The most basic SQL query is to get a list of scalars (values).

sess.createSQLQuery("SELECT \* FROM CATS").list();

sess.createSQLQuery("SELECT ID, NAME, BIRTHDATE FROM CATS").list();

These will return a List of Object arrays (Object[]) with scalar values for each column in the CATS table. Hibernate will use ResultSetMetadata to deduce the actual order and types of the returned scalar values.

To avoid the overhead of using ResultSetMetadata, or simply to be more explicit in what is returned, one can use addScalar():

为了避免hibernate去使用结果集的元数据

可以显示指定字段的类型

sess.createSQLQuery("SELECT \* FROM CATS")

.addScalar("ID", Hibernate.LONG)

.addScalar("NAME", Hibernate.STRING)

.addScalar("BIRTHDATE", Hibernate.DATE)

sess.createSQLQuery("SELECT \* FROM CATS").addEntity(Cat.class);

sess.createSQLQuery("SELECT ID, NAME, BIRTHDATE FROM CATS").addEntity(Cat.class);

Assuming that Cat is mapped as a class with the columns ID, NAME and BIRTHDATE the above queries will both return a List where each element is a Cat entity.

If the entity is mapped with a many-to-one to another entity it is required to also return this when performing the native query, otherwise a database specific "column not found" error will occur. The additional columns will automatically be returned when using the \* notation, but we prefer to be explicit as in the following example for a many-to-one to a Dog:

sess.createSQLQuery("SELECT ID, NAME, BIRTHDATE, DOG\_ID FROM CATS").addEntity(Cat.class);

This will allow cat.getDog() to function properly.

sess.createSQLQuery("SELECT c.ID, NAME, BIRTHDATE, DOG\_ID, D\_ID, D\_NAME FROM CATS c, DOGS d WHERE c.DOG\_ID = d.D\_ID")

.addEntity("cat", Cat.class)

.addJoin("cat.dog");

sess.createSQLQuery("SELECT ID, NAME, BIRTHDATE, D\_ID, D\_NAME, CAT\_ID FROM CATS c, DOGS d WHERE c.ID = d.CAT\_ID")

.addEntity("cat", Cat.class)

.addJoin("cat.dogs");

### 13.1.4. Returning multiple entities

Until now, the result set column names are assumed to be the same as the column names specified in the mapping document. This can be problematic for SQL queries that join multiple tables, since the same column names can appear in more than one table.

Column alias injection is needed in the following query (which most likely will fail):

sess.createSQLQuery("SELECT c.\*, m.\* FROM CATS c, CATS m WHERE c.MOTHER\_ID = c.ID")

.addEntity("cat", Cat.class)

.addEntity("mother", Cat.class)

The query was intended to return two Cat instances per row: a cat and its mother. The query will, however, fail because there is a conflict of names; the instances are mapped to the same column names. Also, on some databases the returned column aliases will most likely be on the form "c.ID", "c.NAME", etc. which are not equal to the columns specified in the mappings ("ID" and "NAME").

The following form is not vulnerable to column name duplication:

sess.createSQLQuery("SELECT {cat.\*}, {m.\*} FROM CATS c, CATS m WHERE c.MOTHER\_ID = m.ID")

.addEntity("cat", Cat.class)

.addEntity("mother", Cat.class)

This query specified:

* the SQL query string, with placeholders for Hibernate to inject column aliases
* the entities returned by the query

The {cat.\*} and {mother.\*} notation used above is a shorthand for "all properties". Alternatively, you can list the columns explicitly, but even in this case Hibernate injects the SQL column aliases for each property. The placeholder for a column alias is just the property name qualified by the table alias. In the following example, you retrieve Cats and their mothers from a different table (cat\_log) to the one declared in the mapping metadata. You can even use the property aliases in the where clause.

String sql = "SELECT ID as {c.id}, NAME as {c.name}, " +

"BIRTHDATE as {c.birthDate}, MOTHER\_ID as {c.mother}, {mother.\*} " +

"FROM CAT\_LOG c, CAT\_LOG m WHERE {c.mother} = c.ID";

List loggedCats = sess.createSQLQuery(sql)

.addEntity("cat", Cat.class)

.addEntity("mother", Cat.class).list()

sess.createSQLQuery("SELECT NAME, BIRTHDATE FROM CATS")

.setResultTransformer(Transformers.aliasToBean(CatDTO.class))

Query query = sess.createSQLQuery("SELECT \* FROM CATS WHERE NAME like ?").addEntity(Cat.class);

List pusList = query.setString(0, "Pus%").list();

query = sess.createSQLQuery("SELECT \* FROM CATS WHERE NAME like :name").addEntity(Cat.class);

List pusList = query.setString("name", "Pus%").list();