TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Dr. Thomas Neumann

Übung zur Vorlesung
***Einsatz und Realisierung von Datenbanksystemen* im SoSe15**
Moritz Kaufmann (moritz.kaufmann@tum.de)
http://db.in.tum.de/teaching/ss15/impldb/

Blatt Nr. 3

Hausaufgabe 1 Beim strikten 2PL commit werden die folgende Schritte in angegebener Reihenfolge ausgeführt:

1. Eintragen des commit in den Logbuffer
2. Persistieren des Logbuffer
3. Freigabe aller Sperren
4. Rückmeldung an den Benutzer

Manche Datenbanksysteme führen Schritt 3 als ersten Schritt aus. Was kann dies für Vorteile bringen? Sind trotzdem noch alle Garantien vom strikten 2PL gewährleistet, wenn ja unter welchen Bedingungen?

In der ursprünglichen Abfolge muss gewartet werden, bis das Log auf die Festplatte geschrieben ist. Dies kann mehrere Millisekunden dauern. Gerade für sehr schnelle Transaktionen kann dies ein vielfaches der eigentlich Verarbeitungsdauer sein. In der vorgeschlagenen Abfolge entfällt das Warten auf die Festplatte und die Sperren können viel früher freigegeben werden. Dadurch wird der Durchsatz des Systems erhöht.
Um die Eigenschaften von strikten 2PL zu erhalten, dürfen nur *serialisierbare* und *strike* Historien möglich sein. Da für Serialisierbarkeit von Transaktionen nur die Schreib- und Leseoperationen betrachtet werden und die Änderung erst im *commit*-Schritt ist, ändert sich in dieser Hinsicht nichts.
Für Striktheit ist es wichtig, dass keine andere Transaktion die veränderten Daten der aktuellen Transaktion an den Benutzer zurückliefert, solange nicht garantiert ist, dass diese nicht mehr zurückgesetzt wird. Zum Zeitpunkt des *commit* ist die einzig mögliche Ursache für ein Abbruch, dass das Log nicht persistiert werden kann.

Zur Analyse eine repräsentative Abfolge von Transaktionen:

Schritt	T_1	T_2
1.	w(A, 4)	
2_0.	Start commit	
2_1.	Freigabe der Sperren	
3.		o(A,a_1)
4_0.		commit start
		...

Falls mehrere Transaktionen gleichzeitig committen dürfen, kann ein Fehlerzustand gezeigt werden. Dazu T_2 schließt alle Schritte des commit ab und das System crasht, bevor T_1 das commit in den Logbuffer einträgt. Somit würde beim Wiederanlauf T_1 zur Losertransaktion und ihre Änderungen zurückgenommen. Damit hätte T_2 Daten gelesen, die nicht mehr in der Datenbank stehen.
Um dieses Problem zu umgehen gibt es zwei Möglichkeiten.

- Immer nur eine Transaktion darf gleichzeitig im commit sein. Somit müsste im Beispiel T_2 warten, bis T_1 das commit abgeschlossen hat. Falls das Log nicht geschrieben werden kann, werden alle Transkationen abgebrochen.
- Die Freigabe der Sperren wird erst nach Eintragen des commit in den Logbuffer durchgeführt. Im commit von T_2 müsste alle vorhergehenden Logeinträge (somit auch das commit von T_1) persistiert werden und damit wäre T_1 nach Wiederanlauf eine Winnertransaktion.

Hausaufgabe 2 Eine statistische Datenbank ist eine Datenbank, die sensitive Einträge enthält, die aber nicht einzeln betrachtet werden dürfen, sondern nur über statistische Operationen. Legale Operationen sind beispielsweise Summe, Durchschnitt von Spalten und Anzahl der Tupel in einem Ergebnis (**count**, **sum**, **avg**, ...).

Nehmen wir an, Sie haben die Erlaubnis, im **select**-Teil einer Anfrage ausschließlich die Operationen **sum** und **count** zu verwenden. Weiterhin werden alle Anfragen, die nur ein Tupel oder alle Tupel einer Relation betreffen, abgewiesen. Sie möchten nun das Gehalt eines bestimmten Professors herausfinden, von dem Sie wissen, dass sein Rang „C4“ ist und er den höchsten Verdienst aller C4-Professoren hat. Beschreiben Sie Ihre Vorgehensweise.

Siehe Übungsbuch.

Hausaufgabe 3 Skizzieren Sie die Funktionsweise von SSL. Erläutern Sie hierzu, wie der einfache TLS Handshake funktioniert. Eine Lösungsmöglichkeit wäre das Zeichnen eines passenden Message Sequence Charts.

Alles ohne Resumed Handshake:

- **Client:** ClientHello, höchste Version, Zufallszahl1, Ciphers

  D.h. der Client übermittelt seine Parameter für die Verbindung, beispielsweise, was er für Techniken unterstützt etc.

- **Server:** ServerHello, genutzte Version, Zufallszahl2, genutztes Cipher

  Hierbei sollte gelten: Cipher$\in$Ciphers, genutzte Version=$max(VersionenClient, VersionenServer)$, genutztes Cipher=$max(CipherClient, CipherServer)$

  Der Server wählt hier also bereits die geltenden Parameter für die Verbindung aus. Ein gängiges Problem ist, dass der Server sich aufgrund der vom Client übermit-

telten Informationen nicht auf zu unsichere Protokolle festlegen darf und potentiell Verbindungen abweisen muss, um effektiven Schutz vor Angriffen zu bieten.

- **Server:** Certificate

  Im Allgemeinen übermittelt der Server ein Zertifikat an den Client, welches es dem Client erlaubt, die Identität des Servers zu verifizieren. Im Internet, d.h. bei https Verbindungen, dienen hierzu typischerweise Zertifikate, die von einer Autorität signiert wurden, der der Client vertraut. Diese sog. root Zertifikate werden vom Browser Hersteller mit dem Browser an den Client ausgeliefert und typischerweise nicht durch den Anwender geprüft.

- **Server:** ServerHelloDone

  Handshake beendet, d.h. alle Informationen für die Einleitung der Verschlüsselung sind ausgetauscht.

- **Client:** ClientKeyExchange, PreMasterSecret, evtl. Public Key

  PreMastersecret wird nun i.A. mit dem Public Key des Servers verschlüsselt. Hierdurch ist diese dem Client bekannt, kann vom Server entschlüsselt werden, jedoch niemand, der die Verbindung abhört kann es einfach rekonstruieren.

- **Client:** ChangeCipherSpec: Ab hier verschlüsselt, Finished: Verschlüsselte Nachricht mit Hashes über alles vorherige.

- **Server** ChangeCipherSpec: Ab hier verschlüsselt, Finished siehe oben.

- Ab jetzt wird die Verbindung von der Applikation verwendet.

Alles streng nach `http://en.wikipedia.org/wiki/Transport_Layer_Security`

Hausaufgabe 4 Sie haben die Users-Tabelle eines Pizzalieferanten ausgelesen, jedoch scheint sein Passwort uncharakteristisch kompliziert zu sein. Das von Ihnen erhaltene Resultat ist das Folgende:

```
id          name        password
----------  ----------  --------------------------------
1           wolfgang    4d75e8db6a4b6205d0a95854d634c27a
```

- Was könnte der Grund für dieses hexadezimale, 32 Stellen lange Passwort sein?
- Können Sie trotzdem den Klartext finden?
- Wie können Sie das Passwort sicherer Speichern?
- Wie können Sie für diese Art von Passwortspeicherung Bruteforce-Attacken erschweren?
- Das Passwort wurde MD5 gehasht.
- Google nach dem Hash, es gibt sog. Rainbow-Tables in denen zahlreiche MD5 Hashes vorberechnet sind.
- MD5 mehrfach anwenden, besser: Einen Salt verwenden, beispielsweise das Passwort zusammen mit dem Erstellungsdatum des Accounts oder dem Accountnamen hashen.
- Eine bessere Hashfunktion verwenden (MD5 und SHA1 werden nicht mehr empfohlen) die mehr Rechenzeit benötigt. Genauso wichtig, den User zwingen komplexere Passwörter zu benutzen damit Wörterbuchangriffe ineffizient werden.

Hausaufgabe 5 Implementieren Sie den RSA - beispielsweise in Python. Verdeutlichen Sie sich die Funktionsweise des RSA an einem geeigneten Beispiel, d.h. verschlüsseln Sie eine kleine Nachricht mit einem (nicht zu großen) Schlüssel und bringen Sie das Beispiel inklusive Verschlüsselung und Entschlüsselung mit in die Übung.

Würde bei RSA eine abgefangene verschlüsselte Nachricht mit bekanntem Inhalt die Suche nach dem Private Key erheblich vereinfachen? Begründen Sie kurz.

Siehe Übungsbuch oder (sehr gut gemacht!) aus Wikipedia unter `http://de.wikipedia.org/wiki/RSA-Kryptosystem`.
Eine einfach Python Implementierung finden Sie unter: `http://code.activestate.com/recipes/578838-rsa-a-simple-and-easy-to-read-implementation/` Nein, ein Angreifer kann über den Public Key immer beliebige Texte verschlüsseln. Würden known Plaintext Attacken das Knacken wesentlich erleichtern, wäre das Verfahren komplett unbrauchbar.

Hausaufgabe 6 Bob hat ein Vorlesungsverzeichnis für die Universität programmiert und unter `http://db.in.tum.de/~kaufmann/sql_verzeichnis.html` online gestellt.

Um die Suche zu erleichtern, kann die Anzahl der SWS durch ein Parameter eingeschränkt werden. Finden sie eines speziell präparierte Parameterm, bei dessen Eingabe statt der Vorlesungen die Liste der Studenten ausgegeben wird. Die Datenbank folgt dem bekannten Universitätsschema.

Bob erfährt von der Sicherheitslücke und schlägt vor die bekannten Tabellen einmalig mit zufälligen Namen umzubenennen, so seien sie nicht zu finden. Würde diese *Sicherheitsmaßname* helfen?

- Injection: `0 union all select name, matrnr, semester from studenten`
- Nein, da z.B. mit `select * from pg_tables` eine Liste der Datenbanken ausgegeben werden kann.