Datenbank-Programmierung

Anhang C: Sperren in DB2

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Sommersemester 2018/19

http://www.informatik.uni-halle.de/~brass/dbp19/

Inhalt

Isolationsstufen (1)

- Bei IBM DB2 (Ver. 8) gibt es vier Isolationsstufen:
  - Repeatable Read (`RR`)

    Höchste Isolationsstufe, vermeidet auch das Phantom Problem, wenig Parallelität.
  - Read Stability (`RS`)

    Vermeidet das Nonrepeatable Read Problem, alle Lost Updates.
  - Cursor Stability (`CS`): Default-Wert

    Vermeidet das Dirty Read Problem sowie Lost Updates bei Updates aus einem Befehl bzw. für das aktuelle Tupel eines Cursors.
  - Uncommitted Read (`UR`)

    Erlaubt das Lesen noch nicht mit `COMMIT` bestätigter Daten. Bei `UPDATE`-Befehlen verhält es sich aber wie `CS`.

Isolationsstufen (2)

Isolationsstufe	Dirty Read	Nonrepeat-able Read	Phantom Problem
RR	—	—	—
RS	—	—	möglich
CS	—	möglich	möglich
UR	möglich	möglich	möglich

Nach der IBM-Dokumentation sind Lost Updates bei jeder Isolationsstufe ausgeschlossen. Das Beispiel eines Lost Updates bei Verwendung eines vorher mit `SELECT` gelesenen Wertes läuft auch bei DB2 mit Isolationsstufe CS durch. Wie oben erläutert, ist dies aber eher ein Fall von Nonrepeatable Read, der dann zu einem Lost Update führt.

Isolationsstufen (3)

- Transaktionen heißen bei IBM auch “Unit of Work” (UOW).
- Wenn man parallele Transaktionen ausprobieren will (z.B. mit zwei “Command Editor” Fenstern), muss man zuerst den Autocommit Modus ausschalten.

  Das geht unter “Tools → Tools Settings → Command Editor”. Danach muss man mit “Tool Settings → Exit” die neuen Setzungen abspeichern. Wenn man mit dem klassischen “Command Window” (CLP) arbeitet, muss man die Option “+c” bei jedem Kommando verwenden, z.B. “`db2 +c insert into ...`”.

Isolationsstufen (4)

- Im "Command Editor" und "Command Window" kann man eine Isolationsstufe mit

  `change isolation level to` ⟨Isolationsstufe⟩

  wählen, allerdings nur vor dem "`connect`".

  Man kann eine Sitzung mit "`terminate`" beenden und dann eine neue Sitzung z.B. mit "`connect to sample`" beginnen.

- Neuerdings kann man Isolationsstufen auch in jeder Anweisung festlegen:

```
SELECT * FROM KONTO WITH RR
```

  Die Isolationsstufe `UR` (Uncommitted Read) kann man nur für Leseanweisungen wählen. Andernfalls ersetzt sie das DBMS durch `CS`.

Inhalt

Sperren (1)

- Sperren haben drei wichtige Attribute:
  - Modus: Legt die erlaubten Zugriffe des Besitzers der Sperre und der anderen Benutzer fest.
  - Objekt: Eine Tabelle, eine Zeile, etc.
  - Dauer: Sperren werden spätestens beim Commit bzw. Rollback freigegeben, manche früher.
- Man kann nur Tabellen explizit sperren:

```
LOCK TABLE ⟨Tabelle⟩ IN EXCLUSIVE MODE
```

Man kann auch "`IN SHARE MODE`" sperren.

Sperren (2)

Modus	Table	Row	Read	Write
IN Intent None	X			
IS Intent Share	X			
IX Intent Exclusive	X			
SIX Share with Int. Excl.	X		X	
S Share	X	X	X	
U Update	X	X	X	
X Exclusive	X	X	X	X
W Weak Exclusive		X	X	X
Z Super Exclusive	X		X	X
NS Next Key Share		X	X	
NW Next Key Weak Excl.		X	X	

Sperren (3)

- Die Intent-Sperren (für Absichtserklärungen) sind nötig, weil es Sperren auf unterschiedlichen Ebenen (Tabellen, Zeilen) gibt.
- Man muss z.B. vermeiden, dass Nutzer A für einige Zeilen der Tabelle `X`-Sperren bekommt, und dann Nutzer B für die ganze Tabelle eine `S`-Sperre.
- Daher sind Sperren auf Zeilen immer mit Intent-Sperren auf der Tabelle gekoppelt: Nutzer A muss zuerst eine `IX`-Sperre auf der Tabelle haben, bevor er `X`-Sperren auf Tupeln bekommen kann.

Sperren (4)

- Wenn der Benutzer B eine `S`-Sperre für die Tabelle anfordert, nachdem A schon eine `IX`-Sperre hat, muss B warten.
- Intent-Sperren sind untereinander kompatibel,
  Nutzer C könnte auch eine `IX` oder `IS` Sperre bekommen, obwohl A schon eine `IX`-Sperre hat.

  Die Intent-Sperren sind ja nur ein Hinweis darauf, dass echte Sperren auf Tupel-Ebene existieren können.
- Intent-Sperren alleine geben noch keine Lese- oder Schreibrechte. Sie werden im Zusammenhang mit richtigen Sperren auf Tupelebene gebraucht.

Sperren (5)

- "Intent None" (`IN`) wird bei der Isolationsstufe "Uncommitted Read" benutzt, um anzuzeigen, dass ein Prozess die Tabelle ohne Sperren liest.
- Man muss für diese Zeit aber z.B. ein "`ALTER TABLE`" oder "`DROP TABLE`" ausschließen.
- Diese Kommandos würden eine `Z`-Sperre ("Super Exclusive") anfordern, die mit `IN` ("Intent None") nicht kompatibel ist: Sie müssten also warten.

Sperren (6)

- Jeder Prozess kann auf ein Objekt gleichzeitig nur eine Sperre haben.
- Fordert er eine zweite Sperre auf ein Objekt an, auf dem er schon eine Sperre hat, wird der Sperrmodus auf den stärkeren von beiden gesetzt.

  Diesen Vorgang der Änderung eines Sperrmodus nennt man “Lock Conversion”.
- Der einzige Fall, bei dem nicht einer von beiden Sperrmodi stärker als der andere ist, sind `S` und `IX`.
- Daher wurde der Modus `SIX` ($=$ `S` $+$ `IX`) eingeführt.

Sperren (7)

- Wenn ein Update ausgeführt werden soll, werden zunächst einige Zeilen gelesen, und dann eine Teilmenge davon geändert.
- Würde man beim Lesen eine S-Sperre verwenden, und dann eine "Lock Conversion" auf eine X-Sperre machen, kann das leicht zu Deadlocks führen.
- Daher werden in der Lese-Phase U-Sperren verwendet: Diese sind kompatibel zu S-Sperren (zu diesem Zeitpunkt ist ja noch nichts verändert), aber nicht zu anderen U-Sperren (auch nicht zu X etc.).

Sperren (8)

- Zur Vermeidung des Phantom-Problems wird beim Einfügen eines Tupels in einen Index das nächste Tupel im Modus `NW` ("Next Key Weak Exclusive") gesperrt.

  Tatsächlich ist es noch etwas komplizierter. Man muss hier auch zwischen Typ-1 und Typ-2 Indexen unterscheiden. Es wird versucht, diese Sperren einzusparen.

- Ein Index Scan im Modus `RR` würde alle gelesenen Tupel, inklusive des ersten Tupels, das die Bedingung nicht mehr erfüllt, im Modus `S` sperren.

- Die Sperrmodi `NW` und `S` sind nicht kompatibel.

Sperren (9)

- Die Einfügung müsste also warten, wenn sie einen Bereich betrifft, für den "Repeatable Read" garantiert werden soll.

  Wie oben schon erläutert, ist das Problem, dass ein nicht existierendes Tupel nicht gesperrt werden kann, daher wäre normalerweise eine Einfügung immer möglich. Der Trick ist nun, das nächste Tupel zu sperren (bzw. das Ende des Indexes, falls es das letzte Tupel war).

- Das eingefügte Tupel selbst wird mit einer `W`-Sperre versehen ("Weak Exclusive").

  Der einzige Unterschied zu einer `X`-Sperre ist, dass `W` kompatibel mit `NW` ist (jemand anders könnte also davor noch ein Tupel einfügen).

Sperren (10)

- Nur RR-Transaktionen müssen die Einfügung von Tupeln in dem von ihnen gelesenen Bereich verhindern.
- Daher wird in niedrigeren Isolationsstufen (CS, RS) die Sperre NS ("Next Key Share") anstelle von S verwendet: Sie ist kompatibel mit NW, ansonsten verhält sie sich wie S.

  Der Name ist etwas unglücklich: Es wird garnicht auf das nächste Tupel gesetzt, sondern es wurde eingeführt, um den Effekt der NW-Sperre abzumildern, die auf das nächste Tupel nach einem eingefügten Tupel gesetzt wird.

Sperren (11)

Angef. Sperre	Existierende Sperre											
	—	IN	IS	IX	SIX	S	U	X	W	Z	NS	NW
IN	+	+	+	+	+	+	+	+	+	−	+	+
IS	+	+	+	+	+	+	+	−	−	−	+	−
IX	+	+	+	+	−	−	−	−	−	−	−	−
SIX	+	+	+	−	−	−	−	−	−	−	−	−
S	+	+	+	−	−	−	+	−	−	−	+	−
U	+	+	+	−	−	−	−	−	−	−	+	−
X	+	+	−	−	−	−	−	−	−	−	−	−
W	+	+	−	−	−	−	−	−	−	−	−	+
Z	+	−	−	−	−	−	−	−	−	−	−	−
NS	+	+	+	−	−	+	+	−	−	−	+	+
NW	+	−	−	−	−	−	−	−	+	−	+	−

Sperren (12)

- Obige Tabelle (die sich so ähnlich in der offiziellen IBM-Dokumentation findet) zeigt Kombinationen, die nicht vorkommen können.
- Man müsste eigentlich eine Kompatibilitätsmatrix für Sperren auf Tabellenebene und eine für Sperren auf Tupelebene aufstellen (Übungsaufgabe).

SQL-Befehle und Sperren (1)

SELECT (Read-Only):

- `RR` (Repeatable Read, höchste Isolationsstufe):
  - Falls ein Table Scan zur Auswertung benutzt wird: `S`-Sperre auf der Tabelle.

    Dies gilt auch, wenn ein vollständiger Index-Scan genutzt wird.
  - Bei Zugriff über einen Index: `IS`-Sperre auf der Tabelle und `S`-Sperren auf den über den Index gelesenen Tupeln (plus das nächste Tupel).

    Es werden dabei alle über den Index zugegriffenen Tupel gesperrt, auch solche, die eventuelle weitere Bedingungen in der `WHERE`-Klausel nicht erfüllen. Die Sperren werden bis zum Transaktionsende gehalten.

SQL-Befehle und Sperren (2)

- `RS` benutzt eine `IS`-Sperre auf der Tabelle und `NS`-Sperren auf den gelesenen Tupeln.
  - Falls ein Tupel die `WHERE`-Bedingung nicht erfüllt, wird die Sperre gleich wieder freigegeben.
  - Ansonsten (das Tupel ist Teil des Ergebnisses) wird die Sperre bis Transaktionsende gehalten.
- `CS` funktioniert ähnlich (`IS`/`NS`-Sperren), aber gibt die Sperren immer gleich wieder frei.
- `UR` verwendet nur eine `IN`-Sperre auf der Tabelle.

SQL-Befehle und Sperren (3)

UPDATE:

- In diesem Fall werden normalerweise
  - eine IX-Sperre auf die Tabelle, und
  - X-Sperren auf die geänderten Tupel

  gesetzt.
- Ausnahmen gibt es nur für die Isolationsstufe RR.

  Hier wird eine X-Sperre auf die ganze Tabelle gesetzt, wenn klar ist, dass ohnehin alle Tupel betroffen sind (UPDATE ohne WHERE).
  Wenn die Bedingung mit einem "Full Table Scan" ausgewertet wird. Es wird eine SIX-Sperre auf die Tabelle gesetzt,

SQL-Befehle und Sperren (4)

Beispiel (Isolationsstufe CS):

- Transaktion A ändert den Stand von Konto 1001 von 100 auf 200 (noch kein `COMMIT`).
- Transaktion B führt folgende Anfrage aus:

```
SELECT * FROM KONTO WHERE STAND > 500
```

- Falls die Anfrage mit "Full Table Scan" ausgeführt wird, trifft sie bei der Ausführung auf das geänderte Tupel und muss warten.
- Wird die Anfrage dagegen mit einem Index ausgeführt, muss sie nicht warten.

SQL-Befehle und Sperren (5)

Bemerkung:

- DB2 bietet (offenbar seit Version 8) eine Option "Evaluate Uncommitted" (für Stufen `CS` und `RS`).
- Die `WHERE`-Bedingung der Anfrage wird dann für die ohne Sperre gelesenen Tupel getestet, und nur Tupel gesperrt, die die Bedingung erfüllen.
- Das gibt natürlich mehr Parallelität.
- Dann werden aber auch Tupel übergangen, die aufgrund eines (später mit `ROLLBACK` zurückgenommenen) Updates die Bedingung temporär verletzen.

Vergröberung von Sperren

- Wenn es für eine Tabelle viele Sperren auf Tupelebene gibt, kann sich das DBMS entscheiden, auf die Tabellenebene zu wechseln ("Lock Escalation").
- Das kann zu weniger Parallelität und möglicherweise zu Deadlocks führen.
- Aber der Overhead für die Verwaltung der Sperren sinkt.
- Mit folgendem Befehl kann man festlegen, dass für eine bestimmte Tabelle immer Sperren auf Tabellenebene gewählt werden sollen.

```
ALTER TABLE ⟨Tabelle⟩ LOCKSIZE TABLE
```

- Das wäre z.B. für Tabellen nützlich, auf die es (praktisch) nur Lese-Zugriffe gibt.

Literatur/Quellen

- Chamberlin: A Complete Guide to DB2 Universal Database. Morgan Kaufmann, 1998.
- H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, P. O'Neil: A critique of ANSI SQL isolation levels. In Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, 1–10, 1995.
- P.C. Zikopoulos, J. Gibbs, R.B. Melnyk: DB2 Fundamentals Certification for Dummies, 2001.
- R. Sanders: DB2 V8.1 Family Fundamentals Certification Prep, Part 6: Data Concurrency, IBM developerWorks, 2003.