Hibernate Querying

Originals of Slides and Source Code for Examples:  
http://courses.coreservlets.com/Course-Materials/hibernate.html

Customized Java EE Training: http://courses.coreservlets.com/  
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.  
Developed and taught by well-known author and developer. At public venues or onsite at your location.

For live Spring & Hibernate training, see courses at http://courses.coreservlets.com/.

Taught by the experts that brought you this tutorial.  
Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall  
  - Java 5, Java 6, intermediate/beginning servlets/JSP, advanced servlets/JSP, Struts, JSF, Ajax, GWT, custom mix of topics  
- Courses developed and taught by coreservlets.com experts (edited by Marty)  
  - Spring, Hibernate/JPA, EJB3, Ruby/Rails

Contact hall@coreservlets.com for details
Topics in This Section

• Briefly review SQL querying
• Discover Hibernate’s different approaches to querying a database
• Get acquainted with the ‘Query by Criteria’ technique
• Introduce the ‘Query by Example’ offshoot
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
    • \( \text{SELECT} \ast \text{FROM EBILL WHERE EBILL ID}=1; \)
    • \( \text{SELECT} \ast \text{FROM EBILL WHERE EBILL ID}=5 \text{\' AND BALANCE}>1000; \)

• Projection
  – Narrowing down the data (columns) we return
  – In SQL, accomplished by identifying columns in the ‘select’ clause
  – Example
    • \( \text{SELECT} \text{EBILL ID FROM EBILL WHERE AMOUNT}>1000; \)
    • \( \text{SELECT} \text{EBILL ID, EBILLER ID, BALANCE, DUE DATE FROM EBILL WHERE EBILL ID}=1; \)

• Aggregation
  – Grouping similar results together based on common attributes
  – Example
    • \( \text{SELECT EBILLER ID, AVG(BALANCE) FROM EBILL GROUP BY EBILLER ID}; \)
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)> // restriction
[ WHERE ]
  <condition(s)> // ordering
[ ORDER_BY ]
  <ordering(s)> // aggregation
[ GROUP BY ]
  <column(s)> // group restriction
[ HAVING ]
  <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
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();

ebl C i i ( S i )

org.hibernate.DetachedCriteria

- No session required to create
  - Created by calling static method, providing a ‘root’ class
    -DetachedCriteria.forClass(Class);
    -DetachedCriteria.forClass(String className);
- Contains most of the methods from the Criteria class
  - Does not contain the ‘execute’ methods
    - list(), uniqueResult() etc...
- To execute, create regular Criteria instance off of detached version
  - getExecutableCriteria(Session);
org.hibernate.criterion.Restrictions

- `lt(String propertyName, String value);`
- `gt(String propertyName, String value);`
- `eq(String propertyName, 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();

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

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

```java
// retrieve an AccountOwner...
Criteria criteria =
    session.createCriteria(AccountOwner.class);

// ...with a specific email address
Criterion restrictByEmail =
    Restrictions.eq("email", "hall@coreservlets");

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

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

---

Creating Criterion: Restrictions and Property

```java
// shortened version from previous slide,
// using Restrictions
AccountOwner marty = (AccountOwner) session
    .createCriteria(AccountOwner.class)
    .add(Restrictions.eq("email","hall@coreservlets"))
    .uniqueResult();
```

---

// could also have written using the Property class
```java
AccountOwner marty = (AccountOwner) session
    .createCriteria(AccountOwner.class)
    .add(
        Property.forName("email").eq("hall@coreservlets.com")
    )
    .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();

// 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 using Or/And

// ‘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 ‘coreservlets.com’
List accountOwners =
  session.createCriteria(AccountOwner.class)
  .add(Restrictions.or(
    Restrictions.and(
      Restrictions.like("lastName","H"),
      Restrictions.like("firstName","M")
    ),
    Restrictions.like("email", "%coreservlets.com")
  ))
  .list()

Or’ing Multiple Criterion using Disjunction/Conjunction

// ‘or’ operations are a bit trickier
// option 2 – use Restrictions.conjunction and
// Restrictions.disjunction

// retrieve accounts where:
// 1) last name starts with ‘H’ and first name with ‘M’
// OR
// 2) email address ends in ‘coreservlets.com’
List accountOwners =
  session.createCriteria(AccountOwner.class)
  .add(Restrictions.disjunction()
    .add(Restrictions.conjunction()
      .add(Restrictions.like("lastName","H"))
      .add(Restrictions.like("firstName", "M"))
    )
    .add(Restrictions.like("email","%coreservlets.com"))
  )
  .list()
Adding Your Own SQL – Using Database Functions

// use the sql 'length' function to return account owners 
// with last names greater then 10 characters
List accountOwners =
    session.createCriteria(AccountOwner.class).add(
        Restrictions.sqlRestriction(
            "length({alias}.LAST_NAME) > ?",
            10,
            Hibernate.INTEGER
        )
    ).list();

Need to put an 'alias' place holder for 
the table name, which will come from 
the root entity, AccountOwner

Adding Your Own SQL – Joining Tables

// retrieve accounts where all the transactions made 
// by the account have been less than $500
List accounts =
    session.createCriteria(Account.class).add(
        Restrictions.sqlRestriction(
            " 500 > all " +
            " (select atx.amount from ACCOUNT_TRANSACTION atx" +
            " where atx.account_id = {alias}.account_id)"
        )
    ).list();

Oracle and Derby evaluate 'all' to 'true' 
if no rows are returned in the subselect statement!

Again, 'alias' place holder for the table name, which will come from the root entity, Account
Creating Detached Criteria Queries

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

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

regularCriteria = detachedCriteria.getExecutableCriteria(session);

List accountOwners = regularCriteria.list();

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

Restricting Data with Joins

```java
// build the EBill criteria
Criteria ebillCriteria = session.createCriteria(EBill.class);

// build the EBiller criteria off the EBill criteria
Criteria ebillerCriteria = ebillCriteria.createCriteria("ebiller");

ebillerCriteria.add(Restrictions.like("name","VISA"));

// return all the EBills issued from VISA
List ebills = ebillCriteria.list();

// can be combined into a single like of code
Criteria ebillCriteria = session.createCriteria(EBill.class)
 .createCriteria("ebiller")
 .add(Restrictions.like("name","VISA"));
```
Restricting Data with Joins – Using an Alias

• Alternate approach using an alias and only creating one Criteria object

```java
// single line approach from previous slide using two Criteria objects
Criteria ebillCriteria = session.createCriteria(EBill.class)
    .createCriteria("ebiller")
    .add(Restrictions.like("name", "VISA"));
```

ebIllers is the name of the association that appears In the root entity’s mapping file

```java
// use an alias, and create a single Criteria object
Criteria ebillCrit = session.createCriteria(EBill.class)
    .createAlias("ebillers", "eb")
    .add(Restrictions.like("eb.name", "VISA"));
```

Criteria Object Joining

• Currently, only joining of associations are supported
  – <one-to-many>
  – <many-to-many>
  – etc...

• Component joining is not supported at this time
  – Receive failure that the property you want to reference does not represent an entity association
  – This feature likely to be added in the near future
Forcing Eager Loading

• Indicate in the Criteria to fully load all objects, and **not** to use proxies
  – If already setup to fully load in the object mapping file (i.e. no lazy="false" etc…), this is unnecessary
  – If referencing an object in a non-SQL restriction, this is also unnecessary. Hibernate will automatically load objects referenced in restrictions

```java
List ebillers =
    session.createCriteria(EBiller.class)
    .setFetchMode("ebills", FetchMode.JOIN)
    .list();
```

Indicates that the "ebills" collection should be fully loaded for each EBiller instance

Result Transformers – Default Behavior

• Transformers map the returned result sets to Java objects
  – By default, maps returned result sets to the Root Entity object

```java
// this...
List accounts =
    session.createCriteria(Account.class)
    .list();
```

Default results transformer maps the result set into the Root Entity indicated in the createCriteria statement

```java
// ...is the same as this
List accounts =
    session.createCriteria(Account.class)
    .setResultsTransformer(Criteria.ROOT_ENTITY)
    .list();
```
Result Transformers – Distinct Entity Results

• DISTINCT_ROOT_ENTITY
  – Eliminates any duplicate objects
  – Similar to the ‘DISTINCT’ SQL keyword

```java
// transformer eliminates any duplicate objects
List accounts = 
    session.createCriteria(Account.class)
    .setResultsTransformer(Criteria.DISTINCT_ROOT_ENTITY)
    .list();
```

Result Transformers – Return Multiple Object Types

• ALIAS_TO_ENTITY_MAP
  – Returns a list of maps, where each map has an instance of all the objects identified in the query (on a per-row basis)

```java
List mapList = 
    session.createCriteria(Account.class)
    .createAlias("accountTransactions", "atx")
    .setResultTransformer(Criteria.ALIAS_TO_ENTITY_MAP)
    .list();

for (Object aResult : result) {
    Map map = (Map) aResult;
    Account account =
        (Account)map.get(Criteria.ROOT_ALIAS);
    AccountTransaction atx =
        (AccountTransaction) map.get("atx");
}
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

// 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")))
  .getResultTransformer(
    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())
    );

• No ‘Having’ feature available at this time
  – Planned for a future release
Query by Example

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.

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

```java
// 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();
```

Setting Up Example Object

- **Certain fields ignored by default**
  - Object Identifiers
  - Version property
  - Associations
  - Any null valued properties

- **Can also specify:**
  - `enableLike(MatchMode mode);`
    - Enable 'like' matching for all String attributes
  - `ignoreCase();`
    - Ignore case for all String attributes
  - `excludeZeroes();`
    - Excludes zero-valued properties
  - `excludeProperty(String name);`
    - Exclude a particular named property
  - `excludeNone();`
    - Do not exclude null or zero-valued properties
Combine Example and Traditional Criterion

Example exampleOwner = Example.create(owner);

// execute the query for owners named 'Hall'
// who also live in the state of Maryland
List hallList =
    session.createCriteria(AccountOwner.class)
        .add(exampleOwner)
        .add(Restrictions.eq("address.state", "MD"))
        .list();

Combine Multiple Example Criterion

// account owners named Hall
AccountOwner owner = new AccountOwner();
owner.setLastName("Hall");
Example exampleOwner = Example.create(owner);

// accounts with a balance of 1000
Account account = new Account();
account.setBalance(1000);
Example exampleAccount = Example.create(account);

// query for owners named 'Hall' with balances of 1000
List hallList =
    session.createCriteria(AccountOwner.class)
        .add(exampleOwner)
        .createCriteria("accounts").add(exampleAccount)
        .list();
Great Place for QBE

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

Example: Carmax.com

Wrap-up

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.
Summary

• In this lecture, we:
  – Reviewed traditional SQL querying and established terms for its capabilities
  – Identified several approaches provided by Hibernate to query a database
  – Walked through Query by Criteria (QBC) and Query by Example (QBE) querying methods, and learned about the core Hibernate classes involved in querying
    • Criteria
    • Criterion
    • Restrictions
    • Property
    • Order
    • Projections

Preview of Next Sections

• Hibernate Query Language
• Calling native SQL through Hibernate
Questions?

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.