Hibernate provides two main approaches for database queries:

HQL (Hibernate Query Language) and native SQL queries. Here are examples of both:

1. Hibernate Query Language (HQL)
HQL is an object-oriented query language similar to SQL but works with persistent objects instead of database tables.

Basic HQL Examples

```
// 1. Simple SELECT query
String hql = "FROM Employee";
Query<Employee> query = session.createQuery(hql, Employee.class);
List<Employee> employees = query.getResultList();

// 2. SELECT with WHERE clause
String hql = "FROM Employee WHERE department = :dept AND salary >
:minSalary";
Query<Employee> query = session.createQuery(hql, Employee.class);
query.setParameter("dept", "Engineering");
query.setParameter("minSalary", 50000.0);
List<Employee> employees = query.getResultList();

// 3. SELECT specific columns (returns Object[])
String hql = "SELECT e.firstName, e.lastName FROM Employee e WHERE
e.joinDate > :date";
Query<Object[]> query = session.createQuery(hql, Object[].class);
query.setParameter("date", LocalDate.of(2020, 1, 1));
List<Object[]> results = query.getResultList();

// 4. JOIN query
String hql = "SELECT e.name, d.name FROM Employee e JOIN e.department d
WHERE d.location = :loc";
Query<Object[]> query = session.createQuery(hql, Object[].class);
query.setParameter("loc", "New York");
List<Object[]> results = query.getResultList();

// 5. UPDATE query
String hql = "UPDATE Employee SET salary = salary * 1.1 WHERE department
= :dept";
Query query = session.createQuery(hql);
query.setParameter("dept", "Engineering");
int updatedCount = query.executeUpdate();

// 6. DELETE query
String hql = "DELETE FROM Employee WHERE active = false AND lastLoginDate
< :date";
Query query = session.createQuery(hql);
query.setParameter("date", LocalDate.now().minusYears(1));
int deletedCount = query.executeUpdate();
```

2. Native SQL Queries
Native queries allow you to write database-specific SQL directly.

Basic Native SQL Examples

```
// 1. Simple SELECT native query
String sql = "SELECT * FROM employees";
Query<Employee> query = session.createNativeQuery(sql, Employee.class);
```

```
List<Employee> employees = query.getResultList();

// 2. SELECT with parameters
String sql = "SELECT * FROM employees WHERE department = ? AND salary >
?";
Query<Employee> query = session.createNativeQuery(sql, Employee.class);
query.setParameter(1, "Engineering");
query.setParameter(2, 50000);
List<Employee> employees = query.getResultList();

// 3. Named parameters in native query
String sql = "SELECT * FROM employees WHERE department = :dept AND
hire_date > :date";
Query<Employee> query = session.createNativeQuery(sql, Employee.class);
query.setParameter("dept", "HR");
query.setParameter("date", LocalDate.of(2021, 1, 1));
List<Employee> employees = query.getResultList();

// 4. JOIN with native query
String sql = "SELECT e.* FROM employees e JOIN departments d ON e.dept_id
= d.id " +
             "WHERE d.location = :location";
Query<Employee> query = session.createNativeQuery(sql, Employee.class);
query.setParameter("location", "Chicago");
List<Employee> employees = query.getResultList();

// 5. UPDATE with native query
String sql = "UPDATE employees SET salary = salary * 1.05 WHERE
department_id = ?";
Query query = session.createNativeQuery(sql);
query.setParameter(1, 10);
int updatedRows = query.executeUpdate();

// 6. Stored procedure call
String sql = "{call calculate_year_end_bonus(:employeeId,
:bonusPercentage)}";
Query query = session.createNativeQuery(sql);
query.setParameter("employeeId", 12345);
query.setParameter("bonusPercentage", 0.15);
query.executeUpdate();
```

Key Differences

HQL:

Object-oriented

Uses entity and property names

Database agnostic

Supports polymorphism

Easier to maintain when schema changes

Native SQL:

Uses actual table and column names

Database specific

Can use database-specific features

Better for complex queries that are hard to express in HQL

May need modification if database changes

Best Practices:

Prefer HQL for most queries as it's more maintainable

Use native SQL for complex queries or when you need database-specific features

Always use parameter binding to prevent SQL injection

For native queries, consider using ResultSetMapping for complex result types

For performance-critical operations, native SQL might be faster