Java EE – JPA

- ORM: Object Relational Mapping
- JPA: Java Persistence API
- Hibernate ORM
- Entity, EntityManager, EntityTransaction
- JPQL
- Progetto di riferimento
  - https://github.com/egalli64/mhja
    - Tomcat 9, Hibernate

ORM

- Object Relational Mapping
- Integrazione tra due paradigmi
  - Object Oriented
  - Relazionale
- Alcuni problemi
  - Diverso approccio per
    - Identità: PK *(database)*, == *(reference),* equals() *(uguaglianza tra oggetti)*
    - Associazione: FK vs has-a
    - Navigazione nei dati: JOIN vs reference
  - Tabelle e oggetti possono definire entità con diversa granularità (es.: indirizzo)
  - Come gestire l'ereditarietà in un RDBMS?

JPA

- Java Persistency API
  - Versione corrente 2.2
- Implementazioni basate su specifiche Oracle
  - Red Hat JBoss Hibernate
  - EclipseLink
  - …
- JPQL: simile a SQL, dialetti per le implementazioni
- Nata come soluzione ORM più leggera rispetto a quella offerta da EJB
- Può essere usata in Java SE e EE

Hibernate

- Framework ORM – Your relational data. Objectively.
  - https://hibernate.org/orm/
  - Nato nel 2001 (Gavin King et al.) come alternativa più semplice a EJB
  - Dal ~2010 implementa (anche) JPA
  - La versione corrente è 5.4
- Mappaggio tra classi Java e tabelle di database
  - File di configurazione XML
  - Annotazioni
- Definizione di un linguaggio simile a SQL: HQL

Session vs EntityManager

- Session
  - concetto nativo Hibernate
  - gestisce la connessione al database via JDBC
- EntityManager
  - standard JPA
  - Costruito in Hibernate come wrapper di Session
    - se necessario, accesso alla session via unwrap()

SessionFactory

- Data una configurazione
  - Proprietà della connessione al database e delle sessioni
  - In src/main/resource
    - hibernate.cfg.xml
    - hibernate.properties
- Permette di creare sessioni
- Tipicamente si crea un solo oggetto nell’applicazione
  - (è thread safe)

Session

- È autocloseable
  - try-with-resources
- Gestisce la connessione col database
  - Query
  - Transazioni
  - …
- I dettagli JDBC sono gestiti internamente

EntityManagerFactory

- Data una configurazione
  - META-INF/persistence.xml
- Permette di creare EntityManager
- In un *container full* Java EE può essere gestita
  - direttamente dal container
  - dall’applicazione, via Persistence
- In Java SE, e in un *container web* come Tomcat
  - va gestita esplicitamente via Persistence
- Persistence.createEntityManagerFactory()

persistence.xml

- Definisce le persistency unit usate nell'app
- Ognuna deve avere un nome univoco
  - Elemento persistence-unit, attributo name
- all'interno di persistence-unit si definisce il data source
  - Per Tomcat
    - non-jta-data-source, ex: java:comp/env/jdbc/me
  - Proprierties
    - hibernate.dialect → org.hibernate.dialect.MySQLDialect
    - …

EntityManager

- Non è autocloseable
  - va esplicitamente chiuso al termine del suo uso
- Gestisce la connessione al database
  - (Hibernate) appoggiandosi a Session

Entity

- Java Bean, POJO annotato
- @Entity
  - Per default fa riferimento a una tabella con lo stesso nome
  - @Table name
- Una proprietà deve essere annotata come chiave @Id
  - Riferimento alla PK
  - @GeneratedValue per generazione automatica dei valori (Identity→Autoincrement MySql)
- Le proprietà sono mappate automaticamente a colonne della tabella
  - Per default si assume che proprietà e colonne abbiano lo stesso nome
  - @Column name
- Eventuali proprietà non persistenti vanno annotate @Transient

Stati di un Entity

- Relativi al contesto di persistenza, gestito via entity manager
  - New: non ancora associata
    - Oggetto creato
  - Managed: associata, sincronizzata con il database
    - persist(), find(), merge()
  - Removed: rimossa dal database
    - Commit di una transazione, remove(), …
  - Detached: non più associata
    - Terminazione di una transazione

EntityTransaction

- Transazione relativa ad un EntityManager
  - getTransaction()
- Va esplicitamente aperta e chiusa
  - begin()
  - commit() / rollback()
    - Prima della chiusura dell’entity manager
- Le operazioni DML devono essere eseguite in una transazione

EntityManager – alcuni metodi

- persist(), in una transazione
  - Rende persistente e managed un’entità
  - o tira una eccezione della famiglia PersistenceException
- find()
  - Cerca una entità via id
  - Ritorna una managed entity o null
- merge(), in una transazione
  - Aggiorna l’entità sul database, se esiste l’id, altrimenti ne crea una nuova
  - L’entità diventa managed
- remove(), in una transazione
  - Elimina una entità managed dal database

JPQL

- Java Persistency Query Language
- Simile a SQL ma basato sulle entità JPA
- Permette di eseguire SELECT, UPDATE, DELETE
- Es: selezione delle *entità* Employee filtrate per salario:
  - SELECT e from Employee e where e.salary > 3000

JPQL Query

- Query EntityManager.createQuery(jpql)
- TypedQuery<Entity> EntityManager.createQuery(jpql, Entity.class)
- Query parametrizzate
  - Posizionali ?1, ?2, …
  - Nome :xyz
  - Query.setParameter(pos/name, value);
- List<Entity> Query.getResultList();
- La entità può essere annotata @NamedQuery name e query
  - TypedQuery<Entity> EntityManager.createNamedQuery(jpql, Entity.class)

Relazioni tra entità

- Definite per mezzo di annotazioni
  - @OneToOne
  - @OneToMany, @ManyToOne
  - @ManyToMany
- La FK è indicata con l’annotazione
  - @JoinColumn su una istanza dell’entità in relazione

@OneToOne

- Relazione one to one tra Coders e Teams
- L'entità Team ha una proprietà leader di tipo Coder
  - @OneToOne(optional = false)
    - ogni team ha un leader
  - @JoinColumn(name="leader_id")
    - nome della FK: Teams.leader_id → Coders.coder_id
- L'entità Coder ha una proprietà Team
  - @OneToOne(optional=true, mappedBy="leader")
    - Un coder non è necessariamente un team leader
    - "leader" è il nome della proprietà di Team che mappa il coder

@OneToMany @ManyToOne

- Relazione many to one tra Countries e Regions
  - L'entità Country ha una proprietà region di tipo Region
    - @ManyToOne @JoinColumn(name="region_id")
- Relazione one to many tra Regions e Countries
  - L'entità Region ha una proprietà Set<Country>
    - @OneToMany(mappedBy="region")
  - La select è by default "lazy", non vengono lette le countries
  - Comportamento "eager"
    - @OneToMany(mappedBy="region", fetch=FetchType.EAGER) – da usare con cautela
    - Preferita la clausola JOIN FETCH su select quando necessario

@ManyToMany

- Relazione many to many tra Coders e Teams
  - Simmetrica, scegliamo noi il master → Teams
- L'entità Team ha una proprietà coders di tipo Set<Coder>
  - @ManyToMany
  - @JoinTable(
    - name = "team_coder",
    - joinColumns = @JoinColumn(name = "team_id"),
    - inverseJoinColumns = @JoinColumn(name = "coder_id"))
- L'entità Coder ha una proprietà teams di tipo Set<Team>
  - @ManyToMany(mappedBy = "coders")
- Valgono le stesse considerazioni lazy-eager indicate per la OneToMany