In order to communicate with java object with Database we are using ORM (Object Relational Mapping) tool. We have tools like ibatis, Hibernate.

But Hibernate is most widely used.

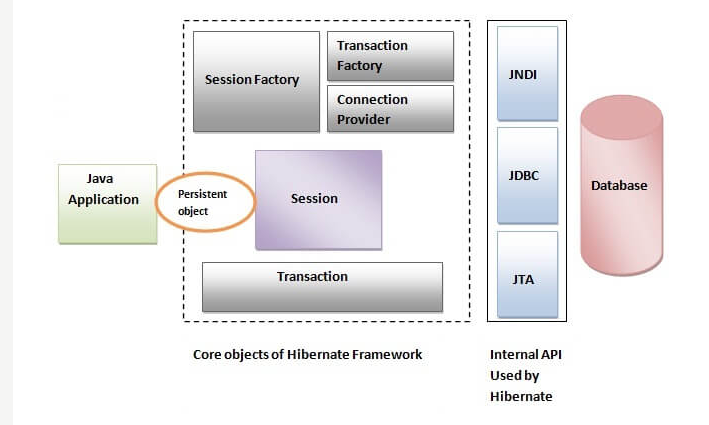
**ORM Tool:**

Simplifies data creation, manipulation and data access.

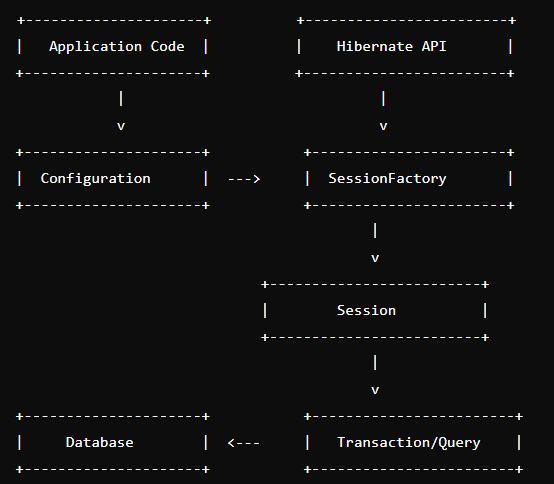
**Advantages:**

* Open source and Light weight
* Fast performance
* Database independent query
* Automation table creation
* Simplifies complex join
* Provides query statistics and Database status

**Architecture**



**Flow :**

****

1. **Configuration Object**
2. **SessionFactory**
3. **Session**
4. **Transaction**
5. **Query and Criteria API**
6. **Hibernate Mapping Files**
7. **Persistent Objects (Entities)**
8. **Hibernate Dialect**
9. **Hibernate Context (Session Context)**

**Overall Flow**

**Configuration Initialization**:

The application starts by loading the Configuration object. Configuration loads settings and mappings from hibernate.cfg.xml or programmatically.



**SessionFactory Creation**:

The SessionFactory is built from the Configuration. SessionFactory is a singleton and is created once.



**Session Management**:

For each database interaction, a Session is opened from the SessionFactory. The Session acts as an interface between the application and the database. Maintains second-level cache (optional).



**Transaction Handling**:

A Transaction is initiated through the Session. Database operations like save, update, delete, etc., are performed within this transaction.



**Query Execution**:

Queries are executed using HQL, Criteria API, or native SQL. Results are fetched and managed as persistent objects.



**Transaction Commit or Rollback**:

After completing the operations, the transaction is committed to persist changes in the database. If there is an error, the transaction is rolled back to maintain database integrity.

**Session Closure**:

The session is closed after completing the operations to release database connections. The SessionFactory remains alive for the application's lifetime.

**Application Shutdown**:

When the application is shut down, the SessionFactory is closed to release all resources.

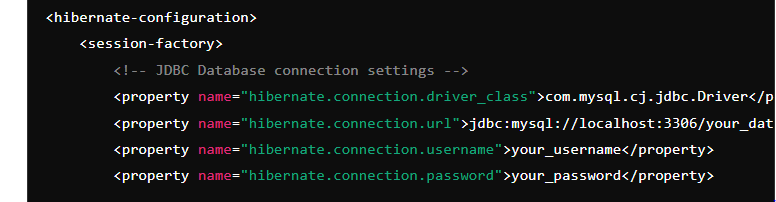
**Configuration:**

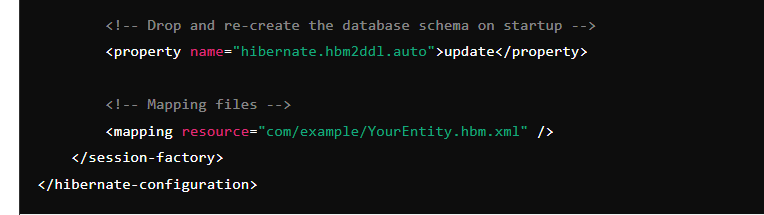
We can able to configure the hibernate by

xml based configuration and java based configuration

**xml based configuration:**

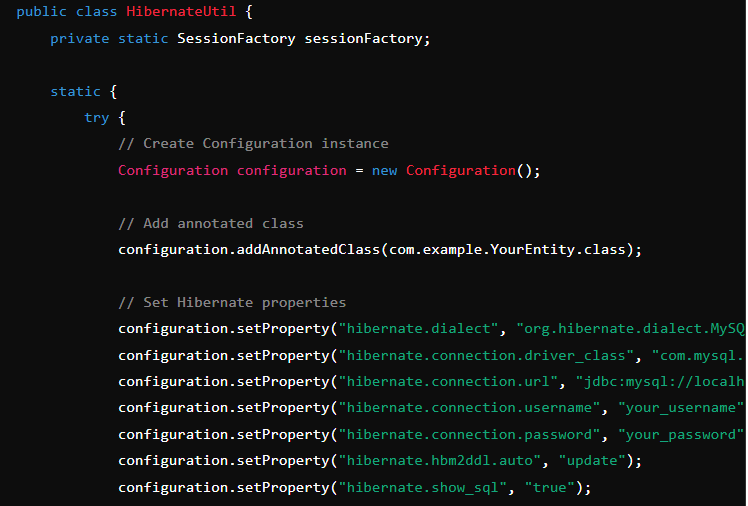
Default file name : **hibernate.cfg.xml**

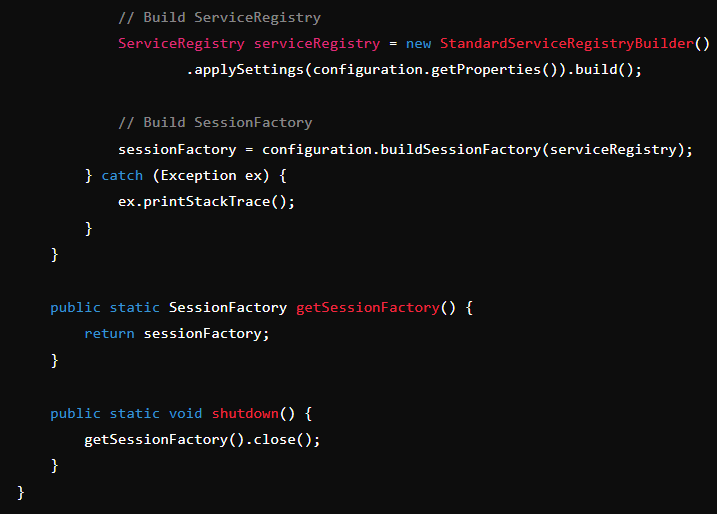




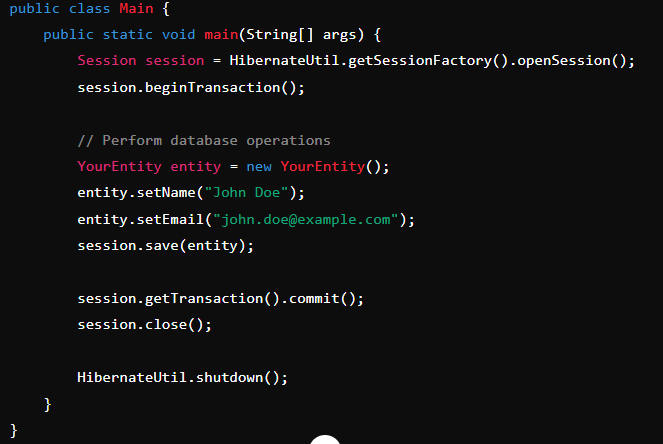
**java based configuration:**

**Configure Hibernate Programmatically**:

****

****

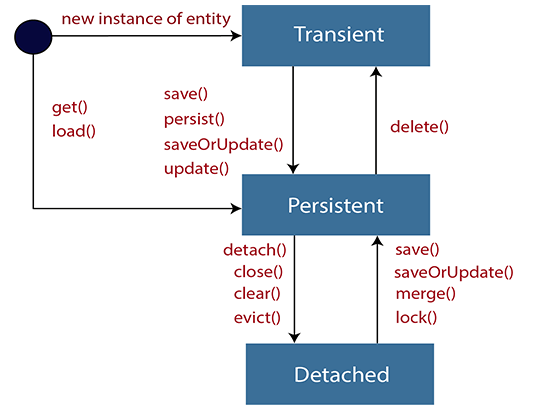
**Use Hibernate in Application**:

****

# Hibernate Lifecycle

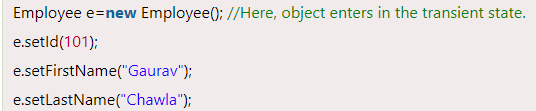
The Hibernate lifecycle contains the following states: -

* Transient state
* Persistent state
* Detached state

****

**Transient state**

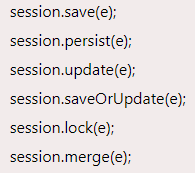
* The transient state is the initial state of an object.
* Once we create an instance of POJO class, then the object entered in the transient state.
* Here, an object is not associated with the Session. So, the transient state is not related to any database.
* Hence, modifications in the data don't affect any changes in the database.
* The transient objects exist in the heap memory. They are independent of Hibernate.

****

**Persistent state**

* As soon as the object associated with the Session, it entered in the persistent state.
* Hence, we can say that an object is in the persistence state when we save or persist it.
* Here, each object represents the row of the database table.
* So, modifications in the data make changes in the database.

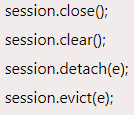
We can use any of the following methods for the persistent state.

****

**Detached State**

* Once we either close the session or clear its cache, then the object entered into the detached state.
* As an object is no more associated with the Session, modifications in the data don't affect any changes in the database.
* However, the detached object still has a representation in the database.
* If we want to persist the changes made to a detached object, it is required to reattach the application to a valid Hibernate session.
* To associate the detached object with the new hibernate session, use any of these methods - load(), merge(), refresh(), update() or save() on a new session with the reference of the detached object.

We can use any of the following methods for the detached state.

****

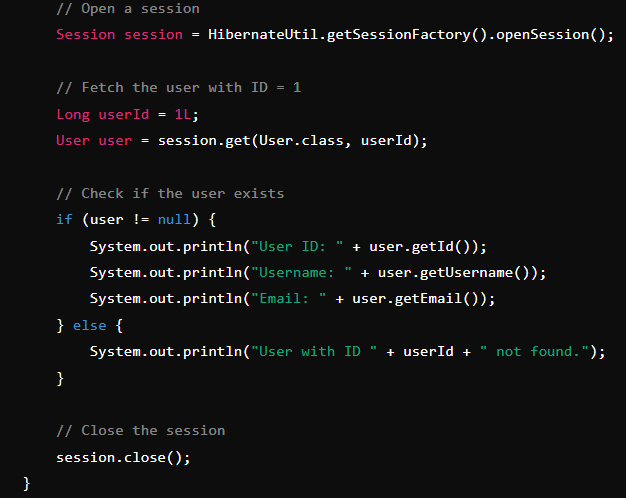
**Annotations :**

Some of the popular annotations are.

1. **@Entity(name= “EntityName”)** -> Specify the Entity Name.
2. **@Table(name= “TableName”)** -> Specify the Table Name.
3. **@Id** -> Specify the primary key of the entity
4. **@Column(name=”columnName”)** -> Specify the column name.
5. **@Generated Value** -> Defines the strategy for generating the primary key value
6. **@Temporal** -> Specifies the temporal data type of field, such as DATE, TIME, or TIMESTAMP.
7. **@OneToOne** -> Specifies a one-to-one relationship between two entities
8. **@OneToMany** -> Specifies a one-to-many relationship between two entities.
9. **@ManyToOne** -> Specifies a many-to-one relationship between two entities.
10. **@ManyToMany** -> Specifies a many-to-many relationship between two entities.
11. **@JoinColumn** -> Specifies the foreign key column for an entity's relationship field.
12. **@JoinTable** -> Specifies the table that is used to join two entities in a many-to-many relationship.
13. **@Embedded** -> Embeds a class as a component within another entity, without a separate database table.
14. **@Embaddable** -> Specifies a class whose instances are stored as intrinsic parts of an owning entity and share the primary key of the entity.
15. **@Transient** -> Marks a field of an entity to be ignored by the persistence mechanism, i.e., it won’t be stored in the database.

**How to Fetch data**

The session.get() method in Hibernate is used to retrieve an entity from the database based on its primary key.

****

**Difference between get() and load()**

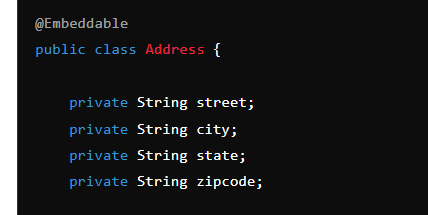
**get()** is best when you need immediate access to the data **and can handle a null return if the entity doesn't exist.**

**load()”** is better when you want to defer the database access until you actually need the data and are certain that the entity exists **(otherwise, be prepared to handle an ObjectNotFoundException)**

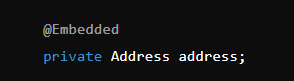
**What is mean by Embeddable and Embadded Object**

In Hibernate, the @Embeddable annotation is used to define a class whose instances are stored as an intrinsic part of the owning entity and do not have a separate identity from the owning entity.

It will not create new table.

****

Of course we need to use getters and setters



**Hibernate Mapping**

* OneToOne
* OneToMany
* ManyToOne
* ManyToMany

**What is Fetch Eager and Lazy:**

We have two type of fetching techniques. By default is **Lazy**

**Lazy Fetching**: Loads associated entities only when they are accessed. Default for collections. Can improve performance by reducing initial data load but may cause additional queries later.

**Eager Fetching**: Loads associated entities immediately with the main entity. Ensures all data is available right away but can lead to performance issues if not managed carefully.

**what is N+1 select problem**

Occurs when the application executes one query to fetch a list of entities, followed by N queries to fetch their associated entities, leading to performance issues.

**How to Avoid the N+1 Select Problem**

Use eager fetching, JOIN FETCH, batch fetching, or sub-select fetching to optimize query execution and reduce the number of queries.

**Hibernate Caching**

**What is Caching:**

**Caching** is a mechanism used to temporarily store copies of data in memory, so that subsequent requests for the same data can be served faster, without needing to repeatedly access the underlying data source (e.g., a database). In the context of Hibernate, caching is a critical feature that helps improve application performance by reducing the number of database queries and speeding up data retrieval.

**Types of Caching in Hibernate**

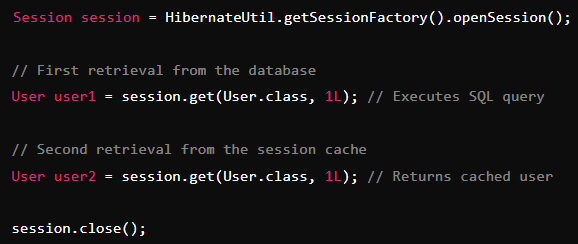
Hibernate supports different levels of caching, each serving different purposes:

1. **First-Level Cache (Session Cache)**
2. **Second-Level Cache**
3. **Query Cache**

**1. First-Level Cache (Session Cache)**

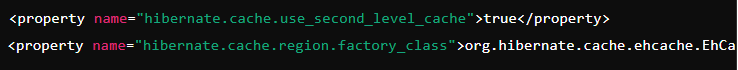
* **Scope**: The first-level cache is associated with the Hibernate Session object. It is enabled by default and cannot be disabled.
* **Functionality**:
  + When you retrieve an entity using the Session.get() or Session.load() method, Hibernate first checks if the entity is already in the session cache. If it is, Hibernate returns the cached entity instead of querying the database.
  + The first-level cache is cleared when the session is closed.

**Example**

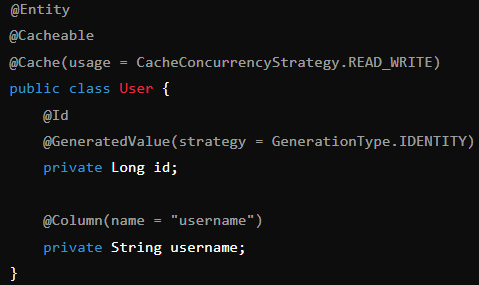
****

**2. Second-Level Cache (SessionFactory Level)**

* **Scope**: The second-level cache is associated with the SessionFactory object and is shared across multiple sessions. It is not enabled by default.
* **Functionality**:
  + It stores entities, collections, or query results across sessions, reducing the number of database hits for frequently accessed data.
  + The second-level cache persists beyond the lifespan of individual sessions but is still limited to the lifecycle of the SessionFactory.
* **Configuration**:
  + To enable the second-level cache, you need to configure a caching provider (e.g., EHCache, Infinispan) and specify which entities or collections should be cached.
  + Need to add EHCache dependency in pom.xml

****

**Annotation Example**:

****

**Benefits**:

* Reduces database load by storing frequently accessed data in the cache.
* Enhances performance for read-heavy applications where the same data is accessed frequently by different sessions.

**Cache Concurrency Strategies**:

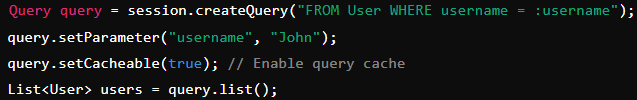
* **READ\_ONLY**: Suitable for entities that do not change.
* **NONSTRICT\_READ\_WRITE**: Allows for potential stale data; no locking is used.
* **READ\_WRITE**: Uses soft locks to ensure consistency; suitable for frequently updated data.
* **TRANSACTIONAL**: Provides full transactional isolation; typically used in JTA environments.

**3. Query Cache**

* **Scope**: The query cache is used to cache the results of query executions. It is typically used in conjunction with the second-level cache.
* **Functionality**:
  + The query cache stores the identifiers of entities returned by a query, along with the query parameters. The actual entities themselves are fetched from the second-level cache.
  + It is useful for caching the results of expensive queries.
* **Configuration**:

****

**Example**:

****

**Benefits**:

* Improves performance by caching the results of commonly executed queries.
* Reduces the load on the database for queries that return the same results over multiple executions.

**Summary of Caching in Hibernate**

* **First-Level Cache**:
  + Default, per-session cache.
  + Caches entities within a session.
  + Automatically enabled.
* **Second-Level Cache**:
  + Optional, shared across sessions.
  + Caches entities, collections, and query results.
  + Requires explicit configuration.
* **Query Cache**:
  + Optional, used in conjunction with the second-level cache.
  + Caches the results of queries (identifier lists).
  + Requires explicit configuration and marking of queries as cacheable.

**Caching Strategies** are key to optimizing the performance of Hibernate-based applications. By minimizing database access and leveraging in-memory data storage, Hibernate's caching mechanisms help speed up data retrieval and reduce the overall load on the database.

**Why we are using 2nd level cache**

The **second-level cache** in Hibernate is used to improve the performance of an application by reducing the number of database queries needed to retrieve frequently accessed data. **Unlike the first-level cache, which is specific to a single session, the second-level cache is shared across multiple sessions** and can persist beyond the lifespan of individual sessions. This makes it especially useful in scenarios where the same data is repeatedly accessed by different parts of the application.

**Example Scenario**

Consider an e-commerce application where product details are frequently displayed to users. If product information is stored in the second-level cache:

* When a user views a product page, the application can retrieve the product details from the cache, bypassing the database.
* If another user views the same product, the cached data can be reused, again avoiding a database hit.
* This significantly reduces the load on the database, speeds up page load times, and improves the overall user experience.

**Difference between 1st level and 2nd level cache**

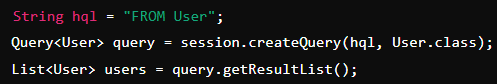
| **Feature** | **First-Level Cache** | **Second-Level Cache** |
| --- | --- | --- |
| **Scope** | Per-session | Per SessionFactory, shared across sessions |
| **Lifespan** | Exists only during the session | Persists as long as SessionFactory is alive |
| **Automatic Behavior** | Enabled by default, cannot be disabled | Optional, requires explicit configuration |
| **Data Sharing** | Not shared across sessions | Shared across sessions |
| **Caching Granularity** | Caches entities and their state within a session | Caches entities, collections, and query results |
| **Cache Clearing** | Cleared when the session is closed | Managed through configuration and cache provider |
| **Performance Impact** | Reduces queries within a session | Reduces queries across sessions |

**Hibernate Query Language:**

**Hibernate Query Language (HQL)** is a powerful query language used in Hibernate to interact with the database

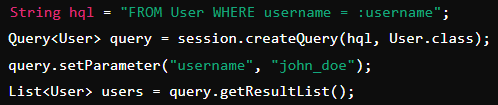
#### 1. ****Simple Select Query****

To fetch all entities of a certain type:

****

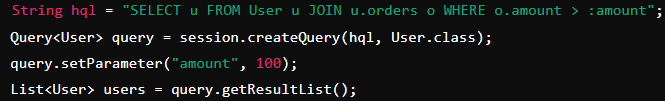
#### 2. ****Select with Condition****

To fetch entities based on a condition:

****

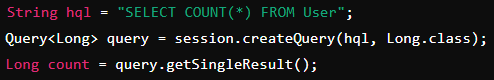
#### 3. ****Select with Joins****

To fetch entities along with their related entities:

****

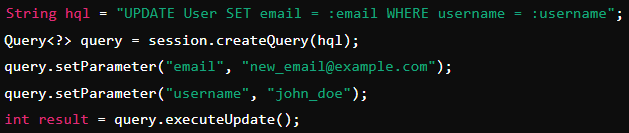
#### 4. ****Select with Aggregation****

To perform aggregate operations:

****

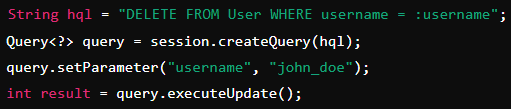
#### 5. ****Update Query****

To update entities:

****

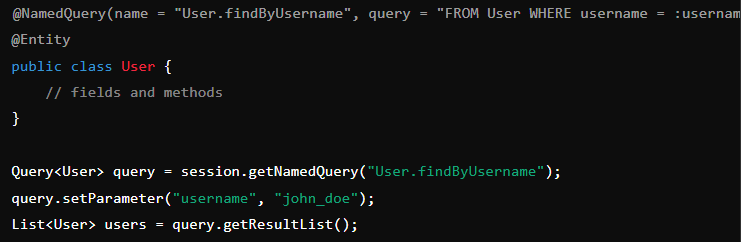
#### 6. ****Delete Query****

To delete entities:

****

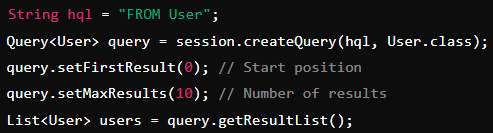
**Named Queries**:

* Named queries are predefined HQL queries that can be reused.

****

**Pagination**:

* HQL supports pagination using setFirstResult() and setMaxResults().

****