Transaktionsverwaltung

Definition Transaktion

Unter einer Transaktion versteht man die *„Bündelung" mehrerer Datenbankoperationen*, die in einem Mehrbenutzersystem ohne unerwünschte Einflüsse durch andere Transaktionen als *Einheit fehlerfrei ausgeführt* werden sollen.[1]

„Bündelung" mehrerer Datenbankoperationen

Einheit fehlerfrei ausgeführt

ACID-Prinzip

Weiterführende Literatur:

- Kemper und Eickler, *Datenbanksysteme*, Kapitel 9.5 „Eigenschaften von Transaktionen", Seite 305
- *Qualifizierungsmaßnahme Informatik - Datenbanksysteme 5*, Seite 1
- *Wikipedia-Artikel „ACID"*

Atomicity Eine Transaktion ist atomar, d. h. von den vorgesehenen Änderungsoperationen auf die Datenbank haben entweder alle oder keine eine Wirkung auf die Datenbank.

Consistency Eine Transaktion überführt einen korrekten (konsistenten) Datenbankzustand wieder in einen korrekten (konsistenten) Datenbankzustand.

Isolation Eine Transaktion bemerkt das Vorhandensein anderer (parallel ablaufender) Transaktionen nicht und beeinflusst auch andere Transaktionen nicht.

Durability Die durch eine erfolgreiche Transaktion vorgenommenen Änderungen sind dauerhaft (persistent).

Operationen auf Transaktions-Ebene

Weiterführende Literatur:

- *Qualifizierungsmaßnahme Informatik - Datenbanksysteme 5*, Seite 5
- Winter, Lindner und Würdinger, *Einführung in relationale Datenbanksysteme & Datenmodellierung*, Seite 211
- Kemper und Eickler, *Datenbanksysteme*, Kapitel 9.3, Seite 302-303

read & write als Operationen zur Ausführung von Änderungen

begin of transaction (BOT) Mit diesem Befehl wird der Beginn einer eine Transaktion darstellende Befehlsfolge gekennzeichnet.

[1] *Qualifizierungsmaßnahme Informatik - Datenbanksysteme 5*, Seite 1.

commit Hierdurch wird die Beendigung der Transaktion eingeleitet. Alle Änderungen der Datenbasis werden durch diesen Befehl festgeschrieben, d. h. sie werden dauerhaft in die Datenbank eingebaut.

Selbstabbruch

abort Dieser Befehl führt zu einem *Selbstabbruch* der Transaktion. Das Datenbanksystem muss sicherstellen, dass die Datenbasis wieder in den Zustand zurückgesetzt wird, der vor Beginn der Transaktionsausführung existierte.

Für den Abschluss einer Transaktion gibt es 2 Möglichkeiten:

- Den erfolgreichen Abschluss mit `commit`.
- Den erfolglosen Abschluss mit `abort`

Zustandsübergänge einer Transaktion

Weiterführende Literatur:

- Kemper und Eickler, *Datenbanksysteme*, Kapitel 9.7, Seite 307
- *Qualifizierungsmaßnahme Informatik - Datenbanksysteme 5*, Seite 5

potentiell Die Transaktion ist codiert und „wartet darauf", in den Zustand aktiv zu wechseln. Diesen Übergang nennen wir inkarnieren.

aktiv Die aktiven (d. h. derzeit rechnenden) Transaktionen konkurrieren untereinander um die Betriebsmittel, wie z. B. Hauptspeicher, Rechnerkern zur Ausführung von Operationen, etc.

wartend Bei einer Überlast des Systems: (z. B. thrashing (Seitenflattern) des Puffers) kann die Transaktionsverwaltung einige aktive Transaktionen in den Zustand wartend verdrängen. Nach Behebung der Überlast werden diese wartenden Transaktionen sukzessive wieder eingebracht, d. h. wieder aktiviert.

abgeschlossen Durch den commit-Befehl wird eine aktive Transaktion beendet. Die Wirkung abgeschlossener TAS kann aber nicht gleich in der Datenbank festgeschrieben werden. Vorher müssen noch möglicherweise verletzte Konsistenzbedingungen überprüft werden.

persistent Die Wirkungen abgeschlossener Transaktionen werden — wenn die Konsistenzerhaltung sichergestellt ist — durch festschreiben dauerhaft in die Datenbasis eingebracht. Damit ist die Transaktion persistent. Dies ist einer von zwei möglichen Endzuständen einer Transaktionsverarbeitung.

gescheitert Transaktionen können aufgrund vielfältiger Ereignisse scheitern. Z. B. kann der Benutzer selbst durch ein abort eine aktive Transaktion abbrechen. Weiterhin können Systemfehler zum Scheitern aktiver oder wartender Transaktionen führen. Bei abgeschlossenen Transaktionen können auch Konsistenzverletzungen festgestellt werden, die ein Scheitern veranlassen.

wiederholbar Einmal gescheiterte Transaktionen sind u.U. wiederholbar. Dazu muss deren Wirkung auf die Datenbasis zunächst zurückgesetzt werden. Danach können sie durch Neustarten wiederum aktiviert werden.

aufgegeben Eine gescheiterte Transaktion kann sich aber auch als „hoffnungslos"herausstellen. In diesem Fall wird ihre Wirkung zurückgesetzt und die Transaktionsverarbeitung geht in den Endzustand aufgegeben über.

Fehlerbehandlung

„Einfacher" Fehlerfall

Änderungen durch nicht zu Ende geführte Transaktionen müssen *rückgängig gemacht werden*. Diesen Vorgang nennt man *Rollback*. Dazu müssen gegebenenfalls die zuletzt gültigen Werte der geänderten Datenbankobjekte in die Datenbank geschrieben werden.

rückgängig gemacht werden

Rollback

Es muss sichergestellt werden, dass die Änderungen erfolgreich zu Ende geführter Transaktionen tatsächlich *permanent* gemacht werden. Dazu müssen gegebenenfalls die *neuen Werte der Datenbankobjekte erneut* auf die Datenbank geschrieben werden.[2]

permanent

neuen Werte der Datenbankobjekte erneut

Das Lost-Update-Problem[3]

Änderungen in einer Datenbank, die durch eine Transaktion vorgenommen wurden, gehen aufgrund unkontrollierter Parallelausführung verloren.[4]

Änderungen

„Überschreiben"

Das Dirty-Read-Problem[5]

Der Wert eines Datenobjekts, der noch nicht permanent gespeichert wurde, wird *gelesen*.[6]

gelesen

„Zwischenstände lesen"

Das Unrepeatable-Read-Problem[7]

Während einer Transaktion wird der Wert eines Datenobjekts *zweimal gelesen*, wobei dieser Wert aber in der Zwischenzeit von einer anderen Transaktion *geändert* wurde.[8]

zweimal gelesen

geändert

„Verschiedene Zwischenstände lesen"

---

[2] *Qualifizierungsmaßnahme Informatik - Datenbanksysteme 5*, Seite 11.

[3] *Wikipedia-Artikel „Verlorenes Update"*.

[4] Kemper und Eickler, *Datenbanksysteme*, 11.1.1 Verlorengegangenee Änderungen, Seite 332.

[5] *Wikipedia-Artikel „Schreib-Lese-Konflikt"*.

[6] Kemper und Eickler, *Datenbanksysteme*, 11.1.2 Abhängigkeit von nicht freigegebenen Änderungen, Seite 332.

[7] *Wikipedia-Artikel „Nichtwiederholbares Lesen"*.

[8] Kemper und Eickler, *Datenbanksysteme*, 11.1.3 Phantomproblem, Seite 333.

Zweiphasige Abarbeitung mit Sperren[9]

Die Abarbeitung von Transaktionen erfolgt zweiphasig oder genügt dem 2-Phasen-Sperrprotokoll, wenn keine Transaktion eine Sperre freigibt, bevor sie alle benötigten Sperren angefordert hat.

- Ein strenges 2-Phasen-Sperrprotokoll wird dadurch realisiert, dass
  - eine Transaktion sukzessive alle benötigten Sperren anfordert (alle Elemente, die eine Transaktion lesen oder ändern möchte, werden von dieser Transaktion reserviert und dürfen nicht von anderen Transaktionen abgegriffen werden),
  - alle Lesesperren (**S**hared oder **R**ead) bis zum Ende, d. h. bis zur Aktion `commit`, hält und
  - alle Schreibsperren (e**X**clusive oder **W**rite) bis nach `commit` hält.
- Eine bereits gesetzte Lesesperre (durch Angabe von `rlock`) kann durch `xlock` in eine *Schreibsperre umgewandelt* werden, ohne dass vorher die Lesesperre aufgehoben werden muss.
- Reine *Lesesperren* auf ein Element können *mehrere* Transaktionen gleichzeitig besitzen, dann ist jedoch *keine Schreibsperre* mehr möglich.[10]

2-Phasen-Sperrprotokoll[11]

Die Serialisierbarkeit ist bei Einhaltung des folgenden Zwei-Phasen-Sperrprotokolls durch den Scheduler gewährleistet. Bezogen auf individuelle Transaktion wird folgendes verlangt:[12]

(a) Jedes Objekt, das von einer Transaktion benutzt werden soll, muss vorher entsprechend gesperrt werden.

(b) Eine Transaktion fordert eine Sperre, die sie schon besitzt, nicht erneut an.

(c) Eine Transaktion muss die Sperren anderer Transaktionen auf dem von ihr benötigten Objekt gemäß der Verträglichkeitstabelle beachten. Wenn die Sperre nicht gewährt werden kann, wird die Transaktion in eine entsprechende Warteschlange eingereiht — bis die Sperre gewährt werden kann.

(d) Jede Transaktion durchläuft zwei Phasen:
  - Eine Wachstumsphase, in der sie Sperren anfordern, aber keine freigeben darf und
  - Eine Schrumpfungsphase, in der sie ihre bisher erworbenen Sperren freigibt, aber keine weiteren anfordern darf.

(e) Bei `EOT` (Transaktionsende) muss eine Transaktion alle ihre Sperren zurückgeben

[9] *Qualifizierungsmaßnahme Informatik - Datenbanksysteme 5*, Seite 15.
[10] Kemper und Eickler, *Datenbanksysteme*, 11.6.2 Zwei-Phasen-Sperrprotokoll, Seite 346.
[11] *Qualifizierungsmaßnahme Informatik - Datenbanksysteme 5*, Seite 16.
[12] *Wikipedia-Artikel „Sperrverfahren“*.

Spezialfälle des 2-Phasensperrprotokolls

Es gibt zwei Spezialfälle von 2PL:

Konservatives 2-Phasen-Sperrprotokoll Das konservative 2-Phasen-Sperrprotokoll (Preclaiming), bei welchem zu Beginn der Transaktion alle benötigten Sperren auf einmal gesetzt werden. Dies verhindert in jedem Fall Deadlocks, führt aber auch zu einem hohen Verlust an Parallelität, da eine Transaktion ihre erste Operation erst dann ausführen kann, wenn sie alle Sperren erhalten hat.[13]

Striktes 2-Phasen-Sperrprotokoll Das strikte 2-Phasen-Sperrprotokoll, bei welchem alle gesetzten Write-Locks erst am Ende der Transaktion (nach der letzten Operation) freigegeben werden. Dieses Vorgehen verhindert den Schneeballeffekt, also das kaskadierende Zurücksetzen von sich gegenseitig beeinflussenden Transaktionen. Der Nachteil ist, dass Sperren häufig viel länger gehalten werden als nötig und sich somit die Wartezeit von blockierten Transaktionen verlängert. Die Read-Locks werden entsprechend dem Standard-2PL-Verfahren entfernt.[14][15]

Deadlock (bei Transaktionen)

Deadlock bei den sperrbasierten Synchronisationsmethoden:

- Beide Transaktionen können zunächst lesen
- Beim Umwandeln der Lesesperre in eine Schreibsperre in Schritt 9 kommt es zu einem Sperrkonflikt, da T2 noch die Lesesperre hält.
- Die Transaktion T1 wird vom System verzögert.
- In Schritt 10 möchte T2 ihre Lesesperre in eine Schreibsperre umwandeln.
- Es kommt erneut zu einem Sperrkonflikt, der wiederum zur Verzögerung der zweiten Transaktion führt.
- Deadlock-Situationen müssen entweder vermieden oder vom DatenbankMS erkannt und aufgelöst werden.
- Das DatenbankMS kann in diesem Fall eine der beiden Transaktionen abbrechen und diese von Neuem starten.

Recovery-Klassen: Wiederherstellungsmechanismen des DatenbankS nach Fehlern

Partial Undo (partielles Zurücksetzen / R1-Recovery) Nach Transaktionsfehler

[13] Kemper und Eickler, *Datenbanksysteme*, 11.7.2 Preclaiming zur Vermeidung von Verklemmungen, Seite 350.
[14] Kemper und Eickler, *Datenbanksysteme*, 11.6.3 Kaskadierendes Rücksetzen, Seite 348.
[15] *Wikipedia-Artikel „Sperrverfahren"*.

- Isoliertes und vollständiges Zurücksetzen der veränderten Daten in den Zustand zu Beginn der Transaktion
- Beeinflusst andere Transaktionen nicht!

Partial Redo (partielles Wiederholen / R2-Recovery) Nach Systemfehler (mit Verlust des Hauptspeicherinhalts)

- Wiederholung aller verlorengegangenen Änderungen (waren nur im Puffer) von abgeschlossenen Transaktionen

Global Undo (vollständiges Zurücksetzen / R3-Recovery) Nach Systemfehler (mit Verlust des Hauptspeicherinhalts), z. B. Stromausfall

- Zurücksetzen aller durch den Ausfall abgebrochenen Transaktionen

Global Redo (vollständiges Wiederholen / R4-Recovery) Nach Gerätefehler

- Einspielen einer Archivkopie auf neuen Datenträger und Nachvollziehen aller beendeten Transaktionen, die nach der letzten beendeten Transaktion auf der Archivkopie noch ausgeführt wurden

Puffer-Verwaltung[16]

Wann werden Änderungen, die von Transaktionen ausgelöst wurden, vom temporären Puffer in die Datenbank geschrieben (permanent)?

Verdrängungsstrategien[17]

Ersetzung „schmutziger" Seiten im Puffer

No-Steal Schmutzige Seiten dürfen nicht aus dem Puffer entfernt und in die Datenbank übertragen werden, solange die Transaktion noch aktiv ist. Die Datenbank enthält keine Änderungen nicht-erfolgreicher Transaktionen. Eine UNDO-Recovery ist nicht erforderlich. langen Änderungs-Transaktionen können zu Problemen führen, da große Teile des Puffers blockiert werden

Steal Schmutzige Seiten dürfen jederzeit ersetzt und in die Datenbank eingebracht werden. Die Datenbank kann unbestätigte Änderungen enthalten. Eine UNDO-Recovery ist erforderlich. Es handelt sich um eine effektivere Puffernutzung bei langen Transaktionen mit vielen Änderungen.

[16] Kemper und Eickler, *Datenbanksysteme*, Kapitel 10.2.1 Ersetzung von Puffer-Seiten Seite 311.

[17] Kemper und Eickler, *Datenbanksysteme*, Kapitel 10.2.1 Ersetzung von Puffer-Seiten Seite 311-312.

Ausschreibestrategien (EOT-Behandlung)[18]

Force Alle geänderten Seiten werden spätestens bei EOT (vor COMMIT) in die Datenbank geschrieben. Bei einem Systemfehler ist keine REDO-Recovery erforderlich. Die Force-Strategie benötigt einen hohen I/O-Aufwand, da Änderungen jeder Transaktion einzeln geschrieben werden. Die Vielzahl an Schreibvorgängen führt zu schlechteren Antwortzeiten, länger gehaltenen Sperren und damit zu mehr Sperrkonflikten. Große Datenbank-Puffer werden schlecht genutzt.

No-Force Änderungen können auch erst nach dem COMMIT in die Datenbank geschrieben werden. Die Änderungen durch mehrere Transaktionen werden „gesammelt". Beim COMMIT werden lediglich REDO-Informationen in die Log-Datei geschrieben. Bei einem Systemfehler ist eine REDO-Recovery erforderlich. Die Änderungen auf einer Seite über mehrere Transaktionen hinweg können gesammelt werden.[19]

Das WAL-Prinzip[20]

Wir hatten gerade bemerkt, dass die Log-Einträge spätestens dann geschrieben werden müssen, wenn sich der zur Verfügung stehende Ringpuffer gefüllt hat. Bei der von uns zugrundegelegten Systemkonfiguration (force, steal, und update-in-place) ist aber zusätzlich unabdingbar, das sogenannte WAL-Prinzip (Write Ahead Log) einzuhalten. Dafür gibt es zwei Regeln, die beide befolgt werden müssen:

(a) Bevor eine Transaktion festgeschrieben (committed) wird, müssen alle „zu ihr gehörenden" Log-Einträge ausgeschrieben werden.

(b) Bevor eine modifizierte Seite ausgelagert werden darf, müssen alle Log-Einträge, die zu dieser Seite gehören, in das temporäre und das Log-Archiv ausgeschrieben werden.

Die erste Regel des WAL-Prinzips ist notwendig, um erfolgreich abgeschlossene Transaktionen nach einem Fehler nachvollziehen (Redo) zu können. Die zweite Regel wird benötigt, um im Fehlerfall die Änderungen nicht abgeschlossener Transaktionen aus den modifizierten Seiten der materialisierten Datenbasis entfernen zu können.

Beim WAL-Prinzip schreibt man natürlich alle Log-Einträge bis zu dem letzten notwendigen aus — d. h. man übergeht keine Log-Einträge, die von Regel 1. und 2. nicht erfasst sind. Dies ist essentiell, um die chronologische Reihenfolge der Log-Einträge im Ringpuffer zu wahren.

Literatur

[1] Alfons Kemper und André Eickler. *Datenbanksysteme. eine Einführung*. 2013.

[18] Kemper und Eickler, *Datenbanksysteme*, Kapitel 10.2.2 Einbringen von Änderungen einer Transaktion Seite 312-313.

[19] *Qualifizierungsmaßnahme Informatik - Datenbanksysteme 5*, Seite 25.

[20] Kemper und Eickler, *Datenbanksysteme*, Kapitel 10.3.5 Seite 318.

[2] *Qualifizierungsmaßnahme Informatik - Datenbanksysteme 5. Transaktionen, Fehler und Recovery*. https://www.studon.fau.de/file2480906_download.html.

[3] *Wikipedia-Artikel „ACID“*. https://de.wikipedia.org/wiki/ACID.

[4] *Wikipedia-Artikel „Nichtwiederholbares Lesen“*. https://de.wikipedia.org/wiki/Nichtwiederholbares_Lesen.

[5] *Wikipedia-Artikel „Schreib-Lese-Konflikt“*. https://de.wikipedia.org/wiki/Schreib-Lese-Konflikt.

[6] *Wikipedia-Artikel „Sperrverfahren“*. https://de.wikipedia.org/wiki/Sperrverfahren.

[7] *Wikipedia-Artikel „Verlorenes Update“*. https://de.wikipedia.org/wiki/Verlorenes_Update.

[8] Stefan Winter, Annabel Lindner und Markus Würdinger. *Einführung in relationale Datenbanksysteme & Datenmodellierung*. https://www.studon.fau.de/file2686598_download.html.