**Hibernate**

Hibernate is the ORM tool given to transfer the data between a java (object) application and a database (Relational) in the form of the objects.

**Advantages**

* Hibernate supports Inheritance, Associations, Collections
* 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
* Hibernate supports relationships like One-To-Many, One-To-One, Many-To-Many-to-Many, Many-To-One
* This will also supports collections like List, Set, Map (Only new collections)
* 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 ;)
* Hibernate has capability to generate primary keys automatically while we are storing the records into database
* Hibernate has its own query language, i.e. hibernate query language which is database independent, So if we change the database, then also our application will works as HQL is database independent
* HQL contains database independent commands
* 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 ;)
* Hibernate supports caching mechanism and 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.
* Hibernate supports annotations, apart from XML
* Hibernate provided Dialect classes, so we no need to write sql queries in hibernate, instead we use the methods provided by that API.

Draw Backs of JDBC:

* In JDBC, if we open a database connection we need to write in try, and if any exceptions occurred catch block will takers about it, and finally used to close the connections.
* Hear as a programmer we must close the connection, or we may get a chance to get our of connections message…!
* Actually if we didn’t close the connection in the finally block, then JDBC doesn’t responsible to close that connection.
* In JDBC we need to write SQL commands in various places, after the program has created if the table structure is modified then the JDBC program doesn’t work, again we need to modify and compile and re-deploy required, which is tedious.
* JDBC used to generate database related error codes if an exception will occurs, but java programmers are unknown about this error codes right.
* In the Enterprise applications, the data flow with in an application from class to class will be in the form of objects, but while storing data finally in a database using JDBC then that object will be converted into text. Because JDBC doesn’t transfer objects directly.

"Lazy loading" means that an entity will be loaded only when you actually accesses the entity for the first time.

In Hibernate, you can configure to lazily load a collection of child entities.

Lazy fetching decides whether to load child objects while loading the Parent Object.

You need to do this setting respective hibernate mapping file of the parent class. Lazy = true (means not to load child).

By default the lazy loading of the child objects is true.

**Hibernate Lifecycle of POJO Class Objects (the types of Hibernate instance states)**

Actually our POJO class object having 3 states like…

* Transient state
* Persistent state
* Detached state

Transient & Persistent states:

Whenever an object of a POJO class is created then it will be in the Transient state

When the object is in a Transient state it doesn’t represent any row of the database, i mean not associated with any Session object, if we speak more we can say no relation with the database it’s just an normal object

If we modify the data of a POJO class object, when it is in transient state then it doesn’t effect on the database table

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

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

If we want to move an object from persistent state into transient state then we need to delete that object permanently from the database

**Associations**

One-to-many

Many-to-one

One-to-one

Many-to-many