Transaktionen in Praxis

Dr. Karsten Tolle
Vorl. 13.06.2017

Probleme bei Transaktionen

- „Lost Update“ und „Inconsistent Retrieval”
  … Sichtweise vom Benutzer

Auszug aus SQL 92

1) P1 ("Dirty read"): SQL-transaction T1 modifies a row. SQL- transaction T2 then reads that row before T1 performs a COMMIT. If T1 then performs a ROLLBACK, T2 will have read a row that was never committed and that may thus be considered to have never existed.

2) P2 ("Non-repeatable read"): SQL-transaction T1 reads a row. SQL-transaction T2 then modifies or deletes that row and performs a COMMIT. If T1 then attempts to reread the row, it may receive the modified value or discover that the row has been deleted.

3) P3 ("Phantom"): SQL-transaction T1 reads the set of rows N that satisfy some <search condition>. SQL-transaction T2 then executes SQL-statements that generate one or more rows that satisfy the <search condition> used by SQL-transaction T1. If SQL-transaction T1 then repeats the initial read with the same <search condition>, it obtains a different collection of rows.

http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt

Transaktions-Isolationsstufen

- **read uncommitted** – schwächste Stufe, uncommittete Änderungen werden auch gelesen
- **read committed** – nur committete Änderungen werden gelesen (keine *dirty reads*)
- **repeatable read** – erneutes Lesen enthält mindestens die vorher gelesenen; es können aber neue hinzugekommen sein (keine *dirty reads* und *non-repeatable reads*)
- **serializable** – garantiert eine serielle Ausführung

Übersicht Isolationsstufen

Isolationsebene	Dirty Read	Non-Repeatable Read	Phantom Read
Read Uncommitted	möglich	möglich	möglich
Read Committed	nicht möglich	möglich	möglich
Repeatable Read	nicht möglich	nicht möglich	möglich
Serializable	nicht möglich	nicht möglich	nicht möglich

MySQL

SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL {READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE}

Praxisbeispiel in Java

```
Connection con = null;
try {
con = DriverManager.getConnection("jdbc:db2:sample");
} catch (Exception e) { e.printStackTrace(); }
con.setAutoCommit(false);
con.setTransactionIsolation(
    Connection.TRANSACTION_READ_UNCOMMITTED);
…
con.commit();
con.close();
```

Transaktions-Isolationsstufen

- Besonderheiten des DBMS beachten!!! … es wird nicht immer alles unterstützt!

- … mit Vorsicht zu nutzen!

Transaktions-Isolationsstufen

- Man beachte in Java:
  Wird eine Isolationsstufe nicht unterstützt, kann diese durch eine höhere Stufe substituiert werden. Eine *SQLException* wird nur dann geworfen, wenn dies nicht möglich ist. Die möglichen Isolationsstufen können über *DatabaseMetaData.supportsTransactionIsolationLevel* erfragt werden.

MySQL 5.1 Referenzhandbuch

Besonderheiten des DBMS beachten!!!

13.4.2. Statements können nicht zurückgerollt werden

Es gibt Anweisungen, für die kein Rollback möglich ist. Hierzu gehören DDL-Anweisungen (Data Definition Language), z. B. solche, mit denen Datenbanken erstellt oder gelöscht oder Tabellen oder gespeicherte Routinen erstellt, gelöscht oder geändert werden.

…

http://dev.mysql.com/doc/refman/5.1/de/cannot-roll-back.html

13.4.3. Anweisungen, die implizite Commits verursachen

Alle folgenden Anweisungen (und ggf. auch ihre Synonyme) beenden eine Transaktion implizit – so als ob Sie vor Ausführung der Anweisung eine COMMIT-Anweisung abgesetzt hätten:

ALTER FUNCTION, ALTER PROCEDURE, ALTER TABLE, BEGIN, CREATE DATABASE, CREATE FUNCTION, CREATE INDEX, CREATE PROCEDURE, CREATE TABLE, DROP DATABASE, DROP FUNCTION, DROP INDEX, DROP PROCEDURE, DROP TABLE, LOAD MASTER DATA, LOCK TABLES, RENAME TABLE, SET AUTOCOMMIT=1, START TRANSACTION, TRUNCATE TABLE, UNLOCK TABLES.

http://dev.mysql.com/doc/refman/5.1/de/implicit-commit.html

Lock Escalation

- Hierunter versteht man den Wechsel auf eine höhere Sperrgranularität, z.B. von Attribut → Tupel → Tabelle
- Dieser Wechsel ist nicht unproblematisch. Es kann zu Verklemmungen (Deadlocks) und Performanceeinbrüchen kommen.

Unterstützung des DBMS

- DBMS Parameter beachten, z.B. wie viel Platz ist zum Halten der Sperren vorgesehen?

  Beispiel IBM DB2:
  - **LOCKLIST** – die Speichergröße für Sperren
  - **MAXLOCKS** – gibt an wie viel Prozent eine Anwendung die Sperrliste belegen muss, damit eine Lock Escalation durchgeführt wird.

Unterstützung des DBMS

- <transaction access mode> ::= READ ONLY | READ WRITE

```
SET TRANSACTION READ ONLY;
```

- `SELECT … [ FOR UPDATE | LOCK IN SHARE MODE ];`

- (DB2): Sperrgranularität kann auch durch den Anwender definiert werden, z.B. mit:

```
ALTER TABLE <table_name>
      LOCKSIZE { ROW | TABLE }
```