GOETHE UNIVERSITÄT FRANKFURT AM MAIN

www.uni-frankfurt.de

Vorlesung Datenbanksysteme 2

Übung – Recovery Manager Undo Redo Algorithmen (Grundlagen)

Aufgabe

Warum sind die Recovery Manager Algorithmen notwendig?

Erläutern Sie die Vor- und Nachteile der folgenden Kombinationen von Recovery Manager Algorithmen.

a) Undo / Redo

b) Undo / No Redo

c) No Undo / Redo

d) No Undo / No Redo

Motivation

Warum sind die ***Recovery Manager (RM)*** Algorithmen notwendig?

- ***Transaktions-Fehler*** - tritt auf, wenn eine Transaktion abbricht.

- ***Medien-Fehler -*** tritt auf, wenn Teile des Festspeichers zerstört werden.(Wenn beispielsweise Sektoren einer Festplatte beschädigt werden.)

- ***System-Fehler*** - bezieht sich auf den Verlust von Inhalten des flüchtigen Speichers (RAM). (Beispielsweise durch einen Stromausfall, oder wenn das Betriebssystem einen Ausfall hat.)

Allgemein

- **Falls** der CM Slot c flushed, **und** die Transaktion T_i wegen eines Systemfehlers abbricht (vor dem Commit), **dann** ist ein **Undo** notwendig.

- **Falls** die Transaktion T_i committed hat **und** ein Systemfehler auftritt **bevor** der CM den Slot c geflushed hat, **dann** ist ein **Redo** notwendig.

- No Undo: Änderungen können erst **nach** commit geflushed werden.

- No Redo: **Vor** Commit müssen Änderungen geflushed sein.

RM Algorithmen Tabelle - Vorschlag

N		Undo/Redo	Undo/ No Redo	No Undo/ Redo	No Undo/ No Redo
1	Maximiert Effizienz im Normalbetrieb	Ja	Nein	Nein	Nein
2	Einfache Cache Verwaltung (bezogen auf flush() und fetch())	Ja	Nein	Nein	Nein
3	Kleinen Arbeitspeicher (RAM) möglich	Ja	Ja	Nein	Nein
4	CM darf nicht beliebig flushen.	Nein	Ja	Ja	Ja
5	Änderungen können gesammelt und auf einmal geschrieben werden (random vs. sequentielle Zugriffszeiten)	Ja	Nein	Ja	Nein
6	Einfache Logverwaltung	Nein	Nein	Nein	Ja
7	Aufwändige Verarbeitung bei Restart nach Fehler/Störungen	Undo und Redo nötig	Kein Redo nötig	Kein Undo nötig	Optimiert (kein Undo und Redo nötig)

Undo / Redo

Vorteile	Nachteile
Maximiert Effizienz im Normalbetrieb	Undo/Redo Logverwaltung ist aufwendig
Einfache Cache Verwaltung	Aufwändige Verarbeitung bei Restart nach Fehler/Störungen Restart: Undo und Redo nötig
Kleinen Arbeitspeicher (RAM) möglich	
Änderungen können gesammelt und auf einmal geschrieben werden (random vs. sequentielle Zugriffszeiten)	

Undo / No Redo

No Redo: Vor Commit müssen Änderungen geflushed sein.

Vorteile	Nachteile
Restart: Kein Redo nötig	RM muss CM zum Flush „zwingen“ (force) können → Commit kann eine schlechtere Performance haben
Kleinen Arbeitspeicher (RAM) möglich	

No Undo / Redo

No Undo: Änderungen können erst nach commit geflushed werden.

Vorteile	Nachteile
Änderungen können gesammelt und auf einmal geschrieben werden (random vs. sequentielle Zugriffszeiten)	CM darf nicht beliebig flushen. Lange Transaktionen blockieren Cache (dürfen erst bei commit ausgeschrieben werden) Extrem Fall: Cache zu klein, um alle „dirty“ Werte aufzunehmen – Swap Speicher nötig
Restart: Kein Undo nötig	

No Undo / No Redo

No Undo: Änderungen können erst **nach** commit geflushed werden.
No Redo: **Vor** Commit müssen Änderungen geflushed sein.
Praktisch nur mit *shadowing* oder anderen Verfahren umsetzbar!

Vorteile	Nachteile
Optimiert für Fehler-Fall (kein Undo und Redo nötig)	CM darf nicht beliebig flushen – genau erst bei Commit! Lange Transaktionen blockieren Cache (dürfen erst bei commit ausgeschrieben werden) Extrem Fall: Cache zu klein, um alle „dirty“ Werte aufzunehmen – Swap Speicher nötig