A Study on Hibernate Framework: Object Relational Mapping Solution For Java

Naresh Purohit[1], Shakti Singh[2]
[1, 2] Dept of Computer Science Engineering
[1]Mahaveer Institute of Technology & Science, Pali, Rajasthan, India
[2]Aishwarya College of Education, Pali, Rajasthan, India

Abstract- *This paper presents a study on Hibernate, an object relational tool for Java based applications. Hibernate is an ambitious project that aims to be a complete solution to the problem of managing persistent data in Java. It mediates the application's interaction with a relational database, leaving the developer free to concentrate on the business problem at hand. Hibernate is a non-intrusive solution. It integrates smoothly with most new and existing applications and does not require disruptive changes to the rest of the application. Hibernate is an open source ORM implementation.*

Keywords- Relational Database, Hibernate, ORM, Open Source

I. INTRODUCTION

Today, many software developers work with Enterprise Information Systems (EIS). When developers work with an object-oriented system, there is a mismatch between the object model and the relational database. RDBMSs represent data in a tabular format whereas object-oriented languages, such as Java or C# represent it as an interconnected graph of objects. First problem, what if developers need to modify the design of database after having developed a few pages or application? Second, loading and storing objects in a relational database exposes us to the following mismatch problems:

Table -1: Configuration Property Classes

Mismatch	Description
Granularity	Sometimes we will have an object model, which has more classes than the number of corresponding tables in the database.
Inheritance	RDBMSs do not define anything similar to Inheritance, which is a natural paradigm in object-oriented programming languages.
Identity	An RDBMS defines exactly one notion of 'sameness': the primary key. Java, however, defines both object identity (a==b) and object equality (a.equals(b)).
Associations	Object-oriented languages represent associations using object references whereas an RDBMS represents an association as a foreign key column.

To overcome these problems, Hibernate is tool that aims to be a complete solution to the problem of managing persistent data in Java. It mediates the application's interaction with a relational database, leaving the developer free to concentrate on the business problem at hand.

First, we define persistent data management in the context of object-oriented applications and discuss the relationship of SQL, JDBC, and Java, the underlying technologies and standards that Hibernate is built on. We then discuss the so-called object/relational paradigm mismatch and the generic problems we encounter in object-oriented software development with relational databases.

A. What is persistence?

Almost all applications require persistent data. Persistence is one of the fundamental concepts in application development. When we talk about persistence in Java, we're normally talking about storing data in a relational database using SQL.[4]

- **Relational Databases**

A relational database management system isn't specific to Java, and a relational database isn't specific to a particular application. Relational technology provides a way of sharing data among different applications or among different technologies that form part of the same application (the transactional engine and the reporting engine, for example). Relational technology is a common denominator of many disparate systems and technology platforms. Hence, the relational data model is often the common enterprise-wide representation of business entities. Relational database management systems have SQL-based application programming interfaces; hence we call today's relational database products SQL database management systems or, when we're talking about particular systems, SQL databases.

- **Understanding SQL**

To use Hibernate effectively, a solid understanding of the relational model and SQL is a prerequisite. We will need to use our knowledge of SQL to tune the performance of our Hibernate application. Hibernate will automate many repetitive coding tasks, but our knowledge of persistence technology must extend beyond Hibernate itself if we want take advantage of the full power of modern SQL databases. Remember that the underlying goal is robust, efficient management of persistent data.

- **Using SQL in Java**

When we work with an SQL database in a Java application, the Java code issues SQL statements to the database via the Java DataBase Connectivity (JDBC) API. The SQL itself might have been written by hand and embedded in the Java code, or it might have been generated on the fly by Java code. We use the JDBC API to bind arguments to query parameters, initiate execution of the query, scroll through the query result table, retrieve values from the result set, and so on. These are lowlevel data access tasks; as application developers, we're more interested in the business problem that requires this data access.

- **Persistence in object-oriented applications**

In an object-oriented application, persistence allows an object to outlive the process that created it. The state of the object may be stored to disk and an object with the same state re-created at some point in the future. An application with a domain model doesn't work directly with the tabular representation of the business entities; the application has its own, object-oriented model of the business entities. If the database has ITEM and BID tables, the Java application defines Item and Bid classes. Then, instead of directly working with the rows and columns of an SQL result set, the business logic interacts with this object-oriented domain model and its runtime realization as a graph of interconnected objects. The business logic is never executed in the database (as an SQL stored procedure), it's implemented in Java. This allows business logic to make use of sophisticated object-oriented concepts such as inheritance and polymorphism.

However, in the case of applications with nontrivial business logic, the domain model helps to improve code reuse and maintainability significantly. We focus on applications with a domain model in this paper, since Hibernate and ORM in general are most relevant to this kind of application.

A. Persistence layers and alternatives

A layered architecture defines interfaces between code that implements the various concerns, allowing a change to the way one concern is implemented without significant disruption to code in the other layers.[4] Layering also determines the kinds of interlayer dependencies that occur. The rules are as follows:

- Layers communicate top to bottom. A layer is dependent only on the layer directly below it.
- Each layer is unaware of any other layers except for the layer just below it.

Typical, proven, high-level application architecture uses three layers, one each for presentation, business logic, and persistence, as shown in figure-1

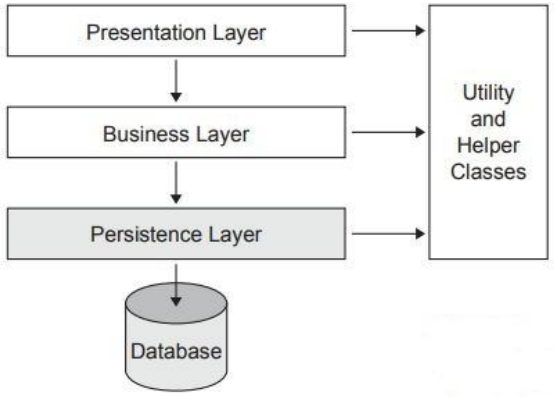

Figure -1 : Layered Architecture

- **Presentation layer**—The user interface logic is topmost. Code responsible for the presentation and control of page and screen navigation forms the presentation layer.
- **Business layer**—The exact form of the next layer varies widely between applications. It's generally agreed, however, that this business layer is responsible for implementing any business rules or system requirements that would be understood by users as part of the problem domain.
- **Persistence layer**—The persistence layer is a group of classes and components responsible for data storage to, and retrieval from, one or more data stores.
- **Database**—The database exists outside the Java application. It's the actual, persistent representation of the system state. If an SQL database is used, the database includes the relational schema and possibly stored procedures.

II. OBJECT - RELATIONAL MAPPING

A. What is ORM?

Briefly, object/relational mapping is the automated (and transparent) persistence of objects in a Java application to the tables in a relational database, using metadata that describes the mapping between the objects and the database.

An ORM solution consists of the following four pieces:

- An API for performing basic CRUD operations on objects of persistent classes
- A language or API for specifying queries that refer to classes and properties of classes
- A facility for specifying mapping metadata
- A technique for the ORM implementation to interact with transactional objects to perform dirty checking, lazy association fetching, and other optimization functions

B. Why ORM?

ORM is an advanced technique to be used by developers who have already done it the hard way. To use Hibernate effectively, we must be able to view and interpret the SQL statements it issues and understand the implications for performance. Some of the benefits of ORM – Hibernate are:

- **Productivity**

Persistence-related code can be perhaps the most tedious code in a Java application. Hibernate eliminates much of the grunt work and concentrate on the business problem. No matter which application-development strategy we prefer—top-down, starting with a domain model, or bottom-up, starting with an existing database schema—Hibernate, used together with the appropriate tools, will significantly reduce development time.

- **Maintainability**

Hibernate application is more maintainable. In systems with hand-coded persistence, an inevitable tension exists between the relational representation and the object model implementing the domain. Changes to one usually involve changes to the other, and often the design of one representation is compromised to accommodate the existence of the other. ORM provides a buffer between the two models, allowing more elegant use of object orientation on the Java side, and insulating each model from minor changes to the other.

- **Performance**

In a project with time constraints, hand-coded persistence usually allows us to make some optimizations. Hibernate allows many more optimizations to be used all the time. Furthermore, automated persistence improves developer productivity so much that we can spend more time hand optimizing the few remaining bottlenecks. ORM software probably had much more time to investigate performance optimizations than we have.

- **Vendor independence**

It is usually much easier to develop a cross-platform application using ORM. Even if we do not require cross-platform operation, an ORM can still help mitigate some of the risks associated with vendor lock-in. In addition, database independence helps in development scenarios where developers use a lightweight local database but deploy for production on a different database.

III. HIBERNATE

Hibernate is an Object-Relational Mapping (ORM) solution for JAVA. It is an open source persistent framework created by Gavin King in 2001. It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application. Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieves the developer from 95% of common data persistence related programming tasks. Hibernate sits between traditional Java objects and database server to handle all the works in persisting those objects based on the appropriate O/R mechanisms and patterns.[2]

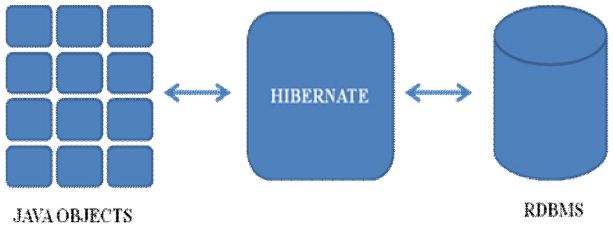

Figure -2: Communication B/W Java Objects & RDBMS using Hibernate

A. Hibernate Architecture

Hibernate has a layered architecture which helps the user to operate without having to know the underlying APIs. Hibernate makes use of the database and configuration data to

provide persistence services (and persistent objects) to the application.

Following is a very high-level view of the Hibernate Application Architecture.

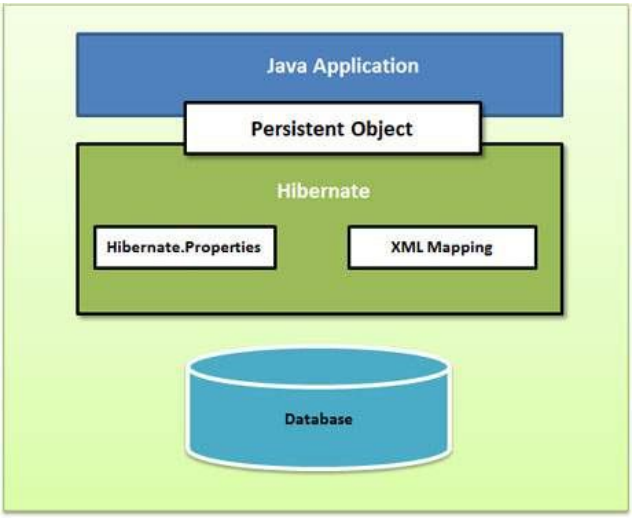

Figure -3: Hibernate Architecture (High Level View)

\

Hibernate uses various existing Java APIs, like JDBC, Java Transaction API(JTA), and Java Naming and Directory Interface (JNDI). JDBC provides a rudimentary level of abstraction of functionality common to relational databases, allowing almost any database with a JDBC driver to be supported by Hibernate. JNDI and JTA allow Hibernate to be integrated with J2EE application servers.[1]

Following is a detailed view of the Hibernate Application Architecture with its important core classes.

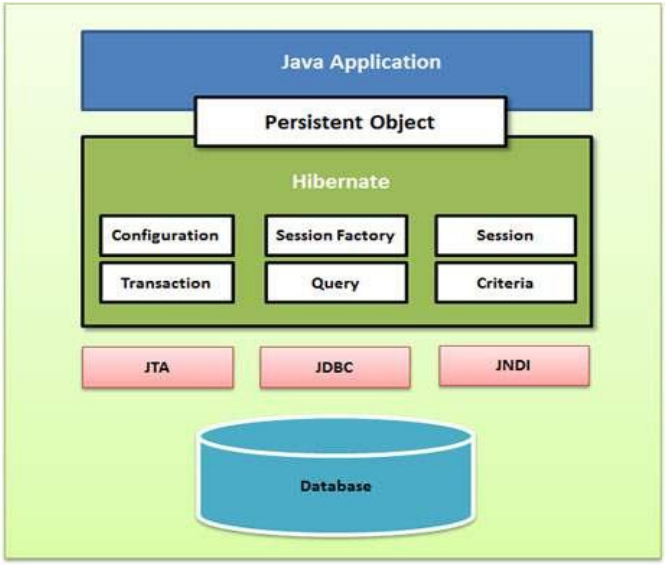

Figure -4: Hibernate Architecture (Core Classes View)

Above is a high level diagram to understand different components of hibernate.

- **Configuration** (org.hibernate.cfg.Configuration)

It allows the application on startup, to specify properties and mapping documents to be used when creating a Session Factory. Properties file contains database connection setup info while mapping specifies the classes to be mapped.

- **SessionFactory** (org.hibernate.SessionFactory)

It's a thread-safe immutable object created per database & mainly used for creating Sessions.It caches generated SQL statements and other mapping metadata that Hibernate uses at runtime.

- **Session** (org.hibernate.Session)

It's a single-threaded object used to perform create, read, update and delete operations for instances of mapped entity classes. Since it's not thread-safe, it should not be long-lived and each thread/transaction should obtain its own instance from a SessionFactory. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.

- **Transaction** (org.hibernate.Transaction)

It's a single-thread object used by the application to define units of work. A transaction is associated with a Session. Transactions abstract application code from underlying transaction implementations(JTA/JDBC), allowing the application to control transaction boundaries via a consistent API. It's an Optional API and application may choose not to use it.

- **Query** (org.hibernate.Query)

A single-thread object used to perform query on underlying database. A Session is a factory for Query. Both HQL(Hibernate Query Language) & SQL can be used with Query object.

Hibernate provides a lot of flexibility in use. It is called "Lite" architecture when we only uses the object relational mapping component. While in "Full Cream" architecture all the three component Object Relational mapping, Connection Management and Transaction Management) are used.

B. Hibernate Configuration

Hibernate allows many configuration options, based on an application's persistence requirements. However, most of these parameters have default values, which relieve us of detailed configuration in most situations. Hibernate allows us to choose either Hibernate-managed JDBC connections or a container-managed data source. If we decide to use Hibernate-managed connections, we need to tell Hibernate about the database properties, such as the name of the driver class, the database JDBC URL, and the database username and password. These are the basic configuration settings for Hibernate-managed connections. Each of these settings is represented by a name, as explained in the following table:

Table -2: Configuration Property Classes

Property name	Property value
hibernate.connection.driver_class	The fully qualified class name of the database's JDBC driver
hibernate.connection.url	The database's JDBC URL
hibernate.connection.username	The database username
hibernate.connection.password	The database password
hibernate.dialect	This property makes Hibernate generate the appropriate SQL for the chosen database.
hibernate.connection.pool_size	Limits the number of connections waiting in the Hibernate database connection pool.
hibernate.connection.autocommit	Allows autocommit mode to be used for the JDBC connection.

C. Hibernate Mappings

An entity/relational mappings are generally defined in an XML document. This mapping file instructs Hibernate how to map the defined class or classes to the database tables. Still many Hibernate users select to write the XML by hand, a number of tools live to create the mapping document. These contain Docket, Middlemen and Andromeda for advanced Hibernate users.[3]

There are three types of mappings in Hibernate:

- **Collections Mappings**

If an entity or class has collection of values for a particular variable, then we can map those values using any one of the collection interfaces available in java. Hibernate can persist instances of java.util.Map, java.util.Set, java.util.SortedMap, java.util.SortedSet, java.util.List, and any array of persistent entities or values.

- **Association Mappings**

The mapping of associations between entity classes and the relationships between tables is the soul of ORM. Following are the four ways in which the cardinality of the relationship between the objects can be expressed. An association mapping can be unidirectional as well as bidirectional.

- **Component Mappings**

It is very much possible that an Entity class can have a reference to another class as a member variable. If the referred class does not have its own life cycle and completely depends on the life cycle of the owning entity class, then the referred class hence therefore is called as the Component class. The mapping of Collection of Components is also possible in a similar way just as the mapping of regular Collections with minor configuration differences.

D. Hibernate - Sessions

A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object. The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed. The main function of the Session is to offer, create, read, and delete operations for instances of mapped entity classes. Instances may exist in one of the following three states at a given point in time –

Transient − A new instance of a persistent class, which is not associated with a Session and has no representation in the database and no identifier value is considered transient by Hibernate.
Persistent − We can make a transient instance persistent by associating it with a Session. A persistent instance has a representation in the database, an identifier value and is associated with a Session.
Detached − Once we close the Hibernate Session, the persistent instance will become a detached instance.

E. *Hibernate - Persistent Class*

The entire concept of Hibernate is to take the values from Java class attributes and persist them to a database table.

A mapping document helps Hibernate in determining how to pull the values from the classes and map them with table and associated fields. Java classes whose objects or instances will be stored in database tables are called persistent classes in Hibernate. Hibernate works best if these classes follow some simple rules, also known as the Plain Old Java Object (POJO) programming model.[5]

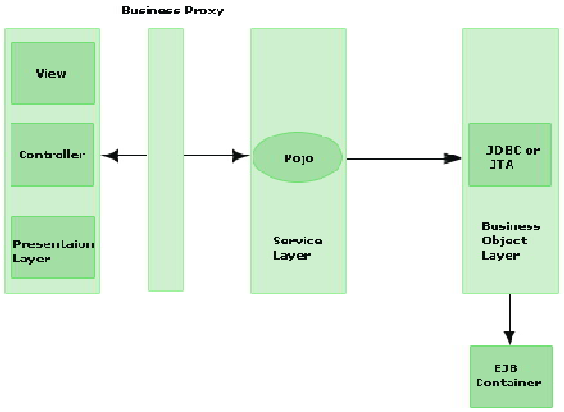

Figure -5: POJO Programming Model

IV. HIBERNATE QUERY LANGUAGE (HQL)

Hibernate Query Language (HQL) is an object-oriented query language, similar to SQL, but instead of operating on tables and columns, HQL works with persistent objects and their properties. HQL queries are translated by Hibernate into conventional SQL queries, which in turns perform action on database. Although we can use SQL statements directly with Hibernate using Native SQL, but I would recommend to use HQL whenever possible to avoid database portability hassles, and to take advantage of Hibernate's SQL generation and caching strategies. Keywords like SELECT, FROM, and WHERE, etc., are not case sensitive, but properties like table and column names are case sensitive in HQL. Like in form clause of HQL query we need to following this way

```
String hql = "FROM Employee";
Query query = session.createQuery(hql);
List results = query.list();
```

V. CONCLUSION

Finally, on discussing Hibernate ORM tool for java is high-performance Object/Relational persistence and query service, which is licensed under the open source GNU Lesser General Public License (LGPL) and is free to download. Hibernate not only takes care of the mapping from Java classes to database tables (and from Java data types to SQL data types), but also provides data query and retrieval facilities.

REFERENCES

[1] Just Hibernate: A Lightweight Introduction to the Hibernate Framework , Madhusudhan Konda, O'reilly
[2] Beginning Hibernate , Jeff Linwood, Dave Minte , Second Edition, Apress
[3] Hibernate: A Developer's Notebook, James Elliott , First Edition , O'reilly
[4] Java Persistance With Hibernate, Christian Bauer, Gavin King, Edition 2009 , DreamTech
[5] Professional Hibernate , Eric Pugh, Joseph D. Gradecki, Third Edition, Wiley