11.1.1 The End of the "One Size Fits All" Era?

RDBMSs pay a lot of attention to data consistency and compliance with a formal database schema. New data or modifications to existing data are not accepted unless they satisfy constraints represented in this schema in terms of data types, referential integrity, etc. The way in which RDBMSs coordinate their transactions guarantees that the entire database is consistent at all times (the ACID properties; see Section 14.5 in Chapter 14). Consistency is usually a desirable property; one normally wouldn't want erroneous data to enter the system, nor for a money transfer to be aborted halfway through, with only one of the two accounts updated.

Yet sometimes this focus on consistency may become a burden, because it induces (sometimes unnecessarily) overhead and hampers scalability and flexibility. RDBMSs are at their best when performing intensive read/write operations on small- or medium-sized datasets, or when executing larger batch processes, but with only a limited number of simultaneous transactions. As the data volumes or the number of parallel transactions increase, capacity can be increased by **vertical scaling** (also called scaling up), i.e., by extending storage capacity and/or CPU power of the database server. However, there are hardware-induced limitations to vertical scaling.

Therefore, further capacity increases need to be realized by **horizontal scaling** (also known as scaling out), with multiple DBMS servers being arranged in a cluster. The respective nodes in the cluster can balance workloads among one another and scaling is achieved by adding nodes to the cluster, rather than extending the capacity of individual nodes. Such a clustered architecture is an essential prerequisite to cope with the enormous demands of recent evolutions such as Big Data (analytics), cloud computing, and all kinds of responsive web

applications. It provides the necessary performance, which cannot be realized by a single server, but also guarantees availability, with data being replicated over multiple nodes and other nodes taking over their neighbor's workload if one node fails.

However, RDBMSs are not good at extensive horizontal scaling. Their approach toward transaction management and their urge to keep data consistent at all times induces a large coordination overhead as the number of nodes increases. In addition, the rich querying functionality may be overkill in many Big Data settings, where applications merely need high capacity to "put" and "get" data items, with no demand for complex data interrelationships nor selection criteria. Also, Big Data settings often focus on semi-structured data or on data with a very volatile structure (consider sensor data, images, audio data and so on), where the rigid database schemas of RDBMSs are a source of inflexibility.

None of this means that relational databases will become obsolete soon. However, the "one size fits all" era, where RDBMSs were used in nearly any data and processing context, seems to have come to an end. RDBMSs are still the way to go when storing up to medium-sized volumes of highly structured data, with strong emphasis on consistency and extensive querying facilities. Where massive volumes, flexible data structures, scalability, and availability are more important, other systems may be called for. This need resulted in the emergence of NoSQL databases.

11.1.2 The Emergence of the NoSQL Movement

The term “NoSQL” has become overloaded throughout the past decade, so the moniker now relates to many meanings and systems. The name “NoSQL” itself was first used in 1998 by the NoSQL Relational Database Management System, a DBMS built on top of input/output stream operations as provided by Unix systems. It implements a full relational database to all effects, but foregoes SQL as a query language.

Drill Down

The NoSQL Relational Database Management System itself was a derivative of an even earlier database system, called RDB. Like RDB, NoSQL follows a relational approach to data storage and management, but opts to store tables as regular textual files. Instead of using SQL to query its data, NoSQL relies on standard command-line tools and utilities to select, remove, and insert data.

The system has been around for a long time and has nothing to do with the more recent “NoSQL movement” discussed in this chapter.

Drill Down

The home page of the NoSQL Relational Database Management System even explicitly mentions it has nothing to do with the “NoSQL movement”.

The modern **NoSQL** movement describes databases that store and manipulate data in other formats than tabular relations, i.e., non-relational databases. The movement should have more appropriately been called NoREL, especially since some of these non-relational databases actually provide query language facilities close to SQL. Because of such reasons, people have changed the original meaning of the NoSQL movement to stand for “not only SQL” or “not relational” instead of “not SQL”.

What makes NoSQL databases different from other, legacy, non-relational systems that have existed since as early as the 1970s? The renewed interest in non-relational database systems stems from Web 2.0 companies in the early 2000s. Around this period, up-and-coming web companies, such as Facebook, Google, and Amazon, were increasingly being confronted with huge amounts of data to be processed, often under time-sensitive constraints. For example, think about an instantaneous Google search query, or thousands of users accessing Amazon product pages or Facebook profiles simultaneously.

Often rooted in the open-source community, the characteristics of the systems developed to deal with these requirements are very diverse. However, their common ground is that they try to avoid, at least to some extent, the shortcomings of RDBMSs in this respect. Many aim at near-linear horizontal scalability, which is achieved by distributing data over a cluster of database nodes for the sake of performance (parallelism and load balancing) and availability (replication and failover management). A certain measure of data consistency is often sacrificed in return. A term frequently used in this respect is **eventual consistency**; the data, and respective replicas of the same data item, will become consistent in time after each transaction, but continuous consistency is not guaranteed.

The relational data model is cast aside for other modeling paradigms, which are typically less rigid and better able to cope with quickly evolving data

structures. Often, the API (application programming interface) and/or query mechanism are much simpler than in a relational setting. The Comparison Box provides a more detailed comparison of the typical characteristics of NoSQL databases against those of relational systems. Note that different categories of NoSQL databases exist and that even the members of a single category can be very diverse. No single NoSQL system will exhibit all of these properties.

Comparison Box

	Relational databases	**NoSQL databases**
Data paradigm	Relational tables	Key–value (tuple) based Document based Column based Graph based XML, object based (see Chapter 10) Others: time series, probabilistic, etc.
Distribution	Single-node and distributed	Mainly distributed
Scalability	Vertical scaling, harder to scale horizontally	Easy to scale horizontally, easy data replication
Openness	Closed and open-source	Mainly open-source
Schema role	Schema-driven	Mainly schema-free or

		flexible schema
Query language	SQL as query language	No or simple querying facilities, or special-purpose languages
Transaction mechanism	ACID: Atomicity, Consistency, Isolation, Durability	BASE: Basically Available, Soft state, Eventually consistent
Feature set	Many features (triggers, views, stored procedures, etc.)	Simple API
Data volume	Capable of handling normal-sized datasets	Capable of handling huge amounts of data and/or very high frequencies of read/write requests

In the remainder of this chapter, we will look closely at some NoSQL databases, and classify them according to their data model. XML databases and object-oriented DBMSs, which could also be kinds of NoSQL databases, were already discussed extensively in previous chapters. Therefore, we will focus on key–value-, tuple-, document-, column-, and graph-based databases. Besides their data model, we will also discuss other defining characteristics of NoSQL databases, such as their capability to scale horizontally; their approach toward data replication and how this relates to eventual consistency; and NoSQL DBMS APIs and how they are interacted with and queried.

Connections

For a discussion on object-oriented database management systems (OODBMSs), see Chapter 8. For XML databases, see Chapter 10. Both can also be regarded as kinds of NoSQL databases.

Retention Questions

- What is meant by vertical scaling and horizontal scaling?
- Which type of scaling are RDBMSs not good at? Why?
- What does the NoSQL movement describe? Is this an appropriate name?
- List the differences between relational databases and NoSQL databases.