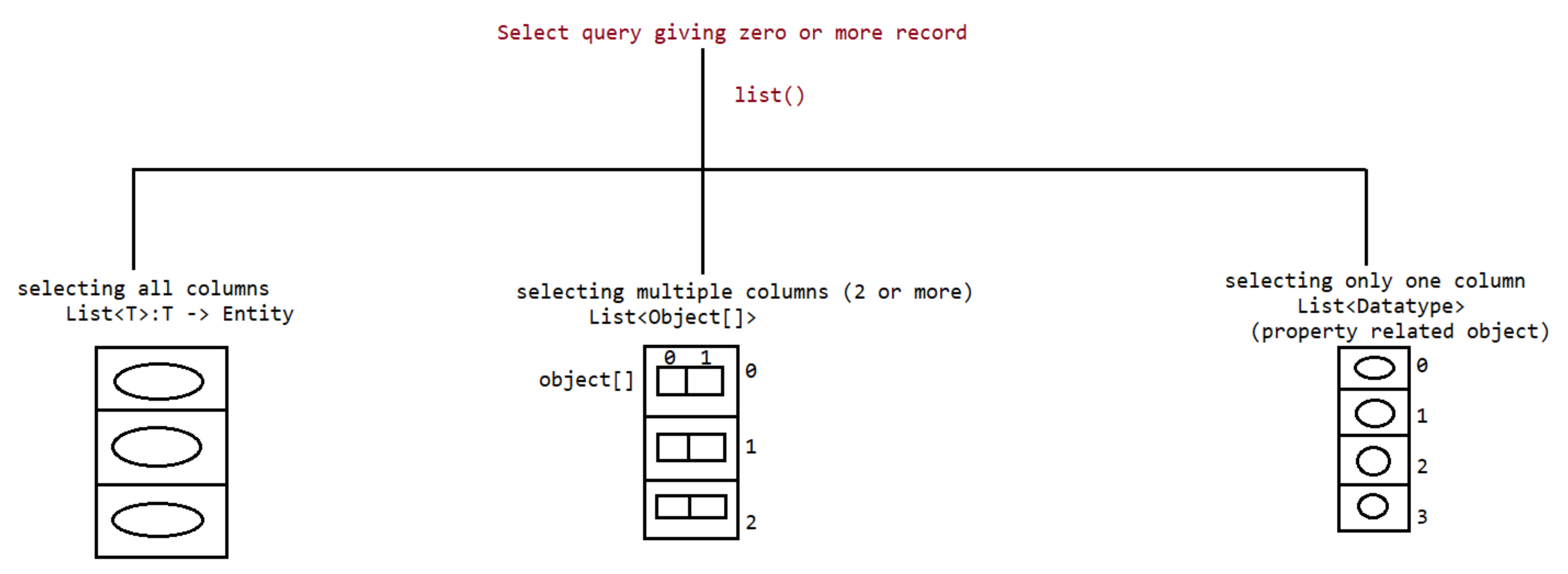
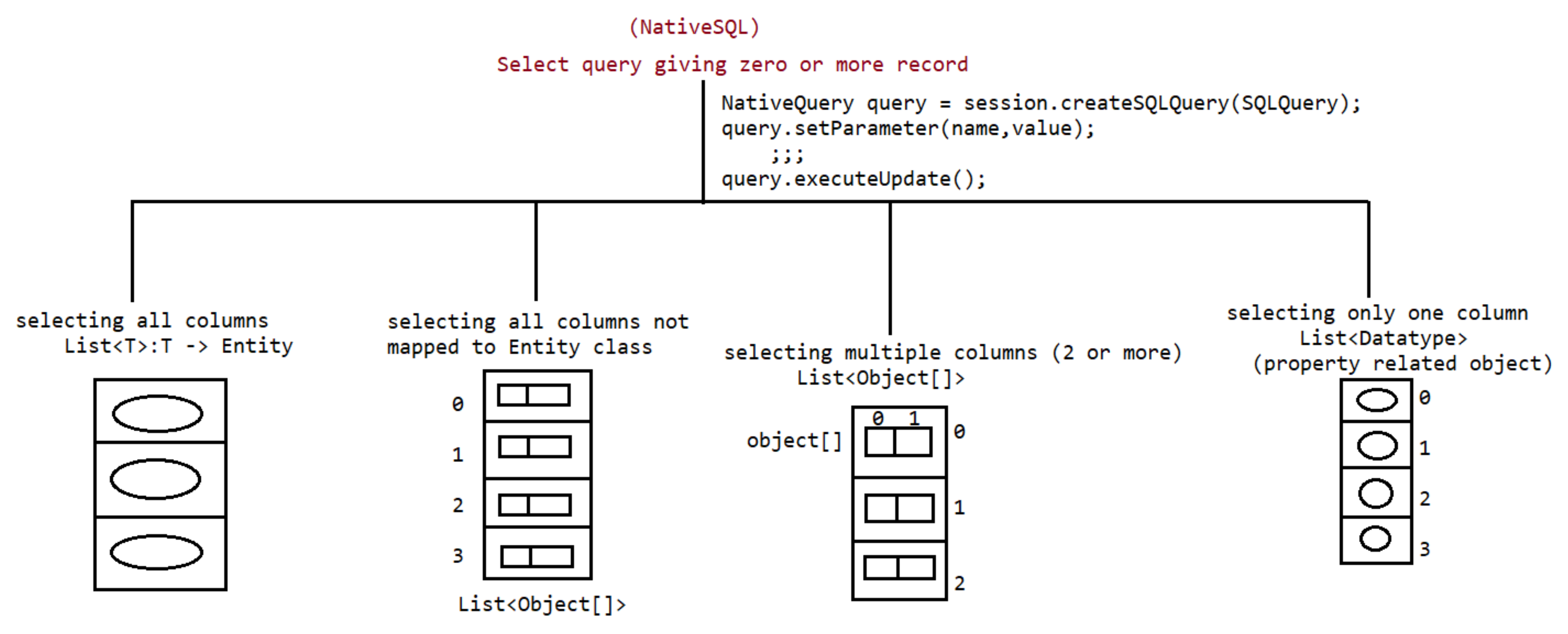
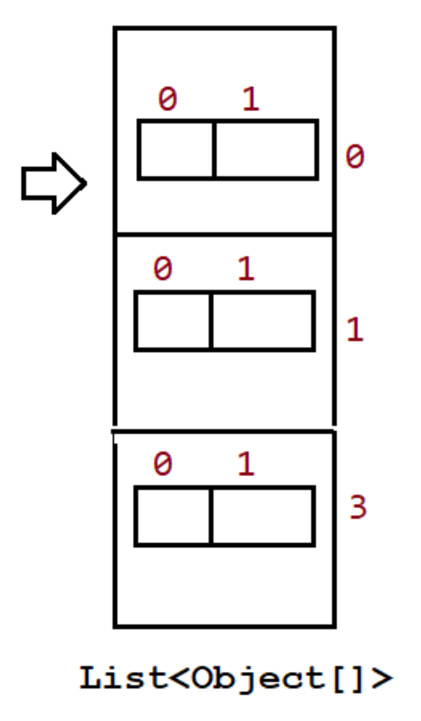

Select query giving zero or more record
list()
selecting all columns
List<T>:T -> Entity
selecting multiple columns (2 or more)
List<Object[]>
object[]
0 1
0
1
2
selecting only one column
List<Datatype>
(property related object)
0
1
2
3

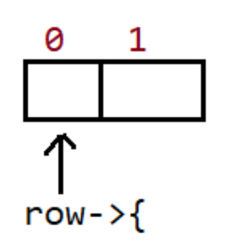

(NativeSQL)
Select query giving zero or more record
selecting all columns
List<T>:T -> Entity
selecting all columns not
mapped to Entity class
0
1
2
3
List<Object[]>
selecting multiple columns (2 or more)
List<Object[]>
object[]
0 1
0
1
2
selecting only one column
List<Datatype>
(property related object)
0
1
2
3

```
NativeQuery query = session.createSQLQuery(SQLQuery);
query.setParameter(name,value);
    ;;;
query.executeUpdate();
```

products

pid	pname	price	qty
1	fossil	15000	2
2	tissot	35000	3
3	omegha	55000	2
4	titan	8000	5





```
products.forEach(
                   row->{
                        for(Object obj: row)
                             System.out.print(obj+"\t");
                        System.out.println();

                      }
                );
```

Bulk operation in hibernate

To select or mainpulate one/more record or object having our choice as criteria value, we need to go for this bulk operation concept

      a. HQL/JPQL
      b. Native SQL
      c. Criteria API

Note: JPQL=> It is a specification given by SUNMS which speaks about the rules to develop object based query Language
        HQL=> It is an implementation of JPQL by hibernate.

1. HQL [Hibernate Query Language]
=> HQL is a pwerfull query language provided by Hibernate inorder to perform manipulations over multiple records.
=> HQL is an object oriented query language, it able to support for the object oriented featureslike encapsulation, polymorphism,.... ,
   but, SQL is structered query language.
=> HQL is database independent query language, but, SQL is database dependent query language.
=> In case of HQL, we will prepare queries by using POJO class names and their properties, but, in case of SQL , we will prepare queries
   on the basis of database table names and table columns.
=> HQL queries are prepare by using the syntaxes which are similar to SQL queries syntaxes.
=> HQL is mainly for retrival operations , but, right from Hibernate3.x version we can use HQL to perform insert , update and delete operations
   along with select operations, but, SQL is able to allow any type of database operation.
=> In case of JDBC, in case of SQL, if we execute select sql query then records are retrived from database table and these records are stored in
   the form of ResultSet object, which is not implementing java.io.Serializable , so that, it is not possible to transfer in the network,
   but, in the case of HQL, if we retrive records then that records will be stored in Collection objects, which are Serializable bydefault, so
   that, we are able to carry these objects in the network.
=> HQL is database independent query language, but, SQL is database dependent query language.
=> In case of Hibernate applications, if we process any HQL query then Hibernate Software will convert that HQL Query into database
   dependent SQL Query and Hibernate software will execute that generated SQL query.

Note: HQL is not suitable where we want to execute Database dependent sql queries
EX: PL/SQL procedures and functions are totally database dependent, where we are unable to use HQl queries.

Note:

```
eg: SQL> SELECT * FROM EMP WHERE EMPNO>? AND EMPNO<?
    HQL> FROM in.ineuron.model.Employee where eno>? and eno<?

    SQL> DELETE FROM EMP WHERE EMPNO=?
    HQL> DELETE FROM in.ineuron.model.Employee where eno=?

    SQL>SELECT ENO,ENAME FROM EMPLOYEE
    HQL>SELECT eno,ename from in.ineuron.model.Employee
    SQL>UPDATE EMPLOYEE SET ENAME=?,ESAL=? WHERE ENO=?
    HQL>UPDATE in.ineuron.model.Employee SET ename=?,esal=? Where eno=?
```

HQL SELECT: we can use select queries to fetch data DB tables (Multiple rows).
Final output is given by Hibernate is java.util.List(no.of rows=no.of objects ----->stored in List Collection only)

FULL LOADING :-select all columns using Query(HQL/SQL) is known as Full loading
(One Full row = One Complete Model class Object).
So final output will be List<T>, T=Type/Model-Class-Name

PARTIAL LOADING : selecting one column (=1 column) or more then one columns (>1 column) is known as Partial loading.
Final outoput is given as:=

```
      1 column  : List<DT DT=DataType of varivales/colum
      >1 column : List<Object[]>
```

Selecting all records

```
Session session = sessionFactory.openSession();
Query<Employee> query = session.createQuery("from in.ineuron.Employee");
System.out.println("Using list() method");
System.out.println("------------------------");
List<Employee> list = query.list();
System.out.println("ENO\tENAME\tESAL\tEADDR");
System.out.println("-------------------------------");
for(Employee e: list) {
    System.out.print(e.getEno()+"\t");
    System.out.print(e.getEname()+"\t");
    System.out.print(e.getEsal()+"\t");
    System.out.println(e.getEaddr());
}
```

Selecting only one record

```
try(Session ses=HibernateUtil.getSf().openSession()){
      String hql="select ename from in.ineuron.model.Employee";
      Query q=ses.createQuery(hql);
      List<String>list=q.list();
      for(String s:list){
            System.out.println(s);
      }
}catch(Exception ex){}
```

Selecting multiple record

```
try(Session ses=HibernateUtil.getSf().openSession()) {
      String hql="select ename,eaddr from in.ineuron.model.Employee";
      Query q=ses.createQuery(hql);
      List<Object[]>list=q.list();
      for(Object[]ob:list){
            System.out.println(ob[0]+","+ob[1]);
      }
}catch(Exception ex){}
```

```
                      or
      list.forEach(row-> {
                                    for(Object obj: row)
                                          System.out.print(obj+":");
                          }
                                    System.out.println();
                  )
```

Named Parameters:- it is used to provide a name in place of ? symbole, to indicate data comes at runtime.
* This is new concept in Hibernate, not exist in JDBC
* Name should be unique in HQL (duplicates not allowed)
**** We can use variableName as parameter name also.
• Name never gets changed if query is changed.
• To pass data at runtime code is : setParameter(name,data)
• Syntax is : name (colon name)

Usage of Named Parameter

```
try(Session ses=HibernateUtil.getSf().openSession()) {
            String hql="from in.ineuron.model.Employee where eid=:id or 
ename=:name";
            Query q=ses.createQuery(hql);

            q.setParameter("id",10);
            q.setParameter("name","sachin");
            List<Employee>list=q.list();
            list.forEach(System.out::println);
}catch(Exception ex){}
```

In clause:- To work with random rows in DB table use in-clause.
Syntax:

```
                  Select ....from ... Where column in (values);
```

• To handle this in Hibernate
• Used named parameters
• Create values collection
• Call setParameterList method

```
============code=========
Test class:// cfg,sf,ses
String hql=”from in.ineuron.model.Employee where empId in (:id)”;
Query q=ses.createQuery(hql);
List<Integer> al=Arrays.asList(10,12,14,8);
q.setParameteList(“id”,al);
List<Employee> e=q.list();
e.forEach(System.out::println);
```

uniqueResult():-
This method is used for select HQL operation.
If query returns one row data then choose this method instead of query.list() method.
      • if will save memory, by avoiding list object for one row data.

Program to demonstrate uniqueResult()

```
-------------------------------------
try(Session ses=HibernateUtil.getSf().openSession()) {
            String hql="from in.ineuron.model.Employee where eid=:id";
            Query q=ses.createQuery(hql);
            q.setParameter("id",10);
            Employee employee=(Employee)query.uniqueResult();
            if(employee!=null)
                  System.out.println(employee);
            else
                  System.out.println("Record not found for the given id :: "+id);
}catch(Exception ex){}
```

Not recomended from JDK8.0,because we have a new API called "Optional(I)".
      Optional(C)-> This api is very useful to hold the object, where it would check the availablity of the object without explicitly performing
                   NullPointerException.

Program to demonstrate uniqueResultOptional()

```
try(Session ses=HibernateUtil.getSf().openSession()) {
            String hql="from in.ineuron.model.Employee where eid=:id";
            Query q=ses.createQuery(hql);
            q.setParameter("id",10);
            Optional<Employee> opt=(Employee)query.uniqueResultOptional();
            if(opt.isPresent())
                  System.out.println(opt.get());
            else
                  System.out.println("Record not found for the given id :: "+id);
}catch(Exception ex){}
```

                                    refer:: HB-25-HQLSelectAPP

Executing HQLNon Select Queries

HQL NON=SELECT OPERATION:- HQL supports non-select operations like
a. Update multiple rows
b. Delete multiple rows
c. Insert operation[Copy rows from one table to another table (backup data)]
- Use method executeUpdate():int return no.of rows effected
- It supports named parameters.

Note: HQL insert query is not given directly, because linking generators with HQL insert query is not possible.
       To link with generators we need to use session.save() method only.

Code for update operation

```
Transaction tx = null;
try (Session ses = HibernateUtil.getSf().openSession()) {
            tx=ses.beginTransaction();
            Employee emp=new Employee();
            String hql="update in.ineuron.model.Employee set ename=:name,esal=:sal 
where eid=:id";

            Query q=ses.createQuery(hql);
            q.setParameter("name", "sachin");
            q.setParameter("sal", 3345);
            q.setParameter("id", 10);
            int count=q.executeUpdate();
            tx.commit();
            System.out.println(count);
} catch (Exception ex) {
      ex.printStackTrace();
}
```

Code for Delete opearation

```
Transaction tx = null;
try (Session ses = HibernateUtil.getSf().openSession()) {
            tx=ses.beginTransaction();
            Employee emp=new Employee();
            String hql="delete in.ineuron.model.Employee where eid=:id";

            Query q=ses.createQuery(hql);
            q.setParameter("id", 10);
            int count= q.executeUpdate();
            tx.commit();
            System.out.println(count);
} catch (Exception ex) {
            ex.printStackTrace();
}
```

refer:: HB-26-HQLNonSelectApp

Code for Insert operation

```
Query query = session.createQuery("insert into
in.ineuron.model.PremiumInsurancePolicy(policyId,policyName,company,policyType,tenu
re)
                                             select
i.policyId,i.policyName,i.company,i.policyType,i.tenure from InsurancePolicy as i
where
      i.tenure>=:min");
Transaction tx = session.beginTransaction();
query.setParameter("min",5);
int rowCount = query.executeUpdate();
tx.commit();
```

NamedHQL Queries

=> So far Our HQL Query is specific to one Session Object becoz Query object created having hard coded HQL Query on session object.
=> To make our HQL query accessible and executable through multiple session objects of multiple DAO classes or client apps we need to go
   for "NamedHQL".
=> we defined HQL query in Entity class using
@NamedQuery(name="HQL_INSERT_QUERY",query="....")

```
@Entity
@NamedQuery(name="HQL_TRANSFER_POLICIES",query="insert into


in.ineuron.model.PremiumInsurancePolicy(policyId,policyName,company,policyType,tenu
re)                                                                          select
i.policyId,i.policyName,i.company,i.policyType,i.tenure from InsurancePolicy
                                                                        as i where
i.tenure>=:min"")
public class PremiumInsurancePolicy implements Serializable
{
      private Long pid;
      private String policyName;
      private String policyType;
      private String company;
      private Integer tenure;
}

Query query= session.getNamedQuery("HQL_TRANSFER_POLICIES");
query.setParameter("min",10);
int rowCount = query.executeUpdate();
```

refer:: HB-27-HQLInsertQuery