Annals of DAAAM for 2011 & Proceedings of the 22nd International DAAAM Symposium, Volume 22, No. 1, ISSN 1726-9679
ISBN 978-3-901509-83-4, Editor B. Katalinic, Published by DAAAM International, Vienna, Austria, EU, 2011
Make Harmony between Technology and Nature, and Your Mind will Fly Free as a Bird
Annals & Proceedings of DAAAM International 2011

HETEROGENEOUS DATABASE INTEGRATION USING HIBERNATE MAPPING FILES

POPA, G[eorge] D[an]

Abstract: *Hibernate ability to work with multiple types of database management systems can be used to integrate a heterogeneous database. Configuring different DMBS with XML mapping files, we can construct a working distributed environment to serve external applications as a single data source. This paper describes a way to achieve both database replication and segmentation using Java and Hibernate technology on a large number of commercial and academic database management systems.*
Key words: *database, heterogeneous, hibernate, replication, segmentation*

1. INTRODUCTION

Heterogeneity in database systems appears at different levels, from technological and structural diversity to semantic meaning of stored data. Distributed environments face new challenges when dealing with heterogeneity, at any level, because of the technological adjustments, data significance and communication issues. The features of distributed systems, such as replication, load balancing or segmentation, cannot be lost due to heterogeneity. This paper is describing a solution to integrate several commercial DBMS into one working environment.

Several database architectures are used for heterogeneous environment, dealing with one or more features of distributed environments, such as Sybase Replication Server (Wiebener, 2010), IBM Sync4J (Suryanarayana, 2005), and many academic solutions. However, none of them offers complete distributed support and most of them are limited to a certain list of backend technologies.

Our solution aims to overcome these limitations, providing a working example, as well as steps for further integration of other relational DBMS that are not mentioned in this paper.

2. SYSTEM ARCHITECTURE

In order to offer full compatibility with all major platforms we used in the proposed architecture the Java Sun technology and Hibernate from JBoss Community, although other tools were available too, for database integration. Client applications can use the distributed database through the Java middle layer, which is transparent to the end user. The Java middle layer, responsible for the database system management, receives database calls from the client applications (in form of either standard SQL queries or Hibernate Query Language) and based on the federation rules of the database system, it forwards the queries to the proper DBMS, and then returns the results to the client.

Considering that the queries will access more than one element of the database system, the Java middle layer application will first prepare the query, splitting it in sub queries for each element and then will translate them to the proper SQL dialect. In the same way, after retrieving results, it will combine them together before returning to user.

The image below shows the system architecture having a central element, Java application with Hibernate (Fig. 1):

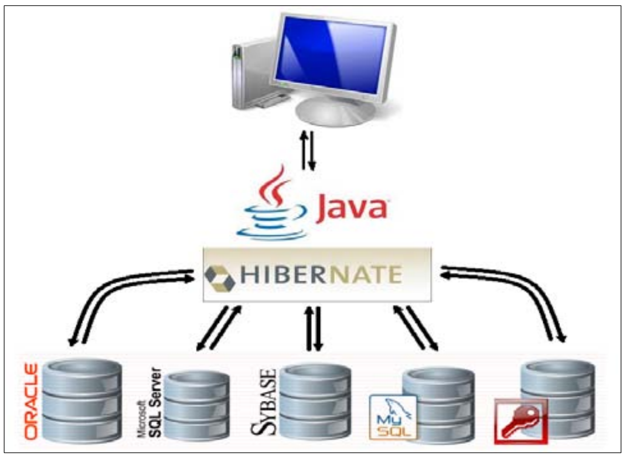

Fig. 1. Centralized environment architecture

Some of the DBMS specific operations, such as ordering, grouping and indexing have to be coordinated by the Java application, in order to retrieve correct results per system.

To have a fully distributed environment, we can split the Java middle layer application, as one for each database element, installed as an agent on their local machine. End user clients can connect to any of these agents (preferably the closest one) and thus access the entire database system (Fig. 2).

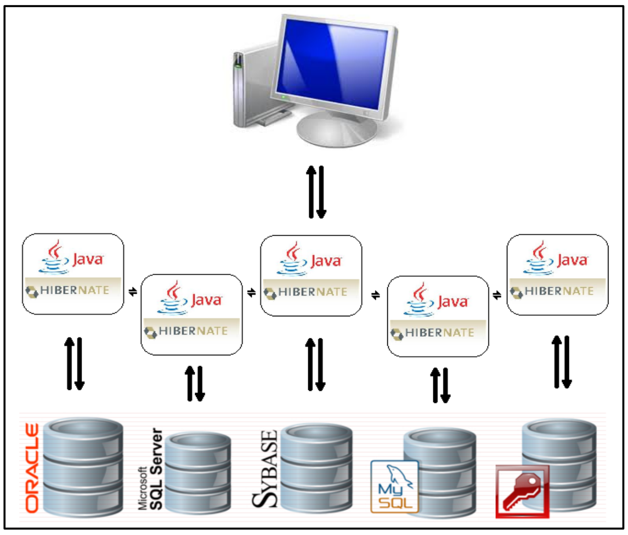

Fig. 2. Java and Hibernate distribution in middle layer

Separating the middle layer leads to index sharing and data transfer between agents. The main advantages of this architecture are the lack of critical points and the load balancing options.

3. ENROLLING MEMBERS

Depending on the chosen architecture and database system functionality, members can be enrolled live, after an agent is installed on their machine.

The decentralized architecture makes automatic enrolling difficult, so one new agent must manually contact any other member, in order to exchange configuration files. The configuration files refer to:

- Connection coordinates and credentials for every agent in the federation;
- Virtual schema description, visible to end users;
- Hibernate mapping files.

The connection coordinates and credentials file contains information about all active members to which a client can connect to. After one member is enrolled, this file is updated and exchanged between all members.

The virtual schema description file is the main database system configuration file and contains metadata of the usable table, views and procedures. This file is also visible by the client applications, which can edit it in order to manage the desired schema. Changes to this file may not affect all members, but it must be acknowledged by all.

The Hibernate mapping files define the actual database tables and connect them with the virtual tables, seen by the users. The mapping files are specific to each database provider and are not shared between members, unless there is table segmentation or schema / table replication. At this level we can configure distributed database properties, such as schema / tables replication or segmentation.

More than that, Hibernate offers a tool for horizontal segmentation, called Hibernate Shards, which is useful in optimizing queries over multiple data sources (Glover, 2010).

If one member is opting out of the distributed system, or is unavailable after a timeout period, it is removed from the environment. This action may lead to performance penalties or data loss, depending on the system configuration (load balancing, schema replication or segmentation).

4. REPLICATION AND SEGMENTATION

The main target of a distributed database environment is to ensure one or more of its features: load balancing, replication and segmentation. As seen in chapter 2, “System Architecture”, using multiple Java agents on every database machine allows us to implement load balancing, using a load balancer installed in front of the Java application, such as Apache HTTP server.

Database replication is the environment’s property to ensure that precious data is saved into multiple targets. The Hibernate mapping files allows us to configure several data sources to our application, specifying the same virtual schema or table to more existing DBMS. Every time the client application is sending a *commit* command to the distributed database, the data is inserted (or updated, removed) in all corresponding data sources. The operation is considered completed only when all affected databases return the success commit code, or else it is rolled back. The data query process is much easier, regarding the synchronization issues. Members enrolling into a replicated environment may copy existing data from other members, or share its data with them.

Database segmentation, on all levels, is available in our distributed system. Hibernate Shards is managing the horizontal segmentation, storing a global index, filtering, sorting and grouping in all segments. Even if it is a good framework for database replication, Hibernate Shards will only be used for segmentation, as advised in its documentation. Vertical segmentation is easier to obtain in our case, as we treat two or more groups of columns from one table as if they were separate tables.

Primary keys and foreign keys are our only special concerns, but these are solved if the operations that involve segmented tables are run transactional.

5. RESULTS AND OPTIMIZATIONS

There are infinite ways to configure a distributed database, but not all of them run as efficient as expected. Heterogeneous environment requires special attention during design phase, in order to achieve the best results. In our case, we planned to build a distributed system, serving several external applications as clients, connecting to agents according to their load, and supporting both horizontal and vertical segmentation on a very large dataset and replication on sensible tables. The available backend providers are Oracle, MySQL and SQL Server DBMS.

The first guidelines were to use Oracle as support for the large dataset (Boicea et al., 2010), MySQL server as a replicated environment and SQL Server machine to serve most of the concurrent clients. After testing several configurations and system tuning the following best practices were confirmed:

- Database vertical segmentation should be done on unrelated tables, to avoid joins and foreign keys in separate segments;
- Horizontal segmentations should take in consideration the usual data grouping criteria, in order to access as few segments as possible;
- Read only data is easier to replicate; frequent writes should be previously cached and flushed to disk when necessary;
- Hibernate and Hibernate Search support indexing of any kind of data types, including Lucene text search. However, when possible, native indexing support should be used for large data sets.

6. CONCLUSIONS

Java and Hibernate support for the major commercial and academic DBMS is the main reason for choosing this technology as middle layer application. As it is build, not only that it can handle a large set of database providers, but also new ones can be configured just by extending an existing Hibernate dialect. The major goal achieved by the presented model is data portability. Even if performance is much lower than commercial homogenous database systems, the ability to share, compare and summarize live information from various data sources with minimal configuration is very useful.

Further goals that should be considered are extending portability to object-oriented databases and other storage systems, as well as developing a discovery process for potential members in the environment.

7. REFERENCES

Boicea, A.; Crivat, A.; Radulescu, F. & Popa, G. (2010). *Performance Evaluation and Tuning in an Oracle DBMS,* DAAAM 2010, ISBN 978-3-901509-73-5, Vienna

Glover, A. (2010). Java development 2.0: Sharding with Hibernate Shards, *Available from:* http://www.ibm.com/developerworks/java/library/j-javadev2-11, *Accessed: 2011-06-24*

Suryanarayana, J. (2005) Heterogeneous database replication with SyncML, *Available from:* http://www.ibm.com/developerworks/java/library/j-sync4j/, *Accessed: 2011-03-15*

Wiebener, R. H. (2010) Synchronizing Data Among Heterogeneous Databases, *Available from:* http://www.sybase.com/files/White_Papers/RepServer-Sync-Heterogeneous-DB-WP.pdf, *Accessed: 2011-03-20*

*** (2011) http://www.hibernate.org/docs - JBoss Community Hibernate, Community Material, *Accessed on: 2010-11-13*