# Q.1) What is NoSQL? Explain features and types of NoSQL databases in detail.

**------->**

# NoSQL Database:

* NoSQL (Not Only SQL) databases are non-relational databases that do not use the traditional tabular relations used by SQL databases.
* Unlike SQL databases that use structured query language, NoSQL databases use different query languages and data models, such as key-value, document, column-family, and graph.
* Examples of popular NoSQL databases include MongoDB, Cassandra, Couchbase, Redis, and Amazon DynamoDB.

# Where are the NoSQL Databases Used:

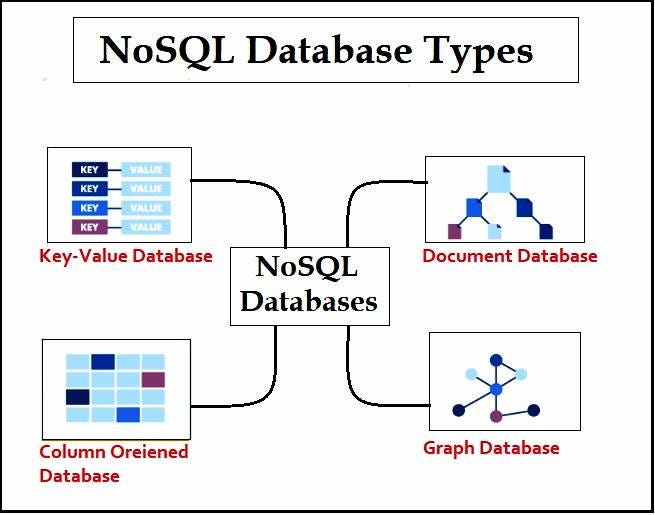
* NoSQL databases are often used for handling large volumes of unstructured or semi-structured data, such as social media data, user-generated content, and web content.
* They are also often used in distributed systems where scalability, performance, and fault-tolerance are critical.

# ****Types Of NoSQL Databases:****

There are several types of NoSQL databases, each with its own unique data model and query language.

The main types of NoSQL databases are:

1. Key-value stores
2. Document-based databases
3. Column-oriented databases
4. Graph-based databases



1. **Key-Value Stores:**
   * These databases store data as key-value pairs and are highly scalable and performant.
   * Examples: Redis, Riak, and Amazon DynamoDB.
2. **Document Databases:**
   * These databases store data in a document-oriented format, such as JSON or BSON.
   * They are flexible and can store semi-structured data.
   * Examples: MongoDB, Couchbase, and Apache CouchDB.
3. **Column-Family Stores:**
   * These databases store data in columns instead of rows and are optimized for managing large amounts of data.
   * Examples: Apache Cassandra and HBase.
4. **Graph Databases:**
   * These databases use graph structures to store and represent data, making them ideal for managing complex relationships and connections.
   * Examples: Neo4j, OrientDB, and Amazon Neptune.
5. **Object Databases:**
   * These databases store objects rather than data in tables, making them suitable for object-oriented programming.
   * Examples: db4o and Objectivity/DB.

# Features of NoSQL Databases:

1. **Flexible schemas**:

Unlike relational databases, we don’t need to specify and declare a table’s schema before inserting data.

1. **Horizontal scaling:**

NoSQL databases have a range of scaling choices available. Earlier, database scaling has been a major concern for big applications and choices have been either limited in number or costly to implement.

1. **Fast queries due to the data model**:

Data model in NoSQL database offers quick execution of queries.

1. **Ease of use for developers:**

Multiple surveys indicate that the NoSQL databases are simple and easy to use.

1. **Distribution Capabilities:**

NoSQL databases can offer data distribution at a world-wide scale. This is accomplished through several cloud regions and data centers for read-and-write operations throughout several locations.

**Q. 2) Write a note on Relational Set operators.**

**------->**

**Relational Set operators**

* + Relational set operators are used to combine or subtract the records from two tables.

* + These operators are used in the SELECT query to combine the records or remove the records.
  + In order to set operators to work in database, it should have same number of columns participating in the query and the data types of respective columns should be same. This is called Union Compatibility.
  + The resulting records will also have same number of columns and same data types for the respective column.

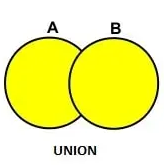
There are 4 main set operators used in the query language.

1. UNION
2. UNION ALL
3. INTERSECT
4. MINUS
5. **UNION**

- The SQL Union operation is used to combine the result of two or more SQL SELECT queries.

- In the union operation, all the number of data type and columns must be same in both the tables on which UNION operation is being applied.

- The union operation eliminates the duplicate rows from its resultset.



**Syntax**

SELECT column\_name FROM table1  UNION  SELECT column\_name FROM table2;

**Example:**

**The First table**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |

**The Second table**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

**Union SQL query will be:**

SELECT \* FROM First   UNION  SELECT \* FROM Second;

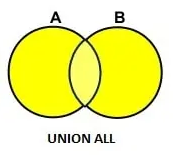
**The resultset table will look like:**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

**2. Union All**

- Union All operation is equal to the Union operation.

- It returns the set without removing duplication and sorting the data.



**Syntax:**

SELECT column\_name FROM table1  UNION ALL  SELECT column\_name FROM table2;

**Example:**

**The First table**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |

**The Second table**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

**Union All query will be like:**

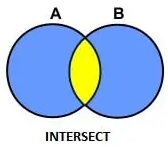
SELECT \* FROM First   UNION ALL  SELECT \* FROM Second;

**The resultset table will look like:**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

**3. Intersect**

* + It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
  + In the Intersect operation, the number of datatype and columns must be the same.
  + It has no duplicates and it arranges the data in ascending order by default.



**Syntax**

SELECT column\_name FROM table1  INTERSECT  SELECT column\_name FROM table2;

**Example:**

**The First table**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |

**The Second table**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

**Intersect query will be:**

SELECT \* FROM First   INTERSECT  SELECT \* FROM Second;

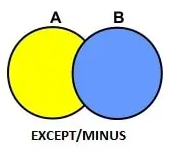
**The resultset table will look like:**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 3 | Jackson |

**4. Minus**

- It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.

- It has no duplicates and data arranged in ascending order by default.



**Syntax:**

SELECT column\_name FROM table1  MINUS  SELECT column\_name FROM table2;

**Example:**

**The First table**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |

**The Second table**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

**Minus query will be:**

SELECT \* FROM First   MINUS  SELECT \* FROM Second;

**The resultset table will look like:**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |

**Q. 3) How CRUD operation perform in Mongo DB. Explain with example.**

**------->**

**Mongodb** is a document-oriented database program widely classified as a NoSQL database program.

MongoDB allows you to manage and interact with data through four fundamental types of data operations mostly known as CRUD operations.

In MongoDB, the **CRUD** operation refers to the creating, reading, updating, and deleting documents.

* + **C**reate operations, which involve writing data to the database
  + **R**ead operations, which query a database to retrieve data from it
  + **U**pdate operations, which change data that already exists in a database
  + **D**elete operations, which permanently remove data from a database

## Create

**Create** (or insert) operations add new documents to a collection.

There are two ways to add new documents to a collection:

* db.collection.insertOne()
* db.collection.insertMany()

1. **insertOne()**

insertOne() operation allows us to create individual documents in a collection.

### Example:

Here we add a single car to the cars collection using the insertOne() operation:

db.cars.insertOne(

{

name: "Bugatti",

model: "2021"

});

1. **insertMany()**

insertmany() operation is used to create multiple documents in a single operation.

### Example:

Here  we can add info of multiple cars to the cars collection with a single operation using insertMany().

db.cars.insertMany([

{

name: "Bugatti", model: "2022"

},

{

name: "Aston", model: "2020"

},

{

name: "Ferrari", model: "2024"

}]);

## Read

**Read** operations retrieve documents from a collection.

There are two ways retrieve information from collection:

* db.collection.find()
* db.collection.find(query)
  1. **find()**

find() operation will return everything from a collection if you call it without any parameters.

### Example:

Here is an example of how we can read information about all cars from the cars collection:

db.cars.find() *// no parameters*

**Output:**

{ "\_id" : ObjectId("1"), "name" : "Bugatti", "model" : "2022" }

{ "\_id" : ObjectId("2"), "name" : "Aston", "model" : "2020" }

{ "\_id" : ObjectId("3"), "name" : "Ferrari", "2013" : "2024" }

* 1. **find(query)**

In find(qury) operation, we can specify any filter or criteria to retrieve information from a collection that will return only those records that matches with criteria specified in query.

### Example:

Here is an example of how we can read information about those cars from the cars collection whose model is 2005:

db.cars.find({"model": "2022"}); *// with one paramter*

**Output:**

{ "\_id" : ObjectId("1"), "name" : "Bugatti", "model" : "2022" }

## Update

**Update** operations modify existing documents in a collection.

There are three ways to update documents of a collection:

* **db.collection.updateOne()**
  + Updates one field in the document where the given criteria or filter meets the condition.
  + Updating a field will not remove the old field instead a new field will be added to the document.
* **db.collection.updateMany()**
  + Updates all fields in the document where the given criteria or filter meets the condition.

### Example

For example, if we have the following document:

{

"\_id" : ObjectId("1"),

"name" : "Bugatti",

"model" : 2022

}

* 1. Using updateOne({"\_id" : ObjectId("1")}, {$set: { " model" : 2023}})

will result in:

{

"\_id" : ObjectId("1"),

"name" : "Bugatti",

"model" : 2023

}

* 1. Using updateMany({"\_id" : ObjectId("1")}, {$set: { "name" : "Jaquar"}, $set: { "new\_model" : 2024}})

will result in:

{

"\_id" : ObjectId("1"),

"name" : " Jaquar ",

"model" : 2024

}

**Delete**

**Delete** operations delete documents from a collection.

There are two methods to delete documents of a collection::

* db.collection.deleteOne()
* db.collection.deleteMany()
  1. **deleteOne()**

deleteOne() method removes only the first document matched by the query filter document.

### Example

Here is an example of how we can remove only one car having the model “2020” from the cars collection:

db.cars.deleteOne({ "model": "2020"})  *//deletes one car having model "2020"*

* 1. **deleteMany()**

deleteMany() deletes multiple objects at once.

**Example**

Here is an example of how all cars, having the model “2020” - can be deleted from the cars collection:

db.cars.deleteMany({ "model": "2020"}) *//delete all cars having model "2020"*

**Q.4) Differentiate between function and stored procedure.**

**------->**

Following are the main differences between functions and procedures:

|  |  |
| --- | --- |
| **Function** | **Stored Procedure** |
| Returns a single value, either as a table or as a scalar, always. | Can return zero, a single value, or several values. |
| Run-time compilation and execution occur for functions. | The database contains stored procedures that have been parsed and compiled. |
| Only Select statements are allowed. Updating and inserting DML statements are allowed. | Capable of carrying out any action on database objects, such as DML and select statements. |
| Only input parameters are permitted. Output parameters are not supported. | Both input and output parameters are supported. |
| Does not permit the usage of Try...Catch blocks are used to handle exceptions. | Allows the use of Try...Catch blocks are used to handle exceptions. |
| Transactions are not permitted within a function. | A stored procedure can contain transactions. |
| A function cannot call a stored procedure. | A stored procedure can be called a function. |
| A Select statement can invoke functions. | Stored procedures can't be accessed by Select/Where or Having statements. To run a stored procedure, use the Execute statement. |
| In JOIN clauses, functions can be used. | JOIN clauses can't use stored procedures. |

**Q.5) What is CouchDB? Give difference between MongoDB and CouchDB.**

**------->**

## ****CouchDB****

* **Apache CouchDB** was developed bythe **Apache Software Foundation** and initially released in 2005.
* CouchDB is written in **Erlang**.
* It is an**open-source database** that uses different formats and protocols to store, transfer, and process its data.
* [Apache CouchDB](https://www.geeksforgeeks.org/introduction-to-apache-couchdb/) uses JSON to store data, [JavaScript](https://www.geeksforgeeks.org/javascript/) as its query language using MapReduce.
* Documents are the primary unit of data in CouchDB and they also include metadata.
* Document fields are uniquely named and contain values of varying types and there is no set limit to text size or element count.

## Difference Between CouchDB and MongoDB

| **Feature** | **CouchDB** | **MongoDB** |
| --- | --- | --- |
| Data Model | Document-oriented | Document-oriented |
| Query Language | MapReduce views | MongoDB Query Language |
| Replication | Multi-Master replication | Master-Slave replication |
| Consistency | Eventual consistency | Strong consistency |
| Conflict Resolution | Application-level conflict resolution | Built-in conflict resolution |
| Storage Format | Single-level storage format | Multi-level storage format |
| Scalability | Limited scalability | Highly scalable |
| Indexing | Secondary indexes | Rich indexing capabilities |
| APIs | RESTful API | Native API |
| Offline Support | Limited offline support | Limited offline support |
| Use Cases | Document synchronization, peer-to-peer | Real-time analytics, IoT, mobile apps |

**Q.6) What is Stored Procedure? How to create a Stored procedure? Illustrate with example.**

**------->**

* A stored procedure in SQL is a group of SQL queries that can be saved and reused multiple times.
* It is very useful as it reduces the need for rewriting SQL queries.
* It enhances efficiency, reusability, and security in database management.
* Users can also pass parameters to stored procedures so that the stored procedure can act on the passed parameter values.
* Stored Procedures are created to perform one or more [DML](https://www.geeksforgeeks.org/dml-full-form/)operations on the Database.
* It is nothing but a group of **SQL statements** that accepts some input in the form of parameters, performs some task, and may or may not return a value.

### Syntax to Create a Stored Procedure

CREATE PROCEDURE procedure\_name  
(parameter1 data\_type, parameter2 data\_type, …)  
AS  
BEGIN  
   — SQL statements to be executed  
END

### Syntax to Execute the Stored Procedure

*EXEC procedure\_name parameter1\_value, parameter2\_value, ..*

## ****SQL Stored Procedure Example****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **NAME** | **AGE** | **ADDRESS** | **SALARY** |
| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
| 2 | Khilan | 25 | Delhi | 1500.00 |
| 3 | Kaushik | 23 | Kota | 2000.00 |
| 4 | Chaitali | 25 | Mumbai | 6500.00 |
| 5 | Hardik | 27 | Bhopal | 8500.00 |
| 6 | Komal | 22 | Hyderabad | 4500.00 |
| 7 | Muffy | 24 | Indore | 10000.00 |

-- **Create a stored procedure named "GetCustomersByCountry"**

CREATE PROCEDURE GetCustomersByCountry  
 @Country VARCHAR(50)  
AS  
BEGIN  
 SELECT CustomerName, ContactName  
 FROM Customers  
 WHERE Country = @Country;  
END;

**-- Execute the stored procedure with parameter "Sri lanka"**

EXEC GetCustomersByCountry @Country = 'Sri lanka';