**SQL / RDBMS / Relational databases**

RDBMS are more widely known and understood than their NoSQL cousins. Relational databases emerged in the 70’s to store data according to a schema that allows data to be displayed as tables with rows and columns. Think of a relational database as a collection of tables, each with a *schema* that represents the fixed attributes and data types that the items in the table will have. RDBMS all provide functionality for reading, creating, updating, and deleting data, typically by means of Structured Query Language (SQL) statements.

The tables in a relational database have *keys* associated with them, which are used to identify specific columns or rows of a table and facilitate faster access to a particular table, row, or column of interest.

Data integrity is of particular concern in relational databases, and RDBMS use a number of constraints to ensure that the data contained in your tables is reliable and accurate.

While there are many relational databases, over time these have become the most popular:

* [Oracle](https://www.oracle.com/database/index.html) - Oracle Database (commonly referred to as Oracle RDBMS or simply as Oracle) is a multi-model database management system produced and marketed by Oracle Corporation.
* [MySQL](https://www.mysql.com/) - MySQL is an open source RDBMS based on Structured Query Language (SQL). MySQL runs on virtually all platforms, including Linux, UNIX, and Windows.
* [Microsoft SQL Server](https://www.microsoft.com/en-us/sql-server/sql-server-2017) - Microsoft SQL Server is an RDBMS, that supports a wide variety of transaction processing, business intelligence and analytics applications in corporate IT environments.
* [PostgreSQL](https://www.postgresql.org/) - PostgreSQL, often simply Postgres, is an object-relational database management system (ORDBMS) with an emphasis on extensibility and standards compliance.
* [DB2](https://www.ibm.com/analytics/us/en/db2/) - DB2 is an RDBMS designed to store, analyze and retrieve data efficiently.

**Advantages**

* Relational databases are well-documented and mature technologies, and RDBMS are sold and maintained by a number of established corporations.
* SQL standards are well-defined and commonly accepted.
* A large pool of qualified developers have experience with SQL and RDBMS.
* All RDBMS are ACID-compliant, meaning they satisfy the requirements of Atomicity, Consistency, Isolation, and Durability.

**Disadvantages**

* RDBMSes don’t work well — or at all — with unstructured or semi-structured data, due to schema and type constraints. This makes them ill-suited for large analytics or IoT event loads.
* The tables in your relational database will not necessarily map one-to-one with an object or class representing the same data.
* When migrating one RDBMS to another, schemas and types must generally be identical between source and destination tables for migration to work (schema constraint). For many of the same reasons, extremely complex datasets or those containing variable-length records are generally difficult to handle with an RDBMS schema.

**NoSQL / Non-relational databases**

NoSQL databases emerged as a popular alternative to relational databases as web applications became increasingly complex. NoSQL/Non-relational databases can take a variety of forms. However, the critical difference between NoSQL and relational databases is that RDBMS schemas rigidly define how all data inserted into the database must be typed and composed, whereas NoSQL databases can be schema agnostic, allowing unstructured and semi-structured data to be stored and manipulated.

**Types**

Note that some products may fall into more than one category. For example, Couchbase is both a document database and a key-value store.

**Key-Value Stores**, such as [Redis](https://redis.io/) and [Amazon DynamoDB](https://aws.amazon.com/dynamodb/), are extremely simple database management systems that store only key-value pairs and provide basic functionality for retrieving the value associated with a known key.

The simplicity of key-value stores makes these database management systems particularly well-suited to embedded databases, where the stored data is not particularly complex and speed is of paramount importance.

**Wide Column Stores**, such as [Cassandra](http://cassandra.apache.org/), [Scylla](http://www.scylladb.com/), and [HBase](https://hbase.apache.org/), are schema-agnostic systems that enable users to store data in *column families or tables, a single row of which can be thought of as a record* — a multi-dimensional key-value store.

These solutions are designed with the goal of scaling well enough to manage petabytes of data across as many as thousands of commodity servers in a massive, distributed system.

Although technically schema-free, wide column stores like Scylla and Cassandra use an SQL variant called [CQL](https://docs.datastax.com/en/cql/3.1/cql/cql_intro_c.html) for data definition and manipulation, making them straightforward to those already familiar with RDBMS.

**Document Stores**, including [MongoDB](https://www.mongodb.com/) and [Couchbase](https://www.couchbase.com/), are schema-free systems that store data in the form of JSON documents. Document stores are similar to key-value or wide column stores, but the *document name* is the key and the contents of the document, whatever they are, are the value.

In a document store, individual records do not require a uniform structure, can contain many different value types, and can be nested. This flexibility makes them particularly well-suited to manage semi-structured data across distributed systems.

**Graph Databases**, such as [Neo4J](https://neo4j.com/) and[Datastax Enterprise Graph](https://www.datastax.com/products/datastax-enterprise-graph), represent data as a network of related nodes or objects in order to facilitate data visualizations and graph analytics.

A node or object in a graph database contains free-form data that is connected by relationships and grouped according to labels. Graph-Oriented Database Management Systems (DBMS) software is designed with an emphasis on illustrating *connections* between data points.

As a result, graph databases are typically used when analysis of the relationships between heterogeneous data points is the end goal of the system, such as in fraud prevention, advanced enterprise operations, or Facebook’s original friends graph.

**Search Engines**, such as [Elasticsearch](https://www.elastic.co/), [Splunk](https://www.splunk.com/), and [Solr](http://lucene.apache.org/solr/), store data using schema-free JSON documents. They are similar to document stores, but with a greater emphasis on making your unstructured or semi-structured data easily accessible via text-based searches with strings of varying complexity.

**Advantages**

Since there are so many types and varied applications of NoSQL databases, it’s hard to nail these down, but generally:

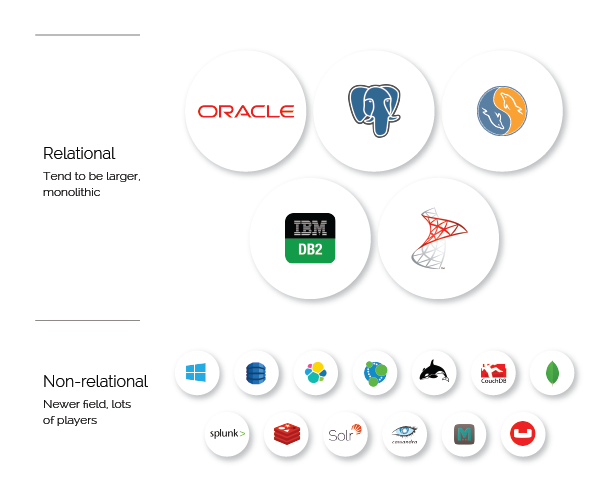
* Schema-free data models are more flexible and easier to administer.
* NoSQL databases are generally more horizontally scalable and fault-tolerant.
* Data can easily be distributed across different nodes. To improve availability and/or partition tolerance, you can choose that data on some nodes be "[eventually consistent](https://en.wikipedia.org/wiki/CAP_theorem)".

**Disadvantages**

These are also dependent on the database type. Principally:

* NoSQL databases are generally less widely adopted and mature than RDBMS solutions, so specific expertise is often required.
* There are a range of formats and constraints specific to each database type.

**Popular Relational and Non-relational databases**



**Which database is right for you?**

This post looks only at the most popular and best-known examples of these types of databases. For a more comprehensive list, including descriptions, look [here](https://db-engines.com/en/ranking).

* If [ACID](https://en.wikipedia.org/wiki/ACID) (Atomicity, Durability, Consistency, and Durability) compliance is your first priority, consider using RDBMS.
* If you have a massively distributed system and can settle for eventual consistency on some nodes/partitions, you might consider a wide column store such as Cassandra or Scylla.
* If your input data is particularly heterogeneous and difficult to encapsulate according to a normalization schema, consider using a NoSQL DBMS.
* If your goal is to scale vertically, consider an RDBMS; conversely, if you want to scale horizontally, a NoSQL DBMS may be preferable.