Praktikum 4 - Praktikumsbericht

Datenbanken 2 - Sommersemester 2019
Gruppe: Fr2y-6
750907 | Selim Sinan
752940 | Ruben van Laack

Durchführung / Umsetzung

Das Programm wurde um die Klasse "SimulationController" erweitert.
Um Spieler, Spiele und Antworten auf Fragen zu erzeugen wird ein Objekt dieser Klasse mit den entsprechenden Parametern erzeugt,
via "runSimulation()" wird dann die Erzeugung der Objekte und deren Persistierung gestartet.

In "runSimulation()" werden zunächst alle verfügbaren Kategorien und Fragen aus der Datenbank geladen.
Danach wird in einem try-catch-block eine neue Transaktion gestartet.
Nach dem öffnen der Transaktion werden in 3 ineinander verschachtelten For-Schleifen die Spieler, die Spiele jedes Spielers, sowie die Antworten zu Fragen für jedes Spiel erzeugt.
Diese Objekte werden nach ihrer Erzeugung direkt dem Persistenzkontext hinzugefügt.
Nach jedem 500sten Objekt im Persistenzkontext wird ein "entityManager.flush()" und ein "entityManager.clear()" durchgeführt.
Um Bach-inserts zu ermöglichen / zu beschleunigen mussten die Einstellungen der "Sequences" für die Generierung der Entity-Ids leicht verändert werden.
Es wird nun eine größe Anzahl an Ids von der Datenbank, bzw. von der "Sequence" geladen und für die Objekterstellung vorgehalten,
statt be jedem flush für jedes Objekt einzeln eine Id abzufragen.

Nach der Erzeugung aller Objekte werden diese via "transaction.commit()" in die Datenbank geschrieben.
Die "runSimulation()" wird durch die Methode "multithreadedSimulation(...)" in mehreren Threads, für je einen Teil der Spieler augerufen, um die Ausführung zu parallelisieren.

Die Parameter für den Aufruf von "multithreadedSimulation(...)" sind so gewählt,
das 4 Threads mit unabhängigen Datenbank-Sessions je 2.500 Spieler, mit je 100 Spielen (2 - 5 Kategorien; 5 - 10 Fragen pro Spiel) erzeugen.

Antworten zu den Fragen in der Praktikumsbeschreibung

- 'Wann öffnen und schließen Sie Transaktionen?'

> Wie oben beschrieben, öffnet jeder der 4 Threads eine Transaktion und schließt diese nach der Erzeugung all seiner 2500 Spieler.

- BatchWriting: Performance-Gewinn

> - Mit den Einstellungen für Batch Writing war zunächst kein Performace-Gewinn zu erkennen. Nachdem die Generierung der Id mit "allocationSize = 10000" ergänzt wurde funktionierte das Batch Writing richtig.
> - Mit Batch Writing läuft ein Commit mit allen 10.000 Spielern ca. 5 mal schneller.

> - Beim Batch Writing werden mehre Aktionen gesammelt und später in einer Anweisung gemeinsam an die Datenbank gesendet und ausgeführt.

- Wie lange dauert die Massendatengenerierung bei Ihrer Anwendung?

> Leider läuft die Datenbank bei mir auf dem Tablet in einer Ubuntu-VM mit nur einem CPU Kern.
> Daher lief es selbst mit den unten Aufgeführten Optimierungen ca. 7 - 10 Minuten.

- Wie haben Sie eine schnelle Erzeugung der Daten bewirkt?

> - Konfiguration des Batch Writing mit in der persistance.xml sowie im Code mittels regelmäßigem Flush.
> - Änderung des "Identifier Generators" in Player, Game & QuestionAsked um IDs zu cachen.
> - Ausführung eines entityManager.clear() nach dem Laden aller Kategorien um Speicher frei zu machen.
> - Ausführung eines entityManager.clear() nach jedem entityManager.flush().
> - Nur ein Commit am Ende der Erzeugung aller Objekte. Der Commit dauert relativ lange, auch wenn er sehr oft aufgerufen wird.
> - Listenrepräsentation der Relation in Game & Question zu Questionasked entfernt. (10 - 20sec pro QuestionAnswer Erstellung schneller)
> - Erstellung und Persistierung der Objekte in mehreren Threads.

- Wie benutzen Sie Transactions und warum?

> - Je eine Transaktion pro Thread, also eine für die gesamte Erzeugung aller Objekte in einer Session.
> - Da die Objekte sehr schnell erzeugt werden und der Commit einer Transaktion auch mit wenigen Objekten länger dauert.

- Wie verwenden Sie flush(), clear(), etc. und warum?

> - Je ein Flush und ein Clear für das Erstellen eines Batches und um Platz für das nächste Batch freizugeben.