**1. HQL (Hibernate Query Language)**

Hibernate Query Language (HQL) is an object-oriented query language that is similar to SQL, but it operates on Java objects rather than database tables. HQL allows developers to query, update, and delete data from the database using Java class names and properties instead of tables and columns. HQL abstracts away database-specific SQL syntax and allows the developer to write queries in terms of domain model objects.

**Key Features of HQL:**

* **Object-Oriented**: It works with Java objects, not tables or columns.
* **Cross-Database Compatibility**: Since it works at the object level, HQL queries are portable across different databases.
* **Supports Relationships**: You can query associated objects (e.g., @OneToMany, @ManyToOne) easily.

**Example of HQL Query:**

String hql = "FROM Person WHERE age > :age";

Query query = session.createQuery(hql);

query.setParameter("age", 18);

List<Person> results = query.list();

In this example:

* "FROM Person" queries the Person entity.
* The :age parameter is used to filter the results, and the query is executed with the specified parameter.

**HQL Basic Syntax:**

* **SELECT Clause**: You specify the properties or objects to retrieve. You can use SELECT to fetch specific properties.
* SELECT p.name, p.age FROM Person p WHERE p.age > :age
* **FROM Clause**: Specifies the entities (or objects) you are querying.
* **WHERE Clause**: Conditions to filter the results.

**2. HCQL (Hibernate Criteria Query Language)**

HCQL, also known as **Hibernate Criteria API**, is a programmatic, object-oriented way of querying the database. Unlike HQL, which uses string-based queries, HCQL uses a Java-based API that allows the construction of queries dynamically and safely, without the need for raw SQL strings.

**Key Features of HCQL:**

* **Type-Safe**: It avoids common errors in HQL (such as misspelled fields) by using strongly typed Java code.
* **Dynamic Queries**: HCQL allows you to construct complex queries dynamically (e.g., based on user input or other conditions).
* **Criteria API**: It provides a richer, more flexible way of querying the database.

**Example of HCQL:**

Criteria criteria = session.createCriteria(Person.class);

criteria.add(Restrictions.gt("age", 18)); // age > 18

List<Person> results = criteria.list();

* createCriteria(Person.class): Specifies the entity (or table) to query.
* Restrictions.gt("age", 18): Adds a condition where the age must be greater than 18.

**HCQL Common Methods:**

* **add()**: Adds conditions to the query (e.g., restrictions like gt, eq, lt).
* **setFirstResult()**: Sets the starting point for the query result.
* **setMaxResults()**: Limits the number of results returned.
* **list()**: Executes the query and returns the results as a list of entities.

**Restrictions in HCQL:**

* **eq**: Equal to.
* **ne**: Not equal to.
* **gt**: Greater than.
* **lt**: Less than.
* **like**: String match using the LIKE operator.

**3. Native SQL Queries**

Native SQL is simply SQL written for a particular database, and it can be used in Hibernate to execute SQL queries directly. Hibernate allows you to execute native SQL queries using the createSQLQuery method.

**Key Features of Native SQL:**

* **Database-Specific**: Native SQL is written according to the specific SQL syntax of the database you are using (e.g., MySQL, PostgreSQL, Oracle).
* **Full SQL Support**: It allows for complex queries and operations not supported by HQL or HCQL.
* **Not ORM-Based**: Native SQL does not interact with Hibernate’s object model; it works directly with the database.

**Example of Native SQL Query:**

String sql = "SELECT \* FROM person WHERE age > :age";

SQLQuery query = session.createSQLQuery(sql);

query.addEntity(Person.class); // Maps the result to the Person class

query.setParameter("age", 18);

List<Person> results = query.list();

In this example:

* createSQLQuery(sql): Executes a raw SQL query.
* addEntity(Person.class): Maps the result set to the Person entity.
* setParameter("age", 18): Sets the parameter for the query.

**When to Use Native SQL:**

* When you need to execute a very complex query that cannot be expressed in HQL or HCQL.
* When you need to use database-specific functions or syntax not supported by HQL.

**4. Named Queries**

A **Named Query** is a pre-defined query in Hibernate that is defined at the class level using annotations or in the Hibernate configuration file. These queries are defined with a specific name and can be reused throughout the application.

**Key Features of Named Queries:**

* **Reusable**: Named queries are reusable and don't require recreation for each query execution.
* **Defined in Annotations or XML**: Named queries can be defined in the entity class using annotations (@NamedQuery) or in the Hibernate configuration (hibernate.cfg.xml).
* **Predefined**: They are defined once and used multiple times, reducing the overhead of query creation.

**Example of Named Query using Annotations:**

import javax.persistence.NamedQuery;

import javax.persistence.Entity;

@Entity

@NamedQuery(name = "Person.findByAge", query = "FROM Person WHERE age > :age")

public class Person {

// fields, getters, setters

}

In this example:

* @NamedQuery defines a query with the name Person.findByAge that retrieves Person entities with an age greater than the specified parameter.

**Example of Using Named Query in Code:**

Query query = session.getNamedQuery("Person.findByAge");

query.setParameter("age", 18);

List<Person> results = query.list();

Named queries can also be defined in the **Hibernate configuration file** (hibernate.cfg.xml), like so:

<hibernate-mapping>

<class name="Person" table="person">

<query name="Person.findByAge">

<![CDATA[

FROM Person WHERE age > :age

]]>

</query>

</class>

</hibernate-mapping>

**Summary**

* **HQL (Hibernate Query Language)**: An object-oriented query language that works with entities (Java objects) rather than database tables, simplifying the interaction with the database.
* **HCQL (Hibernate Criteria Query Language)**: A programmatic, object-oriented way of constructing queries dynamically and safely, using Java objects.
* **Native SQL**: SQL queries executed directly on the database. It is used when you need full control over SQL syntax or need database-specific queries that aren't supported by HQL or HCQL.
* **Named Queries**: Predefined queries that are reusable and defined either via annotations or XML configuration. They are useful when you need to define a query once and reuse it multiple times.

These query mechanisms in Hibernate provide developers with flexibility when it comes to querying databases, allowing them to choose between high-level object-oriented approaches (HQL, HCQL) and low-level direct SQL (Native SQL) based on the complexity of the query.