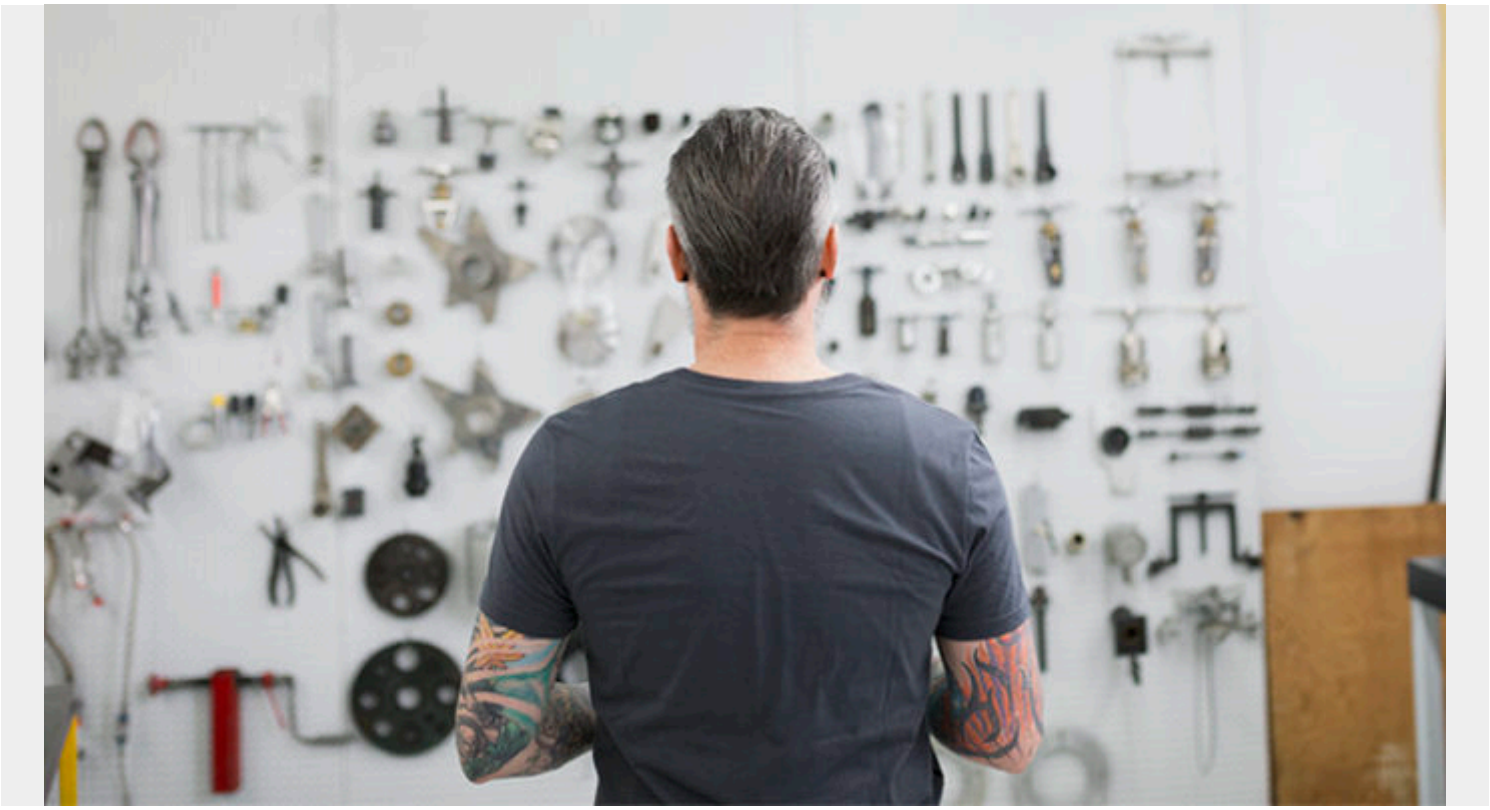
SQL VS NOSQL DATABASES: WHAT'S THE DIFFERENCE?



Larry Ellison is one of the world's richest men, and has been for decades. He founded Oracle in the early 1970s, taking an idea from IBM's Edgar F. Cobb for a SQL relational database to create the Oracle Database RDBMS (relational database management system).

Oracle obtained a monopoly position in this market, since in those days there was virtually no free software and practically all applications ran on IBM mainframes. Even IBM's database, DB2, could not unseat Oracle as the market leader, as that database only ran on mainframes. It was not until racks of PCs began to be used as servers in the 1990s (as they are today) that people used anything but mainframes.

Oracle is still a monopoly for most transactional business applications among the Fortune 500. Oracle has purchased the most common opensource alternative, MySQL, and has even purchased opensource Java itself. But those remain free.

When it comes to choosing a database, one of the biggest decisions an organization may have to make is whether to pick a relational (SQL) or non-relational (NoSQL) data structure. While both of these are good choices, each have clear advantages and disadvantages which must be kept in mind.

In this article, we've broken down the most important differences between SQL and NoSQL, highlighting the benefits of each.

What is SQL?

SQL, short for Structured Query Language, is a programming language that is used to manage data in relational databases. Relational databases use relations (typically called tables) to store data and then match that data by using common characteristics within the dataset.

Some common relational database management systems that use SQL:

- Oracle
- Sybase
- Microsoft SQL Server
- Microsoft Access
- Ingres.

Cobb's breakthrough paper describes a database where objects could be constructed and queried using something he called SQL, structured query language. He used SQL to create both data (in objects called tables) and the schema for that data, which describes fields in columns. A single record in a SQL database is called a row.

What is NoSQL?

A NoSQL database, on the other hand, is self-describing, so does not require a schema. Nor does it enforce relations between tables in all cases. All its documents are JSON documents, which are complete entities that one can readily read and understand.

NoSQL refers to high-performance, non-relational databases that utilize a wide variety of data models. These databases are highly recognized for their:

- Ease-of-use
- Scalable performance
- Strong resilience
- Wide availability

NoSQL database examples include:

- MongoDB
- MarkLogic
- Couchbase
- CloudDB
- Amazon Dynamo DB

SQL vs NoSQL: Major differences

There are many differences between SQL and NoSQL, all of which are important to understand when making a decision about what might be the best data management system for your organization. These include differences in:

- Language
- Scalability
- Community

- Structure

We'll look at each difference in detail.

Language

One of the major differences between SQL relational and NoSQL non-relational databases is the language.

SQL databases use Structured Query Language for defining and manipulating data. This allows SQL to be extremely versatile and widely-used—it also makes it more restrictive. SQL requires that:

- You use predefined schemas to determine the structure of your data *before* you begin to work with it.
- Your data also follow the same structure, which can entail significant upfront preparation along with careful execution.

A NoSQL database features a dynamic schema for unstructured data and the data can be stored in many different ways, whether it be graph-based, document-oriented, column-oriented, or organized as a KeyValue store. This extreme flexibility allows you to:

- Create documents without first having to carefully plan and define their structure
- Add fields as you go
- Vary the syntax from database to database
- Give each document its own unique structure, providing you with more freedom overall

Scalability

Another big difference between SQL and NoSQL is their scalability. Most SQL databases are vertically scalable, which means that you can increase the load on a single server by increasing components like RAM, SSD, or CPU.

In contrast, NoSQL databases are horizontally scalable, which means that they can handle increased traffic simply by adding more servers to the database. NoSQL databases have the ability to become larger and much more powerful, making them the preferred choice for large or constantly evolving data sets.

The community

Due to SQL's maturity, it has a much stronger and more developed community compared to NoSQL. There are thousands of chats and forums available where experts can share knowledge and discuss SQL best practices, continuously enhancing skills.

Although NoSQL is growing rapidly, its community is not as well defined as SQL due to the fact that it is still relatively new.

Structure

Finally, a last thing to consider when debating SQL versus NoSQL is their structures. SQL databases are table-based which makes them a better option for applications that require multi-row transactions. Examples may be accounting systems or legacy systems that were originally built for a

relational structure.

NoSQL databases can be:

- Key-value pairs
- Wide-column stores
- Graph databases
- Document-based

So, which database is right for your business?

The best way to determine which database is right for your business is to analyze what you need its functions to be.

SQL is a good choice for:

- Any organization that will benefit from a predefined structure and set schemas, particularly if they require multi-row transactions.
- Situations when all data must be consistent without leaving room for error, such as with accounting systems.

NoSQL is a good choice for those companies experiencing rapid growth with no clear schema definitions. NoSQL offers much more flexibility than a relational database and is a solid option for companies who must analyze large quantities of data or whose data structures they manage are variable.

Now, let's explore some examples to illustrate the differences in these concepts.

SQL vs NoSQL examples

Below you can clearly see that the first field is student and the second field is class.

```
{ student:  "Walker Rowe",
  class:  "biology"
}
```

In terms of SQL, the user would first create this schema before they could add data to the database:

```
CREATE TABLE studentClasses (
    student varchar,
    class varchar
);
```

Where **varchar** is variable character length.

To add data to that table, one would:

```
INSERT INTO studentClasses (student, class)
VALUES ("Walker Rowe", "biology:);
```

With a NoSQL database, in this example MongoDB, you would use the database API to insert data like this:

```
db.studentClasses.insert( { name: "Walker Rowe", class: "biology" } )
```

And then you can use SQL to create the:

- **Union**, which is all elements from two or more sets
- **Intersection**, which are common elements of two or more sets

The big breakthrough here was to let programmers do all this using easy-to-understand SQL syntax. Then Oracle made further technological advances to ensure database referential integrity and improve performance by indexing fields and caching records.

(Database referential integrity means the completeness of transactions so that there are no orphaned records. For example, a sales record with no corresponding product item. This is what is I meant by saying Oracle can enforce the relationship between tables.)

In the MongoDB example we have described above, Oracle programmers would say that the table studentClasses is an intersection. Because you can determine from it both what classes a student has and which students are in which class. In this case you would also have both student and class records contain things like the class room number and the student phone number.

The Oracle database is called a row-oriented database. Data is grouped into rows and columns. We don't need to mention column-oriented databases here, like Cassandra, as they are different in architecture and not conception to such a large degree. So they are not so fundamentally different as SQL versus NoSQL.

In particular, the Cassandra NoSQL database is used to group similar columns of data near each other so they can be retrieved at the highest possible speed. Also, Cassandra and NoSQL database get rid of the concept of database normalization, which is key to Oracle, as we explain below. And they do not store empty column values, so the row lengths can differ.

Efficiency and Normalization

One thing that Oracle stressed was the relationship between objects. They said that all data should be normalized. This means no data should be stored twice. So instead of putting, for example, the school address in every student record, it would be better to maintain a school table and store the address there.

NoSQL databases have gotten rid of this constraint, to a certain degree.

Disk space was expensive in the 1970s and so was memory, so normalization made sense. But it can take some time to do a joint operation to bring together a record that is stored in different tables into one logical unit. It also requires the overhead of maintaining index files and writing to those as data is added or deleted

NoSQL databases say all that does not matter as disk space and memory are cheap. Proponents of that say it is okay to, regarding the aforementioned case, put the school address in with the student. This speeds data retrieval time and makes coding easier.

NoSQL vs SQL

Oracle's largest competitor in the business market is SAP. They have their own database, Hana. But the only difference between them and Oracle is Hana stores all its records in memory (flushing them

to disk as needed). It does this for speed. Regardless, it is still a RDBMS.

It is difficult to make the case to switch to NoSQL databases in business applications that have been running for decades or to propose those for new applications when companies already have knowledge of RDBMS. There are management issues that Oracle has solved, such as data replication, that could leave someone using, for example, ElasticSearch, without support and with a downed system.

To fill that gap, some companies have taken over the support of—and sometimes most of the programming for—so-called opensource databases, like ElasticSearch. (If you want support for that, you can buy support and a supported version from Elastic.)

The other is the paradigm switch for transactional systems. It is easy to conceive of adding a sale to a sales database. Oracle then would automatically calculate on-hand inventory using a saved SQL operation called a view. For MongoDB, a program would have to sort through the inventory items and subtract the sales to determine the new on-hand inventory.

Common NoSQL databases

If you read the use cases for NoSQL databases, you will find that those tend to be adopted as niche and not enterprise systems.

Apache Cassandra

For example, Uber uses Cassandra to keep track of drivers. But its needs are unique, including the need to write millions of records per second across multiple data centers. They even wrote their own implementation of Cassandra so that it could run on Mesos, an orchestration system similar to containers.

DynamoDB

Amazon markets its DynamoDB database as having "millisecond latency." They also drop the term NoSQL and simply call it a non-relational database.

DynamoDB, like MongoDB, has a JavaScript interface, so you can work with it using that relatively simple programming language as well. For example, to add a record, you first instantiate an instance of the database, then add the JSON item like this:

```
var docClient = AWS.DynamoDB.DocumentClient()
docClient.put("{JSON … }"}
```

One implementation detail is that you can run these operations in MongoDB and DynamoDB using Node.js. That is JavaScript running in the middle tier, so you do not need to create JAR files or middleware servers like Oracle Weblogic.

So, which should you be using for your new project? Your accounting system could very well continue to run on an RDBMS system. But there are alternatives to paying Oracle for licensing fees, like using MySQL.

But will it use MongoDB? That is not very likely for the short term, as there are millions of programmers around the world using Java and Oracle and project managers and users who

understand that.

Use ElasticSearch for logs and Spark for analytics. As for the others, study those on a case by case basis to see which works best given your:

- Resources
- Skill
- Ability to suffer lost transactions
- Other factors

Making the decision

No matter what field you are in, choosing the correct database for your organization is an important decision.

NoSQL databases are quickly becoming a major part of the database landscape today, and they are proving to be a real game-changer in the IT arena. Companies integrating Big Data often favor the many NoSQL benefits, including:

- Lower cost
- Open-source availability
- Easier scalability

They drawbacks of NoSQL is that it's a young technology, making them slightly more volatile.

On the other hand, SQL databases have proven themselves for over 40 years and use long-established standards that are well defined. They have a huge community of experts behind them, and the opportunity for collaboration is limitless.

Overall, the decision of using SQL versus NoSQL for business is not entirely black and white; it requires some comparing and contrasting to determine which database best fits your specific needs. With the proper amount of research and preparation, however, you will ensure that the database you choose provides an efficient and streamlined management system for your organization.

Running SQL on Db2?

Learn about BMC AMI SQL Performance for Db2®. Maximize visibility so you can eliminate the wasteful SQL statements that are slowing you down.

Designed to manage SQL performance throughout the application lifecycle, the tools in this solution will help you:

- Diagnose performance problems and track them to their source, so you can effectively tune your SQL
- Anticipate SQL-related slowdowns so you can resolve them before they impact service levels
- Avoid cumbersome reorganizations and costly CPU upgrades by making the most of your resources
- Ensure your SQL is running efficiently and cost effectively at all times

Related reading

- BMC Machine Learning & Big Data Blog
- BMC Guides, which offer series of tutorials on many of the database options included in this article
- Data Storage Explained: Data Lake vs Warehouse vs Database
- CAP Theorem for Databases: Consistency, Availability & Partition Tolerance
- What Is DBaaS? Database-as-a-Service Explained
- Data Ethics in Companies

Sql vs NoSQL from **RTigger**