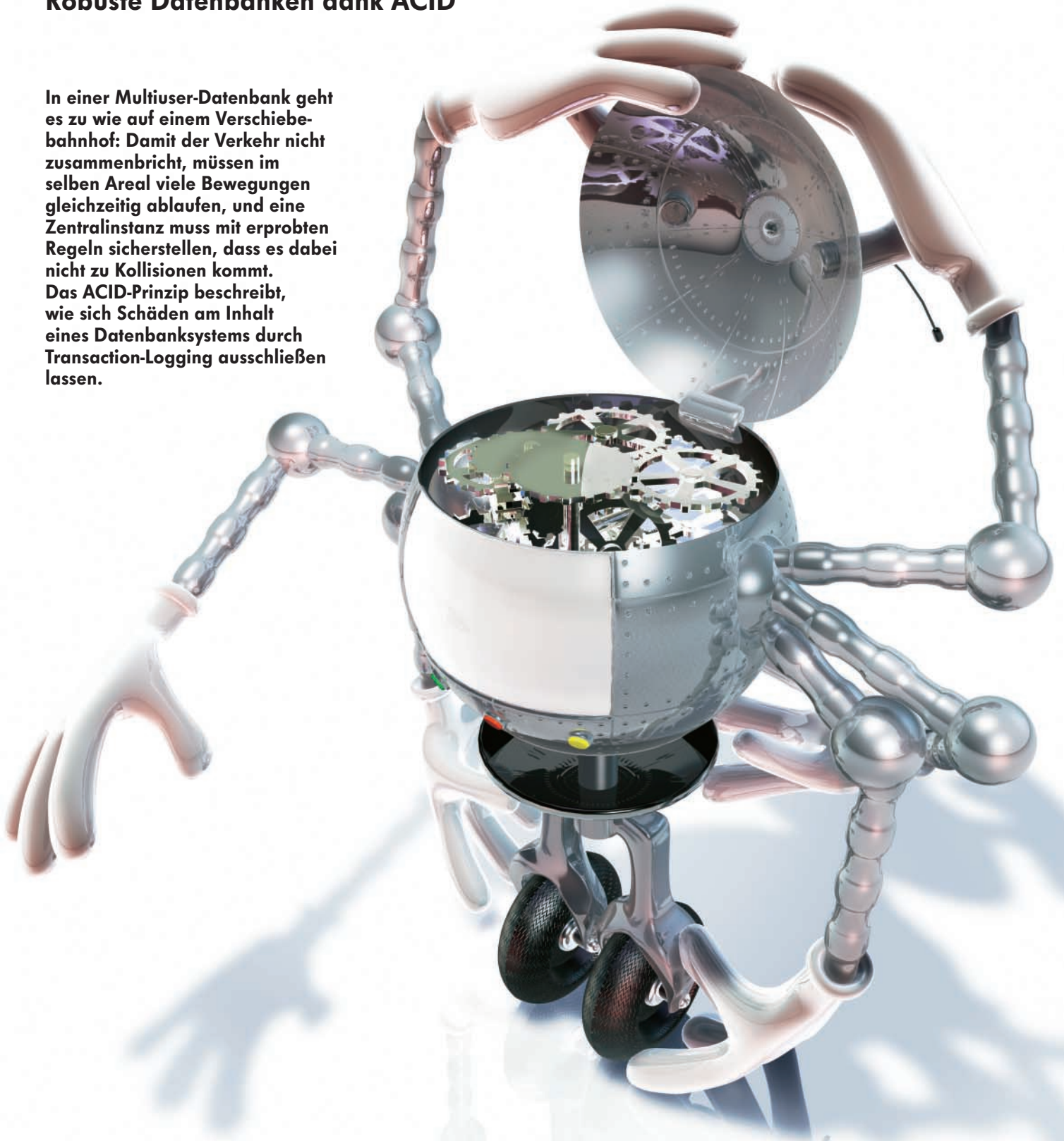


Kaj Arnö

Transaktionen im Griff

Robuste Datenbanken dank ACID

In einer Multiuser-Datenbank geht es zu wie auf einem Verschiebebahnhof: Damit der Verkehr nicht zusammenbricht, müssen im selben Areal viele Bewegungen gleichzeitig ablaufen, und eine Zentralinstanz muss mit erprobten Regeln sicherstellen, dass es dabei nicht zu Kollisionen kommt. Das ACID-Prinzip beschreibt, wie sich Schäden am Inhalt eines Datenbanksystems durch Transaction-Logging ausschließen lassen.



Soll ein Datenbanksystem zugleich mehreren Benutzern Zugriff auf die ihm anvertrauten Inhalte gewähren, muss es mehr können als nur Tabellen und Indizes verwalten. Man stelle sich vor, ein Banker will ein Konto gerade in dem Moment von DM auf Euro umstellen, wo ein Kollege 100 Euro von dort abbucht und den Kontostand um 195 vermeintliche DM reduziert. Ein Bankkunde, der danach den Gegenwert von 95 DM vermisst, wird dem Datenbanksystem vermutlich nicht mehr viel Vertrauen entgegenbringen.

Um solche Pannen zu vermeiden, muss eine Multiuser-Datenbank die Zugriffe der beiden Sachbearbeiter koordinieren. Das muss allerdings nicht auf die klassische Lösung hinauslaufen, Zugriffe des einen so lange zu sperren, bis der andere seine Änderungen abgeschlossen hat. Schließlich entstehen bei diesem 'Locking' genannten Verfahren zumindest für einen Benutzer Wartezeiten. Je seltener es zum Locking kommt, desto zügiger läuft die Datenbank.

Ein Kunstgriff erlaubt es, Datensätze auch dann zu verändern, wenn sie eigentlich gesperrt sein müssten: Man reicht seine Änderungswünsche einfach nur beim Datenbanksystem ein und verlässt sich darauf, dass die Software die gewünschten Aktionen bei passender Gelegenheit ohne Störung durch andere Benutzer und ohne wesentliche Verzögerung ausführt.

Die Kommandofolgen für solche Bearbeitungswünsche erstrecken sich in der Structured Query Language SQL von einem *BEGIN* bis zum abschließenden *COMMIT* und kommen in einem darauf vorbereiteten Datenbanksystem als jeweils eigener Vorgang ohne Fremdeinwirkung zur Ausführung. Diese Vorgänge heißen Transaktionen.

Jeder Banker kann seine Transaktion mit den nötigen Arbeitsschritten in Auftrag geben. Bei geeigneter Ausdrucksweise ist sichergestellt, dass die Datenbank zunächst die Euro-Umstellung abschließt und dann erst den Scheck gutbucht oder umgekehrt. Gegebenenfalls notwendige Sperren errichtet die Datenbank selbstständig.

'Geeignete Ausdrucksweise' will sagen, dass Unstimmigkeiten nicht schon aus der Anwen-

dungslogik entstehen sollen. Anwendungen, die den Erfolg der Arbeitsschritte einer Transaktion auf Plausibilität kontrollieren, können den Vorgang durch den alternativen Transaktionsabschluss mit *ROLLBACK* an Stelle von *COMMIT* für ungültig erklären. In diesem Fall verwirft das Datenbanksystem alle bislang angeforderten Änderungen.

Säureschutz

Eine sinnvolle Benutzung von Transaktionen kann die Leistung, die Sicherheit und sonstige Eigenschaften einer Datenbankanwendung dramatisch beeinflussen. 'Transaktion' ist indes nur eine ungenaue Beschreibung. Ein tiefer gehendes Verständnis des Transaktionsmodells hilft, einzelne Veränderungen am Datenbestand fehlerfrei miteinander zu koordinieren. Dieser Artikel stellt das dafür nützliche ACID-Konzept vor. Einen Ausblick, wie Transaktionsmechanismen realisiert sind und was dabei wichtig ist, liefert der Kasten auf Seite 150.

Die Abkürzung ACID beschreibt allgemein akzeptierte Anforderungen an Datenbanktransaktionen. A steht für Atomicity, C für Consistency, I für Isolation und D für Durability. Betrachtet man jedoch die meistverkauften Datenbanken, etwa Oracle, IBM DB2 und Microsoft SQL Server, stößt man auf unterschiedliche Interpretationen dieser Begriffe. In einigen Engines lassen sich manche ACID-Eigenschaften zur Verbesserung der Performance abschalten, andere fallen gleich ganz unter den Tisch. Die einzelnen Begriffe hinter dem Konzept erklären sich wie folgt.

A wie Atomicity

A steht für Atomarität (Atomicity) und bedeutet, dass eine Transaktion unteilbar ist. Eine Transaktion findet entweder vollständig statt, oder die Datenbank bleibt davon vollkommen unbeeinflusst.

Als das klassische Beispiel für Atomicity gilt eine Banktransaktion, wo 100 Euro von Konto A auf Konto B zu überweisen sind. Entweder wird der Saldo von Konto A um 100 Euro kleiner und von Konto B um 100 Euro größer, oder nichts geschieht.

Ein Zustand, wo der Saldo von Konto A um 100 Euro schrumpft, ohne dass dieses Geld seinen Weg auf Konto B gefunden hat, ist logischerweise unakzeptabel. Das Datenbanksystem muss gewährleisten, dass so ein Zustand auch im schlimmsten Falle, bei einem Stromausfall oder Betriebssystem-Crash, unmöglich ist.

C wie Consistency

Consistency (Konsistenz) stellt sicher, dass eine Transaktion die Datenbank in einem gültigen Zustand hinterlässt. Die Kriterien, welcher Zustand als gültig zu betrachten ist, hängen von der jeweiligen Problemsituation ab: Tauchen etwa in einer Buchungstransaktion unterschiedliche Soll- und Habenbeträge auf, gerät die Datenbank nur durch falsche Anwendungsprogrammierung in die Bredouille. Anders, wenn etwa das Einfügen eines Datensatzes zum Hochzählen eines Autoinkrement-Werts führt: Dann fällt es in die Verantwortung des Datenbanksystems, dafür zu sorgen, dass jede Transaktion einen eindeutigen Wert für dieses Feld erhält.

I wie Isolation

Bei der Isolation einer Transaktion zeigen sich die größten Unterschiede zwischen verschiedenen Implementierungen und die größten Verstöße gegen das Ideal. Die Idee hinter dem Begriff Isolation besagt, dass gleichzeitig ausgeführte Transaktionen keinen Einfluss aufeinander haben sollen. Wenn eine Transaktion anfängt, sieht sie die Datenbank im Idealfall – bis auf die von ihr selbst veranlass-

Kaj Arnö

Der Autor dieses Beitrags ist beim Datenbankhersteller MySQL AB für Schulung sowie Beratung von Anwendern zuständig und arbeitet zurzeit von München aus. Die Beispiele im Text beziehen sich aus diesem Grunde auf MySQL, auch wenn sich die umrissenen Konzepte bei anderen Datenbankprogrammen ähnlich aufzeigen ließen.

ten Änderungen – bis zum Abschluss durch den Befehl *COMMIT* als vollkommen statisch und unverändert an.

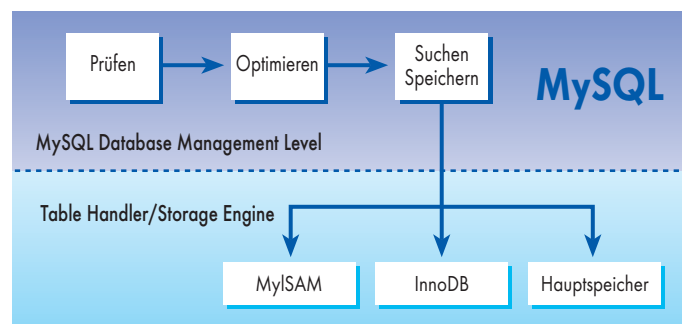
Diese Art von Isolation zu gewährleisten, kostet einigen Aufwand. Theoretisch und auch praktisch können viele parallele Transaktionen die gleichen Daten betreffen. Manche dieser Operationen dauern länger als andere; manche ändern die Dateninhalte, andere lesen sie nur. Das wird an zwei gleichzeitig auszuführenden Gruppen von Transaktionen deutlich:

- Lang dauernde Transaktionen A, die viele Einträge der Tabelle für Aufträge lesen, und
- kurze Transaktionen B, die einen bestehenden Datensatz der Auftrags-tabelle aktualisieren oder einen neuen hinzufügen.

Während einer A-Transaktion laufen mehrere B-Transaktionen. Streng gesehen muss der Befehl

```
SELECT SUM(Umsatz) FROM Auftrag
```

während der ganzen A-Transaktion das gleiche Ergebnis liefern, ohne Rücksicht darauf, ob Umsätze aktualisiert oder wieviele neue Aufträge von anderen Transaktionen hinzugefügt wurden, seit die A-Transaktion



Die Datenbank MySQL bedient unter einem einheitlichen Verwaltungs-Level wahlweise mehrere Storage Engines mit unterschiedlichen Funktionsschwerpunkten.

angefangen hat. Eine Isolation auf diesem Niveau nennt sich im Jargon 'repeatable Read', da man lesende Zugriffe während einer Transaktion wiederholen kann, ohne abweichende Ergebnisse zu erhalten.

Beim Versioning, der elegantesten Realisierung der Isolation, bekommt jede Transaktion eine eigene Momentaufnahme der Datenbank zu sehen. Eigentlich müssten die gesamten Daten dazu kopiert werden – praktisch geht das deutlich eleganter (was hinter den Kulissen passiert, erklärt der Kasten unten für MySQL). Transaktionen, die nur Daten lesen, müssen in diesem Modus nicht auf gesperrte

Inhalte Rücksicht nehmen, sehr zugunsten des Datendurchsatzes.

Transaktionen blockieren sich auf diese Weise nicht gegenseitig, sondern finden quasi in verschiedenen zeitlichen Ebenen der Datenbank statt. Nicht immer ist das erwünscht. Je nachdem, welchen Stellenwert man der Isolation beimisst, kommt die so genannte 'serializable Execution' infrage, oder, bei abnehmenden Isolationsanforderungen, die Praktiken 'repeatable Read', 'Read committed' oder 'Read uncommitted'.

Die serializable Execution isoliert Transaktionen vollkommen voneinander. So genannte Phantomdatensätze können hier

nicht auftauchen; sie kommen bei repeatable Read und niedrigeren Isolationsstufen zustande, wenn eine langlebige Transaktion beim Lesen auch solche Datensätze zu sehen bekommt, die erst nach ihrem Beginn als Ergebnis einer anderen Transaktion entstanden sind.

Eine weniger strenge Form von Isolation resultiert beim Read committed. Hier kann ein wiederholter Lesebefehl innerhalb einer langen Transaktion unterschiedliche Ergebnisse hervorrufen, da auch zwischenzeitlich von anderen Operationen aktualisierte Daten zum Vorschein kommen. Änderungen innerhalb anderer Transaktionen

– etwa dass während einer Banküberweisung der Saldo von Konto A kleiner wurde, bevor der Saldo von Konto B gewachsen ist – sind jedoch nicht sichtbar. Wären sie sichtbar, läge die Isolationsebene 'Read uncommitted' vor, dort besteht überhaupt keine Isolation mehr.

D wie Durability

Durability bedeutet, dass eine Transaktion auch dann Bestand hat, wenn das System eine Mikrosekunde nach ihrer Ausführung zusammenbricht, etwa auf Grund eines Stromausfalls.

Nach einem Ausfall des Betriebssystems oder der Strom-

Hinter den Kulissen: MySQL mit oder ohne Transaktionen

MySQL besteht aus zwei Schichten: Die obere, der Database Management Level, stellt die Schnittstelle zum Benutzer dar, auf die er mit SQL-Befehlen zugreifen kann. Darunter liegt die eigentliche Storage Engine, die sich um das Aufbewahren der Daten und die Zugriffe darauf kümmert. Entsprechend lässt sich diese untere Schicht austauschen, etwa mit Transaktionen (InnoDB), optimiert für Lesezugriffe (MyISAM) oder für optimierte Zugriffszeiten aus dem Hauptspeicher (Heap).

InnoDB verzichtet darauf, Betriebssystemfunktionen zur Dateiverwaltung zu nutzen, und bringt stattdessen ein eigenes Dateisystem mit. Das mag auf den ersten Blick wie unnötige Mehrarbeit aussehen, gibt aber dem Datenbanksystem eine bessere Kontrolle über die Organisation und Antwortzeiten. InnoDB speichert alle Datenbanken, Tabellen inklusive Indizes und Beschreibungen in einer Datei, dem so genannten Table Space.

Das Aktualisieren des Table Space kostet immer etwas Zeit, besonders, wenn es um eine große Transaktion geht, die Hunderte oder gar Tausende Datensätze betrifft. Damit die Datenbank trotzdem stets schnell auf Befehle reagieren kann, puffert das System sowohl den Table Space als auch die Log-Dateien auszugsweise im Hauptspeicher.

InnoDB führt zwei Logs: Das so genannte Redo Log unterstützt InnoDB beim Hochfahren des Datenbanksystems nach einem Ausfall. Hier finden sich Datensatzänderungen, die zwar mittels *COMMIT* abgeschlossen sind und im Interesse der Durability bestehen bleiben sollen, die aber noch nicht ihren Weg in den Table Space gefunden haben.

Das Undo Log enthält Datensätze, die noch nicht per *COMMIT* 'abgesegnet' sind, sowie frühere Versionen dieser Einträge. InnoDB benutzt diese Datei für den Rollback-Befehl und als Zwischenablage, wenn die Aufzeichnungen für besonders große Transaktionen nicht ins Redo Log passen.

Die Parameter für die Größe dieser Pufferbereiche beeinflussen die Performance von MySQL-Datenbanken und bieten sich für eine Optimierung im Einzelfall an.

Die Isolation, das I aus ACID, verlangt, dass unterschiedliche Transaktionen nichts voneinander erfahren. Durch das Versioning soll jede neue Transaktion quasi eine Kopie der ganzen Datenbank bekommen, und zwar genau in dem Augenblick, wo die Transaktion begann. Das Datenbanksystem hat keine Chance, zu jedem Transaktionsbeginn Daten zu kopieren, zumal es zu diesem Zeitpunkt keine Ahnung hat, auf welche Daten die Transaktion über-

haupt zugreifen wird. Deshalb wird Versioning gerade durch das Undo Log erreicht.

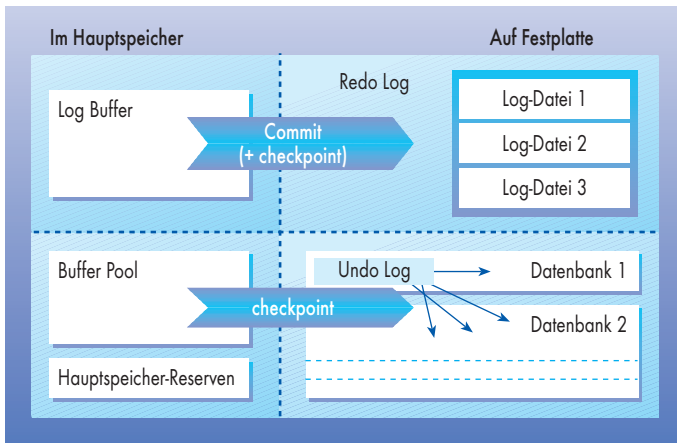
Das Undo Log wird von dem InnoDB Rollback Segment gehandhabt. Jede Transaktion hat in und für sich schon zwei Listen von Datensätzen – eine für solche, die per *INSERT* neu angelegt wurden und im Falle eines *ROLLBACK* einfach verworfen werden können, und eine andere für Veränderungen durch *UPDATE* und *DELETE*, wo bei einem *ROLLBACK* die frühere Version des Datensatzes wieder aktuell werden soll.

Nach einem *COMMIT* werden die früheren Versionen des Datensatzes nicht direkt vergessen. Nicht dass die neue Version jemals wieder in den früheren Stand zurückgesetzt werden könnte (das widerspricht der Forderung nach Durability), doch die älteren Versionen könnten in anderen Transaktionen benötigt werden, die zum Zeitpunkt des *COMMIT* schon aktiv, aber noch nicht beendet waren. Die für andere aktive Transaktionen notwendigen Datensätze stehen in der History-Liste und werden erst bei passender Gelegenheit durch InnoDBs Hintergrundprozess Purge gelöscht.

Neben dem Purge-Hintergrundprozess, der den Speicherplatz für gelöschte Daten wieder zur Verfügung stellt, gibt es einen anderen Hintergrund-

prozess, der beim Passieren so genannter Checkpoints dafür sorgt, dass Daten aus Hauptspeicher in den Table Space auf der Festplatte geschrieben werden. Checkpoints sind unabhängig von *COMMIT*-Befehlen; die Tatsache, dass ein Datensatz im Redo Log vermerkt ist, reicht aus, um die Durability zu gewährleisten. Dahingegen sind Checkpoints notwendig, um die Antwortzeit beim Hochfahren des Datenbanksystems nach einem Ausfall zu begrenzen – je häufiger die Checkpoints, umso weniger Arbeit nach einem Störfall. Beim Setzen dieser Checkpoints lässt sich InnoDB nicht reinreden, weder von Datenbankbenutzern noch von einem Administrator.

Ohne Eingriffe vom Benutzer oder vom Admin läuft auch der Recovery-Prozess. Beim Ausfall des Datenbanksystems gehen die Hauptspeicherinhalte verloren und damit auch die offenen Transaktionen. Abgeschlossene Transaktionen lassen sich im Unterschied dazu anhand des Redo Log rekonstruieren. Das Recovery mit InnoDB unter MySQL macht einerseits die halbfertigen, nicht abgeschlossenen Datenbankänderungen anhand des Undo Log rückgängig, andererseits wiederholt es die fertigen Datenbankänderungen, die bislang nur im Redo Log, aber noch nicht im Table Space registriert sind.



InnoDB speichert das Redo Log vorerst im Log Buffer, der sich auf dem Massenspeicher in drei Log-Dateien für Einfügungen, Löschungen und Änderungen niederschlägt; das Undo Log liegt zusammen mit den Datensätzen aller Datenbanken im gemeinsamen Table Space.

versorgung fehlen die Daten aus dem Hauptspeicher des Rechners oder sind zumindest nicht mehr verlässlich. Durability bedeutet daher, dass der gesamte Hauptspeicher-Inhalt anhand von Log-Dateien rekonstruiert werden kann, die die Datenbank beständig mitführt.

Genau wie bei der Isolation ist die Welt auch in Sachen Durability nicht schwarzweiß. Vielleicht kann man im Einzelfall den Verlust der Transaktionen aus der letzten Sekunde vor einem Betriebssystem-Crash in Kauf nehmen, etwa weil höchstens Web-Statistiken, aber keine Kundenbestellungen betroffen sein könnten. Hier liefern die meisten Datenbanken Optionen, um sie auf den jeweiligen Anwendungsfall anzupassen.

Es kann nur einen geben

Ein Allheilmittel stellt Transaction Logging freilich nicht dar, wenn verschiedene Anwender zugleich dasselbe Datenfeld verändern wollen. Allerdings liefern Transaktionen hierfür verschiedene Lösungsansätze, die dann in der Anwendungslogik zu berücksichtigen sind. Eine Transaktion kann schreibende Zugriffe anmelden (*SELECT FOR UPDATE*) und dadurch andere Bewerber ausstechen; die Datenbank errichtet dann vielleicht ein Row-Level-Lock.

Alternativ kann eine Anwendung Datensätze mit Zeitstempeln versehen und sie bei

UPDATE-Statements als Bedingung einbeziehen. Mit Hilfe eines *ROLLBACK*-Befehls kann die Anwendung komplexe Änderungen, die dem fehlgeschlagenen *UPDATE*-Statement vorausgingen, rückgängig machen und die Daten in einem konsistenten Zustand belassen.

Die Entwickler verbreiteter Multiuser-Datenbanken sind sich im Großen und Ganzen über die Aufgaben dieser Engines einig. Sie verwenden sogar gleiche Begriffe für die gestellten Anforderungen, was den Vergleich der einzelnen Problemlösungen vereinfacht. Es zeigt sich aber, dass im Detail durchaus unterschiedliche Realisierungen infrage kommen.

So verzichtet der Microsoft-SQL-Server aufs Versioning und verstößt damit im voreingestellten Modus gegen eine streng interpretierte Isolation. Nur wer bereit ist, deutliche Einbußen am Datendurchsatz hinzunehmen, kann auch hier die Stufe *SERIALIZABLE* auswählen.

Die verschiedenen Anbieter wählen unterschiedliche Isolationsarten als Standard. In der Regel kann man sie an die eigenen Bedürfnisse anpassen. Nicht für alle Anwendungen benötigt man den höchsten Grad an Transaktionssicherheit. Die optimale Performance bekommt man bei allen Datenbanksystemen nur dann, wenn man den geeigneten Kompromiss zwischen höchster Datensicherheit und schnellster Verarbeitung findet. (hps)