

# Recovery from „Bad“ User Transactions

Frank Weißenborn

28.03.2007

## Einleitung

## Einleitung

## Existierende Lösungen

## Einleitung

## Existierende Lösungen

## Grundlagen der neuen Lösung

Einleitung

Existierende Lösungen

Grundlagen der neuen Lösung

neuer Ansatz

Einleitung

Existierende Lösungen

Grundlagen der neuen Lösung

neuer Ansatz

Datenbankoperationen nach Entfernung der ungültigen Transaktion

Einleitung

Existierende Lösungen

Grundlagen der neuen Lösung

neuer Ansatz

Datenbankoperationen nach Entfernung der ungültigen Transaktion

Anmerkungen

## Einleitung

Existierende Lösungen

Grundlagen der neuen Lösung

neuer Ansatz

Datenbankoperationen nach Entfernung der ungültigen Transaktion

Anmerkungen

- ▶ kein Mechanismus um einen Benutzer zu hindern „schlechte“ Transaktionen einzugeben
  - ▶ falscher Betrag an Bank überwiesen
  - ▶ Abrechnen für Dinge, die nicht versandt worden sind
- ▶ „schlechte“ Transaktion wird zurückgenommen
- ▶ Konsistenz der Datenbank soll erhalten bleiben

Einleitung

**Existierende Lösungen**

Grundlagen der neuen Lösung

neuer Ansatz

Datenbankoperationen nach Entfernung der ungültigen Transaktion

Anmerkungen

## „point in time“-Recovery

- ▶ einspielen eines Backups
- ▶ einspielen aller Transaktionen, die seit dem Backup bis zur „schlechten“ Transaktion ausgeführt worden sind

### Nachteil:

- ▶ alle Transaktionen nach der „schlechten“ werden zurückgenommen und somit auch die Ergebnisse der Transaktionen
- ▶ hunderte bis tausende Transaktionen nachträglich ausgeführt
- ▶ Einspielen des Backups und der Transaktionen dauert lang, lange Ausfallzeiten

Einleitung

Existierende Lösungen

**Grundlagen der neuen Lösung**

neuer Ansatz

Datenbankoperationen nach Entfernung der ungültigen Transaktion

Anmerkungen

Ziel:

- ▶ Automatisches Erkennen und Isolieren der Effekte einer „schlechten“ Transaktion
- ▶ Verhindern der langen Ausfallzeit und Verhindern der Zurücknahme gültiger Transaktionen

Verhindern der langen Ausfallzeit: Verwenden einer „Transaction Time“ Datenbank (ImmortalDB)

Verhindern der Zurücknahme: neuer Ansatz

- ▶ Jede Transaktion erzeugt neuen Stand der DB
- ▶ große Teile der verschiedenen Stände sind gleich
- ▶ nur veränderter Teil wird mit Zeitstempel der Transaktion versehen
- ▶ Status zu gewisser Zeit: Zurückgehen bis zu den Daten mit dem Zeitstempel(oder kleiner)
- ▶ Backup durch „roll-forward“ möglich
- ▶ „point in time“: alle Stände löschen, die Zeitstempel über gewünschten haben

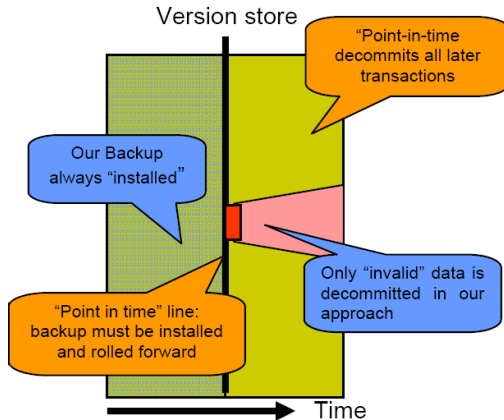
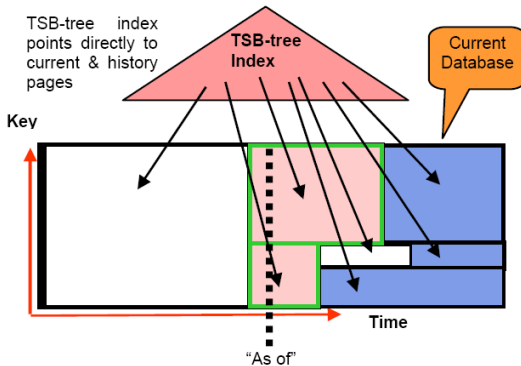


Figure 1: Removing invalid data produced by an erroneous user transaction.

Zeitstempel wird in den Versionen der Daten gespeichert

- ▶  $TT$  - Transaktions Zeit
- ▶  $d.TT^+$  - Startzeit für Gültigkeit der Daten „d“
- ▶  $W(T)$  - Versionen, die von Transaktion T geschrieben worden sind, also alle „d“, die mit „TT“ markiert sind
- ▶  $R(T)$  - Versionen, die von Transaktion T gelesen wurden

Daten sind gültig von  $d.TT^+$  bis  $d.TT^-$ , wobei  $d.TT^-$  Start der nächsten Version ist



Key & Time splitting produce rectangular KT regions.

**Figure 3: A TSB-tree.**

## In einer Seite:

- ▶ als verkettete Liste
- ▶ aktuelle Version ist erste Position
- ▶ Verweis vom Slot Array auf Anfang der Liste
- ▶ gelöschte Daten: neue Version, markiert als gelöscht

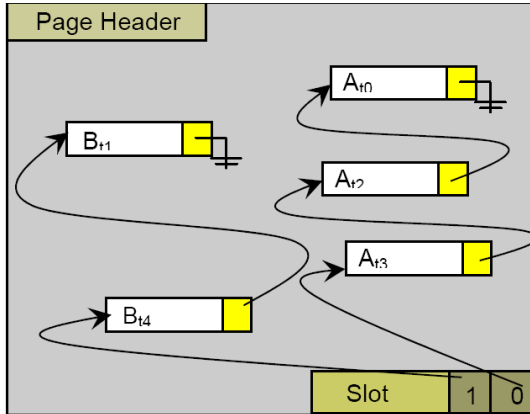


Figure 2: A page containing versioned data for records with keys A and B.

## zwischen den Seiten:

- ▶ Seiten werden zeitlich oder schlüsselbasiert getrennt
- ▶ Zugriff auf Schlüssel/Zeit Regionen mittels eines TSB Baums (Time Split B-Tree)
- ▶ Einführung von „Backpointer“  $\implies$  Speicherung der Seitennummer + Slotarray bei Time-Split
- ▶ gelöschte Daten werden nicht in neue Seite übernommen

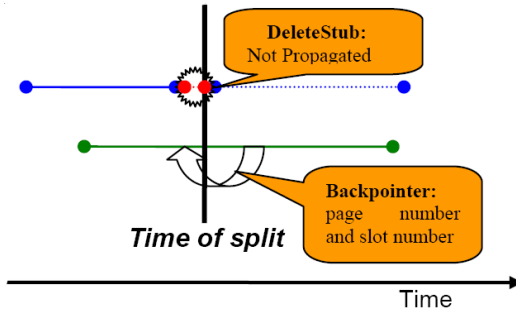


Figure 4: Handling versions between time-split pages.

- ▶  $C(T_c)_0 = W(T_c)$ 
  - ▶  $T_c$  fehlerhafte Transaktion
  - ▶ alles was von einer fehlerhaften Transaktion geschrieben wurde ist fehlerhaft
- ▶  $C(T_c)_{i+1} = C(T_c)_i \cup \{W(T_{i+1}) \mid \exists (T_{i+1}) R(T_{i+1}) \cap C(T_c)_i \neq \emptyset\}$ 
  - ▶ alle Transaktionen, die fehlerhafte Daten lesen sind auch fehlerhaft
- ▶  $C(T_c) = C(T_c)_j$  mit  $\neg \exists T_{j+1} \quad TS(T_j) < TS(T_{j+1})$  mit  $R(T_{j+1}) \cap C(T_c)_j \neq \emptyset$ 
  - ▶ Berechnung ist vollständig, wenn es keine Transaktion  $T_{j+1}$  gibt, die noch fehlerhafte Daten produziert/liest

Beispiel:  $T_0 : W = 10 \ X = 20 \ Y = 30 \ Z = 40$

LOG

T1: R(X <sub>0</sub> ) W(X <sub>1</sub> :21)
T2: R(W <sub>0</sub> , Y <sub>0</sub> ) W(W <sub>1</sub> :12)
T3: R(X <sub>1</sub> , Y <sub>0</sub> ) W(X <sub>2</sub> :33)
T4: R(Y <sub>0</sub> , Z <sub>0</sub> ) W(Y <sub>1</sub> :34)
T5: R(X <sub>2</sub> , Z <sub>0</sub> ) W(Z <sub>1</sub> :45)
T6: R(W <sub>1</sub> , Z <sub>1</sub> ) W(W <sub>2</sub> :16)

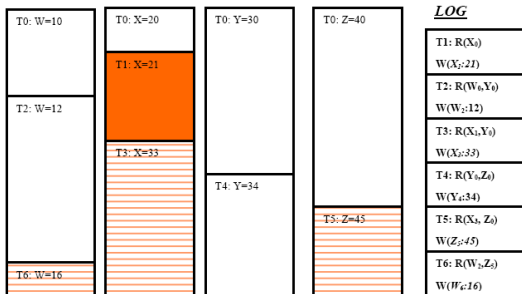


Figure 5: Transaction time database with invalid data.

Zur Bestimmung von  $C(T_c)$  : Registrierung von  $W(T)$  und  $R(T)$  notwendig

- ▶ alle Lesevorgänge von T als ein Feld geloggt
- ▶ aktualisierte Daten werden auch gelesen  $\Rightarrow$  nur Daten loggen, die gelesen aber nicht geschrieben werden
- ▶ nur Transaktionen loggen, die Schreibvorgänge ausführen
- ▶ Transaktion, die ungültige Daten liest, schreibt auch ungültige Daten
- ▶ loggen der Daten zwischen Abfrageoptimierer und Kern
- ▶ Registrierung von einzelnen Datensätzen und Bereichen
- ▶ Optimierung um keine Duplikate zu loggen

## Phantome:

- ▶ Transaktion  $T_1$  löscht einen Datensatz  $D_i$
- ▶ andere Transaktion  $T_2$  liest Daten (sollte  $D_i$  enthalten)
- ▶  $T_1$  ungültig
  - ▶ eingelezene Datensätze von  $T_2$  geloggt  $\Rightarrow D_i$  nicht mit geloggt,  $T_2$  gültig
  - ▶ Bereich von Datensätzen geloggt  $\Rightarrow D_i$  im Bereich,  $T_2$  ungültig
- ▶ große Datenmengen zuerst im Log, da bei der Erkennung früher abgebrochen werden kann

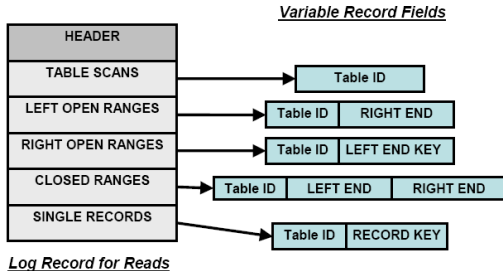


Figure 6: Log Record for Reads

Einleitung

Existierende Lösungen

Grundlagen der neuen Lösung

**neuer Ansatz**

Datenbankoperationen nach Entfernung der ungültigen Transaktion

Anmerkungen

1. Identifizierung der „schlechten“ Transaktion und ihrer Versionen
2. Rekursives Finden von ungültigen Daten, die von einer Transaktion geschrieben wurden, die ungültige Daten gelesen hat
3. Kennzeichnen der ungültigen Daten

- ▶ Datensatz ist vom Benutzer zu einer bestimmten Zeit  $TS_c$  als unzulässig erkannt worden
- ▶ Datenstruktur, die Menge der ausgeführten Transaktionen speichert:  $CS(\text{CommitSet})$
- ▶ Durchgehen der Log Datei von hinten
  - ▶ Commit Log : Speichern der ID, des Zeitstempels und der LSN(Log Sequence Number) der ersten Anweisung der Transaktion
  - ▶ Update/Insert/Delete Log:
    - ▶ Analyse welche Transaktion
    - ▶ falls Zeitstempel  $TS \leq TS_c$  und gleiche Tabelle/Schlüssel  $\Rightarrow T_c$  gefunden

- ▶ Datenstruktur, die Menge der ungültigen Daten speichert:  
*IRS*(Invalid Record Set)
- ▶ Durchgehen der Log Datei von vorn, beginnend bei kleinster LSN aller Transaktionen mit  $TS(T) \geq TS(T_c)$
- ▶ Speichern der Tabelle, des Schlüssels und des kleinsten, als ungültig erkannten, Zeitstempels

1. Schritt: Speichern aller Daten, die von  $T_c$  geschrieben worden sind in *IRS*
2. Schritt: Finden von weiteren ungültigen Daten
  - (a) verändernder Logeintrag:
    - ▶ veränderte Daten von Transaktion T temporär merken
    - ▶ in *IRS* schauen ob vorher schon ungültig, falls ja dann immer noch ungültig  $\Rightarrow$  Transaktion ist ungültig
  - (b) lesender Logeintrag: in *IRS* schauen ob ungültige Daten gelesen wurden, falls ja, Transaktion ungültig
  - (c) Ender der Transaktion: entscheidbar ob Transaktion T gültig
    - ▶ falls ja: Löschen der temporär gespeicherten Updates der Transaktion
    - ▶ falls nein: Übernahme der von T geschriebenen Daten in *IRS*

Ende des Algorithmus: jede Transaktion, die nach  $T_c$  ausgeführt angeschaut, ungültigen Daten stehen in *IRS*

- ▶ ein Bit im Row Header
- ▶ DB ist dabei offline
- ▶ normale Datensätze: jede Version jedes Datensatzes im IRS als ungültig bis zur ersten ungültigen Version
  - ▶ aktuelle Version des Datensatzes abfragen
  - ▶ Pointer benutzen um bis zum gewünschten Zeitstempel zu kommen, alle Versionen als ungültig markieren
- ▶ gelöschte Datensätze:
  - ▶ Datensatz im IRS wurde nicht auf aktuelle Seite übernommen  
⇒ markieren der Seite mit IDS Flag
  - ▶ weiter zur Vorgängerseite gehen und diese auch markieren, falls nötig
  - ▶ wenn Datensatz gefunden, dann markieren als ungültig und wie normalen Datensatz behandeln

Einleitung

Existierende Lösungen

Grundlagen der neuen Lösung

neuer Ansatz

**Datenbankoperationen nach Entfernung der ungültigen Transaktion**

Anmerkungen

## Zugriff auf einzelnen Datensatz

- (a) Seite enthält Version des Datensatzes: zurückgehen bis zur ersten gültigen Version
- (b) Seite enthält keine Version des Datensatzes
  - (1) Seite ist nicht mit IDS markiert  $\Rightarrow$  Zurückgeben der leeren Menge
  - (2) Seite ist mit IDS markiert  $\Rightarrow$  Suche auf Vorgängerseite fortsetzen

## Bereichssuche

- ▶ Jedes Element wird einzeln behandelt
- ▶ 2 Mengen ANSWER und PENDING
- ▶ Seite ohne IDS Markierung
  - ▶ falls gültiges Version des Datensatzes in der Seite ⇒ Version in ANSWER
  - ▶ falls gültiges Löschen des Datensatzes in der Seite ⇒ Datensatz weglassen
  - ▶ falls keine gültige Version des Datensatzes in der Seite ⇒ Datensatz in PENDING
  - ▶ falls alle Datensätze auf der Seite geprüft, weitermachen mit verbliebenen in PENDING auf der nächsten Seite
  - ▶ fertig, wenn PENDING leer
- ▶ Seite mit IDS Markierung
  - ▶ wie bei Seiten ohne IDS Markierung
  - ▶ neben den Elementen in PENDING auch gelöschte mitbetrachten

## ▶ Update

- ▶ finden der aktuellen Version
- ▶ falls gültig: kopieren der Version und Anfang in die Liste einfügen, dann Updaten des neuen Elements
- ▶ falls ungültig: erste gültige Version finden, als Basis nehmen und den Wert aktualisieren

## ▶ Insert

- ▶ suchen ob es den Datensatz schon einmal gibt
- ▶ falls ja, als ungültig: diesen als Basis für den Neuen nehmen
- ▶ falls nein: neuen Datensatz anlegen

## ▶ Delete

- ▶ suchen ob der Datensatz schon existiert
- ▶ falls ja: neue Version die Datensatz als gelöscht markiert
- ▶ falls nein: ignorieren

- ▶ IDS Markierungen sind bei Abfragen teuer  $\Rightarrow$  Versuch Markierungen zu entfernen
- ▶ 2 Möglichkeiten wenn Seite geteilt wird
  1. Time Split: auf beiden Seiten sind die gleichen Datensätze, IDS Markierung bleibt erhalten
  2. Key Split:
    - ▶ falls IDS Markierung, dann mindestens eine der beiden neuen Seiten IDS Markierung
    - ▶ Seiten werden geteilt, Bereich der neuen Seite wird bestimmt, Rückwärtssuche ob gelöscht Element auf der Seite, falls ja IDS Markierung übernehmen
- ▶ Update:
  - ▶ Rückwärtssuche ob nur ein gelöscht Element, falls das Einzige gelöschte Element Schlüssel des Updates hat IDS Markierung entfernen

Einleitung

Existierende Lösungen

Grundlagen der neuen Lösung

neuer Ansatz




Datenbankoperationen nach Entfernung der ungültigen Transaktion

**Anmerkungen**

- ▶ nicht alle Anwendungsfälle brauchen die Komplexität der Datenvollständigkeit  $\Rightarrow$  Implementierung optional
- ▶ manche Unternehmen brauchen die kurzen Ausfallzeiten
  - ▶ erkaufen durch Speicherung von mehr Daten
  - ▶ höherer Plattenbedarf
  - ▶ Administrator kann alte Datenbankversionen löschen

- ▶ Daten wurden isoliert, sind aber nicht mehr abrufbar
- ▶ Administrator möchte Möglichkeit, auf ungültige Daten zuzugreifen
- ▶ Einführung von speziellen SQL Komandos um auf ungültige Daten zugreifen zu können

- ▶ größte Problem der Implementierung: gelöschte Daten die nicht weitergegeben werden
- ▶ hier: ein Bit IDS wegen Beschränkungen im SQL Server
- ▶ andere Möglichkeiten:
  - ▶ IDS Zählerfeld: Vereinfacht die Bereinigung
  - ▶ „Current State Healing“: ungültige gelöschte Elemente auf aktuelle Seite übertragen
    - ▶ mehr Speicherplatzverbrauch
    - ▶ Effekte bei Operation auf aktueller Datenbank werden vermieden
    - ▶ IDS Flags trotzdem noch notwendig für historische Anfragen

-  Lomet, D., Vagena, Z., and Barga, R.  
*Recovery from „Bad“ User Transactions.*  
SIGMOD Conference, Chicago (June 2006)
-  Lomet, D., Barga, R., Mokbel, M., Shegalov, G., Wang, R.,  
and Zhu, Y.  
*Transaction Time Support Inside a Database Engine.*  
ICDE Conference, Atlanta, GA (April 2006)
-  Lomet, D., Barga, R., Mokbel, M., Shegalov, G., Wang, R.,  
and Zhu, Y.  
*Immortal DB: Transaction Time Support for Sql Server.*  
SIGMOD Conference, Baltimore, MD (June 2005)