Kapitel 14

Recovery

Aufgabe der Recovery-Komponente des Datenbanksystems ist es, nach einem Fehler den jüngsten konsistenten Datenbankzustand wiederherzustellen.

14.1 Fehlerklassen

Wir unterscheiden drei Fehlerklassen:

1. lokaler Fehler in einer noch nicht festgeschriebenen Transaktion,
2. Fehler mit Hauptspeicherverlust,
3. Fehler mit Hintergrundspeicherverlust.

14.1.1 Lokaler Fehler einer Transaktion

Typische Fehler in dieser Fehlerklasse sind

- Fehler im Anwendungsprogramm,
- expliziter Abbruch (**abort**) der Transaktion durch den Benutzer,
- systemgesteuerter Abbruch einer Transaktion, um beispielsweise eine Verklemmung (Deadlock) zu beheben.

Diese Fehler werden behoben, indem alle Änderungen an der Datenbasis, die von dieser noch aktiven Transaktion verursacht wurden, rückgängig gemacht werden (*lokales Undo*). Dieser Vorgang tritt recht häufig auf und sollte in wenigen Millisekunden abgewickelt sein.

14.1.2 Fehler mit Hauptspeicherverlust

Ein Datenbankverwaltungssystem manipuliert Daten innerhalb eines *Datenbankpuffers*, dessen Seiten zuvor aus dem Hintergrundspeicher *eingelagert* worden sind und nach gewisser Zeit (durch Ver-

drängung) wieder *ausgelagert* werden müssen. Dies bedeutet, daß die im Puffer durchgeführtenÄnderungen erst mit dem Zurückschreiben in die materialisierte Datenbasis permanent werden. Abbildung 14.1 zeigt eine Seite P_A, in die das von A nach A' geänderte Item bereits zurückgeschrieben wurde, während die Seite P_C noch das alte, jetzt nicht mehr aktuelle Datum C enthält.

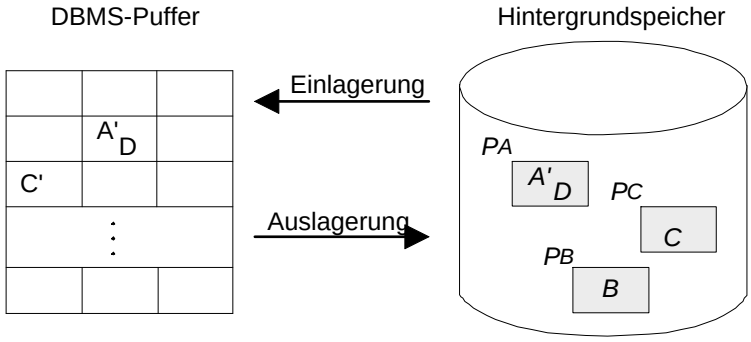

Abbildung 14.1: Schematische Darstellung der zweistufigen Speicherhierarchie

Bei einem Verlust des Hauptspeicherinhalts verlangt das Transaktionsparadigma, daß

- alle durch nicht abgeschlossene Transaktionen schon in die materialisierte Datenbasis eingebrachten Änderungen rückgängig gemacht werden (*globales undo*) und
- alle noch nicht in die materialisierte Datenbasis eingebrachten Änderungen durch abgeschlossene Transaktionen nachvollzogen werden (*globales redo*).

Fehler dieser Art treten im Intervall von Tagen auf und sollten mit Hilfe einer Log-Datei in wenigen Minuten behoben sein.

14.1.3 Fehler mit Hintergrundspeicherverlust

Fehler mit Hintergrundspeicherverlust treten z.B in folgenden Situationen auf:

- *head crash*, der die Platte mit der materialisierten Datenbank zerstört,
- Feuer/Erdbeben, wodurch die Platte zerstört wird,
- Fehler im Systemprogramm (z. B. im Plattentreiber).

Solche Situationen treten sehr selten auf (etwa im Zeitraum von Monaten oder Jahren). Die Restaurierung der Datenbasis geschieht dann mit Hilfe einer (hoffentlich unversehrten) Archiv-Kopie der materialisierten Datenbasis und mit einem Log-Archiv mit allen seit Anlegen der Datenbasis-Archivkopie vollzogenen Änderungen.

14.2 Die Speicherhierarchie

14.2.1 Ersetzen von Pufferseiten

Eine Transaktion referiert Daten, die über mehrere Seiten verteilt sind. Für die Dauer eines Zugriffs wird die jeweilige Seite im Puffer *fixiert*, wodurch ein Auslagern verhindert wird. Werden Daten auf einer fixierten Seite geändert, so wird die Seite als *dirty* markiert. Nach Abschluß der Operation wird der *FIX*-Vermerk wieder gelöscht und die Seite ist wieder für eine Ersetzung freigegeben.

Es gibt zwei Strategien in Bezug auf das Ersetzen von Seiten:

- ¬*steal* : Die Ersetzung von Seiten, die von einer noch aktiven Transaktion modifiziert wurden, ist ausgeschlossen.
- *steal* : Jede nicht fixierte Seite darf ausgelagert werden.

Bei der ¬*steal*-Strategie werden niemals Änderungen einer noch nicht abgeschlossenen Transaktion in die materialisierte Datenbasis übertragen. Bei einem *rollback* einer noch aktiven Transaktion braucht man sich also um den Zustand des Hintergrundspeichers nicht zu kümmern, da die Transaktion vor dem **commit** keine Spuren hinterlassen hat. Bei der *steal*-Strategie müssen nach einem *rollback* die bereits in die materialisierte Datenbasis eingebrachten Änderungen durch ein *Undo* rückgängig gemacht werden.

14.2.2 Zurückschreiben von Pufferseiten

Es gibt zwei Strategien in Bezug auf die Wahl des Zeitpunkts zum Zurückschreiben von modifizierten Seiten:

- *force*: Beim **commit** einer Transaktion werden alle von ihr modifizierten Seiten in die materialisierte Datenbasis zurückkopiert.
- ¬*force*: Modifizierte Seiten werden nicht unmittelbar nach einem **commit**, sondern ggf. auch später, in die materialisierte Datenbasis zurückkopiert.

Bei der ¬*force*-Strategie müssen daher weitere Protokoll-Einträge in der Log-Datei notiert werden, um im Falle eines Fehlers die noch nicht in die materialisierte Datenbasis propagierten Änderungen nachvollziehen zu können. Tabelle 13.1 zeigt die vier Kombinationsmöglichkeiten.

	force	¬force
¬steal	• kein Redo • kein Undo	• Redo • kein Undo
steal	• kein Redo • Undo	• Redo • Undo

Tabelle 13.1: Kombinationsmöglichkeiten beim Einbringen von Änderungen

Auf den ersten Blick scheint die Kombination *force* und ¬*steal* verlockend. Allerdings ist das sofortige Ersetzen von Seiten nach einem **commit** sehr unwirtschaftlich, wenn solche Seiten sehr intensiv auch von anderen, noch aktiven Transaktionen benutzt werden (*hot spots*).

14.2.3 Einbringstrategie

Es gibt zwei Strategien zur Organisation des Zurückschreibens:

- *update-in-place*: Jeder ausgelagerten Seite im Datenbankpuffer entspricht eine Seite im Hintergrundspeicher, auf die sie kopiert wird im Falle einer Modifikation.
- Twin-Block-Verfahren: Jeder ausgelagerten Seite P im Datenbankpuffer werden zwei Seiten P^0 und P^1 im Hintergrundspeicher zugeordnet, die den letzten bzw. vorletzten Zustand dieser Seite in der materialisierten Datenbasis darstellen. Das Zurückschreiben erfolgt jeweils auf den vorletzten Stand, sodaß bei einem Fehler während des Zurückschreibens der letzte Stand noch verfügbar ist.

14.3 Protokollierung der Änderungsoperationen

Wir gehen im weiteren von folgender Systemkonfiguration aus:

- *steal* : Nicht fixierte Seiten können jederzeit ersetzt werden.
- $\neg$*force* : Geänderte Seiten werden kontinuierlich zurückgeschrieben.
- *update-in-place* : Jede Seite hat genau einen Heimatplatz auf der Platte.
- *Kleine Sperrgranulate* : Verschiedene Transaktionen manipulieren verschiedene Records auf derselben Seite. Also kann eine Seite im Datenbankpuffer sowohl Änderungen einer abgeschlossenen Transaktion als auch Änderungen einer noch nicht abgeschlossenen Transaktion enthalten.

14.3.1 Struktur der Log-Einträge

Für jede Änderungsoperation, die von einer Transaktion durchgeführt wird, werden folgende Protokollinformationen benötigt:

- Die *Redo*-Information gibt an, wie die Änderung nachvollzogen werden kann.
- Die *Undo*-Information gibt an, wie die Änderung rückgängig gemacht werden kann.
- Die *LSN (Log Sequence Number)* ist eine eindeutige Kennung des Log-Eintrags und wird monoton aufsteigend vergeben.
- Die *Transaktionskennung TA* der ausführenden Transaktion.
- Die *PageID* liefert die Kennung der Seite, auf der die Änderung vollzogen wurde.
- Die *PrevLSN* liefert einen Verweis auf den vorhergehenden Log-Eintrag der jeweiligen Transaktion (wird nur aus Effizienzgründen benötigt).

14.3.2 Beispiel einer Log-Datei

Tabelle 13.2 zeigt die verzahnte Ausführung zweier Transaktionen und das zugehörige Log-File. Zum Beispiel besagt der Eintrag mit der *LSN* #3 folgendes:

- Der Log-Eintrag bezieht sich auf Transaktion T_1 und Seite P_A.
- Für ein *Redo* muß A um 50 erniedrigt werden.
- Für ein *Undo* muß A um 50 erhöht werden.
- Der vorhergehende Log-Eintrag hat die *LSN* #1.

Schritt	T_1	T_2	Log
			[LSN, TA, PagelD, Redo, Undo, PrevLSN]
1.	**BOT**		[#1, T_1, **BOT**, 0]
2.	$r(A, a_1)$		
3.		**BOT**	[#2, T_2, **BOT**, 0]
4.		$r(C, c_2)$	
5.	$a_1 := a_1 - 50$		
6.	$w(A, a_1)$		[#3, T_1, P_A, A-=50, A+=50, #1]
7.		$c_2 := c_2 + 100$	
8.		$w(C, c_2)$	[#4, T_2, P_C, C+=100, C-=100, #2]
9.	$r(B, b_1)$		
10.	$b_1 := b_1 + 50$		
11.	$w(B, b_1)$		[#5, T_1, P_B, B+=50, B-=50, #3]
12.	**commit**		[#6, T_1, **commit**, #5]
13.		$r(A, a_2)$	
14.		$a_2 := a_2 - 100$	
15.		$w(A, a_2)$	[#7, T_2, P_A, A-=100, A+=100, #4]
16.		**commit**	[#8, T_2,**commit**, #7]

Tabelle 13.2: Verzahnte Ausführung zweier Transaktionen und Log-Datei

14.3.3 Logische versus physische Protokollierung

In dem Beispiel aus Tabelle 13.2 wurden die *Redo*- und die *Undo*-Informationen logisch protokolliert, d.h. durch Angabe der Operation. Eine andere Möglichkeit besteht in der physischen Protokollierung, bei der statt der *Undo*-Operation das sogenannte *Before-Image* und für die *Redo*-Operation das sogenannte *After-Image* gespeichert wird.

Bei der logischen Protokollierung wird

- das *Before-Image* durch Ausführung des *Undo*-Codes aus dem *After-Image* generiert,
- das *After-Image* durch Ausführung des *Redo*-Codes aus dem *Before*-Image generiert.

Um zu erkennen, ob das *Before-Image* oder *After-Image* in der materialisierten Datenbasis enthalten ist, dient die *LSN*. Beim Anlegen eines Log-Eintrages wird die neu generierte LSN in einen reservierten Bereich der Seite geschrieben und dann später mit dieser Seite in die Datenbank zurückkopiert.

Daraus läßt sich erkennen, ob für einen bestimmten Log-Eintrag das *Before-Image* oder das *After-Image* in der Seite steht:

- Wenn die LSN der Seite einen kleineren Wert als die LSN des Log-Eintrags enthält, handelt es sich um das *Before-Image*.
- Ist die LSN der Seite größer oder gleich der LSN des Log-Eintrags, dann wurde bereits das *After-Image* auf den Hintergrundspeicher propagiert.

14.3.4 Schreiben der Log-Information

Bevor eine Änderungsoperation ausgeführt wird, muß der zugehörige Log-Eintrag angelegt werden. Die Log-Einträge werden im *Log-Puffer* im Hauptspeicher zwischengelagert. Abbildung 14.2 zeigt das Wechselspiel zwischen den beteiligten Sicherungskomponenten.

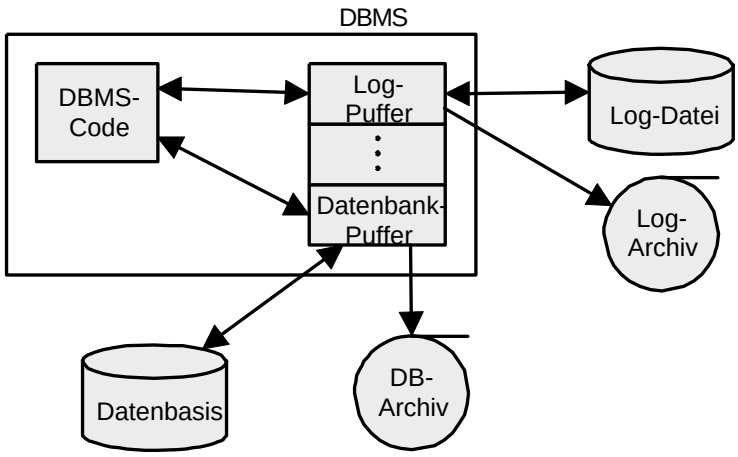

Abbildung 14.2: Speicherhierarchie zur Datensicherung

In modernen Datenbanksystemen ist der Log-Puffer als Ringpuffer organisiert. An einem Ende wird kontinuierlich geschrieben und am anderen Ende kommen laufend neue Einträge hinzu (Abbildung 14.3). Die Log-Einträge werden gleichzeitig auf das temporäre Log (Platte) und auf das Log-Archiv (Magnetband) geschrieben.

14.3.5 WAL-Prinzip

Beim Schreiben der Log-Information gilt das *WAL*-Prinzip (Write Ahead Log):

- Bevor eine Transaktion festgeschrieben (**committed**) wird, müssen alle zu ihr gehörenden Log-Einträge geschrieben werden. Dies ist erforderlich, um eine erfolgreich abgeschlossene Transaktion nach einem Fehler nachvollziehen zu können (*redo*).
- Bevor eine modifizierte Seite ausgelagert werden darf, müssen alle Log-Einträge, die zu dieser Seite gehören, in die Log-Datei geschrieben werden. Dies ist erforderlich, um im Fehlerfall die Änderungen nicht abgeschlossener Transaktionen aus den modifizierten Seiten der materialisierten Datenbasis entfernen zu können (*undo*).

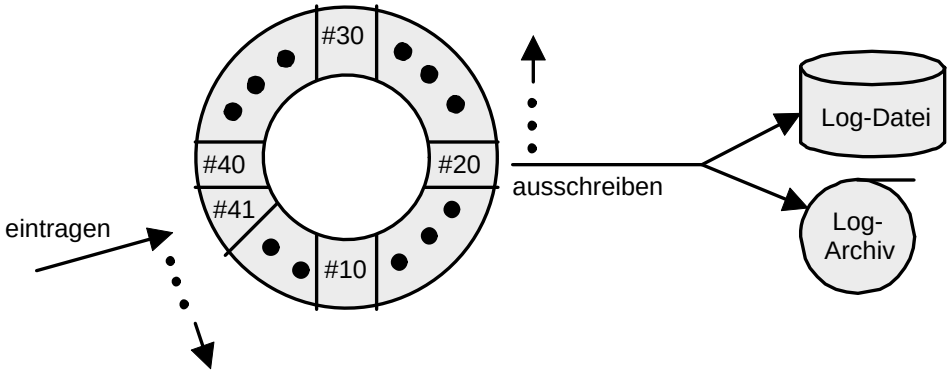

Abbildung 14.3: Log-Ringpuffer

14.4 Wiederanlauf nach einem Fehler

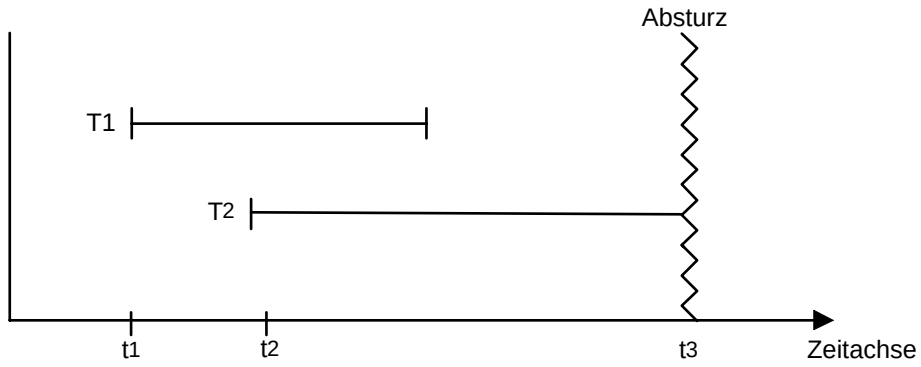

Abbildung 14.4: Zwei Transaktionstypen bei Systemabsturz

Abbildung 14.4 zeigt die beiden Transaktionstypen, die nach einem Fehler mit Verlust des Hauptspeicherinhalts zu behandeln sind:

- Transaktion T_1 ist ein *Winner* und verlangt ein *Redo*.
- Transaktion T_2 ist ein *Loser* und verlangt ein *Undo*.

Der Wiederanlauf geschieht in drei Phasen (Abbildung 14.5):

1. *Analyse*: Die Log-Datei wird von Anfang bis Ende analysiert, um die *Winner* (kann **commit** vorweisen) und die *Loser* (kann kein **commit** vorweisen) zu ermitteln.
2. *Redo*: Es werden alle protokollierten Änderungen in der Reihenfolge ihrer Ausführung in die Datenbasis eingebracht, sofern sich nicht bereits das Afterimage des Protokolleintrags in der materialisierten Datenbasis befindet. Dies ist dann der Fall, wenn die *LSN* der betreffenden Seite gleich oder größer ist als die *LSN* des Protokolleintrags.

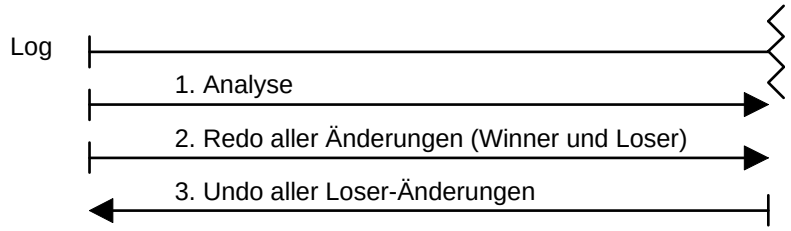

Abbildung 14.5: Wiederanlauf in drei Phasen

3. Undo: Die Log-Datei wird in umgekehrter Richtung, d.h. von hinten nach vorne, durchlaufen. Dabei werden die Einträge von *Winner*-Transaktionen übergangen. Für jeden Eintrag einer *Loser*-Transaktion wird die *Undo*-Operation durchgeführt.

Spezielle Vorkehrungen müssen getroffen werden, um auch Fehler beim Wiederanlauf kompensieren zu können. Es wird nämlich verlangt, daß die *Redo*- und *Undo*-Phasen *idempotent* sind, d.h. sie müssen auch nach mehrmaliger Ausführung (hintereinander) immer wieder dasselbe Ergebnis liefern:

$$undo(undo(...(undo(a))...)) = undo(a)$$
$$redo(redo(...(redo(a))...)) = redo(a)$$

14.5 Lokales Zurücksetzen einer Transaktion

Die zu einer zurückzusetzenden Transaktion gehörenden Log-Einträge werden mit Hilfe des *PrevLSN*-Eintrags in umgekehrter Reihenfolge abgearbeitet. Jede Änderung wird durch eine *Undo*-Operation rückgängig gemacht.

Wichig in diesem Zusammenhang ist die Verwendung von *rücksetzbaren Historien*, die auf den Schreib- und Leseabhängigkeiten basieren.

Wir sagen, daß in einer Historie H die Transaktion T_i von der Transaktion T_j liest, wenn folgendes gilt:

- T_j schreibt ein Datum A, das T_i nachfolgend liest.
- T_j wird nicht vor dem Lesevorgang von T_i zurückgesetzt.
- Alle anderen zwischenzeitlichen Schreibvorgänge auf A durch andere Transaktionen werden vor dem Lesen durch T_i zurückgesetzt.

Eine Historie heißt *rücksetzbar*, falls immer die schreibende Transaktion T_j vor der lesenden Transaktion T_i ihr **commit** ausführt. Anders gesagt: Eine Transaktion darf erst dann ihr **commit** ausführen, wenn alle Transaktionen, von denen sie gelesen hat, beendet sind. Wäre diese Bedingung nicht erfüllt, könnte man die schreibende Transaktion nicht zurücksetzen, da die lesende Transaktion dann mit einem offiziell nie existenten Wert für A ihre Berechnung **commited** hätte.

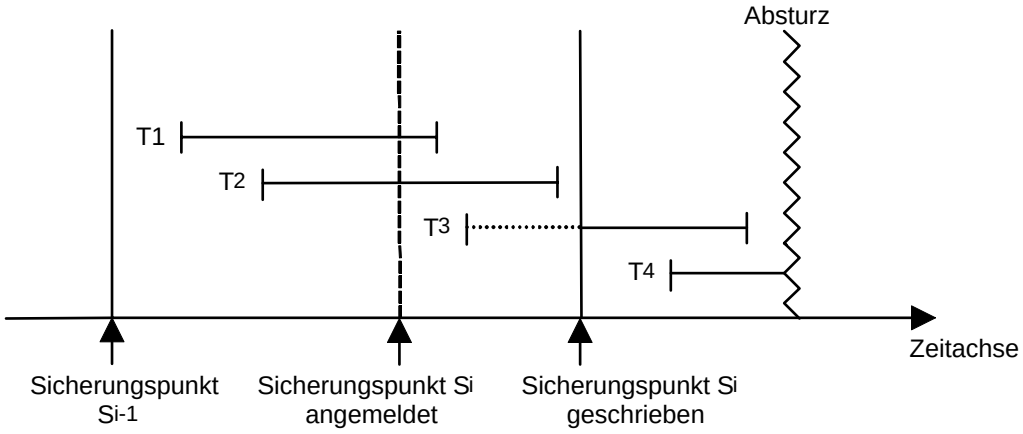

Abbildung 14.6: Transaktionsausführung relativ zu einem Sicherungspunkt

14.6 Sicherungspunkte

Mit zunehmender Betriebszeit des Datenbanksystems wird die zu verarbeitende Log-Datei immer umfangreicher. Durch einen *Sicherungspunkt* wird eine Position im Log vermerkt, über den man beim Wiederanlauf nicht hinausgehen muß.

Abbildung 14.6 zeigt den dynamischen Verlauf. Nach Anmeldung des neuen Sicherungspunktes S_i wird die noch aktive Transaktion T_2 zu Ende geführt und der Beginn der Transaktion T_3 verzögert. Nun werden alle modifizierten Seiten auf den Hintergrundspeicher ausgeschrieben und ein transaktionskonsistenter Zustand ist mit dem Sicherungspunkt S_i erreicht. Danach kann man mit der Log-Datei wieder von vorne beginnen.

14.7 Verlust der materialisierten Datenbasis

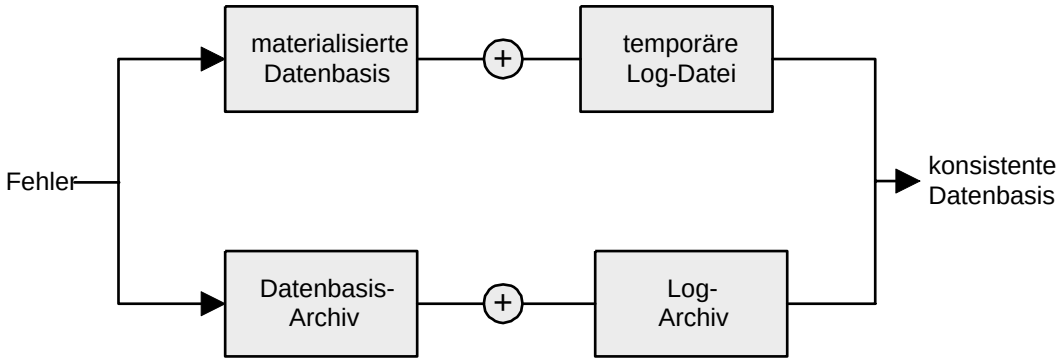

Abbildung 14.7: Zwei Recovery-Arten

Bei Zerstörung der materialisierten Datenbasis oder der Log-Datei kann man aus der Archiv-Kopie und dem Log-Archiv den jüngsten, konsistenten Zustand wiederherstellen.

Abbildung 14.7 faßt die zwei möglichen Recoveryarten nach einem Systemabsturz zusammen:

- Der obere (schnellere) Weg wird bei intaktem Hintergrundspeicher beschritten.
- Der untere (langsamere) Weg wird bei zerstörtem Hintergrundspeichert beschritten.

14.8 Datensicherung beim SQL-Server 2000

```
EXEC sp_addumpdevice                -- fuehre ein Sicherungsmedium ein
     'disk',                        -- als Datei
     'unidump',                     -- logischer Name
     'c:\dump\unidump.dat'          -- physikalischer Speicherort

backup database uni to unidump      -- sichere Datenbank
```

Sicherung der Datenbank

```
restore database uni                -- restauriere Datenbank
from unidump                        -- vom Datenbankarchiv
```

Wiederherstellen der Datenbank

```
EXEC sp_addumpdevice                -- fuehre ein Sicherungsmedium ein
     'disk',                        -- als Datei
     'unilog',                      -- logischer Name
     'c:\dump\unilog.dat'           -- physikalischer Speicherort

backup log uni to unilog            -- sichere Log-File
```

Sicherung des Log-Files

```
restore log uni                     -- restauriere Logfile
from unilog                         -- vom Log-Archiv
```

Wiederherstellen des Log-Files