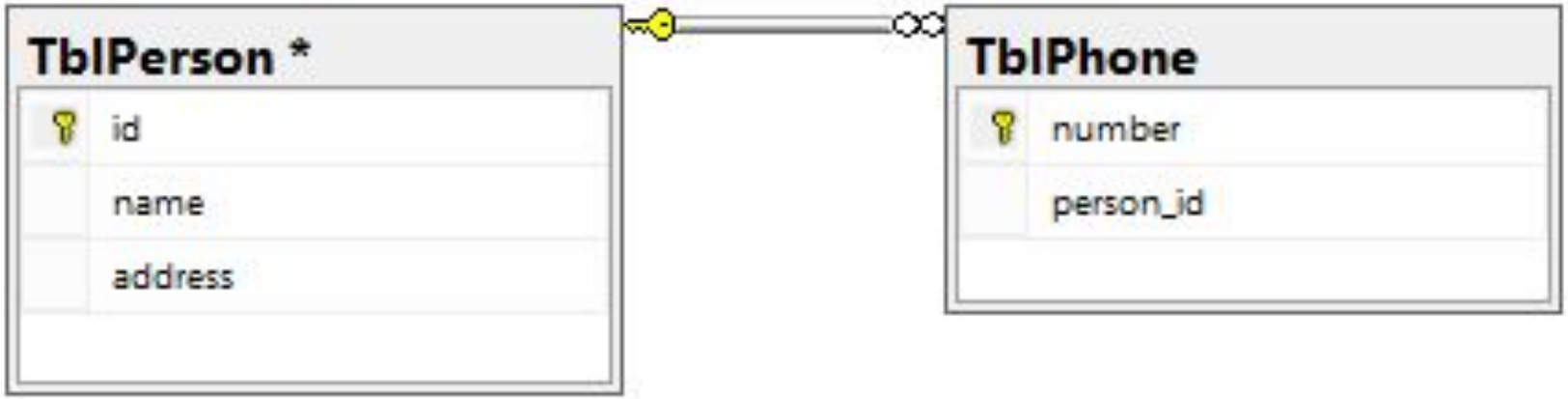
Consultas con Hibernate

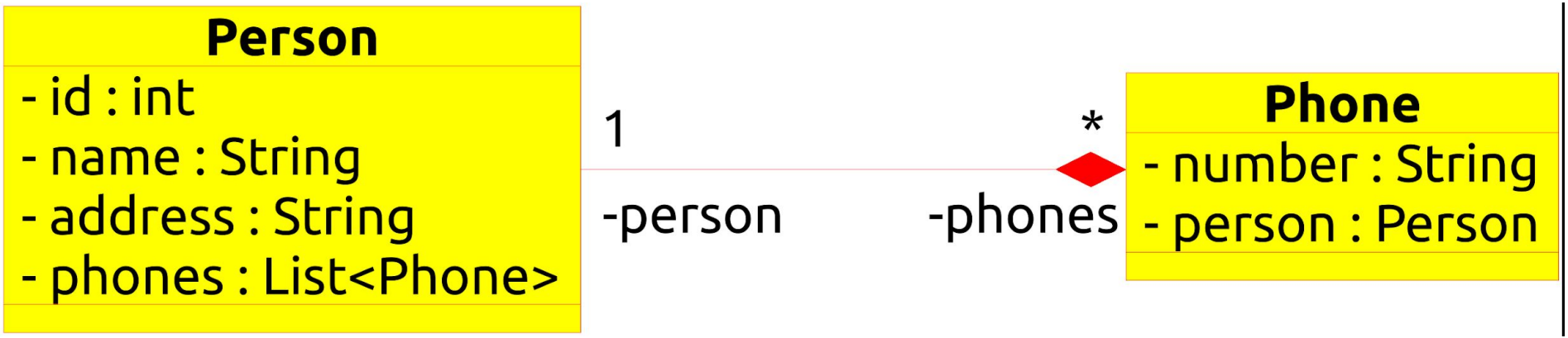
Bases de Datos - 38210
Master Oficial en Desarrollo de Aplicaciones y Servicios Web

Miquel Esplà Gomis
Armando Suarez Cueto

Base de datos de ejemplo: ER

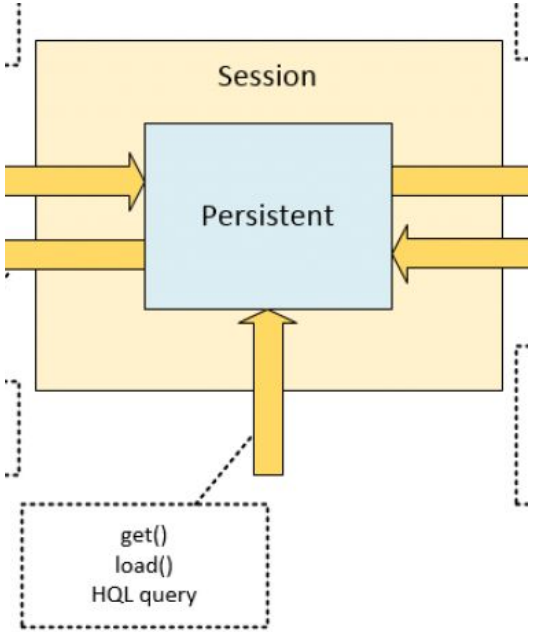


Base de datos de ejemplo: OO



Carga de datos directamente desde la BDR

```
int person_id = 1;
Person person =
    session.get(Person.class, person_id);
```



Diferentes formas de realizar consultas

- **SQL**: consultas en SQL
- **HQL/JPQL**: lenguaje nativo de Hibernate para realizar consultas
- **Criteria**: herramienta para realizar consultas programáticas

Consultas con SQL

Enlace al manual

Consultas SQL en Hibernate

- El método `createNativeQuery()` crea la consulta
- El método `list()` la ejecuta y devuelve una lista de arrays de objetos (`List<Object[]>`)
    - cada elemento de la lista es un ítem de resultados
    - cada objeto del array corresponde a una columna de la tabla donde se ha aplicado la consulta
- El método `uniqueResult()` devuelve un sólo resultado `Object[]`

Consultas SQL en Hibernate

- Hibernate permite ejecutar consultas directamente en SLQ a través del objeto `Session` :

```
Session session = HibernateUtil.getSessionFactory().getCurrentSession();

List<Object[]> persons = session.createNativeQuery(
    "SELECT id, name FROM TblPerson" ).list();

for(Object[] person : persons) {
    Int id = (Integer) person[0];
    String name = (String) person[1];
}
```

Consultas SQL en Hibernate

Cuando hacemos consultas que devuelven entidades, se puede especificar la clase:

```
List<Person> persons = session.createNativeQuery(
    "SELECT * FROM TblPerson", Person.class ).list();
```

Esto no funciona con tipos de datos que no son entidades:

```
List<Integer> ids = session.createNativeQuery(
    "SELECT id FROM TblPerson", Integer.class ).list();
```

Parámetros en las consultas

- Es posible definir consultas con parámetros (más flexible)
- Se puede asignar valor a los parámetros de dos formas:
    - por nombre
    - por posición

Parámetros en las consultas

- Es posible definir consultas con parámetros (más flexible)

  Se puede asignar valor a los parámetros de dos formas:

  - **por nombre**
  - por posición

```
List<Person> persons = session.createNativeQuery(
    "SELECT * FROM TblPerson" +
    "WHERE name like :vname and id > :vid" , Person.class )
.setParameter( "vname", "J%" )
.setParameter( "vid", 5 )
.list()
```

Parámetros en las consultas

- Es posible definir consultas con parámetros (más flexible)
- Se puede asignar valor a los parámetros de dos formas:
  - por nombre
  - **por posición**

```
List<Person> persons = session.createNativeQuery(
    "SELECT * FROM TblPerson" +
    "WHERE name like ? and id > ?" , Person.class )
.setParameter( 1, "J%" )
.setParameter( 2, 5 )
.list()
```

Consultas con HQL

Enlace al manual

HQL/JPQL

- **HQL**: lenguaje de consultas de Hibernate muy similar a SQL pero las consultas se realizan sobre entidades y no tablas
- **JPQL**: lenguaje de consultas de JPA; es un subconjunto de HQL
- **HQL** es independiente del sistema de BDR utilizado

Consultas en HQL

- Consultas en Hibernate: `org.hibernate.query.Query`
- Se generan a través del objeto `Session`
- HQL no es sensible a mayúsculas/minúsculas (sólo para los nombres de clases y propiedades)
- `Query` tiene métodos para controlar su ejecución, como por ejemplo:

```
query.setTimeout(2)
query.readOnly(True)
...
[más opciones]
```

Ejemplo de consultas HQL

Consulta HQL

```
org.hibernate.query.Query query =
session.createQuery(
    "select p " +
    "from Person p " +
    "where p.name like :name" )
.setParameter( "name", "J%");
```

- `Person` es una clase de Java
- `p.name` hace referencia a un parámetro del objeto Person
- `:name` es un parámetro

Joins implícitos

Ejemplo: recuperar todos los teléfonos de una persona que vive en una dirección concreta

Join explícito en SQL:

```
List<Phone> phones= session.createNativeQuery("SELECT * FROM TblPhone ph ” +
    “JOIN TblPerson pr ON ph.person_id = pr.id WHERE pr.address = :address"
    , Phone.class ).setParameter( "address", address ).list();
```

Join implícito: accediendo a las propiedades de los objetos directamente se pueden expresar *joins* de forma implícita

```
List<Phone> phones = sesion.createQuery(“select ph from Phone ph " +
    "where ph.person.address = :address ", Phone.class )
.setParameter( "address", address ).list();
```

Acceso a colecciones

Se puede utilizar la palabra reservada *in* para hacer referencia a todos los elementos de una colección (también se aplica un join implícito)

```
List<Phone> phones = session.createQuery(
    "select ph from Person pr, " +
    "in (pr.phones) ph
    "where pr.address = :address " )
.setParameter( "address", address )
.list();
```

Subqueries

Con el operador *in* también podemos utilizar sub-consultas.

Join implicito vs. sub-consulta:

```
List<Phone> phones = session.createQuery(“select ph from Phone ph " +
   "where ph.person.address = :address ", Phone.class )
   .setParameter( "address", address ).list();
```

```
List<Phone> phones = session.createQuery(“select ph from Phone ph where " +
   "ph.person.id in (select pr.id from Person pr where pr.address = :address) "
    , Phone.class ).setParameter( "address", address ).list();
```

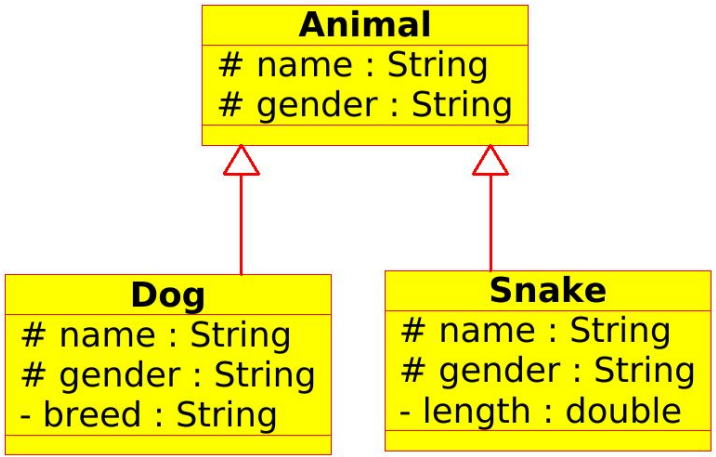
Funciones de agregación

Es posible utilizar funciones de agregación (count, min, max, avg, etc.) de forma muy similar a como se hace en SQL:

```
int sumAge = (Integer) session.createQuery(
    “select SUM(id) from Person pr").uniqueResult();
```

Herencia

Las consultas en HQL gestionan el concepto de herencia intrínsecamente. Supongamos el siguiente caso:



Herencia: Ejemplo

Esta consulta nos devolvería todos los animales (objetos genérico Animal)

```
List<Animal> animals= session.createQuery(
    "select a from Animal a ").list();
```

Todos los animales de la lista contienen también la información específica de su clase heredada, y pueden ser cambiados a ésta en cualquier momento

Polimorfismo: type( )

Se puede verificar el tipo de un objeto en una consulta de HQL. Esto permite discernir entre unas clases heredadas y otras:

```
List<Animal> animals= session.createQuery(
    "select a from Animal a " +
    "where type(a) = ‘Snake’ ").list();
```

Consultas de actualización

Las consultas de actualización en HQL son muy similares a las de SQL:

```
int affectedRows= session.createQuery(
    "delete FROM Animals a WHERE a.name = 'Bear'").executeUpdate();
```

Importante: no se pueden utilizar joins para consultas de actualización

Tipos de consultas

- HQL permite todo tipo de consultas (select, delete, insert y update) con el método `executeUpdate()`
- JPQL no permite inserts
- Sintaxis especificada en la documentación: http://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.html#hql

Consultas con Criteria

Enlace al manual

Interfaz Criteria

- Solución programática para realizar consultas
- A través de la clase `javax.persistence.criteria.CriteriaQuery`
- Prácticamente todas las ventajas de HQL

Interfaz Criteria

- Dos clases de consultas:
  - **Tipadas**: `<T> CriteriaQuery<T> createQuery( Class<T> resultClass )`
  - **Consultas de campos específicos**: `CriteriaQuery<Tuple> createTupleQuery()`

Consultas tipadas

Criteria permite realizar consultas de forma sencilla operando directamente con las clases en Java

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Person> criteria = builder.createQuery( Person.class );
Root<Person> root = criteria.from( Person.class );
criteria.select( root );
criteria.where( builder.equal( root.get( “name” ), "John Doe" ) );

List<Person> persons = session.createQuery( criteria ).list();
```

Consultas de campos específicos

```
CriteriaQuery<Tuple> criteria = builder.createQuery( Tuple.class );
Root<Person> root = criteria.from( Person.class );

Path<Long> idPath = root.get( “id” );
Path<String> namePath = root.get( “name” );

criteria.multiselect( idPath, namePath );
criteria.where( builder.equal( root.get( “name” ), "John Doe" ) );

List<Tuple> tuples = session.createQuery( criteria ).list();

for ( Tuple tuple : tuples )
    Long id = tuple.get( idPath );
```

Join

Se puede realizar *joins* en Criteria definiendo uno de los `JoinType`

```
CriteriaQuery<Tuple> criteria = builder.createQuery( Tuple.class );
Root<Phone> root = criteria.from( Phone.class );
Join<Person, Phone> join_person_phone = root.join(“person”, JoinType.INNER)

Path<String> namePath = join_person_phone.get( "name" );
Path<String> numberPath = join_person_phone.get( "number" );
criteria.multiselect( namePath, numberPath );

List<Tuple> tuples = session.createQuery( criteria ).list();
```

Subqueries

```
Query<Phone> query = builder.createQuery(Phone.class);
Root<Phone> root  = query.from(Phone.class);
Path<Long> idQueryPath = root.get("person").get("id");

Subquery<Integer> subquery = query.subquery(Integer.class);
Root<Person> subroot = subquery.from(Person.class);
subquery.where(builder.equal(subroot.get("name", "John Doe"));

Expression<Integer> idSubquerySelection = subroot.get("id");
subquery.select(idSubquerySelection);

query.where(builder.in(idQueryPath).value(subquery));
List<Person> persons = session.createQuery(query).list();
```

Funciones de agregación

El objeto CriteriaBuilder también nos proporciona funciones de agregación (count, min, max, avg, etc.) para realizar los “select”:

```
CriteriaBuilder builder = session.getCriteriaBuilder();
CriteriaQuery<Tuple> criteria = builder.createQuery( Tuple.class );
Root<Person> root = criteria.from( Person.class );
Path<Long> idPath = root.get( “id” );
criteria.select( builder.sum( idPath );

Tuple persons = session.createQuery( criteria ).uniqueResult();
```

Consultas de actualización

- Como en HQL, es posible realizar consultas de actualización en Criteria
- Se necesita definir el tipo de actualización creando consultas de tipo:
  - `builder.createCriteriaDelete()`
  - `builder.createCriteriaUpdate()`

Named Queries

Named queries

- Es posible definir consultas y darles un nombre para utilizarlas en diferentes puntos del código
- Consultas en SQL: `org.hibernate.annotations.NamedNativeQuery`

```
@Entity
@NamedNativeQueries({
  @NamedNativeQuery(
    name = "find_person_name",
    query = "SELECT name FROM TblPerson "
  );
})
```

```
List<String> names =
    session.getNamedQuery(
    Find_person_name"
).list();
```

Named queries asignados a clases

Una consulta con nombre puede devolver directamente una clase si se define el parámetro *resultClass*

```
@NamedNativeQuery(
    name = "find_phone_by_number",
    query = "SELECT  p.id AS \"id\", p.number AS \"number\", " +
        "p.person AS \"person\", " +
        "FROM TblPhone p " +
        "WHERE p.number LIKE :number",
    resultClass = Phone.class
),
```

Named queries asignados a clases

También se pueden añadir parámetros a las queries que se guardan y definirlos cuando son llamadas

```
List<Person> persons = session.getNamedQuery("find_phone_by_number" )
   .setParameter("number", "687%").list();
```

Named queries HQL

Las consultas de HQL se pueden guardar de la misma forma:

```
@NamedQueries({
    @NamedQuery(
        name = "get_phone_by_number",
        query = "select p from Phone p " +
                "where p.number = :number",
        timeout = 1,
        readOnly = true
    )
})
```