FOR ENTERPRISE ARCHITECTURE PROFESSIONALS

Speedment

Speedment™ - the Java Stream ORM

For Accelerated SQL Database Applications

March 2, 2017

Why Read This White Paper?

The amount of data, the number of transactions and the relationships between data grow more complex and become more dynamic every day. On top of this user demands for agile real-time applications and features are determining the business success for many companies. To solve the performance problems, organizations previously relied on new and more powerful hardware and redesign of application architecture.

This paper addresses these challenges for existing slow databases and presents a more modern solution – Speedment, a Stream ORM Java Toolkit and Runtime with extreme capabilities using an in-JVM-memory data store. It speeds up development time and production performance by orders of magnitudes.

Speedment™- the Java Stream ORM

For Accelerated SQL Database Applications

By Dan Lawesson, PhD
With Carina Dreifeldt and Per Minborg
March 3, 2017

Table of Content

1 Executive Summary

For a typical relational database based application, Speedment cuts costs across the board - savings are made on software development, maintenance and hardware.

The Speedment Java stream ORM leverages standard Java 8 streams to allow fast development of relational database applications that run orders of magnitude faster without upgrading the server hardware. This is possible since the Speedment toolkit fully handles the SQL specifics of the application. Abstracting away the query language, the development becomes faster and less error prone and the execution of the application is accelerated by an in-JVM-memory data store. The speedup of both development and execution time is used by customers to increase customer loyalty, attract new customers, streamline operations, and avoid the costly alternative of proprietary software development.

2 The Problem to Write New Modern Fast Applications for Existing Slow Databases

With rapidly increasing data volumes, new market demands for more advanced analytics and evolving application frameworks, a legacy database solution becomes slow and costly to maintain. Solving the problem by adding hardware or migration to a different database engine entails high costs. In the following we will describe three major challenges.

- **Bottlenecks.** As data volumes grow quickly the database engine performance becomes a bottleneck.
- **Cost.** Increased development and maintenance costs driven by a higher demand for new features.
- **Old APIs.** Legacy applications tied to old APIs are costly to maintain and slow down feature growth.

2.1 The Need for Acceleration of Legacy Database Applications

Many companies have had their databases for decades and the amount of data is growing every day. New modern applications put higher demands on the database in terms of latency and bandwidth. The database then becomes a bottleneck since migration to a more powerful or modern database is often deemed too expensive or even impossible.

Obviously, postponing or avoiding database migration means that new applications with higher database engine demands cannot be realized. This trade-off between cost of migration versus cost of missed market opportunities means a steep price for the organization no matter which path is actually chosen.

Scaling up the server hardware will at best give a linear performance improvement - doubling the computational power may give as much as double the bandwidth, but not more. This may often prove insufficient to keep up with increasing demands that are growing exponentially, and therefore applications with high demands on the database typically need a software upgrade to a database solution more suited to the application in question. For Big Data applications, SQL is often considered infeasible. Upgrading the database software to a different type of engine entails not only costs of server migration but also fundamental application redesign, all the way from the data model to the application logic.

Clearly, the costs of migration thus involve hardware and software but also downtime and maintenance. There are also risks involved - migration is a disruptive operation that if not carefully planned and executed may result in data corruption or loss of data.

2.2 High Cost Development Means Missed Market Opportunities

A legacy application using a relational database entails high development and maintenance costs since the application contains code that is tightly coupled to the database engine interface. Developing a new feature or adjusting an existing feature in the application that requires any change in the data model will also require changes in the code adapting between the database and the application. Therefore, seemingly small changes in the requirements create a need for adapting code at several levels of the system design.

A high cost of maintaining and developing new features does not only constitute a cost per se, but does also deter from creating new features due to cost and risk of introducing regression problems. This lock-in effect to an existing set of features is the very opposite of the agile characteristics of modern agile application development.

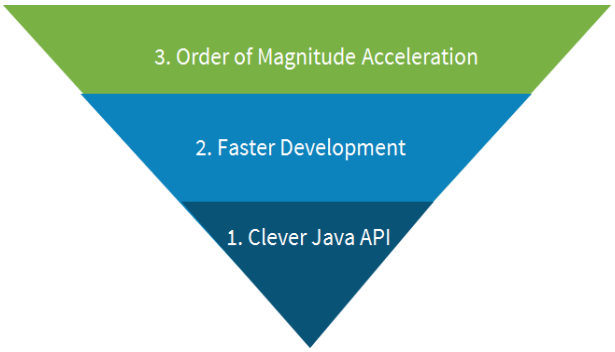
2.3 Legacy Application APIs Stall Development

A legacy application will inevitably be using legacy language constructs and probably outdated database specific code. In particular, modern Java language features such as streams are not supported by an application written before Java 8 and an old framework or customized code for database access means an interface that is not streams oriented. To allow modern server side stream based application development the legacy solution needs to be adapted with increased development and maintenance costs as result. Since the legacy code is not designed to support streams, a simple adaptation will add overhead and will not yield the same benefits as a system designed for streams.

Depending on a particular tailor-made legacy API for database access stalls development since the features of the legacy API will constrain the application to the functionality available at the time when the API was designed. Thus, using a special API based on for example a query language locks the application design to the state of the art at the time of the API design. Leveraging new features of the Java language such as streams then requires major redesign.

3 The Speedment Solution

Speedment addresses the three problems above by providing database access acceleration by orders of magnitude and also faster and less error prone application development enabled by a clean and clever API based on standard Java streams.



3.1 Future-Proof Application Design Using Clever Standard APIs

An application leveraging a legacy database typically adopts dated design patterns with high maintenance costs as well as high costs for migrating the code to a more modern framework. The Speedment API allows modern future-proof application design based on fundamentals of the Java language itself and no domain specific API.

The functional approach to scalable and modular application development is gaining momentum as the community of Java developers is eagerly migrating to Java 8. Widely heralded as the most fundamental step of improvement in many years, Java 8 modernizes the language to gradually step into the realm of functional programming allowing for more efficient, clear and elegant code that therefore also is fundamentally less costly to maintain. Based on pure Java 8 streams, the Speedment API allows the user to access a legacy database with a modern stream based API with lower cost of development, maintenance and testing as a result. Since no specialized interface is needed for Speedment, adopting your business logic to Speedment only amounts to future proofing the application by leveraging the latest improvements of the Java language itself.

As a simplistic example, the following code can be used to count the users in a particular department using Speedment.

```
long count = users.stream()
  .filter(User.DEPARTMENT.equal(1))
  .count();
```

Note how no query language is needed, only standard Java streams operations. This piece of code will call Speedment generated code that uses the Speedment runtime to send the following query to the database engine.

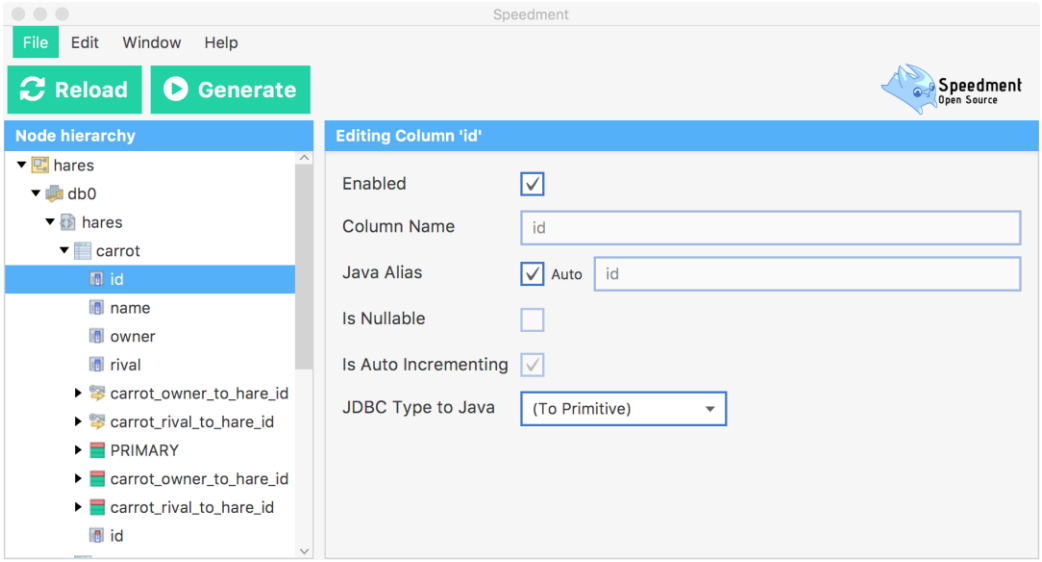
```
SELECT count(*) FROM users WHERE department = 1
```

An API that is fully standard compliant and uses only constructs native to the Java language gives a future-proof application and allows for cost efficient maintenance. In the following section we describe how Speedment accelerates the development process itself.

3.2 A Framework for Fast Development of Database Applications

Speedment is a Java Stream ORM toolkit and runtime for scalable real-time applications using relational databases. The toolkit analyzes the metadata of an existing legacy SQL database and creates a Java representation of the data model that together with the Speedment runtime allows the user to create scalable and efficient Java applications leveraging the latest improvements to the Java language. The user of Speedment interacts with the database using Java 8 streams and no query language such as SQL is needed.

FIGURE 1 Speedment User Interface .



Since the Speedment toolkit generates the code used to access the entities of the data model, that part of the application code base will be maintained by the toolkit itself, simplifying maintenance and cutting costs. The generated code can be inspected, modified and enhanced as needed. Coding resources can focus on real problems at hand instead of maintaining boilerplate code. The extendible API of Speedment allows seamless integration of user defined code generating rules. Elegant and general by the core, the API does not just allow customizing details of the generated code but can be used to plug-in support for new languages. A well-balanced level of abstraction allows the user full flexibility to access any aspect of the data while doing away with the tedious and error prone work of maintaining the code base consistency with respect to the database model.
When any part of the database model is altered, be it a field changing type or whole tables being added or removed, the Java code representing the data will be automatically brought back in sync

with the new data model by the Speedment toolkit. Changes needed in the application to accommodate the database evolution will then be found by the compiler, that is even before any testing has begun, let alone any field trials. Being able to find problems early may be the most important feature of a maintainable and scalable system, since the cost of bugs found late typically grows rapidly with the size of the code base.

Thus having established the way Speedment accelerates the development process by code generation and abstraction of SQL queries we have laid a foundation of conceptual building blocks needed to understand Speedment application acceleration.

3.3 In-JVM Memory Acceleration of a Legacy Relational Database

When the Speedment toolkit and runtime abstracts away all the details of the actual SQL queries it does not only relieve the user of the error-prone and uninspiring labor of maintaining boilerplate code as described above. Perhaps even more importantly this abstraction also creates the opportunity to do better than SQL. Using an in-JVM-memory data store leveraging data grid technology, Speedment can make use of system resources in a more efficient way and decrease data latency by orders of magnitude compared to direct use of the relational database.

Many organizations put huge efforts into building or integrating their own, customized cache code or write complicated optimized SQL queries. This may prove harder than anticipated and not pay off as much as anticipated as well as introduce steep maintenance costs. The learning curve can be hard to climb. Because the complexity of the caching field becomes apparent after some time, it is common for in-house code solutions to undergo several, costly refactoring phases.

Speedment supports an event driven architecture that greatly simplifies coding. This provides the opportunity to have "live" objects in applications, GUIs and web pages with a minimum of coding and no overhead. Events also eliminate resource consuming database polling. As Speedment uses a same-side architecture, it can be much faster than other solutions. Query data does not need to be transmitted over any network. Thus, response times are low, capacity is up and Speedment does not impose load on your internal data network. Still, Speedment allows for remote connections if this is required by system designers.

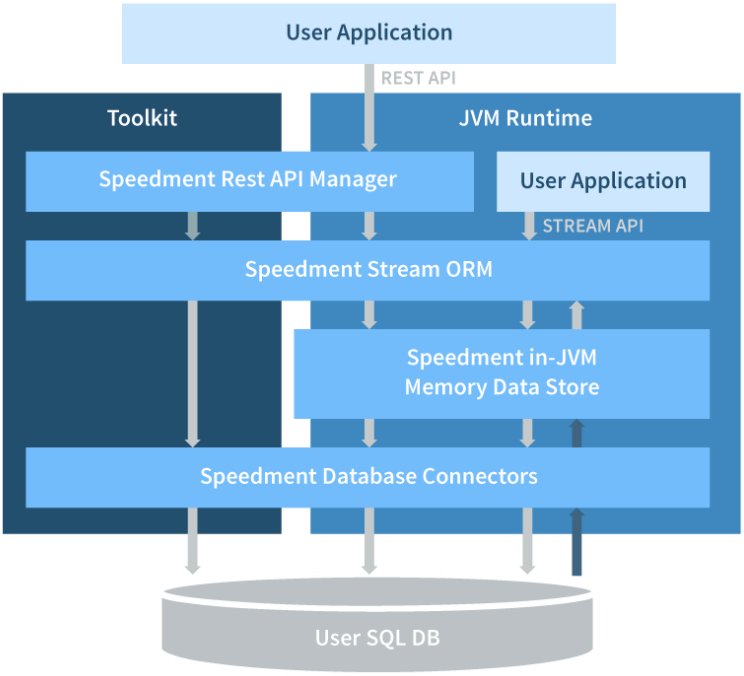
Web pages are growing more and more interactive. Speedment supports modern web pages that typically require a large number of real time queries to be served under harsh latency requirements. Speedment supports secure direct access over the Internet for ease of integration with Flash, javascript and xml access.

4 Architecture Overview

Speedment is a Java toolkit and runtime for accelerated SQL database applications leveraging In-JVM Data Store technology. The toolkit generates code to encapsulate the specifics of the used database contents and the runtime provides the runtime data handling functionality for the user application.

User applications may use two different APIs mainly depending on whether the user application is server or client side. For a client side application Speedment can deliver the data over a REST API, and for a server side application coexisting with the Speedment runtime there is a Java 8 streams based API.

FIGURE 2 Speedment Architecture.



5 Case Study

The following case study will give an order-of-magnitude estimate of real world numbers of execution speed improvements using Speedment. It will also showcase how the performance gains are achieved along two orthogonal vectors; query latency is reduced by orders of magnitude by the in-JVM-memory data store and the effective load of the underlying database is reduced since Speedment handles most of the queries.

A multinational company had severe performance issues in their trouble ticket system. Query times for specific tickets were in the order of minutes even when measuring single user usage, let alone when many users worldwide were creating and searching for tickets. By using the Speedment in-JVM-memory solution, database latency of single queries was reduced by orders of magnitude as shown in the following table.

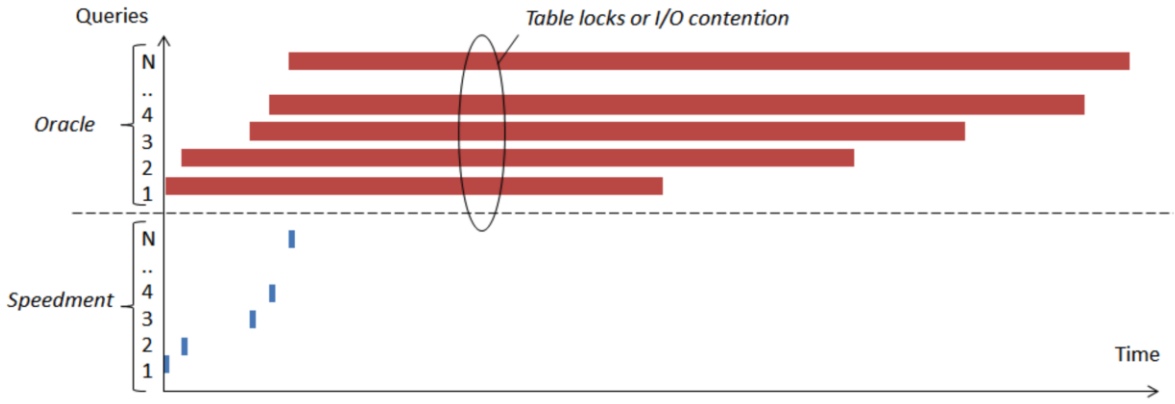
FIGURE 3 Usecase performance measurements.

Query Type	Type Example	Oracle SQL	Speedment
Table Info	How many reports are there in total?	0.5 min	0 s
Selection	Which trouble tickets have no further related information in other tables?	4.3 min	0 s
Like (standard)	Which reports are assigned to persons with a name that starts with “JU”?	6 min	15 s
Like (optimized)	(Same as above but with the query optimized with Speedment in mind.)	6 min	0.5 s
Function	How many SLA reports are still open?	6.3 min	0 s

The improvements were further magnified when comparing the actual use case impact since the legacy Oracle SQL solution was riddled with lock and I/O contention caused by high latency queries hogging the database access for other users, resulting in response times sometimes up to half an hour. Speedment employs a CQRS design where reading and writing of data is separated by using different models, thereby greatly reducing problems of lock and I/O contention.

The large amount of data reads are handled directly by the in-JVM-memory data store, allowing read operations to take place concurrently with other read operations as well as write operations. The write operations are allowed to execute in the database without any contention with the read operations as visualized in the following graph.

FIGURE 4 The Speedment CQRS solution relieves the database of the read operation impact, drastically decreasing I/O contention.



6 Conclusion

In this paper we have shown how businesses with a demand for low latency access to a high data volume relational database can benefit from working with new technology and use software instead of hardware to build scalable high performance systems for agile real-time market demands.

A major obstacle for many companies is that the current application software monolith that are unable to meet modern efficiency demands. With the Speedment Toolkit and Runtime there is no need for risky and costly migrations. The hardware and the database itself will be reused and perform even better than most NoSQL solutions.

A solution leveraging Speedment is not restricted to a particular ecosystem. Over time the solution provides elastic scalability, fast creation and deployment for new features and applications as well as easy maintenance and transparency in the code generated.

470 Ramona Street
Palo Alto 94301 CA, USA
+1 650 387 4069, info@speedment.com