2014

Replikation

[UNTERTITEL DES DOKUMENTS]

VIKTOR KISS & NANAK TATTYREK

TGM | 5AHITS

Inhalt

Aufgabenstellung

Eine Handelsgesellschaft, die mehrere Filialen hat, betreibt eine Online-Plattform für den Verkauf der Produkte. Der Webshop wird mit Hilfe einer Datenbank betrieben. Bei dem Verkauf der Produkte werden Rechnungen in Form von PDF-Dokumenten erzeugt.

Die Daten (Datenbank, Rechnungen) sollen stets auf die Filialrechner repliziert werden, damit die Sachbearbeiter vor Ort diese einsehen und bearbeiten können.

Entwickle ein vereinfachtes Datenbankmodell für den Webshop
Wähle ein Konsistenzprotokoll Deiner Wahl (siehe Theorie bzw. Tanenbaum)
Implementiere einen Replikationsmanager in Java (JDBC, Sockets, o.ä. ...) für Datenbank und Rechnungen
alle Transaktionen im Zuge der Replikation sollen protokolliert werden (zum Beispiel mit Log4J)
Beispiel fuer Log-Eintrag:
Replikation Rechnungen München -> Berlin OKAY
Replikation DB München -> Berlin FEHLGESCHLAGEN

Problemstellungen:

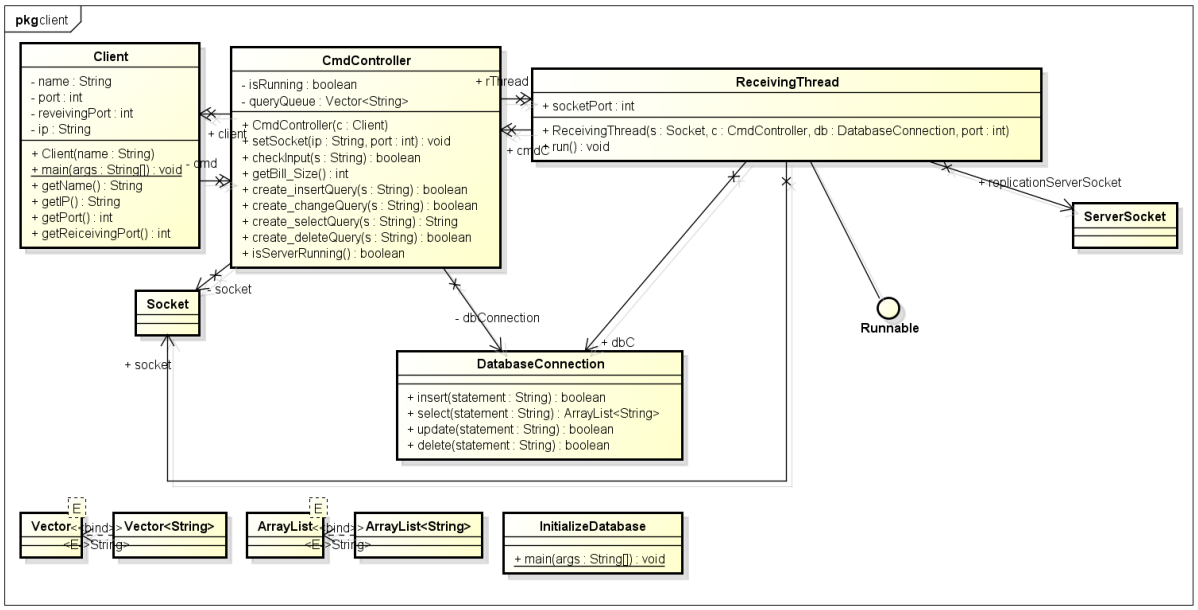
Wie oft wird repliziert?
Wie erfolgt der Aufruf des Replikationsmanager bzw. läuft der Replikationsmanager stets im HIntergrund?
Was passiert im Fehlerfall?
Welche Probleme koennen auftreten?
- Dateien mit gleichen Namen
- Dateien mit gleichen Namen und unterschiedlicher Größe
- Datensatz mit gleichem Schlüssel

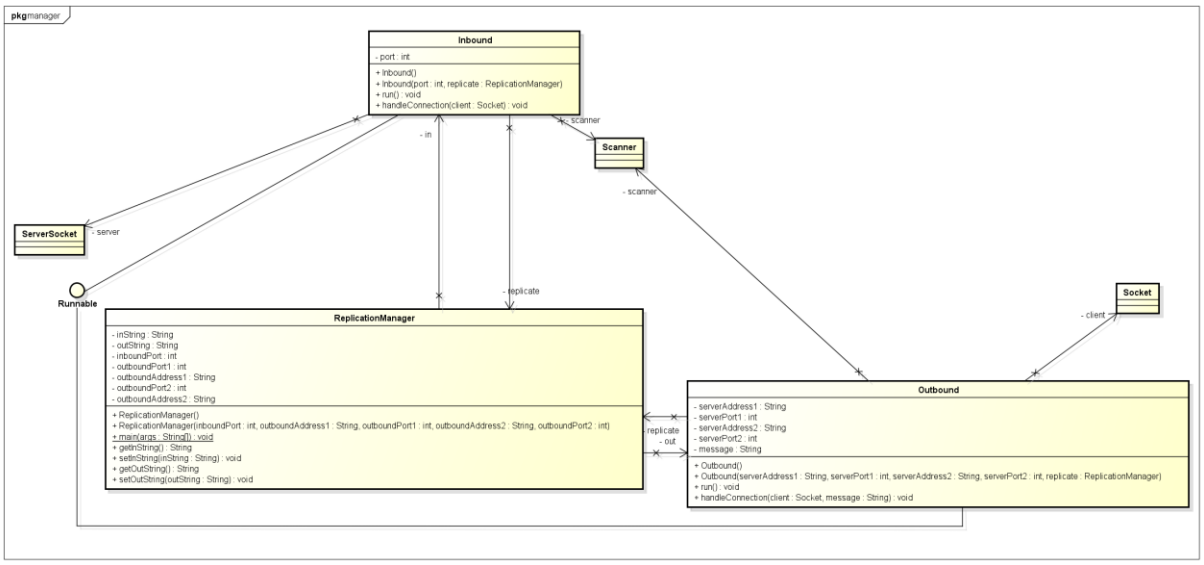
Meilensteine (16Pkt): Gruppengröße 2 Personen

Erstelle ein Replikationskonzept für diese Handelsgesellschaft (4 Punkte)
Implementiere dieses Konzept für zwei Rechner (6 Punkte)
mind. 10 Datensätze pro Tabelle, mind. 10 Rechnungen
Implementierung Logging (2 Punkte)
Dokumentiere drei Fehler-/Problemfälle und entsprechende Lösungvorschläge (4 Punkte)

Designüberlegung / Konzept

Das Konzept für die Replikation war, dass die Clients jeweils eine SQLite Datenbank lokal liegen haben und das die Tabellen jeweils lokal bearbeitet werden. Wenn der Client einen Command eingetippt hat, wird dieser an einen Command-Mapper weiter geschickt, welcher daraus eine Query „bastelt“ und mit Hilfe der „DatabaseConnection“ Klasse das Query ausführt.
Wenn dies erfolgt ist, wird der „Command“ an den Replikationsserver weiter geschickt. Dieser speichert die ankommenden Commands in einem „String“-Vector. Zusätzlich speichert er alle Clients in einer Liste.
Hierfür hat der Replikationsmanager 2 Klassen (Inbound & Outbund). Diese implementieren beide das Interface „Runnable“ und rennen im ReplikationsManager als Thread. Die „Outbound“ Klasse geht die Liste der angemeldeten Clients durch und versendet an diese das Command, welches er vorher von einem Client bekommen hat. Für das erhalten von den Commands wurde die Klasse „Inbound“ implementiert.
Die Clients melden erhält den Command und behandelt diesen wie einen Input vom User, führt ihn aus und schickt eine Message an den ReplikationsManager, ob die Transaktion erfolgreich war oder nicht.
Um die Transaktionen auseinander halten zu können, steht in der Message der Name des Clients, welcher beim start des Clients vom User festgelegt wird, und ein TimeStamp, welcher an die Message angehängt wird.





Erfolge / Misserfolge

Positive Aspekte waren, dass die Verbindung zur Datenbank und der Command-Mapper aufgrund von einer guten SQLite Dokumentation schnell & funktionierend war.
Negativ ist erstens, dass mit diesem Konzept, dennoch das Lost-Update Problem nicht beseitigt ist. Weiters wurde das alte Konzept verworfen und ein komplett neues erstellt, wobei dies aufgrund von schlechtem Zeit-Management erst relativ spät geschah und somit die gesamte Arbeit zu sehr in Verzug kam.
Ein sehr Zeitraubendes Problem, war ebenfalls die Kommunikation über Sockets zu implementieren, da unsere Kommunikation String basierend ist, und sich diesbezüglich selbst auf der offiziellen Oracle Website nur relativ umständliche und aufwendige Erklärungen finden.

Dies in Kombination mit miesem Zeitmanagement, führte schlussendlich zu einem nicht funktionierenden Programm.

Technologien

Wie aus den vorherigen Kapiteln ersichtlich ist, haben wir für die Kommunikation zwischen Client und ReplikationsManager Threads in Kombination mit Sockets verwendet.

Als Datenbank bzw. Datenbanksprache haben wir uns für SQLite entschieden. Bei SQLite handelt es sich um eine kleine Programm-Bibliothek die ein relationales Datenbanksystem darstellt.
Im Gegensatz zu anderen DBMS wie MySQL und co. bestehen SQLite-Datenbanken aus einer einzigen Datei und Zugriffe auf die Datenbank können nicht parallel sondern nur seriell durchgeführt werden.