Cutter IT Journal

CUTTER CONSORTIUM
ACCESS TO THE EXPERTS

21 January 2015

The State of Database Access in Java: Passchendaele Revisited

by Bart Baesens, Aimée Backiel, and Seppe vanden Broucke

This year marks the centennial of the start of the First World War. One of the fiercest battles in WWI was the Battle of Passchendaele in Belgium. This manslaughter took place from July-November 1917, with more than 500,000 men lost on both sides for only a few kilometers gained, which were retaken soon afterwards during the German Spring offensive. It was characterized by ferocious fighting in terrible conditions -- think mud, unceasing rain, and cold weather, but with what were "state-of-the-art" technologies at the time (e.g., tanks, gas, flamethrowers) -- and with the loss of precious resources. The battle symbolizes the horror of the Great War in every respect.

What does this history lesson have to do with the state of database access in Java? The answer is, of course, not a lot. But in this *Advisor*, we want to take you on a history tour through the different technologies that were once considered state of the art concerning database access in Java before they faded out and were replaced with the next fad. At the end of the day, the astute observer cannot help but wonder if we haven't ended up in a situation that is similar to the Battle of Passchendaele: state-of-the-art equipment and technology, but still struggling through the mud. Let us see how this state of affairs came to be, beginning 15 years or so ago.

In 1998, I culminated my studies with a master's thesis entitled, "Object Relational Database Access," in which I explored various ways of reconciling OO programming languages with relational databases. My key findings were that SQL is a very important database manipulation language, OO databases are too complex, and Object Relational Mappers (ORMs) were the best option at the time (this was three years before the advent of Hibernate). When we started writing a book about a year ago entitled *Beginning Java Programming: The Object-Oriented Approach* , a primary focus was accessing databases in Java. After reviewing today's state-of-the-art technologies, we came to the conclusions we describe here.

Custom SQL

The most popular approaches currently adopted in the industry are Java DataBase Connectivity (JDBC) and Hibernate. JDBC is a popular standardized API that gives database-independent access to tabular data typically stored in relational databases or even Microsoft Excel. The technique is characterized by the fact that many bindings are available for it (making it easy to access a wide range of databases) and you can talk at a fairly low level with the

database, meaning that you send standard SQL and get back a result set that you can deal with in your program as you desire. Hibernate is the most popular example of an ORM, a type of middleware that assists in converting data coming from relational databases (which typically are not very object-aware) to objects that are directly usable in your OO programming environment. This allows programmers to talk directly to objects (both for retrieval and saving), but comes at a cost of flexibility, and -- in some cases or when used incorrectly -- speed. Despite this, ORMs have become commonplace in industry environments, with Hibernate still at the forefront. Even although a standard API was added to Java to deal with ORM and data persistence in general, called the Java Persistence API (JPA) which was heavily inspired by Hibernate, most practitioners today still prefer to bypass this standard altogether and go straight to Hibernate.

NoSQL Databases

In our book, we conclude that, for simple applications (working with only one database and let's say less than 10 relational tables), it is advisable to use JDBC, since it's quite easy to set up and work with compared to Hibernate, which has a rather steep learning curve and bigger footprint. Hibernate is typically recommended when working with complex database models consisting of hundreds of relational tables with complex relationships. A key benefit of Hibernate is that it allows you to completely abstract away the complex underlying database design. Managing all these tables and relationships in JDBC would be a very cumbersome exercise. Since many professional software development methodologies are object oriented, another key advantage of Hibernate is that it provides a straightforward mapping from a conceptual OO model to a Java application, since you don't have to bother with relational database design issues. Hibernate is also a very portable solution, making it easy to switch to another ORM if desired. However, a key concern of many Java developers working with ORM frameworks is that many ORMs could benefit from further query optimization and tuning using, for example, improved indices and caching. Because of this performance issue, some developers use a mixed approach, whereby they use native SQL for read operations (which typically make up the majority of an application anyway), and Hibernate for the remaining create, update, and delete operations.

Object Relational Mappers

Apart from recent efforts in improving the aforementioned ORMs, a new type of database has sprung up that rightfully asks: "If the goal is to map a relational structure to objects, then why not move away from relational databases completely?" This led to the development of "pure" OO databases, which deal with objects and their state instead of tables and records, thus doing away with the need for ORM (some of these are even implemented in Java itself). Despite their intrinsic advantages, OO databases are seldom used because they are often perceived (rightfully or not) as complex to work with.

ISSN: 1554-5946

When looking at what's ahead, we see that databases are massively expanding in size. IBM projects that we generate 2.5 quintillion bytes of data every day. In relative terms, this means that 90% of the data in the world has been created in the last two years. As such, new database technologies have to be and have been introduced to efficiently cope with this tsunami of data. NoSQL is one of these newer technologies. NoSQL databases abandon the well-known and popular relational database scheme in favor of a more flexible, schema-less database structure that more closely aligns with the needs of a big data generating business process. One key advantage of NoSQL databases is that they more easily scale horizontally in terms of storage. Four popular types of NoSQL database technologies are: key-value-based, document-based, column-based, and graph-based databases. Despite the name NoSQL, many of these database systems still provide active support for SQL to manipulate the data (this explains why the term is nowadays expanded to "Not Only SQL" instead of the original "No SQL"). Moreover, in response to the NoSQL stream, some vendors have come up with NewSQL database products by equipping traditional RDBMs with facilities to provide the same scalability as their NoSQL counterparts. A popular example here is Google's Spanner. Hence from a Java programmer's perspective, it will remain important to know the basic concepts of SQL in order to develop high-performing Java database applications.

In summary, we have observed that SQL still remains a very important database manipulation language. While ORM goes a long way, we nevertheless still see practitioners reaching for custom SQL every time a complex reporting query with 10 joins needs to be written (and rightly so -- don't underestimate the raw power of an optimized query). Second, pure OO databases are deemed too complex, and currently risk having missed their time to shine altogether in favor of newfangled NoSQL databases. Finally, the cool-headed conclusion at the end of the day remains that ORMs are the best option available. Notice any differences from 10 years ago? Neither did we. It seems the Battle of Passchendaele provides a fitting metaphor here: a terrible amount of resources have been wasted for a limited amount of "gain." On the upside, however, technologies that have survived the hand of time (such as Hibernate) have become increasingly better tested, documented, and are overall more straightforward and easier to get started with and get working.

Java 8 is at the horizon, promising yet another wave of "revolutionary" technologies such as Jinq (inspired by .NET's LINQ) and other stream-based, declarative database APIs. We'll see what the future brings, but we'd advise eager adopters to learn from history and -- perhaps a little bit -- from World War I. At the very least, we can be thankful no one is using SQLJ any more.

For more information, please see our new book, *Beginning Java Programming: The Object-Oriented Approach* . We welcome your comments about this

Advisor and encourage you to send your insights to us at comments@cutter.com.

-- Bart Baesens, Aimée Backiel, and Seppe vanden Broucke

Cutter Consortium | 37 Broadway, Suite 1, Arlington, MA 02474, USA. | Tel: +1 781 648 8700 | Fax: 781 648 8707 | www.cutter.com

This article demostrates how to access MS Access databases from Java. Introduction. This article explains how to use the JDBC ODBC bridge to access an MS-Access database from Java applications. Instead of elaborating on the basics of the database, let's get down to the subject. ODBC driver. In Java, we require a driver to be loaded at runtime to connect to any data source.Â In Java, we would write a similar connection string, but there would be an additional specification that points to the driver that will be required for the connection, that is, jdbc:odbc:. Then, follow it up with the connection string. So the connection string in Java becomes: "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)}; DBQ=myDB.mdb Map location of Passchendaele. No one alive in this generation has lived through the great war. The last combat veteran was Claude Choules who served in the British Royal Navy (and later the Royal Australian Navy) and died 5 May 2011, aged 110.Â The Battle of Passchendaele, also known as the Third Battle of Ypres, was a major campaign of the First World War, fought by the Allies against the German Empire. The battle took place on the Western Front, from July to November 1917, for control of the ridges south and east of the Belgian city of Ypres in West Flanders. The battle lasting a 100 days, claimed British casualties at 244,897 and estimated German losses at 400,000. The city of Passchendaele before and after the battle.