Chapitre 5 : JPA and EclipseLink

Object-relational mapping (ORM)

- The process of mapping Java objects to database tables and vice versa is called "Object-relational mapping" (ORM).
- The Java Persistence API (JPA) is one possible approach to ORM.
- JPA is a specification and several implementations are available.
- Popular implementations are **Hibernate**, **EclipseLink** and **Apache OpenJPA**. The reference implementation of JPA is EclipseLink.

Java Persistence API (JPA)

- Via JPA the developer can map, store, update and retrieve data from relational databases to Java objects and vice versa.
- JPA permits the developer to work directly with objects rather then with SQL statements. The JPA implementation is typically called persistence provider. JPA can be used in Java-EE and Java-SE applications.
- The mapping between Java objects and database tables is defined via persistence metadata. The JPA provider will use the persistence metadata information to perform the correct database operations.

Java Persistence API (JPA)

- JPA typically defines the metadata via annotations in the Java class. Alternatively the metadata can be defined vian XML or a combination of both. A XML configuration overwrites the annotations.
- The following description will be based on the usage of annotations.
- JPA defines a SQL-like Query language for static and dynamic queries.
- Most JPA persistence provider offer the option to create automatically the database schema based on the metadata.

Entity

- A class which should be persisted in a database it must be annotated with javax.persistence.Entity. Such a class is called Entity.
- JPA will create a table for the entity in your database.
- Instances of the class will be a row in the table.
- All entity classes must define a primary key, must have a non-arg constructor and or not allowed to be final. Keys can be a single field or a combination of fields.
- JPA allows to auto-generate the primary key in the database via the @GeneratedValue annotation.
- By default, the table name corresponds to the class name. You can change this with the addition to the annotation @Table(name="NEWTABLENAME").

Persistence of fields

- The fields of the Entity will be saved in the database.
- JPA can use either your instance variables (fields) or the corresponding getters and setters to access the fields. You are not allowed to mix both methods.
- If you want to use the setter and getter methods the Java class must follow the Java Bean naming conventions.
- JPA persists per default all fields of an Entity, if fields should not be saved they must be marked with @Transient.

Persistence of fields

- By default each field is mapped to a column with the name of the field. You can change the default name via @Column (name="newColumnName").
- The following annotations can be used.

Table 1. Annotations for fields / getter and setter	
@Id	Identifies the unique ID of the database entry
@GeneratedValue	Together with an ID this annotation defines that this value is generated automatically.
@Transient	Field will not be saved in database

Relationship Mapping

- JPA allows to define relationships between classes, e.g. it can be defined that a class is part of another class (containment).
  - Classes can have one to one, one to many, many to one, and many to many relationships with other classes.
- A relationship can be bidirectional or unidirectional, e.g.
  - in a bidirectional relationship both classes store a reference to each other while in an unidirectional case only one class has a reference to the other class. Within a bidirectional relationship you need to specify the owning side of this relationship in the other class with the attribute "mappedBy", e.g. @ManyToMany(mappedBy="attributeOfTheOwningClass".

Relationship Mapping

- Relationship annotations:
- @OneToOne
- @OneToMany
- @ManyToOne
- @ManyToMany

Entity Manager

- The EntityManager is created by the EntityManagerFactory.
- The entity manager javax.persistence.EntityManager provides the operations from and to the database, e.g. find objects, persists them, remove objects from the database, etc.
- In a JavaEE application the entity manager is automatically inserted in the web application. Outside JavaEE you need to manage the entity manager yourself.
- Entities which are managed by an Entity Manager will automatically propagate these changes to the database (if this happens within a commit statement).
- If the Entity Manager is closed (via close()) then the managed entities are in a detached state. If synchronize them again with the database a Entity Manager provides the merge() method.

Persistence units

- The EntityManager is created by the EntityManagerFactory which is configured by the persistence unit.
- The persistence unit is described via the persistence.xml file in the META-INF directory of the source folder. A set of entities which are logical connected will be grouped via a persistence unit. The persistence.xml file defines the connection data to the database, e.g. the driver, the user and the password.