Java Persistence: Hibernate and JPA Fundamentals

Object Persistence & Relational Database

- Persistence → The state of an object can be saved to a date store, and re-created at a later point in time.
- Relational Database → A data store that represent data in a table-like format.
- Relational Database Management Systems
  - A database management system designed to manage data in a relational database
  - Can have many different databases.
- SQL Datatypes
  - Depend on the database management system that you are using.
- Entity Integrity Rule
  - Every table has a primary key
  - Null values are not valid values for a primary key
- Primary key
  - No duplicate value should be allowed in the column.
- Referential Integrality
  - A foreign key point to the value that is the primary key of another table
  - Null values are valid in a foreign key column
    - Null Values are valid in a foreign key column, but if a value exists in a foreign key column then it must refer to a valid reference
-

Object Model and Relational Model

- ORM
  - In an object orientated data is represent as interconnect graph of object
    - Use the principles of abstraction, encapsulation, Modularity, Hierarchy, typing, polymorphism, concurrency, persistence
    - An object has
      - Behavior
      - Identify
      - State
  - In a database the data is represented in a table model
    - The structure of data
    - Data Manipulation
    - Data Integrality
  -

Object Relational Impedance Mismatch

- Loading or storing graphs of objects using a relational database causes mismatch problems.
  - Called Object Relational Impedance Mismatch
  - Granularity is the extent to which a system could be broken down into small parts
    - A coarse grained object consists of various fine grained object or object with finer granularity
    - Objects
      - Various levels of granularity
    - Relational Model
      - 2 levels of granularity (tables and columns)
    - Object Model is more granular then the Relational Model
      - More class in the object then the number of corresponding models in the database
  - Subtype Mismatch
    - Object Model
      - Has Inheritance
    - Relational Model
      - No Inheritance
  - Identity Mismatch
    - Object Model
      - Has two way to determine if an object is the same : ==, .equals
    - Relational Model
      - Primary Key

Object Relational Impedance Mismatch

- Associations
  - Object
    - Object References
    - Are directional
  - Relational Model
    - Foreign key
    - Not directional
- Data Navigation
  - Object Model
    - Navigate from one association to another walking the object graph
  - Relational
    - Worry about efficiency : minimize the number of requests to the database
      - Minimize the number of SQL Queries (write a join query)

Object Relational Mapping

- Problems with using JDBC
  - Need to write SQL Queries, a java programmer may not know sql or how to optimize it correctly
  - Writing to many SQL Statements : For three tables you have three joins
  - Manually handling associations.
  - Writing too many parameters to the database.
  - SQL is both ANSI ,but has some DBMS constructors.
- Solution : Object Relational Mapping refers to the technique of mapping the representation of data from Java Objects to Relational Database
- Allows the user of java objects as a representation of the database.

Mysql Commands

- Mysql Command
  - Mysql -u <root> -p <password>
  - Create Database bookstore;
  - Use bookstore
- HeidiSQL → A database explorer for windows (might to be able to use it in Wine)
-

What is Hibernate and Hello World with JPA and annotations

- Central Idea : Create plain old java object with annotations for CRUD – Allow use to use objects as a representation of data
- Important Classes in Hibernate
  - Configuration Parse a hibernate.cfg.xml file
    - Example
      - <hibernate-configuration>
        - <session-factory>
          - <property name =”connection.driver_class”” >com.mtysql.jdbc.Driver </property>
          - <property name=”connection.ur\l” >jdbc:mysql://localhost:3306/hellow-word</property>
          - <property name=”connection.username” >root </property>
          - <property name=”connection.password” >root </property>
          - <property name=”dialect” >orglhibernate.dialect.MySQLDialgect </property>
          - <property name=”show_sql” >true </property>
          - <property name=”hbm2ddlauto” >update </property>
            - Updates the table so it math the new mapping meta data whenever it changes → update parameter
          - <mapping resource=”domain/Message.hbm.xml”>
        - </session-factory>
      - </hibernate-configuration>
    - Example – Object Relational Metadata
      - <hibernate-mapping package=”domain”
        - <class name=”Message” Table=”message”>
          - <id name=”id” column=”ID”><generator class=”native” /></id>
          - <property name=”text column=”TEXT” type=”string” />
        - </class?
      - </hibernate-mapping>
    - Package attribute is the package name of the database tables
    - Generator → delegates key generated by the database
    - Id → Hibernate is smart enough to figure out the type of the id.

Hello World with JPA and annotations

- Session Factory
    - Configuration configuration = new Configuration().configure(“hiberanate.cfg.xml”);   // hibernate.cfg.xml put in classpath
    - Return
        - configuration.buildSessionFacdtory(new
            - StandardServiceRegistryBuild().applySettings(configuration.getProperties())
            - .build();
- Java Persistence
    - Session session = HibernateUtil.getSessionFactory().openSession();
    - Session.beginTransaction();
        - Message message = new Message(“Hibernate Hello World”);
        - Session.save(message);   Session;.getTransaction().commit();   Session.endTransaction();   Session.close();
- Instead of hbm files annotations can be used.
    - @Entity
    - @Table(name=”message”)
    - Public class Message {
        - @Id
        - @GeneratedValue(strategy=GenerationType.AUTO)   // The database will generate an unique id value
        - @Column(name=”ID”)
            - attribute nullable=false   columns are nullable by default
        - Private Long id;
        - @Column(name=”TEXTt”)
        - Private String text;
        - Public Message() {} ; Public Message(String text) { this.text = text; }
    - In the hibernate.cfg.xml the mapping tag will change <mapping class=”entity.Message” /> instead of a resource attribute.
    - The POJO must have a default Constructor so that Hibernate can use reflection to create the class when needed

Lab Notes

- Eclipse : Add to build path menu items puts the select files on the classpath.
-

Logging

- Enable Logging in Hibernate
- Uses Jboss Logging to allow the use of any Logging Frameworks
- Logging Levels
  - Off Fatal Error Warning Info Debug Trace All
- Log4.logger.org.hibernate.type.descriptor.sqlBasicBinder=TRACE; // Show Binding parameter values
- Log4j.rootLogger=OFF, stdout file // Get no logger information
- Log4j.logger.org.hibernate=All // Log Everything
- Log4j.logger.org.hibernate.SQL // Show SQL Statements
-

Manipulating Objects

- Retrieve a row from the database (a fully populated object)
  - Message msg = (Message)session.get(Message.clas,2L);   // 2L is the primary Key
  - If the object cannot be found a null value Is returned
- Update a row → Update the object and then call txn.commit()
  - Automatic Dirty Checking
- Delete an object → session.delete(msg) and call txn.commit();

Aggregation and Composition & Entities and Value Types & Component Mapping

- Aggregation – A relationship between a whole and it parts
  - The parts need not be destroyed when the whole is destroyed
- Composition → A strong part of aggregation. When the object is destroy it parts are destroyed with it
  - Each part may belong to only one whole
- An object of entity type has its own database identity (primary key value)
- An object of value type has no database identity (primary key value); It belongs to the entity
  - Depend on the identity of the entities that they belong to.
  - The lifecycle of a value type is bound to that of the owning entity
  - Classes like String and Integer are most simple value type classes
- Do all persistent classes have their own database identity → No
- Component Mapping
  - A Composition
  - Persisted as a value type
  - Data and Object Mapping → Could have more classes then tables
    - In the database you will Person which will contain all the fields
    - In the object may you will have two Object Person , Address with the annotation @Embedded
  - Mapping a Person
    - In the Person class we have the filed private Address address;
    - @Embeddable – Tells the database this type is embeddable
    - Public class Address {
      - Private String street;
      - Private String city
      - Private zipCode;
      - // Constructors , setters, getters etc.. needed }
  - @Embedded – Used inside another class tell the ORM that these field are part of the table to.

Composition Mapping

- Insert Data
  - Address adderss = new Address(…);
  - Person person = new Person(“Chuck”, address);
  - Session.save(person);
- Map the column name of a database table to custom names
  - Example in a class called Person
  - @Embedded
  - @AttributeOverrides({
    - @AttributeOverride(name=”street”   column = @Column(name=”address_street”))
      - Name attribute is the name of the field in the Class (private String street);
  - Very use if the are Value Type of the Same class inside a row. You can rename both
    - Example Value Address
    - By rename you can have home address and billing address

Mapping Associations

- Many to One
  - Each Guide could guide many students
  - Each student has one guide
  - Example
    - In the Student Object
    - @ManyToOne
    - @JoinColumn(name=”guide_id”).
- Cascades
  - When you persist an object you want it whole graph to persist as well
  - Called Transitive Persistence
  - Example
    - @ManyToOne(cascade=(CascadeType.PERSIST, CascadeType.REMOVE))
    - @JoinColumn(name=”guide_id)
    - Private Guide guide
    - Session.persist(student)
  - CascasdeType.Remove
    - Deletes the whole object graph of Student;
-
  -

Mapping Associations (2)

- One to Many
- One Guide could have many students
- In the Guide Object
  - @OneToMany(mappedBy=”guide”)
    - // Name of Attribute that the column is mapped by in the student object
    - // Requirement in a bidirectional mapping
    - // MappedBy → not owner of relationship
  - Private Set<Student> students = new Set<Student>();
- When both relations are setup then it is bidirectional
- Bidirectional Relationship
  - If the association is bidirectional one of the sides (and only one) has to be the owner of the relationship
  - The owner of the relationship is responsible for the association column(s) update
  - Many side in a one-to-many-bi directional relationship is (almost)) always the owner side
  - If a student is updated at the time of dirty checking the foreign key column will be update as well
  - If a guide is update, which is not the owner of a relationship, at the of dirty checking the guide will not be updated
  - Example
    - Guide guide = (Guide)session.get (Guide.class,2L);
    - Student student = (Student)session.get(Student.class,2);
    - Example 1 guide.getStudent(student) ; txn.commit()   // Will not update the owner since it is not the foreign key
    - Example 2 guide.setSalary(2500) ; guide.getStudent(student) ; txn.commit() /   // Will change the salary and not the relationship
    - Example 2 student.setGuide(guide)   // The Foreign key has been changed since it is the owner

Mapping Associations (3)

    - How do we make a guide responsible for its relationship
    - In the Guide class add the function public void addStudent(Student student) { student.add(student) ; student.setGuide(this); }
  - Owner is the entity that is persisted to the table that has the foreign key column
- One to One
  - Example
    - Each customer can only have one passport
    - Each passhort can be held to one customer.
  - Example
    - In the Customer Class
      - @OneToOne(cascade={CascadeType.PERSIST});
      - @JoinColumn name=”passport_id” unique = true)
      - Private Passport passport;
    - Int he Passport Class
      - @OneToOne(mappedBy=”passport”)
      - Private Customer customer
  - The owner of the relationship is responsible for the association column(s) update
  - To declare a side as not responsible for the relationship the attribute mappedBy is used
  -

ManyToMany Relationship

- Example
  - A move can have many actors
  - An actor can be in many movies.
  - Movie Object
    - @JoinTable(name=”movie_actor”,
      - joinColumns(@JoinColumn(name=”movie_id)),
      - inverseJoinColumns(@JonColumns(name=”actor_id”))
      -)
    - Private Set<Actior> actors = new HashSet<Actor>();
  - Actor Object
    - @ManyToMany(mappedBy=”actors”)
    - Private Set<Movie> movies = new HashSet<Movie>();
- The mapping of many to many relationship is completed using a join relationship
  - Move table, actor table, move_actor table
  -

What is JPA

- JPA is a java specification for accessing, persisting and managing data between java objects and a relational database.
- A set of guideline that a framework can implement
- JPA is a set of interfaces and Hibernate provides the functions that implement the interfaces
  - If you are not happy with hibernate you can swap it out for OpenJPA or EclipseLink
- Hibernate as a JPA Provider
  - Both Hibernate and JPA use the javax.persistence.* Annotations such as Column, Entity
  - Differences
    - Hibernate use a session Factory, JPA uses an entity Manager
    - HibernateUitl creates a sessionFactory Persistence create an EntityManager
- Example code
  - Public classs HelloWorldclient {
    - Public static void main(String[] args) {
      - EntityManagerFactory emf = Peristence.create