Prof. Dr. V. Linnemann
Stefan Werner
Universität zu Lübeck
Institut für Informationssysteme

Lübeck, den 7. Januar 2013

Anfrageverarbeitung

Wintersemester 2012/2013

3. Block

Im 3. Block sollen Transaktionen realisiert werden. Obwohl es sich bei der vereinfachten Datenbankversion nur um ein Einbenutzersystem handelt, können doch beliebige Threads auf die Tabellen der Datenbank zugreifen und so zu unvorhersagbaren Operationsresultaten führen. Um das zu ändern und die $ACID$-Eigenschaften zu garantieren, sollen Datenbankoperationen nur noch über einen Scheduler abgewickelt werden, der Transaktionen unterstützt.

Aufgabe 1: Scheduler

Der Scheduler soll die folgende Schnittstelle implementieren (vgl. Abb. 1).

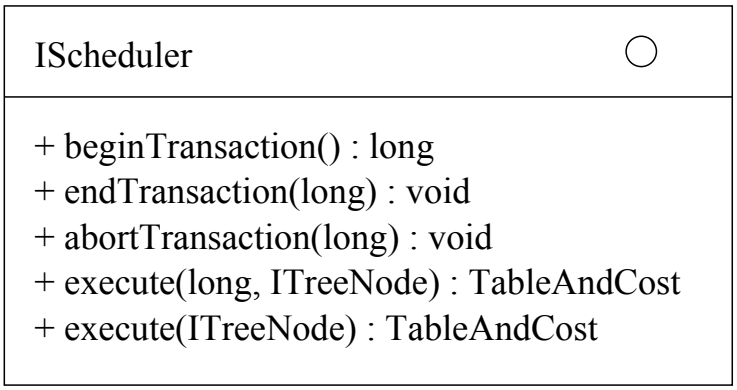

Abbildung 1: Die Scheduler-Schnittstelle

Transaktionen besitzen eine ID, in diesem Fall mittels $long$-Datentyp modelliert. Transaktionen müssen beim Scheduler initiiert bzw. angemeldet werden, können abgebrochen oder beendet werden. Transaktionen sollen die $ACID$-Eigenschaften garantieren.

Mittels der $execute$-Methode ohne Transaktions-ID kann ein einzelner Operationsplan ausgeführt werden. Intern behandelt der Scheduler den Operationsplan aber ebenfalls als Transaktion, die nur aus diesem einzelnen Operationsplan besteht. Der Aufruf

```
scheduler.execute(plan);
```

ist daher äquivalent zu

```
long id = scheduler.beginTransaction();
scheduler.execute(id,plan);
scheduler.endTransaction(id);
```

Ersterer reduziert lediglich den Transaktionsoverhead für den Benutzer. Die $execute$-Methode mit Transaktions-ID führt demnach einen Operationsplan innerhalb einer beliebig langen Transaktion aus. Der Scheduler muss als Singelton implementiert werden, damit der Datenbankzugriff ausschliesslich über eine Instanz abgewickelt wird. Transaktions-IDs müssen eindeutig über die Laufzeit des Schedulers vergeben werden.

Beginnen Sie mit der Implementierung der Klasse $Scheduler$.

Aufgabe 2: Transaktionen

Transaktionen sollen hier mittels privatem Arbeitsbereich implementiert werden. Jede Transaktion arbeitet auf privaten Kopien der von ihr benötigten Datenbanktabellen, beispielsweise kann der Scheduler die getrennten Arbeitsbereiche der Transaktionen mittels Java-Map verwalten. Mittels privatem Arbeitsbereich können alle Transaktionen vollständig parallel ablaufen. Falls ein Abbruch erfolgt, kann der Arbeitsbereich einfach gelöscht werden, die Datenbank ist davon nicht betroffen. Erst im Moment des Festschreibens einer Transaktion ($endTransaction$) muss mit wechselseitigem Ausschluss gearbeitet werden. Es muss geprüft werden, ob die ursprünglichen Tabellenkopien, auf denen die Transaktion ihre Arbeit begonnen hat, immer noch aktuell sind. Wenn das für alle betroffenen Tabellen zutrifft, kann die Transaktion und damit die Tabellen festgeschrieben werden. Wenn das nicht zutrifft, muss die Transaktion abgebrochen werden. Wichtig im Moment des Prüfens und Festschreibens ist der wechselseitige Ausschluss anderer Transaktionen, hier z.B. mittels geeignetem $synchronized$-Einsatz. Bleibt noch die Frage, wie sich die Aktualität der Tabellen bzw. deren Version überprüfen lässt. Zu diesem Zweck kann beispielsweise mit Zeitstempeln gearbeitet werden. Beim global erstmaligen Laden einer Tabelle aus dem Speicher in den Hauptspeicher durch den Scheduler kann dieser in eine Zeitstempel-Map den Tabellennamen und die aktuelle Systemzeit ablegen. Transaktionen haben nun einen Ausgangszeitstempel, der natürlich im Laufe der Zeit entsprechend aktualisiert werden muss.

Stellen Sie die Implementierung der Klasse $Scheduler$ fertig.

Aufgabe 3: Thread-Test

Nun soll der Scheduler inklusive seiner Transaktionsverwaltung getestet werden. Erweitern Sie den Scheduler so, dass er sinnvolle Ausgaben auf der Konsole erzeugt, die über die aktuell laufenden Transaktionsaktionen Auskunft geben. Implementieren Sie eine Klasse $DBClientThread$, die $Thread$ erweitert und als einziges Attribut eine Datei besitzen soll, in der nachher die zu einer Transaktion gehörenden Operationen stehen sollen. Die run-Methode führt diese Transaktion aus. Des weiteren soll eine Klasse $ThreadTest$ implementiert werden, die drei $DBClientThread$-Instanzen mit unterschiedlichen Transaktionsdateien $sql1.txt$, $sql2.txt$ und $sql3.txt$ instanziiert und ausführt. Spielen Sie die Ausführung ein paar Mal durch und Sie werden unterschiedliche Transaktionsabläufe erzielen.

Abgabetermin: Montag, den 4. Februar 2013