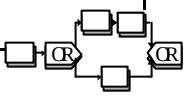
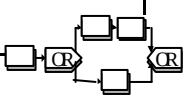
Kapitel 4: Fehlerbehandlung in Workflow-Systemen

- Klassifizierung von Fehlersituationen
- Das ACID-Transaktionsmodell
  - Charakteristika
  - Beschränkungen
- Erweiterte Transaktionsmodelle



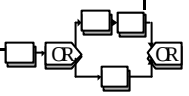
Ausgangssituation

- Workflows
  - sind langlebige Prozesse (Dauer von Tagen, Wochen, Monaten etc.)
  - sind datenintensive Prozesse
  - operieren auf verteilten, heterogenen Datenbeständen (u.U. geographisch verteilt)
  - integrieren manuelle und automatische Arbeitsschritte (mit Anwendungsprogrammen)
- Im Falle von „Fehlersituationen“ erstrebenswert
  - Versetzen der betroffenen Workflow-Instanzen in konsistenten Zustand, der Fortführung nach Behebung des Fehlers ermöglicht
  - Konsistenter Zustand aller Datenquellen und Applikationen, auf die Workflow-Instanz zum Zeitpunkt des Fehlers zugegriffen hat
  - Möglichst geringer Verlust bzgl. bereits durchgeführter Arbeitsschritte
- Sperrung von Datenobjekten durch Workflows sollte auf Minimum reduziert werden
- Zielsetzung: Fehlerbehandlung durch Workflow-orientiertes Transaktionsmanagement



Klassifikation von Fehlersituationen

	Fehlertyp	Beispiele
1.	**Gerätefehler**	Ausfall von Platten, auf denen Workflow-relevante Daten liegen
		...
2.	**Systemfehler**	Absturz von Betriebssystemkomponenten (inkl. Netzwerk)
		Absturz der Workflow-Engine
		Absturz von Anwendungsprogrammen
		...
3.	**Transaktionsfehler**	Division durch Null (durch Anwendungprogramm)
		Workflow-System oder Anwendungsprogram verletzt Datenintegritäts-Constraints
		...
4.	**Logische Fehler (→ Kap. 5)**	Kontrollfluß des Workflows nicht mehr adäquat
		Datenfluß des Workflows nicht mehr adäquat
		Zeitliche Vorgaben (z.B. Deadlines) nicht mehr adäquat
		...



Fehlerbehandlung in „klassischen“ Datenbanksystemen

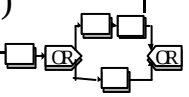
- Transaktions-Management

  „Eine Transaktion ist die Ausführung einer Folge zusammengehöriger Operationen, welche eine Datenbank von einem konsistenten in einen (neuen) konsistenten Zustand überführen“ (Gray 73, Härder & Reuter 83, ...)

- Behandlung der Fehlertypen 1-3 sowie Schutz vor konkurrierenden Zugriffen auf Datenbank-Objekte

- ACID-Prinzip
  - **A**tomicity: Alles oder Nichts’-Eigenschaft
  - **C**onsistency: eine erfolgreiche Transaktion erhält die DB-Konsistenz (Menge der definierten Integritätsbedingungen)
  - **I**solation: alle Aktionen innerhalb einer Transaktion müssen vor parallel ablaufenden Transaktionen verborgen werden (logischer Einbenutzerbetrieb)
  - **D**urability: Überleben von Änderungen erfolgreich beendeter Transaktionen trotz beliebiger (erwarteter) Fehler garantieren (*Persistenz*).

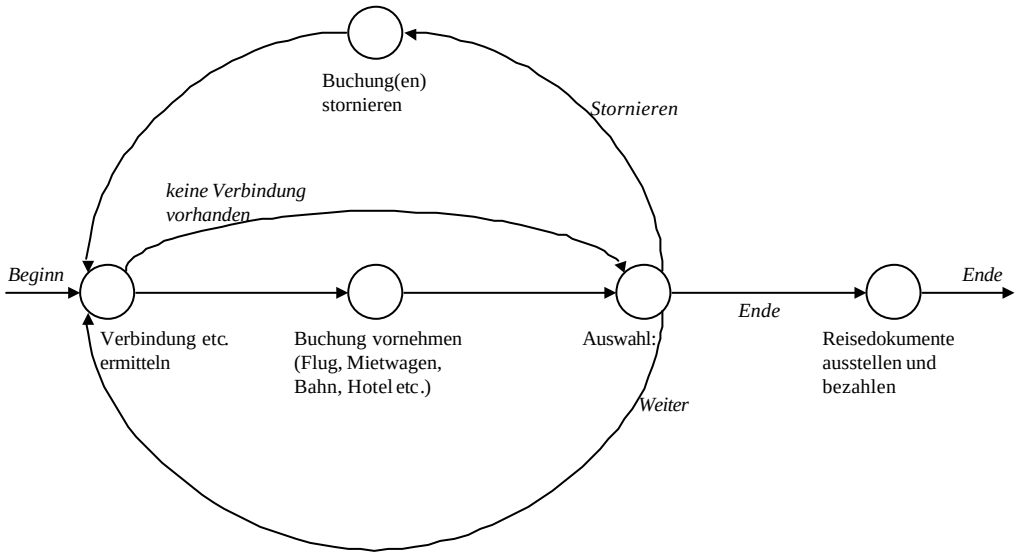
- Entlastung des Anwendungsprogrammierers von Aspekten der Fehlerbehandlung (failure transparency) und der Nebenläufigkeit (concurrency transparency)



Das ACID-Prinzip: Annahmen

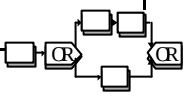
- Basisannahmen
  - Kurze Lebensdauer der transaktionsgeschützten Zugriffe
  - Pro Transaktion nur wenige DB-Objekt betroffen
  - Binnenstruktur der Transaktion muß nach außen hin nicht sichtbar sein
  - Keine Inter-Transaktionsabhängigkeiten
- Konsequenzen
  - Rollback tolerabel
  - Strikte Isolierung von Datenbank-Objekten tolerabel

Fall-Beispiel: Buchen einer Geschäftsreise



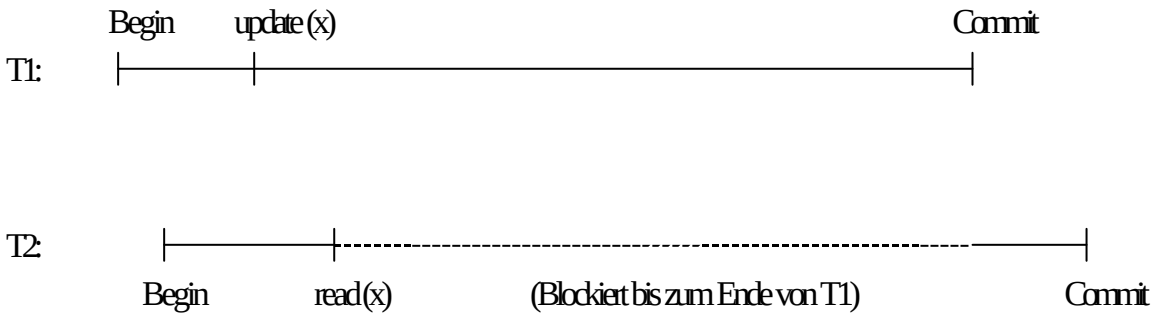
Atomarität und Workflows

- Zu große Verluste bei Rollback von langlebigen Workflows
- Rücksetzen sollte auf unmittelbar betroffene Teilschritte beschränkt sein
- Workflows modellieren oft Prozesse, die nicht rücksetzbar sind (*Mahnbrief verschicken*, *Chemotherapie applizieren*) Ausweg: Logisches Rollback
- Idealfall: Workflow kann exakt an der Stelle fortgeführt werden, an der er bei einem Fehler unterbrochen wurde (*Forward Recovery*)
- Keine zuverlässigen Fehlermeldungen aus ACID-Transaktionen heraus möglich
  - Versenden der Fehlernachricht *mit* Transaktionsschutz → Keine Auslieferung bei Rollback
  - Versenden der Fehlernachricht *ohne* Transaktionsschutz → Auslieferung nicht garantiert
- Binnenstruktur und Kontrollfluß einer ACID-Transaktion nach außen unsichtbar → Migration nicht möglich
- Kontrolle über Transaktionsgrenzen hinaus kaum möglich, z.B.: *Starte T_1 genau dann wenn T_2 mit Abort endet*

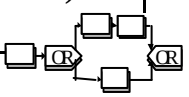


Isolation und Workflows (1)

- Strikte Isolation bei langlebigen Transaktionen zu restriktiv (und oft unnötig)

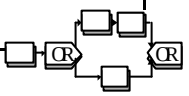


- Sperrverfahren und langlebige Transaktionen: Wartende Transaktionen blockieren ihrerseits viele andere Objekte, die zur Blockade wiederum anderer Transaktionen führen → Drastische Reduktion der Systemleistung, Gefahr einer Verklemmung
- Gray et al. (1981) : Wahrscheinlichkeit für eine Verklemmung steigt mit der vierten Potenz der Transaktionsdauer (gemessen an der Anzahl der berührten Objekte)



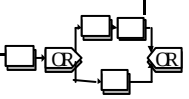
Isolation und Workflows (2)

- Aber: Aufheben von Sperren *nach* Beendigung des Objektzugriffs, aber *vor* Ende einer Transaktion *T* kann zu Dirty-Read- und Dirty-Write-Effekten führen
- Beispiel *Reise-Stornierung*: Falls Reisebuchungs-Einträge in Datenbank frühzeitig (vor COMMIT) freigegeben werden, können bei Scheitern der Transaktion nicht einfach die Datenbank-Werte vor Transaktionsbeginn zurückgeschrieben werden, da bereits andere Transaktionen darauf zugegriffen haben können
- Ausführung der logisch inversen Operation nötig (Stornierung der Buchung + evtl. Stornierungsgebühr)
- Kaskadierender Aufruf der logischen Inversen



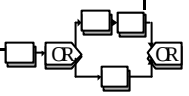
Konsistenz und Workflows

- Implizite Annahme des ACID-Modells: Datenbank ist in einem konsistenten Zustand, wenn eine Transaktion ihr COMMIT ausführt
- Konsistenz gekoppelt an Mächtigkeit der Integritätsregeln des DBMS
  - Temporale Integritätsregeln oft nur ungenügend unterstützt (z.B. Leukozytenwert eines Patienten kann innerhalb von 2 Tagen nicht um 2000 fallen, wohl aber in 4 Tagen)
- Aus Performance-Gründen werden Integritätsregeln oft *nicht* überprüft
- Forderung bzgl. Workflow-orientierter Transaktionsmodelle: Transaktionsbezogene Konsistenzbedingungen



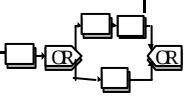
Das ACID-Prinzip: Zusammenfassung

Kontrollflußmodellierung	Atomare Zustandsübergänge; keine Binnenstruktur und keine transaktionsübergreifenden Kontrollstrukturen.
Ablaufsteuerung	Im Transaktionsprogramm: Nur Commit / Rollback Work. Von außen keine Eingriffsmöglichkeit.
Persistenz von (Zwischen-) Ergebnissen	Erst nach erfolgreichem Commit. Keine stabile Verwaltung lokaler Zustandsinformationen.
Begrenzung des Rücksetzbereiches	Zurücksetzen nur auf den Zustand vor Beginn der Transaktion.
Synchronisation	Serialisierbarkeit strikt isolierter Transaktionen. Keine frühzeitige Freigabe von Änderungen, keine anwendungsdefinierten Isolationsbedingungen
Konsistenzsicherung	implizit: Transaktionen sind per Definition korrekt. Keine anwendungsspezifische Lockerung bzw. Verschärfung der globalen Integritätsbedingungen
Behandlung von Konflikten	implizit: Warten, Abbruch.



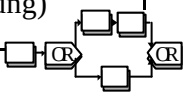
Erweiterungen des klassischen Transaktionskonzeptes

- Transaktionen mit Rücksetzpunkten
- Mini-Batch
- Geschachtelte Transaktionen
- Mehrebenen-Transaktionen
- Sagas
- Forward Recovery (Contract-Modell)
- Sphärenmodelle



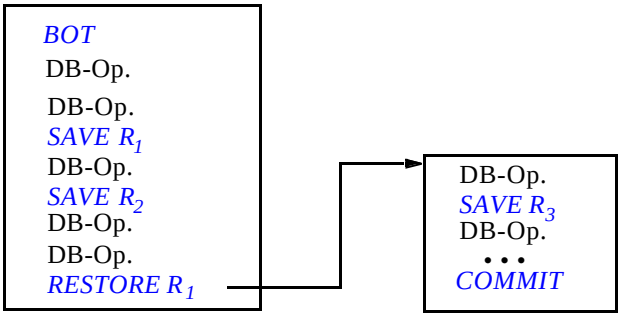
Transaktionen mit Rücksetzpunkten (1)

- Abschwächung der Atomaritätszusicherung
- Partielle Zurücksetzen von Transaktionen durch Zurückgehen auf Rücksetzpunkt
  - bereits von verschiedenen DBS unterstützt
  - Im SQL3-Standard vorgesehen
- Operationen SAVE und RESTORE
- SAVE-Operation
  - bewirkt Setzen eines Rücksetzpunktes und damit Sicherung des erreichten Transaktionszustandes
  - Sicherung der bis dahin durchgeführten DB-Änderungen, inkl. Informationen über die vorliegenden Sperren der Transaktion, Cursor-Positionen usw.
- Im Fehlerfall RESTORE-Operation
  - Anstelle eines ROLLBACK Zurückgehen auf einen Rücksetzpunkt R
  - *Partielles Undo*, bei dem alle nach R erfolgten Änderungen in umgekehrter Reihenfolge zurückgesetzt werden
  - Änderungen, welche vor R ausgeführt wurden, haben jedoch Bestand (keine Wiederholung)

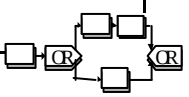


Transaktionen mit Rücksetzpunkten (2)

- Beispiel
  - Transaktion setzt zwei Rücksetzpunkte und geht mit RESTORE-Operation auf den ersten davon (R_1) zurück
  - Änderungen der vor R_1 ausgeführten Operationen weiterhin wirksam
  - Zurücksetzen der danach durchgeführten Operationen
  - Anwendung wird vom Rücksetzpunkt aus „nach vorne“ fortgesetzt, wobei i. allg. einige der zurückgesetzten DB-Operationen erneut ausgeführt werden

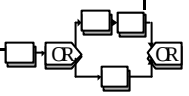


- Verantwortung der Anwendung, daß sich zur Ausführung von RESTORE die Programmvariablen in korrektem Zustand befinden



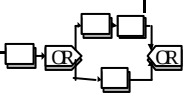
Transaktionen mit Rücksetzpunkten: Bewertung

- Geeignet für langlebige Prozesse (Workflows) durch Abschwächung der Atomarität
- Aber: Sicherung des Zwischenstands nur seitens des DBS, nicht jedoch auf Anwendungsseite
  - Daher nur Unterstützung hinsichtlich Transaktionsfehlern, nicht jedoch gegenüber Systemfehlern
  - Bei Rechnerausfall zwar Rücksetzung auf zuletzt erreichten Rücksetzpunkt möglich
  - Jedoch keinerlei Information über den zugehörigen Status des jeweiligen Anwendungsprogramms verfügbar (Programmzähler, Inhalt lokaler Variablen usw.)
  - Somit Fortführung der Transaktion ausgehend vom Rücksetzpunkt nicht möglich
- Reduzierung des Arbeitsverlustes bei Rechnerausfällen nur durch *persistente Rücksetzpunkte*
  - Koordinierte Sicherung für Anwendungs- und Datenbankobjekte über 2-Phasen-Commit-Protokoll
  - Neben DBS auch Laufzeitsystem für Anwendungsprogramme als transaktionsgeschützter Ressourcen-Manager als realisiert, der analog koordinierte Rücksetzpunkte durchführt
- Keine Lösung der Isolationsproblematik



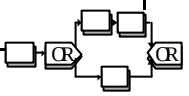
Mini-Batch

- Längere DB-Anwendung wird in kleine Verarbeitungsschritte $S_1,...,S_n$ zerlegt
- Vor Commit vermerkt S_i transaktionsgeschützt, wie weit Gesamtverarbeitung gekommen ist (also bis Ende von S_i)
- Bei Fehler während S_{i+1} wird aktueller Programmschritt zurückgefahren, nach Wiederanlauf letzter erfolgreicher Schritt S_i gesucht und mit S_{i+1} fortgefahren
- Anwendungsszenario: Transaktionsprogramme für Zinsgutschriften im Batch-Modus (Zerlegung in Schritte mit je 1000 Konten)
- Forward Recovery, aber in der Verantwortung des Anwendungsprogrammierers; nur lineare Abläufe ohne parallele Verarbeitungsschritte



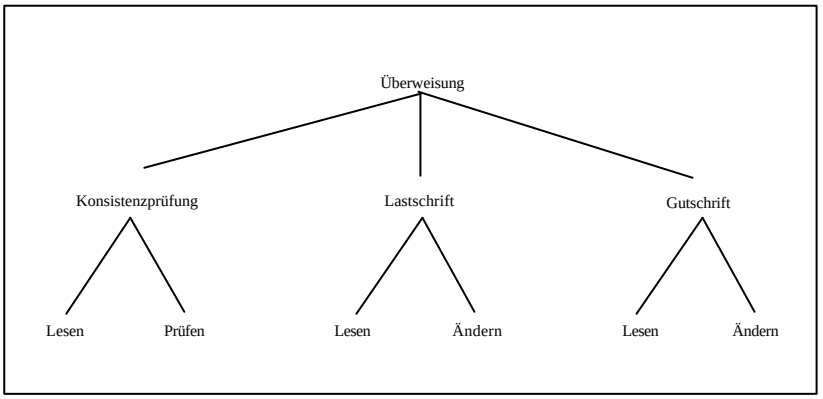
Mini-Batch: Beispiel

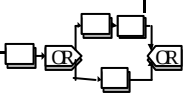
```
EXEC SQL
        SELECT letzter_Verarbeitungsschritt INTO :letzter_Verarbeitungsschritt
        FROM    verarbeitungskontext;
/*    Damit kann die Zinsgutschrift fortgesetzt werden:                    */
      i_ letzter_Verarbietungsschrift;
      WHILE( i < n ) DO
           { EXEC SQL UPDATE konten       /* Verarbeitungsschritt durchfuehren */
                      SET konto_stand _
                            konto_stand * (1+zinssatz)
                      WHERE kto_nr BETWEEN   (i*1000)+1
                                       AND          (i+1)*1000;
             EXEC SQL UPDATE verarbeitungskontext        /* Durchfuehrung*/
                      SET letzter_Verarbeitungsschritt_ i+1;/* v. Schritt i+1  */
             EXEC SQL COMMIT WORK;                       /* protokollieren  */
             i= i+1;}
/* Nachdem alle n Verarbeitungsschritte durchgefuehrt sind, kann die        */
/* Kontextrelation geloescht werden:                                        */
   EXEC SQL DROP TABLE verarbeitungskontext;
   return;
```



Geschachtelte Transaktionen (1)

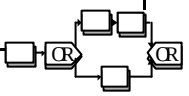
- Transaktion *T* besteht aus mehreren Subtransaktionen, die wiederum aus Subtransaktionen bestehen können
- Jede Subtransaktion atomar und separat rücksetzbar → bei Fehler nicht gesamte Arbeit verloren
- Bei Scheitern einer Subtransaktion:
  - Wiederholung (u.U. mit geänderten Parametern)
  - Aufruf alternativer Subtransaktion
  - Ignorieren des Fehlers
  - Zurücksetzen der Vater-Transaktion





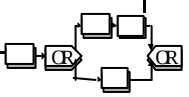
Geschachtelte Transaktionen (2)

- Bessere Modellierungsmöglichkeiten (bei verteilten und parallelen Anwendungen)
- Allgemeinerer Ansatz als Savepoints
  - Partielles Zurücksetzen innerhalb beliebiger Hierachien von Sub-Transaktionen möglich
  - Savepoints: Nur Sequenzen von Teiltransaktionen, die einzeln zurückgesetzt werden können
- Übliche Regeln der Zusammenarbeit in geschachtelten Transaktionen
  - *Rücksetzregel*: Wenn (Sub-) Transaktion auf irgendeiner Schachtelungsebene zurückgesetzt wird, werden alle ihre Sub-Transaktionen ebenso zurückgesetzt Rücksetzung einer Sub-Transaktion führt aber i.allg. nicht zur Rücksetzung der Vater-Transaktion.
  - *Commit-Regel*: (Lokales) Commit einer Sub-Transaktion macht Ergebnisse der Vater-Transaktion zugänglich. Endgültiges Commit einer Sub-Transaktion erfolgt dann und nur dann, wenn für alle Vorfahren bis zur Top-Level-Transaktion das endgültige Commit erfolgreich verläuft. Commit der Top-Level-Transaktion bewirkt Dauerhaftigkeit aller Änderungen der Transaktion.
  - *Sichtbarkeits-Regel*: Alle Änderungen einer Sub-Transaktion werden bei ihrem Commit für Vater-Transaktion sichtbar. Alle Objekte, die einer Vater-Transaktion sichtbar gemacht wurden, können den Sub-Transaktionen zugänglich gemacht werden. Änderungen einer Sub-Transaktion sind für gleichzeitig aktive Geschwister-Transaktionen nicht sichtbar.



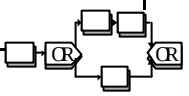
Geschachtelte Transaktionen (3)

- Commit- und Rücksetzregel implizieren Atomarität für Sub-Transaktionen
- Sichtbarkeitsregel: Eigenschaft I (Isolation) bezüglich parallel laufender Sub-Transaktionen
- Aufgrund Abhängigkeit der Sub-Transaktionen zu ihren Vorgängern in der Transaktionshierarchie:
  - keine Dauerhaftigkeit
  - Änderungen einer Sub-Transaktion können auch nach dem lokalen Commit noch zurückgesetzt werden
- Eigenschaft C (Consistency) kann höchstens lokal bezüglich der von der Sub-Transaktion realisierten Funktion erreicht werden
- Bezüglich der Gesamttransaktion wird erst mit Abschluß der Top-Level-Transaktion die Konsistenz der Datenbank sichergestellt
- Somit gelten für Sub-Transaktionen i. allg. nur die Eigenschaften A und I



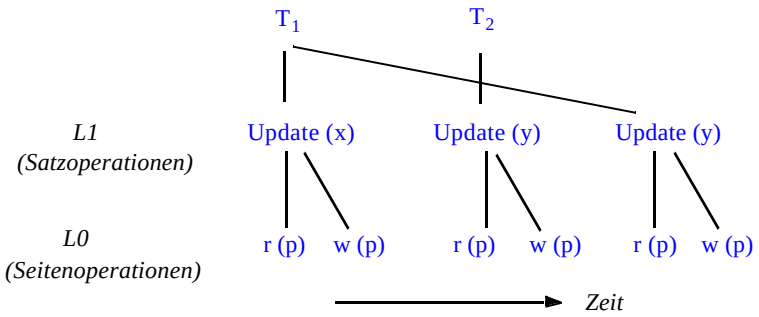
Offen geschachtelte Transaktionen

- Freigabe der lokalen Änderungen (für *unabhängige* Transaktionen) nach Commit der Subtransaktion
- Bei Scheitern der Gesamt-Transaktion “Rollback” nur durch Ausführung der logischen Inversen (Kompensationen) der bereits durchgeführten Subtransaktionen (Beispiel: Lastschrift - Gutschrift) möglich
- Kompensationsschritte können i.allg. nicht automatisch aus der Original-Transaktion abgeleitet werden, sondern müssen pro Transaktion angegeben werden → keine Unabhängigkeit von der Anwendung mehr gegeben
- Kaskadierungsproblem: Bei Kompensations-Rücksetzung einer offen geschachtelten Transaktion mit bereits freigegebenen lokalen Objekt-Änderungen sind andere Transaktionen möglicherweise ungültig und müssen ebenfalls zurückgesetzt bzw. kompensiert werden (Domino-Effekt; kaskadierende Kompensationen)

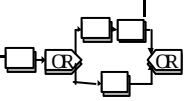


Mehrebenen-Transaktionen

- Sonderfall der Schachtelung von Trans. und ihrer Operationen längs der Abbildungshierarchie einer Schichtenarchitektur
- Feste Anzahl von Schichten mit bestimmten Operationen; Operationen der Ebene i werden durch Operationen auf i-1 realisiert

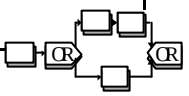


- Ausführung jeder Operation atomar im Rahmen einer Subtransaktion
- Sperren werden nur für die Operationen (Subtransaktionen) auf oberster Ebene bis zum Transaktionsende gehalten
- Für darunterliegende Schichten vorzeitige Sperrfreigabe am Ende der direkt übergeordneten Subtransaktion, unabhängig vom Ende der gesamten Transaktion
- Serialisierbarkeit auf höheren Ebenen gewahrt



Sagas

- Formalisierung verketteter Transaktionen
- Saga ist Folge von n sequentiell verknüpften Teiltransaktionen $T_1,...,T_n$ mit assoziierten Kompensationsschritten $C_1,...,C_n$
- Unabhängiges Commit und Freigabe der Änderungen durch jedes T_i
- Normalfall: $T_1,..., T_{n-1},T_n$
- Fehlerfall: $T_1,..., T_k$ [Error] C_k $C_{k-1},..,C_1$
- Erweiterung: Geschachtelte Sagas: T_k kann aus Subsagas bestehen
- Geschachtelte Sagas Spezialfall offen geschachtelter Transaktionen, daher dieselben Defizite

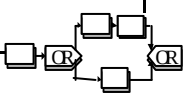


Ereignisorientierte Verknüpfung von Transaktionen

- Kontrollfluß der (Teil-)Transaktionen wird nicht in globaler Ablaufbeschreibung spezifiziert, sondern durch lokale Verknüpfungsregeln (ECA-Regeln) (Ereignisorientierte Kontrollfluß-Modellierung)

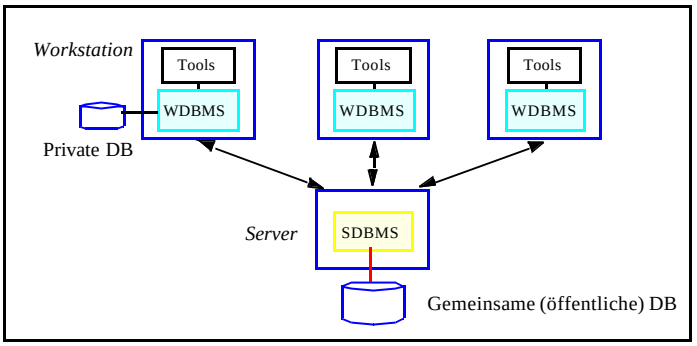
Ereignis (e)	Der Bestandszähler eines Artikels A im Lager wird reduziert.
Bedingung (c)	FALLS (Bestand(A) ≤ Minimum) UND (Keine offene Bestellung für Artikel A)
Aktion (a)	Nachbestellung für A veranlassen

- 3 prinzipielle Kopplungsmöglichkeiten bzgl. zwischen ereignisauslösender und bedingungsauswertender Transaktion sowie bedingungsauswertender und aktionsausführender Transaktion
  - Unmittelbare (immediate) EC-Kopplung: Bedingung wird sofort und in derselben Transaktion ausgewertet, die Ereignis auslöste. Unmittelbare (immediate) CA-Kopplung: Aktion wird sofort und in derselben Transaktion ausgeführt, die Bedingung auswertete
  - Verzögerte (deferred) Kopplung: C- bzw. A-Transaktion erfolgt erst am Ende der aktuellen Top-Level-Transaktion, unmittelbar vor deren Commit-Verarbeitung
  - Entkoppelter (detached) Modus: Unabhängige Bearbeitung von C bzw. A in einer separaten und zur aktuellen Transaktion parallelen Top-Level-Transaktion. Spezifikation einer kausalen Commit-Abhängigkeit zwischen auslösender und ausgelöster Transaktion möglich: Abbruch einer auslösenden Transkation zieht Rücksetzen aller entkoppelt aktivierten Transaktionen nach sich, aber nicht umgekehrt
- Bewertung: Entkoppelter und kausal unabhängiger Modus entspricht offen geschachtelten Transaktionen, alle anderen Modi entsprechen geschlossen geschachtelten Transaktionen



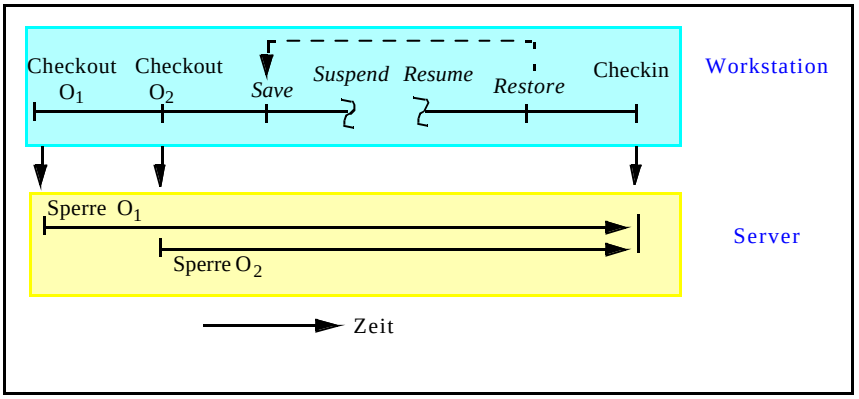
Transaktionsmodelle für Entwurfsumgebungen (1)

- Spezifische Anforderungen im datenbank-gestützten CAD- und CASE-Bereich
  - Komplexe Datenobjekte auf der Basis objektorientierter oder objektrelationaler Modelle
  - Versionsmanagement erforderlich
  - Langlebige Prozesse (ACID-Transaktionen nicht sinnvoll)
  - Gruppenarbeit → statt strikter Isolation kontrollierte Nebenläufigkeit bzgl. Objektzugriffen nötig
- Effiziente Realisierung durch Workstation/Server-Architekturen
- Checkout/Checkin-Modell
  - Benötigtes Objekt wird zu Beginn eines Entwurfsschrittes vom Server angefordert und in privater Datenbank abgelegt (Checkout)
  - Server setzt für die Dauer des Checkouts Sperre auf Objekt
  - Nach Beendigung des Entwurfsschrittes Checkin in Server-Datenbank und Freigabe der Sperren
  - Abschwächung der Isolation durch versionsbezogene Sperren



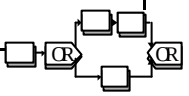
Transaktionsmodelle für Entwurfsumgebungen (2)

- Vermeidung von hohen Arbeitsverlusten durch persistente Rücksetzpunkte
  - Benutzerkontrolliert unter Berücksichtigung der Design-Semantik
  - Verwendung von Save- und Restore-Operationen
  - Zusätzliche Suspend- und Resume-Operationen zur Unterbrechung des Entwurfsvorgangs (z.B. bei Pause oder Feierabend)
  - Bei Verletzung von Integritätsbedingungen nicht automatisch Rollback zum letzten Rücksetzpunkt, sonderen benutzerkontrollierte Fehlerbehandlung (z.B. für Durchführung konsistenzerhaltender Objekt-Änderungen)



Transaktionsmodelle für Entwurfsumgebungen (3)

- Isolationsmodell zur Unterstützung der Kooperation
  - Mehrstufiges Checkout/Checkin der Objekte: Zunächst Checkin eines Objektes in Team-Datenbank, danach Checkin von Teilobjekten in Privat-Datenbank
  - Explizites Verfügbarmachen und Zurückgeben von (Teil-)Objekten durch Transaktionen über grant / return Protokoll
  - Transaktion des Empfängers wird dabei zu einer abhängigen Subtransaktion der abgebenden Transaktion
  - Schachtelung entsprechend der Objekt-Ausleihbeziehungen
  - Ausgeliehenes Objekten wird in serialisierbaren Transaktion bearbeitet und dann an Ausleiher zurückgegeben
  - Wenn Transaktion keine Objekte des Vorgängers mehr besitzt und alle an ihre Nachfolger weitergegebene Objekt zurückbekommen hat, wird sie wieder zu einer unabhängigen Top-Level-Transaktion



Diskussion bisheriger Erweiterungsvorschläge

- Aufhebung der strikten Atomarität durch partielles Rücksetzen
- Aber: Erforderliche Sicherung des Zwischenstands i. allg. nur seitens des DBS, nicht jedoch auf Anwendungsseite
  - Bei Systemfehlern keinerlei Information über den zugehörigen Status des jeweiligen Anwendungsprogramms verfügbar (Programmzähler, Inhalt lokaler Variablen usw.)
  - Für Workflow-Management nicht akzeptabel
  - Stabile Kontextverwaltung erforderlich
- Eingeschränkte Möglichkeiten der Spezifikation von Rücksetzbereichen
- Unbefriedigende Lösung der Isolationsproblematik
  - Kaskadierende Kompensationen bei offen geschachtelten Transaktionen
  - Bisherige Ansätze zur Isolation *bereichs*orientiert (nach welchem Transaktionsbereich wird Sperre freigegeben?), nicht *prädikat*orientiert (was dürfen andere Transaktionen mit dem Objekt tun, was nicht?)