DB1 CheatSheet #HSR HS 2013

Emanuel Duss, Marcel Loop

SQL

DDL: Data Definition Language

Create Table mit Column Constraints

```
CREATE TABLE Projekt(
ProjNr INTEGER PRIMARY KEY,
Bezeichnung VARCHAR(20) NOT NULL UNIQUE,
Start DATE NOT NULL DEFAULT CURRENT_DATE,
Dauer INTEGER NULL
  CHECK(DAUER BETWEEN 10 AND 100),
AbtNr INTEGER NOT NULL REFERENCES Abt);
```

Table statt Column Constraints:

```
CREATE TABLE Projekt(... --- vgl. oben
PRIMARY KEY (ProjNr),
FOREIGN KEY (AbtNr) REFERENCES Abt);
```

Constraint mit Name

```
ALTER TABLE Projekt
  ADD CONSTRAINT fk_ProjAng
  FOREIGN KEY (ProjLeiter) REFERENCES
    Angestellter(PersNr);
```

Referenzielle Integrität

```
CREATE TABLE foo( ...
ON DELETE [CASCADE | RESTRICT | SET NULL |
SET DEFAULT]);
```

Tabelle löschen

```
DROP TABLE foo [CASCADE]
```

DML: Data Manipulation Language

```
INSERT INTO Abteilung (Name, AbtrNr)
VALUES ('Entwicklung', 20);
```

DQL: Data Query Language

WHERE, ORDER BY, LIKE

```
SELECT Name, Wohnort FROM Angestellter
WHERE AbtNr = 1
  AND (Salaer > 500)
  OR WOHNORT IN ('Luzern', 'Zug') -- = any
  OR FOO LIKE 'Zu% OR FOO LIKE 'L____n'
ORDER BY Name;
```

Window Function

```
SELECT DISTINCT abtnr, SUM(salaer)
OVER (PARTITION BY abtnr)
FROM angestellter ORDER BY 1;
```

Common Table Expressions (CTE)

```
WITH tmp_tbl AS (SELECT * FROM tabelle
WHERE number = 23),
SELECT name, foo from tmp_tmp_tbl;
```

Weitere Keywords

`DISTINCT`: Duplikate unterdrücken

Transaktionen

→ Bei **Repeatable Read** wird nichts geschrieben, was auf andere Transaktionen Einfluss hat.

→

JDBC by Example: SQL-Queries mit Java

Start Connection Try-Block

```
final String u = "username";
final String p = "password";
final String d =
  "jdbc:postgresql://localhost/db_name";
try(Connection c =
  DriverManager.getConnection(d,u,p)){
c.setAutoCommit(false);
c.setTransactionIsolation
  (Connection.TRANSACTION_SERIALIZABLE);
```

Plain SQL: Normales Statement (SQLi Gefahr!)

```
try(Statement s = conn.createStatement()){
  ResultSet rs1 = stmt.executeQuery
    ("SELECT CURRENT_USER");
  while (rs1.next())
    System.out.println(rs1.getString(1));
  rs1.close(); }
```

Prepared Statement (SQLi Schutz)

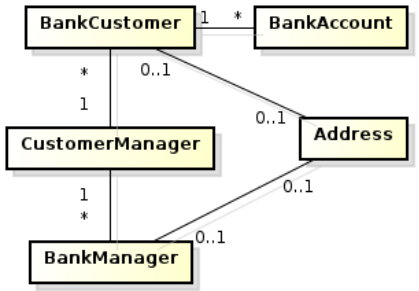
```
final String query = "SELECT *
  FROM customers WHERE name LIKE ?"
try (PreparedStatement ps =
    c.prepareStatement(query)){
  ps.setString(1, readInput());
  ResultSet rs2 = ps.executeQuery();
  while (rs.next())
    String n = rs2.getString("NAME");
    float g = rs2.getFloat("GEHALT");
    System.out.println(n + ": CHF " + g);
  }}
```

Ende Connection Try-Block

```
} catch (SQLException e) {
  Connection.rollback();
  System.err.println(e.getMessage()); }
```

OR Mapping mit JPA

Domain Model



db_schema.sql

```
CREATE TABLE BankCustomer(
  CustomerId SERIAL NOT NULL PRIMARY KEY,
  Name TEXT NOT NULL, Birthdate DATE,
  Customer_AddressId INTEGER);
CREATE TABLE Address(
  AddressId SERIAL NOT NULL PRIMARY KEY,
  Street TEXT NOT NULL, Zip INTEGER,
  City TEXT NOT NULL);
CREATE TABLE BankAccount(
  AccountId SERIAL NOT NULL PRIMARY KEY,
  Account_CustomerId INTEGER NOT NULL,
  Balance DOUBLE PRECISION NOT NULL,
  Currency TEXT NOT NULL DEFAULT 'CHF');
CREATE TABLE BankManager(
  ManagerId SERIAL NOT NULL PRIMARY KEY,
  Name TEXT NOT NULL,
  Manager_AddressId INTEGER);
CREATE TABLE CustomerManager(
  CustomerId INTEGER NOT NULL,
  ManagerId INTEGER NOT NULL,
  PRIMARY KEY(CustomerId, ManagerId));
```

Java-Klassen (keine Klasse CustomerManager)

Tabellen und Attribute Mappen

```
@Entity
@Table(name = "bankcustomer")
public class BankCustomer {
  @Id
  @Column (name = "customerid")
  private Integer id;
  private String name; // Auto Mapping
  @Temporal(TemporalType.TIMESTAMP)
  private Date birthdate;
  @Transient // Nicht Mappen
  private Integer foo;
```

Beziehung: ManyToOne (B.Account – B.Customer)

→ BankAccount.java

```
@ManyTo One(optional = false)
@JoinColumn(name = "account_customerid")
private BankCustomer bankcustomer;
```

→ BankCustomer.java

```
@OneToMany
@JoinColumn(name = "account_customerid",
  referencedColumnName = "customerid")
private Collection<BankAccount>
  accounts = new ArrayList<>();
```

Beziehung: OneToOne (Address – B.Customer)

→ Address.java

```
@OneToOne(mappedBy = "address") // Member
private BankManager bankmanager;
```

→ BankManager.java

```
@OneToOne(optional = true)
@JoinColumn(name = "manager_addressid")
private Address address;
```

Beziehung: ManyToMany (B.Customer – B.Manager)

→ BankManager.java (Attribut managerid zuerst!)

```
@ManyToMany
@JoinTable(name = "customermanager",
  joinColumns =
    {@JoinColumn(name = "managerid")},
  inverseJoinColumns =
    {@JoinColumn(name = "customerid")})
private Collection<BankCustomer>
  customers = new ArrayList<>();
```

→ BankCustomer.java

```
@OneToMany
@JoinColumn(name = "account_customerid",
  referencedColumnName = "customerid")
private Collection<BankAccount>
  accounts = new ArrayList<>();
```

Aggregation / Komposition

→ Aggregation (Leere Raute)

```
@OneToMany(cascade =
  CascadeType.PERSIST, ...)
private Collection<BankAccount>
  accounts = new ArrayList<>();
```

→ Komposition (Ausgefüllte Raute)

```
@OneToMany(cascade =
  {CascadeType.PERSIST,
    CascadeType.REMOVE }, ...)
private Collection<BankAccount>
  accounts = new ArrayList<>();
```

Vererbung in JPA (RetailB.M extends B.M)

Single Table (Tabelle für Oberklasse)

→ BankCustomer.java

```
@Entity
@Inheritance(strategy =
  InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "type")
public abstract class BankCustomer { ...
```

→ RetailBankCustomer.java (erbt von BankCustomer)

```
@Entity
@DiscriminatorValue("Retail")
public class RetailBankCustomer extends ..
```

Joined Table (Tabelle pro Klasse: Jede Klasse)

→ BankManager.java

```
@Entity
@Inheritance(strategy =
  InheritanceType.JOINED)
@DiscriminatorColumn(name = "type")
public abstract class BankCustomer { ...
```

→ RetailBankCustomer.java

```
@Entity
@DiscriminatorValue("Retail")
public class RetailBankCustomer extends ..
```

Table per Class (Tabelle pro Subklasse)

→ BankManager.java

```
@Entity
@Inheritance(strategy =
  InheritanceType.TABLE_PER_CLASS)
public abstract class BankCustomer { ...
```

→ RetailBankCustomer.java

```
@Entity
public class RetailBankCustomer
extends BankCustomer { ...
```

JPQL (Abfragen auf Entities statt DB Modell)

```
Query q =
  em.createQuery("select c from
BankCustomer c join c.accounts a");
```

Diverses

Last Change

2013-01-17