JSPIDERS BASAVANAGUDI

HIBERNATE

ADV JAVA

By JSPIDERS FACULTY TEAM

BTR

HIBERNATE SYLLABUS

- Introduction
- ORM (Object Relational Mapping)
- Hibernate
- Hibernate Architecture
- Building Simple Application using Hibernate with Maven
- Hibernate Annotations
- Defining Entities
- CRUD operations using Session methods
- Mapping relations
- Lazy and Eager loading
- HQL
- Pagination and Caching

PREREQUISITES

- CORE JAVA
- JDBC
- SQL

JSPIDERS BASAVANAGUDI

ORM

ORM

- Object-Relational Mapping (**ORM**) is the process of converting Java objects to database tables.
- **ORM is a concept and ORM itself is not a TOOL**
- **Popular ORM tools :** Hibernate , Ibatis , TopLink etc..
- By using ORM we can interact with R-DBMS without SQL
- **Entity classes** that we create are going to represent the **TABLE**
- **Object** of Entity class will represent the **ROW** or **RECORD** of the **TABLE**
- Java by default comes with an API called **Java Persistence API (JPA)** which is the implementation of the ORM concept
- **JPA** allows programmers to do DB operations like CRUD, JOINS, Manage Primary Key and Foreign Key easily

JSPIDERS BASAVANAGUDI

HIBERNATE

HIBERNATE

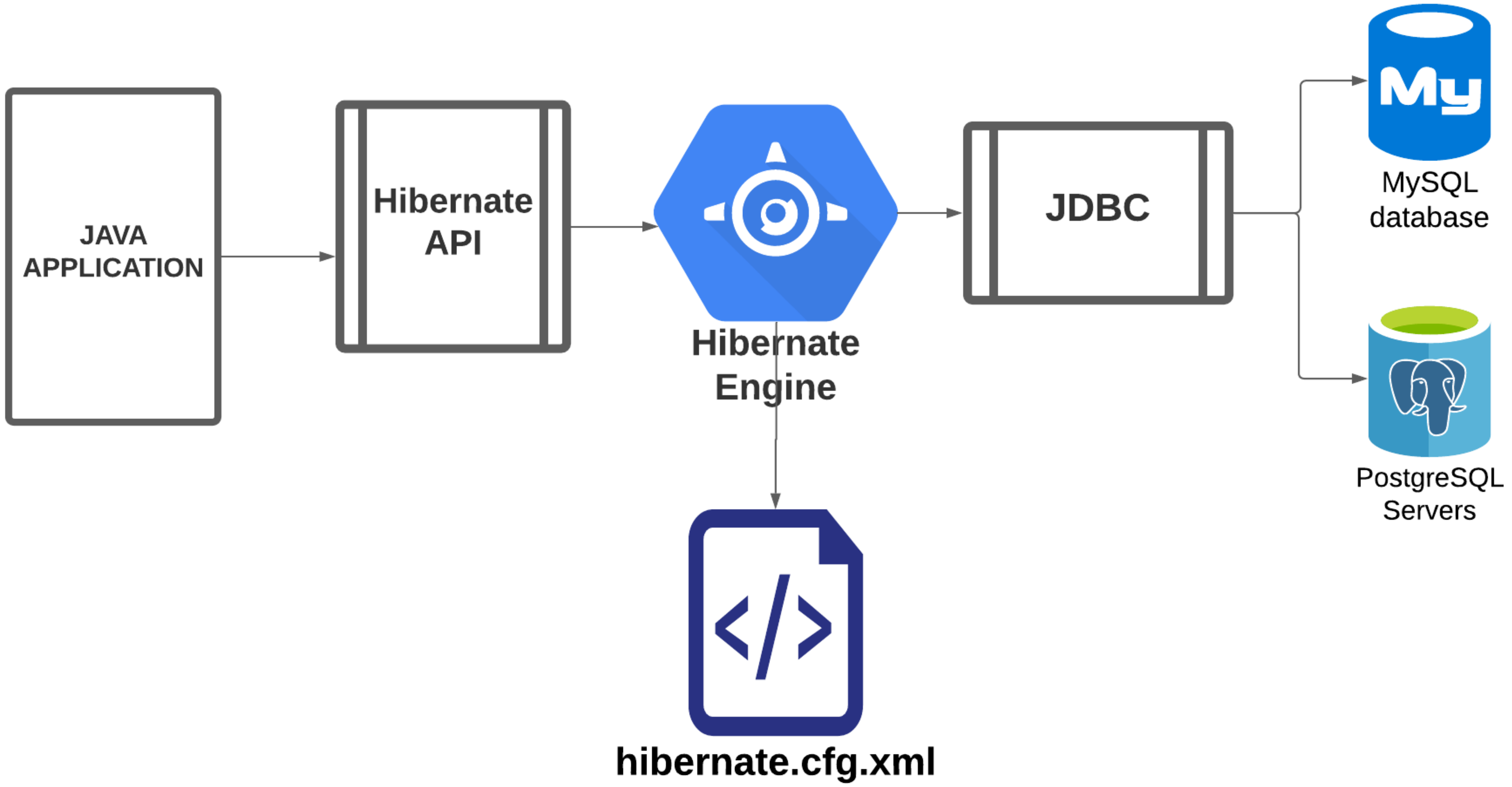
- Hibernate is an object-relational mapping (ORM) tool for the Java language, which provides a framework for mapping java object to a relational database table.
- Hibernate solves object-relational impedance mismatch problems by replacing direct persistence-related database accesses with high-level object handling functions
- Hibernate's primary feature is mapping from Entity classes to database tables (and from Java data types to SQL data types).
- Hibernate also provides data query and retrieval facilities which is called as **HQL**
- Hibernate generates the SQL calls and relieve the developer from manual ResultSet handling and object conversion.

JSPIDERS BASAVANAGUDI

HIBERNATE ARCHITECTURE

By JSPIDERS FACULTY TEAM

HIBERNATE ARCHITECTURE



JSPIDERS BASAVANAGUDI

CREATING HIBERNATE PROJECT WITH MAVEN

By JSPIDERS FACULTY TEAM

INSTALLING JBOSS-PLUGIN AND CREATING HIBERNATE

- Create a maven project in eclipse
- Add mysql and hibernate dependencies to pom.xml
- Install hibernate plugin to eclipse in order create hibernate configuration file
- Enter the details of Database to create hibernate configuration file contains all the information such as DBURL,Username,Password,dialect etc.. which will be used by hibernate tool to connect to DataBase.

Understanding hibernate.cfg.xml

- **<hibernate.connection.driver_class> :** represents the JDBC driver class.
- **<hibernate.connection.url> :** represents the JDBC URL.
- **<hibernate.connection.username> :** represents the database username.
- **<hibernate.connection.password> :** represents the database password.
- **<hibernate.dialect>:** represents the type of database used in hibernate to generate SQL statements for a particular relational database.
- **<hibernate.connection.pool_size> :** represents the maximum number of connections available in the connection pool.
- **<hibernate.show_sql> :** It is used to display the executed SQL statements to console.
- **<hibernate.transaction.auto_close_session> :** If it is enabled, the session will be automatically closed during the after completion phase of the transaction.

Understanding <hibernate.hbm2ddl.auto>

- **create :** it will first drop all the tables with respect to to the entities specified in <mapping> and **re-creates** the tables with no data every-time we run the JAVA CODE
- **update :** it will update the data to the existing tables and won't drop the tables every-time we run the JAVA CODE

JSPIDERS BASAVANAGUDI

HIBERNATE ANNOTATIONS

Annotation	Description	Properties
@Entity	this annotation is used to mark the class as an entity and to create the table in Database.	**name** : used specify the name for the table. If you don't specify name, then hibernate will use the class name as the table name by default
@Table	Specifies the primary table for the annotated entity properties	**name :** specifies name for the table **schema :** specifies to which schema table belongs
@Id	this annotation marks the PK for this entity	
@Column	annotation specifies the details of the column for this property or field. If @Column annotation is not specified, property name will be used as the column name by default.	**name :** specifies name for the table **schema :** specifies to which schema table belongs

Annotation	Description	Properties
@Column	annotation specifies the details of the column for this property or field. If @Column annotation is not specified, property name will be used as the column name by default.	**name :** The name of the column. Defaults to the Data member name. **insertable :** Whether the column is included in SQL INSERT statements **length :** The column length. (Applies only if a string-valued column is used.) **nullabel :** If the column allows null values **unique :**If the column is a unique key. **precision :** The precision for a decimal(exact numeric) column. (Applies only if a decimal column is used **updateable :** Whether the column is included in SQL UPDATE statements generated by the persistence provider.
@GeneratedValue	annotation is to configure the way of increment of the specified column(field). For example when using Mysql, you may specify auto_increment in the definition of table to make it self-incremental.	**strattergy** = The primary key generation strategy AUTO : for auto incremenr

JSPIDERS BASAVANAGUDI

HIBERNATE SESSIONFACTORY AND SESSION

SessionFactory

- Hibernate SessionFactory is the factory class through which we get sessions and perform database operations.
- Hibernate SessionFactory provides methods through which we can get Session object – openSession() , getCurrentSession(
- **openSession()** : always opens a new session. We should close this session object once we are done with all the database operations.
- **close()** : closes the specified session factory.

Session

- A Session is used to get a physical connection with a database to perform any operations on the database.
- objects are saved and retrieved through a Session object.

Session

- A Session is used to get a physical connection with a database to perform any operations on the database.
- objects are saved and retrieved through a Session object.
- **beginTransaction() :** Begin a unit of work and return the associated Transaction object.
- **save(Object object) :** insert the the given object data to the row of the table, first assigning a generated identifier(ID).
- **get(entityName, id) :** Return the object of the given named entity with the given identifier(ID), or null if there is no such persistent instance.
- **update(Object object) :** Update given object data with the identifier(ID) to the table.
- **delete(entityName,id) :** Remove a persistent instance from the table.
- **close() :** End the session by releasing the JDBC connection and cleaning up.

Transaction

- A unit of work is called as transaction. In a transaction there may be multiple steps involved in such case, if one step fails, the whole transaction fails. objects are saved and retrieved through a Session object.
- In hibernate framework, we have Transaction interface that defines the unit of work
- A transaction is associated with Session and instantiated by calling **session.beginTransaction().**
- **begin() :** starts a new transaction.
- **commit() :** ends the unit of work

JSPIDERS BASAVANAGUDI

MAPPING RELATIONS

HIBERNATE MAPPINGS

- Entities can contain references to other entities.
- These associations are represented using foreign key relationships in the underlying tables.
- These foreign keys will depend on the primary ids used by participating tables.
- When only one out of the two entities contains a reference to the other, the association is uni-directional.
- If the association is mutual and both entities refer to one another, it is bi-directional.

TYPES OF MAPPINGS

Annotation	Mapping	Usage
@OneToOne	**OneToOne**	Either end can be made the owner, but and only one)of them should be made a owner
@OneToMany	**OneToMany**	The many end must be made the owner of the association.
@ManyToOne	**ManyToOne**	This is the same as the one-to-many relationship viewed from the opposite perspective, so the same rule applies: the many end must be made the owner of the association.
@ManyToMany	**ManyToMany**	Either end of the association can be made the owner.

JSPIDERS BASAVANAGUDI

LAZY AND EAGER LOADING

LAZY LOADING

- **Lazy Loading :** A design pattern that we use to delay initialization of an object as long as it's possible.

- **Advantages:**
- Much smaller initial load time than in the other approach
- Less memory consumption than in the other approach

- **Disadvantages:**
- Delayed initialization might impact performance during unwanted moments.
- In some cases we need to handle lazily initialized objects with special care, or we might end up with an exception.

EAGER LOADING

- **Eager Loading :** A design pattern in which data initialization occurs on the spot.
- **Advantages**:
- No delayed initialization-related performance impacts
- **Disadvantages**:
- Long initial loading time
- Loading too much unnecessary data might impact performance

- Hibernate Query Language (HQL) query language designed as an object-oriented extension to SQL.
- HQL bridges the gap between the object-oriented systems and relational databases.
- The data from object-oriented systems are mapped to relational databases with a SQL-based schema.
- The HQL syntax is very similar to the SQL syntax.

JSPIDERS BASAVANAGUDI

HIBERNATE QUERY LANGUAGE
HQL

By JSPIDERS FACULTY TEAM

HQL

- Hibernate Query Language (HQL) query language designed as an object-oriented extension to SQL.
- HQL bridges the gap between the object-oriented systems and relational databases.
- The data from object-oriented systems are mapped to relational databases with a SQL-based schema.
- The HQL syntax is very similar to the SQL syntax.
- HQL can also be used to retrieve objects from database.

HQL

USING HQL WE PERFORM FOLLOWING ACTIONS ON DATABASE

- Apply restrictions to properties of objects
- Arrange the results returned by a query by using the order by clause
- Paginate the results
- Aggregate the records by using group by and having clauses
- Use Joins
- Create user-defined functions
- Execute subqueries

HQL

CREATING A QUERY WITH QUERY INTERFACE

- Query object represents HQL query in OBJECT ORIENTED format.

- using session object we can create Query object by calling createQuery().
  Ex : Query query=session.createQuery("from EntityName”);

- createQuery() : returns the object of Query which can be used to create and execute HQL on Database.

HQL

METHODS OF QUERY

- **executeUpdate() :** is used to execute the update or delete query.
- **list() :** returns the result of the SELECT operation as a list.
- **setFirstResult(int rowno) :** specifies the row number from where record will be retrieved.
- **setMaxResult(int rowno) :** specifies the no. of records to be retrieved from the table.
- **setParameter(int position, Object value):** it sets the value to the JDBC style query parameter like PreparedStatement.
- **setParameter(String name, Object value) :** it sets the value to a named query parameter like PreparedStatement.

HQL

METHODS OF QUERY

- **executeUpdate() :** is used to execute the update or delete query.

Example:

```
Query q=session.createQuery("update Enityname set column1=:n where column=:i");

q.setParameter("n",value);
q.setParameter("i",value);

q.executeUpdate();
```

Example :

```
Query query=session.createQuery("delete from Enityname  where column= value");
query.executeUpdate();
```

JSPIDERS BASAVANAGUDI

HIBERNATE CACHING

HIBERNATE CACHING

- Hibernate caching will improve the performance of the application by storing the objects in the cache memory which are results of Select operation.

- It is very helpful whenever we have to Select the same set of records multiple times.

- **There are two types of caching:**
- First Level Cache
- Second Level Cache

HQL

FIRST LEVEL CACHE

- Session object stores the first level cache data.
- First Level Cache is enabled by default.
- The first level cache data will not be available to entire application.
- An application can use many session object.

SECOND LEVEL CACHE

- SessionFactory object holds the second level cache data.
- Second Level Cache uses a common cache for all the session object of a session factory. It is useful if you have multiple session objects from a session factory.
- The data stored in the second level cache will be available to entire application.
- Second Level Cache should be enabled explicitly by Programmer

ADDING SECOND LEVEL CACHE

- Ehcache, a widely used, open-source Java-based cache. It features memory and disk stores, listeners, cache loaders
- Add maven dependency for Ehcache

```
<dependency>
   <groupId>org.ehcache</groupId>
   <artifactId>ehcache</artifactId>
   <version>3.1.3</version>
</dependency>
```

- Add maven dependency for Ehcache

```
@Cacheable

@Cache(usage=CacheConcurrencyStrategy.READ_ONLY)
```

THE END