**Introduction to Hibernate:-**

🡪Hibernate is an **ORM (Object-Relational Mapping)** framework, which helps map Java objects to database tables and vice versa.

🡪Normally, we have to write SQL queries to store or retrieve data from a database but using Hibernate, we can skip a lot of that SQL coding and directly work with Java objects.

* **Features of Hibernate/Why to use Hibernate:-**
  + **Simplifies Database Work:** Instead of writing SQL, we can just create and work with Java objects.
  + **Automatic Mapping:** Hibernate knows how to match Java classes with database tables and columns.
  + **Less Code:** we can save and load objects without writing complex SQL codes.

**Architecture of Hibernate:-**

🡪The Hibernate architecture consists of several key components that interact to enable database operations in Java applications.

* **Features of Hibernate/Why to use Hibernate:-**

1. **Hibernate Application Layer:**

* The topmost layer is the application code that interacts with Hibernate by calling the necessary API methods for CRUD operations.

1. **Configuration:**

* The Configuration object is responsible for loading and reading the hibernate.cfg.xml file, where the database connection, Hibernate settings, and mapping information are stored.

1. **SessionFactory:**

* The SessionFactory is created by passing the Configuration object. It holds the configuration settings and provides the Session objects to interact with the database.

1. **Session:**

* A session represents a single unit of work with the database. All database interactions are done using the Session interface, which handles operations like saving, updating, and deleting persistent objects.

1. **First-Level Cache:**

* The Session object maintains a cache of all objects loaded during the current session. This cache reduces database calls by reusing the objects that were loaded previously in the same session.

1. **Persistent Objects:**

* These are Java objects that are mapped to database tables using Hibernate annotations or XML configurations. They are created or fetched by the Session object and are automatically synchronized with the database.

1. **Transaction:**

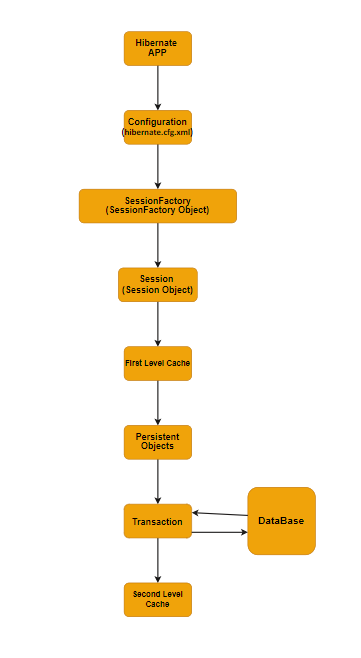
* The Transaction object controls the transaction management in Hibernate. You can begin, commit, or roll back transactions using this object**.**

1. **Database:**

* This is the actual database where all the data resides. Hibernate interacts with the database using JDBC to execute SQL queries on the backend.

1. **Second-Level Cache:**

* This optional cache is shared across multiple sessions and stores frequently accessed data to reduce database load.

**(Architecture of Hibernate)**

CODE EXAMPLE:- (Insertdata.java,Student.java) FILE\_Name:UNIT4

**Hibernate Object-Relational Mapping:-**

🡪Hibernate ORM (Object-Relational Mapping) is a technique for mapping Java objects (entities) to database tables.

* **Key Components of Hibernate OR Mapping:-**

1. **Entity Class:-**

* An entity class in Hibernate is a Java class that represents a database table.
* These classes are annotated with **@Entity** and are typically mapped to a specific table in the database using **@Table**
* **@Entity**: This annotation tells Hibernate that the class is an entity and should be mapped to a table in the database.
* **@Table**: Specifies the name of the database table to which the entity class is mapped.

1. **Primary Key Mapping:-**

* Every entity needs a primary key that uniquely identifies each row in the database table. The primary key is mapped using the **@Id** annotation.
* **@Id:** Specifies the field that acts as the primary key for the entity.
* **@GeneratedValue:** Specifies how the primary key is generated. This can be done in various ways:

1. **GenerationType.IDENTITY**: Relies on the database to generate the primary key.
2. **GenerationType.AUTO**: Let Hibernate choose the best strategy for the database.
3. **Column Mapping:-**

* The **@Column annotation** is used to specify the properties of the columns in the database.
* **@Column**: Specifies the column name in the database table and allows to configure other properties like column name, column length, nullable, etc.

1. **Relationship Mapping:-**

* Hibernate supports various relationships between entities, such as One-to-One, One-to-Many, Many-to-One, and Many-to-Many.

1. **@OneToOne**: Represents a one-to-one relationship between two entities.
2. **@OneToMany**: Represents a one-to-many relationship where one entity can have multiple related entities.
3. **@ManyToOne**: Represents a many-to-one relationship where multiple entities can be associated with one entity.
4. **@ManyToMany**: Represents a many-to-many relationship where multiple entities are associated with multiple related entities.
5. **Lazy and Eager Fetching:-**

* Hibernate provides two fetching strategies for handling how associations (like collections or related entities) are loaded:

1. **Lazy Fetching**: Associations are not loaded from the database until they are explicitly accessed.
2. **Eager Fetching**: Associations are loaded immediately when the entity is loaded.
3. **Inheritance Mapping:-**

* Hibernate supports object-oriented inheritance and provides strategies to map inheritance structures to database tables. There are three main inheritance strategies:-

1. **Single Table Strategy**: All classes in the inheritance hierarchy are mapped to a single table.
2. **Table per Class Strategy**: Each class in the hierarchy is mapped to its own table.
3. **Joined Strategy**: Each class in the hierarchy has its own table, and the tables are joined via foreign keys.
4. **Embeddable Objects:-** Sometimes objects that don’t need a separate table but instead should be embedded in an existing entity. These objects are modeled using the **@Embeddable** annotation.

* **Embeddable**: Indicates that the class can be embedded in another entity.
* **@Embedded**: Used within an entity class to embed an embeddable object.

**EXAMPLE Related it present in (Hibernate OR Mapping.ipynb)**

**Configuring Hibernates development environment:-**

🡪 Configuring a Hibernate development environment involves setting up various components and tools to enable Hibernate to work.

* **Key Components Required:-**

1. **JDK (Java Development Kit)**: The latest version of Java that will be used for compiling and running java application.
2. **Hibernate Libraries**: These are the core libraries of Hibernate ORM that enable ORM functionality.
3. **IDE (Integrated Development Environment)**: An IDE like **Eclipse**, **IntelliJ IDEA**, or **NetBeans** is used to write and run Java code.
4. **Database**: A relational database management system (RDBMS) like **MySQL**, **PostgreSQL**, **Oracle**, etc., to persist data.
5. **Hibernate Configuration File**: The hibernate.cfg.xml file contains settings required for connecting to the database and configuring Hibernate.
6. **JPA Annotations**: Java Persistence API (JPA) annotations like @Entity, @Id, @Column, etc., are used for mapping Java objects to the database tables.

**Implementing hibernate OR Mapping:-**

* Implementing Hibernate Object-Relational Mapping involves setting up a Java application where Java objects are mapped to relational database tables.
* Hibernate is an ORM (Object-Relational Mapping) framework that simplifies database interactions by using Java objects to represent database records.
* This eliminates the need for manual JDBC code for CRUD operations, and it automates SQL generation, making database interactions more efficient.

CODE EXAMPLE:- (Main.java,Student.java) FILE\_Name:UNIT4\_part2

**Hibernate Query Language (HQL):-**

🡪 Hibernate Query Language (HQL) is a powerful, object-oriented query language used in Hibernate ORM to query data from the database.

🡪 HQL is similar to SQL, but it operates on persistent Java objects rather than directly on database tables.

🡪 HQL abstracts the underlying database-specific SQL syntax, making queries more portable across different databases.

* **Key Features of HQL**:-

1. **Object-Oriented:** Unlike SQL, which works with tables, HQL works with Java objects.
2. **Portability**: Hibernate translates HQL into the database's native SQL syntax at runtime, which means the same HQL query works across different database (Mysql,Oracle,etc).
3. **Supports All SQL Operations**: HQL supports common SQL operations, such as SELECT, UPDATE, DELETE, INSERT, and JOIN, but operates on objects.
4. **Named Parameters**: HQL supports named parameters for query conditions, making it more readable and preventing SQL injection attacks.
5. **Support for Relationships**: HQL allows you to easily work with relationships between entities e.g. such as one-to-one, one-to-many, and many-to-many, by using association names.

* Basic Syntax of HQL:-

1. **Select Query:-** A typical HQL query starts with from, followed by the class name instead of the table name.
2. **Select with Conditions (WHERE Clause):-**

* we can add conditions to filter results using where, similar to SQL’s WHERE clause.
* Here, **:courseName** is a named parameter, and **query.setParameter()** is used to bind the value "Computer Science" to the query.

1. **Ordering Results (ORDER BY):-**

* We can order the results in ascending or descending order using the order by clause.
* Ascending:- ASC
* Descending:- DESC

1. **SELECT Specific Column:-**

* We can also select specific fields (columns) rather than fetching the entire Column.

1. **Aggregate Functions:-**

* HQL supports aggregate functions like count(), sum(), avg(), min(), and max().
* HQL Advanced Features:-

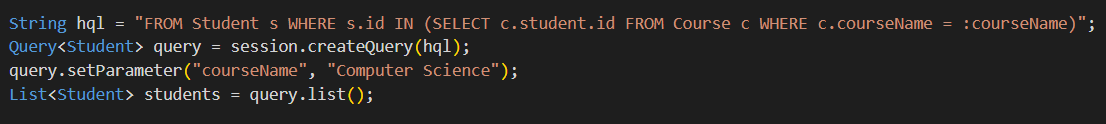
1. **Bulk Operations:-**

* We can perform bulk update or delete operations directly in HQL.

1. **Subqueries:-**

* HQL supports subqueries for complex queries. we can use subqueries inside the SELECT, FROM, or WHERE clauses.

Code Example:-



CODE EXPLANATION:-

1. **String hql = "FROM Student s WHERE s.id IN (SELECT c.student.id FROM Course c WHERE c.courseName = :courseName)";**

* **FROM Student s**:- Student is the entity class (mapped to a database table), and s is an alias for this entity.
* **WHERE s.id IN (subquery)**:- This condition filters the results by checking if the id of the Student (s.id) exists in the set of results returned by the subquery.
* **(SELECT c.student.id FROM Course c WHERE c.courseName = :courseName)**:- This is a **subquery**. It selects the id of the student (who is enrolled in a course) from the Course table (mapped to the Course entity).

**🡪** c is an alias for the Course entity.

**🡪 WHERE c.courseName = :courseName** is a condition inside the subquery that filters the courses where the course name matches the parameter :courseName

1. **Query<Student> query = session.createQuery(hql);**

* The **session.createQuery(hql)** method creates a query that can be executed on the current Hibernate session.
* **Query<Student>** specifies that this query will return a list of Student objects.

1. **query.setParameter("courseName", "Computer Science");**

* This line sets the value for the **:courseName** named parameter in the HQL query. The **setParameter** method binds the value "Computer Science" to the **:courseName** placeholder in the query.

1. **List<Student> students = query.list();**

* This line executes the query using **query.list().** This method returns a list of Student objects that match the condition defined in the HQL query.

**EXAMPLE Related BASIC AND ADVANCED HQL are present in (HQL\_SYNTAX\_EXAMPLE.ipynb)**