Drawbacks of JDBC:

1. JDBC code is dependent upon the Database software being using.

2. It's very expensive to change the database in the middle of the development.

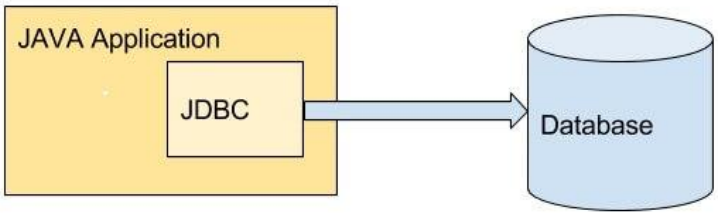
3. JDBC code is not portable over multiple database softwares.

4. We need to repeat the same lines of code over and over again in your application for fetching data from the database

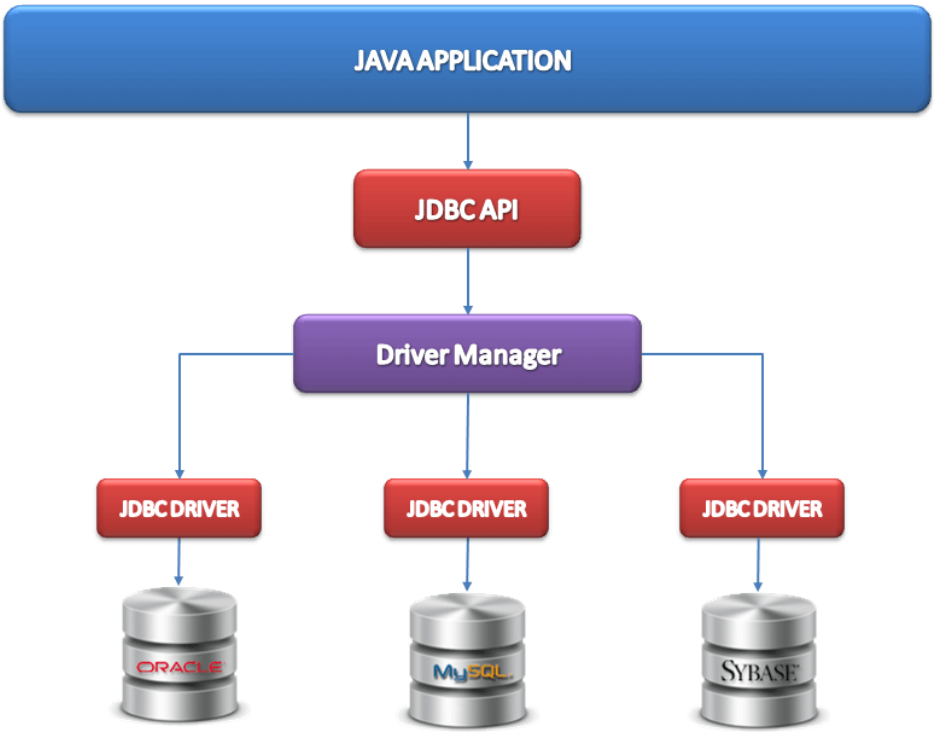
5. There is no support for Object level mapping.

6. To migrate from one database to another, we need to re write all the queries again

7. Associations among the table is also difficult.



JDBC Workflow:



Hibernate:

Hibernate is a free, open source object-relational mapping library for Java which is designed to map objects to an RDBMS and to implement the object-oriented programming concepts in a relational database.

Hibernate is a framework which provides some **abstraction layer** means programmer don't have to worry about the implementations, Hibernate do implementations for you internally like **Establishing a connection with the database, writing query to perform CRUD operations etc.**

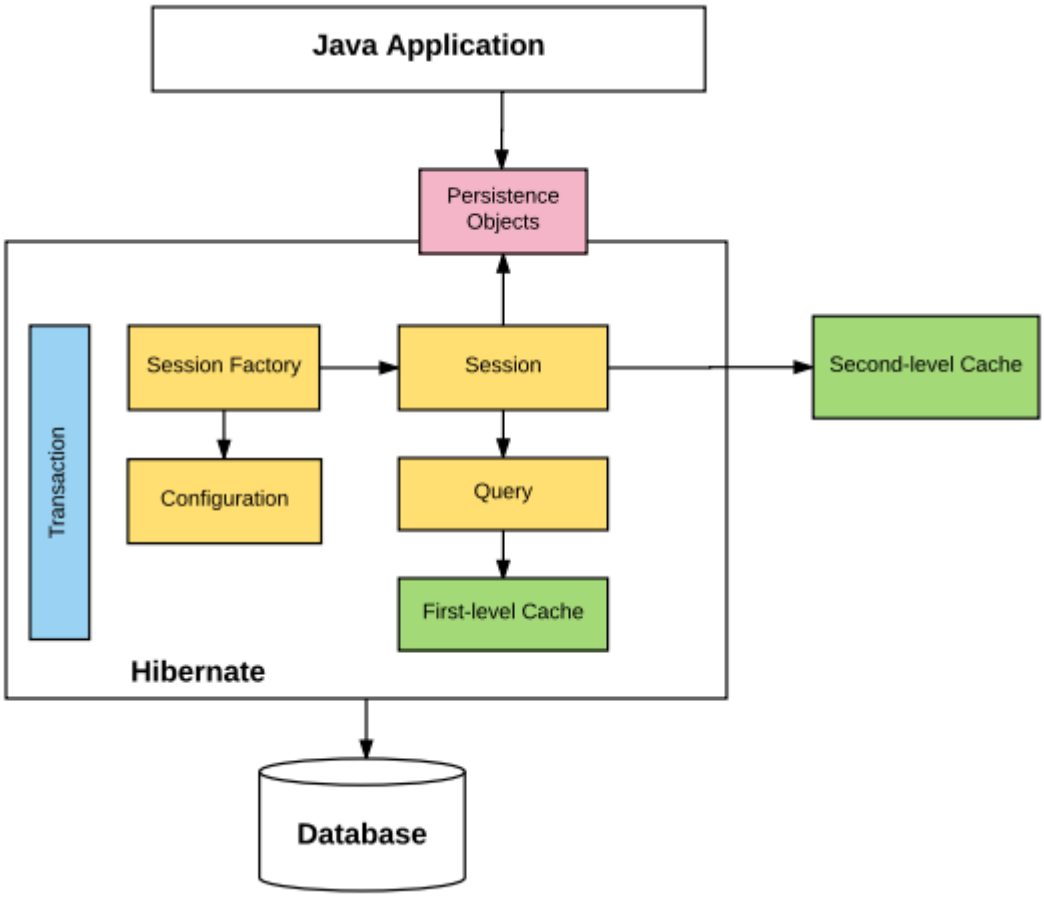
It is a java framework which is used to develop persistence logic. Persistence logic means to store and process the data for long use.

Hibernate Advantages:

1. **Open Source and Light weight**
2. Hibernate provides Object/Relational mappings. Here is different O/R mapping strategies as multiple-objects to single-row mapping, Polymorphic associations, bi-directional association, association filtering. It also provide XML mapping documents.
3. **Hibernate supports Inheritance, Associations, Collections**
4. In hibernate if we save the derived class object, then its base class object will also be stored into the database, it means hibernate supporting inheritance
5. Hibernate supports relationships like One-To-Many, One-To-One, Many-To-Many-to-Many, Many-To-One
6. This will also supports collections like List, Set, Map (Only new collections)
7. In jdbc all exceptions are checked exceptions, so we must write code in try, catch and throws, but in hibernate we only have Un-checked exceptions, so no need to write try, catch, or no need to write throws. Actually in hibernate we have the translator which converts checked to Un-checked
8. Hibernate has its own query language, I.e hibernate query language which is database independent
9. So if we change the database, then also our application will works as HQL is database independent
10. HQL contains database independent commands
11. While we are inserting any record, if we don't have any particular table in the database, JDBC will rises an error like "View not exist", and throws exception, but in case of hibernate, if it not found any table in the database this will create the table for us
12. Hibernate supports caching mechanism by this, the number of round trips between an application and the database will be reduced, by using this caching technique an application performance will be increased automatically.
13. Hibernate supports annotations, apart from XML
14. It provides session level caching which is default cache and optional second-level cache (at SessionFactory level).
15. It introduces Lazy initialization.

16. Hibernate provide outer join fetching.

Hibernate Architecture



1. **Configuration**: Generally written in `hibernate.cfg.xml` files. For Java configuration, you may find class annotated with `@Configuration`. It is used by Session Factory to work with Java Application and the Database. It represents an entire set of mappings of an application Java Types to an SQL database.
2. **Session Factory**: Any user application requests Session Factory for a session object. Session Factory uses configuration information from above listed files, to instantiates the session object appropriately.
3. **Session**: This represents the interaction between the application and the database at any point of time. This is represented by the `org.hibernate.Session` class. The instance of a session can be retrieved from the `SessionFactory` bean.
4. **Query**: It allows applications to query the database for one or more stored objects. Hibernate provides different techniques to query database, including `NamedQuery` and `Criteria API`.
5. **First-level cache**: It represents the default cache used by Hibernate Session object while interacting with the database. It is also called as session cache and caches objects within the current session. All requests from the Session object to the database must pass through the

first-level cache or session cache. One must note that the first-level cache is available with the session object until the Session object is live.

6. **Transaction**: enables you to achieve data consistency, and rollback in case something goes unexpected.
7. **Persistent Objects**: These are plain old Java objects (POJOs), which get persisted as one of the rows in the related table in the database by hibernate. They can be configured in configurations files (`hibernate.cfg.xml`) or annotated with `@Entity` annotation.
8. **Second-level cache**: It is used to store objects across sessions. This needs to be explicitly enabled and one would be required to provide the cache provider for a second-level cache. One of the common second-level cache providers is *EhCache*.

Hibernate Configuration File:

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>: This property helps hibernate to generate database-specific SQL statements. This is an optional property. According to hibernate documentation, hibernate will be able to choose the correct implementation of dialect automatically using the JDBC metadata returned by the JDBC driver.

<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>: Using this property, we can provide the Fully Qualified Name (FQN) of the java driver name for a particular database. The driver class is implemented using Java and resides in the JAR file and contains the driver that should be placed in our classpath.

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/MyDB</property>: Using this property, we can provide the physical location of the database; however, the connection URL may vary from database to database.

<property name="hibernate.connection.username">root</property>: Using this property, we can provide the username to access a particular database.

<property name="hibernate.connection.password">root</property>: Using this property, we can provide the password to access a particular database.

<property name="show_sql">true</property>: The possible value for this property is either true or false. This is an optional property. Hibernate logs all the generated queries that reach the database to the console if the value of show_sql is set to true.

<property name="hbm2ddl.auto">create</property>: The possible values are validate, update, create or create-drop. This is also an optional property. Here, we will set value=create so that it will remove all the schemas and create a new one using the hibernate mapping on each build of sessionfactory. For value=update, hibernate will update the new changes in the database.

<mapping resource="Employee.hbm.xml"/>: All of the mapping file is declared in the mapping tag, and the mapping file is always named xx.hbm.xml. We can use multiple mapping tags for multiple mapping files.

hibernate.format_sql:

The possible values are true and false

It shows the hibernate-generated queries in the pretty format if set as true

hibernate.connection.pool_size:

The possible value is always greater than 1 (value >= 1)

It limits the maximum number of pooled connections

Building a SessionFactory

1. A SessionFactory has the following features:
2. It's an interface implemented using the singleton pattern.
3. It's created using the configuration provided by the configuration file.
4. It's thread-safe, so it's created once during the application's lifetime, and multiple users or threads can access it at the same time without any concurrency issue.
5. As a SessionFactory object is immutable, changes made to the configuration will not affect the existing factory object.
6. It's a factory class, and its main duty is to create, manage, and retrieve a session on request. A Session is used to get a physical connectivity with the database.

SessionFactory Object

```
//Prior to Hibernate Version 4

Configuration cfg = new Configuration();

cfg = cfg.configure();

SessionFactory sessionFactory = cfg.buildSessionFactory();
```

As the buildSessionFactory() method of the Configuration class is deprecated in the version 4 of hibernate, you can use the following code to create a SessionFactory

```
//After version 4

Configuration configuration = new Configuration();

configuration = configuration.configure();

StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder();

builder = builder.applySettings(configuration.getProperties());

SessionFactory sessionFactory = configuration.buildSessionFactory(builder.build());
```

Opening a new session

A Session is also known as an interface that is used to get a physical connectivity with a database. It is instantiated every time we need to interact with the database for the CRUD (Create, Read, Update, Delete) operations. Persistent objects always travel from the application to the database and vice versa only through the Session.

Session session = sessionFactory.openSession();

- This will open a brand new Session for us. It opens the database connection when it is created and holds it until the session is closed. A Session created using these methods is not associated with any thread, so it's our responsibility to flush or close it once we are done with the database operation.
- A Session is a bridge between the Java application and hibernate. The Session interface wraps the JDBC connection. A Session always tries to be in sync with the persistent store where all transactions are made.
- A Session is always a part of first-level cache; it caches all the objects that are transmitted through that particular session. All cached objects will be destroyed once this session is closed.
- Opening a new Session for every database transaction is considered to be a good practice for a multithreaded application.

We can use the same session instead of creating a brand-new session; hibernate provides the facility to reuse an already created session.

SessionFactory sessionFactory = configuration.getSessionFactory();

Session session = sessionFactory.getCurrentSession();

It may seem easy to get the current session, but the twist here is that you have to provide more configuration to the Configuration object if you plan to reuse the Session

<property name="hibernate.current_session_context_class">

Thread

</property>

In the preceding code, we set a thread value for the hibernate.current_session_context_class key, meaning that the context of the current Session is limited to the life of the current thread only.

For example, in a non-multithreaded environment, a Session is created when the main thread is started. It will close automatically once the SessionFactory is closed.

1. Transient Object State:

An object which is not associated with hibernate session and does not represent a row in the database is considered as transient. It will be garbage collected if no other object refers to it.

An object that is created for the first time using the new() operator is in transient state. When the object is in transient sate then it will not contain any identifier (primary key value). You have to use save, persist or saveOrUpdate methods to persist the transient object.

Employee emp = new Employee();

emp.setName("Ravi");

// here emp object is in a transient state

2. Persistent State:

1. When the object is in persistent state, then it represent one row of the database, if the object is in persistent state then it is associated with the unique Session
2. if we want to move an object from persistent to detached state, we need to do either closing that session or need to clear the cache of the session
3. if we want to move an object from persistent state into transient state then we need to delete that object permanently from the database

An object that is associated with the hibernate session is called as Persistent object. When the object is in persistent state, then it represent one row of the database and consists of an identifier value.You can make a transient instance persistent by associating it with a Session.

3. Detached Object State:

Now, if we close the Hibernate Session, the persistent instance will become a detached instance: it isn't attached to a Session anymore (but can still be modified and reattached to a new Session later though).

- Object which is just removed from hibernate session is called as detached object.The state of the detached object is called as detached state.
- When the object is in detached sate then it contain identity but you can’t do persistence operation with that identity.
- Any changes made to the detached objects are not saved to the database. The detached object can be reattached to the new session and save to the database using update, saveOrUpdate and merge

get() vs load():

Load() Method:

1) Throws ObjectNotFoundException if object is not found.

2) load() method doesn't hit the database.

3) It returns proxy object.

4) It should be used if you are sure that instance exists.

Get() Method

1) Returns null if object is not found.

2) get() method always hit the database.

3) It returns real object not proxy.

4) It should be used if you are not sure about the existence of INSTANCE.

openSession() vs getCurrentSession():

If we talk about SessionFactory.openSession()

1. It always creates a new Session object.

2. You need to explicitly flush and close session objects.

3. In single threaded environment it is slower than getCurrentSession().

4. You do not need to configure any property to call this method.

And If we talk about SessionFactory.getCurrentSession()

1. It creates a new Session if not exists, else uses same session which is in current hibernate context.

2. You do not need to flush and close session objects, it will be automatically taken care by Hibernate internally.

3. In single threaded environment it is faster than openSession().

4. You need to configure additional property. "hibernate.current_session_context_class" to call getCurrentSession() method, otherwise it will throw an exception.