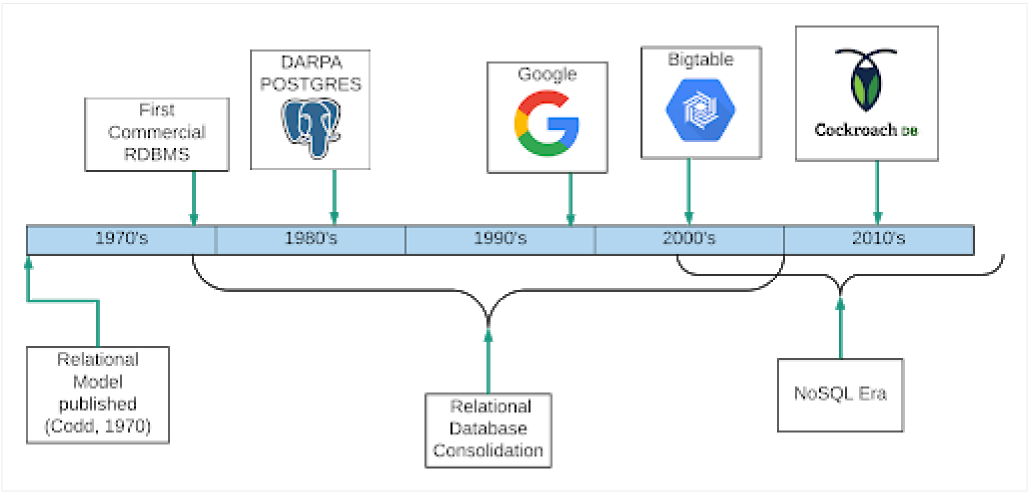
The birth of the relational database

In June of 1970, a computer scientist from IBM named Edgar F. Codd published an academic paper titled, A Relational Model of Data for Large Shared Banks. That paper introduced a new way to model data. It elaborated a way of building a bunch of cross-linked tables that would allow you to store any piece of data just once. A database with this structure could answer any question, so long as the answer was stored somewhere in it. Disk space would be used efficiently, at a time when storage was expensive. This paper launched databases into the future.

Oracle brought the first commercial relational database to market in 1979 followed by DB2, SAP Sysbase ASE, and Informix.

In the 1980s and '90s, relational databases grew increasingly dominant, delivering rich indexes to make any query efficient. Table joins, the term for read operations that pull together separate records into one, and Transactions, which means a combination of reads and especially writes spread across the database, were essential. SQL, the Structured Query Language, became the language of data, and software developers learned to use it to ask for what they wanted, and let the database decide how to deliver it. Strict guarantees were engineered into the database to prevent surprises.



The arrival of the NoSQL database

Relational databases were architected around the assumption that they would be run on a single machine. Also, they were architected before the internet gained massive popularity. The volume of data that can be created by millions, or billions, of networked humans and devices, is far more than any single server can handle.

When the workload grows so heavy that no single computer can bear the load, and the most expensive hardware on the market would be brought to its knees by the weight of an application, the only path is to move from a *single* database server to a *cluster* of database *nodes* working in concert.

For a traditional SQL database, architected to run on a single server, this is a painful process. It requires massive investments of time and often involves tradeoffs that sacrifice many of the features that brought developers to these databases in the first place.

By the late 2000's, SQL databases were still extremely popular, but for those who needed scale, NoSQL had become a viable option. Google BigTable, HDFS, and Cassandra are a few examples. These NoSQL data stores are built to scale easily and to tolerate node failures with minimal disruption. But they come with compromises in functionality, typically a lack of joins and transactions, or limited indexes. These are shortcomings that developers have to engineer their way around. NoSQL can scale beautifully, but relational guarantees are elusive.

Distributed SQL is the next evolution of the database

Traditional SQL databases have tried to solve their scale problem (and hold onto their market share) by bolting on features to help reduce the pain of sharding. At the same time, NoSQL systems have been building out a subset of their missing SQL functionality. But neither class of database was architected from the ground up to deliver the transactional guarantees of relational databases *and* the scale of NoSQL databases.

In 2012 Google Research published what has come to be known as the Spanner paper in which they introduce Google Cloud Spanner, a database that was architected to distribute data at global scale and support consistent transactions. This new breed of database is known as Distributed SQL. There are five conditions that must be met in order for a database to fall into the distributed SQL category: scale, consistency, resiliency, SQL, and geo-replication. The presence of each of these capabilities means that a mission-critical workload that is run in multiple regions of the world, can be accessed as a single data store, and can be scaled by simply adding nodes to a cluster. If you want to learn more about Distributed SQL check out this free O'Reilly Distributed SQL book: *What is Distributed SQL?*