Recovery

Um Durability zu gewährleisten, müssen Datenbanken so arbeiten, dass committete Änderungen auch nach einem Crash noch zuverlässig Bestand haben, andere jedoch nicht. Dies ließe sich mit der Policy

- **non steal** (uncommitete Änderungen dürfen nicht auf die Festplatte geschrieben werden) und
- **force** (commitete Änderungen müssen sofort auf die Festplatte geschrieben werden)

erreichen. Dies würde allerdings die Parallelisierbarkeit viel zu stark einschränken, so dass reale Datenbanken fast ausschließlich steal und non force verwenden. Stattdessen verwenden sie ein sogenanntes Write-Ahead-Log (WAL).

- non force macht redo notwendig
- steal macht undo notwendig

Write Ahead Log

- Speichert jede Änderung
- muss bei jedem Commit auf die Festplatte geschrieben werden
- muss bei jeder Verdrängung von Seiten auf die Festplatte geschrieben werden
- In den Seiten auf der Festplatte wird vermerkt, von welcher LSN ihr Stand ist
- Attribute
  - LSN: Fortlaufende Nummerierung der Logeinträge
  - TransaktionsID
  - PageID
  - Redo
  - Undo
  - Prev-LSN: die vorherige LSN der gleichen Transaktion

Beispiel:

LSN	TA-ID	Page-ID	Redo	Undo	Prev-LSN
#1	T_1	BOT	-	-	-
#2	T_1	P_{CD}	$D-=10$	$D+=10$	#1
#3	T_2	BOT	-	-	-
#4	T_2	P_B	$B+=2$	$B-=2$	#3
#5	T_1	commit	-	-	#2
#6	T_3	BOT	-	-	-
#7	T_2	P_{CD}	$C+=17$	$C-=17$	#4
#8	T_3	P_A	$A-=15$	$A+=15$	#6
#9	T_3	commit	-	-	#8
#7	T_2	P_{CD}	$D+=3$	$D-=3$	#7

Recovery-Vorgang

- Redo **alle** Logbucheinträge, die noch nicht in der Datenbank stehen
- Undo alle Logbucheinträge von Loser(uncommitteten)-Transaktionen
  - Schreibe hierbei für jeden Eintrag einen CLR (Compensation Log Record) ins Logbuch
    - Ohne Undo-Eintrag
    - Mit Undo-Next-LSN
    - Mit <LSN>

Sollte es währenddessen zu einem erneuten Absturz kommen, sind diese CLR schon Teil der Redo-Phase und die Undo-Phase kann für jede Transaktion beim `Undo-Next-LSN` des letzten CLR fortgesetzt werden.

Checkpoints

brauchen wir, damit Redo und Undo nicht beliebig weit in die Vergangenheit zurück gemacht werden müssen

Arten:

- Transaktionskonsistente Sicherungspunkte
  - Alle Transaktionen zu Ende laufen lassen
  - keine neuen beginnen
  - Vorteil: Logbuch von davor kann weg
  - Nachteil: Datenbank lange komplett geblockt
- Aktionskonsistente Sicherungspunkte
  - wartet, bis laufende Operationen abgeschlossen sind

    - speichert aktive Transaktionen und minLSN
        - nur für diese muss im WAL zurückgegangen werden
- Unscharfe Sicherungspunkte
    - asynchron, mehr Aufwand
    - speichert dirty pages und deren LSN, aktive Transaktionen und minLSN
    - keine Unterbrechung des Datenbankbetriebs

Revision #5
Created 24 July 2024 15:57:05 by Kuchenmampfer
Updated 25 July 2024 10:20:10 by Kuchenmampfer