Transaktionen und die Integrität von Datenbanken

Udo Kelter

15.04.2003

Zusammenfassung dieses Lehrmoduls

Dieses Lehrmodul führt in die Problematik der Erhaltung der Integrität einer Datenbank ein. Es werden drei Arten von Gefährdungen der Integrität unterschieden: Fehler bei Dateneingaben oder in Anwendungsprogrammen, Betriebsstörungen wie z.B. Systemabstürze oder Mediendefekte und paralleler Zugriff mehrerer Anwendungen auf die gleichen Daten. Für alle drei Arten von Gefährdungen werden Gegenmaßnahmen im DBMS beschrieben; alle beruhen auf dem zentralen Konzept der Transaktion. Einleitend wird analysiert, welche Arten von Korrektheit unterschieden und sinnvollerweise gefordert werden können.

Vorausgesetzte Lehrmodule:

empfohlen: – Datenverwaltungssysteme

Stoffumfang in Vorlesungsdoppelstunden: 1.0

Inhaltsverzeichnis

1 Die Integrität von Datenbanken

Die Korrektheit bzw. Integrität der Daten in einer Datenbank genießt sehr hohe Priorität; dementsprechend enthalten DBMS vielfältige Mechanismen, deren Zweck darin besteht, die Integrität der Daten sicherzustellen. Man unterscheidet drei Hauptklassen von Gefährdungen der Integrität:

1. Fehler bei Dateneingaben oder in Anwendungsprogrammen
2. Betriebsstörungen wie z.B. Systemabstürze oder Mediendefekte
3. paralleler Zugriff mehrerer Anwendungen auf die gleichen Daten

Die erste Klasse von Gefährdungen kann nur dadurch bekämpft werden, daß die korrekten Inhalte einer Datenbank dem DBMS möglichst genau beschrieben werden, so daß das DBMS Veränderungen, die zu einem inkorrekten Zustand führen, ablehnen kann. Entsprechende Konzepte können nur in engem Zusammenhang mit dem Datenbankmodell diskutiert werden.

Die zweite und dritte Klasse von Gefährdungen, auf die wir uns in diesem Lehrmodul konzentrieren werden, sind dagegen weitgehend unabhängig vom Datenbankmodell.

Bevor wir die Gefährdungen und die Maßnahmen dagegen diskutieren können, muß zunächst analysiert werden, was genau unter Korrektheit zu verstehen ist. Es zeigt sich, daß intuitiv naheliegende Interpretationen dieses Begriffs nicht brauchbar sind.

1.1 Eine Hierarchie von Korrektheitsbegriffen für Datenbank-Zustände

Integrität. Unter **Integrität** (Adjektiv: **integer**) sei verstanden, daß der Inhalt einer Datenbank einen interessierenden Teil bzw. eine Abstraktion der Realität genau (exakt), richtig und aktuell gültig wiedergibt. Beispiele für unerwünschte bzw. falsche Datenbankinhalte sind:

- unmögliche Werte (Kinderzahl = -3),
- Inkonsistenzen (z.B. eine Person ist verheiratet, hat aber Steuerklasse I)

– veraltete Angaben.

Es gibt somit verschiedene Aspekte der Korrektheit einer Datenbank. Unter Integrität wird diese Korrektheit in ihrem weitesten, umfassendsten Sinn verstanden. Oft wird diese generelle Korrektheit auch mit Konsistenz oder einfach nur mit Korrektheit bezeichnet.

Wir befassen uns hier in diesem Lehrmodul nur mit Maßnahmen, die ein DBMS durchführen kann, um die Korrektheit der Daten sicherzustellen. Nicht garantieren kann das DBMS die Aktualität der Daten, insoweit diese davon abhängt, daß rechtzeitig Daten eingegeben oder Änderungsprogramme gestartet werden.

Nicht befassen werden wir uns in diesem einführenden Lehrmodul auch mit Problemen, die spezifisch für einzelne Datenbankmodelle sind. Viele Probleme bei der Integritätssicherung sind in der Tat unabhängig vom Datenbankmodell. Wir werden daher i.f. vereinfachend annehmen, daß eine Datenbank eine Menge von **Datenbank-Objekten** enthält, die einen Zustand haben, und daß die Datenbank durch die Operationen des Datenbankmodells, die wir hier auch als **Aktionen** bezeichnen werden, verändert wird.

Statt vom Inhalt einer Datenbank, was vielleicht anschaulicher wäre, werden wir meist vom **Zustand der Datenbank** sprechen.

Erreichbarkeit. Unser Ziel ist die Beschreibung von integren Zuständen, also "völlig korrekten" Zuständen. Diesem Ziel nähern wir uns in mehreren Schritten. Zunächst können wir uns auf erreichbare Zustände beschränken. Ein Zustand heißt **erreichbar**, wenn er nach Initialisierung der Datenbank durch eine endliche Folge von Aktionen (mit zulässigen bzw. syntaktisch korrekten Parametern) konstruiert werden kann.

Logische Konsistenz. Ein erreichbarer Zustand ist nicht sinnvoll, wenn er etwas Unmögliches darstellt, also unrealistisch ist. Ein erreichbarer Zustand heißt **logisch konsistent**, wenn eine Realität möglich ist, die durch ihn korrekt wiedergegeben wird.

Nicht jeder erreichbare Datenbank-Zustand ist realistisch. Bei-

spielsweise können Werte einzelner Datenbank-Objekte schon für sich allein unrealistisch sein (Kinderzahl = -3), oder es kann bei redundanten Angaben in mehreren Objekten ein Widerspruch enthalten sein.

Unter Konsistenz versteht man oft einen speziellen Aspekt der logischen Konsistenz, wie sie oben definiert wurde, nämlich die gegenseitige Konsistenz zwischen mehreren, insgesamt redundanten Einzelangaben und ggf. Eigenschaften der Objektmenge. Beispiele für einzuhaltende Konsistenzbedingungen sind:

- die Summe aller Konten ist gleich 0,
- der Inhalt eines Felds "Zahl der Angestellten" ist gleich der Zahl der Angestelltensätze oder -tupel.

Wir werden hier jedoch unter **Konsistenz** stets die oben definierte allgemeinere Form von logischer Konsistenz verstehen.

Die vorstehenden Konsistenzbedingungen können i.a. nicht durch die elementaren Schemadefinitionskonzepte ausgedrückt werden; es kann also erreichbare, aber nicht logisch konsistente Zustände geben.

Physische Konsistenz. Mit **physischer Konsistenz** bezeichnet man den ordnungsgemäßen Zustand der internen Speicherungsstrukturen der Datenbank, vor allem der Zugriffspfade. Diese sind auf der Ebene von Datenbank-Objekten nicht sichtbar bzw. zugreifbar, sondern nur innerhalb des Laufzeitkerns des DBMS, in dem die Aktionen realisiert werden. Wir gehen davon aus, daß alle Aktionen die physische Konsistenz erhalten. Jeder erreichbare Zustand ist also physisch konsistent. Physisch konsistente, aber nicht erreichbare Zustände können z.B als Zwischenzustände innerhalb von Aktionsausführungen auftreten.

Damit verfügen wir bereits über eine kleine Hierarchie von Arten der Korrektheit eines Datenbank-Zustandes:

- Integrität
- logische Konsistenz
- Erreichbarkeit
- physische Konsistenz

Da wir das richtige Funktionieren der Aktionen voraussetzen, stellt die physische Konsistenz für uns kein Problem dar; wir werden uns nur mit erreichbaren Zuständen befassen.

Die logische Konsistenz ist eine notwendige, aber keine hinreichende Voraussetzung für die Integrität. Die logische Konsistenz der Datenbank ist eine minimale Form der Korrektheit, die fast immer unverzichtbar ist. Sie besagt noch nicht, daß der Zustand der Datenbank auch die Realität aktuell korrekt wiedergibt.

Auf die völlige Integrität der Datenbank kann hingegen bei manchen Anwendungen verzichtet werden; oft werden Abstriche an der Aktualität der Daten in Kauf genommen, um den Aufwand zu reduzieren. In diesem Fall liegt eine Zwischenform von Korrektheit vor. Wir werden hier solche anwendungsbezogenen Zwischenformen nicht betrachten.

1.2 Die Integritätsanforderung - 1. Ansatz

Da das Hauptziel die Sicherstellung der Korrektheit der Datenbank ist, könnte man auf die Idee kommen, folgende Korrektheitsanforderung aufzustellen: Die Datenbank soll *ständig* korrekt (integer oder konsistent) sein. Diese Anforderung ist jedoch nicht sinnvoll, denn im Verlauf mancher Veränderungen der Datenbank müssen zwangsläufig logisch inkonsistente Zwischenzustände auftreten. Das Standardbeispiel hierfür ist eine Umbuchung von X DM von Konto A auf Konto B, die wir uns vereinfacht in 4 Zugriffen auf die Datenbank vorstellen:

Zugriff 1: alten Stand von Konto A lesen, X DM abziehen
Zugriff 2: neuen Stand von Konto A schreiben
Zugriff 3: alten Stand von Konto B lesen, X DM addieren
Zugriff 4: neuen Stand von Konto B schreiben

In unserem Beispiel könnte das Konsistenzkriterium gelten, daß die Summe aller Kontostände 0 sein muß. Nach dem zweiten obigen Zugriff ist dieses Kriterium offenbar verletzt. Erst nach dem vierten Schritt ist die Konsistenz wiederhergestellt, die Verletzung war also nur *temporär*. Diese Beobachtung ist extrem wichtig und kann

dahingehend verallgemeinert werden, daß bei allen komplexen Konsistenzbedingungen temporäre Verletzungen *unvermeidlich* sind, da die Datenbank i.a. nicht durch eine einzige Datenmanipulationsoperation vom bisherigen Zustand in einen neuen konsistenten Zustand überführt werden kann. Letzteres ist nur durch *mehrere* Zugriffe möglich.

Die obige Anforderung muß daher dahingehend modifiziert werden, daß die Datenbank immer nur dann korrekt sein soll, wenn sie **ruht**, d.h. wenn gerade keine komplexen Änderungen mehr stattfinden.

Da man einer Aktion nicht ansieht, ob sie die letzte in einer zusammengehörigen Folge ist, müssen Anfang und Ende von Änderungen explizit gekennzeichnet werden. Andernfalls wäre nie entscheidbar, ob die Datenbank gerade ruht oder nicht.

1.3 Transaktionen

Das Konzept zur Lösung dieses und anderer Probleme sind Transaktionen. Eine **Transaktion** ist eine vom Benutzer definierte Folge von Aktionen[1], sozusagen eine "elementare Arbeitseinheit".

Transaktionen können nur von Applikationsprogrammen aus ausgenutzt werden (interaktive Schnittstellen zu Datenbanken sind hier selbst als Applikationen aufzufassen). Die Frage ist nun, wie eine Applikation, hier im Sinne eines laufenden Prozesses verstanden, eine Folge von Aktionen definieren kann, die eine Transaktion bilden soll. Im Detail hängt dies von der Gastsprache und dem Betriebssystem ab; wir schildern i.f. als Beispiel die Vorgehensweise bei JDBC.

1.3.1 Transaktionen in JDBC

Zunächst muß ein Applikationsprozeß, der auf einer (SQL-) Datenbank arbeiten will, eine Verbindung zum Datenbankserver aufbauen.

[1] Üblicherweise wird unter einer Transaktion sowohl eine Folge ausgeführter Aktionen verstanden als auch ein Programm bzw. Programmstück, welches eine einzige bzw. eine Menge solcher Folgen definiert. Das ist zwar terminologisch etwas unsauber, aber bequem. Um nicht zu sehr vom üblichen Sprachgebrauch abzuweichen, werden wir hier ebenso verfahren. Aus dem Kontext ergibt sich stets, ob ein Programm (oder Programmteil) oder eine Ausführung gemeint ist.

Details übergehen wir hier, die Verbindung wird letztlich durch ein Objekt des Typs `Connection` repräsentiert und durch die Operation `getConnection` initialisiert, z.B.

```
Connection verbindung = DriverManager.getConnection (...)
```

Ein explizites Kommando zum Beginnen einer Transaktion ist nicht vorhanden; nach Aufbau der Verbindung und nach der Beendigung einer Transaktion wird immer implizit eine neue Transaktion begonnen.

Das (erfolgreiche) Ende einer Transaktion wird dem DBMS durch die Operation `commit` bekanntgegeben, z.B. durch folgenden Aufruf:

```
verbindung.commit();
```

Nach Aufbau der Verbindung befinden wir uns allerdings zunächst in einem Arbeitsmodus, bei dem sozusagen implizit nach jeder Aktion ein `commit` ausgeführt wird. Mit anderen Worten wird *jedes einzelne* SQL-Kommando als Transaktion behandelt, d.h. es werden nicht wirklich Gruppen von Aktionen gebildet. Das automatische `commit` kann man abschalten mit

```
verbindung.setAutoCommit(false);
```

1.3.2 Transaktionseigenschaften

Bei der Ausführung einer Transaktion werden die folgenden wichtigen Eigenschaften garantiert:

1. **Konsistenzerhaltung**: Eine Transaktion, die in einem logisch konsistenten Zustand startet, hinterläßt die Datenbank am Ende wieder in einem logisch konsistenten Zustand. Verantwortlich hierfür ist in erster Linie der Benutzer, der die Transaktion programmiert oder Daten eingibt.
2. **(Fehler-) Atomarität**: eine Transaktion wird entweder *ganz oder gar nicht* wirksam. Wenn also eine Transaktion, gleichgültig aus welchen Gründen, fehlschlägt und abgebrochen wird, verändert sie die Datenbank nicht. Dies bedeutet, daß alle bisher von ihr verursachten Änderungen rückgängig gemacht werden. Verantwortlich für die Atomarität ist das DBMS.

3. **Dauerhaftigkeit**: Sobald dem Benutzer die erfolgreiche Ausführung einer Transaktion gemeldet wurde, müssen ihre Wirkungen erhalten bleiben und dürfen nicht infolge von Störungen oder Beschädigungen der Datenbank, die der Benutzer nicht zu verantworten hat und von denen er ggf. nichts erfährt, verloren gehen. Verantwortlich hierfür ist das DBMS.
4. **Serialisierbarkeit**: Bei parallel ausgeführten Transaktionen entspricht der Gesamteffekt mehrerer überlappend ausgeführter Transaktionen dem Effekt, der bei einer denkbaren seriellen Ausführung der Transaktionen erreicht worden wäre. Jede Transaktion hat also den Eindruck, daß sie *isoliert* auf der Datenbank arbeiten würde. Verantwortlich hierfür ist das DBMS.
5. **Endliche Ausführungszeit**: Die Ausführung einer Transaktion darf nicht durch Synchronisationsvorgänge im DBMS immer wieder hinausgeschoben werden. Der Benutzer muß in endlicher Zeit nach dem Start einer Transaktion eine Rückmeldung über die erfolgreiche Ausführung oder den Abbruch erhalten. Verantwortlich hierfür ist das DBMS.

   Trivialerweise setzen wir hier voraus, daß jede Transaktion nur aus endlich vielen Aktionen besteht und daß das Transaktionsprogramm keine Endlosschleifen enthält (wofür der Benutzer die Verantwortung trägt).

In vielen älteren Quellen wird statt der Serialisierbarkeit als eine Transaktionseigenschaft die Isolation gefordert. Unter **Isolation** versteht man, daß alle Effekte innerhalb einer Transaktion, also vor ihrer Beendigung, für parallele Transaktionen unsichtbar sind. Diese Anforderung ist naheliegend, denn, wie oben gezeigt, können logisch inkonsistente Zwischenzustände auftreten. Bei "normalen" Transaktionen (insb. solchen, die keine Datenwerte blind überschreiben, ohne sie vorher zu lesen) impliziert die Isolation die Serialisierbarkeit, die Umkehrung gilt nicht.

In der Literatur werden die Transaktionseigenschaften oft mit der Abkürzung *ACID* für *atomicity, consistency preservation, isolation, durability* zusammengefaßt.

Transaktionen sind das Schlüsselkonzept schlechthin in der Architektur moderner (und der meisten älteren) DBMS; es ist nahezu obligatorisch. Änderungen und Abfragen in Datenbanken werden daher i.d.R. durch Transaktionen mit den oben erwähnten Eigenschaften durchgeführt[2]. Aus der Sicht des Benutzers ist ein DBMS somit ein Transaktionsverarbeitungssystem.

Obwohl das Stichwort Parallelität bereits mehrfach gefallen ist, werden wir im Rest dieses Lehrmoduls annehmen, daß Transaktionen *sequentiell* ausgeführt werden. Mit parallel ausgeführten Transaktionen befassen sich eigene Lehrmodule.

1.4 Die Integritätsanforderung - 2. Ansatz

Kommen wir nun auf die oben gestellte Anforderung zurück, daß der Datenbank-Zustand zumindest dann logisch konsistent sein soll, wenn die Datenbank ruht. Durch die Einführung des Transaktionskonzepts und die Eigenschaft "Konsistenzerhaltung" ist diese Anforderung offenbar bereits erfüllt. Wir setzen natürlich voraus, daß die Datenbank nach ihrer Initialisierung logisch konsistent ist.

Das Integritätsproblem wird durch die Transaktionen nicht gelöst. Ein logisch konsistenter Zustand beschreibt irgendeine denkbare Realität, der integre nur die aktuell vorhandene.

Die in der Datenbank darzustellende Realität ändert sich im Laufe der Zeit; jedenfalls nehmen wir dies an. Die Integrität der Datenbank kann über die Zeit hinweg nur dann erreicht werden, wenn nach jeder Änderung der Realität rechtzeitig der Datenbank-Inhalt entsprechend korrigiert wird. Rechtzeitig bedeutet: vor erneuter Benutzung der korrekturbedürftigen Daten. Nach den obigen Festlegungen sind Korrekturen in Form von Transaktionen durchzuführen. Zur Sicherstellung der Integrität muß also für folgendes gesorgt werden:

[2]Die im Prinzip angenehmen Transaktionseigenschaften haben ihren Preis, nämlich eine verschlechterte Performance des Systems. In performancekritischen Systemen muß daher bisweilen auf Transaktionseigenschaften verzichtet werden. In SQL kann man durch "Isolationsstufen" schrittweise Transaktionseigenschaften zugunsten besserer Performance opfern (s. [SPV])

A) Korrigierende Transaktionen sind mit den richtigen Argumenten rechtzeitig zu starten.

B) Die Transaktionen müssen geeignet programmiert sein, um den Datenbank-Zustand richtig zu korrigieren, so daß die Änderung der Realität in der Datenbank nachvollzogen wird.

C) Die Transaktionen müssen korrekt ausgeführt werden (vgl. die obigen, vom DBMS zu garantierenden Ausführungseigenschaften).

Für A und B kann das DBMS wenig oder keine Verantwortung tragen: Ursache ist, daß es i.a. keine direkten "Wahrnehmungsorgane" für die Realität hat, die die Datenbank darstellen soll. Statt dessen nimmt es die Realität durch die "Brille" der Änderungen bzw. Transaktionen wahr, die von Benutzern oder automatischen Datenerfassungsgeräten veranlaßt werden. Das DBMS kann die Exaktheit von Angaben nicht prüfen, es kann auch die Rechtzeitigkeit von Änderungen nicht erzwingen. In den Bereichen A und B sind vorrangig die Benutzer bzw. die Systemumgebung verantwortlich. Das DBMS kann allenfalls gewisse grobe Fehler erkennen und Transaktionen abbrechen.

Der Bereich C steht hingegen voll in der Verantwortung des DBMS. Es muß dafür sorgen, daß der Zustand der Datenbank genau der ist, der sich gemäß den Transaktionen ergibt, deren Ausführung den Benutzern bestätigt worden ist. Diesen Zustand nennen wir **technisch korrekt**.

In diesem und den aufbauenden Lehrmodulen werden wir das Problem der Sicherstellung der Integrität auf das Problem der Sicherstellung der technischen Korrektheit reduzieren, denn es werden nur Maßnahmen behandelt, die *das DBMS* zur Sicherstellung der Integrität ergreifen kann.

2 Maßnahmen des DBMS zur Integritätssicherung

2.1 Klassifikation der Schutzmechanismen

Die Integrität bzw. Korrektheit der Datenbank wird durch viele verschiedene Ursachen gefährdet. Die Aufgabe an ein DBMS besteht also darin, durch geeignete Schutzmechanismen den Verlust der Integrität zu verhindern. Die Schutzmechanismen können in drei Kategorien eingeteilt werden:

1. **semantische Integritätsprüfungen**: Verhinderung von Änderungen, die zu einem inkorrekten Datenbank-Zustand führen; Ursache können z.B. versehentlich oder absichtlich falsch eingegebene Daten oder fehlerhafte Applikationsprogramme sein.
2. **Recovery**: Wiederherstellung der Datenbank nach einer Beschädigung infolge einer Störung; Ursachen können Fehler in der Hardware oder Software oder auch eine Fehlbedienung sein.
3. **Concurrency Control**: Verhinderung von Interferenzen zwischen parallelen Transaktionen oder Behebung unzulässiger Effekte eingetretener Interferenz.

Transaktionen bilden die konzeptionelle Grundlage bei allen oben erwähnten Arten von Schutzmaßnahmen, dies unterstreicht ihre Wichtigkeit.

Die einzelnen Arten von Maßnahmen decken im Prinzip unabhängige Problemstellungen ab, sie weisen aber, wie man bei näherer Analyse feststellt, signifikante Querbezüge untereinander auf. So wird z.B. für bestimmte Arten von semantischen Integritätsprüfungen auf Recovery-Mechanismen zurückgegriffen. Die folgenden Kurzdarstellungen der Einzelgebiete sollen u.a. diese Querbezüge verdeutlichen.

2.2 Semantische Integritätsprüfungen

Wir hatten schon oben erwähnt, daß im Prinzip der Programmierer von Transaktionen, die späteren Nutzer des Informationssystems und

ggf. weitere Personen dafür verantwortlich sind, daß Änderungen rechtzeitig durchgeführt, richtige Daten eingegeben und die richtigen Aktionen ausgeführt werden. Fehler passieren natürlich dennoch, und es stellt sich die Frage, inwieweit das DBMS solche Fehler erkennen und abwehren kann.

Das DBMS weiß von sich aus natürlich nicht, wann ein Fehler vorliegt, d.h. unser vorstehender Satz, daß "das DBMS Fehler erkennt bzw. abwehrt", kann nur so verstanden werden, daß einem Programmierer bzw. DB-Administrator zusätzliche Funktionen angeboten werden, durch die Fehler beschrieben bzw. Reaktionen auf Fehler programmiert werden können. In vielen Fällen kann man solche Angaben als eine Erweiterung des Datenbankschemas oder als eine andere Art von Transaktionsprogrammierung ansehen.

Die einzelnen Datenbankmodelle (bzw. DBMS-Produkte) unterscheiden sich erheblich hinsichtlich der Funktionen, mit denen die Integrität der Daten überwacht werden kann. Diese Vielfalt wollen wir hier nicht auffächern, sondern direkt auf einen im Kontext dieses Lehrmoduls relevanten Aspekt zusteuern: In vielen Fällen können die Integritätsprüfungen erst durchgeführt werden, *nachdem* eine Transaktion schon Inhalte der Datenbank verändert hat. Sofern die Prüfung negativ ausgeht, muß die Transaktion abgebrochen werden. Hierzu können die ohnehin vorhandenen Recovery-Mechanismen, speziell die Funktion zum Zurücksetzen einer Transaktion (s.u.) verwendet werden, d.h. hier besteht ein Querbezug zwischen Integritätsprüfungen und Recovery.

Ein offener Punkt ist hier, von wo aus das Zurücksetzen einer Transaktion ausgelöst wird, infrage kommen das DBMS und die Applikation. Der erste Fall erfordert z.B. ein deskriptives Verfahren, durch das unzulässige, die Zurücksetzung auslösende Datenbankinhalte beschrieben werden können. Das Rücksetzen der Transaktion wäre dann ein "Nebeneffekt" eines Versuchs, die Datenbank in einen unzulässigen Zustand zu überführen. Dieser Ansatz ist aber wenig praxisgerecht, weil das Rücksetzen der laufenden Transaktion keine Nebensache ist, sondern z.B. die folgenden Ausgaben bzw. sonstigen Arbeitsschritte erheblich beeinflussen wird. Daher ist es sinnvoller, das Zurücksetzen

einer Transaktion durch das Transaktionsprogramm selbst steuern zu lassen. Um dies technisch zu ermöglichen, muß es eine API-Funktion geben, durch die die laufende Transaktion zurückgesetzt werden kann; in JDBC ist dies die Operation `rollback`.

Obwohl die Integritätsprüfungen nicht immer perfekt sind, bezeichnet man Transaktionen als "*Einheit logisch konsistenter Zustandsübergänge*".

2.3 Recovery

Maßnahmen zur Wiederherstellung (von Teilen) einer Datenbank nach einer Beschädigung werden unter der Bezeichnung Recovery zusammengefaßt. Die wichtigsten (allerdings nicht alle) hierbei angewandten Techniken beruhen im Prinzip darauf, zunächst in geschickter Weise redundante Hilfsdaten zu erzeugen und im Falle einer Beschädigung der Datenbank entweder

a. beim sogenannten **Rückwärts-Recovery** die beschädigten bzw. unsicheren Teile der Datenbank in einen früheren Zustand zurückzuversetzen, also sozusagen die Datenbank "zu reparieren", oder

b. beim sogenannten **Vorwärts-Recovery** die Datenbank völlig neu aufzubauen.

Die Alternative a wird dann gewählt werden, wenn nur kleine Teile der Datenbank beschädigt sind, insbesondere beim Zurücksetzen von Transaktionen; Alternative b hingegen, wenn sehr große Teile der Datenbank verfälscht worden sind oder wenn infolge eines Hardware-Fehlers des Speichermediums gar nicht bekannt ist, welche Teile (physisch) defekt sind. In der Praxis sollten beide Alternativen verfügbar sein.

Beide Arten benötigen spezielle Vorbereitungen und Hilfsdaten, sogenannte **Recovery-Daten**. Die meisten Systeme verfahren stark vereinfacht wie folgt:

1. Zu bestimmten Zeitpunkten (z.B. einmal wöchentlich) wird eine **Backup-Kopie** der Datenbank auf einem anderen Medium angefertigt, in der Regel aus Kostengründen auf Bandkassetten.

2. Bei jeder Änderung eines Datenbank-Objekts werden die alten und neuen Werte in einem sogenannten **Log** (Logdatei) festgehalten. Beim Einfügen eines neuen Objekts existiert natürlich kein alter Wert, bei der Löschung kein neuer.

Tritt nun ein Fehler ein, so wird bei den beiden oben genannten Alternativen wie folgt verfahren:

a. Alle "verdächtigen" Änderungen werden rückgängig gemacht (Rollback von Transaktionen); die alten Werte werden dem Log entnommen.
b. Die letzte Kopie der Datenbank wird geladen und alle seit dem Zeitpunkt ihrer Anfertigung durchgeführten Änderungen werden mit Hilfe des Logs nachgeholt (Neustart mit "redo" von Transaktionen).

Die Recovery-Algorithmen sehen in der Praxis jedoch wesentlich komplizierter aus, da zur Steigerung der Effizienz bzw. Minimierung der Kosten eine Reihe von Detailänderungen vorgenommen werden; ferner spielen oft die Strukturen des zugrundeliegenden Betriebssystems eine wichtige Rolle in diesen Algorithmen, da in gewissen Fällen auch im Betriebssystem Reparaturen bzw. Neustart erforderlich sind. In allen Fällen sind Transaktionen ein wichtiges Konzept. Transaktionen werden auch als "*Einheit der Wiederherstellung*" ("unit of recovery") bezeichnet.

2.4 Concurrency Control

Es ist in wesentliches Ziel von Datenbanken, vielen Benutzern Zugriff zu gemeinsamen Daten zu ermöglichen. Bei vielen Anwendungen muß dies gleichzeitig möglich sein. Bei der parallelen Ausführung von Transaktionen, die auf gemeinsame Daten zugreifen, können einige typische unzulässige Interferenz-Effekte auftreten, sogenannte **Parallelitätsanomalien**, z.B. Verluste von Änderungen oder inkonsistente Datenbank-Zustände. Durch diese geht die Korrektheit der Datenbank verloren, obwohl die Transaktionen, wenn sie alleine ausgeführt würden, die Korrektheit der Datenbank erhielten.

Die durch Parallelität verursachten Probleme können nach sehr unterschiedlichen Verfahren behandelt werden; die wichtigsten sind:

1. **Sperrverfahren**: Interferenzen werden durch geeignetes Blockieren von Zugriffen zu einzelnen Datenbank-Objekten vermieden. Hierdurch wird i.w. der wechselseitige Ausschluß realisiert, wie er in ähnlicher Form auch in anderen parallel arbeitenden Systemen bekannt ist, z.B. in Betriebssystemen. Die Transaktionen spielen hierbei die Rolle der parallelen Prozesse, die einzelnen Datenbank-Objekte sind die gemeinsam benutzten Ressourcen.

   Je nach der Technik des Sperrens können Deadlocks eintreten. Transaktionen sind somit auch "*Einheiten des Sperrens*".

2. **Zeitstempel-Verfahren**: Jede Transaktion erhält eine Zeitmarke (timestamp). Ziel ist, die Transaktionen zwar verzahnt, aber so auszuführen, als ob sie (aus der Sicht ihres Effekts) in der Reihenfolge ihrer Zeitmarken sequentiell ausgeführt worden wären. Stellt man fest, daß eine Transaktion eine von ihrer Zeitmarke abweichende Wirkung hat, wird sie abgebrochen und mit einer anderen Zeitmarke neu gestartet.

   Da kein Warten auftritt, sind Deadlocks ausgeschlossen. Es besteht allerdings die Gefahr des zyklischen Neustarts.

3. **Optimistische Verfahren**: Bei diesen geht man davon aus, daß Interferenzen nur sehr selten eintreten, was bei vielen Anwendungen tatsächlich der Fall ist. Vorkehrungen gegen Interferenzen werden überhaupt nicht getroffen. Statt dessen wird durch Überwachung der Datenflüsse festgestellt, ob eine unzulässige Interferenz eingetreten ist. Falls ja, werden die betroffenen Transaktionen abgebrochen und neu gestartet.

   Deadlocks können nicht auftreten, aber auch hier droht zyklischer Neustart.

Alle drei Grundtypen der Verfahren haben viele Varianten, alle wiederum mit speziellen Vor- und Nachteilen; ferner wurden Mischformen angeregt.

Praktische Bedeutung haben vor allem Sperrverfahren erlangt: in den heute vorhandenen DBMS werden fast ausschließlich Sperrverfahren angewandt.

Zeitstempel-Verfahren wurden speziell für verteilte Datenbanken entwickelt, sind aber auch in zentralen Datenbanken anwendbar. Auch bei verteilten Datenbanken sind die meisten Verfahren Sperrverfahren; die Brauchbarkeit von optimistischen Verfahren ist noch unklar.

Zeitstempel- und optimistische Verfahren wurden zwar aus verschiedenen Motivationen heraus entwickelt, weisen aber sehr viele Gemeinsamkeiten auf. Insbesondere arbeiten beide in der Grundform nach dem Prinzip, die Korrektheit von Abläufen durch Neustart von Transaktionen zu gewährleisten. Deshalb werden sie gemeinsam als "validierende Verfahren" bezeichnet.

Die oben aufgeführten Transaktionseigenschaften sind nur sinnvoll und technisch realisierbar, wenn die Transaktionen nur wenige Objekte verändern, Laufzeiten im Sekundenbereich haben und wiederholbar, also nicht interaktiv sind. Wenn diese Voraussetzungen nicht erfüllt sind, sind "nichtkonventionelle" Transaktionskonzepte erforderlich, oft gleichzeitig mit nichtkonventionellen DBMS. Einen Überblick über eine Vielzahl von nichtkonventionellen Transaktionskonzepten gibt [El92].

In verteilten Datenbanken, auf die wir hier nicht näher eingehen können, kommen zwei weitere Problemkomplexe hinzu:

1. bei replizierten Datenbanken die Erhaltung der Konsistenz der Replikate, insb. also die Propagation von Änderungen
2. die verteilte Ausführung von Transaktionen: Eine Transaktion kann Daten auf mehreren Rechnern ändern; Netzwerkausfälle und Systemabstürze einzelner Rechner bedrohen hier zusätzlich die Fehleratomarität.

Literatur

[El92] Elmagarmid, A. (ed.): Database transaction models for advanced applications; Morgan Kaufmann; 1992

[SPV] Kelter, U.: Lehrmodul "Sperrverfahren"; 2003

Glossar

Aktion: im Kontext von Transaktionen: Ausführung einer der Operationen des Datenbankmodells unter Verwendung zulässiger bzw. korrekter Parameter

Concurrency Control: Verhinderung von Interferenzen zwischen parallelen Transaktionen oder Behebung unzulässiger Effekte eingetretener Interferenz

erreichbar: ein Inhalt bzw. Zustand einer Datenbank ist erreichbar, wenn nach Initialisierung der Datenbank durch eine endliche Folge von Aktionen (mit zulässigen bzw. syntaktisch korrekten Parametern) konstruiert werden kann

Fehleratomarität: Eigenschaft einer Transaktion, daß die Folge von Aktionen ganz oder gar nicht ausgeführt wird

Integrität: Korrektheit des Datenbankinhalts in einem umfassenden Sinne

Isolation: Eigenschaft einer Transaktionsausführung, demzufolge alle Effekte innerhalb einer Transaktion, also vor ihrer Beendigung, für parallele Transaktionen unsichtbar sind

Konsistenz: Synonym zu logische Konsistenz

logisch konsistent: ein erreichbarer Zustand einer Datenbank ist logisch konsistent, wenn eine Realität möglich ist, die durch ihn korrekt wiedergegeben wird

physisch konsistent: ein interner Zustand einer Datenbank ist physisch konsistent, wenn sich die internen Speicherungsstrukturen der Datenbank in einem ordnungsgemäßen Zustand befinden; auf einem physisch inkonsistenten Datenbankzustand kann der Laufzeitkern i.a. nicht mehr korrekt arbeiten

optimistische Concurrency-Control-Verfahren: validierende Verfahren, die von der Annahme, daß Konflikte sehr selten sind, ausgehen und die nur beim Commit einen Validationstest durchführen

Recovery: Wiederherstellung der Datenbank nach einer Beschädigung infolge einer Störung

Rollback: Aufheben der bisherigen Wirkungen einer Transaktion

Serialisierbarkeit: Eigenschaft parallel überlappend ausgeführter Transaktionen, daß deren Effekt der gleiche ist, der bei irgendeiner denkbaren seriellen Ausführung der Transaktionen erreicht worden wäre

Sperrverfahren: Concurrency-Control-Verfahren, in dem Sperren eingesetzt werden, um nichtserialisierbare Verzahnungen von Transaktionen zu verhindern

technisch korrekt: Zustand der Datenbank, der sich gemäß den Transaktionen ergibt, deren Ausführung den Benutzern bestätigt worden ist

Transaktion: Folge von Datenbankzugriffen (Aktionen), die einen konsistenten Datenbankzustand in einen neuen konsistenten Datenbankzustand überführt; Transaktionsausführungen haben folgende Eigenschaften: Fehler-Atomarität, Dauerhaftigkeit, Serialisierbarkeit, endliche Ausführungszeit

validierendes Concurrency-Control–Verfahren: Verfahren, in denen mit Hilfe eines Validationstests bestimmt wird, ob eine Verzahnung von Transaktionen als zulässig angesehen werden kann; im negativen Fall wird eine der involvierten Transaktionen zurückgesetzt und neu gestartet

Zeitstempel-Verfahren: validierendes Concurrency-Control-Verfahren, das Zeitstempel an Objekten und Transaktionen für die Validationstests benutzt und bei jedem Zugriff einen Validationstest durchführt

Index