Hibernate Overview

Topics

- What is Hibernate?
- Benefits of Hibernate
- Code Snippets

What is Hibernate?

- A framework for persisting / saving Java objects in a database
  - www.hibernate.org

Benefits of Hibernate

- Hibernate handles all of the low-level SQL
- Minimizes the amount of JDBC code you have to develop
- Hibernate provides the Object-to-Relational Mapping (ORM)

Object-To-Relational Mapping (ORM)

- The developer defines mapping between Java class and database table

Java Class

Database Table

Saving a Java Object with Hibernate

```
// create Java object
Student theStudent = new Student(“John”, “Doe”, “john@luv2code.com");

// save it to database
int theId = (Integer) session.save(theStudent);
```

Retrieving a Java Object with Hibernate

```
// create Java object
Student theStudent = new Student(“John”, “Doe”, “john@luv2code.com");

// save it to database
int theId = (Integer) session.save(theStudent);

// now retrieve from database using the primary key
Student myStudent = session.get(Student.class, theId);
```

Querying for Java Objects

```
Query query = session.createQuery("from Student");

List<Student> students= query.list();
```

Hibernate CRUD Apps

- Create objects
- Read objects
- Update objects
- Delete objects

Hibernate is actually more than ORM!

Hibernate Overview

Hibernate and JDBC

How does Hibernate relate to JDBC?

Hibernate and JDBC

- Hibernate uses JDBC for all database communications

SET UP YOUR ENVIRONMENT

Must Have Java Development Kit (JDK)

You Must Have the Java Development Kit (JDK) Installed

1. Check out my YouTube video for this:

   http://www.luv2code.com/install-java

Required Software

To Build Hibernate Applications, you need the following:

1. Java Integrated Development Environment (IDE)
2. Database Server
3. Hibernate JAR files and JDBC Driver

INSTALL ECLIPSE

MS WINDOWS

Install MySQL on MS Windows

Topics

- Download MySQL
- Install MySQL
- Verify Installation

Setup Database Table

Two Database Scripts

1. Folder: **sql-scripts**
   - **01-create-user.sql**
   - **02-student-tracker.sql**

About: 01-create-user.sql

1. Create a new MySQL user for our application
   - user id: **hbstudent**
   - password: **hbstudent**

About: 02-student-tracker.sql

1. Create a new database table: **student**

Setup Hibernate in Eclipse

To Do List

1. Create Eclipse Project

2. Download Hibernate Files

3. Download MySQL JDBC Driver

4. Add JAR files to Eclipse Project … *Build Path*

Test JDBC Connection

Hibernate Dev Process

To Do List

1. Add Hibernate Configuration file

2. Annotate Java Class

3. Develop Java Code to perform database operations

Hibernate Configuration

Hibernate Dev Process - To Do List

1. Add Hibernate Configuration file

2. Annotate Java Class

3. Develop Java Code to perform database operations

Configuration File

Annotate Java Class

Hibernate Dev Process - To Do List

1. Add Hibernate Configuration file

2. Annotate Java Class

3. Develop Java Code to perform database operations

Terminology

Entity Class

Java class that is mapped to a database table

Object-to-Relational Mapping (ORM)

Java Class

Database Table

Two Options for Mapping

- Option 1: XML config file (legacy)
- Option 2: Java Annotations (modern, preferred)

Java Annotations

- Step 1: Map class to database table
- Step 2: Map fields to database columns

Step 1: Map class to database table

```
@Entity
@Table(name="student")
public class Student {

  ...

}
```

Java Class

Database Table

Step 2: Map fields to database columns

```
@Entity
@Table(name="student")
public class Student {

   @Id
   @Column(name="id")
   private int id;

   @Column(name="first_name")
   private String firstName;
  …
}
```

Save a Java Object

Hibernate Dev Process - To Do List

1. Add Hibernate Configuration file

2. Annotate Java Class

3. Develop Java Code to perform database operations

Two Key Players

Class	Description
SessionFactory	Reads the hibernate config file Creates Session objects Heavy-weight object Only create once in your app
Session	Wraps a JDBC connection Main object used to save/retrieve objects Short-lived object Retrieved from SessionFactory

Java Code Setup

```
public static void main(String[] args) {

    SessionFactory factory = new Configuration()
                                .configure("hibernate.cfg.xml")
                                .addAnnotatedClass(Student.class)
                                .buildSessionFactory();

    Session session = factory.getCurrentSession();

    try {

        // now use the session object to save/retrieve Java objects

    } finally {
        factory.close();
    }

}
```

Save a Java Object

```
try {

    // create a student object
    Student tempStudent = new Student("Paul", "Wall", "paul@luv2code.com");

    // start transaction
    session.beginTransaction();

    // save the student
    session.save(tempStudent);

    // commit the transaction
    session.getTransaction().commit();

} finally {
    factory.close();
}
```

Hibernate and Primary Keys

Terminology

Primary Key

Uniquely identifies each row in a table

Must be a unique value

Cannot contain NULL values

MySQL - Auto Increment

```
CREATE TABLE student (

  id int(11) NOT NULL AUTO_INCREMENT,
  first_name varchar(45) DEFAULT NULL,
  last_name varchar(45) DEFAULT NULL,
  email varchar(45) DEFAULT NULL,
  PRIMARY KEY (id)

)
```

Hibernate Identity - Primary Key

```
@Entity
@Table(name="student")
public class Student {

  @Id
  @Column(name="id")
  private int id;


 ...
}
```

Hibernate Identity - Primary Key

```
@Entity
@Table(name="student")
public class Student {

  @Id
 @GeneratedValue(strategy=GenerationType.IDENTITY)
  @Column(name="id")
  private int id;


 ...
}
```

ID Generation Strategies

Name	Description
GenerationType.AUTO	Pick an appropriate strategy for the particular database
GenerationType.IDENTITY	Assign primary keys using database identity column
GenerationType.SEQUENCE	Assign primary keys using a database sequence
GenerationType.TABLE	Assign primary keys using an underlying database table to ensure uniqueness

Bonus Bonus

- You can define your own CUSTOM generation strategy :-)
- Create subclass of **org.hibernate.id.SequenceGenerator**
- Override the method: **public Serializable generate(…)**

Retrieving an Object

Hibernate CRUD Apps

- Create objects
- Read objects
- Update objects
- Delete objects

Retrieving a Java Object with Hibernate

```
// create Java object
Student theStudent = new Student("Daffy", "Duck", "daffy@luv2code.com");

// save it to database
session.save(theStudent);
...

// now retrieve/read from database using the primary key
Student myStudent =
        session.get(Student.class, theStudent.getId());
```

Querying Objects

Hibernate CRUD Apps

- Create objects
- Read objects
- Update objects
- Delete objects

Hibernate Query Language (HQL)

- Query language for retrieving objects
- Similar in nature to SQL
  - where, like, order by, join, in, etc…

Retrieving all Students

```
List<Student> theStudents =
                    session
                        .createQuery("from Student”)
                        .getResultList();
```

Retrieving Students: lastName = ‘Doe’

```
List<Student> theStudents =
        session
          .createQuery("from Student s where s.lastName='Doe'")
          .getResultList();
```

Retrieving Students using OR predicate:

```
List<Student> theStudents =
        session
          .createQuery("from Student s where s.lastName='Doe'"
                  + " OR s.firstName='Daffy'")
          .getResultList();
```

Retrieving Students using LIKE predicate:

```
List<Student> theStudents =
        session
          .createQuery("from Student s where"
                  + " s.email LIKE '%luv2code.com'")
          .getResultList();
```

Updating Object(s)

Hibernate CRUD Apps

- Create objects
- Read objects
- Update objects
- Delete objects

Update a Student

```
int studentId = 1;

Student myStudent = session.get(Student.class, studentId);

// update first name to "Scooby"
myStudent.setFirstName("Scooby");

// commit the transaction
session.getTransaction().commit();
```

Update email for all students

```
session
    .createQuery("update Student set email='foo@gmail.com'")
    .executeUpdate();
```

Deleting Object(s)

Hibernate CRUD Apps

- Create objects
- Read objects
- Update objects
- Delete objects

Delete a Student

```
int studentId = 1;

Student myStudent = session.get(Student.class, studentId);

// delete the student
session.delete(myStudent);

// commit the transaction
session.getTransaction().commit();
```

Another way of deleting

```
session
    .createQuery("delete from Student where id=2")
    .executeUpdate();
```

Hibernate Advanced Mappings

Basic Mapping

Java Class

Database Table

Advanced Mappings

- In the database, you most likely will have
  - Multiple Tables
  - Relationships between Tables
- Need to model this with Hibernate

Advanced Mappings

- One-to-One
- One-to-Many, Many-to-One
- Many-to-Many

One-to-One Mapping

- An instructor can have an "instructor detail" entity
  - Similar to an "instructor profile"

One-to-Many Mapping

- An instructor can have many courses

Many-to-Many Mapping

- A course can have many students
- A student can have many courses

Important Database Concepts

- Primary key and foreign key
- Cascade

Primary Key and Foreign Key

- Primary key: identify a unique row in a table

- Foreign key:
  - Link tables together
  - a field in one table that refers to primary key in another table

Foreign Key Example

Foreign key column

Table: instructor

id	first_name	last_name	instructor_detail_id
1	Chad	Darby	100
2	Madhu	Patel	200

Table: instructor_detail

id	youtube_channel	hobby
100	www.luv2code.com/youtube	Luv 2 Code!!!
200	www.youtube.com	Guitar

Cascade

- You can **cascade** operations

- Apply the same operation to related entities

Cascade

- If we delete an **instructor**, we should also delete their **instructor_detail**
  - This is known as "CASCADE DELETE"

Cascade Delete

Foreign key column

Table: instructor

id	first_name	last_name	instructor_detail_id
~~1~~	~~Chad~~	~~Darby~~	~~100~~
2	Madhu	Patel	200

Table: instructor_detail

id	youtube_channel	hobby
~~100~~	~~www.luv2code.com/youtube~~	~~Luv 2 Code!!!~~
200	www.youtube.com	Guitar

Cascade Delete

- Cascade delete depends on the use case
- Should we do cascade delete here???

Developer can configure cascading

Fetch Types: Eager vs Lazy Loading

- When we fetch / retrieve data, should we retrieve EVERYTHING?
  - **Eager** will retrieve everything
  - **Lazy** will retrieve on request

Uni-Directional

Bi-Directional

Hibernate One-to-One

One-to-One Mapping

- An instructor can have an "instructor detail" entity
  - Similar to an "instructor profile"

Uni-Directional

Development Process: One-to-One

Step-By-Step

1. Prep Work - Define database tables
2. Create InstructorDetail class
3. Create Instructor class
4. Create Main App

table: instructor_detail

File: create-db.sql

```
CREATE TABLE `instructor_detail` (

  `id`              int(11)      NOT NULL AUTO_INCREMENT,
  `youtube_channel` varchar(128) DEFAULT NULL,
  `hobby`           varchar(45)  DEFAULT NULL,


);

...
```

table: instructor_detail

File: create-db.sql

```
CREATE TABLE `instructor_detail` (

  `id`              int(11)      NOT NULL AUTO_INCREMENT,
  `youtube_channel` varchar(128) DEFAULT NULL,
  `hobby`           varchar(45)  DEFAULT NULL,

  PRIMARY KEY (`id`)
);

...
```

table: instructor

File: create-db.sql

```
...

CREATE TABLE `instructor` (

  `id`          int(11) NOT NULL AUTO_INCREMENT,
  `first_name`  varchar(45) DEFAULT NULL,
  `last_name`   varchar(45) DEFAULT NULL,
  `email`       varchar(45) DEFAULT NULL,
  `instructor_detail_id` int(11) DEFAULT NULL,

  PRIMARY KEY (`id`)
    ...
);
```

Foreign Key

- Link tables together
- A field in one table that refers to primary key in another table

Foreign Key Example

Foreign key column

Table: instructor

id	first_name	last_name	instructor_detail_id
1	Chad	Darby	100
2	Madhu	Patel	200

Table: instructor_detail

id	youtube_channel	hobby
100	www.luv2code.com/youtube	Luv 2 Code!!!
200	www.youtube.com	Guitar

Defining Foreign Key

File: create-db.sql

```
...
CREATE TABLE `instructor` (
   ...

   CONSTRAINT `FK_DETAIL` FOREIGN KEY (`instructor_detail_id`)
   REFERENCES `instructor_detail` (`id`)

);
```

More on Foreign Key

- Main purpose is to preserve relationship between tables
  - Referential Integrity
- Prevents operations that would destroy relationship
- Ensures only valid data is inserted into the foreign key column
  - Can only contain valid reference to primary key in other table

Development Process: One-to-One

Step-By-Step

1. Prep Work - Define database tables
2. Create InstructorDetail class
3. Create Instructor class
4. Create Main App

Step 2: Create InstructorDetail class

```
@Entity
@Table(name="instructor_detail")
public class InstructorDetail {

}
```

Step 2: Create InstructorDetail class

```
@Entity
@Table(name="instructor_detail")
public class InstructorDetail {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private int id;



}
```

Step 2: Create InstructorDetail class

```
@Entity
@Table(name="instructor_detail")
public class InstructorDetail {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @Column(name="youtube_channel")
    private String youtubeChannel;



}
```

Step 2: Create InstructorDetail class

```
@Entity
@Table(name="instructor_detail")
public class InstructorDetail {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @Column(name="youtube_channel")
    private String youtubeChannel;

    @Column(name="hobby")
    private String hobby;



}
```

Step 2: Create InstructorDetail class

```
@Entity
@Table(name="instructor_detail")
public class InstructorDetail {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @Column(name="youtube_channel")
    private String youtubeChannel;

    @Column(name="hobby")
    private String hobby;

    // constructors

    // getters / setters
}
```

Step 3: Create Instructor class

```
@Entity
@Table(name="instructor")
public class Instructor {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @Column(name="first_name")
    private String firstName;

    @Column(name="last_name")
    private String lastName;

    @Column(name="email")
    private String email;

    ...
    // constructors, getters / setters
}
```

Step 3: Create Instructor class - @OneToOne

```
@Entity
@Table(name="instructor")
public class Instructor {
    ...

    @OneToOne
    @JoinColumn(name="instructor_detail_id")
    private InstructorDetail instructorDetail;

    ...
    // constructors, getters / setters
}
```

Entity Lifecycle

Operations	Description
Detach	If entity is detached, it is not associated with a Hibernate session
Merge	If instance is detached from session, then merge will reattach to session
Persist	Transitions new instances to managed state. Next flush / commit will save in db.
Remove	Transitions managed entity to be removed. Next flush / commit will delete from db.
Refresh	Reload / synch object with data from db. Prevents stale data

Entity Lifecycle - session method calls

Cascade

- Recall: You can **cascade** operations

- Apply the same operation to related entities