Hibernate topics pending list
=======================
1. Bulk Operation
2. Locking
3. Stored Procedure
4. Mapping(1-1,1-*,*-*,*-1)
5. Connection pooling
6. Pagination
7. NamedQuery

Connection pooling
 => SessionFactory object holds jdb connection pool having set of ready made jdbc connection objects and uses them in the
      creation of HB session objects.
=> By default hibernate uses built in jdbc connection pool which is not suitable for production environment because of
     performance issue.
=> To control hibernate build in jdbc connection pool we write the following property in hibernate.cfg.xml
            <property name="hibernate.connection.pool_size">25</property>

Which jdbcconnection pool is best with hibernate integration?
      standalone mode -> Don't use hibernate built in jdbc connection pool
                                    use Third party supplied jdbcconnection pool like
                                    hikaricp(best in market),proxool,viboor,agroal,c3po.....
      webapplication mode-> Don't use 3rd party supplied only, use underlying server provided
                                          connection pool from servers like weblogic,tomcat,wildfly,....

Configuration of hibernate.cfg.xml file for hikaricp production
==
<!-- Hikari cp configuration -->
      <property name="hibernate.connection.provider_class">org.hibernate.hikaricp.internal.HikariCPConnectionProvider</property>

<!-- Maximum waiting time for a connection from the pool (20sec)-->
      <property name="hibernate.hikari.connectionTimeout">20000</property>

<!-- Minimum number of ideal connections in the pool(10 objects) -->
      <property name="hibernate.hikari.minimumIdle">10</property>

<!-- Maximum number of actual connection in the pool(20objects0 -->
      <property name="hibernate.hikari.maximumPoolSize">20</property>

<!-- Maximum time that a connection is allowed to sit ideal in the pool(300secs) -->
      <property name="hibernate.hikari.idleTimeout">30000</property>

                  refer: HibernateBuiltInConnectionPool

Note: DataSource(I)---------> jars provider should implement and give the class.

Why are we not configuring the Datasource class directly in hibernate? why are configuring connection provider class name?

Answer. hibernate f/w is designed to pickup the datasource class based on the connection provider that we have configured.
              by configuring in this style, we can restrict datasource and jdbc connection pool associated with hibernate.
                  hibernate will give support only for few thirdparty vendors like
                              a. hikaricp(best) b. c3po c.proxool d. viboor e. agroal

BulkOperation
============
=> To select or manipulate one or more record/object having our choice criterial value we need to go for "Bulk operation".
      a. HQL.
      b. Native SQL.
      c. Criterian API.

HQL
====
 1. HQL stands for Hibernate Query Language.
 2. It uses Objects based Query Language(these queries will be written based on the entity class names and properties name)
 3. Hibernate dialect internally converts HQL queries to DB specific SQL Queries.
 4. HQL queries are DBIndependent and they supports portability.
 5. HQL supports both select and non-select operation
 6. HQL can also be used to perform SingleRowOperation(SRO) and also for bulk operation having our choice conditions/
      criteria.
 7. HQL supports positional params(?)(supported only in older versions) and also it supports named params(=:name)
 8. HQL keywords are not case sensitive,but entity class names and properties names are case sensitive.
 9. HQL supports relational operators, conditional statements, joins,aggregate functions, projections,....

eg:
SQL> SELECT * FROM EMP WHERE EMPNO>=? AND EMPNO<=?
HQL> FROM in.ineuron.entity.Employee WHERE eno>=? AND eno<=?(positional param)
HQL> FROM in.ineuron.entity.Employee WHERE eno=:firstNum AND eno=:secondNum(named param)

SQL> DELETE FROM EMP WHERE JOB=?
HQL> DELETE FROM in.ineuron.entity.Employee WHERE job=?
HQL> DELETE FROM in.ineuron.entity.Employee WHERE job=:desg

Note: if we are selecting all columns/properties in the HQL Select query then placing select keyword is optional.

HQL select Queries
      a. Entity Queries(Getting all properties values of the record)
                  eg: FROM in.ineuron.entity.Employee(with or without condition)
      b. Scalar Queries(Getting specific column or specific multiple column values)
                  eg: SELECT eno,ename,eaddr FROM in.ineuron.entity.Employee (with or without condition)
                        SELECT eno From in.ineuron.entity.Employee(with or without condition)
                        SELECT count(*) From in.ineuron.entity.Employee

Example to get All the records from the DBTable using hibernate
==========
```
Query<Employee> query = session.createQuery("FROM in.ineuron.Model.Employee");
List<Employee> employees = query.list();
employees.forEach(employee -> System.out.println(employee));
```

Note:
If we use xml approach setter and getter methods are mandatory, but if we use Annotations for mapping setter and getter
methods are not required, hibernate internally uses reflection api and it binds the value from ResultSet to private properties
of the Model.

Note:
In plain jdbc converting ResultSet object to DTO object is a manual process, where as in orm framework like hibernate,
spring jdbc, spring orm and spring datajpa same happens internally using "rowmapper"concept.

list() or getResultList()
==========
1. It internally uses eager loading for bulk operations.
2. It returns the collection directly.
3. Generates only one query to get all the records.
4. suitable for good performance.
5. It won't generate proxy object.
6. It is not deprecated and it is the industry standard approach.

iterate()
======
1. It internally uses lazy loading for bulk operation
2. It returns the iterator pointing to collection object.
3. Generates n+1 query to get all the records.
4. Not suitable, it degrades the performance.
5. It generates Proxy object.
6. It is deprecated becoz of performance issue.

Bulk operation for retreiving the record
---
a. All columns ========> List<EntityType>
b. multiple columns ====> List<Object[]>
c. only one column ====> List<datatype of property>

Note: To avoid null checking in our application,we can use JDK8 supplied api called "Optional".
```
Employee employee = query.uniqueResult();
 if (employee != null)
      System.out.println(employee);
else
      System.out.println("Record not found for the given id ::"+id);

Optional<Employee> optional = query.uniqueResultOptional();
if (optional.isPresent()) {
            Employee employee = optional.get();
            System.out.println(employee);
} else{
            System.out.println("Record not found for the given id :: " + id);
}
```

Note: we should use get()/load() if we are getting record based on primary key value,
           we can use uniqueResult()/uniqueResultOptional() if we are getting record based on non-primary key value.

refer: HibernateBulkOperation

HQLInsert operation

 It is not possible to insert one record to the database directly using insert query,becoz linking generators with HQL insert query
 is not possible. so we use session.save() method to insert a record.
We can use HQL insert query to insert bulk record into one db table by selecting them from another db table.
            eg: insert into .... values (query is not given)
                  insert into ... SELECT FROM ..... (given to perform bulk operation)

insurance policy (table filled with records) ---------->
premium_insurance_policy(new table) where tenure>=25 years