UE (GK) Transaktionen

✔ **Done:** View | **To do:** Make a submission | ✔ **Done:** Receive a grade

Transaktionen

Ausgangslage

Gegeben ist ein fertiges Programm, ein Mini-Webshop, der aus einem einfachen Webservice besteht. Leider wurde bei der Entwicklung komplett auf Nebenlaeufigkeiten vergessen, wodurch im Webshop verschiedene Fehler auftreten koennen. Deine Aufgabe ist es nun, diese Fehler zu beseitigen. Richte zunaechst eine leere Postgres-Datenbank ein und lege die entsprechenden Tabellen und Testdaten an, indem du die gegebene `webshop.sql` ausfuehrst:

```
postgres=# \c webshop
webshop=# \i webshop.sql
```

Danach richte in der `db.properties` die Zugangsdaten zum Webserver ein und starte diesen. Dafuer benoetigst du noch den JDBC-Treiber fuer Postgres [1] sowie die JSON-Java Library [2]. Du kannst diese mittels Gradle installieren oder die entsprechende JAR-Dateien selbst herunterladen und in dein Projekt einbinden. Alternativ lassen sich die Libraries mittels gradle ueber die folgenden Dependencies einrichten:

```
dependencies {
     implementation 'org.json:json:20171018'
     implementation 'org.postgresql:postgresql:42.2.8'
}
```

Funktionsweise des Webshops

Standardmaessig laeuft der Webshop auf Port 8000; falls dieser Port bei dir belegt ist, kannst du ihn mittels dem Property `Server.port` aendern. Du kannst das laufende Webservice dann entsprechend unter http://127.0.0.1:8000 aufrufen. Wie du siehst, existieren Methoden zum Anzeigen von Kunden, Bestellungen, und Artikeln, sowie eine Methode zum Anzeigen genereller Statistiken und zum Aufgeben von Bestellungen. Alle Methoden koennen im Browser per Adresszeile (d.h. per GET-Request) aufgerufen werden und liefern eine Antwort im JSON-Format.

Transaktionen in JDBC

Die Dokumentation zu Transaktionen in JDBC findest du unter https://docs.oracle.com/javase/tutorial/jdbc/basics/transactions.html. Eine neue Transaktion startest du (wenn `conn` deine Verbindung zur Datenbank ist), indem du mittels `conn.setAutoCommit(false)` Postgres anweist, bei einer neuen Query automatisch eine Transaktion zu starten. Mit `conn.commit()` laesst sich diese dann committen, waehrend sie sich mit `conn.rollback()` wieder abbrechen laesst. Das Isolation Level laesst sich *vor* der ersten Query mittels `conn.setTransactionIsolation(<isolation level>)` setzen.

Arbeisanweisungen

Folgende Fehlerfaelle wurden bei der Entwicklung des Webshops nicht berueecksichtigt:

1. **Generieren einer id fuer neue Bestellungen** Beim Anlegen einer Bestellungen wird zunaechst die hoechste vergebene Order-Id gesucht und dann eine neue Bestellung entsprechend erhoeht eingefuegt. Demonstriere, wieso das zu Problemen fuehren kann. Fasse danach diese beiden Operationen in eine Transaktion zusammen. Laesst sich das Problem durch Setzen eines Isolation Levels loesen? Beseitige das Problem nun endgueltig, indem du vor Auslesen der maximalen ID die Tabelle fuer alle anderen Teilnehmer lockst. Welche Art von Lock ist hier am sinnvollsten?
2. **Atomizitaet bei Bestellungen** Tritt waehrend dem Speichern der einzelnen Positionen einer Bestellung ein Fehler auf, so koennen unvollstaendige Bestellungen im System verbleiben oder auch Warenstaende faelschlich verringert werden, was sicher ungewuenscht ist. Demonstriere und dokumentiere dieses Verhalten. Fuehre diese Schritte daher nun in einer Transaktion durch, sodass sichergestellt ist, dass eine Bestellung ganz oder gar nicht erfasst wird. Nachdem das Anlegen in der Tabelle `orders` selbst nach Aufgabe 1 eine eigene Transaktion ist, wird dir im Fehlerfall eine "leere" Bestellung uebrigbleiben - dies stellt hier kein Problem dar. (*Zusatzfrage: Wie muesste man die Anwendung bzw. die Datenbank aendern, um auch das Anlegen leerer Bestellungen zu vermeiden ohne gleichzeitig die Performance stark zu beeintraechtigen?*)
3. **Gleichzeitiges Abschicken zweier Bestellungen** Werden zwei Bestellungen fuer den gleichen Artikel gleichzeitig abgeschickt, koennen mehr Artikel bestellt werden, als vorhanden (und der Lagerstand dabei ins Negative gehen). Demonstriere dieses Verhalten und verwende danach sinnvolle Locks auf Zeilenebene um solche Fehler zu Verhindern.
4. **Anzeige von Statistiken** Der Aufruf von http://127.0.0.1:8000/stats liefert Statistiken ueber Bestellungen nach Laendern aufgeschluesselt. Wird waehrend dem Erstellen der Statistik eine Bestellung abgeschickt, so kann diese Statistik inkonsistent werden (zB. in der Uebersicht weniger Bestellungen anfuehren, als spaeter in der Detailsansicht). Demonstriere dieses Verhalten stelle danach sicher, dass solche Phaenomene ausgeschlossen werden koennen. Um die Performance nicht zu beeintraechtigen soll deine Loesung aber keine Locks verwenden, sondern durch Setzen eines entsprechenden Isolation Levels realisiert werden.

Anmerkungen

- Parallele Requests auf das Webservice kannst du auf der Kommandozeile mit Tools wie `curl`, generell mit Anwendungen wie Postman, aber auch einfach mit deinem Webbrowser simulieren. Beachte bei letzterem, dass es noetig sein kannst, dass du zwei verschiedene Browser wir Firefox und Chrome verwendest, da zB Chrome gerne ansich parallele Requests zum gleichen Server trotzdem hintereinander ausfuehrt.
- Zum Demonstrieren von Fehlern, die durch Nebenlaeufigkeiten entstehen, gibt es die Methode `sleep(<seconds>)`, mit der du den aktuellen Thread fuer eine bestimmte Anzahl an Sekunden warten lassen kannst.
- Der Shop soll nicht als Vorlage fuer sauberers API-Design dienen -- insbesondere das Abschicken von Bestellungen. Der Fokus lag hier auf einfachem Code und einfacher Testbarkeit im Webbrowser. (*Zusatzfrage: Wie wuerdest du in einem Webservice einen API Endpoint fuer Bestellungen besser realisieren?*)
- Bei Problemen/Unklarheiten, frage bitte fruehzeitig nach. Die Arbeitsanweisungen erforden *keinen* grossen Programmieraufwand.

Abgabe

Abzugeben ist ein Arbeitsprotokoll im PDF-Format. Danach kannst du dich zu einem Abgabegespraech anmelden.

[1] https://jdbc.postgresql.org/download.html

[2] https://github.com/stleary/JSON-java

db.properties	13 September 2023, 8:34 AM
Server.java	13 September 2023, 8:34 AM
webshop.sql	13 September 2023, 8:34 AM

Add submission

Submission status

Submission status	No submissions have been made yet
Grading status	Graded
Last modified	-
Submission comments	▸ Comments (0)

Feedback

Grade	GK vollständig
Graded on	Wednesday, 22 May 2024, 1:38 PM
Graded by	UM Umar Muhammad-Adeel