
2.What is HQL?

HQL or Hibernate Query Language is an Object-oriented query language, it is like SQL. The difference is that SQL queries deal with tables and columns while HQL queries deal with objects and their properties. Hibernate translates the HQL queries to SQL and eventually the SQL queries are executed on the database.

The advantage of using HQL is that you can write queries regardless of the database and the supporting language. by changing hibernate configurations you can easily change your database from Oracle to Postgres or MySQL or etc. Hibernate

handles the translation and there is no need to change your code. Although it is possible to use native SQL, but it is

not recommended due to portability problems that may occur.

FROM, SELECT, WHERE, ORDER BY, GROUP BY, UPDATE, DELETE, INSERT Clauses are very similar to SQL.

There are also named parameters, that makes queries accept input from users and prevent SQL injection.

source:

https://www.tutorialspoint.com/hibernate/hibernate_query_language.htm

----

3.What are Naming Strategies in Hibernate?

Logical name is the name that is stored in java / hibernate

Physical name is the name in the database

There are to phases for naming, so two different teams with different concerns (object modeling and database) don't have

conflicts in naming, and can use their specific naming strategies like camelCase for object-oriented modeling and sneak_case

for database table and column naming.

-Logical Naming Strategy

 -explicit naming strategy

  we can use @Table @Column annotations to rename the table and column names explicitly

 -implicit naming strategy

  by adding a property tag to cfg and mentioning which implicit strategy to use for logical names

```
<property name="hibernate.implicit_naming_strategy" value="jpa" />
```

  types of implicit naming: default,jpa,legacy-hbm,legacy-jpa,component-path

  jpa:The logical name of an entity class is either the name provided in the @Entity annotation or the unqualified class name

  if you use @Entity (name = "MyName") the MyName should be used in the HQL or JPQL queries

  but if you don't mention a name in @Entity by default the class name is used.

-Physical Naming Strategy

by default the physical name will be the same as the logical name

-CamelCaseToUnderscoresNamingStrategy in Hibernate 5.5.4

by adding a property to hibernate cfg we can use this strategy, which changes all camelCase names to all lower-case

and containing underscores

```
<property name="hibernate.physical_naming_strategy"
   value="org. hibernate.boot.model.naming.CamelCaseToUnderscoresNamingStrategy"/>
```

-Implementing a custom physical naming strategy

by implementing the PhysicalNamingStrategy interface or

extending Hibernate’s PhysicalNamingStrategyStandardImpl class

and adding a property tag in cfg addressing the custom implemented class

source:

https://thorben-janssen.com/naming-strategies-in-hibernate-5/

https://www.baeldung.com/hibernate-naming-strategy#:~:text=3.-,Implicit%20Naming%20Strategy,defined%20explicitly%20by%20using%20annotations

https://stackoverflow.com/questions/46200399/difference-between-attribute-name-logical-name-and-physical-name-in-hibernate

----

4.How to implement Hibernate configurations in Java?

We can implement Hibernate Configurations without a cfg.xml file, by importing java.util.Properties and assigning the

values for each property, these properties are all present in the xml file such as:

hibernate.connection.url

dialect

hbm2ddl.auto

show_sql

We assign the desired values to these properties, make a new Configuration, and add these properties to it, and then we have to add the annotated classes we have defined as @Entity in our model. by this call a new SessionFactory is created and can be used to open Sessions in the program.

The code for the above explanations is available in q1. model.repository.DBConfig

Thanks to Hibernate! Happy Programming!