Programmierung 3

im Wintersemester 2024/25

Praktikumsaufgabe 7

Mehr Funktionalität durch Bibliotheken und Frameworks

3. Dezember 2024

In diesem Arbeitsblatt verwenden Sie JDBC und jOOQ, um von Ihrem Java-Code aus auf eine Datenbank zuzugreifen. Dabei verwenden Sie Maven, um die Abhängigkeiten zu verwalten und das Projekt zu konfigurieren.

In Aufgabe 1 greifen Sie mit JDBC, also mit den Bordmitteln von Java, auf eine Datenbank zu.

In Aufgabe 2 migrieren Sie Ihren Code nach jOOQ, damit Sie SQL-Statements typsicher in Java schreiben können.

In Aufgabe 3 können Sie die SQL-Beispiele aus der Paralleveranstaltung Datenbanken mit jOOQ lösen.

Arbeiten Sie bei der Lösung der Aufgaben in einem Team von 3 bis 4 Personen zusammen, so dass Sie die Aufgaben kollaborativ lösen und Ihre Ergebnisse untereinander diskutieren können. Lösen Sie dabei die Programmieraufgaben im Pair Programming und bearbeiten Sie die anderen Aufgaben in einer für Sie geeigneten Weise.

1 SQL-Anfragen mit JDBC

Lernziele

- ☐ Sie senden SQL-Anfragen an eine Datenbank und verarbeiten die Ergebnisse in Java.
- ☐ Sie verwenden die Java Database Connectivity (JDBC) API, um eine Verbindung zu einer Datenbank herzustellen und SQL-Anfragen auszuführen.
- ☐ Sie nutzen JDBC, um Daten aus einer Datenbank abzufragen, einzufügen und zu aktualisieren.

In dieser Aufgabe nutzen Sie die Java Database Connectivity (JDBC) API, um SQL-Anfragen an eine Datenbank zu senden.

Heads up! Als Datenbank verwenden Sie in dieser und in den folgenden Aufgaben eine SQLite-Datenbank, wie Sie sie bereits aus der Veranstaltung Datenbanken von Prof Dr.-Ing. Klaus Berberich kennen.

1.1 Datenbank vorbereiten

Nutzen Sie folgende SQL-Statements, um die Datenbank `sample-database.sqlite3` zu erstellen:[1]

```
CREATE TABLE language (
  id              NUMBER(7)      NOT NULL PRIMARY KEY,
  cd              CHAR(2)        NOT NULL,
  description     VARCHAR2(50)
);

CREATE TABLE author (
  id              NUMBER(7)      NOT NULL PRIMARY KEY,
  first_name      VARCHAR2(50),
  last_name       VARCHAR2(50)   NOT NULL,
  date_of_birth   DATE,
  year_of_birth   NUMBER(7),
  distinguished   NUMBER(1)
);

```

[1] Diese Datenbank entspricht der Beispieldatenbank aus der Dokumentation von jOOQ.

```
16 CREATE TABLE book (
17   id               NUMBER(7)      NOT NULL PRIMARY KEY,
18   author_id        NUMBER(7)      NOT NULL,
19   title            VARCHAR2(400) NOT NULL,
20   published_in     NUMBER(7)      NOT NULL,
21   language_id      NUMBER(7)      NOT NULL,
22
23   CONSTRAINT fk_book_author     FOREIGN KEY (author_id)   REFERENCES
      ↪  author(id),
24   CONSTRAINT fk_book_language   FOREIGN KEY (language_id) REFERENCES
      ↪  language(id)
25 );
26
27 CREATE TABLE book_store (
28   name             VARCHAR2(400) NOT NULL UNIQUE
29 );
30
31 CREATE TABLE book_to_book_store (
32   name             VARCHAR2(400) NOT NULL,
33   book_id          INTEGER        NOT NULL,
34   stock            INTEGER,
35
36   PRIMARY KEY(name, book_id),
37   CONSTRAINT fk_b2bs_book_store FOREIGN KEY (name)        REFERENCES
      ↪  book_store (name) ON DELETE CASCADE,
38   CONSTRAINT fk_b2bs_book       FOREIGN KEY (book_id)     REFERENCES book
      ↪  (id)          ON DELETE CASCADE
39 );
```

Befüllen Sie die Datenbank mit folgenden SQL-Statements:[2]

```
1 INSERT INTO language (id, cd, description) VALUES (1, 'en', 'English');
2 INSERT INTO language (id, cd, description) VALUES (2, 'de', 'Deutsch');
3 INSERT INTO language (id, cd, description) VALUES (3, 'fr', 'Français');
4 INSERT INTO language (id, cd, description) VALUES (4, 'pt', 'Português');
5
6 INSERT INTO author (id, first_name, last_name, date_of_birth,
   ↪  year_of_birth)
7   VALUES           (1 , 'George'  , 'Orwell' , '1903-06-26' , 1903
     ↪  );
8 INSERT INTO author (id, first_name, last_name, date_of_birth,
   ↪  year_of_birth)
```

[2]Der einzige Unterschied zu den Statements aus der jOOQ-Dokumentation ist, dass hier auf `DATE` verzichtet wird, da SQLite die `DATE`-Notation nicht unterstützt (SQLite hat keine eigene `DATE`-Datentypunterstützung).

```
9    VALUES           (2 , 'Paulo'   , 'Coelho' , '1947-08-24' , 1947
      ↪  );
10
11  INSERT INTO book (id, author_id, title         , published_in,
     ↪  language_id)
12    VALUES         (1 , 1         , '1984'       , 1948        , 1
      ↪  );
13  INSERT INTO book (id, author_id, title         , published_in,
     ↪  language_id)
14    VALUES         (2 , 1         , 'Animal Farm' , 1945        , 1
      ↪  );
15  INSERT INTO book (id, author_id, title         , published_in,
     ↪  language_id)
16    VALUES         (3 , 2         , 'O Alquimista', 1988        , 4
      ↪  );
17  INSERT INTO book (id, author_id, title         , published_in,
     ↪  language_id)
18    VALUES         (4 , 2         , 'Brida'      , 1990        , 2
      ↪  );
19
20  INSERT INTO book_store VALUES ('Orell Füssli');
21  INSERT INTO book_store VALUES ('Ex Libris');
22  INSERT INTO book_store VALUES ('Buchhandlung im Volkshaus');
23
24  INSERT INTO book_to_book_store VALUES ('Orell Füssli'             , 1,
     ↪  10);
25  INSERT INTO book_to_book_store VALUES ('Orell Füssli'             , 2,
     ↪  10);
26  INSERT INTO book_to_book_store VALUES ('Orell Füssli'             , 3,
     ↪  10);
27  INSERT INTO book_to_book_store VALUES ('Ex Libris'                , 1, 1
     ↪  );
28  INSERT INTO book_to_book_store VALUES ('Ex Libris'                , 3, 2
     ↪  );
29  INSERT INTO book_to_book_store VALUES ('Buchhandlung im Volkshaus', 3, 1
     ↪  );
```

1.2 Maven-Projekt einrichten

Richten Sie ein Maven-Projekt ein, das die für die Nutzung einer SQLite-Datenbank benötigten Abhängigkeiten enthält.

Tipp: Eine gängige Bibliothek für die Verwendung von SQLite mit JDBC in Java ist SQLiteJDBC, entwickelt von Xerial. Diese Bibliothek ermöglicht es Java-Anwendungen, auf SQLite-Datenbanken zuzugreifen und diese zu erstellen, ohne dass eine separate Konfiguration erforderlich ist. Sie enthält alle notwendigen nativen SQLite-Bibliotheken für verschiedene Betriebssysteme (Windows, macOS, Linux) in einer einzigen JAR-Datei.

1.3 Eine einfache Abfrage ausführen

Formulieren Sie eine SQL-Abfrage, die die Titel der Bücher sowie die Vor- und Nachnamen der jeweiligen Autoren aus den Tabellen `BOOK` und `AUTHOR` ermittelt. Die Tabellen sollen so verknüpft werden, dass die `AUTHOR_ID` in der Tabelle `BOOK` mit der `ID` in der Tabelle `AUTHOR` übereinstimmt. Filtern Sie die Ergebnisse so, dass nur Bücher berücksichtigt werden, die im Jahr 1948 veröffentlicht wurden.

Eine passende SQL-Abfrage könnte wie folgt aussehen:

```
SELECT BOOK.TITLE, AUTHOR.FIRST_NAME, AUTHOR.LAST_NAME
FROM BOOK
JOIN AUTHOR
  ON BOOK.AUTHOR_ID = AUTHOR.ID
WHERE BOOK.PUBLISHED_IN = 1948;
```

Führen Sie Ihre SQL-Abfrage mit JDBC gegen die SQLite-Datenbank aus und geben Sie die Ergebnisse auf der Konsole aus.

Hinweis: Erstellen Sie dazu die Klasse `JdbcTestDrive` mit einer `main`-Methode, die die SQL-Abfrage ausführt.

1.3.1 Die Abfrage mit Parametern ausführen

Erweitern Sie Ihre Implementierung so, dass das Jahr von der Kommandozeile als Argument übergeben werden kann. Passen Sie die SQL-Abfrage entsprechend an, sodass das Jahr dynamisch aus dem Argument gelesen wird.

1.4 Daten einfügen und aktualisieren

Fügen Sie eine neue Autor:in in die Tabelle `AUTHOR` ein. Die Autor:in kann z. B. die folgenden Eigenschaften haben:

ID `1`

Vorname `"Herta"`

Nachname `"Müller"`

Distinguished `1`

Falls eine Autor:in mit der `ID` bereits existiert, soll der bestehende Datensatz aktualisiert werden, so dass die Werte für Vorname, Nachname und Distinguished überschrieben werden.

Hinweis: Verwenden Sie geeignete `System.out`-Ausgaben, um den Erfolg des Einfügens bzw. der Aktualisierung zu überprüfen.

1.4.1 Das Einfügen und Aktualisieren mit Parametern ausführen

Erweitern Sie Ihre Implementierung so, dass alle 4 Werte für von der Kommandozeile als Argumente übergeben werden können. Passen Sie die SQL-Statements entsprechend an, so dass die Werte dynamisch aus den Argumenten gelesen werden.

2 Typsicherheit und Fluent Interfaces mit jOOQ

Lernziele

- ☐ Sie verwenden jOOQ, um SQL-Anfragen an eine Datenbank zu senden.
- ☐ Sie generieren jOOQ-Klassen, um typsichere SQL-Abfragen zu erstellen und auszuführen.
- ☐ Sie nutzen das Fluent Interface von jOOQ, um SQL-Abfragen zu formulieren.

In dieser Aufgabe verwenden Sie jOOQ Object Oriented Querying (jOOQ), um SQL-Anfragen an eine Datenbank zu senden. Hierfür ersetzen Sie die JDBC-Abfragen aus der vorherigen Aufgabe durch jOOQ-Abfragen.

2.1 Datenbank vorbereiten

Verwenden Sei die gleiche SQLite-Datenbank wie in der vorherigen Aufgabe. Gegebenenfalls kann es hilfreich sein, die Datenbank zu löschen und neu zu erstellen.

2.2 Maven-Projekt einrichten

Erweitern Sie das Maven-Projekt aus der vorherigen Aufgabe um die Abhängigkeiten für jOOQ.

Hinweis: Nutzen Sie das Kapitel jOOQ in 7 easy steps aus der jOOQ-Dokumentation, um die benötigten Abhängigkeiten zu identifizieren.

2.3 Typsicheres SQL

Wenn auch optional, ist die automatische Code-Generierung mit jOOQ ein mächtiges Werkzeug, um typsichere SQL-Abfragen zu erstellen. Führen Sie die Code-Generierung mit jOOQ aus, um die benötigten Klassen für die Tabellen aus der Datenbank `sample-database.sqlite3` zu generieren.

Hinweis: Nutzen Sie das Kapitel Code Generation aus der jOOQ-Dokumentation, um die Code-Generierung zu konfigurieren. Insbesondere das Unterkapitel Running the code generator with Maven ist hierbei hilfreich.

2.4 Eine einfache Abfrage ausführen

Migrieren Sie die Abfrage aus der vorherigen Aufgabe von JDBC nach jOOQ. Verwenden Sie die generierten Klassen, um die Abfrage typsicher zu formulieren.

Tipp: Sie brauchen hierzu jOOQ-Klassen wie `DSLContext` und `Result`.

2.4.1 Reflexion im Team

Diskutieren Sie im Team, inwiefern das Fluent Interface von jOOQ die Lesbarkeit und Wartbarkeit Ihres Codes im Vergleich zu JDBC verbessert.

2.5 Daten einfügen und aktualisieren

Migrieren Sie die Datenbearbeitungen der vorherigen Aufgabe von JDBC nach jOOQ. Verwenden Sie die generierten Klassen, um die Operationen typsicher zu formulieren.

Tipp: Hier wird die von jOOQ generierte Klasse `AuthorRecord` sehr hilfreich sein.

2.5.1 Reflexion im Team

Vergleichen Sie im Team die Lösung mit jOOQ mit der Lösung mit JDBC. Diskutieren Sie, welche Vorteile und Nachteile Sie in der Verwendung von jOOQ sehen.

3 Musicbrainz aus der Veranstaltung Datenbanken

Strecklernziele

- ☐ Sie verwenden jOOQ, um SQL-Abfragen zu formulieren und auszuführen.
- ☐ Sie verwenden Selektionen (wie `SELECT`), um Daten auszuwählen, die bestimmte Kriterien erfüllen.
- ☐ Sie verwenden Joins (wie `JOIN`), um Daten aus mehreren Tabellen auf der Grundlage von Schlüsselspalten zu kombinieren.
- ☐ Sie verwenden Aggregation (wie `COUNT`), um Summen, Durchschnittswerte, Anzahlen usw. zu berechnen.
- ☐ Sie verwenden Mengenoperationen (wie `INTERSECT`), um mit Ergebnismengen zu arbeiten, z. B. Vereinigung, Schnittmenge, Differenz.
- ☐ Sie verwenden Sortierung (wie `DESC`), um Ergebnisse auf- oder absteigend zu sortieren.
- ☐ Sie verwenden Deduplizierung (wie `DISTINCT`), um Duplikate zu entfernen.
- ☐ Sie verwenden Bedingungen (wie `LIKE`), um Daten zu filtern.

In dieser Aufgabe können Sie die Beispielaufgaben aus der Veranstaltung Datenbanken mit jOOQ lösen. So können Sie den Umgang mit jOOQ in der Praxis üben.

Hinweis: Dazu benötigen Sie die SQLite-Datenbank `musicbrainz.sqlite`, die Sie wie die Beispielaufgaben im Moodle-Kurs des Kurses finden.

3.1 SQL-Abfragen

Implementieren Sie folgende SQL-Abfragen mit jOOQ![3]

1. Listen Sie den Namen und Anzahl der Lieder aller Alben auf, die zwischen 1990 und 2000 erschienen sind und mehr als 15 Lieder beinhalten.

2. Geben Sie Name und Jahr aller Alben aus, die von der Gruppe “The Beatles” sind.

[3] Diese sind 1:1 aus den Beispielaufgaben von Prof Dr.-Ing. Klaus Berberich übernommen.

3. Geben Sie Namen von Künstlern aus, die sowohl bei männlichen Künstlern als auch weiblichen Künstlern vorkommen.

4. Listen Sie die Namen aller Künstler auf, die ein Album haben, was einen Song mit dem Titel “One Love” enthält.

5. Erzeugen Sie eine Übersicht über alle Verläge die “dream” oder “magic” im Namen haben. In der Übersicht soll Name und Location aufgeführt sein.

6. Geben Sie die Namen und die aktiven Jahre der 10 Bands (Group) aus, die am längsten Musik gemacht haben.

7. Listen Sie die Namen und Erscheinungsjahre aller Alben von Metallica auf, sortiert nach dem Erscheinungsjahr. Es sollen dabei keine Duplikate vorkommen

8. Location absteigend sortiert nach der Anzahl der Künstler

9. Bestimmen Sie die durchschnittliche Songlänge pro Band (Gruppe). Geben Sie die 15 Bands mit den kürzesten Songs aus

10. Geben Sie alle Alben an, die von Verlag DreamWorks Records veröffentlicht wurden.

11. Welcher Verlag hat insgesamt die meisten Alben veröffentlicht? (Alben haben eine eindeutige RecordID) Geben Sie den Namen des Verlags und die Anzahl der Alben an.