**Hibernate**

**Hibernate 6.5**

**│**

**├── Core**

**│ ├── Configuration**

**│ │ ├── ConfigurationBuilder**

**│ │ ├── ConfigurationService**

**│ │ └── StandardServiceRegistry**

**│ ├── Session**

**│ │ ├── SessionFactory**

**│ │ ├── Session**

**│ │ └── StatelessSession**

**│ ├── Transaction**

**│ │ ├── Transaction**

**│ │ └── TransactionCoordinator**

**│ ├── EntityManager**

**│ │ ├── EntityManagerFactory**

**│ │ ├── EntityManager**

**│ │ └── EntityTransaction**

**│ ├── Query**

**│ │ ├── HQL (Hibernate Query Language)**

**│ │ ├── Criteria API**

**│ │ └── Native SQL Queries**

**│ └── Caching**

**│ ├── First-Level Cache (Session Cache)**

**│ ├── Second-Level Cache (SessionFactory Cache)**

**│ └── Query Cache**

**│**

**├── ORM (Object-Relational Mapping)**

**│ ├── Entity Mapping**

**│ │ ├── @Entity**

**│ │ ├── @Table**

**│ │ ├── @Id**

**│ │ ├── @Column**

**│ │ └── @ManyToOne, @OneToMany, etc.**

**│ ├── Relationships**

**│ │ ├── One-to-One**

**│ │ ├── One-to-Many**

**│ │ └── Many-to-Many**

**│ ├── Inheritance**

**│ │ ├── Single Table Inheritance**

**│ │ ├── Table per Class Inheritance**

**│ │ └── Table per Subclass Inheritance**

**│ └── Mapping Files**

**│ ├── XML Mapping**

**│ └── Annotation Mapping**

**│**

**├── Tools**

**│ ├── Hibernate Tools**

**│ │ ├── HBM2DDL (Schema Generation)**

**│ │ ├── HQL Query Testing**

**│ │ └── Code Generation**

**│ ├── Hibernate Metamodel Generator**

**│ └── Performance Tuning**

**│ ├── Query Optimization**

**│ └── Caching Strategies**

**│**

**└── Integration**

**├── Spring Framework Integration**

**├── Java EE Integration**

**├── JPA (Java Persistence API)**

**│ ├── JPA Annotations**

**│ └── JPA Query Language (JPQL)**

**└── Other Integrations**

**├── Jakarta EE**

**└── MicroProfile**

**Hibernate is an object-relational mapping (ORM) framework for Java that provides a framework for mapping an object-oriented domain model to a relational database. It simplifies the development of Java applications to interact with the database by allowing developers to work with objects rather than SQL queries.**

[**https://docs.jboss.org/hibernate/orm/6.5/quickstart/html\_single/**](https://docs.jboss.org/hibernate/orm/6.5/quickstart/html_single/)

[**https://hibernate.org/orm/documentation/6.5/**](https://hibernate.org/orm/documentation/6.5/)

[**https://docs.jboss.org/hibernate/orm/6.5/introduction/html\_single/Hibernate\_Introduction.html**](https://docs.jboss.org/hibernate/orm/6.5/introduction/html_single/Hibernate_Introduction.html)

**### What is Hibernate?**

**Hibernate is an open-source object-relational mapping (ORM) framework for Java applications. It provides a framework for mapping an object-oriented domain model to a relational database, simplifying the development process by allowing developers to interact with the database using Java objects instead of SQL queries.**

**### Introduction to Hibernate**

**Hibernate was developed by Gavin King in 2001 to provide a better persistence framework than EJB (Enterprise JavaBeans) and other ORM tools available at the time. It abstracts the complexities of database interactions, making it easier for developers to work with data in a more natural, object-oriented way.**

**### Object-Relational Mapping (ORM)**

**ORM is a technique that allows developers to convert data between incompatible type systems in object-oriented programming languages. ORM tools like Hibernate map Java objects to database tables and manage the relationships between these objects and the database.**

**### Benefits of Hibernate for Application Development**

* **Simplifies Development: Hibernate reduces the need for extensive JDBC code, making database interactions easier.**
* **Database Independence: Hibernate abstracts database-specific code, allowing for easy migration between different databases.**
* **Automatic Table Creation: Hibernate can automatically create and update database tables based on the Java class structure.**
* **Caching: Hibernate provides caching mechanisms to improve application performance.**
* **Lazy Loading: Hibernate supports lazy loading, which means that associated data is loaded on demand, improving efficiency.**
* **HQL and Criteria Queries: Hibernate offers powerful query capabilities through HQL (Hibernate Query Language) and Criteria queries.**

**### Comparing Hibernate with EJB**

* **Ease of Use: Hibernate is generally easier to use and configure compared to EJB.**
* **Lightweight: Hibernate is considered lightweight, whereas EJB is heavier and more complex.**
* **Flexibility: Hibernate offers more flexibility in terms of configuration and customization.**
* **Performance: Hibernate can be more performant due to its caching and lazy loading features.**
* **Transaction Management: EJB has built-in support for distributed transactions, whereas Hibernate typically relies on third-party transaction management systems.**

**### Hibernate Framework**

**The Hibernate framework consists of several key components:**

**1. Core: Provides the core functionality of Hibernate.**

**2. Annotations: Allows configuration using Java annotations.**

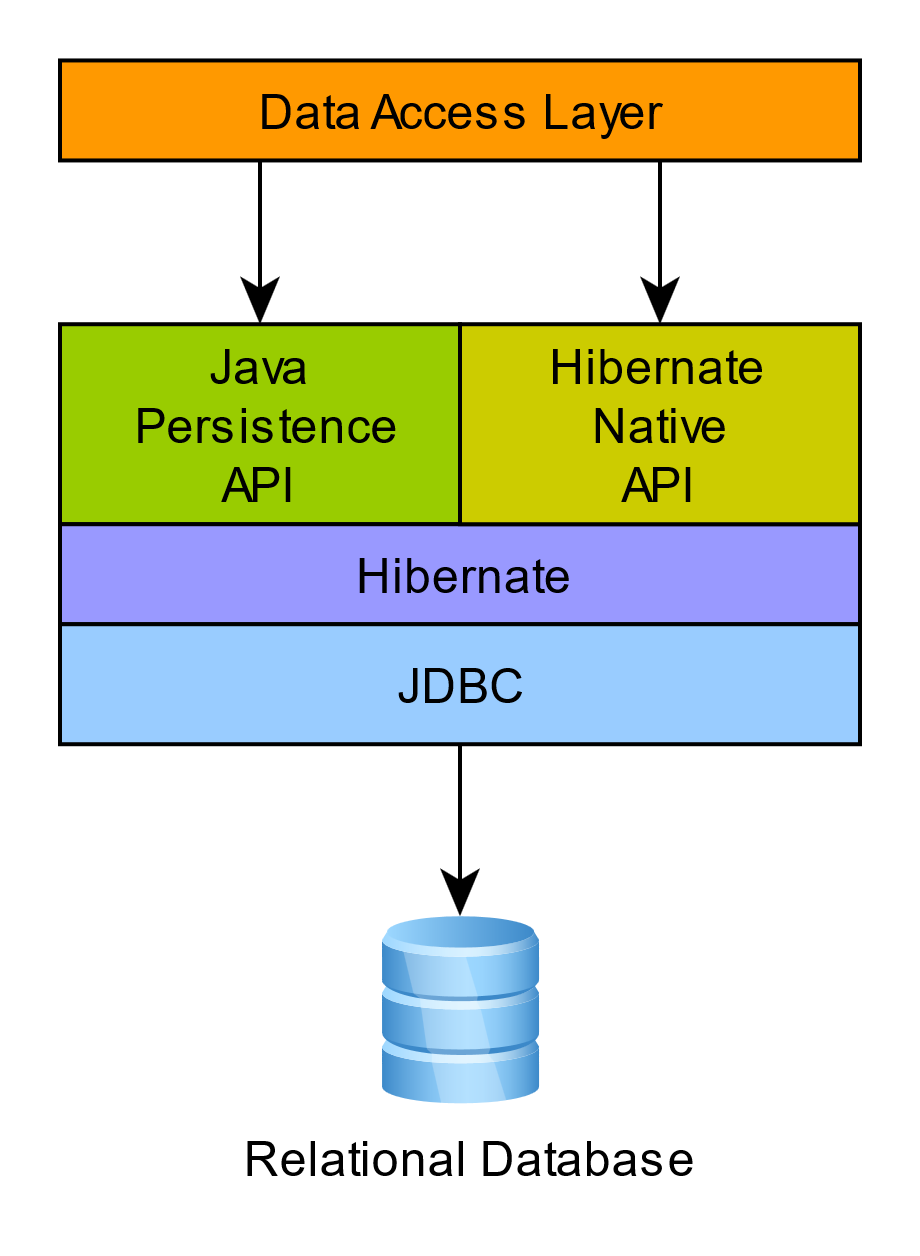
**3. EntityManager: Implements the JPA (Java Persistence API) standard.**

**4. Validator: Provides a framework for validating entities.**

**5. Search: Integrates full-text search capabilities into Hibernate.**

**### Hibernate Architecture**

**The Hibernate architecture includes several layers:**

****

**The Hibernate architecture is structured around the following core components:**

**1. Configuration Object: Configures Hibernate and provides a way to load Hibernate settings from configuration files (usually `hibernate.cfg.xml`).**

**2. SessionFactory: A factory for `Session` objects. It is created once during application initialization and is an immutable, thread-safe object.**

**3. Session: A single-threaded, short-lived object representing a conversation between the application and the database. It provides methods to create, read, and delete operations for instances of mapped entity classes.**

**4. Transaction: Manages transaction boundaries. It abstracts application code from the underlying transaction management system.**

**5. Query: Represents a Hibernate query object, used to execute queries against the database.**

**6. Criteria: Provides a simplified API for retrieving entities by composing criteria objects.**

**-------------------------------------------------------------------------------------------------------------------------------**

**### Jakarta (Java) Persistence API (JPA)**

**The Jakarta Persistence API (JPA) is a specification within the Jakarta EE (formerly Java EE) platform that provides a framework for managing relational data in Java applications. It defines a set of interfaces and standards for object-relational mapping (ORM), which allows developers to interact with databases using Java objects, rather than writing raw SQL queries.**

**### Key Features of JPA:**

**1. Object-Relational Mapping (ORM): JPA allows you to map Java classes to database tables and Java objects to rows in those tables. This mapping is typically done through annotations or XML configuration.**

**2. Entity Management: In JPA, a database table is represented by an entity class. Each instance of an entity class corresponds to a row in the table. JPA provides mechanisms to perform CRUD (Create, Read, Update, Delete) operations on these entities.**

**3. EntityManager: The `EntityManager` is the primary interface used to interact with the persistence context. It handles operations such as persisting entities, removing entities, querying the database, and managing transactions.**

**4. Query Language: JPA includes JPQL (Java Persistence Query Language), a query language similar to SQL but designed to work with JPA entities. JPQL is database-agnostic, meaning it works across different database systems.**

**5. Transactions: JPA provides transaction management features, allowing developers to manage database transactions declaratively or programmatically.**

**6. Cascading and Relationships: JPA supports relationships between entities, such as one-to-one, one-to-many, many-to-one, and many-to-many. It also supports cascading operations, where actions on one entity can cascade to related entities.**

**7. Lifecycle Callbacks: JPA allows developers to define methods in entity classes that will be called automatically at certain points in the entity's lifecycle, such as before an entity is persisted or after an entity is updated.**

**8. Criteria API: In addition to JPQL, JPA offers a Criteria API for building type-safe, dynamic queries programmatically, which is useful when you need to construct queries dynamically at runtime.**

**### Benefits of Using JPA:**

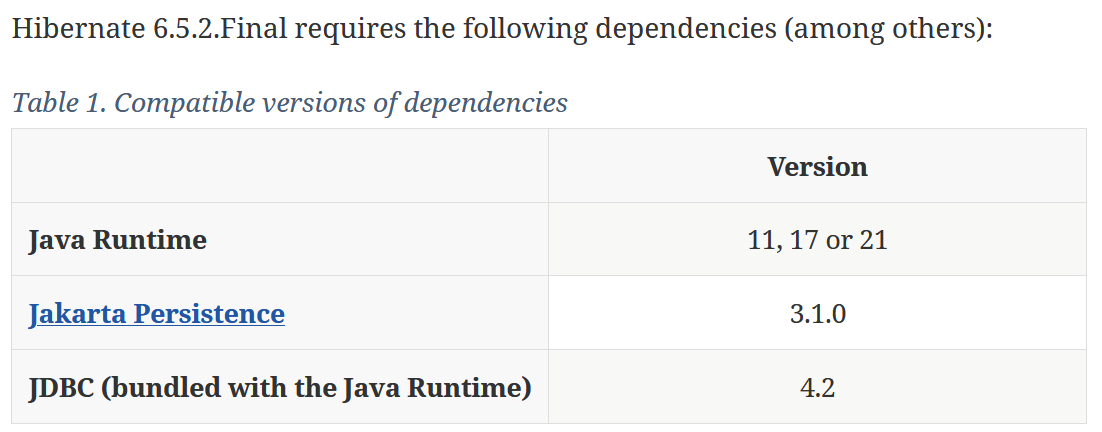
**- Simplified Data Access: JPA abstracts the underlying database interactions, allowing developers to work with Java objects instead of raw SQL.**

**- Portability: JPA is a standard specification, so applications developed using JPA can be ported across different Jakarta EE-compliant application servers and database systems.**

**- Integration with Jakarta EE: JPA integrates seamlessly with other Jakarta EE technologies like Jakarta CDI (Contexts and Dependency Injection) and Jakarta Transaction.**

**JPA is often used with Hibernate, which is a popular implementation of the JPA specification, offering additional features beyond the standard.**

**### Compatibility**

****

**If you get Hibernate from Maven Central, it is recommended to import Hibernate Platform as part of your dependency management to keep all its artifact versions aligned.**

**BOM stands for "Bill of Materials".**

**A Bill of Materials (BOM) is a special kind of POM (Project Object Model) file that defines a set of dependencies and their versions, which can be used across multiple projects. It's essentially a centralized repository of dependency versions that can be easily managed and shared.**

**A BOM typically contains:**

1. **Dependency versions: A list of dependencies with their specified versions.**
2. **Dependency management: A way to manage dependencies and their versions across multiple projects.**

**By using a BOM, you can:**

1. **Simplify dependency management: No need to specify versions for each dependency in every project.**
2. **Ensure consistency: Use the same versions across multiple projects.**
3. **Easily update dependencies: Update versions in one place (the BOM) and it will be reflected in all projects.**

**How to integrate Hibernate and other related dependencies in a Maven project**

**Using the Jakarta EE Platform BOM and hibernate BOM is a good practice when working with multiple Jakarta EE APIs, ensuring that your project remains consistent and easy to maintain.**

**<dependencyManagement>**

**<dependencies>**

**<!-- Jakarta EE Platform BOM -->**

**<dependency>**

**<groupId>jakarta.platform</groupId>**

**<artifactId>jakarta.jakartaee-bom</artifactId>**

**<version>10.0.0</version> <!-- Version aligned with Jakarta EE 10 -->**

**<type>pom</type>**

**<scope>import</scope>**

**</dependency>**

**<dependency>**

**<groupId>org.hibernate.orm</groupId>**

**<artifactId>hibernate-platform</artifactId>**

**<version>6.5.2.Final</version>**

**<type>pom</type>**

**<scope>import</scope>**

**</dependency>**

**</dependencies>**

**</dependencyManagement>**

**<!-- use the versions from the platform -->**

**<dependencies>**

**<dependency>**

**<groupId>org.hibernate.orm</groupId>**

**<artifactId>hibernate-core</artifactId>**

**</dependency>**

**<dependency>**

**<groupId>jakarta.transaction</groupId>**

**<artifactId>jakarta.transaction-api</artifactId>**

**</dependency>**

**<dependency>**

**<groupId>jakarta.persistence</groupId>**

**<artifactId>jakarta.persistence-api</artifactId>**

**</dependency>**

**<!-- https://mvnrepository.com/artifact/jakarta.servlet.jsp/jakarta.servlet.jsp-api -->**

**<dependency>**

**<groupId>jakarta.servlet.jsp</groupId>**

**<artifactId>jakarta.servlet.jsp-api</artifactId>**

**</dependency>**

**<!-- https://mvnrepository.com/artifact/jakarta.servlet/jakarta.servlet-api -->**

**<dependency>**

**<groupId>jakarta.servlet</groupId>**

**<artifactId>jakarta.servlet-api</artifactId>**

**</dependency>**

**<!-- Jakarta Activation API (optional, add if needed) -->**

**<dependency>**

**<groupId>jakarta.activation</groupId>**

**<artifactId>jakarta.activation-api</artifactId>**

**</dependency>**

**<!-- https://mvnrepository.com/artifact/jakarta.servlet.jsp.jstl/jakarta.servlet.jsp.jstl-api -->**

**<dependency>**

**<groupId>jakarta.servlet.jsp.jstl</groupId>**

**<artifactId>jakarta.servlet.jsp.jstl-api</artifactId>**

**</dependency>**

**<!-- https://mvnrepository.com/artifact/org.glassfish.web/jakarta.servlet.jsp.jstl -->**

**<dependency>**

**<groupId>org.glassfish.web</groupId>**

**<artifactId>jakarta.servlet.jsp.jstl</artifactId>**

**<version>3.0.0</version>**

**</dependency>**

**<!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->**

**<dependency>**

**<groupId>com.mysql</groupId>**

**<artifactId>mysql-connector-j</artifactId>**

**<version>8.4.0</version>**

**</dependency> </dependencies>**

**### Generator Class**

**The generator class in Hibernate is used to generate unique identifiers for the database records. Common generator strategies include:**

**- `increment`: Generates identifiers by incrementing the maximum identifier value found in the database.**

**- `identity`: Relies on auto-increment columns in the database.**

**- `sequence`: Uses a database sequence to generate identifiers.**

**- `hilo`: Uses the hi/lo algorithm to generate identifiers.**

**### Dialects**

**Hibernate supports various SQL dialects to accommodate different types of databases. Each dialect allows Hibernate to generate optimized SQL for the specific database. Some common dialects include:**

**- `org.hibernate.dialect.MySQLDialect`**

**- `org.hibernate.dialect.OracleDialect`**

**- `org.hibernate.dialect.PostgreSQLDialect`**

**### Mapping**

**Mapping is the process of linking a class to a database table. It can be done using:**

**- XML Mapping Files: Define mappings in XML files.**

**- Annotations: Use JPA annotations directly in the Java classes.**

**### Annotations**

**Hibernate uses JPA annotations for mapping entities to database tables:**

**- `@Entity`: Declares a class as an entity.**

**- `@Table`: Specifies the table name.**

**- `@Id`: Declares the primary key.**

**- `@GeneratedValue`: Specifies the strategy for generating primary key values.**

**- `@Column`: Specifies the column name.**

**- `@OneToOne`, `@OneToMany`, `@ManyToOne`, `@ManyToMany`: Defines relationships between entities.**

**### Transaction Management**

**Transaction management in Hibernate can be handled in different ways:**

**- JDBC Transactions: Use JDBC to manage transactions.**

**- JTA Transactions: Use Java Transaction API for managing distributed transactions.**

**### Hibernate Query Language (HQL)**

**HQL is an object-oriented query language, similar to SQL but operates on persistent objects and their properties rather than tables and columns.**

**### Hibernate Criteria Query Language (HCQL)**

**HCQL is a simplified API for retrieving entities by composing criteria objects. It provides a more object-oriented way of querying the database.**

**### CRUD Operations**

**CRUD operations refer to the basic operations of Create, Read, Update, and Delete. In Hibernate, these can be performed using:**

**- Create: Use the `save()` or `persist()` method to insert records.**

**- Read: Use the `get()`, `load()`, or `createQuery()` methods to retrieve records.**

**- Update: Use the `update()` or `merge()` methods to update records.**

**- Delete: Use the `delete()` method to remove records.**

**Reading Material:**

[**https://hibernate.org/orm/books/**](https://hibernate.org/orm/books/)

1. **Java Persistence with Hibernate**

**Christian Bauer, Gavin King, and Gary Gregory**

* **ISBN 9781617290459**
* **608 pages**

Hibernate

Architecture, JPA, Generator class, Dialects, Mapping, Annotations,

Transaction Management, HQL, HCQL, CRUD operations

What is hibernate ?, introduing about hibernate, ORM, benefits of hibernate for application development, comparing hiberate with EJB, discuss hiberate framework, Overview of Hibernate Architecture, configuring hibernate

--------------------------

Create hibernate session, configuring mapping properties, discuss hibernate buit-in types, mapping beans with the database

Mapping value type objetcs, discuss different components used in class, mapping collections, what is relation ?, how to map database relationship, discuss associations between entities,

Implementing query language using HQL, discuss HQL synatx with SQL, discuss how to bind parameters and iternate result

what is criteria query using HQL, implementing criteria query, dosciss various methods from restrictions class

Identifying the Object States, discuss life cycle of an object, discuss in details about various states of an object

Working with persistent objects, discuss CRUD operations using persistent object

Implementing transaction, discuss the various properies of a Transaction, states of transaction, Configuring Transactions either JDBC or JPA, what is concurrency ?, discuss various issues, controlling concurrency issues, discuss with an example

P4: Java Persistence with JPA

Overview of Java Persistence API (JPA) and Object-Relational Mapping (ORM) concepts.

Setting up a database connection and configuring JPA.

Implementing CRUD (Create, Read, Update, Delete) operations using JPA