Transaktionen in Praxis

Dr. Karsten Tolle
Vorl. 12.12.2018

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

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 / MariaDB

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

Beispiel in Workbench …

```
start transaction;
use airport;
select * from plane;insert into plane values (17, 19, 'a212', 'Frankfurt');
## ALTER TABLE customer ADD COLUMN MilesAndMore INT;
insert into plane values (19, 29, 'a212', 'Frankfurt am Main');
select * from plane;
ROLLBACK; # oder commit;
select * from plane;
```

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.

…

MySQL v.8.0 Referenzhandbuch

13.3.2 Statements That Cannot Be Rolled Back

Some statements cannot be rolled back. In general, these include data definition language (DDL) statements, such as those that create or drop databases, those that create, drop, or alter tables or stored routines.

You should design your transactions not to include such statements. If you issue a statement early in a transaction that cannot be rolled back, and then another statement later fails, the full effect of the transaction cannot be rolled back in such cases by issuing a ROLLBACK statement.

https://dev.mysql.com/doc/refman/8.0/en/cannot-roll-back.html

13.3.3 Statements That Cause an Implicit Commit

The statements listed in this section (and any synonyms for them) implicitly end any transaction active in the current session, as if you had done a COMMIT before executing the statement.

Most of these statements also cause an implicit commit after executing. The intent is to handle each such statement in its own special transaction. Transaction-control and locking statements are exceptions: If an implicit commit occurs before execution, another does not occur after.

Data definition language (DDL) statements that define or modify database objects. ALTER EVENT, ALTER FUNCTION, ALTER PROCEDURE, ALTER SERVER, ALTER TABLE, ALTER VIEW, CREATE DATABASE, CREATE EVENT, CREATE FUNCTION, CREATE INDEX, CREATE PROCEDURE, CREATE ROLE, CREATE SERVER, CREATE SPATIAL REFERENCE SYSTEM, CREATE TABLE, CREATE TRIGGER, CREATE VIEW, DROP DATABASE, DROP EVENT, DROP FUNCTION, DROP INDEX, DROP PROCEDURE, DROP ROLE, DROP SERVER, DROP SPATIAL REFERENCE SYSTEM, DROP TABLE, DROP TRIGGER, DROP VIEW, INSTALL PLUGIN, RENAME TABLE, TRUNCATE TABLE, UNINSTALL PLUGIN.

…

https://dev.mysql.com/doc/refman/8.0/en/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 }
```