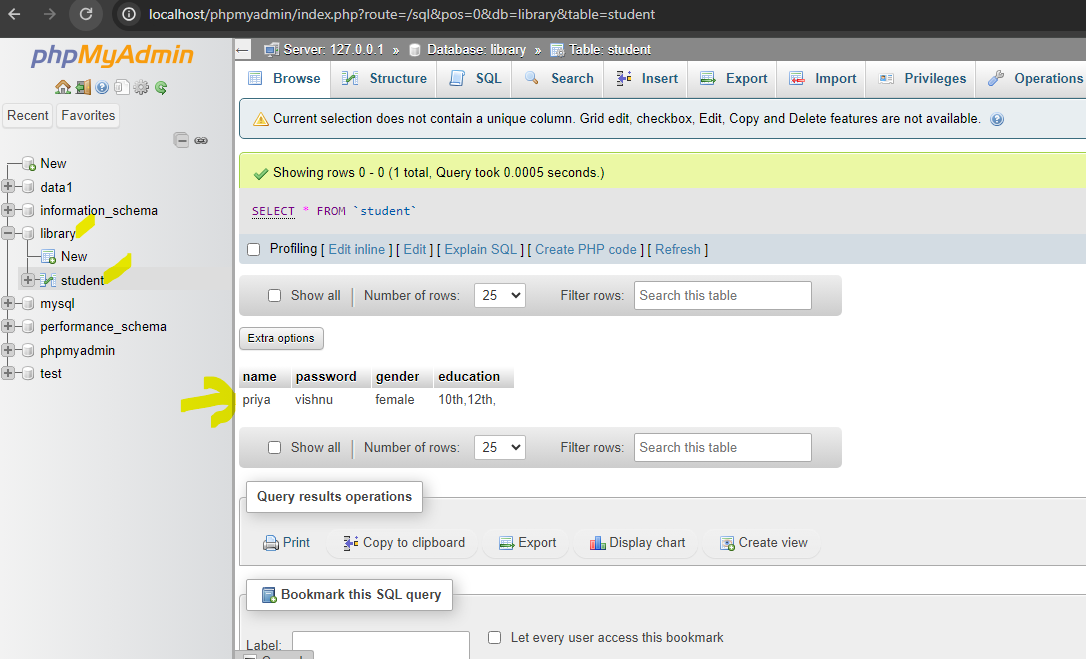
1. For storing the data in the database we need to create a new file as “process.php”



1. After the completion of “process.php” go to localhost/phpu/insert.php which means execute “insert.php” enter the data in the form and submit.



1. Now go to “ localhost/phpmyadmin/ “ and go to the database we have created “library” open the table “student” then we will observe new record of data that we have entered in the form.



**PREPARED STATEMENTS**

**PHP Prepared Statements** are a powerful feature that enhances the security and performance of database interactions.

They provide a way to separate SQL logic from the data being passed into queries. With prepared statements, SQL queries are precompiled and stored on the database server, allowing for efficient execution and reuse.

Prepared statements are an essential tool for developing secure and efficient database-driven applications in PHP.

In traditional SQL queries, input values are directly concatenated into the query string, which can lead to malicious data altering the query's structure or unintended execution.

Prepared statements, on the other hand, work by separating the query logic from the data values. The query is precompiled and sent to the database server, where it is optimized and cached.

Placeholders are used to represent the input data, and the actual values are bound to these placeholders when executing the statement.

By using php-prepared statements, the database server can distinguish between the query structure and the data values. This separation eliminates the risk of SQL injection, as the database treats the input values as data, not executable code.

The values are automatically sanitized or escaped, ensuring that they do not alter the query's intended functionality.

A prepared statement (also known as parameterized statement) is simply a SQL query template containing placeholder instead of the actual parameter values. These placeholders will be replaced by the actual values at the time of execution of the statement.

MySQLi supports the use of anonymous positional placeholder (?), as shown below:

INSERT INTO persons (first\_name, last\_name, email) VALUES (?, ?, ?);

The prepared statement execution consists of two stages: prepare and execute.

* **Prepare** — At the prepare stage a SQL statement template is created and sent to the database server. The server parses the statement template, performs a syntax check and query optimization, and stores it for later use.
* **Execute** — During execute the parameter values are sent to the server. The server creates a statement from the statement template and these values to execute it.

Prepared statements are very useful, particularly in situations when you execute a particular statement multiple times with different values, for example, a series of INSERT statements.

**Advantages of Using Prepared Statements**

A prepared statement can execute the same statement repeatedly with high efficiency, because the statement is parsed only once again, while it can be executed multiple times.

It also minimize bandwidth usage, since upon every execution only the placeholder values need to be transmitted to the database server instead of the complete SQL statement.

Prepared statements also provide strong protection against [SQL injection](https://www.tutorialrepublic.com/sql-tutorial/sql-injection.php), because parameter values are not embedded directly inside the SQL query string.

The parameter values are sent to the database server separately from the query using a different protocol and thus cannot interfere with it.

The server uses these values directly at the point of execution, after the statement template is parsed.

That's why the prepared statements are less error-prone and thus considered as one of the most critical element in database security.

The following example shows you how prepared statements actually work:

<?php

$servername = "localhost";

$username = "root";

$password = "";

$dbname = "myDB";

// Create connection

$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection

if ($conn->connect\_error) {

die("Connection failed: " . $conn->connect\_error);

}

// prepare and bind

$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)");

$stmt->bind\_param("sss", $firstname, $lastname, $email);

// set parameters and execute

$firstname = "vishnu";

$lastname = "priya";

$email = "priyajv34@gmail.com";

$stmt->execute();

$firstname = "priya";

$lastname = "jv";

$email = "jv@gmail.com";

$stmt->execute();

$firstname = "Julie";

$lastname = "Dooley";

$email = "julie@example.com";

$stmt->execute();

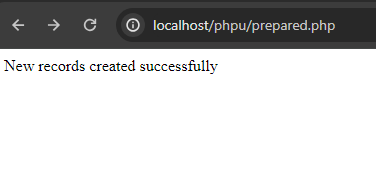
echo "New records created successfully";

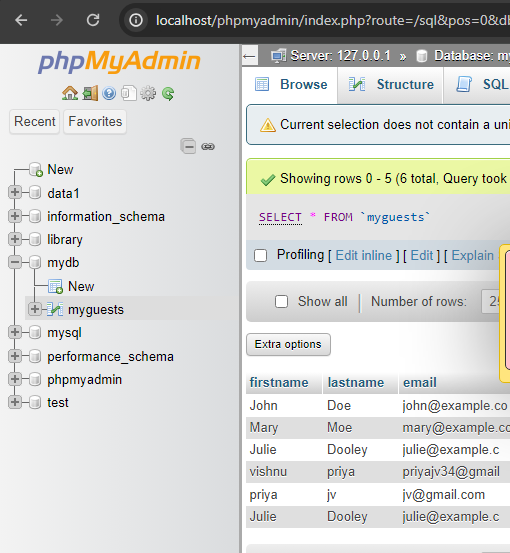
$stmt->close();

$conn->close();

?>

As you can see in the above example we've prepared the INSERT statement just once but executed it multiple times by passing the different set of parameters.





**Explanation of Code (Procedural style)**

Inside the SQL INSERT statement (*line no-12*) of the example above, the question marks is used as the placeholders for the *first\_name*, *last\_name*, *email* fields values.

The mysqli\_stmt\_bind\_param() function (*line no-16*) bind variables to the placeholders (?) in the SQL statement template. The placeholders (?) will be replaced by the actual values held in the variables at the time of execution. The *type* definition string provided as second argument i.e. the "sss" string specifies that the data type of each bind variable is string.

The type definition string specifies the data types of the corresponding bind variables and contains one or more of the following four characters:

* **b** — binary (such as image, PDF file, etc.)
* **d** — double (floating point number)
* **i** — integer (whole number)
* **s** — string (text)

The number of bind variables and the number of characters in type definition string must match the number of placeholders in the SQL statement template.

**NOSQL**



NoSQL stands for ***Not Only SQL***, meaning that NoSQL databases have the specificity of not being relational because they can store data in an unstructured format.

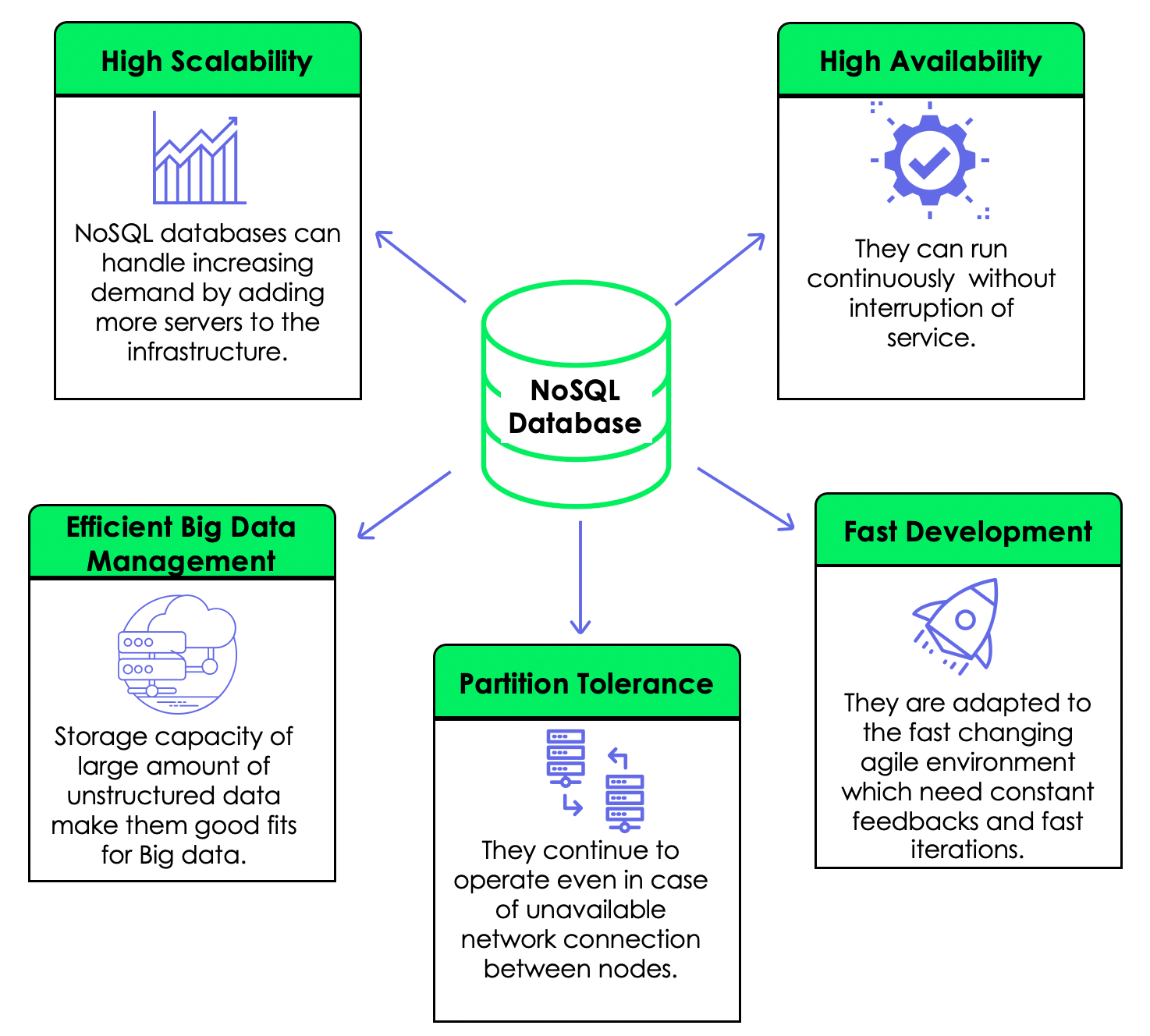
Traditional SQL databases have been used as the only type of database for years. However, due to the extreme popularity of the internet in the mid-1990s and digital transformation, a new data type became prominent: ***NoSQL databases***. They were introduced in response to the weakness of traditional SQL databases. NoSQL databases allow developers to store huge amounts of unstructured data, giving them a lot of flexibility.

NoSQL databases emerged in the late 2000s as the cost of storage dramatically decreased. Gone were the days of needing to create a complex, difficult-to-manage data model in order to avoid data duplication. Developers (rather than storage) were becoming the primary cost of software development, so NoSQL databases optimized for developer productivity.

As storage costs rapidly decreased, the amount of data that applications needed to store and query increased. This data came in all shapes and sizes — [structured, semi-structured,](https://www.mongodb.com/unstructured-data) and [polymorphic](https://www.mongodb.com/developer/how-to/polymorphic-pattern/) — and defining the schema in advance became nearly impossible.

**What are NoSQL databases?**

NoSQL stands for ***Not Only SQL***, meaning that NoSQL databases have the specificity of not being relational because they can store data in an unstructured format. The following graphic highlights the main five key features of NoSQL databases.



**Why are NoSQL databases important?**

NoSQL databases have become popular in the industry because of the following benefits:

* **Multi-mode data**: NoSQL databases offer more flexibility than traditional SQL databases because they can store structured (e.g. data captured from sensors), unstructured (images, videos, etc.), and semi-structured (XML, JSON, etc.) data.
* **Easy scalability**: this is made simple because of their peer-to-peer architectures, meaning multiple machines can be added to the architecture.
* **Global availability**: this makes it possible to access the same data simultaneously through different machines from different geographical zones because the database is shared globally.
* **Flexibility**: NoSQL databases can rapidly adapt to changing requirements with frequent updates and new features.

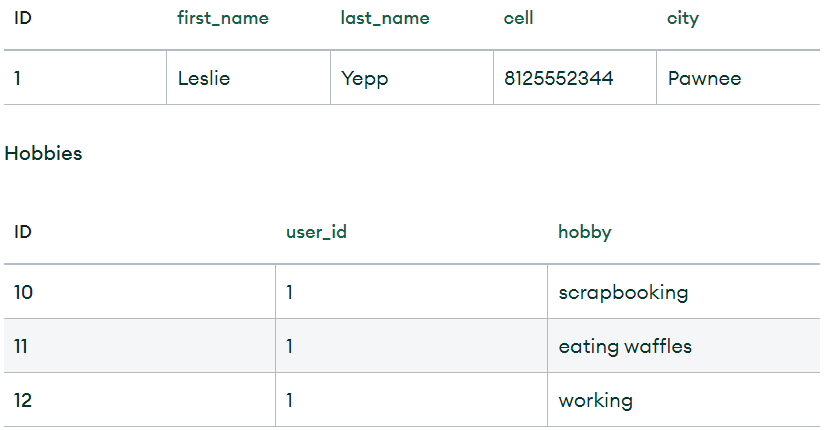
**NoSQL Databases vs. SQL Databases**

|  |  |  |
| --- | --- | --- |
|  | **SQL Databases** | **NoSQL Databases** |
| **Language** | SQL databases use structured query languages to perform operations, requiring the use of predefined schema to better interact with the data. | On the other hand, NoSQL databases use a dynamic schema to query data. Also, some NoSQL databases use SQL-like syntax for document manipulation. |
| **Data Schema** | SQL databases have a predefined and fixed format, which cannot be changed for new data. | NoSQL databases are more flexible. This flexibility means that records in the databases can be created without having a predefined structure, and each record has its own structure. |
| **Scalability** | SQL databases are only vertically scalable, meaning that a single machine needs to increase CPU, RAM, SSD, at a certain level to meet the demand. | NoSQL databases are horizontally scalable, meaning that additional machines are added to the existing infrastructure to satisfy the storage demand. |
| **Big Data Support** | The vertical scaling makes it difficult for SQL databases to store very big data (petabytes). | The horizontal scaling and dynamic data schema make NoSQL suitable for big data. Also, NoSQL databases were developed by top internet companies (Amazon, Google, Yahoo, etc.) to face the challenges of the rapidly increasing amount of data. |
| **Properties** | SQL databases use the ACID (Atomicity, Consistency, Isolation, Durability) property. | NoSQL databases, on the other hand, use the CAP (Consistency, Availability, Partition Tolerance) property. |

While a variety of differences exist between relational database management systems (RDBMS) and NoSQL databases, one of the key differences is the way the data is modeled in the database.

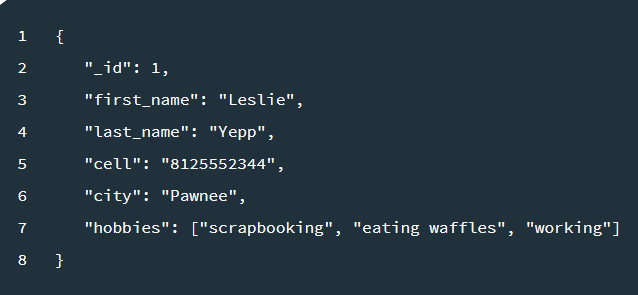
###### RDBMS vs NoSQL: Data Modeling Example

Let's consider an example of storing information about a user and their hobbies. We need to store a user's first name, last name, cell phone number, city, and hobbies. In a relational database, we'd likely create two tables: one for Users and one for Hobbies. USERS



In order to retrieve all of the information about a user and their hobbies, information from the Users table and Hobbies table will need to be joined together.

The data model we design for a NoSQL database will depend on the type of NoSQL database we choose. Let's consider how to store the same information about a user and their hobbies in a [document database](https://www.mongodb.com/document-databases) like MongoDB.



In order to retrieve all of the information about a user and their hobbies, a single document can be retrieved from the database. No joins are required, resulting in faster queries.

## When should NoSQL databases be used?

In this fast-growing and competitive environment, industries need to collect as much data as possible to satisfy their business goals. Collecting data is one thing, but storing them in the right infrastructure is another challenge. The difficulty comes because data can be of different types such as images, videos, text, and sounds. Using relational databases to store these different data types is not always a smart move. However, the question remains:

**When to use NoSQL instead of SQL?**

You should consider using NoSQL when you are in the following scenario:

* **Constant changing of data**: when you do not know how your system or applications will grow in the future, meaning that you might want to add new data types, new functions, etc.
* **A lot of data**: when your business is dealing with huge data that might grow over time.
* **No consistency**: when data consistency and 100% integrity are not your priority. For example, when you develop a social media platform for your business, all the employees seeing your posts at once might not be an issue.
* **Scalability and cost**: NoSQL databases allow greater flexibility and can control costs as your data needs change.

## 4 Main Types of NoSQL Databases

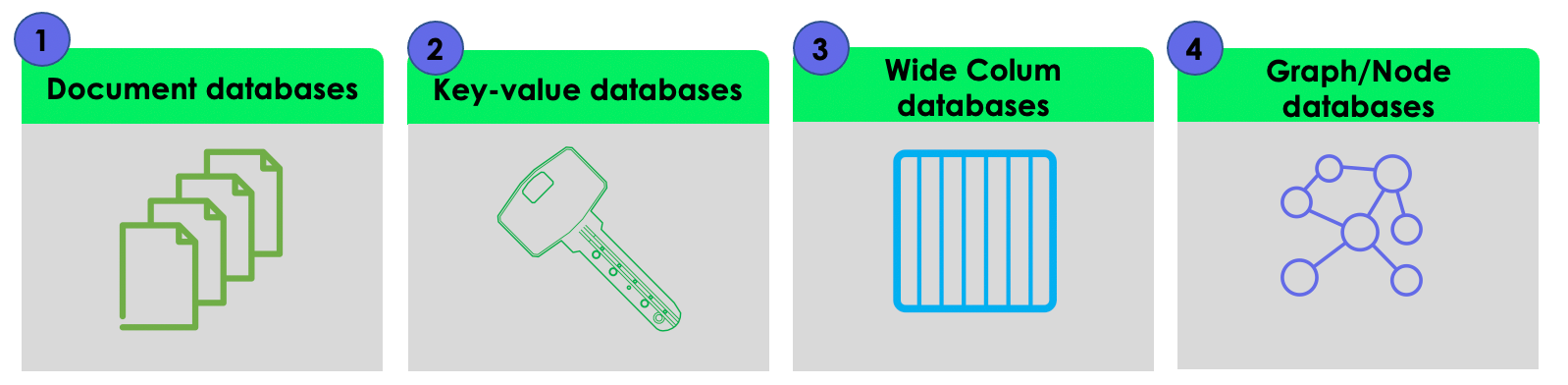
NoSQL databases are divided into four main categories.

1. Document Databases

2. Key-Value Databases

3. Wide column Databases

4. Graph/Node Databases



### 1. Document Databases

#### A document database (also known as a document-oriented database or a document store) is a database that stores information in documents.

## What are documents?

A document is a record in a document database. A document typically stores information about one object and any of its related metadata.

Documents store data in field-value pairs. The values can be a variety of types and structures, including strings, numbers, dates, arrays, or objects. Documents can be stored in formats like JSON, [BSON](https://www.mongodb.com/json-and-bson), and XML.

Below is a JSON document that stores information about a user named Tom.



### Collections

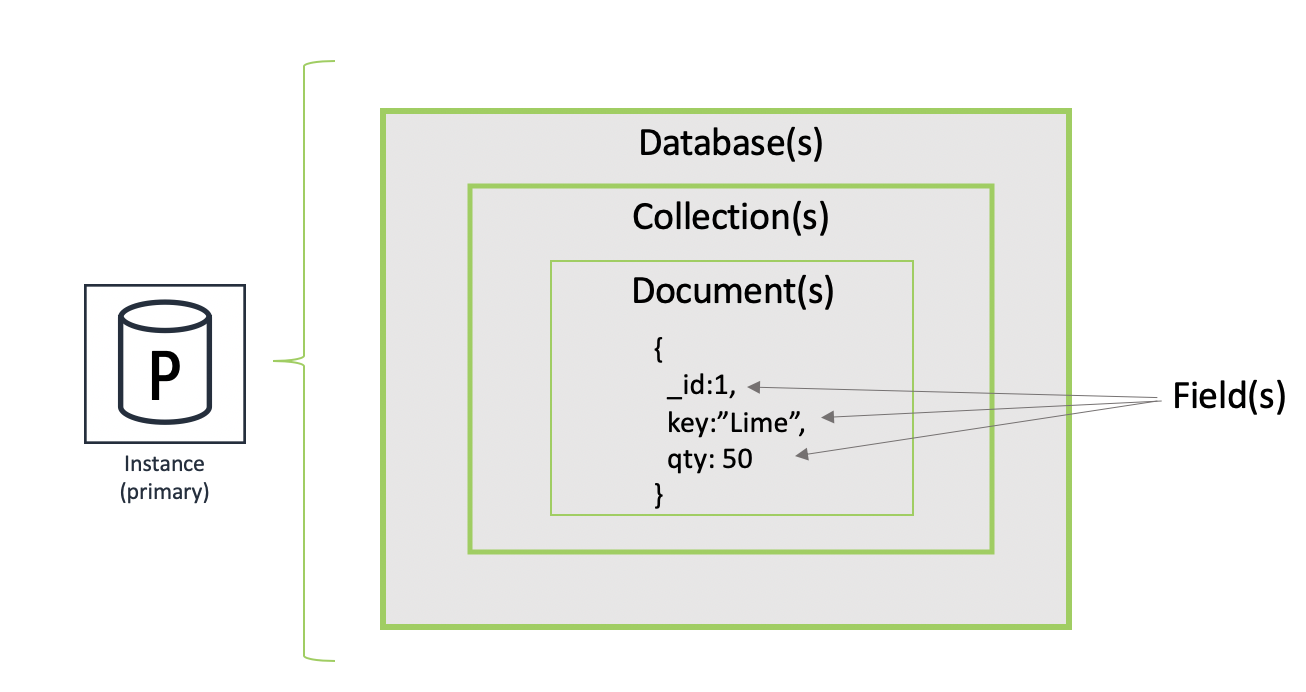
A collection is a group of documents. Collections typically store documents that have similar contents.

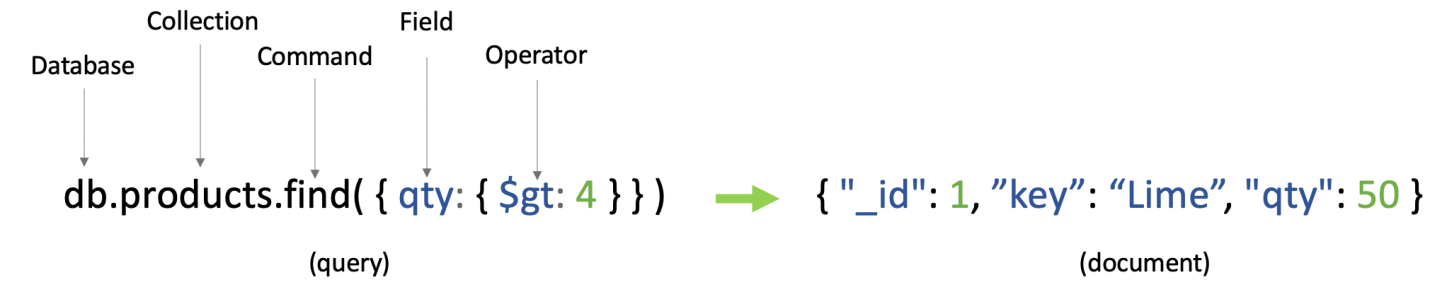
Not all documents in a collection are required to have the same fields, because document databases have a flexible schema. Note that some document databases provide [schema validation](https://docs.mongodb.com/manual/core/schema-validation/), so the schema can optionally be locked down when needed.

Continuing with the example above, the document with information about Tom could be stored in a collection named users. More documents could be added to the users collection in order to store information about other users. For example, the document below that stores information about Donna could be added to the users collection.



Note that the document for Donna does not contain the same fields as the document for Tom. The users collection is leveraging a flexible schema to store the information that exists for each user.





### Document database operations

You can create, read, update, and delete entire documents stored in the database. Document databases provide a query language or API that allows developers to run the following operations:

#### Create

You can create documents in the database. Each document has a unique identifier that serves as a key.

#### Read

You can use the API or query language to read document data. You can run queries using field values or keys. You can also add indexes to the database to increase read performance.

#### Update

You can update existing documents flexibly. You can rewrite the entire document or update individual values.

## Advantages of document databases

Document databases enable flexible indexing, powerful ad hoc queries, and analytics over collections of documents.

### Ease of development

JSON documents map to objects—a common data type in most programming languages. When building applications, developers can flexibly create and update documents directly from the code. This means they spend less time creating [data models](https://aws.amazon.com/what-is/data-modeling/) beforehand. Therefore, application development is more rapid and efficient.

### Flexible schema

A document-oriented database allows you to create multiple documents with different fields within the same collection. This can be handy when storing unstructured data like emails or social media posts. However, some document databases offer schema validation, so you can impose some restrictions on the structure.

### Performance at scale

Document databases offer built-in distribution capabilities. You can scale them horizontally across multiple servers without impacting performance, which is cost-efficient as well. Moreover, document databases provide fault tolerance and availability through built-in replication.

A four tiered diagram:
1. Document Model
2. Key-Value Pairs, Relational, Objects, Graph, Geospatial
3. Unified Interface
4. Transactional, Search, Mobile, Real-Time Analytics, Data Lake

The document model is a superset of other data models

Due to their rich data modeling capabilities, document databases are general-purpose databases that can store data for a variety of use cases.

**use cases of document databases**

The document model works well with use cases such as content management, catalogs, sensor management, and more. For each use case, each document is unique and evolves over time.

### Content management

A document database is an excellent choice for content management applications such as blogs and video platforms. With a document database, each entity the application tracks can be stored as a single document. The document database is a more intuitive way for a developer to update an application as the requirements evolve.

In addition, if the data model needs to change, only the affected documents need to be updated. No schema update is required and no database downtime is necessary to make the changes.

### Catalogs

Document databases are efficient and effective for storing catalog information. For example, in an e-commerce application, different products usually have different numbers of attributes. Managing thousands of attributes in [relational databases](https://aws.amazon.com/relational-database/) is inefficient, and the reading performance is affected. Using a document database, each product’s attributes can be described in a single document for easy management and faster reading speed. Changing the attributes of one product won’t affect others.

### Sensor management

The [Internet of Things (IoT)](https://docs.aws.amazon.com/whitepapers/latest/aws-overview/internet-of-things-services.html) has resulted in organizations regularly collecting data from smart devices like sensors and meters. Sensor data typically comes in as a continuous stream of variable values. Due to latency issues, some data objects might be incomplete, duplicated, or missing. Additionally, you must collect a large volume of data before you can filter or summarize it for analytics.

Document stores are more convenient in this case. You can quickly store the sensor data as it is, without cleaning it or making it conforms to pre-determined schemas. You can also scale it as required and delete entire documents once analytics is done.

**2. KEY VALUE DATABASE**

A [key-value database](https://aws.amazon.com/dynamodb/) is a type of non-relational database, also known as [NoSQL database](https://aws.amazon.com/nosql/), that uses a simple key-value method to store data. It stores data as a collection of key-value pairs in which a key serves as a unique identifier. Both keys and values can be anything, ranging from simple objects to complex compound objects. Key-value databases (or key-value stores) are highly partitionable and allow horizontal scaling at a level that other types of databases cannot achieve.

**Advantages of key-value databases**

### Scalability

Many key-value databases provide built-in support for advanced scaling features. They scale horizontally and automatically distribute data across servers to reduce bottlenecks at a single server.

### Ease of use

Instead of mapping their code objects to multiple underlying tables, engineers can create key-value pairs that match their code objects. This makes key-value stores more intuitive for developers to use.

### Performance

key-value databases don't have to perform resource-intensive table joins, which makes them much faster.

Use cases of key-value databases

### Session management

A session-oriented application, such as a web application, starts a session when a user logs in to an application and is active until the user logs out or the session times out. During this period, the application stores all user session attributes either in the main memory or in a database. User session data may include profile information, messages, personalized data and themes, recommendations, targeted promotions, and discounts.

Each user session has a unique identifier. Session data is never queried by anything other than a primary key, so a fast key-value store is a better fit for session data. In general, key-value databases may provide smaller per-page overhead than relational databases.

### Shopping cart

An e-commerce website may receive billions of orders per second during the holiday shopping season. A key-value database can handle the scaling of large amounts of data and extremely high volumes of state changes, while also servicing millions of simultaneous users through distributed processing and storage. Key-value stores also have built-in redundancy, which can handle the loss of storage nodes.

### Metadata storage engine

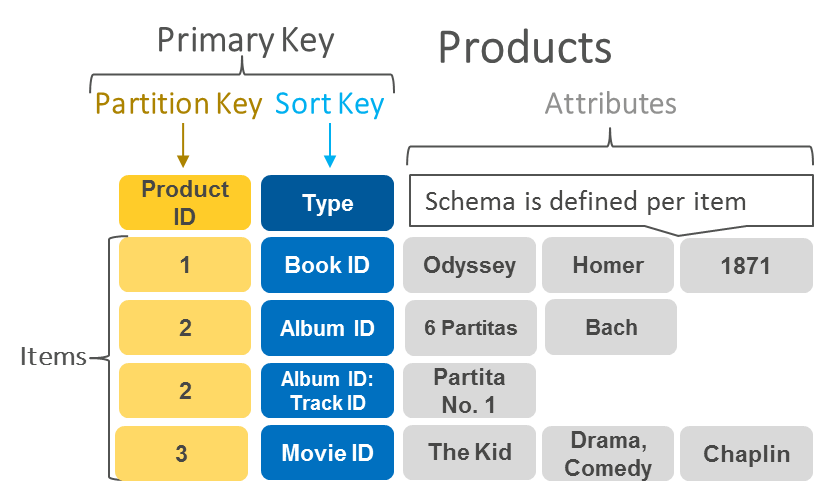
Your key-value store can act as an underlying storage layer for higher levels of data access. For example, you can scale throughput and concurrency for media and entertainment workloads such as real-time video streaming and interactive content. You can also build out your game platform with player data, session history, and leaderboards for millions of concurrent users.

### Caching

You can use a key-value database for storing data temporarily for faster retrieval. For example, social media applications can store frequently accessed data like news feed content. In-memory data caching systems also use key-value stores to accelerate application responses.

**How do key-value databases work**

Key-value databases work by organizing all data as a set of key-value pairs. You can think of the key as a question and the value as the answer to the question. In the example below, the primary key is a composite of two keys, Product ID and Type. The Product ID is the partition key which describes the partition in which the item will be stored. The Type is the sort key, which determines the order in which items will be stored on disk. The combination of the Partition Key and the Sort Key forms a unique primary key, which maps to a single value in the database.



In this example, the data object book has attributes like title, author, and publishing date. Every book data object has a key called BookID. You can directly link the BookID and associated book object in the key-value store. In addition, you can retrieve data by looking up the BookID in the table. Also, each item has its own schema, making key-value stores highly flexible for storing data of varying structures.

**FEATURES OF KEY-VALUE DATABASE:**

### Support for complex data types

Key-value stores provide support for defined data types like integers and text. However, many of them can also support more complex objects like arrays, nested dictionaries, images, videos, and semi-structured data. By giving the database more information about your data, there is room for more storage and query performance optimization.

### No need for table joins

Key-value databases don't need to perform any resource-intensive table joins. Their flexibility accommodates all the needed information in a single table. This is one of the reasons key-value stores perform so well.

### Sorted keys

A key-value store can sort keys so that data is stored systematically and for implementing partitioning. For example, keys may be sorted:

* Alphabetically or numerically
* Chronologically
* By data size

Consider a key-value store that uses the customer's email address as the unique key. Email addresses can be sorted alphabetically, so all data for A-J email lists are stored on server 1, K-S on server 2, and so on.

### Secondary key support

Some key-value stores allow you to define two or more different keys or secondary indexes to access the same data. For example, you can store customer data by key email address and key phone number.

### Replication

Many key-value stores offer built-in replication support by automatically copying data across multiple storage nodes. This helps with auto-recovery from disasters; you still have your data in case of server failure.

### Partitioning

Partitioning is how you distribute data across nodes. Many key-value databases provide default partitioning options. Some also give you the option to define input parameters for your partitions. For example, you could partition numerical keys into groups of 1000. Advanced key-value databases also provide automatic support for distributing your key-value database across multiple geographical locations. This improves application availability and reliability because you can respond to queries close to the user's location.

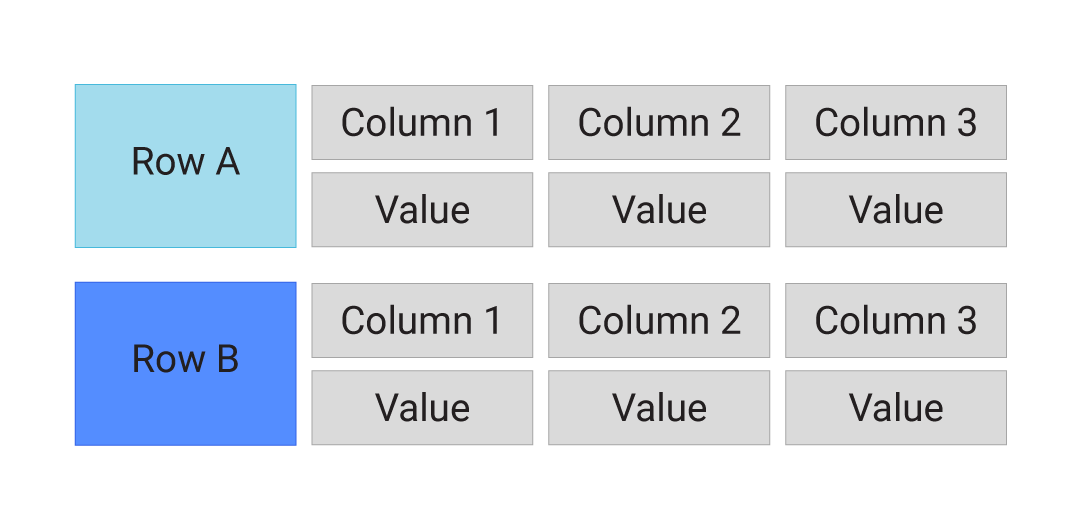
### ACID support

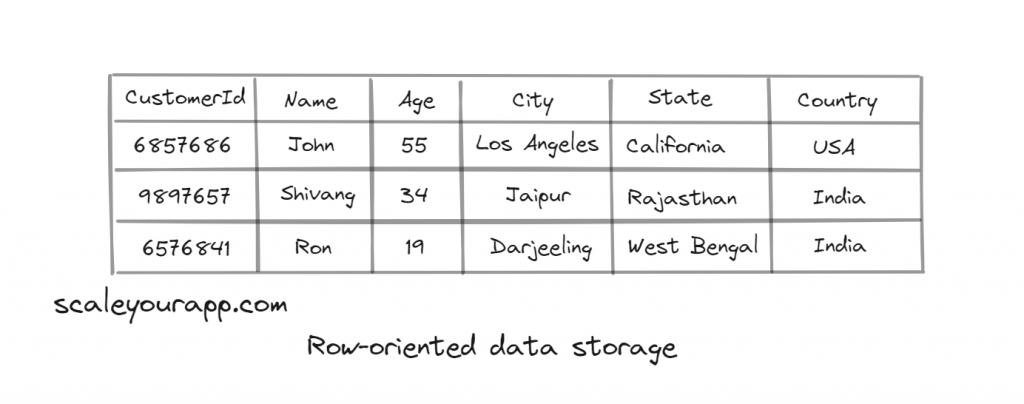
Atomicity, Consistency, Isolation, and Durability ([ACID](https://docs.aws.amazon.com/athena/latest/ug/acid-transactions.html)) are database properties that ensure data accuracy and reliability in all circumstances. For instance, if you are making multiple changes to your data in a sequence, atomicity requires that all changes go through in order. If one change fails, everything fails.

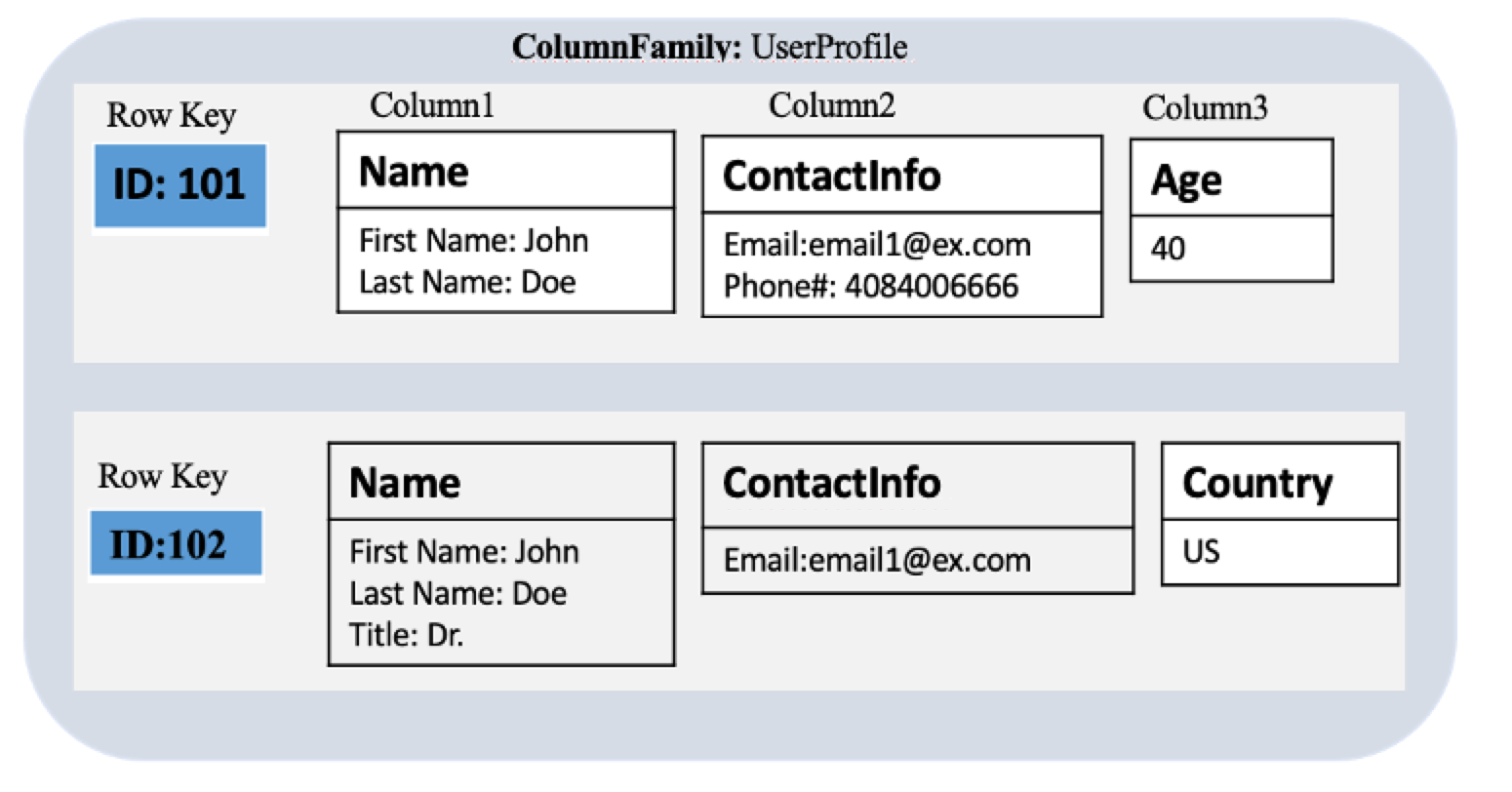
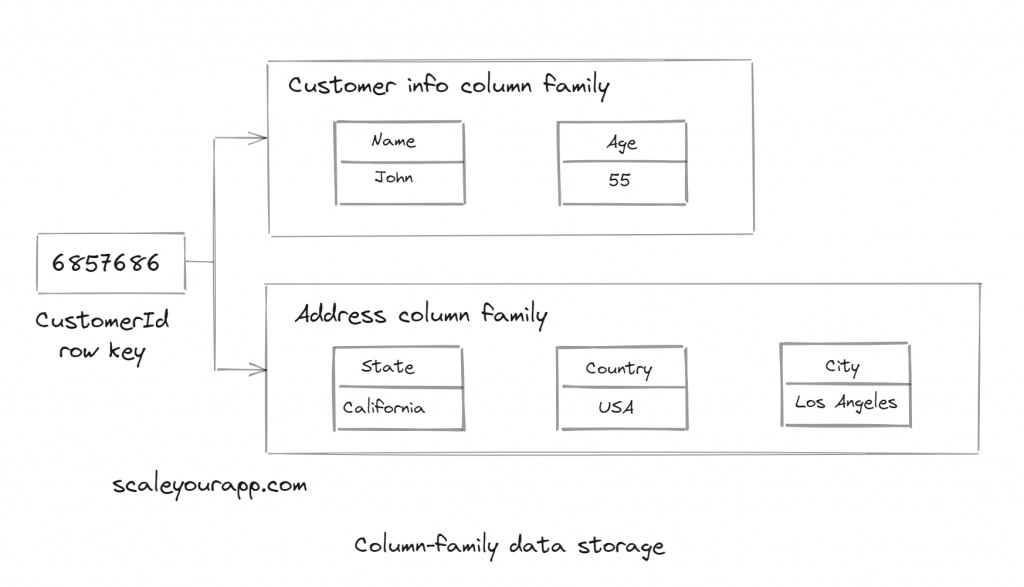
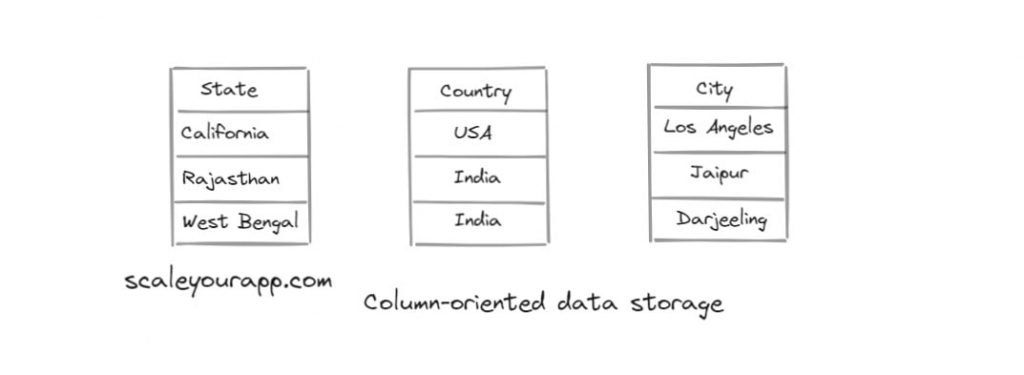
Advanced key-value databases provide native, server-side support for ACID. This simplifies the developer experience of making coordinated, all-or-nothing changes to multiple items both within and across tables. With transaction support, developers can extend the scale, performance, and enterprise benefits to a broader set of mission-critical workloads.

## 3. Wide-column Database

A wide-column database is a type of [NoSQL database](https://www.scylladb.com/resources/what-is-nosql/) in which the names and format of the columns can vary across rows, even within the same table. Wide-column databases are also known as column family databases. Because data is stored in columns, queries for a particular value in a column are very fast, as the entire column can be loaded and searched quickly

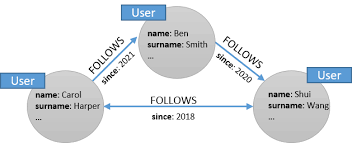






**4. GRAPH DATABASE:**

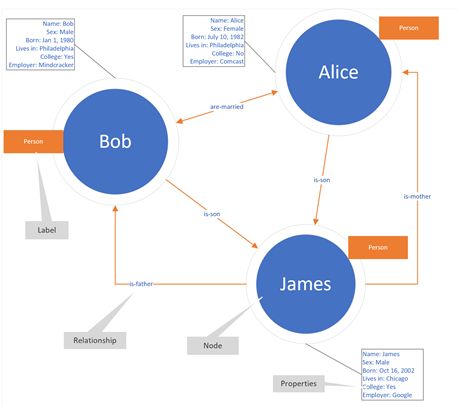
The [NoSQL database](https://aws.amazon.com/nosql/) uses mathematical graph theory to show data connections. Unlike relational databases, which store data in rigid table structures, graph databases store data as a network of entities and relationships. As a result, these databases often provide better performance and flexibility as they are more suited for modeling real-world scenarios.

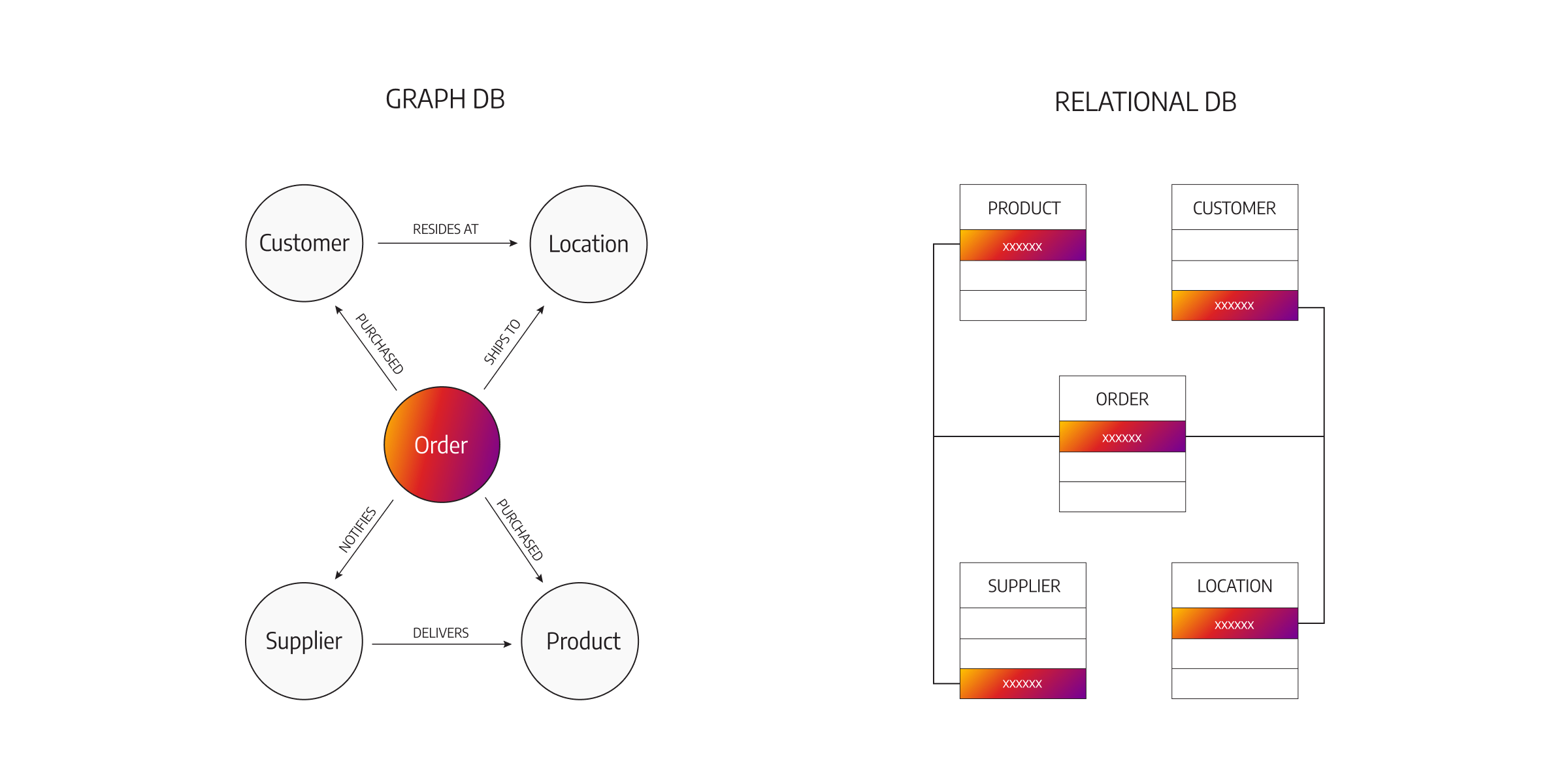


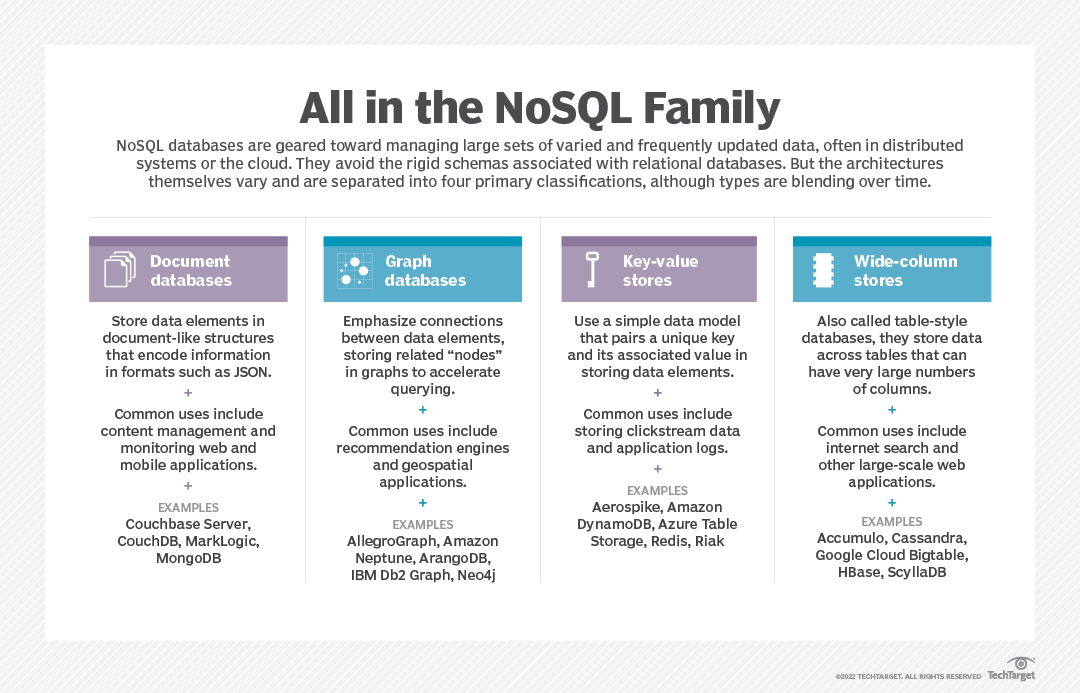
A representation of relationships in a social network graph database

**USECASES:**

1. SOCIAL NETWORKS
2. RECOMMENDATION ENGINES
3. FRAUD DETECTION





­­­­

**Query Mechanisms for NoSQL Databases:**

What is the CAP Theorem?

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 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.

**Saving the user object in MongoDB:**

To store the user object, use Save

Mongo>db.user.save({

“\_id”:1234,

“name”: { “first”:”vishnu”, “last”:”priya”},

“topics”:[“reading”,”music”]

});

**Querying the user object in MongoDB:**

Use find to filter on any attribute or Sub-attribute(s):

Mongo>db.user.find({

“\_id”: 1234

});

Mongo>db.user.find({

“name.first”:”Vishnu”

});

**Querying using $query $operators:**

Mongo>db.user.find({

“$or”:[

{ “name.first”:”ram”},

{“topics”:{ “$in”:[“skating”]

}

}

]

});

**Example:**

Query documents that belong to a specific customer.

We use find method to query documents from a mongo database. If used without any arguments or collections, find method retrieves all documents.

If we want to see the document belongs to customer Vishnu so the name field needs to be specified in the find method.

**>db.customer.find({name:”Vishnu”})**

**{“\_id”:objected(“600c1a”),”name”:”Vishnu”,”dept”:”cse”}**

We can attach Pretty method to make the document seem more appealing.

**>db.customer.find({name:”Vishnu”}).pretty()**

**{**

**“\_id”:\_\_\_**

**“name”:\_\_\_**

**}**

**Example 2:**

Query documents that belong to customers older than 40

**>db.customers.find({age:{$gt:40}}).pretty()**

