Query level cache is an optional feature
Query level cache requires two additional physical cache regions that hold the cached query results and the timestamps when a table was last updated
Query level cache is only useful for queries that are run frequently with the same parameters

SessionFactory object is used by all the threads of an application.
It is a thread safe object.
One SessionFactory object is created per database.
Multiple SessionFactory objects (each requiring a separate configuration) are created when connecting to multiple databases.
to create Hibernate Configuration file? **hibernate.cfg.xml** in **src** folder of the application
Configuration ? object is used to create SessionFactory object in hibernate?
Hibernate SessionFactory represent which level of cache? **Second Level**
Creating sessionFactory object in hibernate ? **Prototype design pattern**

Session is a lightweight non-threadsafe object
Session represents a single unit-of-work with the database
Session is the primary interface for the persistence service
Session can't share the session between threads
Session is created per thread in hibernate? **True**
Methods of session ? **Session.save() Session.saveorupdate() Session.persist()**

Only the Session that you obtained with **sf.getNewSession()** is flushed and closed automatically
What does the Session object hold ? **First Level Cache**
methods returns proxy object? **load()**
methods hits database always? **Get()**

```
// Create Session Factory Object using Annotation Configuration
    SessionFactory sessionFactory = new AnnotationConfiguration().configure().buildSessionFactory();

   //Create Session object from session factory object
    Session session = sessionFactory.openSession();
    session.beginTransaction();

    //Use the session to save model objects
    session.save(user);
    session.getTransaction().commit();
    session.close();
```

Here below property configure driver of the specific database

<property name="connection.driver_class">com.mysql.jdbc.Driver</property>
Next line we configure the Database **connection url**
suppose we want to connect the database name **hibernateDB**
<property name="connection.url">jdbc:mysql://localhost:3306/hibernateDB</property>
Next two lines set the user name and password of the connecting database **hibernateDB**
<property name="connection.username">username</property>
<property name="connection.password">password</property>
Next line configure Dialect of the database **MySQL, every database has its own Dialect.**
What is Dialect? means Dialect is configuration specify here so hibernate knows whats kind of language we are used and what type database we are used. we can say it is database dependent. It connects the database specific query language which we want to use.
<property name="dialect">org.hibernate.dialect.MySQLDialect</property>

The below line configured the **model class** name **UserDetails** its object we want save on the database **hibernateDB**
<mapping class="com.sdnext.hibernate.tutorial.dto.UserDetails"/>

state of object in Hibernate? **transient, persistent and detached**. Object before association session it is in Transient. When associated session it is in Persistent. Detaching from session it is in Detached.

```
UserDetails userDetails = new UserDetails();                                  (TRANSIENT)
        →userDetails = (UserDetails) session.get(UserDetails.class, 1);       (PERSISTENT)
        →session.beginTransaction();
         userDetails.setUserName("User Updated after session close");         (DETACHED)
        →  session.update(userDetails);                                       (PERSISTENT)
```

an inheritance mapping strategies?

- **A single table per class hierarchy.**
- **A table per concrete entity class.**
- **A "join" strategy, whereby fields or properties that are specific to a subclass are mapped to a different table than the fields or properties that are common to the parent class.**

@ElementCollection: Defines a collection of instances of a basic type or embeddable class. Must be specified if the collection is to be mapped by means of a collection table.

@Embeddable : for value object it is not is an entity object. Value object means does not have real meaning for self individually.

Table per Concrete class, there will be three tables in the database having no relations to each other. Which are the ways to map the table**? By union-subclass element, By self-creating the table for each class ,Both A & B**

We need to specify @Inheritance(strategy=InheritanceType.JOINED) in the parent class and @PrimaryKeyJoinColumn annotation in the subclasses. ? **TRUE**

hibernate.hbm2ddl.auto - create tables automatically

The property **cascade = CascadeType.ALL** indicates that when we persist, remove, refresh or merge this entity all the entities held in this field would be persist, remove, delete or update.

The main difference between is **HQL uses class name instead of table name, and property names** instead of column name. **Named Query** is very useful concept in hibernate. It lets you separate queries from coding section of the application to the mapping xml file**(.hbm files**). The query is given unique name for the entire application.

```
<sql-query name="findStudentByRollNumber">
    <return alias="student" class="com.sdnext.hibernate.tutorial.dto.Student">
     <!--[CDATA[
       select * from Student student where student.rollNumber= :rollNumber
     ]]-->
   </return>
   </sql-query>
 </hibernate-mapping>
```

The Criteria API supports all the comparision operators

such as **=, <, >, >=,=<,** etc in the supported Expression class **eq(), lt(), gt(), le() , ge()** respectively.

```
Criteria criteria = session.createCriteria(Student.class);                  // CREATE  OBJECT
criteria.add(Restrictions.eq("studentName", "Dinesh Rajput"));              // WHERE IS
criteria.add(Restrictions.le("rollNumber", 1));                             // LESS THAN
criteria.addOrder(Order.asc("studentName"));                                // ORDER  BY ASC
criteria.setMaxResults(10);                                                 // LIMIT PAGINATION
criteria.setFirstResult(5);                                                 // Starting from the 5th record
```

The first-level cache:

The first level cache type is the session cache. The session cache caches object within the current session but this is not enough for long level i.e. session factory scope.

The second-level cache:

The second-level cache is called 'second-level' because there is already a cache operating for you in Hibernate for the duration you have a session open.

A Hibernate Session is a transaction-level cache of persistent data. It is possible to configure a SessionFactory-level cache on a class-by-class and collection-by-collection basis.

— Across sessions in an Application

— Across applications (different applications on same servers with same database)

— Across clusters (different applications on different servers with same database)