6 Hibernate Querying

Objectives

At the end of this lesson, you will be able to:

- Briefly review SQL querying
- Discover Hibernate's different approaches to querying a database
- Get acquainted with the 'Query by Criteria' technique
- Introduction to Query by Example

Overview

- The easiest way to retrieve data with Hibernate is to use the criteria query API.
- Instead of writing an SQL SELECT statement or an HQL statement, you can build up a query programmatically.

Querying for Data

- **Pull back specific data from a database based on specific requirements**
  - Retrieve a specific object
  - Retrieve a collection of a specific object type
  - Retrieve a collection of different object types
- **In the database, handled through the Structured Query Language (SQL)**
  - SELECT statements for querying

Querying Terminology

- **Restriction**
  - Narrowing down the results (rows) based on specified criteria
  - In SQL, accomplished with the use of conditions appearing in a 'where' clause
  - Example
    - `SELECT * FROM EBILL WHERE EBILL_ID=1;`
    - `SELECT * FROM EBILL WHERE EBILLER_ID='5' AND BALANCE > 1000;`
- **Projection**
  - Narrowing down the data (columns) we return
  - In SQL, accomplished by identifying columns in the 'select' clause
  - Example
    - `SELECT EBILL_ID FROM EBILL WHERE AMOUNT > 1000;`
    - `SELECT EBILL_ID, EBILLER_ID, BALANCE, DUE_DATE FROM EBILL WHERE EBILL_ID=1;`
- **Aggregation**
  - Grouping similar results together based on common attributes
  - Example
    - `SELECT EBILLER_ID, AVG(BALANCE) FROM EBILL GROUP BY EBILLER_ID;`

Querying for data

- SQL statements can be simple...
  - `SELECT * FROM ACCOUNT;`
    - Returns all the records from the account table
  - `SELECT * FROM ACCOUNT WHERE ACCOUNT_ID=1;`
    - Return the row of data for the record with account id 1
- ...or more involved

```
SELECT
     AO.*, A.BALANCE
FROM
     ACCOUNT_OWNER AO,
     ACCOUNT_ACCOUNT_OWNER AAO,
     ACCOUNT A
WHERE

     A.BALANCE   > 500 AND
     A.ACCOUNT_ID = AAO.ACCOUNT_ID AND
     AO.ACCOUNT_OWNER_ID = AAO.ACCOUNT_OWNER_ID;
```

  - Returns a list of all the account owners who have accounts with a balance over $500

SQL Query Options

SELECT <*column(s)*> // projection

FROM <table(s)>

[WHERE] // restriction

<condition(s)>

[ORDER_BY] // ordering

<ordering(s)>

[GROUP BY] // aggregation

<column(s)>

[HAVING] // group restriction

<condition(s)>

Querying with Hibernate

- **Provides multiple interfaces for querying**
  - Query by ID
  - Query by Criteria
  - Query by Example
  - Hibernate Query Language
  - Native SQL calls through Hibernate
- **Relieves the developer of having to know SQL or hand-write complicated queries**
- **Hibernate understands the underlying database and knows how to write more optimized queries**
- **Flexible**
  - If you don't like the way Hibernate tries to retrieve your data, it provides hooks for you to write your own

Query by ID

- **Retrieve an object by its ID**
- **Most common and easiest approach to obtaining objects**
- **Fastest type of query, but can only return a single object at a time**
  - Assuming primary key index on the id column

```
Account account =
  (Account)session.get(
 Account.class,accountId);
```

Query by Criteria (QBC)

Build a query by defining multiple 'Criterion'

- Specify constraints without direct string manipulations
- May result in less readable code than other methods

Realized through four Hibernate interfaces

- **org.hibernate.Criteria**
  - Base object for QBC, created off the Session,
  - Contains all the restriction/projection/aggregation/order information for a single query
- **org.hibernate.DetachedCriteria**
  - Same as Criteria, but created without the presence a Session
  - Later, attached (like detached objects) to a session and executed
- **org.hibernate.criterion.Criterion**
  - Represents a single restriction for a particular query
  - Think of each Criterion as a 'where' clause addition
- **org.hibernate.criterion.Restrictions**
  - Utility class used to create Criterion objects

org.hibernate.Criteria

Create by passing the session a 'root' class

- session.createCriteria(Class);
- session.createCriteria(String className);

Add restrictions

- addCriterion(Criterion criterion);
  - Think 'where' clause

Join to an association, assigning the association an alias

- createAlias(String associationPath, String alias);
  - Still thinking 'where' clause

Add order

- addOrder(Order order);
  - Think 'order by'

org.hibernate.Criteria

- **Get results of the query as a List of root objects**
  - list();
- **Get result of the query as a single root object**
  - uniqueResult();

org.hibernate.criterion.Restrictions

- lt(String propertyName, String value);
- gt(String propertyName, String value);
- eq(String properyName, String value);
- ne(String propertyName, String value);
- like(String propertyName, String value);
- isEmpty(String propertyName, String value);
- isNotEmpty(String propertyName, String value);
- isNull(String propertyName, String value);
- isNotNull(String propertyName, String value);
- in(String propertyName, Collection values);
- allEq(Map propertyNameValues);
- between(String propertyName, Object lo, Object hi);

Hibernate Criterion Interfaces

- **org.hibernate.criterion.Property**
  - Way to represent query restrictions
  - Contains many similar methods as the 'Restrictions' class
- **org.hibernate.criterion.Order**
  - Data container used to represent an ordering scheme
  - Single Criteria object can contain multiple orders
  - Created through static method, passing in attribute to sort on
- **org.hibernate.criterion.Projections**
  - Hibernate utility class to help identify returned columns
  - Created through static methods on either the Projections of Property classes

Query Building

Retrieve List of Objects

```
// retrieves list of all Savings Account Objects
Criteria criteria =
     session.createCriteria(SavingsAccount.class);
List savingAccounts = criteria.list();
```

Root Entity: SavingsAccount

```
// order savings accounts by balance, ascending
Criteria criteria =
     session.createCriteria(SavingsAccount.class)
     .addOrder(Order.asc("balance"));
List savingAccounts = criteria.list();
```

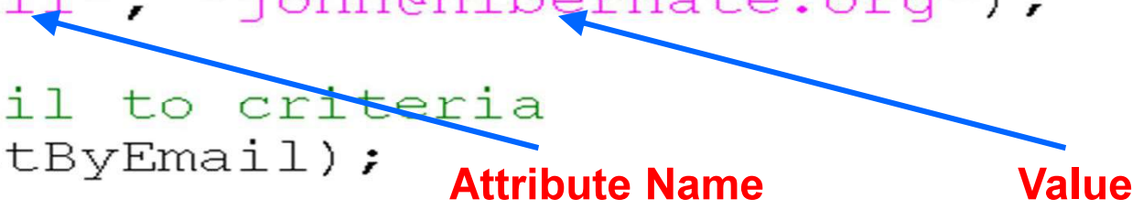
Attribute Name

```
// polymorphic query – returns list of ALL accounts
Criteria criteria =
   session.createCriteria(Account.class);
List accounts = criteria.list();
```

Root Entity: Account

Retrieve Single Object

```
// retrieve an AccountOwner...
Criteria criteria =
session.createCriteria(AccountOwner.class);
// ...with a specific email address
Criterion restrictByEmail =
Restrictions.eq("email", "john@hibernate.org");

// add restrictByEmail to criteria
criteria add(restrictByEmail);

criteria.// ...and we know there can be only one
AccountOwner marty =
(AccountOwner) criteria.uniqueResult();

// same commands - shortened to one line of code
AccountOwner john = (AccountOwner)session
.createCriteria(AccountOwner.class)
.add(Restrictions.eq("email","john@hibernate.org"))
.uniqueResult();
```



Creating Criterion:

```
// shortened version from previous slide,
// using Restrictions
AccountOwner marty = (AccountOwner)session
   .createCriteria(AccountOwner.class)
   .add(

      Restrictions.eq("email","john@hibernate.org");
   )
   .uniqueResult();
```

```
// could also have written using the Property class
AccountOwner marty = (AccountOwner)session
   .createCriteria(AccountOwner.class)
   .add(
      Property.forName("email"). Eq(", john@hibernate.org");
)
.uniqueResult();
```

Using Dot Notation on Objects

```
// can use dot notation to access component attributes
List accounts = session
   .createCriteria(AccountOwner.class)
   .add(Restrictions.eq("address.zipCode","21045"))
   .list();
```

```
// also valid with Property object
List accounts = session
  .createCriteria(AccountOwner.class)
  .add(Property.forName("address.zipCode").eq("21045"))
  .list();
```

'Between' and 'In' Clauses

```
// using 'between'
List accounts =
   session.createCriteria(Account.class)
   .add(Restrictions.between("balance",100, 500))
   .list();
```

```
// 'in' clause – can use Object[] or Collection
long[] accountIds = {1L, 2L, 3L};
List accounts =
  session.createCriteria(Account.class)
  .add(Property.forName("accountId").in(accountIds))
  .list();
```

String Matching (like clause)

```
// String matching
List accountOwners =
   session.createCriteria(AccountOwner.class)
   .add(Restrictions.like("lastName","H%").ignoreCase())
   .list();
```

```
// Alternate String matching.
// MatchMode.END and MatchMode.EXACT also available
List accountOwners =
     session.createCriteria(AccountOwner.class)
     .add(Property.forName("lastName")
          .like("H", MatchMode.START).ignoreCase())
     .list();
```

And'ing Multiple Criterion

```
// adding multiple criteria results in an 'and' operation
List accountOwners =
   session.createCriteria(AccountOwner.class)
   .add(Restrictions.ilike("lastName","H%"))
   .add(Restrictions.ilike("firstName","M%"))
   .list();
```

Can also use 'ilike' for case insensitive search when using Restrictions

```
// works with Properties also
List accountOwners =
      session.createCriteria(AccountOwner.class)
      .add(Property.forName("lastName").like("H%"))
      .add(Property.forName("firstName").like("M%"))
      .list();
```

Or'ing Multiple Criterion

// 'or' operations are a bit trickier

// option 1 – use Restrictions.or and Restrictions.and

// retrieve accounts where:

// 1) last name starts with 'H' and first name with 'M'

// OR

// 2) email address ends in 'john@hibernate.org'

```
List accountOwners =
   session.createCriteria(AccountOwner.class)
   .add(Restrictions.or(
     Restrictions.and(
        Restrictions.like("lastName","H%"),
        Restrictions.like("firstName","M%")
     ),
        Restrictions.like("email"," john@hibernate.org");
   ))
   .list()
```

Creating Detached Criteria Queries

```
DetachedCriteria detachedCriteria =
   DetachedCriteria.forClass(AccountOwner.class)
  .add(Restrictions.ilike("lastName","H%"));

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

Criteria regularCriteria =
 detachedCriteria.getExecutableCriteria(session);

List accountOwners = regularCriteria.list();
```

```
// can also be combined into one line
List accountOwners =
   detachedCriteria.getExecutableCriteria(session)
   .list();`
```

Hibernate Projections

- **Restricting returned columns**
  - Only bring back objects/columns you want to see
- **org.hibernate.criterion.Projections**
  - Created off of either the Projections of Property classes

    Projections.property()

    Property.forName ()
- **Can map results to a non-managed Java class**

Hibernate Projections

```
// retrieve id, balance and creation date
// and set them on a non-managed summary dto
List summaryList =
   session.createCriteria(Account.class)
   .setProjection(Projections.projectionList()
   .add(Projections.id().as("accountId"))
   .add(Projections.property("creationDate"))
   .add(Projections.property("balance")))
   .setResultTransformer(
      new AliasToBeanResultTransformer(
        AccountSummaryDTO.class)
  );
```

Hibernate Projections

```
// can also be coded using Property class
// instead of Projections class
List summaryList =
   session.createCriteria(Account.class)
   .setProjection(Projections.projectionList()
   .add(Property.forName("id").as("accountId"))
   .add(Property.forName("creationDate"))
   .add(Property.forName("balance")))
   .setResultTransformer(
      new AliasToBeanResultTransformer(
        AccountSummaryDTO.class)
 );
```

Counting and Grouping

```
// Returns a collection of Objects[] with 4 fields
// 1) Account id
// 2) A single EBiller id on that account
// 3) The number of ebills received by said ebiller
// 4) Average amount of bills for said ebiller
session.createCriteria(Account.class)
   .createAlias("ebills", "e")
   .setProjection(Projections.projectionList()
      .add(Property.forName("accountId")
         .group())
      .add(Property.forName("e.ebiller.ebillerId")
         .group())
      .add(Property.forName("e.ebiller.ebillerId")
         .count())
      .add(Property.forName("e.balance")
         .avg())
   );
```

Query by Example (QBE)

- **Set up example object(s) for Hibernate to use to generate a database query**
  - Create ‘Example’ criterion based on these objects
- **org.hibernate.criterion.Example**
  - Create by using static create(Object obj);
  - Creates an Example object used for querying for the supplied object type

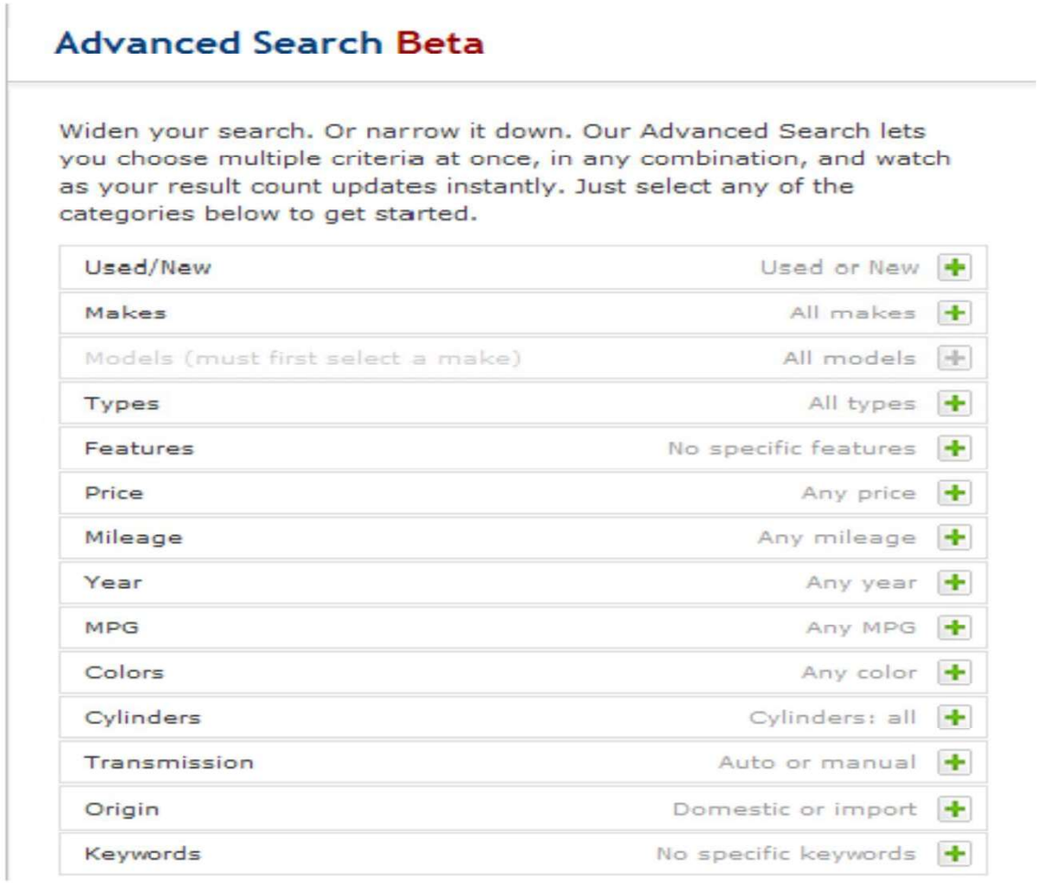
Query by Example

```
// setup an owner with last name 'Hall'
AccountOwner owner = new AccountOwner();
owner.setLastName("Hall");

// create an 'example' criterion based on user
Example exampleOwner =
   Example.create(owner);

// use the criterion to execute the
// query for owners named 'Hall'
List hallList =
   session.createCriteria(AccountOwner.class)
   .add(exampleOwner)
   .list();
```

Great Place for QBE

Applications with GUI pages that allow users to select multiple attributes as search criteria

Example: Carmax.com



Summary

In this lesson, we have learned

- Briefly review SQL querying
- Discover Hibernate's different approaches to querying a database
- Get acquainted with the 'Query by Criteria' technique