Kap. 2: Relationale Zugriffsschnittstellen

- **Static SQL**: SQL ist in die Programmiersprache eingebettet. Syntaxprüfung möglich!
  SQL-Anweisungen werden zur Kompilierzeit kompiliert.
- **Dynamic SQL:** SQL wird als Strings in der Programmiersprache verwendet!
  SQL-Anweisungen werden zur Laufzeit kompiliert.
- **Datenbankintern**: Die Datenbank speichert Funktionen (mit Rückgabetyp) oder Prozeduren (ohne Rückgabetyp) (PL/SQL)

Handhabung von NULL-Werten

```
if (rs.wasNull()) { ... }
```

JDBC – Aufbau einer Verbindung

- teurere Operation (1x zum Programm start!)

DDL-Befehl

```
public static void tabelleerstellen() throws SQLException {
    String createOrderItems =
            "CREATE TABLE order_items(" +
                    "     order_id NUMBER(8), " +
                    "     name VARCHAR2(100),"
                    "     PRIMARY KEY (order_id, name))";
    try (Statement stmt = conn.createStatement()) {
        stmt.executeUpdate(createOrderItems);
    }
}
```

DML-Befehl

- *Wichtig: Rückgabewert von `executeUpdate()` : Anzahl der geänderten Datensätze*

```
public static void prepareStatement (int id, String name) throws SQLException {
    String insertItem = "INSERT INTO rezept VALUES (?, ?)";
    try (PreparedStatement stmt = conn.prepareStatement(insertItem)) {
        stmt.setInt(1, id);
        stmt.setString(2, name);
        stmt.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

DQL im Detail

```
public static void rezeptAusgeben(int id) throws SQLException {
   try (Statement stmt = conn.createStatement()) {
       String query = "SELECT first_name, last_name, salary "
               + " FROM hr.employees WHERE salary > 5000";
        try (ResultSet rs = stmt.executeQuery(query)) {
            while (rs.next()){
                String last_name = rs.getString("last_name");
                double sal = rs.getDouble(3);
                System.out.println(last_name + "\t" + sal);
            }
        }
    } catch(SQLException e) {
        e.printStackTrace();
    }
}
```

Erzeugung von Primärschlüsseln

Primärschlüssel Bedingung: müssen eindeutig sein & unveränderlich

```
create sequence employee_seq;
select employee_seq.nextval from dual;
insert into employee values (employee_seq.nextval, ...);
```

Probleme mit JDBC: Code evtl. schwer zu warten

Active Record

- Es werden Entities für jede Relation erstellt
- Um Entities von der Datenbank zu erzeugen, wird zu jedem Entitytyp eine Factory erstellt, oder jede Entity hat statische Methoden welche dann die Entity erzeugen
- Die technischen Details des DB-Zugriffs (SQL, JDBC) sind in der darunterliegenden Persistenzschicht zusammengefasst

Vorteile: Einfaches Muster; einfache Kapselung der SQL-Zugriffe
Nachteile: Immer noch redundanter Code; keine wirkliche Trennung zwischen Persistenzschicht und Geschäftslogik

Performance-Aspekte:

- **Schichtentrennung** führt ggf. zu ineffizienten SQL-Abfolgen.
- Nur optimieren, wenn nötig

```java
public void ausleihen(int fahrrad_id, int kunde_id) {
    boolean ok = false;
    String SQL = "SELECT K_id FROM Fahrrad WHERE F_id = ?";
    try {
        try (PreparedStatement stmt = conn.prepareStatement(SQL)) {
            stmt.setLong(1, fahrrad_id);
            try (ResultSet rs = stmt.executeQuery()) {
                rs.next();
                rs.getInt("K_id");

                if (!(rs.wasNull())) {
                    throw new SQLException("Fahrrad ist nicht verfuegbar!");
                }
            }
        }
        SQL = "UPDATE Fahrrad SET FK-K_id = ? WHERE PK-F_id = ?";

        try (PreparedStatement stmt = conn.prepareStatement(SQL)) {
            stmt.setString(1, kunde_id);
            stmt.setLong(2, fahrrad_id);
        }
        conn.commit();
        ok = true;
    } finally {
        if(!ok)
            conn.rollback();
    }
}
```

Transaktionen

- **Eine Transaktion ist eine Menge von DB-Operationen, die eine DB von einem konsistenten Zustand in einen weiteren konsistenten Zustand überführt.**
- Mehrere SQL Befehle müssen als Einheit ausgeführt werden!
- Entweder alle Transaktionen oder gar keine!

→ **Fehlerhafte Datenbankzustände werden ausgeschlossen → Datenbankintegrität**

Nutzen von Transaktionen (Zwei Aufgabenblöcke für das Transaktionsmanagement)

- **Synchronisation** - Organisation des Mehrbenutzerbetriebs
- **Recovery** - Wiederherstellung konsistenter Zustände nach Fehlern

ACID-Eigenschaften von Transaktionen

Atomicity / Unteilbarkeit:	Transaktion ganz oder gar nicht durchführen
Consistency / Konsistenz:	Nach Transaktion ist DB in konsistent Zustand
Isolation / Isolation:	Parallele Transaktion beeinflussen sich nicht
Durability / Dauerhaftigkeit:	Wirkung einer Transaktion ist dauerhaft

Parallele Transaktionen

- **Mehrbenutzerbetrieb** kann zu inkonsistenten Daten führen:
  - **Synchronisationsprobleme**
- **Lost Update**: X hat falschen Wert, da Aktualisierung durch T1 verloren
- **Dirty Read:** X muss auf den alten Wert zurückgesetzt werden; T2 hat den falschen Wert schon verarbeitet
- **Non-Repeatable Read:** Am Ende besitzt X neuen Wert. Ein Vergleich auf Basis des ersten Lesens ist nicht möglich
- **Falsche Summenbildung:** T2 liest X nach Subtraktion durch T1 und Y vor Addition
- **Unterschiedliche Operationenreihenfolge**
- **Phantome**

→ **Lösung: Transaktionen**

JDBC - Transaktionen

Connection-Objekt bestimmt Transaktionssteuerung

1) Auto-Commit abschalten: *Das sollten Sie sich grundsätzlich angewöhnen!* `conn.setAutoCommit(false);`

- Transaktionen starten, wenn ein Befehl an die Datenbank gesendet wird
- Transaktion wird mit `conn.commit()` oder `conn.rollback()` beendet.
- Fehler / Exceptions beachten: Immer für Transaktionsende sorgen!

```
public static void complexBusinessMethod() throws SQLException {
     boolean ok = false;
     try {
          Person person = new Person();
          ...
          person.insert();

          Movie movie = new Movie();
          ...
          movie.insert();

          ConnectionManager.getConnection().commit();
          ok = true;
     } finally {
          if (!ok)
              ConnectionManager.getConnection().rollback();
     }
}
```