Prof. Dr. Jörg Hettel

Datenbankprogrammierung
Arbeitspaket 1

Aufgabe 1 – Einfacher SELECT

Implementieren Sie ein einfaches Java-Programm, das mit Hilfe von JDBC die Namen aller in der Datenbank vorhandenen Verlage ausgibt.

Aufgabe 2 – SQL-Injection

Entwickeln Sie nun ein einfaches Java-Programm, das mit Hilfe von JDBC Autoren in der Datenbank sucht. Als Suchkriterium soll der (exakte) Nachnamen benutzt werden.

Lagern Sie die Suche in eine Methode aus, die die **`Statement`**-Klasse für die Query benutzt:

```
public static List<String> findAutor(String autorname)
{
  Objects.requireNonNull(autorname);

  String sql = "SELECT ... FROM Autor a WHERE a.nachname = '" + autorname + "'";

  ...

  return verlage;
}
```

Mit welchem Parameter könnte die Methode aufgerufen werden, damit sie alle Autoren zurückliefert?

Implementieren Sie nun eine zweite Variante, die ein **`PreparedStatement`** benutzt und vergewissern Sie sich, dass der SQL-Injektion-Angriff nun nicht mehr funktioniert.

Aufgabe 3 – Meta-Information

Über die MetaDaten der Datenbank-Connection können verschiedene Informationen über die Datenbank abgefragt werden. Schreiben ein kleines Java-Programm, das folgende Informationen über die Datenbank ausgibt:

- Datenbankname und Version
- Version des JDBC-Treibers
- Unterstützung von Batch-Updates
- Isolation-Level der Verbindung

Aufgabe 4 – INSERT-Benchmark

Das Einfügen großer Datenmengen (viele Zeilen) kann auf verschiedene Arten erfolgen.

1. Mit Hilfe der Statement-Klasse im Autocommit-Modus
2. Mit Hilfe der Statement-Klasse mit autocommit=false
3. Mit Hilfe der PreparedStatement-Klasse im Autocommit-Modus
4. Mit Hilfe der PreparedStatement -Klasse mit autocommit=false
5. Mit Hilfe eines Batch-Updates (im Autocommit-Modus)
6. Mit Hilfe eines Batch-Updates mit autocommit=false

Schreiben Sie nun einen kleinen Performance-Benchmark, in dem diese fünf Arten verglichen werden. Speichern Sie jeweils immer neue 2.000 Adressen (Ortsnamen) in der Adresse-Tabelle.

Aufgabe 5 – Isolation-Level

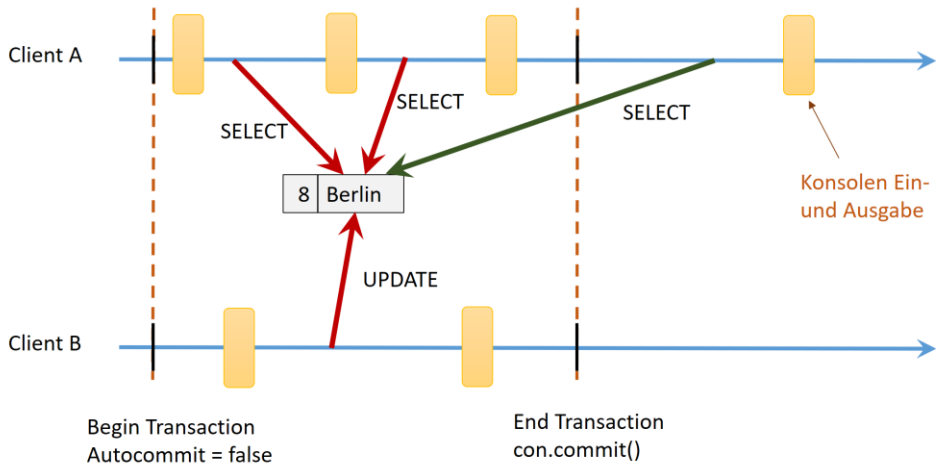
Greifen gleichzeitig mehrere Clients (Programme) auf die Datenbank zu, so müssen diese konkurrierende Zugriffe koordiniert werden, insbesondere wenn mehrere SQL-Anweisungen in einer Transaktion zusammengefasst werden (autocommit = false).

Mit Hilfe eines Isolation-Levels kann ein Client dabei die „Datengüte“ seiner gelesenen Daten bestimmen bzw. die Sichtbarkeit seiner Änderungen innerhalb einer Transaktion. In der Regel werden von den Datenbanken folgende Isolation-Levels unterstützt:

- TRANSACTION_READ_UNCOMMITTED
- TRANSACTION_READ_COMMITTED
- TRANSACTION_REPEATABLE_READ
- TRANSACTION_SERIALIZABLE

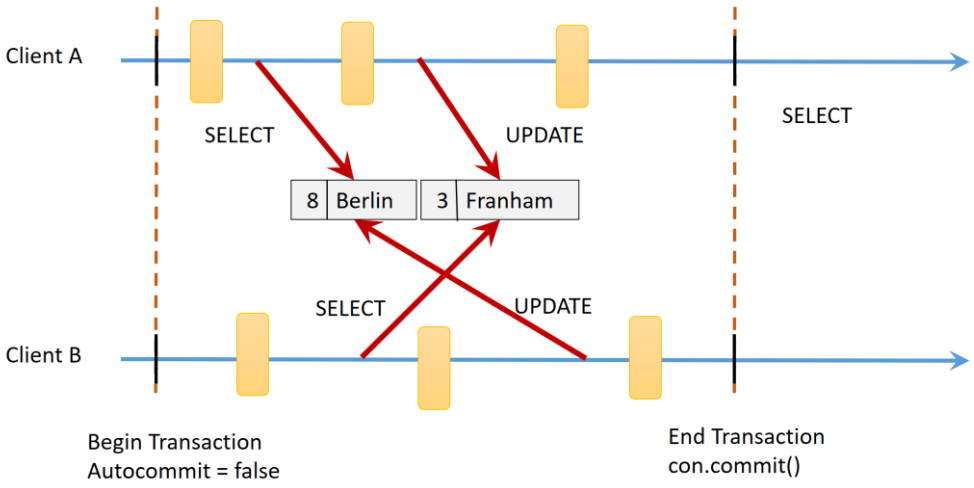
Implementieren Sie zwei einfache Client-Anwendungen und führen Sie folgende Experimente durch. Der zeitliche Verlauf ist jeweils von links nach rechts. Der Zugriff erfolgt auf die Adresse-Tabelle in der Datenbank. Zur Simulation der zeitlichen Verschränkungen können Sie an den gezeigten Stellen (helle Rechtecke) jeweils Konsolen Aus- und Eingaben einbauen.

Experiment 1: Client A liest in einer Transaktion zweimal den Datensatz mit der ID 8. Parallel dazu führt Client B ein Update des Ortsnamens durch.

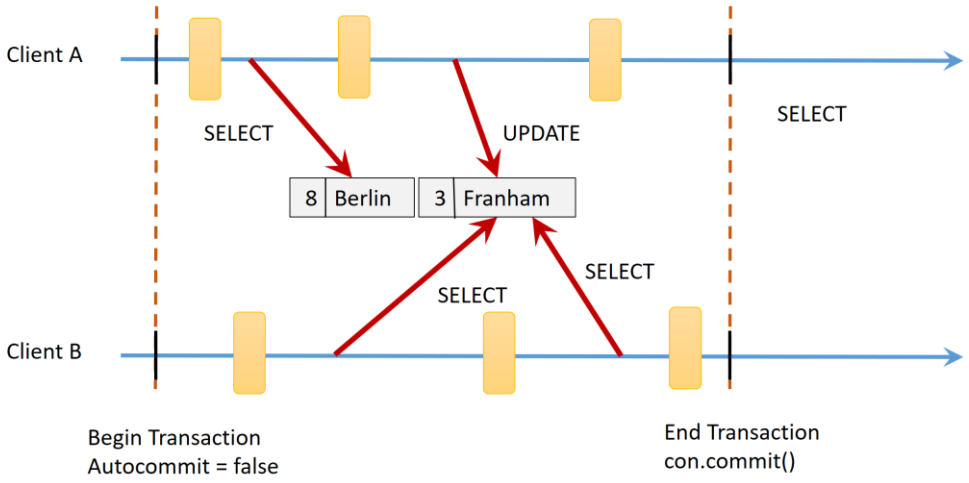


Datenbankprogrammierung
Arbeitspaket 1

Experiment 2: Client A liest einen Datensatz (ID = 8) und möchte einen anderen Datensatz (ID = 3) ändern. Client B macht dasselbe nur mit vertauschter Reihenfolge. Er liest zuerst den Datensatz mit ID=3 und verändert anschließen den Datensatz mit der ID = 8.



Experiment 3: Client A liest einen Datensatz (ID = 8) und möchte einen anderen Datensatz (ID = 3) ändern. Client B liest zuerst den Datensatz zweimal in einer Transaktion den Datensatz mit ID=3.



Variieren Sie dabei jeweils die Isolation-Level der beiden Clients. Können Sie das Verhalten der Clients bzw. deren Ausgaben erklären?

Datenbankprogrammierung
Arbeitspaket 1

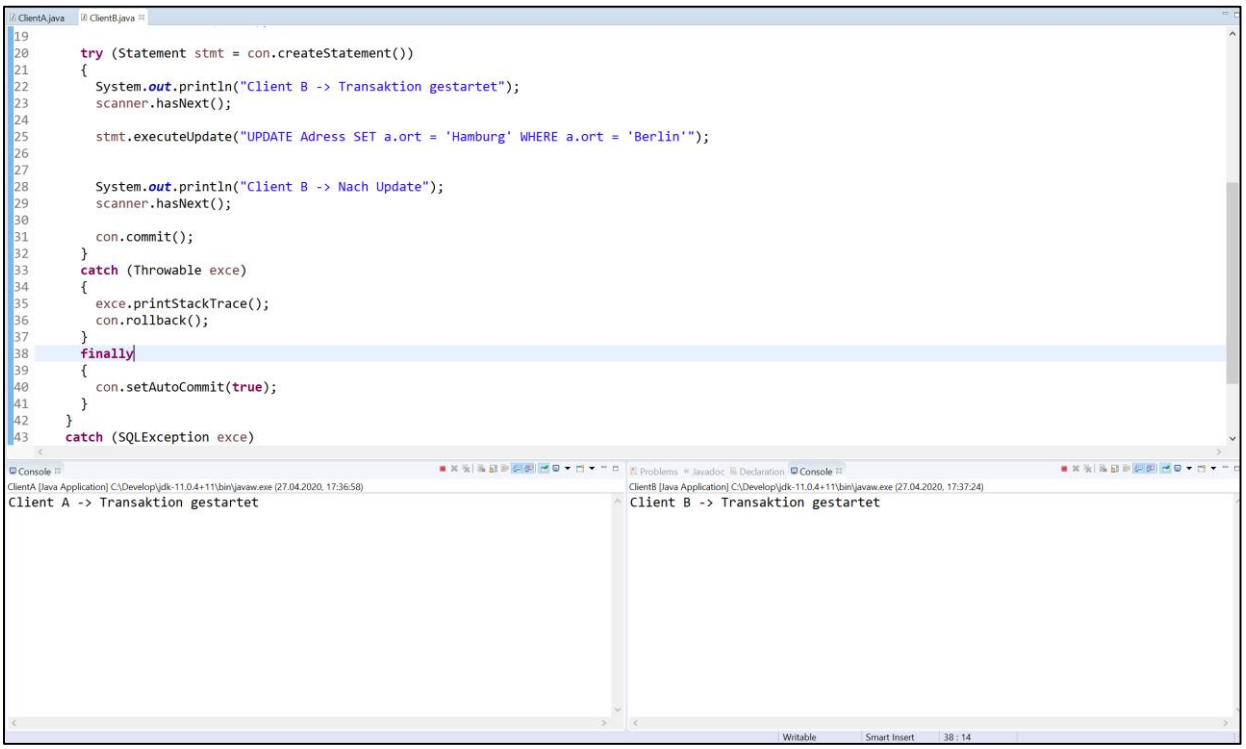
Bemerkungen zum Arbeiten mit Eclispe

Die Konsolen Eingabe kann z.B. mit einem Scanner realisiert werden. Hier eine Beispielimplementierung für den Client B aus dem Experiment 1:

```
public static void main(String[] args)
{
  Scanner scanner = new Scanner(System.in);

  try (Connection con = ConnectionManager.getInstance().getNewConnection())
  {
    con.setAutoCommit(false);

    try (Statement stmt = con.createStatement())
    {
      System.out.println("Client B -> Transaktion gestartet");
      scanner.nextLine();

      System.out.println("Client B -> Führt UPDATE aus");
      int count = stmt.executeUpdate(
                  "UPDATE Adresse a SET a.ort = 'Hamburg' WHERE a.ort = 'Berlin'");
      System.out.println("Änderungen " + count );

      System.out.println("Client B -> Warte nach Update auf Commit");
      scanner.nextLine();

        con.commit();
    }
    catch (Throwable exce)
    {
      exce.printStackTrace();
      con.rollback();
    }
    finally
    {
      con.setAutoCommit(true);
    }
  }
  catch (SQLException exce)
  {
    exce.printStackTrace();
  }
}
```

Datenbankprogrammierung
Arbeitspaket 1

Eclispe erlaubt es, dass auch zwei Konsolen parallel angezeigt werden.



Öffnen einer neuen Konsole, die dann frei platziert werden kann. Die Ausgabe eines Prozesses muss dann anschließen explizit an eine der geöffneten Konsolen gebunden werden.

Menüleiste der Konsole: