Object/Relational Mapping

Devdatta Kulkarni

Issues with JDBC

- Abstraction mismatch between Java Domain Objects and Relational tables
  - In our code we work with Java objects, but for storage and retrieval from the database, we need to work with *ResultSet*
  - Wouldn’t it be nice to not have to worry about low level classes, such as *PreparedStatement* and *ResultSet*?
- SQL is relatively low level interface for our needs
  - Wouldn’t it be nice to write queries referencing Java objects rather than tables and columns of a database?

Issues with JDBC

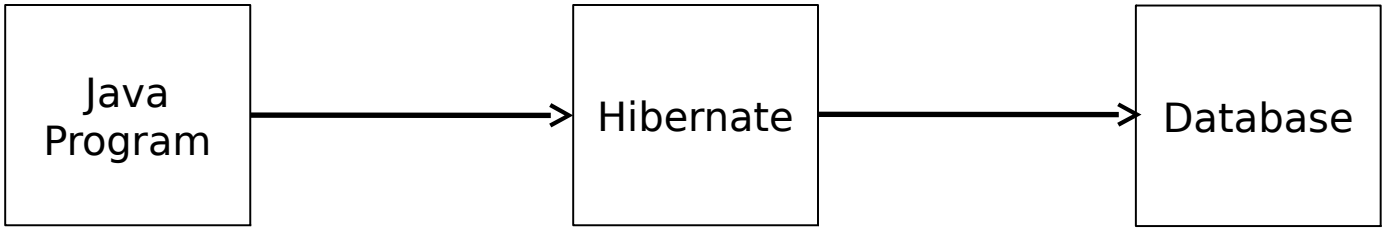
- *PreparedStatement* does not help with parameter types
  - As application developer you need to know the position and type of each parameter in a *PreparedStatement*

Object Relational Mapping

- Technology that maps Java objects to database tables and vice versa

- Specification
  - Java Persistence Architecture (JPA)
    - An API for Java O/RM
  - JPA implementations provided by:
    - Hibernate
    - iBatis
    - MyBatis
    - EclipseLink

Hibernate Main Concepts



- Configuration
  - Specified through “hibernate.cfg.xml” file
- Entities
  - Support JPA annotations
- Sessions

Configuration File

- Name
  - hibernate.cfg.xml
- Preferred location
  - src/main/resources/hibernate.cfg.xml
- Important fields
  - Connection fields
  - Cache provider
    - Session level cache (first level cache)
      - http://howtodoinjava.com/hibernate/understanding-hibernate-first-level-cache-with-example/
    - SessionFactory level cache (second level cache)
      - http://howtodoinjava.com/2013/07/02/how-hibernate-second-level-cache-works/
  - hbm2ddl.auto
    - Field which controls schema creation

Configuration File

- Important Parameters
  - hibernate.connection.driver_class
  - hibernate.connection.url
  - hibernate.connection.username
  - hibernate.connection.password
  - hibernate.connection.pool_size

Hibernate Entities

- A database table is represented using a Java class marked with the @Entity annotation
- The table that the entity maps to is marked with @Table annotation
  - Specifies the primary table for an annotated entity
  - The name of the table can be set by specifying the “name” attribute value
- Each entity instance corresponds to a row in that table
- The table columns are represented as *fields* in the entity class

Entities

@Entity @Table(name = “assignments”)

public class Assignment { ... }

Entities

@Entity @Table(name = “assignments")

public class Assignment { ... }

- Each field gets translated into a table column
- A field that is to be used as the primary_key should be marked with @Id annotation

Requirements for Entity Classes

- The class must be annotated with the javax.persistence.Entity annotation.
- The class must have a public or protected, no-argument constructor. The class may have other constructors.
- The class must not be declared final. No methods or persistent instance variables must be declared final.
- If an entity instance be passed by value as a detached object, the class must implement the Serializable interface.
- Entities may extend both entity and non-entity classes, and non-entity classes may extend entity classes.
- Clients must access the entity's state through *accesssor or business methods*.

Annotations within an Entity

- The persistent state of an entity can be accessed either through the entity's instance variables or through JavaBeans-style properties.
- Entities may either use *persistent fields* or *persistent properties*.
- When the mapping annotations are applied to the entity's *instance variables:*
  - It means that the entity uses *persistent fields*
- When the mapping annotations are applied to the entity's *getter methods* for JavaBeans-style properties
  - It means that the entity uses *persistent properties*
- You cannot apply mapping annotations to both fields and properties in a single entity.

Primary Keys in Entities

- Each entity has a unique object identifier
  - The primary key
- An entity may have either a simple or a composite primary key
- Simple primary keys use the javax.persistence.Id annotation to denote the primary key property or field

Primary key – Id generation

- Option 1: Use JPA strategies

  @GeneratedValue(strategy=GenerationType.IDENTITY)
- JPA strategies
  - AUTO
  - IDENTITY
  - SEQUENCE
  - TABLE
- Option 2: Use Hibernate’s generator

  @GeneratedValue(generator="increment")

  @GenericGenerator(name="increment", strategy = "increment")

Hibernate Session

- Main interface between a Java application and Hibernate
- Life cycle of a session is bounded by beginning and end of a transaction
- The function of a session is to offer create, read and delete operations for instances of mapped entity classes

Session

- org.hibernate.SessionFactory create and pool JDBC connections
- Open a new Session
  - Session session = sessionFactory.openSession();

Session details

- Session object is not thread-safe. Instead each thread/transaction should obtain its own instance from a SessionFactory.
- If the Session throws an exception, the transaction must be rolled back and the session discarded:
  - The internal state of the Session might not be consistent with the database after the exception
- https://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/Session.html

Session pattern

```
Session session = factory.openSession();
 Transaction tx;
 try {
     tx = session.beginTransaction();
     //do some work
     tx.commit();
 }
 catch (Exception e) {
     if (tx!=null) tx.rollback();
     throw e;
 }
// continue using the session

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

Hibernate Object states

- Objects may exist in one of three states:
  - *transient:* just created, has never been made persistent, not associated with any Session
  - *persistent:* associated with a unique Session
  - *detached:* previously persistent; not associated with a Session anymore

Object state: Transient

- An object is transient if it has just been instantiated using the *new* operator, and it is not associated with a Hibernate Session.

Examples

```
public Long addAssignment(String title) throws Exception {
   Session session = sessionFactory.openSession();
   Transaction tx = null;
   Long assignmentId = null;
   try {
   tx = session.beginTransaction();
(transient) Assignment newAssignment = new Assignment(title, new
Date(), new Long(1));
(persistent)   session.save(newAssignment);
                      tx.commit();
               } catch (Exception e) {
                   tx.rollback();
               }
}
```

Object state: Persistent

A persistent instance has a representation in the database and an identifier value.

It might just have been saved or loaded.

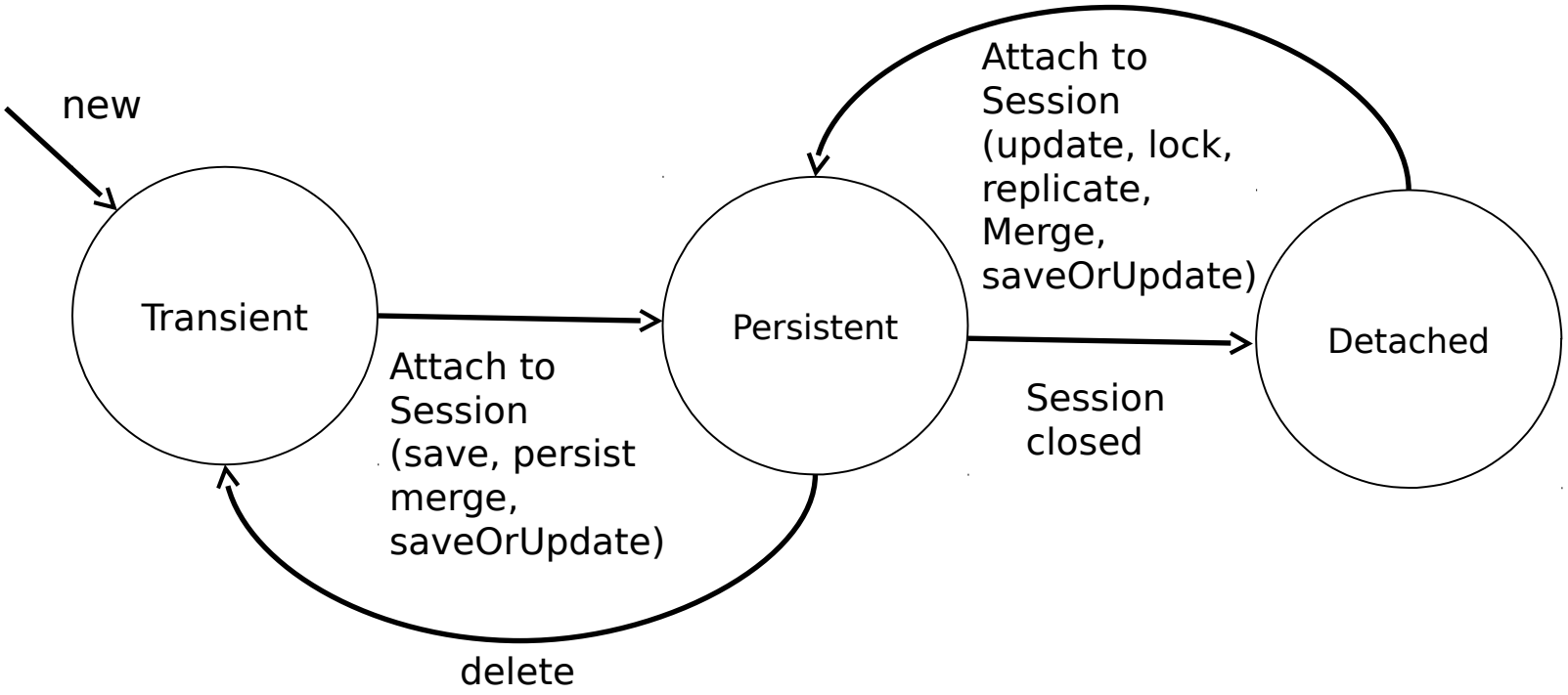
It is by definition in the scope of a Session.

Hibernate will detect any changes made to an object in persistent state and synchronize the state with the database when the unit of work completes.

Detached state and state transitions

- Object state: Detached
  - Not associated with any Session
  - Was previously persistent
- State transitions
  - Transient instances may be made persistent by calling save(), persist(), merge(), saveOrUpdate()
  - Persistent instances may be made transient by calling delete()
  - Detached instances may be made persistent by calling update(), saveOrUpdate(), lock(), merge() or replicate()

State transitions



https://docs.jboss.org/hibernate/core/3.3/reference/en/html/objectstate.html

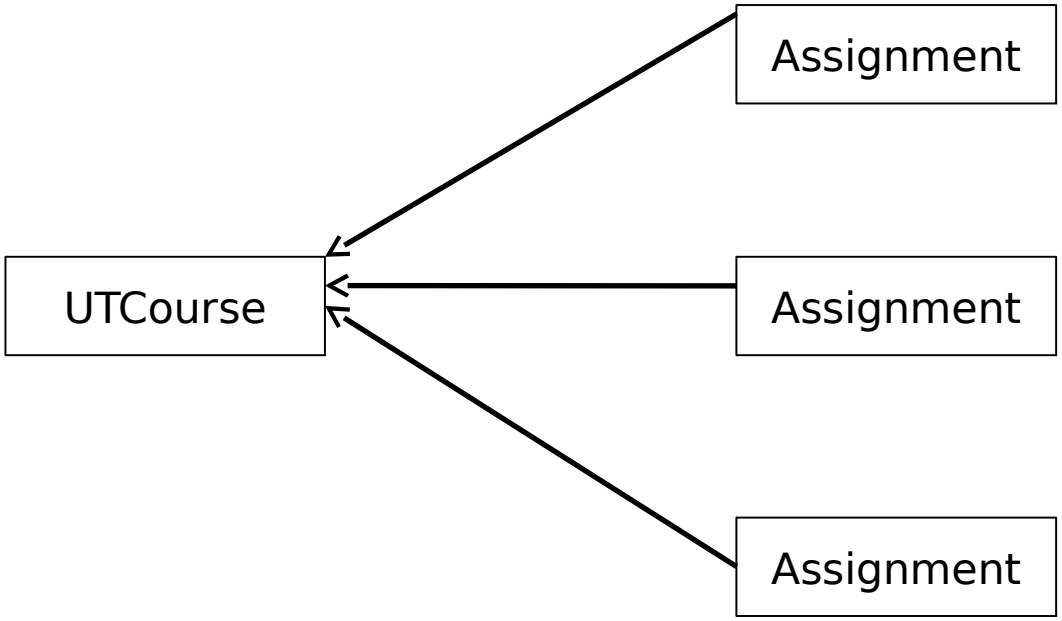
Hibernate and SQL correspondence

Hibernate API methods	Corresponding SQL actions
save(), persist()	insert
delete()	delete
update(), merge()	update
saveOrUpdate()	insert or update

JPA Entity Relationships

Example

- A UTCourse can have one or more assignments



JPA Entity Relationships

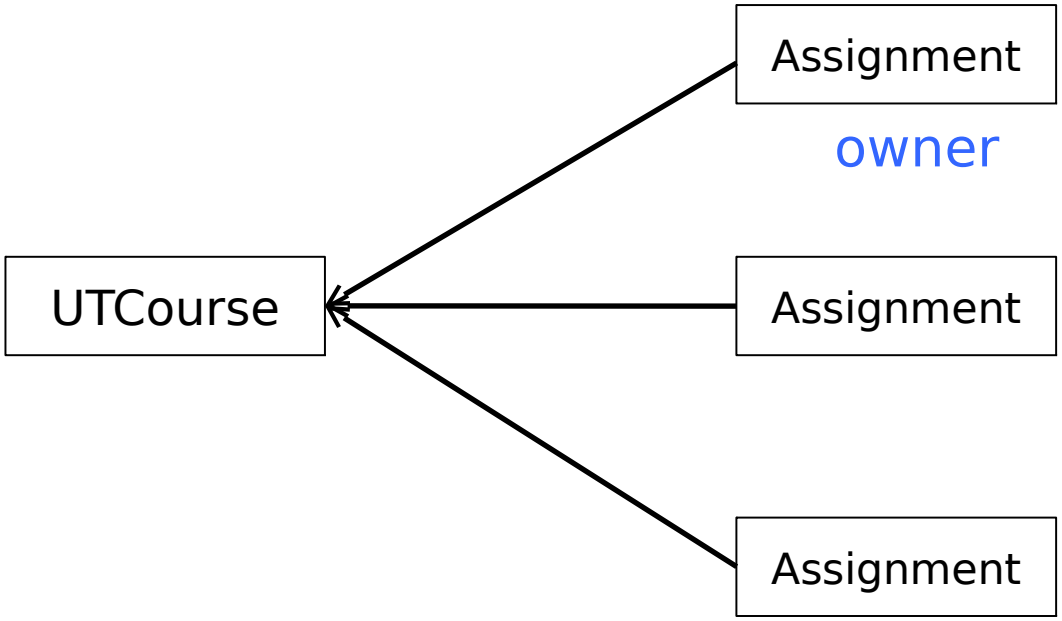
- **One-to-one**: Each entity instance is related to a single instance of another entity.
- **One-to-many**: An entity instance can be related to multiple instances of the other entities.
- **Many-to-one**: Multiple instances of an entity can be related to a single instance of the other entity.
- **Many-to-many**: The entity instances can be related to multiple instances of each other.

Direction in Entity Relationships

- Unidirectional
  - Has only an owning side
- Bidirectional
  - Has an owning side and an inverse side
- The owning side of the relationship is the entity that has reference to the other entity
- The owning side determines how Hibernate makes updates to the relationship in the database.

Hibernate Entity Association

- A UTCourse can have one or more assignments



Hibernate Entity Relationship Annotations

- Annotations
  - @ManyToOne
    - Added on the “owning” side of a relationship
    - Typically combined with @JoinColumn annotation
  - @OneToMany
    - Added on the “inverse” side of the relationship
  - Show example

Entity relationships

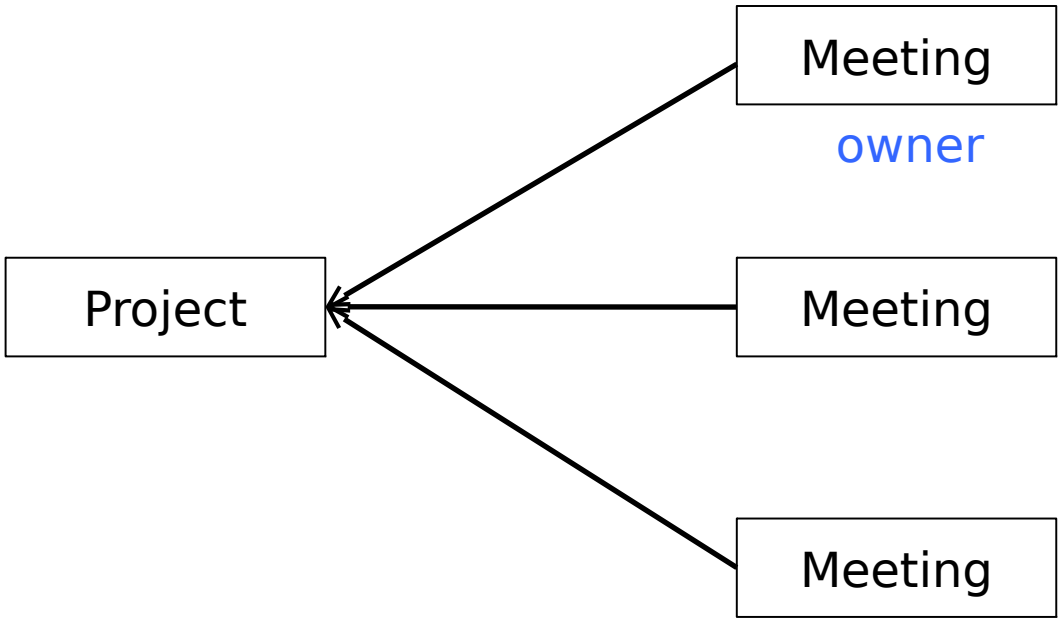
```
 // In the “owner” side – Assignments class
@ManyToOne
@JoinColumn(name="course_id")
 public UTCourse getCourse123() {
        return this.utcourse;
 }

// In the reverse side  -- UTCourse class
@OneToMany(mappedBy="course123")
public Set<Assignment> getAssignments() {
     return this.assignments;
}
```

Exercise: 10 minutes

- Model a project that can have one or more meetings:
  - Define the data model
    - Entities
    - Fields/columns
    - Relationships

Hibernate Entity Association



Fields/Columns

- Project
- Meetings

Entity relationships

Hibernate Operations

Hibernate Operations

- Insert (REST: Create/POST)
- Select (REST: Read/GET)
  - From a single table without any criteria
  - From a single table with a selection criteria
  - From two tables with join condition
- Delete (REST: Delete/DELETE)
- Update (REST: Update/PUT)

Hibernate: Insert

```
Session session = sessionFactory.openSession();
session.beginTransaction();

Assignment newAssignment = new
Assignment( title, new Date() );

session.save(newAssignment );

session.getTransaction().commit();
session.close();
```

Hibernate Querying

- HQL
  - Hibernate Query Language
    - Similar to SQL, but closer to the Java world
    - Supported via the *Criteria* API
- Native SQL

HQL: Criteria API

Criteria criteria =
session.createCriteria(Assignment.class).
add(Restrictions.*eq("title", "ETL"));*

Query condition

Session.m1(param).m2(param2)

Method chaining

Selection Query using Native SQL

```
List<Assignment> assignments =
session.createQuery("from Assignment
where course=1").list();
```

“Assignment” → Entity class name

Parameterized variables

```
String query = "from Assignment a where a.title = :title”;

Assignment a = (Assignment)session.createQuery(query).setParameter("title", title).list().get(0);
```

Join Query

Use names from Java side when creating the query

- Entity class name
- Property name

String query = "from Assignment a join a.course123 c where c.courseName = :cname";

Assignment: Entity class name

course123, courseName: Property names

Hibernate Delete

- Deleting a row in a table that *is not* part of any relationship(s)
- Deleting a row in a table that *is* part of a relationship
  - leads to ConstraintViolationException if no cascaded delete specified
- Cascading delete
  - Add @Cascade({CascadeType.DELETE}) on the inverse side of relationship
- Parameterized delete

Hibernate Criteria API

- http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html/querycriteria.html#querycriteria-associations
- http://levelup.lishman.com/hibernate/hql/joins.php

Advanced Hibernate concepts

- Hibernate caching
  - Session cache
  - SessionFactory cache
  - Query cache
  - http://www.tutorialspoint.com/hibernate/hibernate_caching.htm
- N+1 query issue
  - http://stackoverflow.com/questions/97197/what-is-the-n1-selects-issue

H2

- In memory database
- Useful for testing and debugging

References

- ORM/JPA
  - http://martinfowler.com/bliki/OrmHate.html
  - http://hibernate.org/orm/
  - https://docs.jboss.org/author/display/AS7/JPA+Reference+Guide
  - http://hibernate.org/orm/what-is-an-orm/
  - http://www.h2database.com/html/tutorial.html
  - http://docs.oracle.com/javaee/6/tutorial/doc/bnbpy.html

References

- http://viralpatel.net/blogs/hibernate-one-to-many-annotation-tutorial/
- http://hibernate.org/orm/what-is-an-orm/
- http://hibernate.org/orm/documentation/getting-started/

Hibernate Operations

- Hibernate operations are performed as part of a session
- A session is
  - Main interface between a Java application and Hibernate
  - Life cycle of a session is bounded by beginning and end of a transaction
  - The function of a session is to offer create, read, and delete operations for the entities
  - Pattern
    - Begin a transaction
    - Add entities to be “saved/modified” in the session
    - Perform commit action
    - Close the transaction
    - On error, roll-back the transaction

Will Hibernate add an Id?

- *No*

- Hibernate expects that each entity has some field with the Id annotation
  - hibernate.AnnotationException is raised if such a field is not present

Schema updates and data migrations

- Hibernate supports schema changes
  - <property name="hibernate.hbm2ddl.auto" value="create-drop" />
  - <property name="hibernate.hbm2ddl.auto" value="update" />
  - http://docs.jboss.org/hibernate/core/3.3/reference/en/html/session-configuration.html#configuration-optional
- Hibernate does not support data migration
  - Typically, separate tool is used
    - https://flywaydb.org/
    - http://www.liquibase.org/
    - http://www.mybatis.org/migrations/