Introduction to Hibernate

Dr. Harshad Prajapati
17 Sep 2023

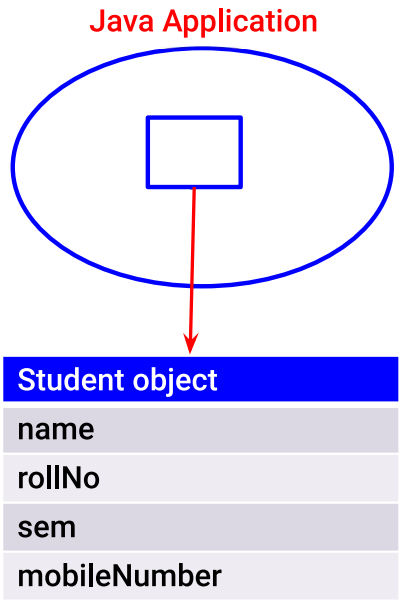
Prerequisites to Learn Hibernate

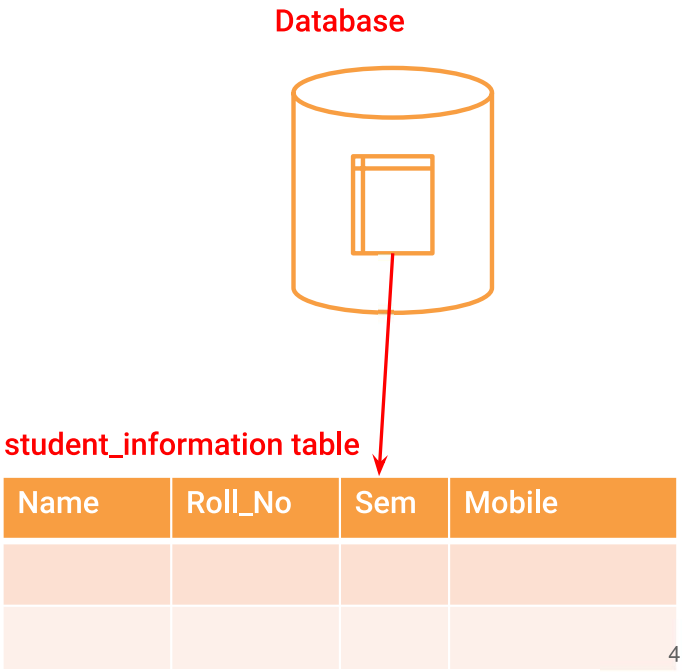
- Relational Databases, SQL
- Java Language
- Java Annotations

Overview of SQL Concepts

- Data Definition Language (DDL) is used for creating schema for artifacts (tables and constraints). If business requirements change, schema would change.
  - Creating
  - Altering
  - Dropping
- Data Manipulation Language (DML) is used to perform operations on data:
  - Insert
  - Update
  - Delete
- SQL is used to join, aggregate, and group data.

Data in Java Application versus in Database





Student s=new Student("Tom Cruise", 1, 6, 111111111);

Relational World and Object World

- Relational world uses
  - Foreign key constraint for creating a relationship between two tables.
  - For many-to-many relationships, a database needs a link table.
- Object world uses
  - Object references for creating a relationship between two entities.
  - Object references are directional.
  - To define bidirectional relationships, we need to define object references in both the associated entities.

Why Object Relational Mapping (ORM)?

- Object oriented software or application development uses
  - Classes and objects.
- Data store, such as DBMS, uses
  - Table and rows of data.
- Thus, there is a mismatch between two different worlds:
  - Object model (Interconnected graph of objects)
  - Relation model (Connected Tables)
- ORM is a connection between two different worlds, object and relational.
  - ORM is a programming technique for converting data between RDBMS and OOPL.

Mismatch Problems

Mismatch	Meaning
Granularity	Two or more objects may represent a single database table
Inheritance	RDBMS systems do not have anything like Inheritance available in OO programming languages
Identity	RDBMS defines exactly one meaning of 'sameness' : primary key. Java has object identity (ob1==ob2) and object equality (ob1.equals(ob2))
Associations	OOPL represents association using object references. RDBMS represents it using foreign key column.
Navigation	Ways of navigation are different in RDBMS and Java

Advantages of ORM

- Business logic code accesses data as object not as DB tables.
  - Software entities are based on business concepts rather than DB tables.
- Hides details of SQL queries.
- Internally uses JDBC.
- Business logic code is independent of database implementation.
- Automatic key generation and transaction management.
- Fast development and easy maintenance of applications.

What does an ORM provide?

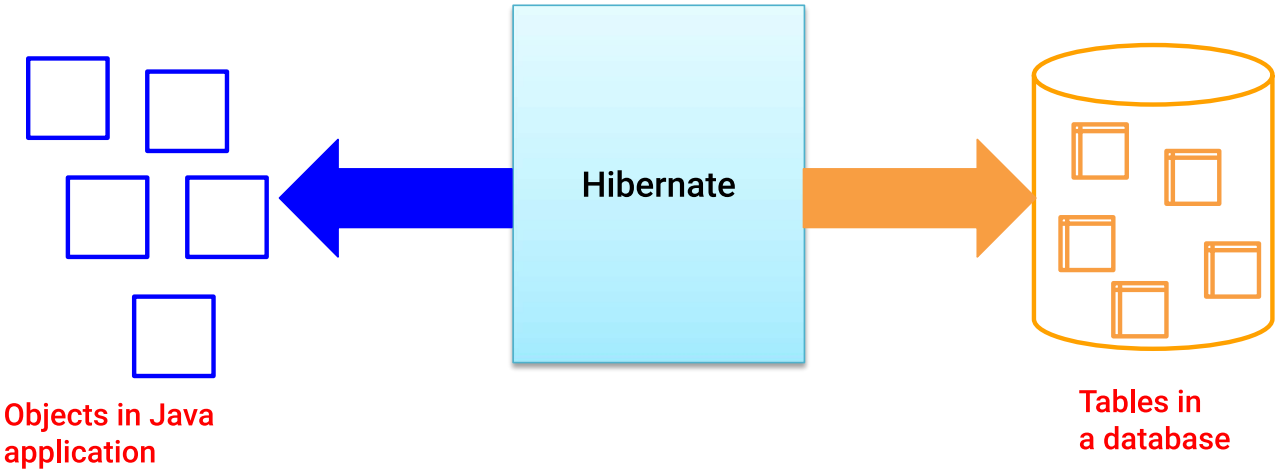
- API to perform CRUD operations.
- Configuration for specifying mapping between object and table.
- DBMS independent query language or API (e.g., HQL).
- ORM is the automated and transparent persistence of objects in a Java application to the tables in an SQL database.
- ORM works by
  - Transforming data from one representation to another representation.

ORM Frameworks in Java

- Enterprise Java Beans (EJB) Entity Beans
- JDO-Java Data Objects
- Castor
- TopLink
- Spring DAO
- Hibernate
- Apache iBATIS, and many other

What is Hibernate?

- Hibernate is an ORM tool that provides Object Relational Mapping.
- It is open source persistent framework created by Gavin King in 2001.



Using SQL in Java with JDBC

- Java application can issue SQL statements to the database using JDBC (Java Database Connectivity).
  - JDBC works directly with the rows and columns using java.sql.ResultSet.

Java Application can use JDBC (Low Level API)

Student s=new Student("Tom Cruise", 1, 6, 111111111);

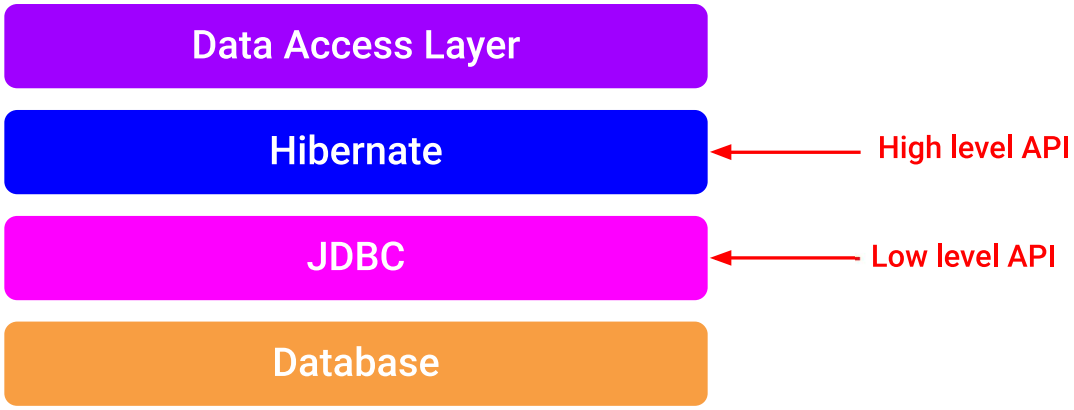
```
public void insert(Student st){
    Connection con=null; Statement stmt=null;
    // Register JDBC driver
    Class.forName(…);
    // Open a DB connection
    con=DriverManager.getConnection(…);
    // Create a statement
    stmt=con.createStatement();
    String query="INSERT INTO STUDENT_INFORMATION(Name, Roll_No, Sem, Mobile) " +
    "VALUES("+st.getName()+","+st.getRollNo+","+ st.getSem()+","+st.getMobileNumber()+")";
    //Execute update query
    stmt.executeUpdate(query);
}
```

Hibernate provides High Level API, hides/abstracts many details.

How is Hibernate Useful?

- Developers need to write minimal code to deal with data.
  - Hibernate relives the developer from writing more than 95% of common data persistent related code.
- In Hibernate, database connection is configurable.
  - Possible to change database in configuration file.
- Hibernate takes care of
  - Making database connection,
  - Releasing db connection,
  - Writing queries.

Position of Hibernate Layer



Using SQL in Java with Hibernate

- Hibernate allows to interact with the application-specific object-oriented domain model.
  - Instead of reading ResultSet for database tables,
  - Our application reads objects of classes.
- What is domain model?
  - Object-oriented representation of our application's data model.

How Application Code is Written using Hibernate

Student s=new Student("Tom Cruise", 1, 6, 111111111);

```
public void insert(Student st){
      Session session=factory.openSession();
      Transaction tx=null;
      try{
        tx=session.beginTransaction();
        session.save(st);
        tx.commit();
      }
      catch(HibernateException ex){
        …
      }
      finally{
        session.close();
      }
}
```


Data Retrieval using JDBC versus Hibernate

```
//JDBC code
ResultSet rs;
rs=stmt.executeQuery("SELECT * FROM
      STUDENT_INFORMATION");
List stList=new LinkedList();
while(rs.next()){
      String name=rs.getString("Name");
      int rollNo=rs.getInt("Roll_No");
      int sem=rs.getInt("Sem");
      String mobile=rs.getString("Mobile");

      Student st=new Student(name, rollNo, sem,
      mobile);
      stList.add(st);
}
```

```
//Hibernate code
List stList= session.createQuery("FROM
      STUDENT_INFORMATION").list();
```

Other features of Hibernate

- Good caching mechanism for faster retrieval of data.
  - Avoid database query if object is in cache.
- Opening and closing db connection handled by Hibernate.
- Application can support almost all relational databases.
- A little modification is needed in java application for any change in database.
  - If we change the name of a column in a database table:
    - Using direct JDBC code, need to reflect at all places.
    - Using Hibernate, just change in the configuration file or in annotation.

Supported Databases

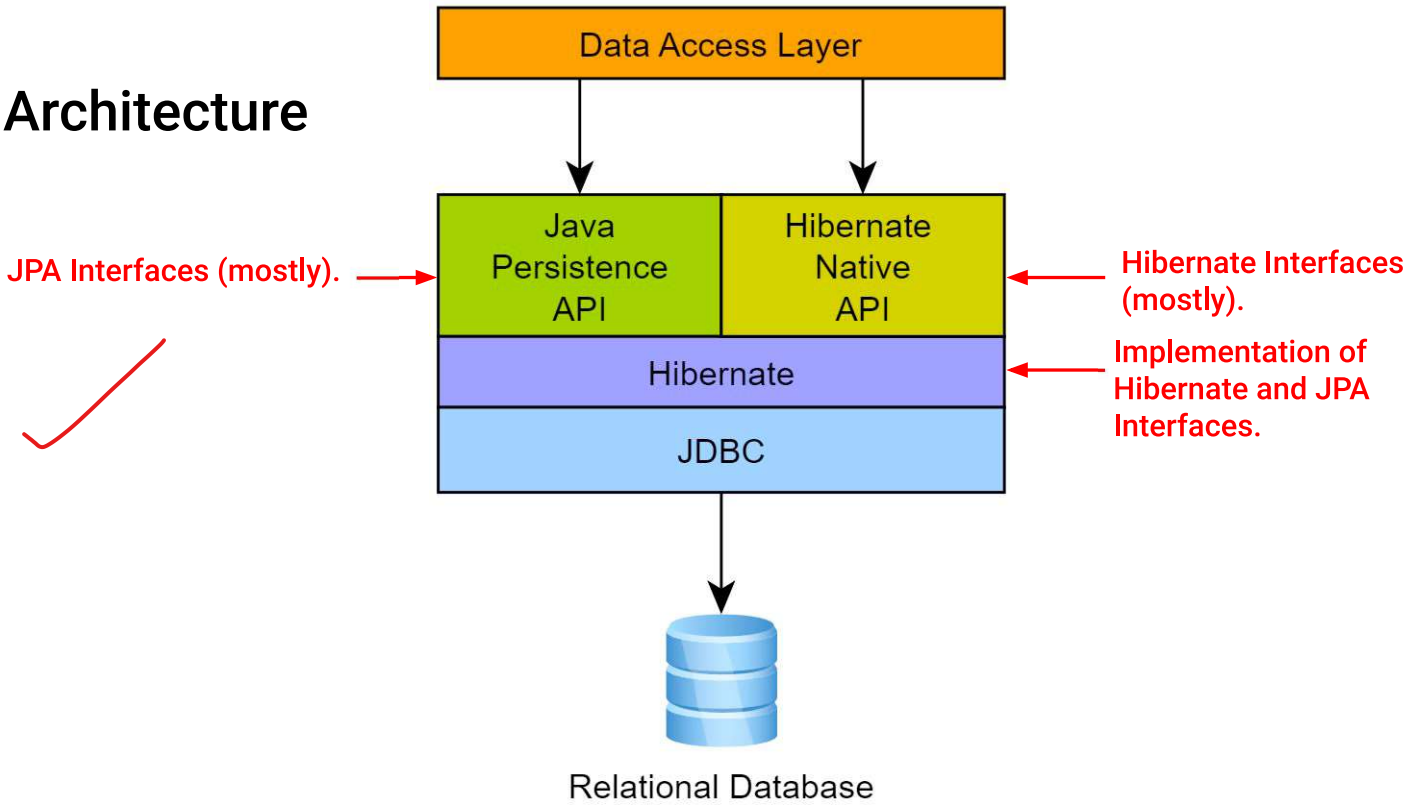
- HSQL Database Engine
- DB2
- MySQL
- PostgreSQL
- FrontBase
- Oracle
- Microsoft SQL Server Database
- Sybase SQL Server
- Informix Dynamic Server

Benefits of Hibernate

- Productivity:
  - Reduces development time.
- Maintainability:
  - Makes system understandable and easier to refactor.
- Performance:
  - Provides efficient and tunable caching in the application tier.
- Vendor Independence:
  - We can change DBMS.

Hibernate Architecture

Architecture



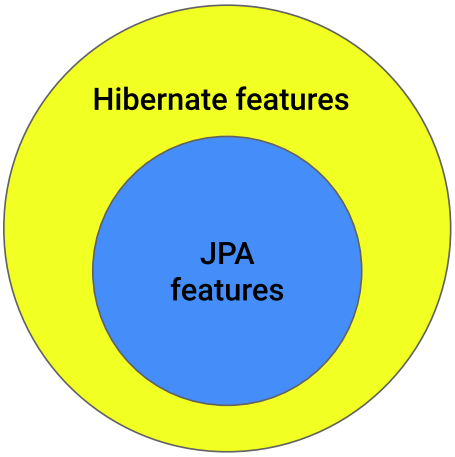
Source: https://docs.jboss.org/hibernate/orm/6.2/userguide/html_single/Hibernate_User_Guide.html

What is Persistence?

- Object **persistence** means
  - Objects can **outlive** the application process.
  - Objects can be **saved** to a **data store** and can be **re-created** at a **later** point in time.

Persistence APIs available in Hibernate

- Hibernate native persistence API:
  - Hibernate's native features are a superset of the JPA persistence features.



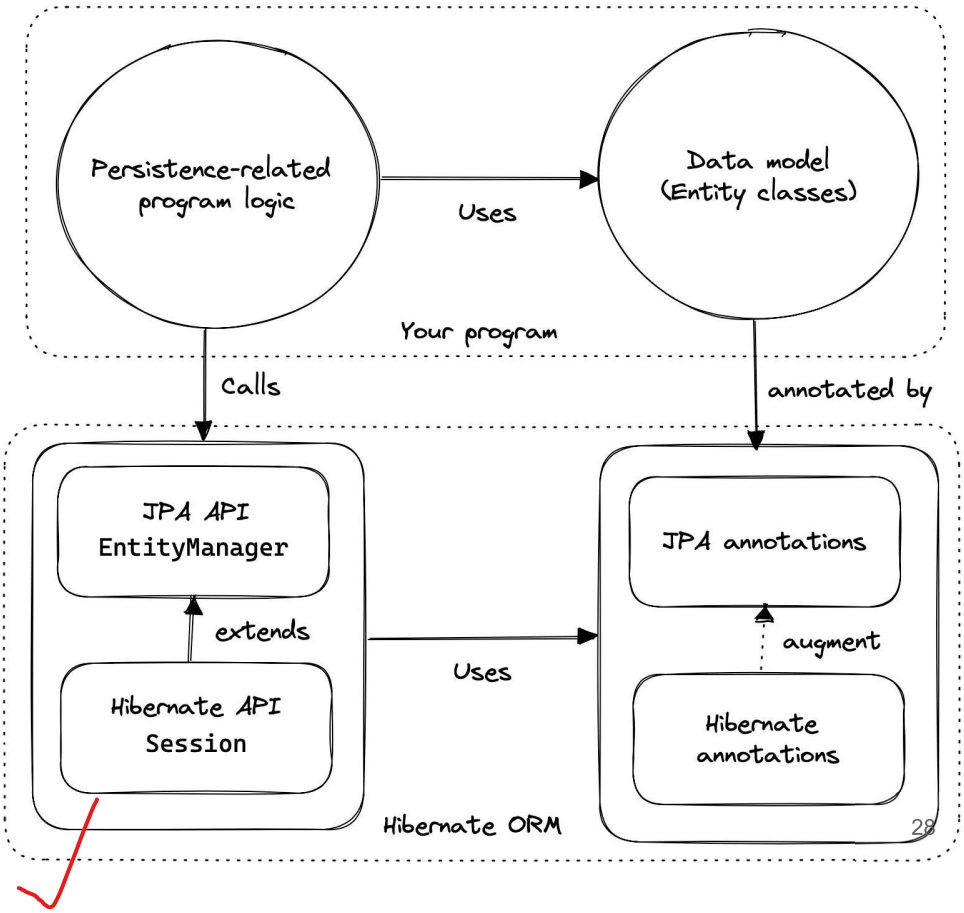
JPA Specifications

- JPA specifications define the following:
  - A facility for specifying mapping metadata.
    - Using annotations.
    - Using XML files.
  - APIs for performing basic CRUD operations on instances of persistent classes using jakarta.persistence.EntityManager.
  - Java Persistence Query Language (JPQL) for specifying queries that refer to entity classes and properties of entity classes. Criteria queries.
  - JPA specification covers some basic caching strategies.

Hibernate is a JPA Implementation

- Hibernate is a provider of Java Persistence API (which is a specification).
  - Hibernate is one of implementations of JPA.
- Important information:
  - Hibernate was the inspiration behind the Java (now Jakarta) Persistence API (JPA).
  - Java Platform EE was too complex.

Hibernate API



Source: https://docs.jboss.org/hibernate/orm/6.3/introduction/html_single/Hibernate_Introduction.html

Hibernate API

- API of Hibernate in terms of three basic elements:
  - An implementation of the JPA-defined APIs (important interfaces EntityManagerFactory and EntityManager) and JPA-defined ORM annotations.
  - A native API exposing the full set of available functionality (around SessionFactory and Session)
    - SessionFactory extends EntityManagerFactory and Session extends EntityManager.
  - A set of mapping annotations which augment the ORM annotations defined by JPA.

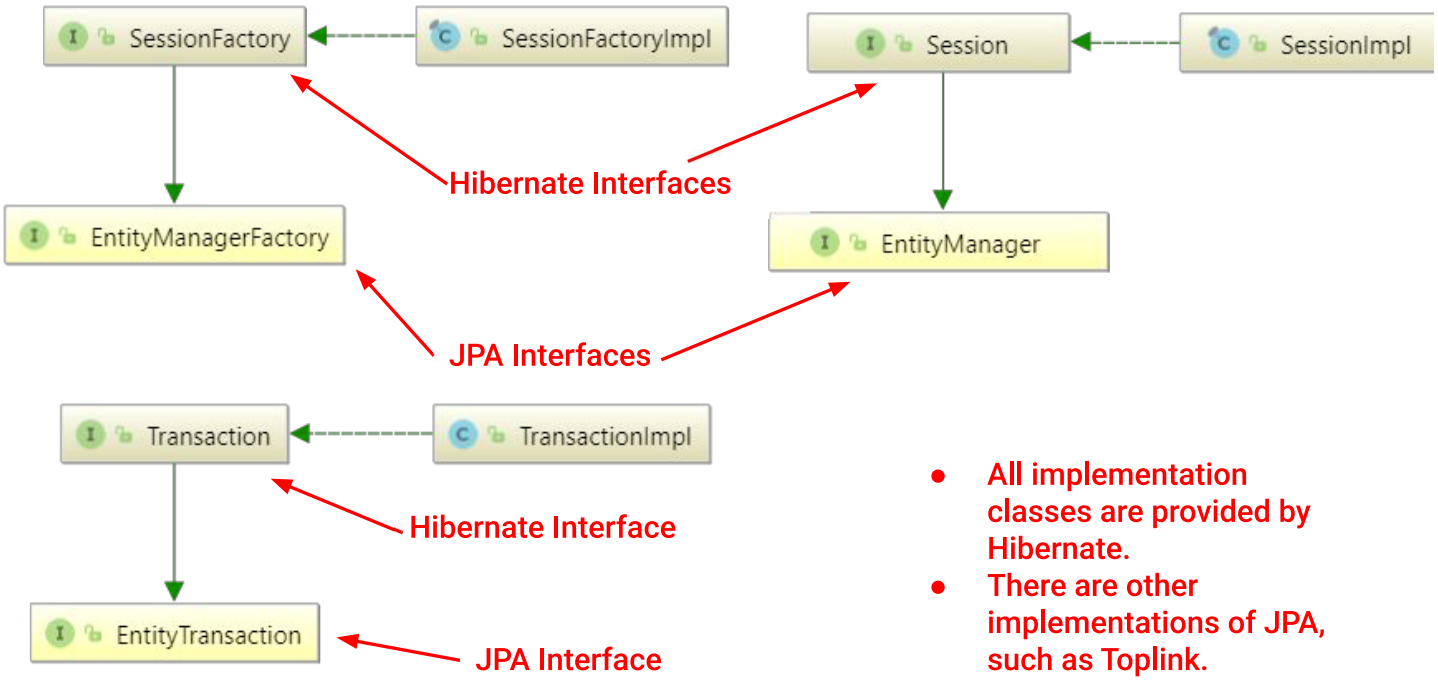
Source: https://docs.jboss.org/hibernate/orm/6.3/introduction/html_single/Hibernate_Introduction.html

How Hibernate API Impacts to Developers?

- As an application developer, we need to decide:
  - Write our programs in terms of Session and SessionFactory OR
  - Write in terms of EntityManager and EntityManagerFactory
    - Allows maximum portability,
    - We can go from one JPA implementation to another easily.

Source: https://docs.jboss.org/hibernate/orm/6.3/introduction/html_single/Hibernate_Introduction.html

Hibernate and JPA Interfaces and Hibernate Implementation



- All implementation classes are provided by Hibernate.
- There are other implementations of JPA, such as Toplink.

Hibernate Interfaces

- SessionFactory (org.hibernate.SessionFactory)
- Session (org.hibernate.Session)
- Transaction (org.hibernate.Transaction)

Hibernate Objects

- Major objects
  - Configuration
  - Session Factory
  - Session
  - Transaction
  - Query
  - Criteria

Configuration object

- Configuration object is created during application initialization.
- It represents configuration or property file needed by the Hibernate.
- Configuration object provides two things:
  - Database connection:
    - It is handled using one or more configuration files.
      - For example, hibernate.properties and hibernate.cfg.xml.
  - Class mapping:
    - Establishes connection between Java classes and database tables.

SessionFactory object

- SessionFactory object is created using Configuration object.
  - SessionFactory is created during application startup and is maintained for later use.
  - SessionFactory is used to create a Session.
- SessionFactory is thread safe object.
  - Multiple threads can use it.
- SessionFactory is heavyweight object.
- We need one SessionFactory object per DBMS.

Session Object

- A Session object provides a physical connection with the database.
  - It is instantiated each time an application needs interaction with the database.
- Session object is lightweight.
- Persistent objects are stored and retrieved through a Session object.
- Session objects are not thread safe.
  - Therefore, we should release them once use is over.

Transaction Object

- It represents a unit of work (atomic operations) to be performed with the database.
- Transaction concept is provided by RDBMS.
- Hibernate handles transactions using underlying transaction manager and transaction.

Query Object

- Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects.
- A Query object is used
  - to bind query parameters,
  - to limit the number of results returned by the query,
  - to execute the query.

Criteria Object

- Criteria API is a programmatic and type-safe way to build queries for database operations.
- Central element of Criteria API is Criteria object.
- Criteria object is used to create database queries in object oriented manner compared to writing raw SQL queries.

Application Development Steps

Adding Dependencies

Using Hibernate in our Application

- Gradle:

```
dependencies {
  implementation "org.hibernate.orm:hibernate-core:6.3.0.Final"
}
```

- Maven:

```
<dependency>
   <groupId>org.hibernate.orm</groupId>
   <artifactId>hibernate-core</artifactId>
   <version>6.3.0.Final</version>
</dependency>
```

Source: https://docs.jboss.org/hibernate/orm/6.3/quickstart/html_single/

Using Hibernate in our Application via BOM

- Hibernate also provides a platform (BOM in Maven terminology) module.
  - BOM stands for Bill Of Materials.
  - BOM is a special kind of POM (Project Object Model) file used to manage dependencies in a consistent and centralized manner.
- The Hibernate Platform can be used to align versions of the Hibernate modules along with the versions of its libraries.
- The platform artifact is named hibernate-platform.

Source: https://docs.jboss.org/hibernate/orm/6.3/quickstart/html_single/

Using Hibernate in our Application via BOM

- Gradle

```
dependencies {
  implementation platform "org.hibernate.orm:hibernate-platform:6.3.0.Final"
  // use the versions from the platform
  implementation "org.hibernate.orm:hibernate-core"
  implementation "jakarta.transaction:jakarta.transaction-api"
}
```

Source: https://docs.jboss.org/hibernate/orm/6.3/quickstart/html_single/

Using Hibernate in our Application via BOM

- Maven

```
<dependency>
   <groupId>org.hibernate.orm</groupId>
   <artifactId>hibernate-core</artifactId>
</dependency>
<dependency>
   <groupId>jakarta.transaction</groupId>

<artifactId>jakarta.transaction-api</artifactId>
</dependency>
```

```
<dependencyManagement>
 <dependency>
   <groupId>org.hibernate.orm</groupId>
   <artifactId>hibernate-platform</artifactId>
   <version>6.3.0.Final</version>
   <type>pom</type>
   <scope>import</scope>
 </dependency>
</dependencyManagement>
```

Source: https://docs.jboss.org/hibernate/orm/6.3/quickstart/html_single/

Widely used Libraries

- Hibernate ORM modules:
  - hibernate-core: The core object/relational mapping engine.
  - hibernate-envers: Entity versioning and auditing.
- Integration-oriented modules:
  - There are many connection pooling library
    - hibernate-hikaricp: Support for HikariCP connection pooling.
- Testing-oriented modules:
  - hibernate-testing: A series of JUnit extensions for testing Hibernate ORM functionality.

Source: https://docs.jboss.org/hibernate/orm/6.3/quickstart/html_single/

Dependency for the JDBC driver for Database

- PostgreSQL or CockroachDB org.postgresql:postgresql:{version}
- MySQL or TiDB com.mysql:mysql-connector-j:{version}
- MariaDB org.mariadb.jdbc:mariadb-java-client:{version}
- DB2 com.ibm.db2:jcc:{version}
- SQL Server com.microsoft.sqlserver:mssql-jdbc:${version}
- Oracle com.oracle.database.jdbc:ojdbc11:${version}
- H2 com.h2database:h2:{version}
- HSQLDB org.hsqldb:hsqldb:{version}

Hibernate Configuration

Hibernate Configuration

- Hibernate requires the following configuration:
  - DBMS related configuration parameters.
  - Mapping of Java Classes with Tables.
  - Other parameters (e.g., hibernate supported features)
- Configuration information can be kept in hibernate.cfg.xml file.
- This file is stored in the root directory of our application's classpath.

Hibernate Configuration: Properties

- hibernate.dialect:
  - This property enables Hibernate to generate the appropriate SQL for the chosen database. Different DBMSes can have different SQL code.
- hibernate.connection.driver_class:
  - It indicates the JDBC driver class to use for JDBC connectivity.
- hibernate.connection.url:
  - The JDBC URL to the database instance.
- hibernate.connection.username:
  - Username for the database.
- hibernate.connection.password:
  - Password for the database.

Hibernate Configuration: Properties

- hibernate.connection.pool_size:
  - Upper limit on the number of connections waiting in the Hibernate database connection pool.
- hibernate.connection.autocommit:
  - It allows auto-commit mode to be used for the JDBC connection.
- hibernate.connection.datasource:
  - The JNDI name defined in the application server context.

Hibernate Configuration: Properties

- hibernate.jndi.class:
  - The InitialContext class for JNDI.
- hibernate.jndi.<JNDIpropertyname>:
  - Passes any JNDI related property we want to pass to the JNDI InitialContext.
- hibernate.jndi.url:
  - It indicates the URL for JNDI.

JPA Configuration Properties

- jakarta.persistence.jdbc.url:
  - JDBC URL of your database
- jakarta.persistence.jdbc.user:
  - Database username
- jakarta.persistence.jdbc.password:
  - Database password

Databases Dialect Values for Various Databases

DBMS	Property value
DB2	org.hibernate.dialect.DB2Dialect
Microsoft SQL Server 2008	org.hibernate.dialect.SQLServer2008Dialect
MySQL	org.hibernate.dialect.MySQLDialect
Oracle (any version)	org.hibernate.dialect.OracleDialect
Oracle 11g	org.hibernate.dialect.Oracle10gDialect
PostgreSQL	org.hibernate.dialect.PostgreSQLDialect

- In Hibernate 6, we don't need to specify hibernate.dialect. The correct Hibernate SQL Dialect will be determined for us automatically.
- Similarly, neither hibernate.connection.driver_class nor jakarta.persistence.jdbc.driver is needed when working with one of the supported databases.

Sample JPA Configuration (META-INF/persistence.xml)

```
<persistence xmlns="https://jakarta.ee/xml/ns/persistence"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd"
          version="3.0">
   <persistence-unit name="example">
   <class>org.hibernate.example.Book</class>
    <properties>
        <!-- H2 in-memory database -->
        <property name="jakarta.persistence.jdbc.url" value="jdbc:h2:mem:db1"/>
```


Sample JPA Configuration (META-INF/persistence.xml)

```
        <!-- Credentials -->
        <property name="jakarta.persistence.jdbc.user" value="sa"/>
        <property name="jakarta.persistence.jdbc.password" value=""/>

        <!-- Agroal connection pool -->
        <property name="hibernate.agroal.maxSize" value="20"/>

        <!-- display SQL in console -->
        <property name="hibernate.show_sql" value="true"/>
        <property name="hibernate.format_sql" value="true"/>
        <property name="hibernate.highlight_sql" value="true"/>
      </properties>
   </persistence-unit>
</persistence>
```

Sample Hibernate Configuration File (hibernate.cfg.xml) in root classpath

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
 <session-factory>
    <property name="connection.url">jdbc:mysql://localhost/test</property>
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect </property>
    <property name="connection.username">username</property>
    <property name="connection.password">password</property>
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>

    <!-- DB schema will be updated if needed -->
    <!-- <property name="hibernate.hbm2ddl.auto">update</property> -->
    <mapping class="org.example.entities.Student"/>
 </session-factory>
</hibernate-configuration>
```


Sample Configuration (hibernate.properties in root classpath)

Using hibernate properties

```
# Database connection settings
hibernate.connection.url=jdbc:h2:mem:db1;DB_CLOSE_DELAY=-1
hibernate.connection.username=sa
hibernate.connection.password=

# Echo all executed SQL to console
hibernate.show_sql=true
hibernate.format_sql=true
hibernate.highlight_sql=true

# Automatically export the schema
hibernate.hbm2ddl.auto=create
```

Sample Configuration (hibernate.properties in root classpath)

```
# PostgreSQL
jakarta.persistence.jdbc.url=jdbc:postgresql://localhost/example

# Credentials
jakarta.persistence.jdbc.user=hibernate
jakarta.persistence.jdbc.password=zAh7mY$2MNshzAQ5

# SQL statement logging
hibernate.show_sql=true
hibernate.format_sql=true
hibernate.highlight_sql=true
```

Using JPA properties

Persistence Unit in JPA (Standardized by JPA)

- Persistence unit is a starting point in JPA.
- Persistence unit is
  - A pairing of Domain model class mappings with Database connection.
  - Some other configuration settings.
- Every application has at least one persistence unit.
- The standard configuration file for persistence units is located in the classpath in META-INF/persistence.xml.

Mapping Data Types

- The types specified in mapping files are neither Java data types nor SQL database types.
- The specified types in mapping files are called Hibernate mapping types.

Hibernate mapping type	Java data type	SQL data type
integer	int or java.lang.Integer	INTEGER
long	long or java.lang.Long	BIGINT
short	short or java.lang.Short	SMALLINT
float	float or java.lang.Float	FLOAT
double	double or java.lang.Double	DOUBLE
big_decimal	java.math.BigDecimal	NUMERIC
character	java.lang.String	CHAR(1)
string	java.lang.String	VARCHAR
byte	byte or java.lang.Byte	TINYINT
boolean	boolean or java.lang.Boolean	BIT

Hibernate mapping type	Java data type	SQL data type
date	java.util.Date or java.sql.Date	DATE
time	java.util.Date or java.sql.Time	TIME
timestamp	java.util.Date or java.sql.TimeStamp	TIMESTAMP
calendar	java.util.Calendar	TIMESTAMP
calendar_date	java.util.Calendar	DATE

References

- Java Persistence with Hibernate, Second Edition, 2016, Christian Baver, Gavin King, Gary Gregory, Publisher: Manning.
- Official Documentation https://docs.jboss.org/hibernate/orm/6.3/introduction/html_single/Hibernate_Introduction.html