Chapter 6
Integration of Relational and NoSQL Databases

ABSTRACT

The chapter proposes three ways of integration of the two different worlds of relational and NoSQL databases: native, hybrid, and reducing to one option, either relational or NoSQL. The native solution includes using vendors' standard APIs and integration on the business layer. In a relational environment, APIs are based on SQL standards, while the NoSQL world has its own, unstandardized solutions. The native solution means using the APIs of the individual systems that need to be connected, leaving to the business-layer coding the task of linking and separating data in extraction and storage operations. A hybrid solution introduces an additional layer that provides SQL communication between the business layer and the data layer. The third integration solution includes vendors' effort to foresee functionalities of "opposite" side, thus convincing developers' community that their solution is sufficient.

INTRODUCTION

Starting any information technology project that will result in new software almost certainly involves selecting an appropriate database. The market is big, so making a choice is not easy and requires examining features. Though all databases have the same or similar purpose of storing and extracting data,

DOI: 10.4018/978-1-5225-3385-6.ch006

there are many differences among them. Selecting a programming language plays a significant role in achieving the ultimate goal of a project; choosing the appropriate database is important as well. What is important when selecting a database? Project objectives drive the answer to this question, along with some global expectations of modern information systems. These expectations include a large number of users, high availability, and throughput of the system with huge amounts and consistency of data.

In the present world of software development, the dominant solution for high scalability and throughput requirements is NoSQL databases. However, this selection imposes upon business-layer programmers the task of solving a number of deficiencies that the world of relational databases had previously solved and standardized. Integrity and consistency of the data based on ACID transactions are the foremost of these problems. Developers spend huge amounts of time developing complex software mechanisms to manage the eventual consistency of data. Bembach (2014) implies the need for a thorough knowledge of data consistency theory as a prerequisite for work on such development solutions. Nevertheless, most software developers think in "transactional" terms, thanks to their education and to development trends that prevailed in the world until recently. Google Spanner's development team (Corbett et al., 2013), defining the rationale behind its solution, set the following thesis:

We believe it is better to have application programmers deal with performance problems due to overuse of transactions as bottlenecks arise, rather than always coding around the lack of transactions. (p. 8)

Developers have a particular problem in the absence of SQL, the standardized structured query language. Every NoSQL system has its implementation of nonSQL programming solutions to run queries on a database. Comparing these solutions with SQL databases, most relational databases are found to offer developers a much richer software interface than NoSQL databases that do not have SQL available. When evaluating usability, performance, and user-friendliness of various query-programming languages of databases, one usually asks questions such as:

- Does the language support aggregate functions?
- Does the language support windows functionality when working with huge amounts of data?