Many new developers wonder what the difference is between SQL and NoSQL. The subjects come up often in theoretical discussions and systems design interview preparation.

In this tutorial, we will be looking at the difference between SQL and NoSQL based on certain parameters. Before we begin, let’s analyze each individually to develop our understanding of these contrasting concepts.

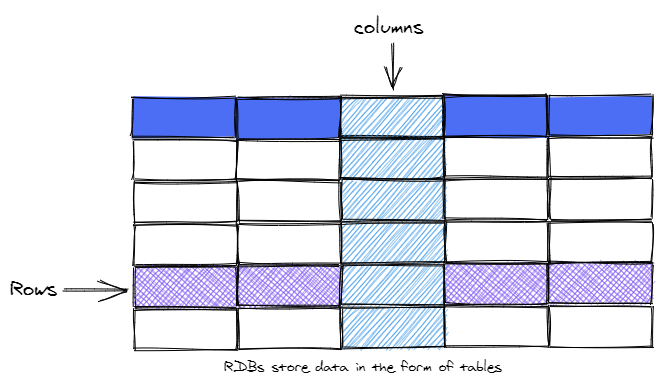
What is SQL?

Some definitions to start:

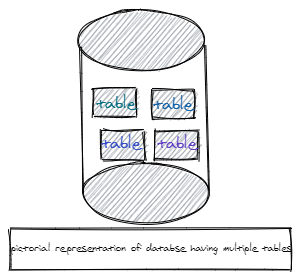
SQL stands for **Structured Query Language**. As evident from the name, it is a querying language that is used to perform various data operations. It is supported on almost all **Relational Databases.**

*And what is a Relational Database?* A relational database, often synonymous as a SQL Database, is a collection of data records that may have predefined associations or relationships with each other.

By extension, a **Relational Database Management System (RDMS)** is an interface (application or UI) to manage the records, usually by way of SQL. It's used to store, edit, read, and write such data in the form of tables. It's also often visually shown as a spreadsheet-like shape.



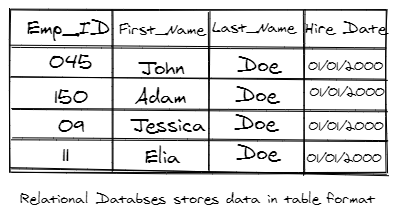
SQL Databases



SQL databases, or Relational Databases, employ the traditional way of storing data (where each database may contain several tables). However, it is worth mentioning that in RDBs, each table contains rows that have the same attributes or "shape". As an example, in a "restaurant" database, you may have one table for customers, another for orders, and yet another for restaurant locations.

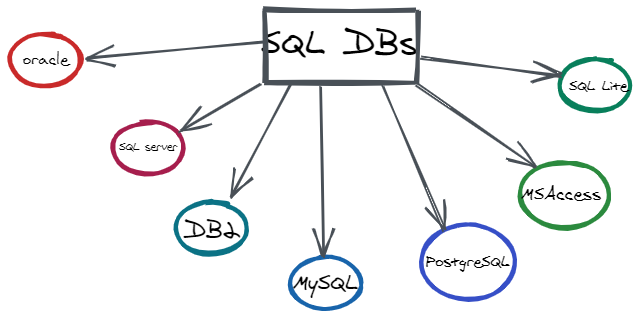
Now here comes the **importance of columns**. Each column describes one specific piece of information about the record. The type of data that each column will contain is also predefined. For example, in a customer table, one column may contain the customer id (an integer type column). Another may show the customer order id (another integer), and a third may contain information about customer address (varchar).

Properly defining tables and columns is upfront work to be done to gainfully use a database. Then, the day to day task becomes adding new rows. This is where the SQL language comes in handy. It offers a convenient way to create tables, define columns, add records, delete them, etc. It also allows us to "connect" or associate (create relationships for) multiple tables by using the concept of primary and foreign keys.

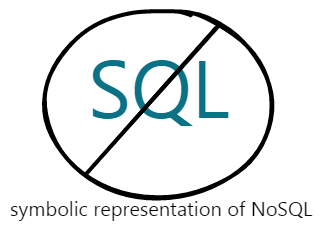


Types of SQL DBs

Here's a selection of the various SQL Databases.



What is NoSQL?



Let's move on to the definition of NoSQL. Several NoSQL database systems have grown rapidly in popularity over the last few years. They are non-relational DMSs that support structured data, unstructured, semi-structured, *and* polymorphic data.

At a high level, NoSQL just means **not only SQL**. Thus, it doesn't speak to a single database offering-- rather, it's a collection of diverse technologies. Don't assume that the databases are similar in nature either.

As stated, there are no official rules as to what makes a database NoSQL. The term describes more **about what they aren’t**.

NoSQL Databases:

* typically don't use SQL
* usually don't store data in tables
* usually don't care about relationships
* usually don't provide ACID transaction
* usually don't require formal data schemas (making them more flexible)

Types of NoSQL Technologies

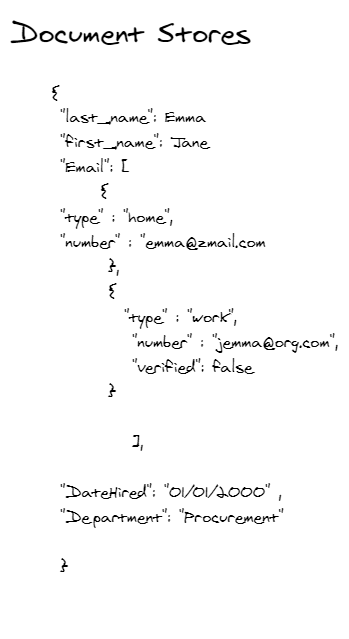
Let's run through a list of data store offerings within the NoSQL flavor.

1. **Document DB/Document Stores:** These are organized around the idea that the fundamental thing to store is a self-contained piece of data (called a document). Such a document describes its own schema-- this is in opposition to individual rows of data constrained in well-defined columns. With documents, there generally are no restrictions in shape or structure. It may be in XML format, but often it's JSON-- a loose structure based on plain old Javascript objects.

**Example:** store a simple piece of data consisting of two pieces of information: title and rating. The title is a string and rating is an integer.



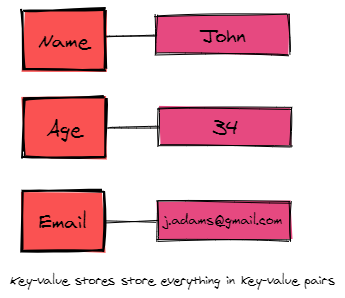
In the next piece of data, we have different descriptions and utilize nesting of information.



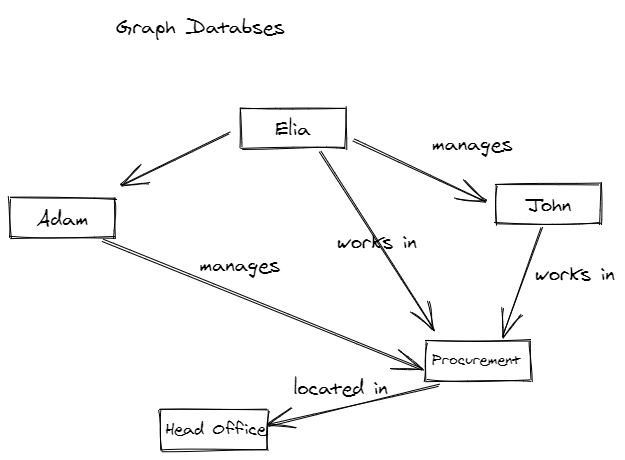
When we are schema-less, we can simply add new documents in this data store freely. The only constraint is that each document will be given a unique id. But beyond that, you often have total flexibility. There's no need to provide a formal schema and no need to define relationships. One downfall of this is that the database needs to provide a more flexible way of querying the data-- thus arrived solutions like CouchDB and MongoDB.

1. **Key-value databases** are another category. Its main emphasis is on having **no predefined schema** for your data. All it does is store and retrieve everything based on key-value pairs. In some ways, it's like a two-column table (if you had a key column and value column), and nothing more is enforced. It could be any data type. You could fit in bits of XML or JSON.

Furthermore, It is worth mentioning that many products in this category are specifically designed with fault-tolerant distributed architecture. Simplified, this means you can easily install them across multiple machines. As such, no one machine is a point of failure-- the database can survive machine failures and continue functioning.

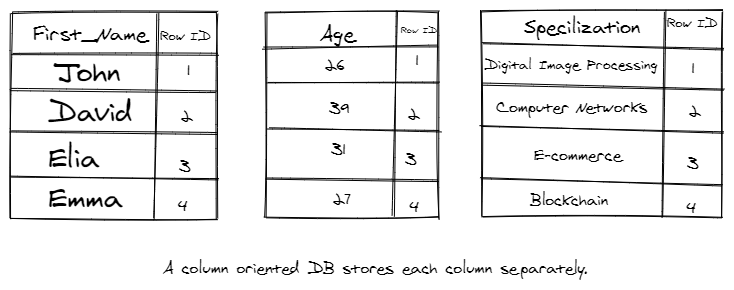


1. **Graph:** Yet another category of NoSQL tech. It is a data store in which everything is kept in the form of connecting nodes, in a graph structure. There is no one "master" point. Although many other NoSQL DBMS tend to de-emphasize relationships, graph databases are all about having nodes of data connected to each other, describing relationships among them.



1. **Column-oriented database:** In a traditional row-oriented database, when we try to retrieve a specific record, every single row is scanned. Indexing certain columns may improve the lookup speed-- however, indexing every column slows down the updating (write) speed. Sometimes it may require you to lock the tables-- if not completely, then partially-- by locking certain parts of your table.

This is where column-oriented databases come in handy. They store individual columns separately, allowing an efficient scan when we have a limited number of columns (because there is only one data type in each table). Therefore, it is very efficient to add new columns. However, adding an entire record becomes more difficult. Although they may look similar to traditional databases, the method of storing and retrieving data is where the actual difference lies. They are best for analytics. Examples of the column-oriented database include but not limited to Cassandra (released by Facebook as an open-source project), Hypertable, Google BigTable, and Apache HBase.



Comparison of the Two

Now that we know the fundamentals of both SQL and NoSQL, we can dive into the technical details and compare them across certain factors. Analyzing across these parameters will help us understand their differences.

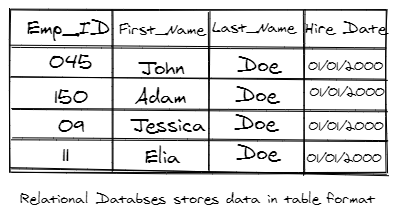
Theoretical difference

In short, SQL databases are primarily Relational Database Management Systems, while NoSQL databases are non-relational distributed databases and datastores.

Schema:

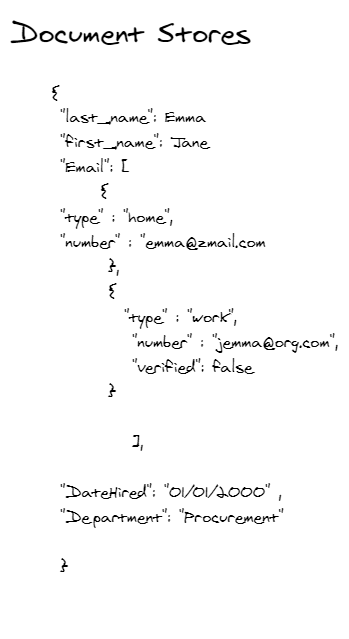
A schema refers to the *organization and shape of records* within a database. Both of these database types have a very contrasting data storage models.

**SQL databases** usually deal with *structured data* that is organized in **the form of tables**. On the other hand, **NoSQL databases**, along with support for structured data, offer the convenience of storing unstructured, semi-structured, and polymorphic data as well.



NoSQL databases can store information in the form of document stores, key-value pairs, graphs databases, and column stores. These DBs do not have predefined schemas that they need to adhere to.

This distinction makes them useful for specific scenarios. For example, RDBs are intuitively useful for accounting systems because they model the accounting ledger well. NoSQL is preferred in "Big Data" scenarios where the flexibility of data types are essential.



Scalability

Scalability refers to how a database technology adapts to an ever-increasing amount of data without sacrificing performance.

In this regard, singular SQL databases tend to be **vertically scalable**, in the sense that additional load can be handled by using more efficient and newer hardware (CPUs, RAM and SSD). On the flip side, NoSQL databases tend to be more **horizontally scalable** (they can automatically handle more traffic by distributing it among more servers in the database cluster).

Note that relational databases can also be scaled via more hardware-- however, there is some additional work to be done to unify the various database instances. This is why NoSQL databases are preferred in the case when our data is increasing at a very high rate.

Support

Historically, SQL database were in the market long before the NoSQL ones arrived. Therefore, it shouldn’t come as a surprise that almost all the SQL database vendors provide substantial support to their users. Moreover, a huge community of independent consultants is there to help with the large deployment of SQL databases.

Of course, the same is starting to also be true for NoSQL databases. However, many are still in their embryonic stage, and thus depend more (or often solely) on the open source community for support. Furthermore, independent consultants are also more scarce when it comes to the setup and deployment of large-scale NoSQL databases.

Languages Used

**SQL databases** support Structured Query Language (SQL) which allows us to perform various operations on the database. It accomplishes this by providing different types of commands. These commands can be categorized based on their functionality:

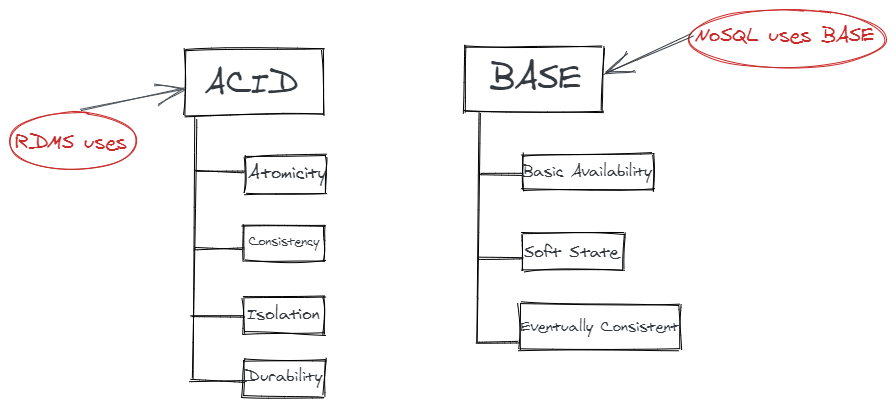
1. **Data Definition Language** (DDL)
2. **Data Manipulation Language** (DML)
3. **Data Control Language** (DCL)
4. **Transaction Control Language** (TCL)
5. **Data Query Language** (DQL)

Though SQL is the most stable and widely used option for performing database operations, it may be restricting in the sense that it requires users to predefine the structure of tables and records. There's also the constraint that all data it stores must be consistent in observing that structure.

On the other hand, **NoSQL databases** are *dynamic and flexible*. They allow storing unstructured data in multiple ways. Depending on the nature of data, NoSQL options include document stores, key-value pairs, graph databases, and column-oriented stores. As mentioned, you are not required to come up with the structure of data before-hand. Each document may have its own unique structure. This is referred to as Unstructured Query Language (UnQL) and syntax will vary between products.

ACID vs BASE Model:

SQL databases never compromise on ACID (Atomicity, Consistency, Isolation, Durability) properties. On the flip side, most of the NoSQL DBs observe the Brewers CAP theorem (Consistency, Availability and Partition tolerance) and the BASE Model (Basic Availability, Soft-state, Eventual Consistency).



Examples

Finally, let's provide some examples. Although there are many RDBs and NoSQL, we will restrict this list to the most popular ones:

**SQL DBs include** MS-SQL, Oracle, SQLite, MySQL, and Postgres.

**NoSQL DBs include** CouchDB, MongoDB, BigTable, Cassandra, Redis, RavenDB, HBase, and Neo4j.