Choosing a DB engine

Tags: ACG

From time to time we receive a question about which DB engine is better and when to choose SQL over NoSQL. This has the potential to be a large and complex discussion or a full course, but we will keep this help article high level.

What is the difference and how do I choose?

As this article is high level, please be sure to follow the discussions in the Associate level course.

The two primary decisions to consider are how you plan to use the data and how you want to manage it.

The choice between a NoSQL database vs. a SQL database

This is really a choice that should be driven by how you are planning to use the data.

If it is very structured data where you plan to use table joins and complex data relationships then you should probably use a SQL (Relational) DB. Table joins and efficient complex queries are strengths of DBs with well structured data.

If the data needs to have flexible structures but fairly clear use patterns, then a NoSQL may be better. NoSQL DBs are often the choices for web applications where the ability to accommodate various data structures is a design advantage, and complexity of analysis is not an immediate concern.

There are many types of DB engine. Common SQL (Relational) engines include; Aurora, MySQL, MariaDB, Oracle, SQL Server (MS), and PostgreSQL. There are also literally hundreds of NoSQL engines of different types (external site, opens in new tab). AWS promotes their own NoSQL-as-a-service platforms (external site, opens in new tab) (DynamoDB, Neptune, and ElastiCache) however you have a wide choice beyond that. All have their relative pros and cons, and as a designer you should choose the DB based on the features that you need vs. the cost.

Traditionally, a business would only have one or two different DB engines due to the cost of having a team of DBAs to administer each engine. With DB-as-a-service there is no reason to not mix and match and use the best tool in the right place.

How to manage the DB

In the context of AWS, the first choice you need to make is about how you want to manage the Database engine. Do you want or need to do it yourself, or would you prefer that AWS managed the hardware, OS, database engine and fault tolerance so that you can just focus on the data? The latter is DB-as-a-service—RDS, DynamoDB, Elasticache, Redshift, Neptune (external site, opens in new tab). You pay a predictable fee based on the size or the

number of transaction, and AWS takes responsibility for all the hardware and patching and fault tolerance. The advantages are predictable cost, and a reduced level of infrastructure skill to achieve the same outcome.

The primary disadvantages of this is that:

- You must operate within the offered versions and state of the server
- You cannot make use of admin controlled functions such as Log Shipping or customized disk configurations
- You must trust the service that AWS offers.
- You must trust the compliance certification AWS maintain.

If you have reasons why this is not acceptable, then DB-as-a-service is not for you. In which case you can choose to build a Database engine on an EC2 instance and have full control of the OS and engine, and if it is justified the choice of dedicated hardware including the recently announced Bare Steel offerings (external site, opens in new tab).

There is a lot more to Databases and the process of choosing the right one for your implementation. If you have an interested in Database solutions then I would encourage you to read read some of the very good industry discussions and then start experimenting with the various types to gain a more intimate understanding of the strengths ad weaknesses or each.
