Wiederholung (5)

Aufgaben des Logging:

Jede Änderungsoperation auf der DB im Normalbetrieb wird protokolliert

- *REDO*: Informationen zum Nachvollziehen der Änderungen erfolgreicher TAs
- *UNDO*: Informationen zum Zurücknehmen der Änderungen unvollständiger TAs

Klassifikation von Logging-Verfahren:

- *physisch*: Protokoll auf Ebene der Seiten, Datensätze, Indexeinträge
  - *Zustands-Logging*: alter Zustand *Before-Image (BFIM)*, neuer Zustand *After-Image (AFIM)*:
  - *Übergangs-Logging*: Speicherung der Zustandsdifferenz
- *logisch*: Speicherung der Änderungsoperationen mit ihren Parametern (z.B. INSERT mit Attributwerten) $\rightarrow$ kurze Logeinträge
- *physiologisch*: Kombination von physischem und logischem Logging

Struktur der Log-Einträge für Änderungen:

$$(LSN, TA\text{-}Id, Page\text{-}Id, REDO, UNDO, PrevLSN)$$

- LSN *Log Sequence Number*: eindeutige Kennung des Log-Eintrags in chronologischer Reihenfolge
- $TA\text{-}Id$: eindeutige Kennung der TA, die die Änderung durchgeführt hat
- $Page\text{-}Id$: Kennung der Seite auf der die Änderungsoperation vollzogen wurde (ein Eintrag pro geänderter Seite)
- $REDO$: gibt an, wie die Änderung nachvollzogen werden kann
- $UNDO$: beschreibt, wie die Änderung rückgängig gemacht werden kann
- $PrevLSN$: Zeiger auf vorhergehenden Log-Eintrag der jeweiligen TA (Effizienzgründe)

Log-Sätze:

Ein Log-Satz wird für jede der folgenden Aktionen geschrieben:

- *UPDATE*: nach Modifikation einer Seite wird $pageLSN$ auf LSN des UPDATE-Log-Record gesetzt
- *COMMIT*: bei Entscheidung für COMMIT wird ein COMMIT-Record mit der $TA-Id$ geschrieben und auf Platte gezwungen (erst dann ist COMMIT der TA wirklich bestätigt)
- *ABORT*: wird bei ABORT einer TA geschrieben, UNDO wird angestossen
- *END*: nach Beendigung zusätzlicher Aktionen bei COMMIT oder ABORT
- *Compensation Log Records (CLR)*: beim Zurücksetzen einer TA (UNDO) müssen Aktionen rückgängig gemacht werden; das Logging hiervon geschieht durch das Schreiben eines *CLR*.

Weitere Datenstrukturen beim Recovery:

- *TA-Tabelle*:
  - ein Eintrag pro aktive TA
  - enthält $TA\text{-}Id$, $Status$ (running/ commited/ aborted) und $lastLSN$ (letzte vergebene LSN)
- $DirtyPage$-*Tabelle*: Nachführen aller bereits abgeschlossenen TAs,die *noch nicht* in die DB eingebracht wurden
  - ein Eintrag pro schmutzige Seite im Puffer
  - enthält $recLSN$ (LSN des Log-Satzes, durch den *erstmals* die Seite schmutzig gemacht wurde)

Sicherstellung der Idempotenz von Undo und Redo:

Redo- und Undo-Phasen müssen idempotent sein, d.h. sie müssen auch bei mehrfacher Ausführung immer wieder das selbe Ergebnis liefern.

Man nennt eine einstellige Funktion $f : X \to X$ *idempotent*, falls gilt:

$$f(f(x)) = f(x)$$

Die Idempotenz gewährleistet das fehlerfreie Verhalten bei Systemfehlern während der Recovery.

- **Idempotenz von Redo**

  Beispiel: Franz zahlt 100 Euro auf sein Konto ein. Die entsprechende Transaktion committet, wird aber vor dem Systemfehler noch nicht festgeschrieben (No-Force). Bei der Recovery wird in der Redo-Phase die Einzahlung von Franz durch den entsprechenden Log-Eintrag wiederholt und festgeschrieben. Direkt danach (vor Abschluss der Recovery) gibt es einen erneuten Systemausfall. Die Recovery beginnt von neuem. Jetzt muss dafür gesorgt werden, dass der Log-Eintrag von Franz nicht erneut wiederholt wird, sonst erhält Franz für den zweiten Crash (und jeden weiteren Crash) 100 Euro *freu*.

  $\Rightarrow$ Idempotenz des Redo durch Check der $pageLSN$ (auf Platte)

- **Idempotenz von Undo**

  Beispiel: Hans zahlt 100 Euro an Franz für eine große Kiste Gummibärli. Das Geld wird von Hans' Konto abgebucht und der neue Kontostand festgeschrieben (Steal). Aber bevor das Geld auf dem Konto von Franz gutgeschrieben wird (also bevor die TA committet), gibt es einen Systemausfall. In der Recovery wird festgestellt, dass es sich hier um eine Verlierer-TA handelt. Bei selektivem Redo ist in der Redo-Phase nichts zu tun. In der Undo-Phase wird der entsprechende Log-Eintrag von Hans gelesen, und es werden die 100 Euro zurück auf Hans' Konto gebucht. Direkt danach kommt es erneut zu einem Systemfehler. Jetzt muss dafür gesorgt werden, dass bei der erneuten Recovery in der Undo-Phase nicht erneut 100 Euro auf das Konto von Franz gebucht werden.

  $\Rightarrow$ Idempotenz des Undo wird gewährleistet durch die CLRs. Ein CLR bekommt genau wie ein "normaler" Log-Eintrag eine eindeutige LSN zugeteilt. Die Redo Information eines CLR entspricht der während der Undo-Phase des Wiederanlaufs ausgeführten Undo Operation. Bei einem erneuten Wiederanlauf wird diese Operation innerhalb der Redo-Phase ausgeführt. CLRs benötigen keine Undo-Information, da sie bei nachfolgenden Undo-Phasen übersprungen werden. Um das Überspringen zu gewährleisten, enthalten CLRs ein Feld *UndoNextLSN*. Dieses Feld enthält die LSN der zu dieser TA gehörenden Änderungsoperation, die der kompensierten Operation vorausging. Damit wird in der Undo Phase direkt an die Stelle zurückgesprungen, an der die letzte Undo Phase abgebrochen wurde.

- **Vermeiden von Verhungern**

  Zusätzlich stellen die CLRs sicher, dass die Recovery bei häufigen Systemfehlern Fortschritte macht. Beispiel ohne CLRs: Der letzte Sicherungspunkt vor dem Systemfehler ist lange her. Das Logfile ist entsprechend groß. Nehmen wir an, in der Undo-Phase wird nur die *PageLSN* überprüft, und kein CLR geschrieben. Nehmen wir außerdem vollständiges Redo an: Die Redo-Phase läuft einwandfrei ab, die *pageLSNs* werden auf die neuesten Werte gesetzt. In der Undo-Phase werden die *pageLSNs* der Verlierer wieder verringert (d.h. ältere Versionen werden hergestellt). Nehmen wir an, am Ende der Redo-Phase gibt es einen erneuten Systemausfall. Im nächsten Anlauf muss in der Redo-Phase erneut jede Aktion der Verlierer-TAs wiederholt werden. Problem: Möglicherweise kein Fortschritt der Recovery bei häufigen Systemfehler - die Recovery terminiert nicht.

  Die CLRs stellen sicher, dass solch ein endloses Hin-und-Her zwischen alter und neuer Version nicht passieren kann. Nach einmaligem Durchlaufen eines Log-Eintrags in der Undo-Phase wird dieser in der Undo-Phase nie wieder betrachtet. Stattdessen wird bei späteren Anläufen ein entsprechender CLR in der Redo-Phase betrachtet. Dieser hat eine größere LSN als die ursprünglichen Log-Einträge (die CLRs erhalten, wie alle Log-Einträge, eine aufsteigende LSN) und wird deshalb von späteren Redo-Läufen nicht wieder rückgangig gemacht.

  Bei häufigen Systemfehlern kann das DBMS nun entscheiden, Redo-Aktionen sofort auf Platte auszuschreiben, um einen Fortschritt auch bei kurzen Anläufen sicherzustellen.

Phasen der Crash-Recovery:

(a) ***Analysephase***: Bestimmen der Gewinner- und Verlierer-TAs seit dem Checkpoint

- Bestimme den Punkt im Log, an dem die REDO-Phase beginnen muss
- Bestimme die Menge der Seiten im Puffer, die zum Crash-Zeitpunkt schmutzig waren
- Identifiziere die Verlierer-TAs, die zum Crash-Zeitpunkt aktiv waren und rückgängig gemacht werden müssen (UNDO nötig)
- *Rekonstruiere Zustand zum Checkpoint-Zeitpunkt*
- *Lese sequentiell vorwärts vom Checkpoint*

(b) ***REDO-Phase***: Ablauf wird wiederholt, um den Zustand zum Zeitpunkt des Crash zu rekonstruieren (Generell gilt: es müssen nur Aktionen wiederholt werden, die noch nicht in der Datenbank stehen)

- Vollständiges REDO: Wiederholung *aller* Updates (auch der abgebrochenen TAs!), Wiederholung der CLRs
- Selektives REDO: Wiederholung der Updates aller Gewinner-TAs, Wiederholung der CLRs
- Vorwärtslesen vom Log-Record, dessen LSN gleich der kleinsten $recLSN$ in $DirtyPage$-Tabelle. Für jede LSN eines CLR oder $Update$-Log-Record, mache ein REDO der Aktion unter folgenden Voraussetzungen:
  - betroffene Seite ist nicht in $DirtyPage$-Tabelle, oder
  - betroffene Seite ist in $DirtyPage$-Tabelle, aber hat $recLSN > LSN$, oder
  - $pageLSN$ (in DB) $< LSN$
- REDO einer Aktion:
  - wiederholte Ausführung der geloggten Aktion
  - setze $pageLSN$ auf LSN (kein zusätzliches Logging)

(c) ***UNDO-Phase***: Zurücksetzen der Verlierer-TAs in umgekehrter Reihenfolge

- Vollständiges REDO: UNDO der Updates aller zum Fehlerzeitpunkt noch laufenden TAs. LSN-Prüfung ist nicht nötig, da beim REDO auch Wiederholungen der Aktionen von Verlierer-TAs durchgeführt wurden, die jetzt wieder zurückgenommen werden müssen.
- Selektives REDO: UNDO der Updates *aller* Verlierer-TAs. UNDO für jeden Log-Eintrag einer Verlierer-TA. UNDO nur durchführen, falls ***LSN*** $\leq$ ***PageLSN***, also wenn bereits in die DB eingebrachte Änderungen von Verlierer-TAs wieder zurückgenommen werden müssen.
- für jede ausgeführte UNDO-Operation wird ein CLR angelegt, der genau wie normale Log-Sätze eine eindeutige LSN zugeteilt bekommt (Wichtig für Fehlertoleranz der Recovery); der CLR enthält die folgenden Informationen:
  - LSN
  - $TA\text{-}Id$
  - $Page\text{-}IdN$
  - $REDO$: entspricht der während der UNDO-Phase ausgeführten UNDO-Operation
  - $prevLSN$
  - $UndoNxtLSN$: gewährleistet, dass CLR während nachfolgenden UNDO-Operationen übersprungen wird