Begriff

Eine Transaktionen ist eine logische Zusammenfassung von Datenmanipulationen, die in ihrer Gesamtheit entweder ganz oder gar nicht durchgeführt werden.

Änderungen des Datenbestands durch Insert-, Update- oder Delete-Anweisungen sind erst nach Abschluss einer Transaktion (COMMIT) permanent in der Datenbank gespeichert.

Beispiel

Es soll ein Wareneingang über 10 Stück für ein Kaufteil gebucht werden. Hierzu muss der Lagerbestand des Teils erhöht und ein Protokollsatz in das Lagerbuchungsjournal eingefügt werden. Der gesamte Buchungsprozess soll entweder komplett oder überhaupt nicht, d.h. als Transaktion, durchgeführt werden.

Tabelle ART (Artikel)

ARTIKEL	BESTAND
10.1001	5

Tabelle LBJ (Lagerbuchungsjournal)

BUCHZEIT	ARTIKEL	BUCHART	MENGE
10.01.2000 11:09:31	10.1001	I (Inventur)	5

Anweisungen:

```
UPDATE ART SET BESTAND=15 WHERE ARTIKEL='10.1001';
INSERT INTO LBJ VALUES(TO_DATE('11.01.2000 17:21:38','DD.MM.YYYY HH:NN:SS'),'10.1001','E',10);
COMMIT;
```

Annahme: Eine neue Transaktion ist eröffnet.

Phase-1
- prüfen, ob der Satz aus Tabelle ART mit der Teilenummer 10.1001 bereits durch eine andere Session gesperrt wurde. Falls ja, innerhalb der festgelegten Wartezeit auf Freigabe warten, danach Fehlermeldung.
- Eintrag im Sperrprotokoll anlegen

Table	Record	Session
ART	128	12

Anmerkung:

Die Einzelheiten des Sperrkonzepts werden durch die Datenbankhersteller generell nicht veröffentlicht. Die nachfolgenden Ausführungen beziehen sich daher auf ein fiktives, beispielhaftes Modell.

Phase-2
- Erzeugen der Tabellenstruktur ART in einer sitzungsspezifischen Datei des Datenbankservers und Speichern des geänderten aktuellen Datensatzes in diesem temporären Bereich.

Phase-3
- prüfen, ob bereits eine Einfügesperre (RECORD=NULL) für die Tabelle LBJ im Sperrprotokoll enthalten ist. Falls ja, innerhalb der festgelegten Wartezeit auf Freigabe warten, danach Fehlermeldung.
- Eintrag im Sperrprotokoll anlegen

Table	Record	Session
ART	128	12
LBJ	NULL	12

Phase-4
- Erzeugen der Tabellenstruktur LBJ in einer sitzungsspezifischen Datei des Datenbankservers und Speichern des neuen Datensatzes in diesem temporären Bereich.

Phase-5 (Commit)
- Speichern der temporären sitzungsspezifischen Daten in der Datenbank. Eventuell Anlegen eines AfterImages mit den durchgeführten Änderungen, um eine Rekonstruktion der Datenbank mittels Änderungsprotokollen zu ermöglichen.
- Aufheben der Sperren für Session 12.

Alternativ

Phase-1
unverändert

Phase-2
- Kopieren der bisherigen Feldinhalte des Datensatzes aus Tabelle ART als BeforeImage in eine Log-Datei.
- Ändern der entsprechenden Felder gemäß Update-Anweisung direkt in der Datenbank.

Phase-3
unverändert

Phase-4
- Protokollieren der Einfügeaktion in der Log-Datei.
- Einfügen des Buchungssatzes Tabelle LBJ in die Original-Datenbank.

Phase-5 (Commit)
- BeforeImage löschen oder evtl. als AfterImage zum Zwecke der Datenrekonstruktion beibehalten.
- Aufheben der Sperren für Session 12.

Rollback

Wiederherstellen des Datenbestands auf den Zustand vor Beginn einer Transaktion.

Ein Rollback kann programmgesteuert per Anweisung (Rollback) durchgeführt werden.

Nach Systemabstürzen werden nicht abgeschlossene Transaktionen automatisch durch den Datenbankserver zurückgesetzt.

Backup

Die meisten Datenbanksysteme verfügen über ein integriertes Datensicherungsprogramm, mit dem die Daten einer geöffneten Datenbank (24-Stunden-Betrieb) in eine gesonderte Sicherungsdatei kopiert werden können. Vor dem Erstellen eines Online-Backups sollte, falls die Datenbank die Funktionalität bietet, eine Integritätsprüfung vorgenommen werden, um das Erstellen eines Backups von einer beschädigten Datenbank zu vermeiden.

Die Sicherungsdatei kann dann, gemeinsam mit anderen Daten, auf ein externes Medium ausgelagert werden.

Vorhandene AfterImage-Dateien können nach Erstellen eines Datenbankbackups gelöscht werden.

Recovery

Infolge gravierender Systemstörungen (Plattencrash etc.) kann das Einspielen der Sicherungsdatei erforderlich werden. Falls AfterImages nicht vorhanden sind, müssen alle Transaktionen, die nach dem letzten Backup durchgeführt worden sind, manuell wiederholt werden (Horror).

Unter Zuhilfenahme des Backups und der AfterImages kann die Datenbank auf den Zeitpunkt unmittelbar vor dem Ausfall rekonstruiert werden.

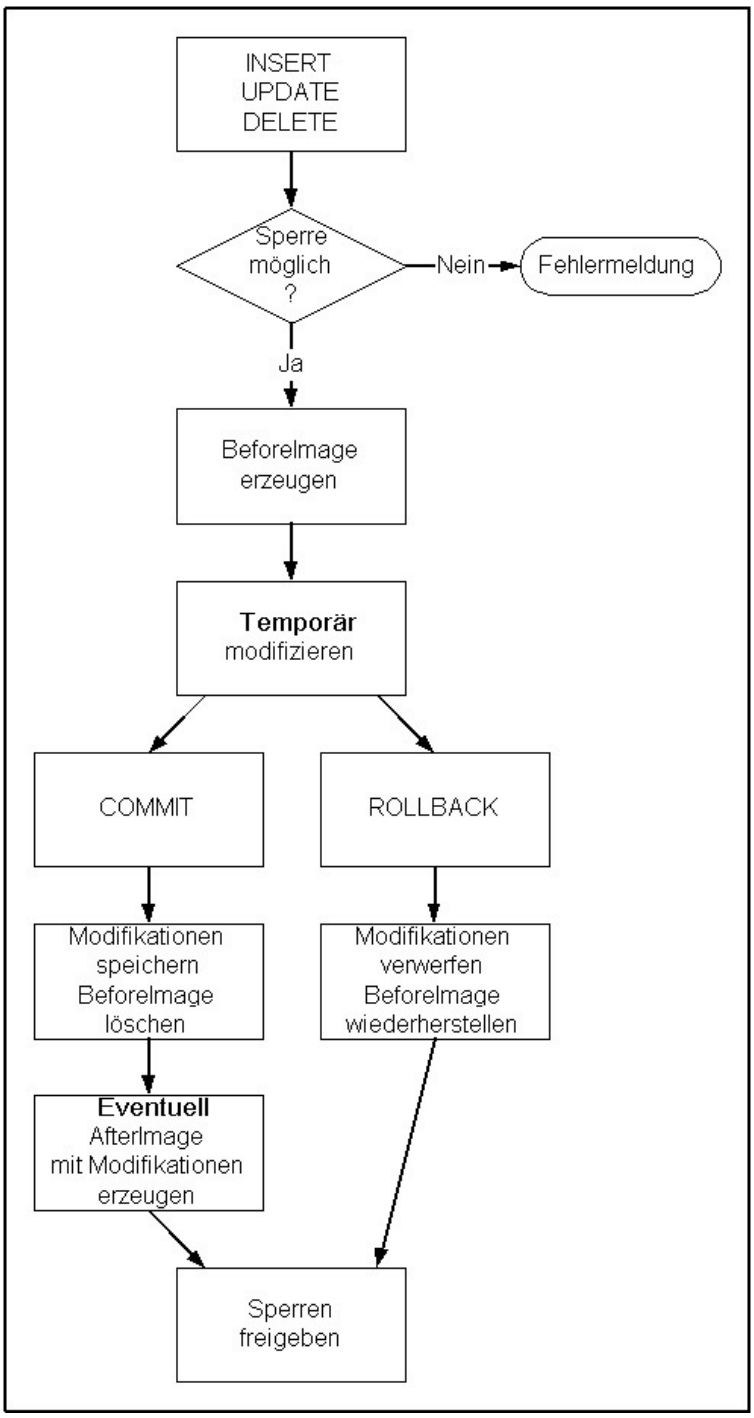
ReadOnly / ReadWrite

Transaktionen können mit ausschließlichem Lesezugriff eröffnet werden. Datenmanipulationen sind dann nicht möglich.

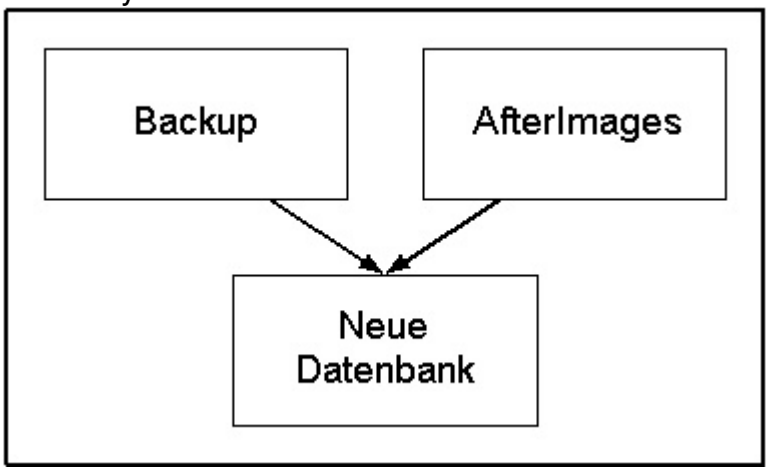
Merke

- Eine Transaktion wird entweder komplett oder gar nicht durchgeführt. (Alles oder Nichts)
- Transaktionen im Programm niemals durch Benutzerdialoge unterbrechen (Sperren!!!)
- BeforeImages enthalten den Datenzustand vor Beginn einer Transaktion (notwendig für Rollback)
- AfterImages enthalten die Modifikationen durch Transaktionen (notwendig für Recovery)

Transaktion



Recovery



Konkurrierende Zugriffe

Die erste und wichtigste Einstellung für einen reibungslosen Mehrplatzbetrieb ist der **ISOLATION LEVEL**.

Praxisrelevant sind die nachfolgenden Einstellungen:
- READ COMMITTED
- SERIALIZABLE

Beide ISOLATION LEVEL sind so implementiert, dass Änderungen der eigenen Session für andere Sessions erst nach Abschluss der eigenen Transaktion sichtbar sind.

READ COMMITTED

Änderungen einer fremden Session (INSERT,UPDATE,DELETE) sind unmittelbar nach Abschluss der Transaktion durch die fremde Session für alle anderen beteiligten Sessions sichtbar.

Eine wiederholt angeforderte Datenmenge mit identischer Sql-Select-Anweisung kann sich von der erstmalig angeforderten Datenmenge erheblich unterscheiden.

SERIALIZABLE

Änderungen fremder Sessions (INSERT,UPDATE,DELETE) sind erst nach Beendigung der eigenen Transaktion durch COMMIT bzw. ROLLBACK innerhalb der eigenen Session sichtbar.

Eine wiederholt angeforderte Datenmenge mit identischer Sql-Select-Anweisung ist garantiert mit der erstmalig angeforderten Datenmenge identisch.

Der ISOLATION LEVEL wird mit folgender Sql-Anweisung eingestellt:

SET TRANSACTION ISOLATION LEVEL {READ COMMITTED | SERIALIZABLE}

Die zweite wichtige Einstellung ist für eine ergebnisorientierte Zusammenfassung mehrerer Sql-Anweisungen zu einer Transaktion unabdingbar:

SET AUTOCOMMIT OFF

Beispiele (ISOLATION LEVEL)

Wir aktivieren zwei Instanzen des Oracle-Dienstprogramms SqlPlus und erzeugen somit zwei Datenbanksessions, mit denen wir den Mehrplatzbetrieb und konkurrierende Zugriffe simulieren.

ISOLATION LEVEL READ COMMITTED

Session A und Session B:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET AUTOCOMMIT OFF;
SELECT ARTIKEL,PREIS1 FROM ART ORDER BY ARTIKEL;

ARTIKEL PREIS1
------- ------
10.1001  39,99
10.1016  99,99
25.3282   4,98
56.7954   0,40
80.0001  49,95
```

Session A:

```
UPDATE ART SET PREIS1=37.95 WHERE ARTIKEL='10.1001';
SELECT ARTIKEL,PREIS1 FROM ART WHERE ARTIKEL='10.1001';

ARTIKEL PREIS1
------- ------
10.1001  37,95
```

Anmerkung: Die Änderung ist innerhalb der eigenen Session, auch ohne COMMIT bzw. ROLLBACK, sofort sichtbar.

Session B:

```
SELECT ARTIKEL,PREIS1 FROM ART WHERE ARTIKEL='10.1001';

ARTIKEL PREIS1
------- ------
10.1001  39,99
```

Anmerkung: Die Änderung durch Session A ist noch nicht sichtbar, da Session A die Transaktion noch nicht beendet hat.

Session A:

```
COMMIT;
```

Session B:

```
SELECT ARTIKEL,PREIS1 FROM ART WHERE ARTIKEL='10.1001';

ARTIKEL PREIS1
------- ------
10.1001  37,95
```

Anmerkung: Session A hat die Transaktion mit COMMIT abgeschlossen. Damit ist der neue Preis von 37,95 € permanent in der Datenbank gespeichert. Da unsere eigene Session mit dem ISOLATION LEVEL READ COMMITTED arbeitet, unterscheidet sich die wiederholt angeforderte Datenmenge, trotz identischer SQL-Anweisung, von der erstmalig angeforderten Datenmenge.

ISOLATION LEVEL SERIALIZABLE

Session B:

```
COMMIT;
```

Anmerkung: Eine Änderung des ISOLATION LEVEL ist nur nach unmittelbar vorhergehendem COMMIT bzw. ROLLBACK möglich.

Session A und Session B:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET AUTOCOMMIT OFF;
SELECT ARTIKEL,PREIS1 FROM ART ORDER BY ARTIKEL;

ARTIKEL PREIS1
------- ------
10.1001  37,95
10.1016  99,99
25.3282   4,98
56.7954   0,40
80.0001  49,95
```

Session A:

```
UPDATE ART SET PREIS1=38.75 WHERE ARTIKEL='10.1001';
SELECT ARTIKEL,PREIS1 FROM ART WHERE ARTIKEL='10.1001';

ARTIKEL PREIS1
------- ------
10.1001  38,75
```

Session B:

```
SELECT ARTIKEL,PREIS1 FROM ART WHERE ARTIKEL='10.1001';

ARTIKEL PREIS1
------- ------
10.1001  37,95
```

Session A:

```
COMMIT;
```

Session B:

```
SELECT ARTIKEL,PREIS1 FROM ART WHERE ARTIKEL='10.1001';

ARTIKEL PREIS1
------- ------
10.1001  37,95
```

Anmerkung: Obwohl Session A die Transaktion mit COMMIT beendet hat und somit der neue Preis von 38,75 € permanent in der Datenbank gespeichert ist, sieht Session B, auch nach wiederholter Anforderung der Ergebnismenge, die Änderung nicht. Durch die Wahl des ISOLATION LEVEL SERIALIZABLE ist sichergestellt, dass beim wiederholten Anfordern einer Ergebnismenge mit identischer Sql-Anweisung auch ein mit der ersten Ergebnismenge identischer Dateninhalt bereitgestellt wird.

Session B:

```
COMMIT;
SELECT ARTIKEL,PREIS1 FROM ART WHERE ARTIKEL='10.1001';

ARTIKEL PREIS1
------- ------
10.1001  38,75
```

Anmerkung: Der eigenen Session wird bei Verwendung des ISOLATION LEVEL SERIALIZABLE erst nach eigenem COMMIT bzw. ROLLBACK eine aktualisierte Ergebnismenge zur Verfügung gestellt.

Hinweise für Softwareentwickler

Entwickler, die Datenbankanwendungen für den Mehrplatzbetrieb konzipieren und programmieren, kennen nur eine einzige sinnvolle Einstellung:

R E A D C O M M I T T E D

Qualifizierte Datenbankentwickler wissen, dass sie beim wiederholten Anfordern von Ergebnismengen, trotz identischer Sql-Select-Anweisung, mit unterschiedlichen Ergebnismengen rechnen müssen und verzichten daher bewusst auf den Einsatz dieser Option. Falls ein wiederholter Zugriff auf bereits gelesene Daten unvermeidbar ist, speichern diese die erstmalig gelesenen Daten in dynamisch erzeugten Hauptspeicherbereichen und greifen dann, im weiteren Programmverlauf, auf die beim erstmaligen Lesen im Arbeitsspeicher hinterlegten Daten zurück. Als angenehmer Nebeneffekt kann zusätzlich eine erhöhte Verarbeitungsgeschwindigkeit der gesamten Datenbankanwendung verbucht werden. Einer möglichen Diskussion bezüglich der Verschwendung von Hauptspeicherplatz entziehe ich mich durch die nachfolgende Aussage: Schnee von Gestern!

Beispiel (Konkurrierende Zugriffe)

Wir starten erneut zwei Instanzen (Sessions) des Oracle-Dienstprogramms SqlPlus.

Session A und Session B:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET AUTOCOMMIT OFF;
SELECT ARTIKEL,PREIS1 FROM ART ORDER BY ARTIKEL;

ARTIKEL PREIS1
------- ------
10.1001  38,75
10.1016  99,99
25.3282   4,98
56.7954   0,40
80.0001  49,95
```

Session A:

```
UPDATE ART SET PREIS1=39.99 WHERE ARTIKEL='10.1001';
SELECT ARTIKEL,PREIS1 FROM ART WHERE ARTIKEL='10.1001';

ARTIKEL PREIS1
------- ------
10.1001  39,99
```

Session B:

```
UPDATE ART SET PREIS1=40 WHERE ARTIKEL='10.1001';
```

Oops! Abgeschmiert? Nein!

Anmerkung: Unsere Session wartet auf Freigabe der Sperre, die das RDBMS nach dem Update durch Session A erzeugt hat. Wir warten solange, bis die Session A die Transaktion mit COMMIT oder ROLLBACK beendet. Oracle kennt, im Gegensatz zu anderen Datenbanken, keine maximale Wartezeit für Sperrversuche. Eine Fehlermeldung bezüglich der vorhanden Sperrsituation wird deshalb seitens Oracle, ebenfalls im Gegensatz zu anderen Datenbanksystemen, nicht ausgelöst.

Session A:

```
COMMIT;
SELECT ARTIKEL,PREIS1 FROM ART WHERE ARTIKEL='10.1001';

ARTIKEL PREIS1
------- ------
10.1001  39,99
```

Anmerkung: Session B ist wieder da. Obwohl seitens der Session B bereits eine erneute Preisänderung vorgenommen wurde, bekommen wir den alten Wert (39,99 statt 40,00), da Session B die Änderung noch nicht mit COMMIT abgeschlossen hat.

Session A:

```
UPDATE ART SET PREIS1=98.5 WHERE ARTIKEL='10.1016';
SELECT ARTIKEL,PREIS1 FROM ART WHERE ARTIKEL='10.1016';

ARTIKEL PREIS1
------- ------
10.1016  98,50
```

Anmerkung: Gesperrt wurde seitens des RDBMS nur der durch Session B veränderte Satz mit der Teilenummer 10.1001. Die Änderung eines anderen Satzes (Teilenummer 10.1016) ist daher problemlos möglich. Nach dieser Änderung hat das RDBMS allerdings auch eine Sperre für den Satz mit der Teilenummer 10.1016 eingetragen.

Session B:

```
UPDATE ART SET PREIS1=41.95 WHERE ARTIKEL='10.1001';
```

Session A:

```
UPDATE ART SET PREIS1=42.95 WHERE ARTIKEL='10.1001';
```

Oops! Und wartet! Und wartet! Und wartet!!!

Session B:

```
UPDATE ART SET PREIS1=92.75 WHERE ARTIKEL='10.1016';
```

Oops! Und wartet! Und wartet! Und wartet!!!

Session A:

Fehlermeldung: DEADLOCK

Das Datenbanksystem hat erkannt, dass zwei Sessions gegenseitig auf das Aufheben der Sperre der jeweiligen anderen Session warten. Diese Situation würde ohne Eingreifen des RDBMS in einer unendlichen Warteschleife für beide Sessions enden. Deshalb wird für eine der beiden Sessions eine Fehlermeldung erzeugt und die Aktion abgebrochen.

```
ROLLBACK;
SELECT ARTIKEL,PREIS1 FROM ART ORDER BY ARTIKEL;

ARTIKEL PREIS1
------- ------
10.1001  39,99
10.1016  99,99
25.3282   4,98
56.7954   0,40
80.0001  49,95
```

Anmerkung: Die Änderungen durch die Anweisungen
`UPDATE ART SET PREIS1=98.5 WHERE ARTIKEL='10.1016';`
`UPDATE ART SET PREIS1=42.95 WHERE ARTIKEL='10.1001';`
wurden durch den durchgeführten ROLLBACK verworfen.

Session B:

```
ROLLBACK;
SELECT ARTIKEL,PREIS1 FROM ART ORDER BY ARTIKEL;

ARTIKEL PREIS1
------- ------
10.1001  39,99
10.1016  99,99
25.3282   4,98
56.7954   0,40
80.0001  49,95
```

Anmerkung: Die Änderung durch die Anweisung
`UPDATE ART SET PREIS1=92.75 WHERE ARTIKEL='10.1016';`
wurde durch den durchgeführten ROLLBACK verworfen.

Hinweise für Softwareentwickler

Für das Zusammenfassen von Sql-Anweisungen zu einer Transaktion gilt: So viel wie nötig, so wenig wie möglich.

Transaktionen dürfen niemals durch Benutzerdialoge unterbrochen werden. Sperren!!!

Zeitstempelverfahren

Innerhalb unserer mehrplatzfähigen Datenbankanwendung gibt es ein Formular, mit dem Adressdaten hinzugefügt, geändert oder gelöscht werden können. Diese Daten sind in nachfolgender Tabelle gespeichert.

```
CREATE TABLE ADR (PERSON   NUMBER(6,0) NOT NULL,  /* Person */
                  VORNAME  VARCHAR2(40) NOT NULL, /* Vorname */
                  FAMNAME  VARCHAR2(40) NOT NULL, /* Name */
                  STRASSE  VARCHAR2(40) NOT NULL, /* Straße */
                  PLZORT   VARCHAR2(40) NOT NULL, /* Plz/Ort */
                  PRIMARY KEY (PERSON));
```

Nach Eingabe der Personennummer und Verlassen des Feldes Person (z.B. mittels Tabulatortaste oder durch Anklicken eines anderen Objekts mit der Maus) wird eine Ereignisbehandlungsmethode aktiviert, aus der die folgende Sql-Select-Anweisung über Datenbankzugriffsobjekte ausgeführt führt:

```
SELECT VORNAME,FAMNAME,STRASSE,PLZORT FROM ADR
INTO :VORNAME,:FAMNAME,:STRASSE,:PLZORT WHERE PERSON=:PERSON;
```

Anmerkung: Die mit : gegenzeichneten Parameternamen repräsentieren den Inhalt der Eingabefelder.
D.h.: Nach Ausführung der SELECT-Anweisung sind die Tabellendaten in den Feldern des Formulars sichtbar.

Annahme

Benutzer A wird informiert, dass sich die Postleitzahl bei Person 1 von 33604 auf 33608 geändert hat.
Benutzer B wird informiert, dass sich die Hausnummer bei Person 1 von 17 auf 2 geändert hat.
Benutzer A und B verfügen nicht über die Informationen des jeweils Anderen.

In der Ereignisbehandlungsmethode für das Betätigen der Schaltfläche Speichern werden folgende Anweisungen unter Verwendung von Datenbankzugriffsobjekten ausgeführt:

```
UPDATE ADR SET VORNAME=:VORNAME,FAMNAME=:FAMNAME,STRASSE=:STRASSE
PLZORT=:PLZORT WHERE PERSON=:PERSON;
COMMIT;
```

Ablauf

Benutzer A speichert die Daten.
Benutzer B speichert die Daten.

Herzlichen Glückwunsch, Benutzer B.
Sie haben gewonnen!

Leider gibt es auch einen Verlierer. Die Änderung von Benutzer A (Postleitzahl 33608) wurde durch Benutzer B unbeabsichtigt mit der veralteten Postleitzahl überschrieben.

Aktueller Inhalt der Adresstabelle:

PERSON	VORNAME	FAMNAME	STRASSE	PLZORT
1	Fritz	Müller	Waldweg 2	33604 Bielefeld
...				

Das RDBMS kann nicht unterscheiden, ob eine Datenmodifikation sinnvoll ist oder nicht. Das unbeabsichtige Überschreiben bereits aktualisierter Werte durch veraltete Werte kann das RDBMS deshalb nicht erkennen.

Die geschilderte Problematik tritt bei allen Anwendungen auf, die gelesene Dateninhalte in Programmvariablen bzw. Fensterobjekten zwischenspeichern und Änderungsanweisungen mit den zwischengespeicherten Werten durchführen (Schätzwert: 99 % aller Datenbankanwendungen).

Die geschilderten Probleme können durch Einsatz des Zeitstempelverfahrens wirksam vermieden werden

Voraussetzungen

1. Die Tabellen werden um eine Spalte zur Speicherung des aktuellen Datums und der aktuellen Uhrzeit des Datenbankservers zum Zeitpunkt der Durchführung von Insert- bzw. Update-Anweisungen erweitert.

   Beispiel (Oracle):

   ```
   ALTER TABLE ADR ADD ZSTEMPEL DATE;
   UPDATE ADR SET ZSTEMPEL=SYSDATE;
   ALTER TABLE ADR MODIFY ZSTEMPEL DATE NOT NULL;
   COMMIT;
   ```

   Anmerkungen: Die Update-Anweisung belegt alle Sätze der Tabelle ADR mit dem aktuellen Datum und der aktuellen Uhrzeit des Datenbankservers (Oracle: SYSDATE)

2. Die Select-Klausel zur Selektion der Daten eines Tabellensatzes wird um die Zeitstempelspalte erweitert.

   ```
   SELECT VORNAME,FAMNAME,STRASSE,PLZORT,ZSTEMPEL FROM ADR
   INTO :VORNAME,:FAMNAME,:STRASSE,:PLZORT,:ZSTEMPEL WHERE PERSON=:PERSON;
   ```

   Anmerkungen: Der Inhalt der Zeitstempelspalte (:ZSTEMPEL) wird nicht im Formular angezeigt, sondern in einer Programm- bzw. Formularvariablen zwischengespeichert.

3. Bei Insert-Anweisungen wird die Zeitstempelspalte mit dem aktuellen Datum und der aktuellen Uhrzeit des Datenbankservers (Oracle: SYSDATE) versorgt.

   Beispiel (Oracle):

   ```
   INSERT INTO ADR VALUES(:PERSON,:VORNAME,:FAMNAME,:STRASSE,:PLZORT,SYSDATE);
   ```

4. Bei Update-Anweisungen wird die Zeitstempelspalte ebenfalls mit dem aktuellen Datum und der aktuellen Uhrzeit des Datenbankservers versorgt. Die Update-Anweisung darf aber zusätzlich nur dann ausgeführt werden, wenn der Zeitstempel des aktuellen Satzes in der Datenbank dem vorher beim SELECT gelesenen und in einer Programmvariablen zwischengespeicherten Zeitstempel entspricht. Zu diesem Zweck nehmen wir eine zusätzliche Zeitstempelbedingung in die WHERE-Klausel auf.

   Beispiel (Oracle):

   ```
   UPDATE ADR SET VORNAME=:VORNAME,FAMNAME=:FAMNAME,STRASSE=:STRASSE
   PLZORT=:PLZORT,ZSTEMPEL=SYSDATE WHERE PERSON=:PERSON AND ZSTEMPEL=:ZSTEMPEL;
   ```

Anmerkungen: ...,ZSTEMPEL=SYSDATE
Zeitstempelspalte mit dem aktuellen Datum und der aktuellen Uhrzeit des Datenbankservers belegen.
... AND ZSTEMPEL=:ZSTEMPEL
Inhalt der Zeitstempelspalte des zu verändernden Datensatzes mit dem beim SELECT gelesenen und in einer Programmvariablen zwischengespeicherten Zeitstempelinhalt vergleichen, um sicherzustellen, dass die Update-Anweisung nur dann ausgeführt wird, wenn die Vergleichsbedingung zutrifft, d.h. der Satz nicht in der Zeit zwischen SELECT und UPDATE anderweitig verändert wurde.

5. Eine Delete-Anweisung darf nur dann ausgeführt werden, wenn der Zeitstempel des aktuellen Satzes in der Datenbank mit dem vorab beim SELECT gelesenen und in einer Programmvariablen zwischengespeicherten Zeitstempel übereinstimmt. Zu diesem Zweck nehmen wir, wie bei der UPDATE-Anweisung, eine zusätzliche Zeitstempelbedingung in die WHERE-Klausel auf.

   Beispiel (Oracle):

```
DELETE FROM ADR WHERE PERSON=:PERSON AND ZSTEMPEL=:ZSTEMPEL;
```

Anmerkungen

- Das Verfahren ist nur dann sinnvoll einsetzbar, wenn sich die Anweisungen SELECT, INSERT, UPDATE und DELETE auf genau einen Datensatz beziehen (WHERE-Klausel mit PRIMARY KEY)
- Falls nach einer Update- bzw. Delete-Anweisung festgestellt wird, dass der Satz nicht geändert bzw. gelöscht wurde, ist der Grund hierfür in der nicht zutreffenden WHERE-Klausel zu suchen, d.h. die Daten wurden zwischenzeitlich (Zeitspanne zwischen Lesen und Modifizieren) anderweitig geändert oder gelöscht.
  In diesem Fall sollte der Benutzer eine aussagekräftige Fehlermeldung erhalten und zusätzlich einen Hinweis bekommen, dass dieses Problem durch erneutes Einlesen der Daten in das Formular beseitigt werden kann.

Szenario unter Einsatz des Zeitstempelverfahrens

Inhalt der Tabelle ADR:

PERSON	VORNAME	FAMNAME	STRASSE	PLZORT	ZSTEMPEL
1	Fritz	Müller	Waldweg 17	33604 Bielefeld	15.03.02 11:33:17
...					

Benutzer A und B lesen die Daten in das Formular ein.
Der gelesene Zeitstempel (15.03.02 11:33:17) wird in einer Programmvariablen zwischengespeichert.

Benutzer A ändert die Postleitzahl von 33604 auf 33608 und betätigt die Schaltfläche Speichern.
Der Zeitstempel des Datensatzes in der Datenbank entspricht dem zwischengespeicherten Zeitstempel, so dass die Update-Anweisung für den entsprechenden Datensatz ausgeführt wird. Danach enthält die Zeitstempelspalte des Datensatzes das Datum und die Uhrzeit des Datenbankservers zum Zeitpunkt der Modifikation.

Inhalt der Tabelle ADR nach erfolgtem Update:

PERSON	VORNAME	FAMNAME	STRASSE	PLZORT	ZSTEMPEL
1	Fritz	Müller	Waldweg 17	33608 Bielefeld	15.03.02 11:33:58
...					

Benutzer B ändert die Hausnummer von 17 auf 2 und betätigt die Schaltfläche Speichern.
Der in der Programmvariablen zwischengespeicherte Zeitstempel (15.03.02 11:33:17) entspricht nicht mehr dem aktuellen Zeitstempel des Datensatzes (15.03.02 11:33:58), so dass die WHERE-Klausel der UPDATE-Anweisung für den Datensatz nicht zutrifft und die Modifikation deshalb nicht ausgeführt wird.

Benutzer B wird mittels einer Fehlermeldung über die aufgetretene Situation informiert:

Nach erneutem Einlesen der Formulardaten durch Benutzer B werden die Änderungen des Benutzers A sichtbar. Der aktuelle Zeitstempel des Datensatzes (15.03.02 11:33:58) wird erneut zwischengespeichert.

Benutzer B ändert die Hausnummer von 17 auf 2 und betätigt die Schaltfläche Speichern.
Der Zeitstempel des Datensatzes in der Datenbank entspricht dem zwischengespeicherten Zeitstempel, so dass die Update-Anweisung für den entsprechenden Datensatz ausgeführt wird. Danach enthält die Zeitstempelspalte des Datensatzes Datum und Uhrzeit des Datenbankservers zum Zeitpunkt der Änderung.

Inhalt der Tabelle ADR nach erfolgtem Update:

PERSON	VORNAME	FAMNAME	STRASSE	PLZORT	ZSTEMPEL
1	Fritz	Müller	Waldweg 2	33608 Bielefeld	15.03.02 11:34:45
...					

Fazit

Das unbeabsichtigtes Überschreiben von aktuelleren Daten mit zwischengespeicherten veralteten Daten aus Formularobjekten bzw. Programmvariablen kann durch den Einsatz des Zeitstempelverfahrens wirksam verhindert werden.