# Übung 1: OR Mapper (Einführung)

Datenbanksysteme 2, FS 2018

PROF. STEFAN KELLER, MARCEL HUBER

2018-02-20

# Übung 1: OR Mapper (Einführung)

## Zielsetzung

Einsatz von JPA als OR-Mapper

- Sammeln von Erfahrung in der Abbildung von Entitäten und Beziehungen zur DB.
- Umsetzung von Lese- und Schreiboperationen auf dem persistenten Objektmodell.

## Vorbereitung

Für diese Übung verwenden Sie JPA mittels EclipseLink[1] basierend auf einer PostgreSQL[2] Datenbank.

### Eingesetzte Software

Die eingesetzten Tools können Sie entweder als *Native Programme* oder als *Docker services* laufen lassen. Eine Mischung ist natürlich auch möglich.

**Native Programme** Stellen Sie sicher, dass nachfolgende Programme auf Ihrem Rechner verfügbar sind.

- PostgreSQL >=9.6[3] (enthält `psql` command line Tool)
- pgAdmin[4]
- Java 8 JDK
- Java-IDE; Eclipse-JEE-oxygen[5] (empfohlen), Eclipse-Java-oxygen[6] oder IntelliJ-IDEA[7]

**Programme als Docker services** Linux, respektive Docker Benutzer, können die aufgelisteten Programme auch als Container laufen lassen. Folgende Container brauchen wir in dieser Übung:

- `db2-postgres`
- `psql`
- `eclipse-jee`
- `pgadmin`

Die Spezifikation der verwendeten Services ist in der Datei `crane.yml`[8] aufgelistet. Ein Service kann dann mit dem Tool `crane`[9] gestartet werden. Eine kurze Anleitung dazu finden Sie auf dieser Seite[10].

[1] https://wiki.eclipse.org/EclipseLink
[2] https://www.postgresql.org/
[3] https://www.postgresql.org/download/
[4] https://www.pgadmin.org/
[5] https://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/oxygen2
[6] https://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/oxygen2
[7] https://www.jetbrains.com/idea/
[8] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/blob/v1.1/Helpers/crane.yml
[9] https://github.com/michaelsauter/crane
[10] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/blob/v1.1/Helpers/README.md#crane

### Vorlagen

Die Projektvorlage für diese Aufgabe finden Sie in dieser Zip-Datei[11]. In dieser finden Sie das Eclipse-Projekt `dbs2.jpa_exercise`[12] mit Java Klassen, jedoch noch ohne JPA-Annotationen. Die JPA Libraries und PostgreSQL-JDBC Drivers für Java-8 sind im Ordner `lib/`[13] **bereits enthalten**. Daher können Sie den Schritt JPA Libraries und PostgreSQL-JDBC Drivers beziehen überspringen wenn die verwendete Software dazu passt.

In der Vorlage zur Übung finden Sie die Datei `persistence.xml`[14], welche Ihnen als Vorlage zur Konfiguration Ihres JPA-Programms dienen kann. Sie müssen dieses eventuell noch an Ihre Umgebung anpassen wenn Sie beispielsweise einen anderen Port zum PostgreSQL[15] Server verwenden.

Die Verzeichnisstruktur eines Java Eclipse Projektes für JPA sieht demnach ungefähr wie folgt aus:

```
|-- lib
|   |-- eclipselink.jar
|   |-- javax.persistence_2.2.0.v201708071007.jar
|   `-- postgresql-42.2.1.jar
|-- src
|   |-- ch
|   |   `-- hsr
|   |   ...
|   `-- META-INF
|       `-- persistence.xml
```

**Hinweis:**

- Eine gute Dokumentation zu JPA und Annotationen finden Sie bei `objectdb.org`[16]. Sie können aber auch in der neuen EclipseLink Dokumentation[17] oder **besser** in der alten EclipseLink Dokumentation[18] nachschlagen.

**Datei `persistence.xml`** Die Datei `persistence.xml`[19] wird von JPA im Folder `src/META-INF`[20] im Eclipse-Projekt bzw. im Folder `META_INF` im JAR-File benötigt.

Mit der Einstellung `<property name="eclipselink.logging.level" value="ALL"/>`[21] können Sie den Log Level von EclipseLink konfigurieren damit Sie beispielsweise die generierten SQL Queries sehen.

[11] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/-/jobs/artifacts/master/raw/OR-Mapper_JPA.zip?job=OR-Mapper_JPA
[12] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/blob/v1.1/OR-Mapper_JPA/.project
[13] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/blob/v1.1/OR-Mapper_JPA/lib
[14] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/blob/v1.1/OR-Mapper_JPA/src/META-INF/persistence.xml
[15] https://www.postgresql.org
[16] https://objectdb.org/java/jpa
[17] https://wiki.eclipse.org/EclipseLink
[18] https://wiki.eclipse.org/EclipseLink/UserGuide/JPA
[19] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/blob/v1.1/OR-Mapper_JPA/src/META-INF/persistence.xml
[20] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/blob/v1.1/OR-Mapper_JPA/src/META-INF
[21] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/blob/v1.1/OR-Mapper_JPA/src/META-INF/persistence.xml#L14

**JPA Libraries und PostgreSQL-JDBC Drivers beziehen** Wenn Sie die Libraries selber einrichten wollen oder Treiber für eine andere Java Version oder Datenbank benötigen müssen Sie diese vorgängig herunterladen. Vergessen Sie dabei nicht, die entsprechenden `.jar` Dateien im `classpath` einzutragen!

- Beziehen Sie die neuste Version von EclipseLink Installer Zip[22]. Für diese Übung haben wir die Version aus diesem Link[23] verwendet. Im Paket finden Sie zwei JAR-Libraries, die Sie für das JPA Programm benötigen:
  - `eclipselink/jlib/eclipselink.jar`[24]
  - `eclipselink/jlib/jpa/javax.persistence_2.2.0.v201708071007.jar`[25]
- JDBC-Driver für PostgreSQL[26].

### Datenbank `bank`

- Für diese Übung benötigen Sie PostgreSQL[27] und die Datenbank `bank`[28]

  Import von `bank` geschieht durch Eingabe des folgenden Kommandos innerhalb des entsprechenden Unterverzeichnis

  ```
  psql -U postgres -f 0_runAllScripts.sql
  ```

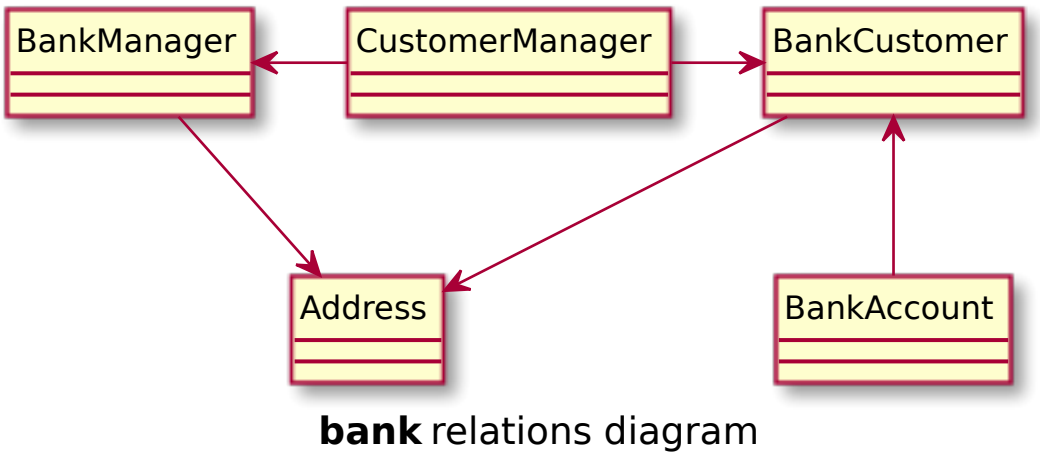
  oder durch Ausführen von `run.bat`[29] oder `run.sh`[30] im entsprechenden Verzeichnis.

  Die Daten befinden sich in diesem Ordner[31] und in dieser zip Datei[32].

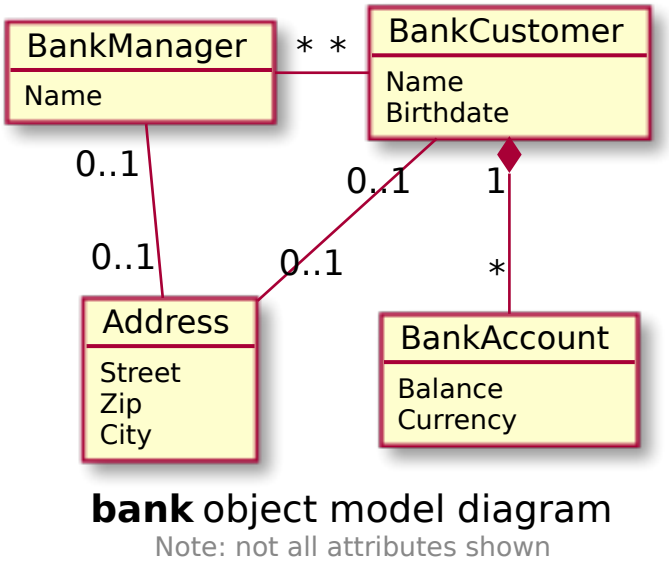
## Ausgangslage

Ausgangslage ist die Datenbank `bank`[33] mit folgendem Relationenmodell:

[22] https://www.eclipse.org/eclipselink/#download
[23] https://www.eclipse.org/downloads/download.php?file=/rt/eclipselink/releases/2.7.1/eclipselink-2.7.1.v20171221-bd47e8f.zip&r=1
[24] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/blob/v1.1/OR-Mapper_JPA/lib/eclipselink.jar
[25] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/blob/v1.1/OR-Mapper_JPA/lib/javax.persistence_2.2.0.v201708071007.jar
[26] https://jdbc.postgresql.org/download.html
[27] https://www.postgresql.org
[28] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/blob/v1.1/Databases/bank
[29] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/blob/v1.1/Databases/bank/run.bat
[30] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/blob/v1.1/Databases/bank/run.sh
[31] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/blob/v1.1/Databases/bank
[32] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/-/jobs/artifacts/master/raw/Databases.zip?job=Databases
[33] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/blob/v1.1/Databases/bank




Ergänzen Sie die vorbereiteten Klassen sukzessive mit den entsprechenden JPA Annotationen, sodass folgendes Objektmodell daraus resultiert:




## Aufgaben

### Aufgabe 1: Entity Mapping

1. Bilden Sie die Tabellen `BankCustomer`, `BankAccount`, `BankManager` und `Address` mit JPA als entsprechende Entity-Klassen **vorerst ohne Beziehungen** in ein Java-Programm ab. Setzen Sie zunächst die einzelnen Klassen mit den Basisattributen (Integer, Double, String, Date, etc.) um.

   **Tipp** Beginnen Sie mit einer einfachen Abbildung, z.B. `Address`

2. Schreiben Sie ein Programm, welches alle Instanzen der Entity-Klassen von der DB lädt und deren Zustände in der Standard Output Konsole ausgibt.

### Aufgabe 2: Entity Relations

Erweitern Sie das Programm der Aufgabe Entity Mapping so, dass es auch Beziehungen zwischen den Entities unterstützt.

1. Implementieren Sie alle Beziehungen zwischen den Datenbank-Tabellen als entsprechende Beziehungen zwischen den Entities.
2. Schreiben Sie ein Programm, welches alle Instanzen der `BankCustomer` direkt von der DB lädt und dann über den Objektgraph zu allen benachbarten Entities navigiert (`BankAccount`, `Address`, `BankManager`).
3. Bei welchen der modellierten Beziehungen verwendet JPA Lazy Loading? Konfigurieren Sie diese Relationen auch für Eager Loading.

### Aufgabe 3: Entity Updates

Implementieren Sie eine Methode, die eine Banküberweisung zwischen zwei Konten durchführt. Dies soll über JPA mittels Änderungen auf den Entities realisiert werden.

```
transfer(fromAccountId, toAccountId, amount)
```

Bei nicht existenten Konten soll eine Exception geworfen werden.

### Aufgabe 4: Entity Deletes

Entwickeln Sie eine Funktionalität, um ein Bankkonto zu schliessen. Dazu soll die entsprechende `BankAccount` Entity in JPA entfernt werden.

```
closeAccount(accountId)
```

Falls das Konto nicht existiert, soll eine Exception geworfen werden.

### Aufgabe 5: Entity Inserts

Programmieren Sie die Funktionalität zum Erfassen eines neuen Bankkunden. Konkret soll via JPA eine neue persistente `BankCustomer` Entity alloziert werden und in die DB gespeichert werden.

```
openAccount(name, birthDate)
```

**Hinweis:**

- Verwenden Sie die Annotation `@GeneratedValue` bei dem Id-Field/Property der Entity-Klasse. Dies bewirkt, dass automatisch ein neuer Primary Key für die Id der Entity beim Speichern generiert wird.

```
@Entity
public class BankCustomer {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private long customerId;
    ...
}
```

Denken Sie daran, dass Sie eventuell selber sicherstellen müssen, dass die entsprechende `SEQUENCE` den richtigen Startwert gesetzt hat. Das passiert dann, wenn Sie Datensätze mit expliziten `id` Werten in die Datenbank einfüllen weil Sie diese beispielsweise zur Referenzierung aus einem anderen Datensatz verwenden. Mehr dazu finden Sie in diesem Issue[34].

### Fakultativ

- Beim Erfassen des neuen Kunden kann auch gleichzeitig ein neues zugehöriges BankAccount-Objekt mit Kontostand 0 angelegt und in der DB gespeichert werden.

## Musterlösung

Die Musterlösung zu den Übungen finden Sie im Branch OR-Mapper_JPA-Solutions[35] oder als zip-Datei[36].

[34] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/issues/2
[35] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/tree/OR-Mapper_JPA-Solutions/OR-Mapper_JPA
[36] https://gitlab.dev.ifs.hsr.ch/m1huber/Dbs2Uebungen/-/jobs/artifacts/OR-Mapper_JPA-Solutions/raw/OR-Mapper_JPA.zip?job=OR-Mapper_JPA