7. Transaktionsverarbeitung

7.1 Einführung

7.1.1 Client/Server-Modell

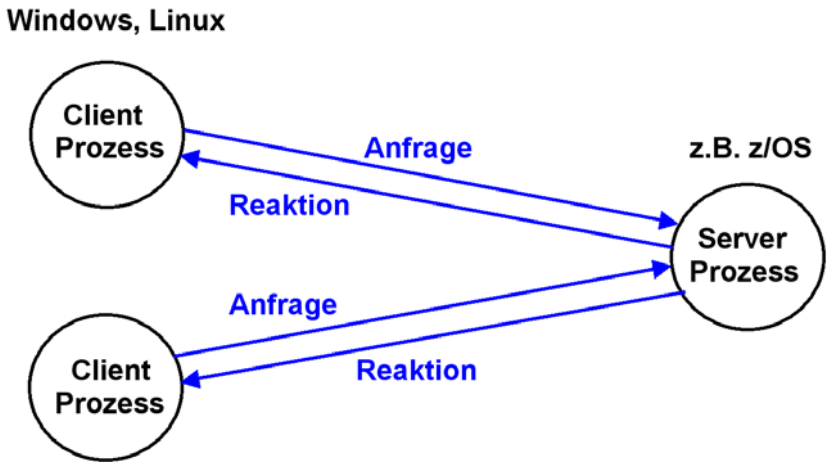


Abb. 7.1.1
Client/Server-Modell

Prozesse auf einem Klienten-Rechner nehmen die Dienstleistungen eines Servers in Anspruch. Ein Server bietet seine Dienste (Service) einer Menge a priori unbekannter Klienten (Clients) an.

Klient:	**Nutzer eines Server Dienstes**
Server:	**Rechner, der Dienst-Software ausführt**
Dienst (Service):	**Software-Instanz, die auf einem oder mehreren Server Rechnern ausgeführt wird**
Interaktionsform:	**Anfrage / Reaktion (Request/Reply)**

Ein Rechner kann gleichzeitig mehrere Serverdienste (Services) anbieten.

Der größere Teil (etwa 60-70 %) aller Anwendungen in Wirtschaft und Verwaltung läuft (interaktiv) auf einer Client/Server Konfiguration. Etwa 30-40 % laufen als Stapelverarbeitungsprozesse (batch processing)

Client/Server Systeme werden auch als „Interaktive“ Systeme, im Gegensatz zur Stapelverarbeitung, bezeichnet.

Die meisten interaktiven Anwendungen und viele Stapelverarbeitungsanwendungen werden als „Transaktionen“ ausgeführt.

7.1.2 Datenbanken

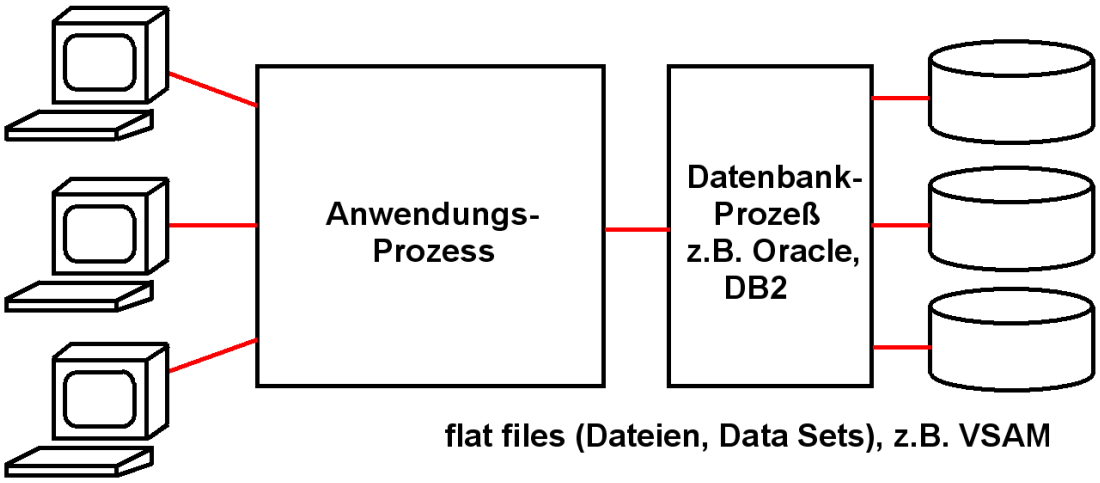
Für einfache Interaktionen eines Anwendungsprogramms mit seinen Daten werden „Flat Files“ (z.B. VSAM) eingesetzt. Unter z/OS verringern Access Methods den Programmieraufwand für den Zugriff. Access Methods werden größtenteils vom Betriebssystem Kernel ausgeführt, aber VSAM ist eng in den Benutzer- (Anwendungs-) Prozess integriert.

Komplexere Interaktionen unterstützen ein Anwendungsprogramm mit Hilfe eines getrennten „Datenbank“-Prozesses.

Eine Datenbank besteht aus einer Datenbasis (normalerweise Plattenspeicher) und Verwaltungsprogrammen (Datenbank Software), welche die Daten entsprechend den vorgegebenen Beschreibungen abspeichern, auffinden oder weitere Operationen mit den Daten durchführen.

Wenn wir im Zusammenhang mit Client/Server Systemen von einer Datenbank sprechen meinen wir in der Regel damit die Verwaltungsprogramme (Datenbank im engeren Sinne).

Ein Datenbankprozess läuft fast immer in einem eigenen virtuellen Adressenraum. Häufig sind es sogar mehrere virtuelle Adressenräume (z.B. drei im Fall von z/OS DB2).



PC
Registrierkasse
Geldausgabe-
automat

flat files (Dateien, Data Sets), z.B. VSAM

nicht relationale Datenbanken, z.B.
ADABAS
IMS

relationale Datenbanken, z.B.
DB2
Oracle
MS SQL
mysql

Abb. 7.1.2
Datenbank Server in einer typische Client/Server Anwendung

Es existieren unterschiedliche Arten von Datenbanken. Die drei wichtigsten Grundtypen sind:

Relationale Datenbanken, z.B.:

- **Oracle**
- **DB2 – wichtigste Datenbank unter z/OS**
- **Mircrosoft SQL**

Nicht-relationale Datenbanken, unter z/OS häufig eingesetzt:

- **IMS**
- **ADABAS**

Objekt orientierte Datenbanken (wenig Bedeutung unter z/OS), z.B.

- **Poet**

Auf einem Mainframe dominieren die DB2, IMS und ADABAS Datenbanksysteme.

Der Server Teil einer Client/Server Anwendung besteht aus zwei Teilen: einem Anwendungsprozess und einem Datenbankprozess.

Dargestellt in Abb. 7.1.2 ist eine logische Struktur. 2-Tier, 3-Tier oder n-Tier Konfigurationen unterscheiden sich dadurch, wie diese Funktionen auf physische Server abgebildet werden.

Einige der Alternativen sind:

In einer 2-Tier Konfiguration befindet sich der Anwendungsprozess auf dem gleichen Rechner wie der Klient.

In einer 3-Tier Konfiguration befinden sich Anwendung und Datenbank als getrennte Prozesse auf getrennten Rechnern.

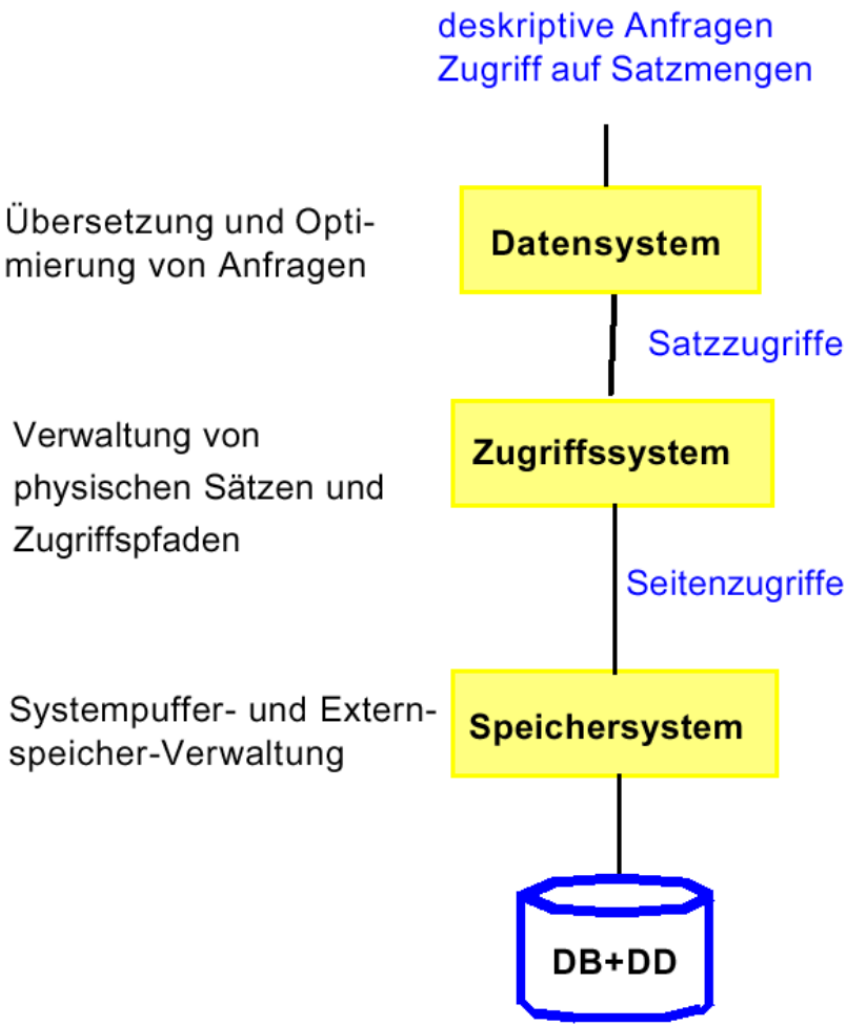


Abb. 7.1.3
Drei Basiskomponenten eines Datenbanksystems

Ein Datenbanksystem ist ein eigenständiger Systemprozess, der in einem eigenen virtuellen Adressenraum (Region) läuft (bei z/OS DB2 sind es 3 Regions).

Das Kernelement ist das Zugriffssystem. Es bildet die auf dem Plattenspeicher befindlichen ungeordneten Daten in der Form von Tabellen (z.B. DB2) oder Hierarchien (z.B. IMS) ab.

Das Speichersystem optimiert die Zugriffe auf die Daten eines Plattenspeichers, z. B. durch die Verwaltung von Ein/Ausgabepuffern (Buffer Pool).

Das Datensystem erlaubt Abfragen und Änderungen der Datenbestände mittels einer benutzerfreundlichen Schnittstelle, z.B. SQL.

Ein wichtiger Bestandteil einer Relationalen Datenbank ist ihr Schema. Das Schema legt fest, welche Daten in der Datenbank gespeichert werden und wie diese Daten in Beziehung zueinander stehen. Den Vorgang zum Erstellen eines Schemas nennt man Datenmodellierung.

Ein Datenbanksystem hat drei Basiskomponenten:

Das Datensystem übersetzt Anfragen von Transaktionsprogrammen, die z.B. in SQL (Structured Query Language) gestellt werden:

```
Select * FROM PERS
WHERE ANR = 'K55'
```

Das Zugriffssystem setzt die Anfrage in logische Seitenreferenzen um.

Das Speichersystem setzt die logische Seitenreferenzen in physische Seitenreferenzen um. Das DB2 Datenbanksystem speichert alle Daten in „Slots“, deren Größe 4096 Bytes, also der Größe eines Seitenrahmens, entspricht.

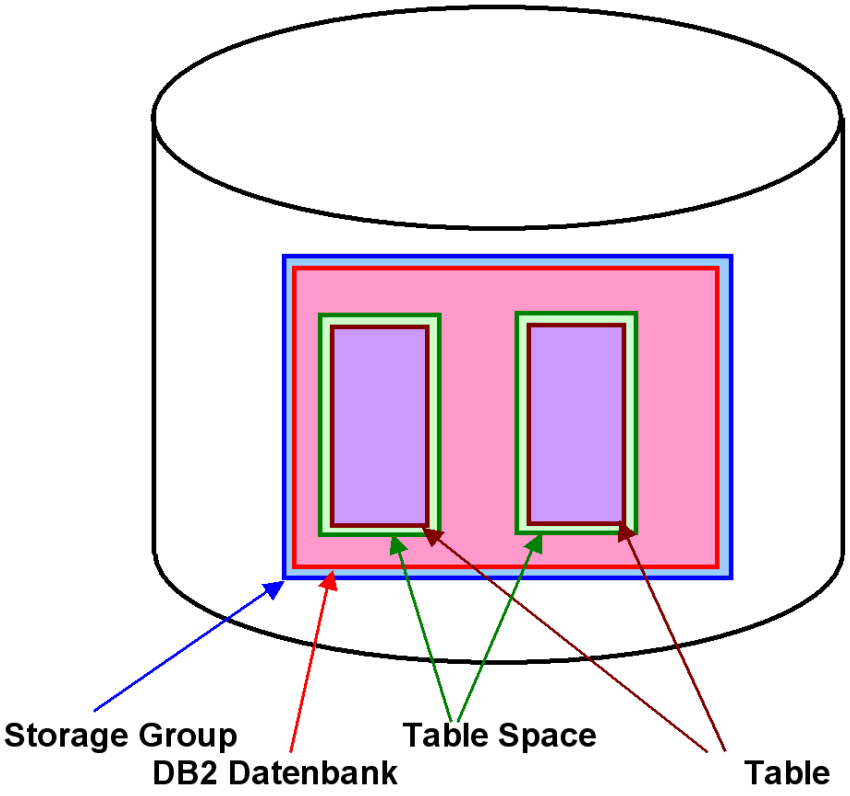


Abb. 7.1.4
Platzzuweisung für eine DB2 Datenbank

Aus der Sicht des Benutzers besteht eine DB2 Datenbank aus mindestens einer, meistens aber mehreren Tabellen.

Auf einem Plattenspeicher wird hierfür Platz in der Form einer Storage Group angelegt. Diese nimmt eine DB2 Datenbank auf.

Innerhalb der Storage Group werden eine oder mehrere Table Spaces angelegt. Jeder Table Space nimmt eine DB2 Table auf.

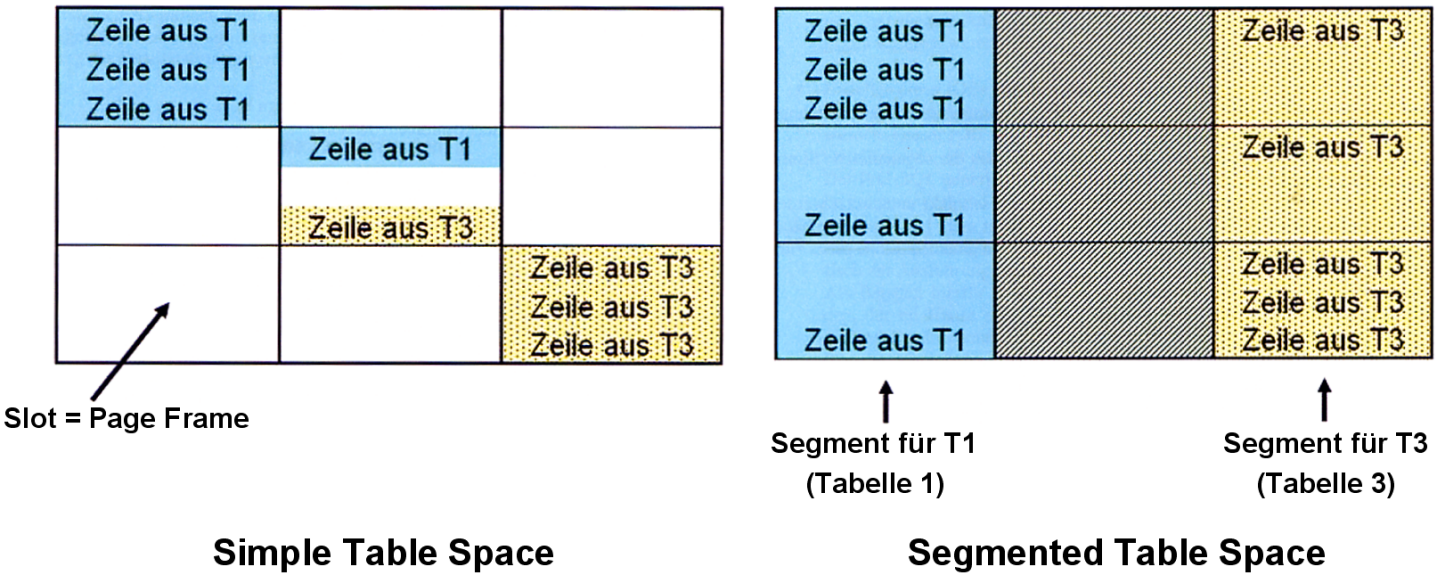


Abb. 7.1.5
Unterschiede in der Datenspeicherung

Ein Table Space besteht aus Slots. Slots sind typischerweise 4096 Bytes groß. Bei einen Zugriff auf die in einer DB2 Tabelle gespeicherten Daten wird ein Slot aus dem Plattenspeicher ausgelesen, und in einen als „Buffer“ bezeichneten 4096 Byte großen Rahmen des realen Hauptspeichers (und die entsprechende Seite des virtuellen Speichers) transportiert.

Ein Table Space kann auch zwei oder mehrere DB2 Tabellen aufnehmen. Bei einem „simple Table Space“ können die Zeilen von zwei Tabellen in dem gleichen Slot untergebracht sein. Der Plattenspeicherbereich von einem „segmented Table Space“ ist in mehrere Segmente aufgeteilt. Ein Segment enthält eine bei der Anlage des Table Spaces angegebene Anzahl von Slots (4, 8, ..64). Jedes Segment enthält Einträge von nur einer Tabelle.

7.1.3 Pufferverwaltung im Datenbanksystem

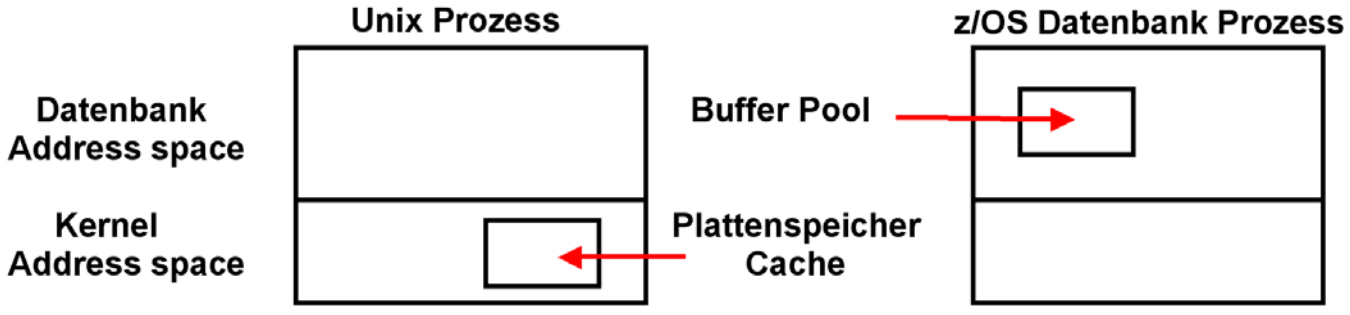


Abb. 7.1.6
Unterschiede zwischen Unix und z/OS

Unix Systeme verbessern den Zugriff auf Daten, indem sie im Hauptspeicher einen Plattenspeicher-Cache für alle Arten von I/O Daten verwalten. Dieses Konzept existiert nicht unter z/OS.

z/OS DB2 und z/OS IMS speichern eine Anzahl von derzeitig benutzten Daten Items in I/O-„Buffern“ im Hauptspeicher. Ein Buffer speichert in der Regel mehrere Records, z.B. ein VSAM Control Intervall oder einen mehrere Zeilen umfassenden Teil einer DB2 Tabelle. Die Menge aller derzeitig angelegten Buffer wird als Bufferpool bezeichnet.

Der Bufferpool befindet sich im Addresspace des Datenbanksystems, welches eine aufwendige Verwaltung seiner I/O Puffer vornimmt. Die Suche im Datenbankpuffer ist in Software implementiert. Typische Referenzmuster in einem Datenbank-System sind:

- **Sequentielle Suche (z.B.: Relationen- Scan)**
- **Hierarchische Pfade (z.B.: Suchen über Bäume)**
- **Zyklische Pfade**

7.1.4 Komponenten der Transaktionsverarbeitung

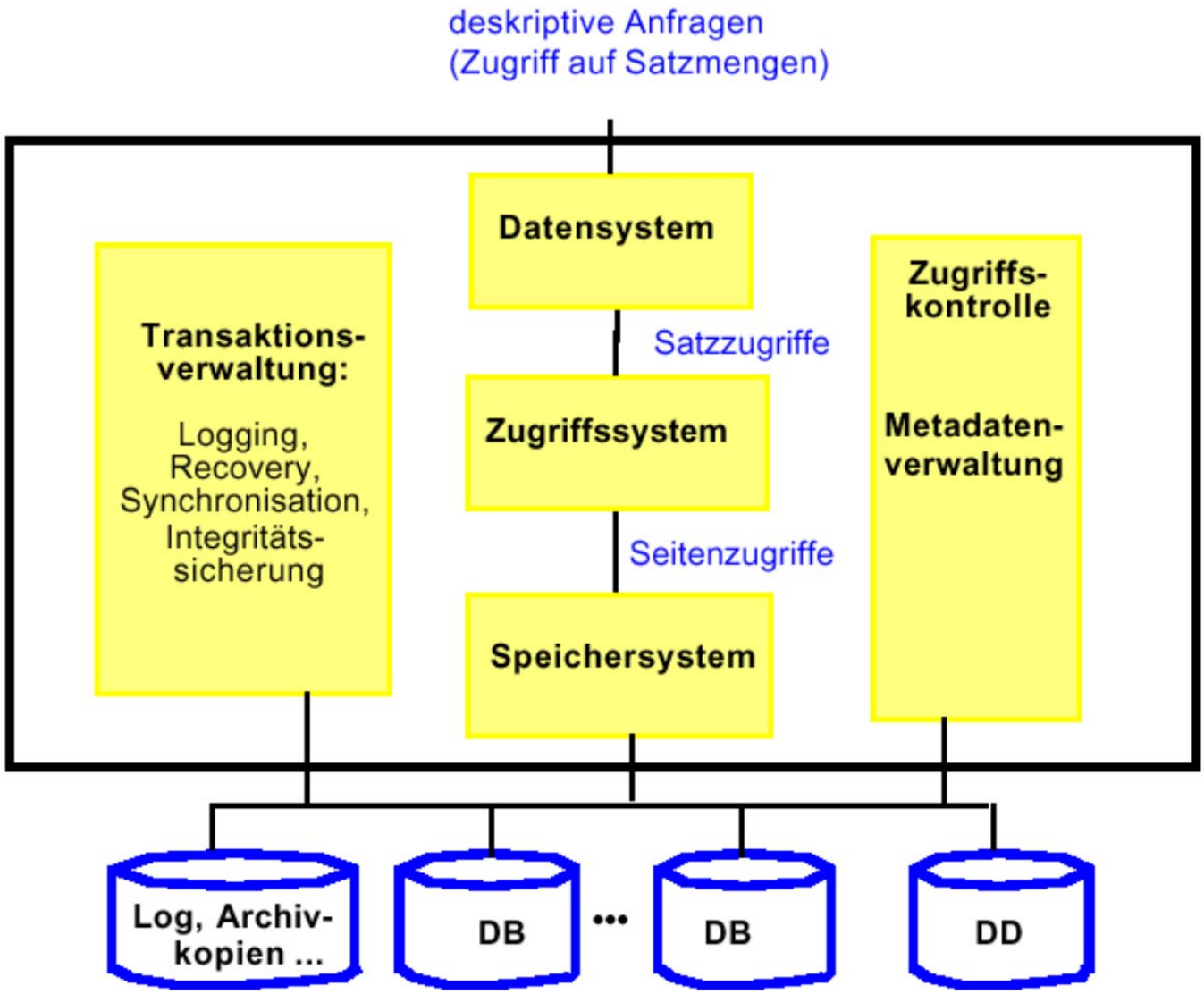


Abb. 7.1.7
Aufbau eines Datenbanksystems

Neben den Komponenten Datensystem, Zugriffssystem und Speichersystem enthält ein Datenbanksystem in der Regel eine Komponente für die Transaktionsverarbeitung, Einrichtungen für die Zugriffskontrolle, Administrationskomponenten sowie Einrichtungen, welche die gespeicherten Daten beschreiben (Metadaten, Data Definition (DD)).

Schließlich sind Einrichtungen für die Verwaltung einer Log-Datei sowie für die Archivierung von Daten vorhanden.

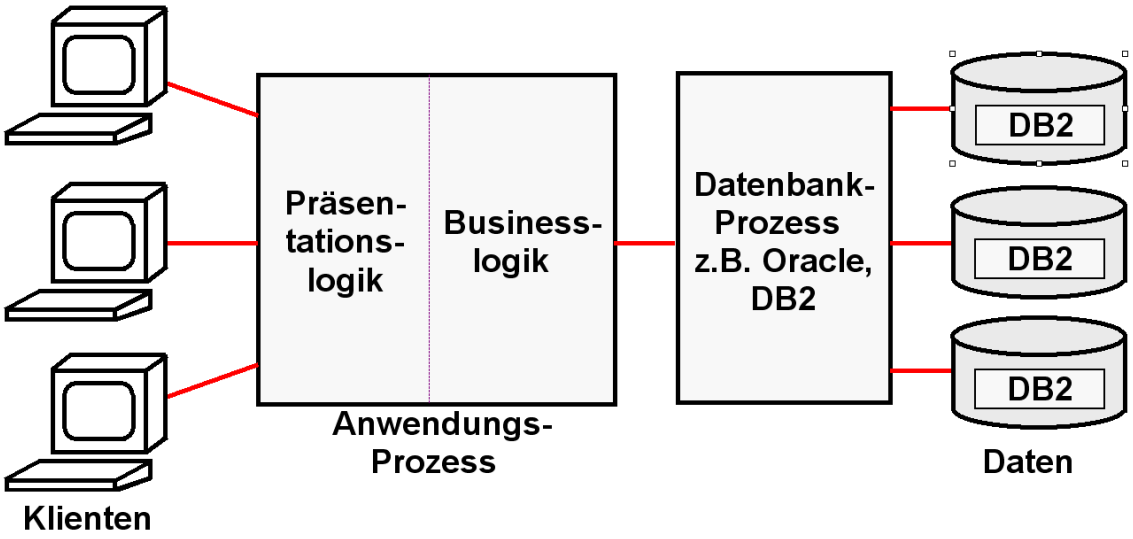


Abb. 7.1.8
Aufteilung des Anwendungsprogramms

Der Anwendungsprozess besteht aus zwei Teilen:

Business Logic (Anwendungslogik) verarbeitet die Eingabedaten des Endbenutzers und erzeugt Ausgabedaten für den Endbenutzer, z.B. in der Form einer wenig strukturierten Zeichenkette (oft als „Unit Record“ bezeichnet).

Präsentationslogik formt die rohen Ausgabedaten in eine für den Endbenutzer gefällige Form um, z.B. in Form einer grafischen Darstellung. Unterschiedliche Klienten können die gleiche Business Logik benutzen um mit unterschiedlicher Präsentationslogik die Ausgabedaten unterschiedlich darzustellen.

7.1.5 Zwei-Tier Client/Server Architektur

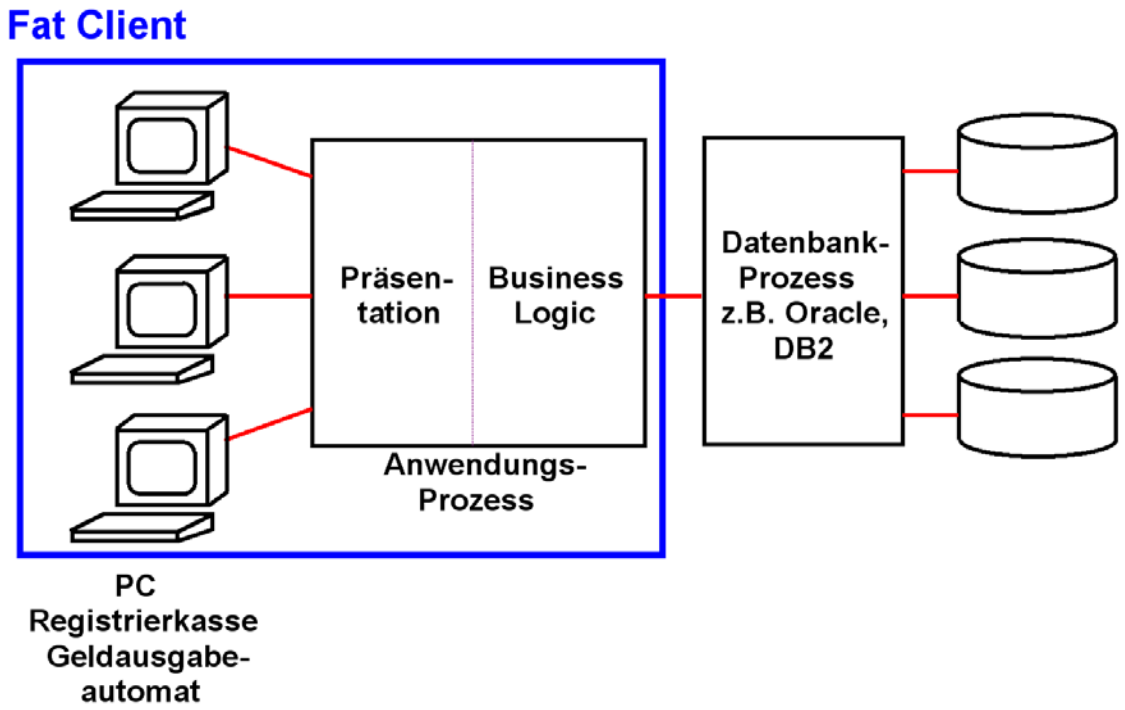


Abb. 7.1.9
Der Anwendungsprozess läuft auf dem Klienten

In einer als „Fat Client“ bezeichneten Konfiguration laufen Business Logic und Präsentationslogik beide auf dem Klienten. Auf dem Server läuft nur der Datenbankprozess.

Dies ist eine häufig in kleinen Betrieben anzutreffende Konfiguration, auch als „2-Tier“ bezeichnet. z.B. teilen sich 12 PC Bildschirm-Arbeitsplätze einer Abteilung einen Datenbankserver.

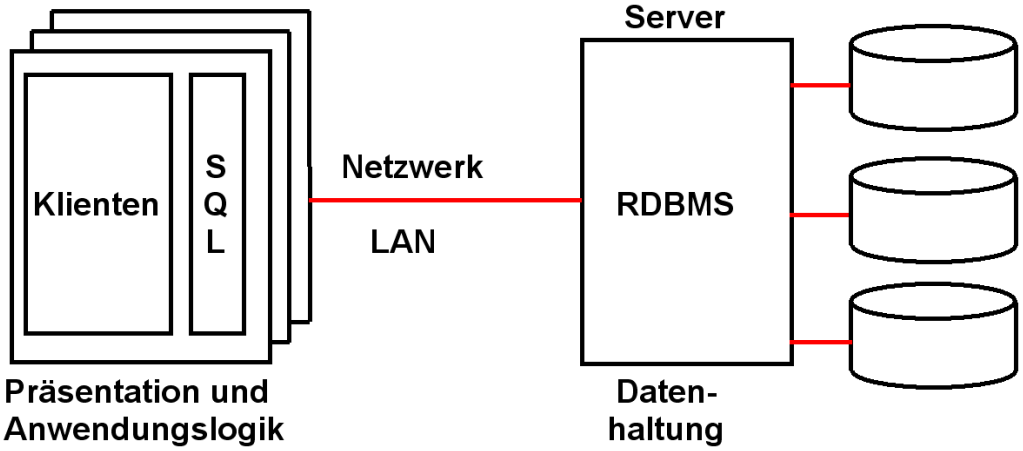


Abb. 7.1.10
SQL Client Komponente

Auf dem Klientenrechner befindet sich eine SQL-Client Komponente, die in der Lage ist, SQL Anfragen über das Netz an den Datenbankserver zu senden.

Typische Umgebungen haben folgende Eigenschaften:

< 200 Klienten
< 100 000 Transaktionen / Tag
LAN Umgebung
1 oder wenige Server
Mäßige Sicherheitsanforderungen

Die Anwendungsentwicklung verwendet häufig Power Builder oder Visual Basic. 2-Tier Konfigurationen werden häufig in reinen Microsoft Umgebungen bei kleinen Unternehmen eingesetzt. Diese Konfiguration skaliert sehr schlecht; deshalb ist sie in größeren Unternehmen nur sehr selten anzutreffen.

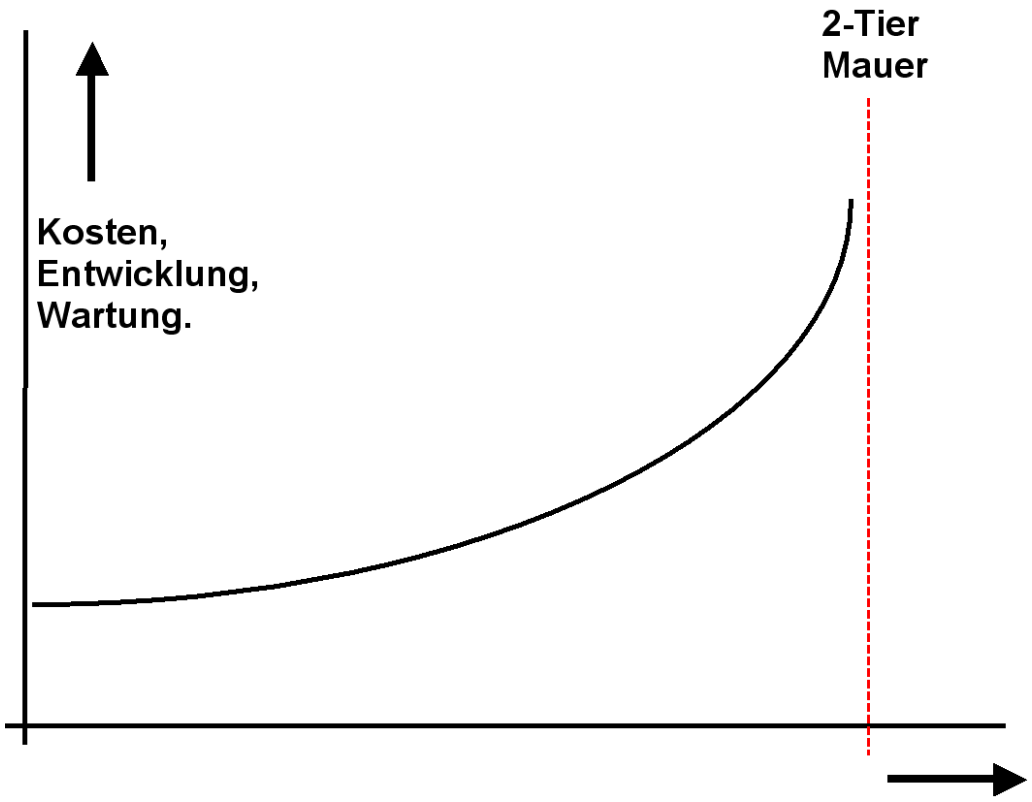


Abb. 7.1.11
Skalierung der 2-Tier Architektur

Die 2-Tier Konfiguration ist sehr populär, leicht zu programmieren, ist aber nur bis zu einer gewissen maximalen Größe möglich (als 2-Tier Mauer bezeichnet). Gründe sind:

- **Datenvolumen auf dem Netzwerk**
- **Datensatz Lock Contention. Was passiert, wenn 2 Klienten gleichzeitig auf den gleichen Datensatz zugreifen ?**
- **Verteilung (Administration) der Klienten Software auf einer Vielzahl von Klienten Rechnern**

7.1.6 Drei-Tier Client/Server Architektur

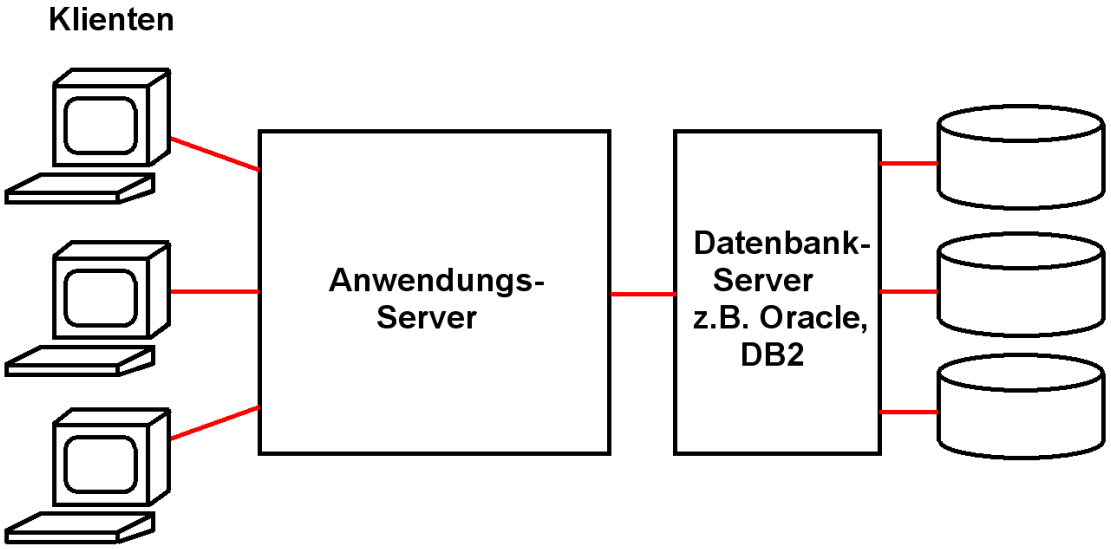


Abb. 7.1.12
Alle Anwendungen laufen auf einem getrennten Anwendungsserver

Bei der 3-Tier Konfiguration existiert ein von den Klienten getrennter Anwendungsserver, wobei entweder zwei getrennte Rechner für Anwendung und Datenbank benutzt werden, oder alternativ der Anwendungs- und der Datenbankprozess auf dem gleichen Rechner laufen. Letzteres ist eine typische z/OS Konfiguration. Für den Anwendungsserver existieren zwei Arten:

- **Präsentationslogik läuft auf dem Klienten. Business Logik läuft auf dem Anwendungsserver. Beispiel: SAPGUI des SAP R/3 Systems**
- **Präsentationslogik und Business Logik laufen auf dem Anwendungsserver Beispiel: Servlets, Java Server Pages und Enterprise Java Beans.**

Die 3-Tier Konfiguration skaliert wesentlich besser als die 2-Tier Konfiguration:

- **Service Anforderungen generieren weniger Datenvolumen**
- **Anwendungsserver optimiert Lock Contention**
- **Mehrfache Anwendungsserver sind möglich (Anwendungsreplikation)**

Die Zugriffskontrolle erfolgt auf Service Basis.

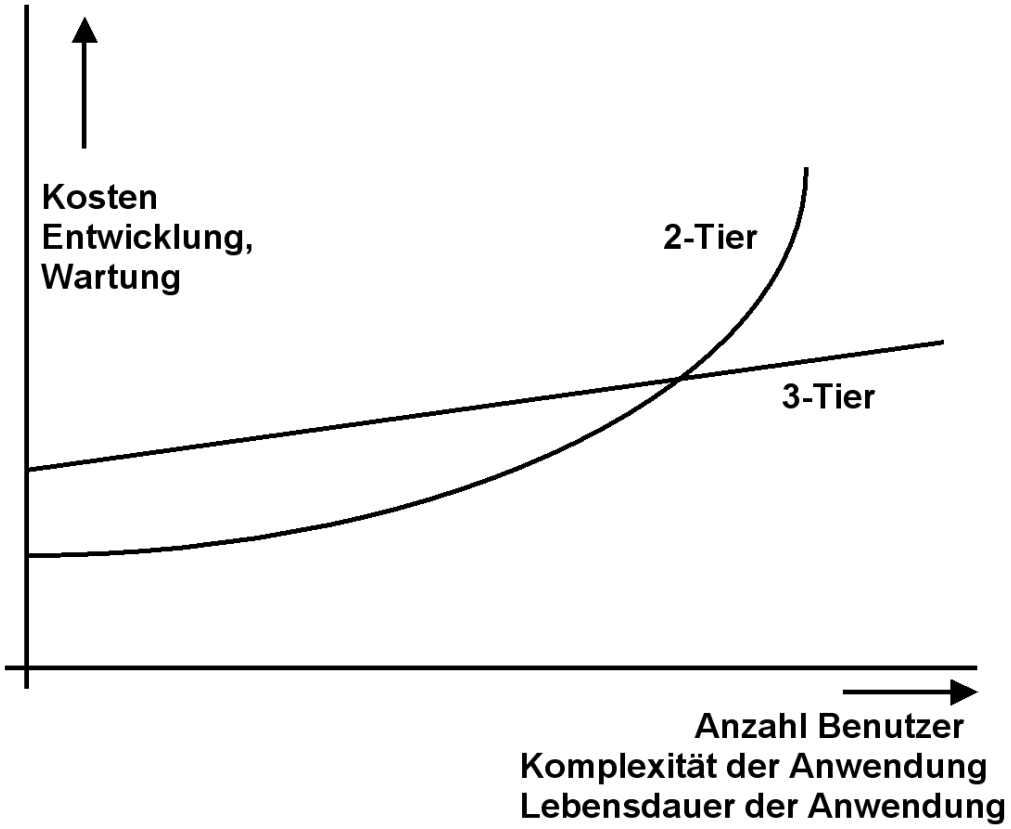


Abb. 7.1.13
Überwindung der 2-Tier Mauer

Bei geringer Belastung ist die 2-Tier Konfiguration überlegen – im Mainframe Bereich ein nicht sehr realistischer Fall.

Bei großer Belastung skaliert die 3-Tier Konfiguration wesentlich besser als die 2-Tier Konfiguration.

7.1.7 Transaktionen

Transaktionen sind Stapelverarbeitungs- oder Client/Server-Anwendungen, welche die auf einem Server gespeicherten Daten von einem definierten Zustand in einen anderen überführen. Eine Transaktion wird wie jede Anwendung als ein Prozess oder Thread ausgeführt.

Eine Transaktion ist eine atomare Operation. Die Transaktion wird entweder ganz oder gar nicht durchgeführt.

Eine Transaktion ist die Zusammenfassung von mehreren Datei- oder Datenbankoperationen, welche entweder

erfolgreich abgeschlossen wird, oder
die Datenbank(en) unverändert lässt

Die Datei/Datenbank bleibt in einem konsistenten Zustand: Entweder der Zustand vor Anfang, oder der Zustand nach erfolgreichem Abschluss der Transaktion

Im Fehlerfall, oder bei einem Systemversagen werden alle in Arbeit befindlichen Transaktionen abgebrochen und alle evtl. bereits stattgefundenen Datenänderungen automatisch rückgängig gemacht. Dieser Vorgang wird als Rollback, Backout oder Abort bezeichnet; die Begriffe sind Synonyme.

Wird eine Transaktion abgebrochen, werden keine Daten abgeändert.

7.1.8 ACID Eigenschaften

Eine Transaktion ist ein Anwendungsprozess, der die ACID Eigenschaften implementiert:

Atomizität (Atomicity)

Eine Transaktion wird entweder vollständig ausgeführt oder überhaupt nicht.

Der Übergang vom Ursprungszustand zum Ergebniszustand erfolgt ohne erkennbare Zwischenzustände, unabhängig von Fehlern oder Crashes. Änderungen betreffen Datenbanken, Messages, Transducer und andere.

Konsistenzerhaltung (Consistency)

Eine Transaktion überführt die transaktionsgeschützten Daten des Systems von einem konsistenten Zustand in einen anderen konsistenten Zustand.

Diese Eigenschaft garantiert, dass die Daten der Datenbank nach Abschluss einer Transaktion schemakonsistent sind, d. h. alle im Datenbankschema spezifizierten Integritätsbedingungen erfüllen.

Daten sind konsistent, wenn sie nur durch eine Transaktion erzeugt oder abgeändert werden.

Isolation

Die Auswirkungen einer Transaktion werden erst nach ihrer erfolgreichen Beendigung für andere Transaktionen sichtbar.

Single User Mode Modell. Selbst wenn 2 Transaktionen gleichzeitig ausgeführt werden, wird der Schein einer seriellen Verarbeitung gewahrt.

Dauerhaftigkeit (Durability)

Die Auswirkungen einer erfolgreich beendeten Transaktion gehen nicht verloren.

Das Ergebnis einer Transaktion ist real, mit allen Konsequenzen. Es kann nur mit einer neuen Transaktion rückgängig gemacht werden. Die Zustandsänderung überlebt nachfolgende Fehler oder Systemcrashes.

Bei Einhaltung der ACID Eigenschaften wird angenommen, dass

- **alle Daten auf Plattenspeichern (oder Solid State Disks) mit RAID oder vergleichbaren Eigenschaften, oft mit zusätzlicher Spiegelung, gespeichert werden und**
- **alle Fehlerfälle wie Plattenspeichercrashes oder andere katastrophale Ausfälle unbeschadet überstehen.**

Als Persistenz bezeichnet man eine Datenspeicherung, welche die Durability Bedingung der ACID Eigenschaften erfüllt.

7.1.9 Beispiel für eine Transaktionsverarbeitungsanwendung

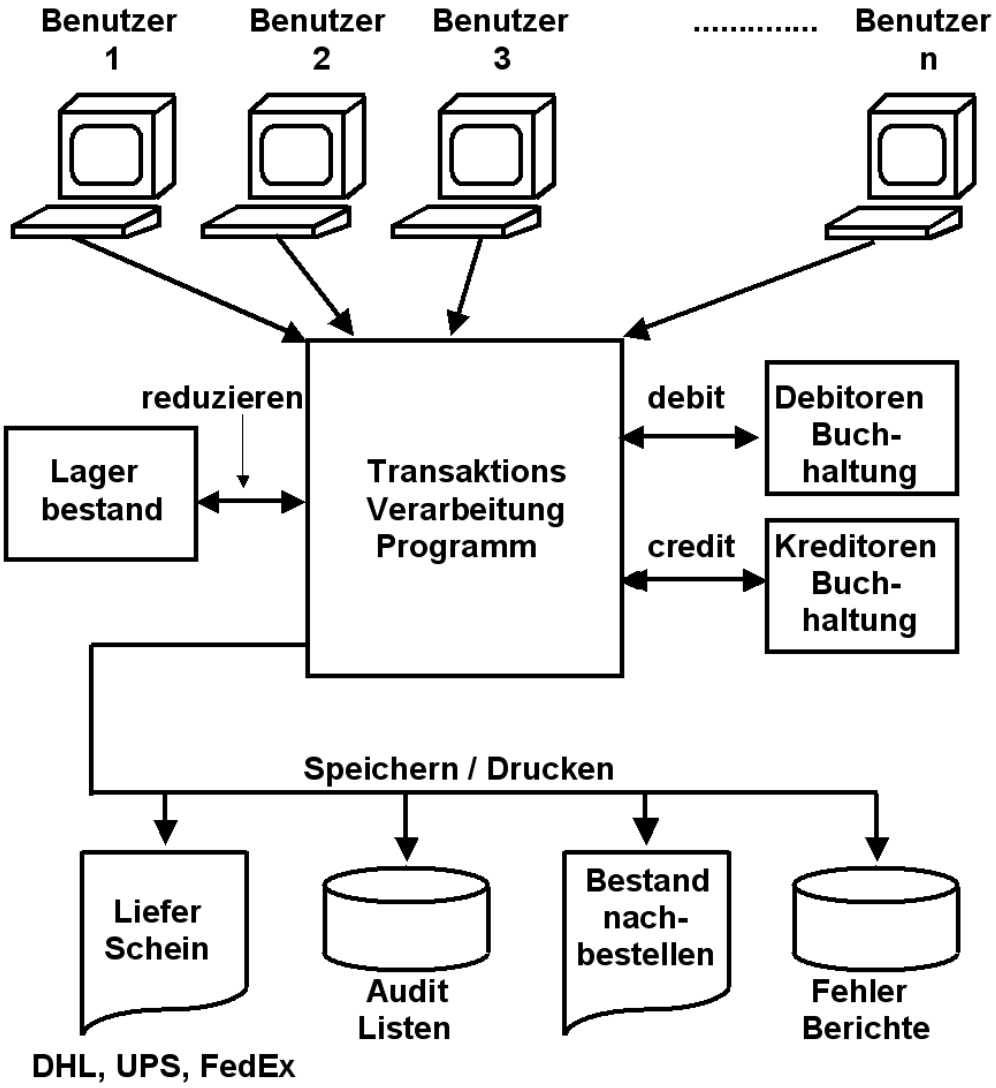


Abb. 7.1.14
Auftragseingang-Bearbeitung

Das Beispiel in Abb. 7.1.14 zeigt das Auftragseingangssystem eines Handelsunternehmens (z. B. Otto Versand), das Aufträge ausliefert, Bezahlungen aufzeichnet, den Orderstatus überprüft und den Warenbestand im Lager überwacht. Die Benutzer sind Kunden, die sich mittels Ihres PCs über das Internet einloggen und jeweils eine Transaktion in der Form eines Bestellvorgangs auslösen. Der Code für ein Beispiel dieser Art ist als ein weit verbreitetes Benchmark (TPC-C) verfügbar (http://www.tpc.org/tpcc/default.asp).

Bei der Bestellung wird der Lagerbestand des bestellten Artikels (z.B. Herrenpullover Gr.43, Bestell Nr. 4711) um die bestellte Menge verringert. Die Debitoren- und Kreditoren-Buchhaltung nehmen Finanzbuchungen vor. Ein Paketdienst (z.B. DHL, UPS, Fedex usw.) erhält einen elektronischen Lieferschein und Auslieferungsauftrag. Ein Zulieferant erhält eine Nachbestellung um den Bestand des bestellten Artikels wieder aufzufüllen. Es werden Audit-Listen und Fehlerberichte erstellt.

Eine teilweise ausgeführte Transaktion wird evtl. wieder rückgängig gemacht, z.B. weil der bestellende Kunde seinen Kreditrahmen überzogen hat oder die angegebene Versandanschrift von keinem Paketdienst angefahren werden kann.

Etwa 80 % aller betrieblichen Anwendungen werden als Transaktionen verarbeitet. Die transaktionalen Verarbeitungsabläufe erfolgen teilweise interaktiv als Client/Server Anwendungen, teilweise als Stapelverarbeitung (Batch Processing).

Interaktive Beispiele:

Auskunftsysteme
Buchungssysteme (z.B. Flugplatzreservierung)
Geldausgabeautomaten
Auftragsbearbeitung, Buchbestellung bei Amazon
Angebot bei eBay abgeben

Stapelverarbeitung Beispiele:

Monatliche Lohn/Gehaltsabrechnung
Jährliche Bilanz erstellen

Das Verhältnis der Rechnerbelastung interaktiv zu Stapel beträgt bei den meisten Unternehmen etwa 60-70% zu 30-40%.

7.1.10 Transactional Memory

Transationaler Memory ist eine zukünftige Entwicklung mit noch unbekannten Potential. Es handelt sich um ein Hauptspeicherkonzept für parallele Berechnungseinheiten, die auf gemeinsame Speicherbereiche zugreifen, wie z.B. Threads oder Mehrprozessorsysteme. Ziel ist es, damit die Ausführungsgeschwindigkeit gegenüber bisherigen Synchronisationsverfahren zu steigern, sowie die Schwierigkeiten der Synchronisierung zu lösen

Transationaler Memory wird mit Hilfe von einer Gruppe von neuartigen Maschinenbefehlen implementiert. Diese erlauben eine Gruppe von regulären Maschinenbefehlen zu definieren, die atomar ausgeführt werden soll.

Transationaler Memory wurde erstmalig im IBMs BlueGene/Q und zEC12 Rechner eingeführt. Intels nächste Prozessorgeneration Haswell wird ebenfalls Transactional Memory in Hardware unterstützten.

TSX: Transactional Synchronisation Extentions, so heißt die Befehlserweiterung, deren Beschreibung man jetzt als bei Intel in der Neufassung der "Intel Architecture Instruction Set Extensions, Programming Reference" herunterladen kann (PDF). Mit TSX soll die Synchronisation zwischen Threads verbessert und vor allem beschleunigt werden. Threads müssen sich bei Zugriffen auf gemeinsame Bereiche miteinander synchronisieren, was viel Zeit kosten kann. Bei Transactional Memory arbeiten die Threads stattdessen zunächst einmal unsynchronisiert und erst beim "Commit" wird überprüft, ob es einen Konflikt gegeben hat. In dem Fall muss die Transaktion verworfen und wiederholt werden, aber das ist vergleichsweise selten.

Konzepte in Software gibt es schon lange, doch die sind bislang zumeist zu ineffizient. Intels TSX bieten dem Programmierer zwei Schnittstellen: Hardware Lock Elison (HLE) mit den neuen Präfixen XACQUIRE und XRELEASE sowie eine Variante namens Restricted Transactional Memoyr RTM, die die neuen Instruktionen XBEGIN, XEND und XABORT bietet. HLE ist die klassische Form, die sich in bestehende Programmkonzepte mit "mutual exclusion" leicht einbringen lässt, RTM ist flexibler, erfordert aber eine Neufassung des Konzepts.

http://www.heise.de/newsticker/meldung/Transactional-Memory-fuer-Intels-Haswell-Prozessor-1432877.html
10.02.2012

7.2 SQL

7.2.1 Structured Query Language

SQL (Structured Query Language) ist eine Programmiersprache für die Abfrage (query) und Abänderung (modify) von Daten und für die Administration von relationalen Datenbanken. Für die hierarchische IMS Datenbank existiert DL/1 als Alternative Programmiersprache.

SQL erlaubt

- **retrieval,**
- **insertion,**
- **updating, and**
- **deletion von Daten.**

In den 70er Jahren entwickelte eine Gruppe am IBM San Jose Research Laboratory das "System R" relationale Database Management System. Diese Arbeit basierte auf einer bahnbrechenden Veröffentlichung von IBM Fellow Edgar F. Codd: "A Relational Model of Data for Large Shared Data Banks". Donald D. Chamberlin und Raymond F. Boyce (beide IBM) entwickelten in 1974 einen ersten Prototypen unter dem Namen System/R. Die "Structured English Query Language" (SEQUEL) wurde für die Manipulation von Daten des System R entworfen. Das Acronym SEQUEL wurde später in SQL abgeändert.

Das erste relationale Datenbank Produkt wurde von der Firma Relational Software herausgebracht (später in Oracle umbenannt). IBM zögerte mit dem eigenen DB2 Produkt, weil das Leistungsverhalten gegenüber IMS als unzureichend angesehen wurde. Die Firma Oracle erbrachte aber den Nachweis, dass relationale Datenbanken ein attraktives Produkt trotz des deutlich schlechteren Leistungsverhaltens sind, und wurde damit Marktführer auf dem relationalen Datenbank-Sektor.

Das Leistungsverhalten von Relationalen Datenbanken wurde in den folgenden Jahrzehnten deutlich verbessert. IMS und Adabas haben aber nach wie vor ein besseres Leistungsverhalten, und werden deshalb immer noch eingesetzt.

Heute zerfällt der relationale Datenbankmarkt in 2 Teile: Relationale Datenbanken für Unix/Linux/Windows und für z/OS. Unter Unix/Linux/Windows hat Oracle nach wie vor einen höheren Marktanteil als DB2. Unter z/OS dominiert DB2; Oracle hat hier eine nur minimale Bedeutung. Ein interessantes Streitgespräch ist zu finden unter

http://mainframeupdate.blogspot.com/2010/02/oracle-versus-ibm-and-db2.html

Die DB2 Version unter Unix/Linux/Windows (distributed Version) ist kompatibel mit der DB2 Version unter z/OS. Die z/OS Version weist aber zahlreiche zusätzliche Funktionen auf, die in der Unix/Linux/Windows nicht vorhanden sind, z.B. Unterstützung für den Sysplex, die Coupling Facility, den Workload Manager, CICS usw.

SQL wurde später von der ANSI und danach von der ISO standardisiert.. Die meisten Datenbank Management Systeme implementieren diese Standards, oft unter Hinzufügung proprietären Erweiterungen. Die proprietären Erweiterungen bringen in der Regel wesentliche Vorteile, und werden deshalb gerne benutzt. Trotz aller Standardisierungsbemühungen ist es aus diesem Grunde in der Regel nicht möglich, SQL Code von einer Datenbank auf eine andere Datenbank ohne Eingriffe zu portieren.

Dies gilt für die Nutzung unter Cobol, Fortran, REXX, C++, PL/1 oder ADA. Für die Nutzung unter Java existieren die Java Standards SQLJ und JDBC mit besserer Kompatibilität.

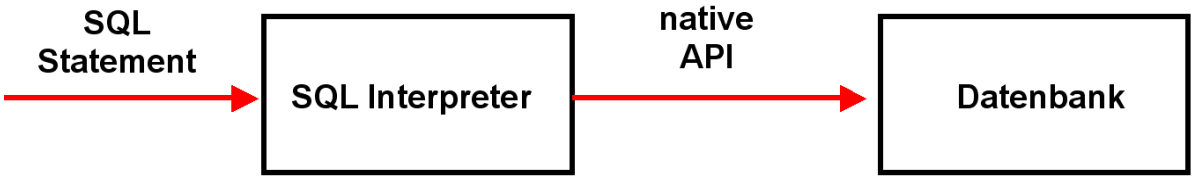


Abb. 7.2.1
Übersetzung von SQL Statements

Neben SQL existiert für jede Datenbank eine nicht standardisierte, proprietäre, native Application Programming Interface (API). Ein Interpreter übersetzt bei jeder Anfrage das SQL Statement in die native API. Da dies Aufwand bedeutet, sind Zugriffe direkt in der nativen API in der Regel performanter als Zugriffe in SQL.

7.2.2 Embedded SQL

Ein Anwendungsprogramm implementiert Datenbankzugriffe über eingebettete SQL-Anweisungen. Diese werden durch "exec sql" eingeleitet und durch ein spezielles Symbol (Delimiter, ";" in C++) beendet. Dies erlaubt einem Precompiler die Unterscheidung der exec sql Anweisungen von anderen Anweisungen.

```
main( )
{
   ........
   ........
   exec sql insert into PERS (PNR, PNAME) values
                  (4711, ´Ernie´);
   ........
   ........
}
```

Abb. 7.2.2
Embedded SQL Beispiel 1

Cobol verwendet statt dessen „END EXEC als Delimiter:

```
  EXEC SQL
       UPDATE CORPDATA/EMPLOYEE
           SET SALARY = SALARY * :PERCENTAGE
           WHERE COMM >= :COMMISSION
  END-EXEC.
```

Abb. 7.2.3
Embedded SQL Beispiel 2

Ein Programm kann viele EXEC SQL Statements enthalten

```
exec sql include sqlca; /* SQL Communication Area*/
main ()
{
exec sql begin declare section;
 char X[8];
 int   GSum;
exec sql end declare section;
exec sql connect to dbname;
exec sql insert into PERS (PNR, PNAME) values (4711, 'Ernie');
exec sql insert into PERS (PNR, PNAME) values (4712, 'Bert');
printf ( "ANR ? ") ; scanf(" %s", X);
exec sql select sum (GEHALT) into :GSum from PERS  where ANR = :X;
printf ("Gehaltssumme %d\n", GSum)
exec sql commit work;
exec sql disconnect;
}
```

Abb. 7.2.4
Embedded SQL Beispiel 3

Eingebettete SQL Anweisungen werden durch "EXEC SQL" eingeleitet und durch spezielles Symbol (hier ";" oder „END EXEC“) beendet, um einem Precompiler die Unterscheidung von anderen Anweisungen zu ermöglichen

Der Oracle oder DB/2 Precompiler greift sich die exec sql Befehle heraus und übersetzt sie in Anweisungen, die der C-Compiler versteht.

Die connect Anweisung baut die Verbindung zwischen Klienten und Server auf.

Es wird eine Kopie von einem Teil der DB Tabelle erstellt, gegen die alle SQL Befehle Änderungen vornehmen. Die commit Anweisung macht die vorhergehenden SQL Befehle entgültig.

Der Kommunikationsbereich SQLCA dient der Rückgabe von Statusanzeigern u.a..

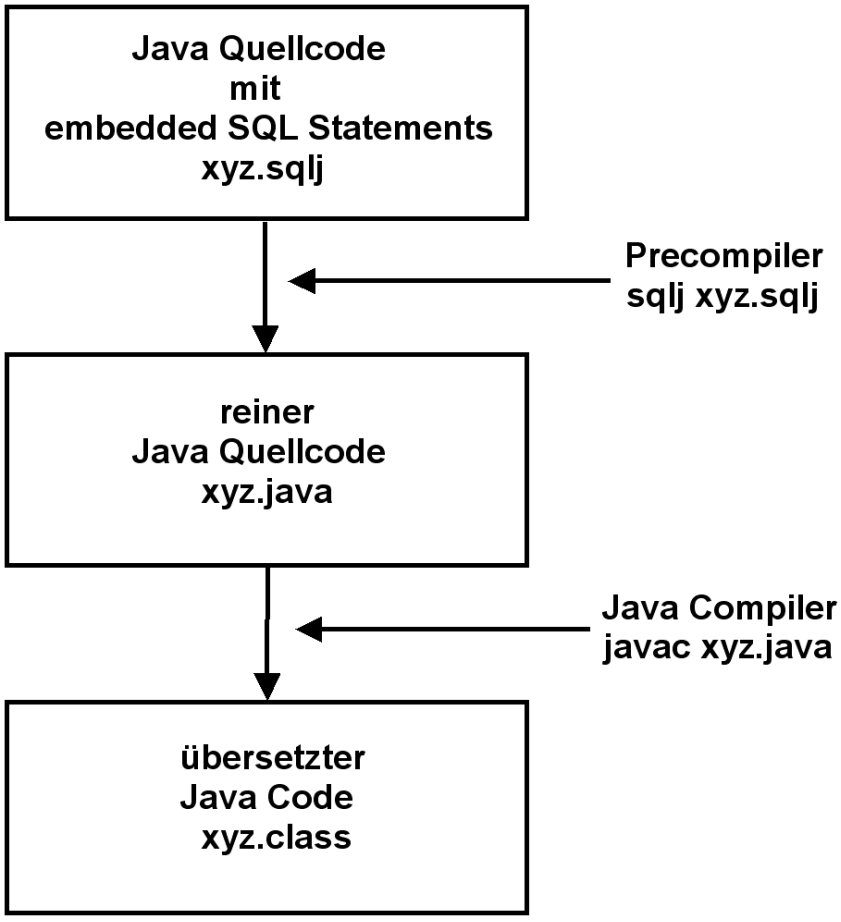


Abb. 7.2.5
SQLJ Precompiler

Embedded SQL bedeutet, wir fügen SQL Kommandos in normale Programmiersprachen wie Cobol, PL/1, C/C++ oder Java ein.

Der Cobol, C/C++ oder Java Compiler versteht die SQL Kommandos nicht. Sie müssen zunächst mit einem Pre-Compiler in Funktionsaufrufe der entsprechenden Programmiersprache übersetzt werden.

Der Precompiler übersetzt embedded SQL Kommandos in entsprechende Datenbank API Calls.

Relationale Datenbanken wie DB2, Oracle, Sybase erfordern unterschiedliche Precompiler

Literatur: IBM Redbook e-business Cookbook for z/OS Volume II: Infrastructure,July 2002, section 4.5.

7.2.3 Zugriff auf eine SQL Datenbank

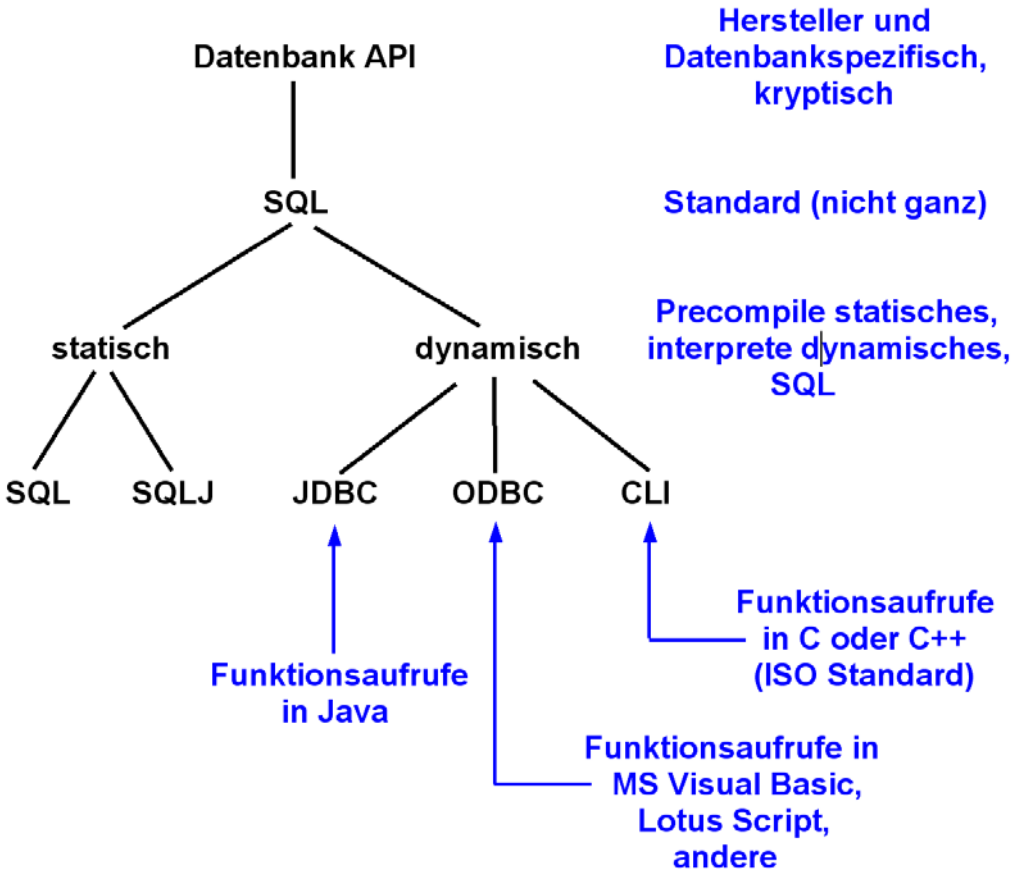


Abb. 7.2.6
Alternative Zugriffsverfahren

Wir unterscheiden zwischen statischem und dynamischem SQL.

Statisches SQL arbeitet mit einem Precompiler. Alle Angaben zu dem Datenbankzugriff müssen zur Compile Zeit festgelegt werden.

Dynamisches SQL ermöglicht Angabe von Parametern erst zur Laufzeit. Z.B. Benutzer gibt gewünschten Datenbanknamen und Tabellennamen in Datenfeld einer HTML Seite ein.

JDBC, ODBC und CLI ist dynamisches SQL für unterschiedliche Programmiersprachen

DB2connect ist ein Klient für statische SQL Zugriffe auf eine DB2 Datenbank. SQLJ ist die Java - spezifische Version von DB2connect.

7.2.4 Vor- und Nachteile von statischem gegenüber dynamischem SQL

```
#sql iterator SeatCursor(Integer row, Integer col,
         String type, int status);
     Integer status = ?;
     SeatCursor sc;
     #sql sc = {
     select rownum, colnum from seats where status <= :status
        };
while(sc.next())                                    SQLJ
          {
 #sql { insert into categ values(:(sc.row()),
         :(sc.col())) };
          }
sc.close();
```

Abb. 7.2.7
Statisches SQL Programmierbeispiel

```
Integer status = ?;
PreparedStatement stmt = conn.prepareStatement("select
         row, col from seats where status <= ?");
if (status == null) stmt.setNull(1,Types.INTEGER);
else   stmt.setInt(1,status.intValue());
ResultSet sc = stmt.executeQuery();
while(sc.next())
{
int row = sc.getInt(1);
boolean rowNull = sc.wasNull();                    JDBC
int col = sc.getInt(2);
boolean colNull = sc.wasNull();
PreparedStatement stmt2 =
         conn.prepareStatement("insert into categ
         values(?, ?)");
if (rowNull) stmt2.setNull(3,Types.INTEGER);
else stmt2.setInt(3,rownum);
if (colNull) stmt2.setNull(4,Types.INTEGER);
else stmt2.setInt(4,colnum);
stmt2.executeUpdate();
stmt2.close();
}
sc.close();
stmt.close();
```

Abb. 7.2.8
JDBC Programmierbeispiel

Was sind die Vor- und Nachteile von SQLJ gegenüber JDBC in Bezug auf Performance?

Die Antwort ist, es hängt sehr stark von den Umständen ab. Ein guter Vergleich ist zu finden unter
http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp?topic=/com.ibm.db2.udb.doc/ad/c0005785.htm .

Ein dynamischer Datenbankzugriff auf eine relationale Datenbank aus einem Java Anwendungsprogramm heraus verwendet das JDBC (Java Data Base Connectivity) Zugriffsverfahren. Es bietet höhere Flexibilität als das statische Verfahren. Z.B. muss der Name der Datenbank erst im Augenblick des Zugriffs und nicht schon zur Compile-Zeit angegeben werden.

Ein Nachteil ist häufig eine schlechtere Performance. Auch ist der Programmieraufwand größer, wie ein Vergleich mit dem vorhergehenden statischen SQLJ Beispiel zeigt.

Im Allgemeinen hat ein Anwendungsprogramm mit dynamischen SQL größere Start-up (oder anfängliche) Kosten pro SQL Statement, weil das SQL Statement vor der Benutzung erst übersetzt (compiled) werden muss. Nach der Übersetzung sollte die Ausführungszeit beim dynamischen SQL ähnlich sein wie beim statischen SQL. Wenn mehrere Klienten das gleiche dynamische Programm mit den gleichen Statements aufrufen, benötigt nur der erste Benutzer den zusätzlichen Übersetzungsaufwand.

7.3 Stored Procedures

7.3.1 Zwei-Tier Client/Server Architecture

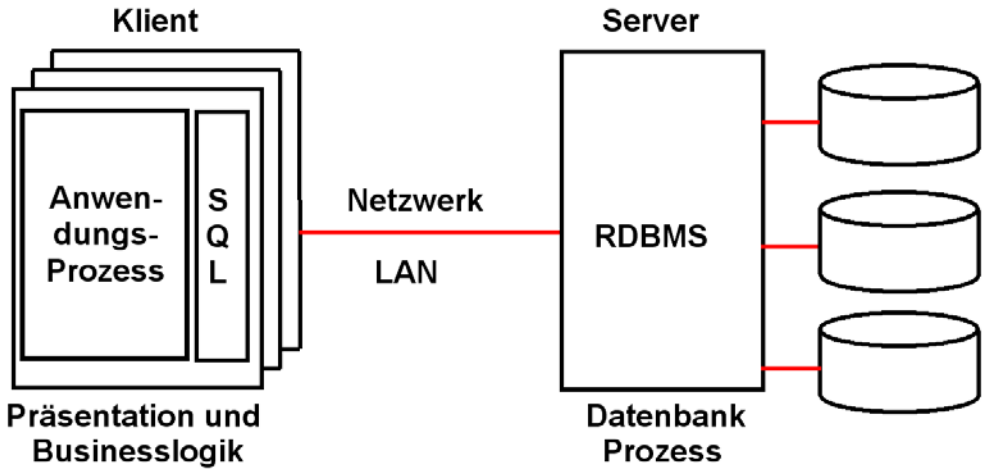


Abb. 7.3.1
Ein SQL Client wird für die Kommunikation mit einer relationalen Datenbank benutzt

Der Anwendungsprozess auf dem Klientenrechner sendet SQL Befehle an den Datenbankrechner. Auf dem Client Rechner befindet sich eine SQL Client Komponente, die in der Lage ist, SQL Anfragen über das Netz an den Datenbankserver zu senden.

Der Datenbankprozess stellt sicher, dass jedes einzelne SQL Statement unter Einhaltung der ACID Bedingungen ausgeführt wird.

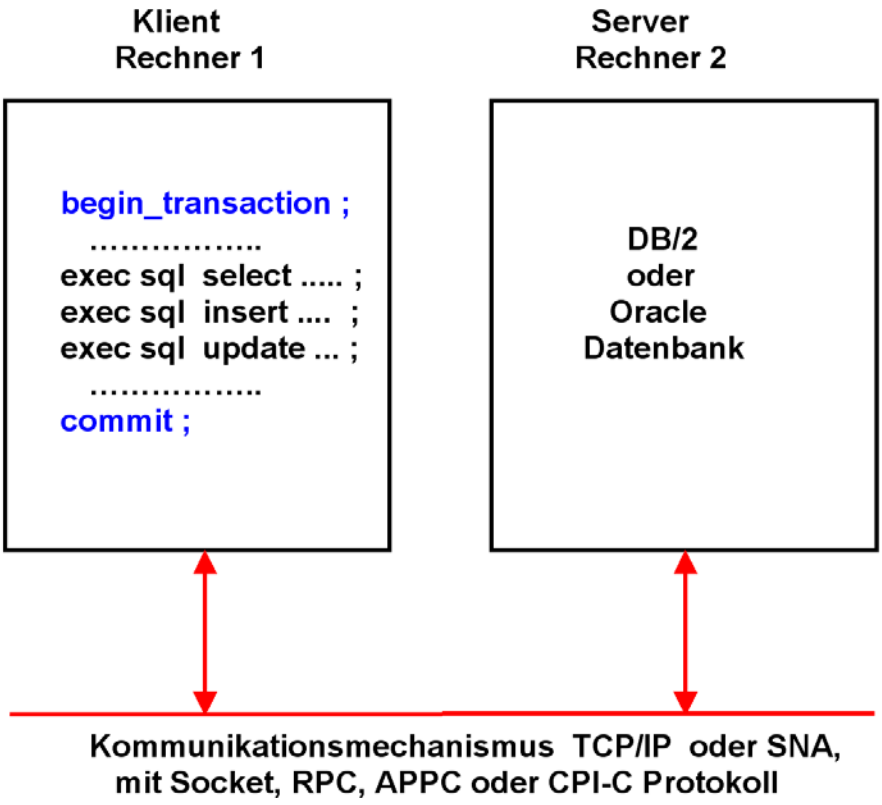


Abb. 7.3.2
ACID Eigenschaften für eine Gruppe von SQL Aufrufen

Der Anwendungsprozess verfügt über einen SQL Klienten, der die Kommunikation mit dem Datenbank Prozess herstellt. Dabei ist es gleichgültig, ob der Datenbank Prozess auf dem gleichen Rechner wie der Anwendungsprozess läuft, oder auf einem entfernten (remote) Rechner läuft, der mit dem Anwendungsrechner über TCP/IP verbunden ist.

In vielen Fällen führt der Klient eine Reihe von EXEC SQL Kommandos aus, die insgesamt als Gruppe ACID Eigenschaften haben sollen.

Der Rechner führt die Anweisungen zwischen

```
exec sql begin_transaction
```

und

```
exec sql commit
```

als Arbeitseinheit (Work Unit) entsprechend den ACID Anforderungen aus.

Dies stellen die beiden Schlüsselwörter begin_transaction und commit sicher.

Eine Gruppe mit mehrfachen SQL Statements, die ACID Eigenschaften haben, wird als „embedded SQL“ Transaktion bezeichnet.

7.3.2 Embedded SQL Transaktion mit mehrfachen SQL Statements

```
DebitCreditApplication( )
{   receive input message;
    exec sql begin_transaction ;      /* start transaction */
    Abalance = DoDebitCredit (.....);
                        •
                        •
                        •
      exec sql  select ..... ;
          exec sql  insert ....  ;
          exec sql  update ... ;
                        •
                        •
                        •
    if (Abalance < 0 && delta < 0)
         {   exec sql rollback ;    }
    else {
         send output message;
         exec sql commit ;             /* end transaction */
         }
}
```

Abb. 7.3.3
Beispiel einer embedded SQL Transaktion

In dem gezeigten Code Fragment fängt der transaktionale Abschnitt mit exec sql begin_transaction an, und hört mit entweder exec sql commit oder mit exec sql rollback auf. Rollback bewirkt, dass alle Datenbank-Änderungen wieder rückgängig gemacht werden.

An Stelle von Schlüsselworten wie rollback sind auch Schlüsselworte wie abort oder backout gebräuchlich.

Eine Embedded SQL Transaktion mit mehrfachen SQL Statements stellt ein relativ simples Programmiermodell dar. Sie ist einfach zu entwerfen und zu implementieren. Da der ganze Anwendungscode auf dem Klienten läuft, kann ein einzelner Programmierer die Verantwortung für die ganze Anwendung übernehmen.

Es existiert jedoch eine Reihe von Nachteilen.

Da die gesamte Anwendung auf dem Klienten-Rechner läuft, ist für jedes EXEC SQL Statement eine getrennte Netzwerk I/O Operation erforderlich. Dies kann die Performance stark herabsetzen. Weiterhin muss das Anwendungsprogramm über detaillierte Kenntnisse des Datenbank-Designs verfügen. Jede Änderung im Datenbank-Design erfordert eine entsprechende Änderung im Anwendungsprogramm. Schließlich erfordern mehrfache Kopien des Anwendungsprogramms auf mehreren Arbeitsplatzrechnern einen erhöhten Administrationsaufwand.

Stored Procedures lösen diese Probleme.

7.3.3 Stored Procedure

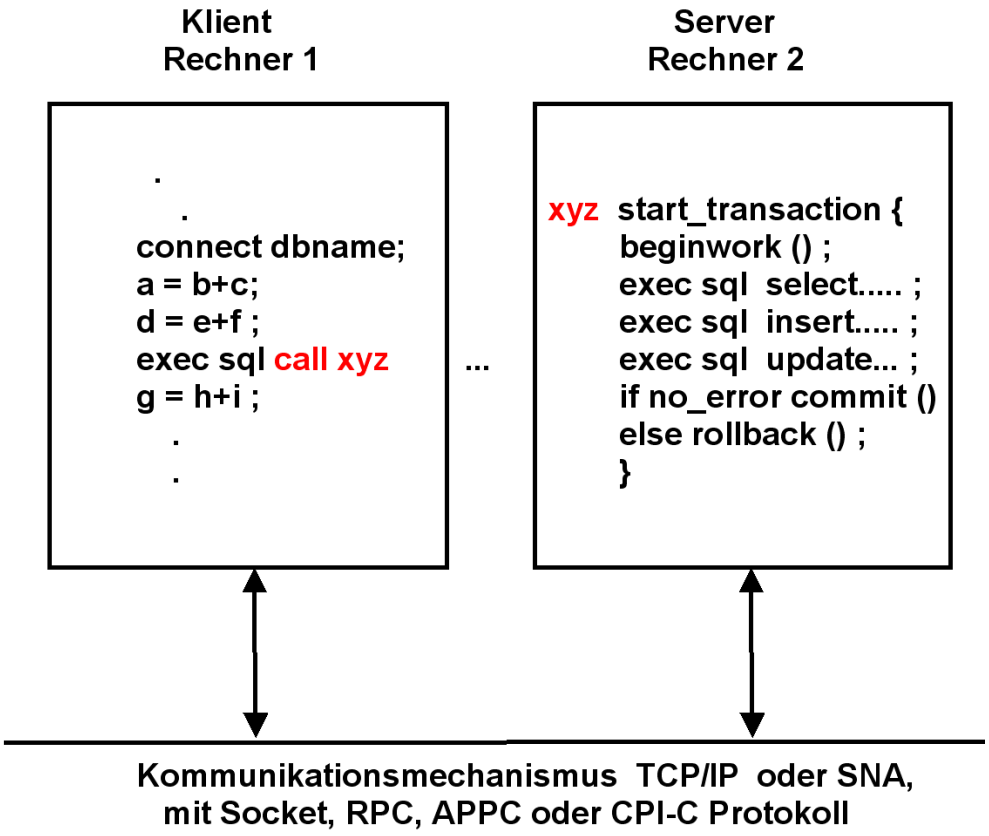


Abb. 7.3.4
ACID Eigenschaften für eine Gruppe von SQL Aufrufen

Mit Hilfe einer "Stored Procedure" wird die Gruppe von SQL Aufrufen nicht auf dem Klienten sondern auf dem Server ausgeführt. Der Klient ruft die Stored Procedure mit Hilfe eines „call" statements auf.

Die Stored Procedure führt eine Gruppe von zusammenhängenden SQL Statements aus. Die Gruppe hat ACID Eigenschaften.

Mit Hilfe des „connect" Statements wählt der Klient die Datenbank mit dem Namen „dbname" (an Stelle einer evtl. anderen angeschlossenen Datenbank) aus.

Abb. 7.3.5 stellt dar, wie durch den Einsatz einer Stored Procedure die Anzahl der SQL Zugriffe über das Netz deutlich reduziert werden kann. Dies verringert das Datenvolumen, welches über das Netz geht.

Zusätzlich wird auch der Aufwand für die TCP/IP Verarbeitung deutlich verringert. Die Stored Procedure läuft in der Regel auf dem gleichen Server wie die Datenbank, und zwar in einem eigenen Address Space, getrennt von dem Datenbank Address Space. Die Kommunikation zwischen den beiden Address Spaces erfolgt aber über den Betriebssystem Kernel und erfordert wesentlich weniger CPU Zyklen als die TCP/IP Verarbeitung.

Als Ergebnis laufen Stored Procedures deutlich performanter als dies bei einzelnen SQL Aufrufen der Fall ist.

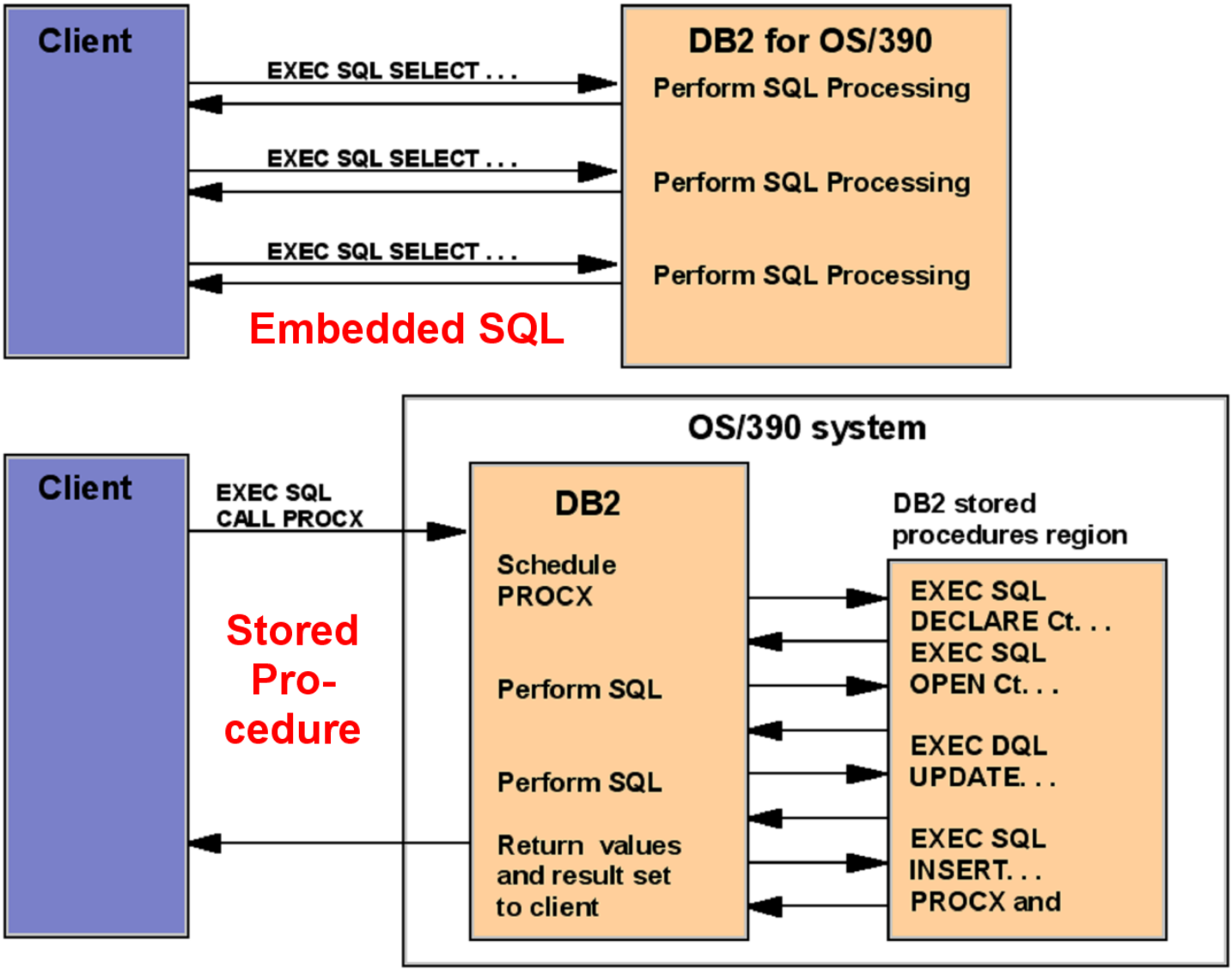


Abb. 7.3.5
Eine Stored Procedure verursacht weniger Kommunikation über das Netz

Stored Procedures werden manchmal als „TP light“ bezeichnet, im Gegensatz zu einem „TP heavy“ Transaktionsmonitor, siehe Abschnitt 7.4. Letzterer startet eigene Prozesse für mehrfache Aufrufe; innerhalb der Prozesse können nochmals Threads eingesetzt werden.

```
exec sql include sqlca; /* SQL Communication Area*/
main ()
{
exec sql begin declare section;
  char X[8];
  int  GSum;
exec sql end declare section;
exec sql insert into PERS (PNR, PNAME) values (4711, 'Ernie');
exec sql insert into PERS (PNR, PNAME) values (4712, 'Bert');
printf ( "ANR ? ") ; scanf(" %s", X);
exec sql select sum (GEHALT) into :GSum from PERS
  where ANR = :X;
printf ("Gehaltssumme %d\n", GSum)
exec sql commit work;
exec sql disconnect;
}
```

Abb. 7.3.6
Beispiel für eine Stored Procedure

Die in Abb. 7.3.6 gezeigte Stored Procedure enthält neben SQL Anweisungen auch Statements in einer regulären Programmiersprache. Eine Stored Procedure kann relativ komplexen Programm Code enthalten.

Traditionell verwenden Stored Procedures unter z/OS hierfür reguläre Programmiersprachen wie COBOL, PL/I, C/C++, Assembler, REXX, and Java (external high-level language stored procedure). z/OS unterstützt neben den traditionellen "external high-level language stored procedures" noch "SQL procedures". SQL procedures sind Stored Procedures, die in der Sprache SQL/PSM (SQL Persistent Modules) geschrieben sind. SQL/PSM ist ein ISO Standard.

Es existiert eine ähnliche, aber proprietäre Programmiersprache PL/SQL (Procedural Language/SQL) für Oracle Datenbanken, die sich in der Syntax an die Programmiersprache Ada angelehnt.

Stored Procedures werden vom Datenbank Server in einer Library abgespeichert und mit einem Namen aufgerufen.

Das Klienten Anwendungsprogramm stellt Verbindung zur Datenbank her mit

```
EXEC SQL CONNECT TO dbname
```

und ruft Stored Procedure auf mit

```
EXEC SQL CALL ServProgName (parm1, parm2)
```

Hierbei ist:

parm1 **Variablen Name (Puffer) der eine Struktur definiert (als SQLDA bei DB2 bezeichnet), die zum Datenaustausch in beiden Richtungen benutzt wird.**

parm2 **Variablen Name (Puffer) der eine Struktur definiert die für Return Codes und Nachrichten an den Klienten benutzt wird.**

Stored Procedures bündeln SQL Statements bei Zugriffen auf Relationale Datenbanken. Sie ersetzen viele, vom Klienten an den Server übergebene, SQL Statements durch eine einzige Stored Procedure Nachricht

Beispiel:

Bei einem Flugplatzreservierungssystem bewirkt eine Transaktion die Erstellung oder Abänderung mehrerer Datensätze:

- **Passenger Name Record (neu)**
- **Flugzeugauslastung (ändern)**
- **Platzbelegung (ändern)**
- **Sonderbedingungen (z.B. vegetarische Verpflegung) (ändern)**

7.3.4 Stored Procedure Datenbank Client

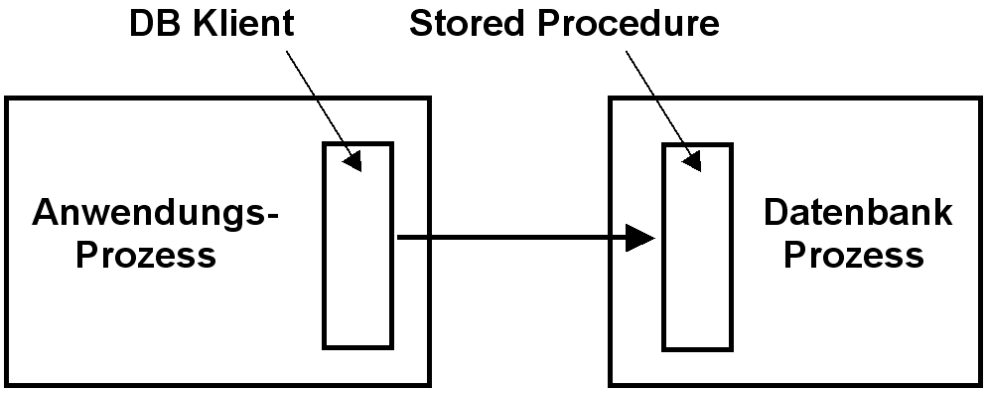


Abb. 7.3.7
Der Aufruf einer Stored Procedure erfolgt nicht über SQL

Bei einer Stored Procedure führt nicht der Anwendungsprozess, sondern der Datenbankprozess, die Gruppe von SQL Statements aus.

Datenbank Hersteller, z.B. IBM, Oracle, Sybase etc. stellen eigene (proprietäre) Datenbank-Klienten zur Verfügung, die mit den Stored Procedures der Datenbank kommunizieren. Datenbank Klienten sind nicht kompatibel miteinander.

Der Klient enthält einen Treiber (DB Klient), der ein proprietäres "Format and Protocol" (FAP) definiert.

FAP unterstützt mehrere Schicht 4 Stacks (TCP/IP, SNA, NetBios, ...)

7.3.5 DB2 Stored Procedures

DB2 ist verfügbar für z/OS, z/VSE, i5/OS (OS/400), and als „DB2 UDB“ (Universal Data Base) für alle verbreiteten Unix (einschließlich Linux) und Windows Betriebssysteme. Es benötigt standardmäßig 3 Address Spaces unter z/OS.

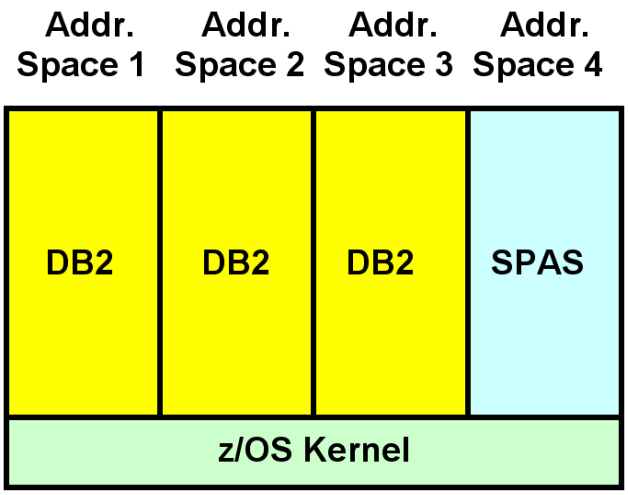


Abb. 7.3.8
Unter z/OS werden Stored Procedures in einem eigenen Address Space (SPAS) ausgeführt

Die Benutzung von Stored Procedures erfordert Verhandlungen mit dem Datenbank Administrator. Das Entwickeln von Stored Procedures ist eine komplexe Aufgabe; das Debuggen kann schwierig sein.

7.3.6 Leistungsverhalten

Das Lehrbuch von R. Orfali, D. Harkey: „Essential Client/Server Survival Guide“. John Wiley, 1994, S. 178, enthält einen detaillierten Leistungsvergleich mit einer einfachen Transaktion, für die der vollständige Code vorliegt. Die Messungen wurden unter dem OS/2 Betriebssystem auf einem Intel 80486 Rechner durchgeführt.

Dies sind die Messergebnisse in ausgeführten Transaktionen pro Sekunde:

Dynamic SQL	**2,2**	**Transaktionen/s**
Static SQL	**3,9**	**Transaktionen/s**
Stored Procedure	**10,9**	**Transaktionen/s**

Zu sehen ist ein deutlicher Performance Vorteil bei der Benutzung von statischen gegenüber dynamischen SQL (Faktor 1,75). Noch deutlicher ist der Performance Vorteil beim Einsatz von Stored Procedures (Faktor 2,79).

Mit einem modernen Rechner wäre die Transaktionsrate (Anzahl ausgeführter Transaktionen pro Sekunde) sehr deutlich höher. Am Verhältnis der Ausführungszeiten würde sich vermutlich nicht viel ändern.

7.3.7 Probleme mit Stored Procedures

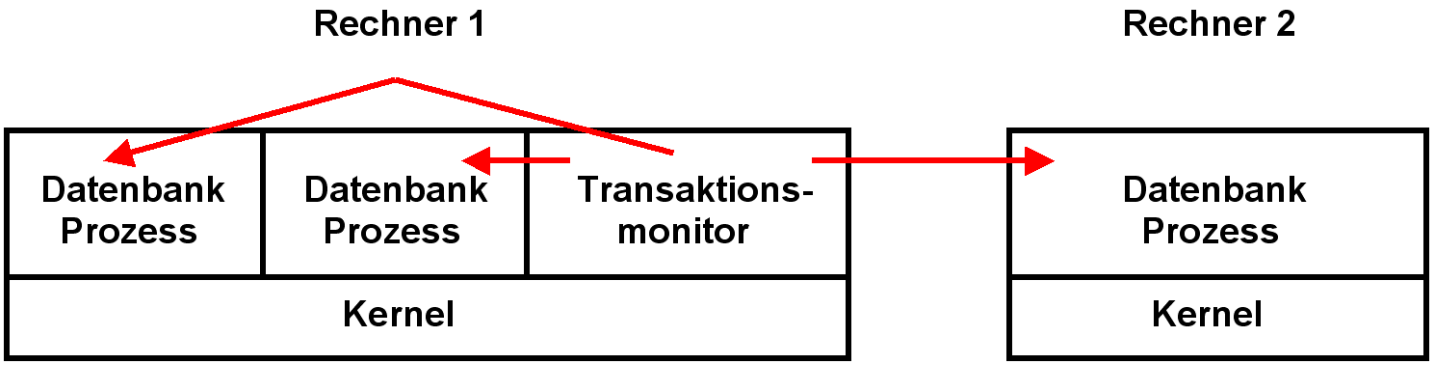


Abb. 7.3.9
Eine Stored Procedure ist Bestandteil einer Datenbank

Die Stored Procedure Laufzeitumgebung ist Bestandteil eines Datenbankprozesses. Häufig ist es jedoch erforderlich, dass eine transaktionale Anwendung auf zwei oder mehr Datenbanken zugreift. Die Zugriffe müssen die ACID Bedingungen erfüllen.

Ein weiteres Problem tritt auf, wenn eine Transaktion auf mehrere heterogene Datenbanken, z.B. von unterschiedlichen Herstellern zugreift. Beispielsweise ist bei der Buchung einer Urlaubsreise in einem Reisebüro ein Zugriff auf die Datenbank einer Fluglinie sowie auf die Datenbank eines Hotels erforderlich. Diese Datenbanken sind nicht identisch.

Zur Lösung implementiert man die Laufzeitumgebung für die Ausführung von Transaktionen als einen getrennten Prozess, der unabhängig vom Datenbankprozess ist, und als „Transaktionsmonitor“ oder „Transaktionsserver“ bezeichnet wird.

Der Transaktionsmonitor (TP Monitor) Prozess übernimmt die Rolle des DB2 Stored Procedure Address Space (SPAS).

Es existieren weitere Vorteile beim Einsatz eines Transaktionsmonitors an Stelle von Stored Procedures:

- **Lastverteilung - Leistungsverhalten - Skalierung**
- **Prioritätssteuerung**
- **Verfügbarkeit (High Availability)**

In Mainframe Installationen werden Transaktionen ganz überwiegend mittels eines Transaktionsmonitors und nicht mittels Stored Procedures ausgeführt. Auch das Leistungsverhalten eines Transaktionsmonitors ist häufig deutlich besser.

7.3.8 Beispiel für Transaktionsmonitore

Transaktionsmonitore sind Softwarepakete (auch als Middleware bezeichnet), die von verschiedenen Herstellern vertrieben werden. Die wichtigsten sind:

- **CICS der Firma IBM**
- **SAP R/3 der Firma SAP**
- **Transaction Server (MTS) der Firma Microsoft**
- **Tuxedo der Firma Bea (heute ein Tochterunternehmen von Oracle)**

Daneben von Bedeutung sind :

- **IMS-DC der Firma IBM**
- **TPF (Transaction Processing Facility) der Firma IBM**
- **UTM der Firma Siemens**
- **NonStop / Guardian / Pathway der Firma Hewlett Packard**

Für die objekt-orientierte Programmierung sind von wachsender Bedeutung:

- **Corba OTS (Object Transaction Service)**
- **EJB JTS (Enterprise Java Bean Transaction Service)**

Unter allen Transaktionsmonitoren hat CICS eine absolut dominierende Position. Weit mehr als die Hälfte aller täglich weltweit ausgeführten Transaktionen werden von CICS verarbeitet.

7.3.9 Begriffe

Die Begriffe Transaction Monitor, Transaction Server und Transaction Service werden (grob gesehen) austauschbar verwendet und bedeuten mehr oder weniger das Gleiche. Ein Transaction Manager ist eine Komponente, die mehrere Transaction Monitore steuert, und sollte mit den Begriffen Transaction Monitor, Transaction Server und Transaction Service nicht verwechselt werden.

Resource Manager ist ein Überbegriff. Ein Resource Manager kann – muss aber nicht – ein Transaktionsmonitor sein.

7.4 Transaction Processing Monitor

7.4.1 Locking

Ein fundamentales Problem der Transaktionsverarbeitung besteht darin, dass ein Transaktionsmonitor aus Performance Gründen in der Lage sein muss, Hunderte oder Tausende von Transaktionen gleichzeitig auszuführen.

Um die Isolation (das `i` in ACID) zu gewährleisten muss der Anschein erweckt werden, dass alle Transaktionen der Reihe nach bearbeitet werden, also dass Transaktion Nr. 2 erst dann an die Reihe kommt, wenn die Verarbeitung von Transaktion Nr. 1 abgeschlossen ist. Wenn beide Transaktionen auf unterschiedliche Daten zugreifen, ist es kein Problem sie trotzdem parallel zu verarbeiten. Wenn beide Transaktionen aber auf die gleichen Daten zugreifen, muss sichergestellt werden, dass keine Verletzung der ACID Eigenschaften auftritt. Dies geschieht mit Hilfe von Locks (Sperren).

Angenommen zwei Transaktionen, die auf die beiden zwei Variablen d1 und d2 zugreifen.

Anfangswerte: d1 = 15, d2 = 20

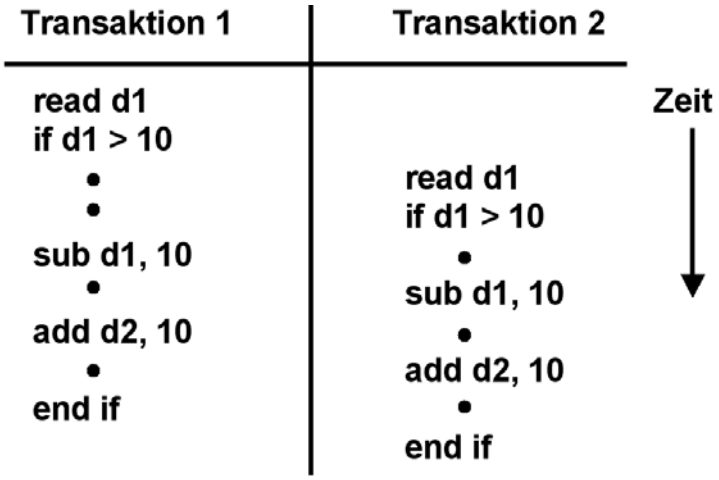


Abb. 7.4.1
Anfangswerte: d1 = 15, d2 = 20

Das Ergebnis ist: d1 = - 5, d2 = 40, obwohl die if-Bedingung ein negatives Ergebnis verhindern sollte.

7.4.2 Concurrency Control

Concurrency Control (Synchronisierung) ist zwischen den beiden Transaktionen erforderlich, um dies zu verhindern. Der Begriff, der dies beschreibt ist Serialisierung (serializability).

Hierfür wird jedes Datenfeld, auf das zwei Transaktionen gleichzeitig zugreifen können, mit einem Lock (einer Sperre) versehen.

Eine Transaktion, welche auf dieses Datum zugreifen will, erwirbt zunächst das Lock. Dies geschieht dadurch, dass das Lock, welches normalerweise den Wert 0 hat, auf 1 gesetzt wird.

Die Transaktion verrichtet nun ihre Arbeit. Nach Abschluss wird das Lock wieder freigegeben (auf 0 gesetzt).

Eine andere Transaktion, die während dieser Zeit ebenfalls versucht, auf das gleiche Datenfeld zuzugreifen, wird daran gehindert, weil das Lock den Wert 1 hat. Sie muss warten, bis das Lock wieder den Wert 0 hat.

Bei großen Datenbeständen können viele Millionen von Locks für unterschiedliche Datenobjekte vorhanden sein

In der Praxis unterscheidet man zwischen Read und Write Locks. Read Locks verhindern „Dirty Reads“. Write Locks verhindern “Lost Updates”.

Mehrere Transaktionen können gleichzeitig ein Read Lock besitzen. Sie können sich gleichzeitig über den Inhalt eines Datenfeldes informieren. Solange der Inhalt des Datenfeldes nicht geändert wird, ist noch nichts passiert.

Wenn eine Transaktion den Inhalt des Datenfeldes ändern will, konvertiert sie das Read Lock in ein Write Lock. Dies bewirkt, dass eine Nachricht an alle anderen Transaktionen geschickt wird, die ein Read Lock für das gleiche Datenfeld besitzen. Letztere wissen nun, dass der Wert, den sie gelesen haben, nicht mehr gültig ist.

Hierzu ist erforderlich, dass nur eine Transaktion in jedem Augenblick ein Write Lock für ein bestimmtes Datenfeld besitzen darf. Write Locks werden deshalb auch als „Exclusive Locks“ bezeichnet.

Angenommen zwei Transaktionen, die auf die beiden zwei Variablen d1 und d2 zugreifen.

Anfangswerte: d1 = 15, d2 = 20

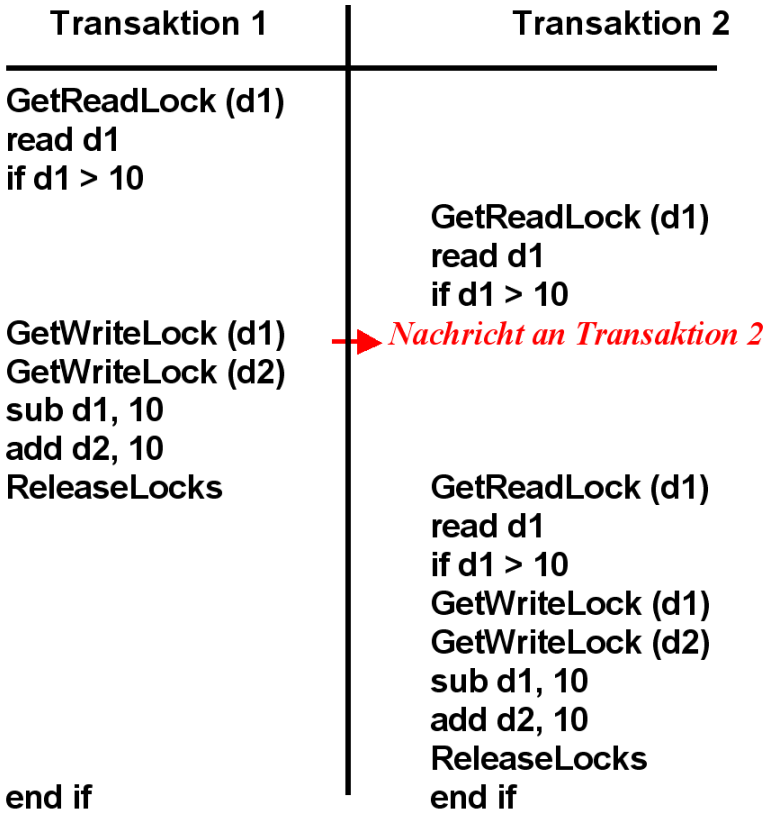


Abb. 7.4.2
Benachrichtigung an Transaktion 2 betr. Ungültiges ReadLock

Ergebnis: d1 = + 5, d2 = 30. Transaktion 2 ändert die beiden Variablen d1 und d2 nicht mehr

Fred Brooks erfand Locks in 1964 bei der Entwicklung von OS/360, damals als "exclusive control" bezeichnet.

7.4.3 Two-Phase Locking

In Transaktionssystemen und Datenbanksystemen werden Locks (Sperren) benutzt, um Datenbereiche vor einem unautorisierten Zugriff zu schützen. Jedem zu schützenden Datenbereich ist ein Lock fest zugeordnet. Ein Lock ist ein Objekt welches über 4 Methoden und zwei Zustände S und E verfügt. Die Methode

- **GetReadLock** **reserviert S Lock (shared),**
- **GetWriteLock** **reserviert E Lock (exclusive),**
- **PromoteReadtoWrite** **bewirkt Zustandswechsel S $\rightarrow$ E,**
- **Unlock** **gibt Lock frei.**

Mehrere Transaktionen können ein S Lock für den gleichen Datenbereich (z.B. einen Datensatz) besitzen. Nur eine Transaktion kann ein E Lock für einen gegebenen Datenbereich besitzen. Wenn eine Transaktion ein S Lock in ein E Lock umwandelt, müssen alle anderen Besitzer des gleichen S Locks benachrichtigt werden.

Normalerweise besitzt eine Transaktion mehrere Locks.

In einer Two-Phase Locking Transaktion finden alle Lock Aktionen zeitlich vor allen Unlock Aktionen statt. Eine Two-Phase Transaktion hat eine Wachstumsphase (growing), während der die Locks angefordert werden, und eine Schrumpf- (shrink) Phase, in der die Locks wieder freigegeben werden.

Two Phase Locking ist nicht zu verwechseln mit dem 2-Phase Commit Protokoll der Transaktionsverarbeitung (siehe in diesem Abschnitt weiter unten).

7.4.4 Programmierung eines Locks

Die Implementierung eines Locks wird durch spezielle Maschinenbefehle unterstützt, z.B. `Test und Set` sowie `Compare und Swap` im Falle von System z. Diese ermöglichen eine atomare (gleichzeitige) Änderung mehrerer Datenelemente.

Höhere Programmiersprachen benutzen diese Befehle um Lock Funktionen zu ermöglichen, z.B. in C++:

Um eine Funktion `incr` zu implementieren, die einen Zähler x hochzählt, könnte Ihr erster Versuch sein:

```
void incr()
     {
          x++;
     }
```

Abb. 7.4.3
Zählerbeispiel 1

Dies garantiert nicht die richtige Antwort, wenn die Funktion `incr` von mehreren Threads gleichzeitig aufgerufen wird. Das Statement x++ besteht aus drei Schritten: Abrufen des Werts x; Erhöhung um 1, Ergebnis in x speichern. In dem Fall, dass zwei Threads dies im Gleichschritt tun, lesen beide den gleichen Wert, erhöhen um 1, und speichern den um 1 erhöhten Wert. x wird nur um 1 an Stelle von 2 inkrementiert. Ein Aufruf von `incr()` verhält sich nicht atomar; es ist für den Benutzer sichtbar, dass er aus mehreren Schritten besteht. Wir könnten das Problem durch die Verwendung eines Mutex lösen, welcher nur von einem Thread zu einem Zeitpunkt gesperrt werden kann:

Ein Mutex benutzt Maschinenbefehle wie `Test und Set` oder `Compare und Swap` um eine atomare Ausführung der Increment Funktion zu gewährleisten.

```
void incr()
     {
          mtx.lock();
          x++;
          mtx.unlock();
     }
```

Abb. 7.4.4
Zählerbeispiel 2

In Java könnte das so aussehen

```
void incr()
     {
          synchronized(mtx) {
               x++;
          }
     }
```

Abb. 7.4.5
Zählerbeispiel 3

Hans-J. Boehm, Sarita V. Adve: You Don't Know Jack About Shared Variables or Memory. Models. Communications of the ACM, February 2012, vol. 55, no. 2.

Sie können ein transaktionales Anwendungsprogramm schreiben, welches diese Konstrukte benutzt. Das ist jedoch extrem aufwendig und fehleranfällig. Sie benutzen deshalb einen Transaction Processing Monitor, der Ihnen diese Arbeit abnimmt.

7.4.5 Struktur eines Transaction Processing Monitors

Das Zusammenspiel der einzelnen Komponenten wird in Abb. 7.4.6 dargestellt.

Endbenutzer kommunizieren mit dem TP Monitor mit Hilfe von Nachrichten. Klienten (Arbeitsplatzrechner) werden häufig als „Terminals“ bezeichnet.

1. Presentation Services empfangen Nachrichten und bilden die Datenausgabe auf dem Bildschirm des Benutzers ab.

2. Eingabe-Nachrichten werden mit einer TRID (Transaction ID) versehen und in einer Warteschlange gepuffert.

3. Aus Zuverlässigkeitsgründen muss diese einen Systemabsturz überleben und ist persistent auf einem Plattenspeicher gesichert. Eine ähnliche Warteschlange existiert für die Ausgabe von Nachrichten.

4. Die Benutzer-Autorisation erfolgt über ein Sicherheitssystem, z.B. Kerberos.

5. Der Scheduler verteilt eingehende Bearbeitungsanforderungen auf die einzelnen Server Prozesse.

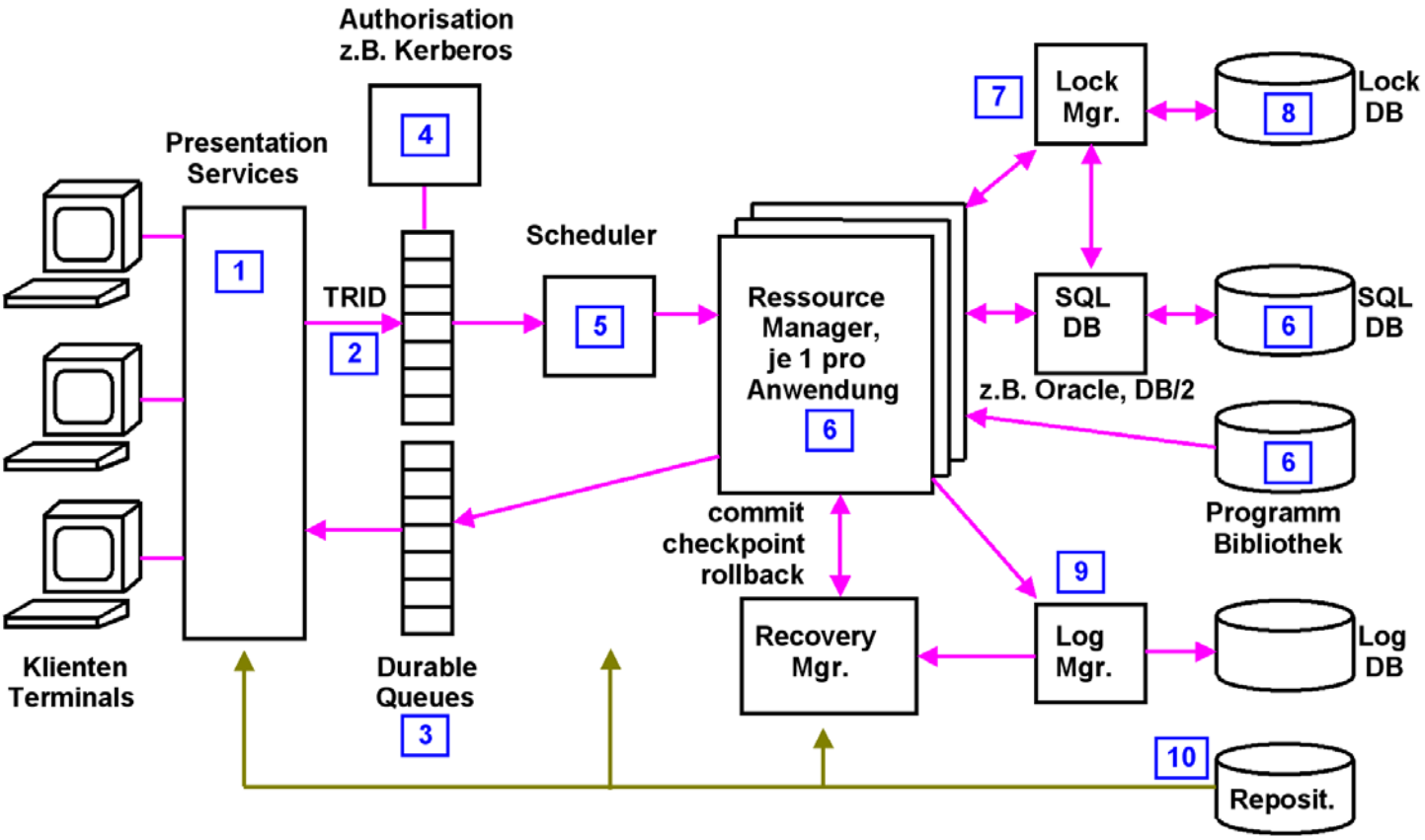


Abb. 7.4.6
TP Monitor

6. **Ein TP Monitor bezeichnet seine Server Prozesse als Ressource Manager. Es existiert ein Ressource Manager pro (aktive) Anwendung. Ressource Manager sind multithreaded; ein spezifischer Ressource Manager für eine bestimmte Anwendung ist in der Lage, mehrere Transaktionen gleichzeitig zu verarbeiten.**

7. **Der Lock Manager blockiert einen Teil einer Datenbanktabelle. Alle in Benutzung befindlichen Locks werden in einer Lock Datei gehalten,**

8. **Aus Performance Gründen befindet sich diese in der Regel im Hauptspeicher. In Zusammenarbeit mit dem Datenbanksystem stellt der Lock Manager die „Isolation“ der ACID Eigenschaft sicher.**

9. **Der Log Manager hält alle Änderungen gegen die Datenbank fest. Mit Hilfe der Log Datenbank kann der Recovery Manager im Fehlerfall den ursprünglichen Zustand der Datenbank wiederherstellen (Atomizität der ACID Eigenschaft).**

10. **In dem Repository verwaltet der TP Monitor Benutzerdaten und -rechte, Screens, Datenbanktabellen, Anwendungen sowie zurückliegende Versionen von Anwendungen und Prozeduren.**

7.4.6 Fehlerbehandlung

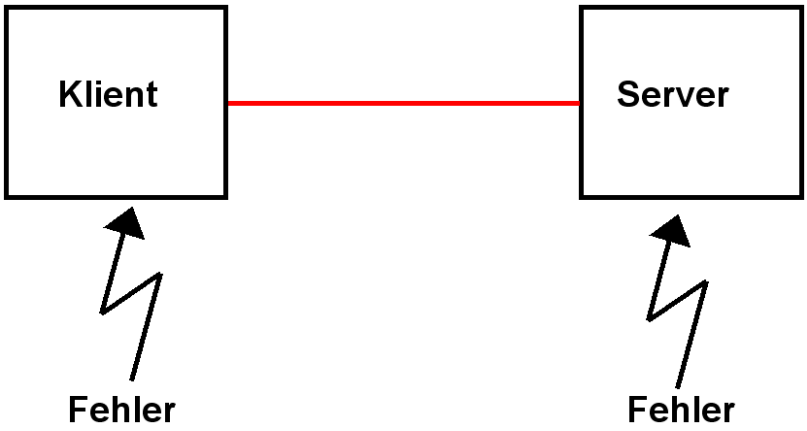


Abb. 7.4.7
Absturz beim Klienten oder beim Server

In einer Client/Server Konfiguration existieren drei grundsätzliche Fehlermöglichkeiten:

1. **Fehler in der Verbindung zwischen Client und Server**
2. **Server kann Prozedur nicht beenden (z.B. Rechnerausfall, SW-Fehler)**
3. **Client-Prozess wird abgebrochen ehe Antwort vom Server eintrifft. Er kann die Antwort des Servers nicht entgegennehmen.**

Fehler in der Verbindung zwischen Client und Server werden in der Regel von TCP in Schicht 4 automatisch behoben und sind deshalb unkritisch. Die beiden anderen Fehlermechanismen erfordern Fehlerbehandlungsmaßnahmen.

7.4.7 Ausfall des Servers oder des Klienten

1. Idempotente Arbeitsvorgänge

"Idempotent" sind Arbeitsvorgänge, die beliebig oft ausgeführt werden können; Beispiel: Lese Block 4 der Datei xyz. Nicht idempotent ist die Überweisung eines Geldbetrages von einem Konto auf ein anderes.

2. Nicht-idempotente Arbeitsvorgänge

Problem: Wie wird festgestellt, ob Arbeitsgang durchgeführt wurde oder nicht?

"Genau einmal" (exactly once).
Überweisung von Konto x nach y

"Höchstens einmal" (at most once).
Stimmabgabe Bundestagswahl

"Wenigstens einmal" (at least once). Ideal für idempotente Vorgänge.
Update Virenscanner

Transaktionen stellen eine „exactly once“ Semantik sicher.

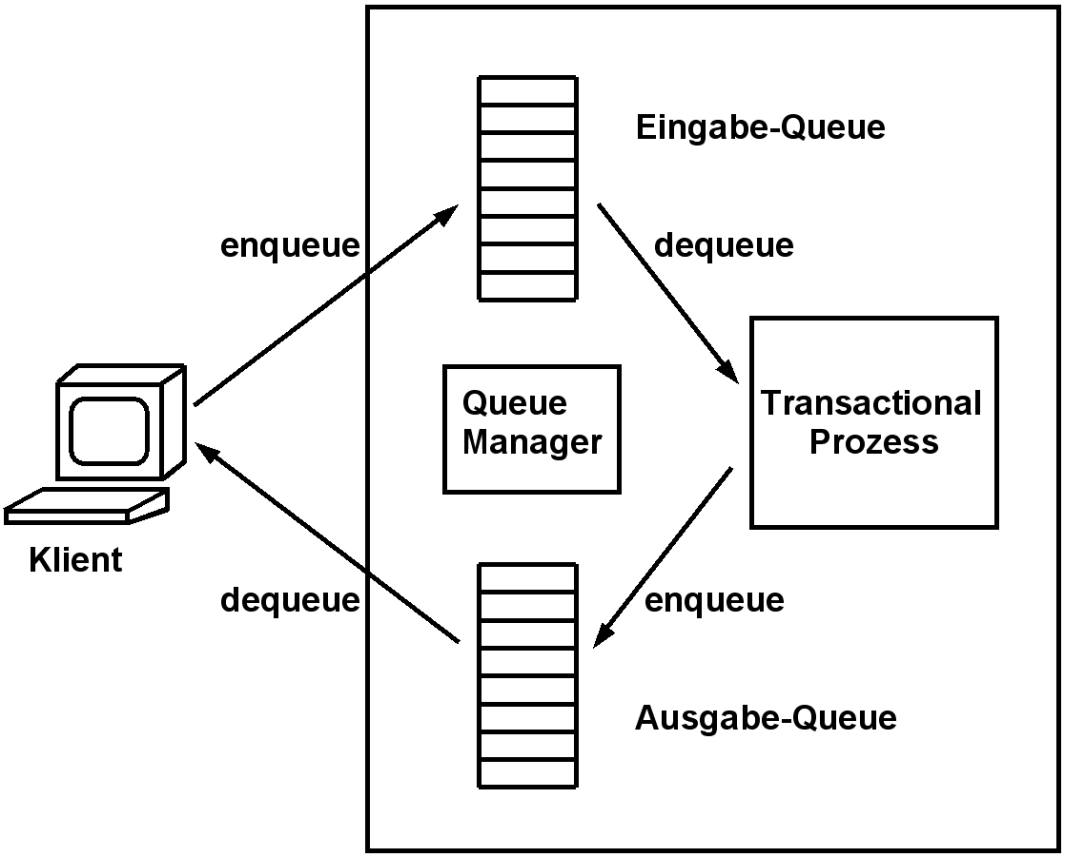


Abb. 7.4.8
Aufgaben der Warteschlangen

Jede einzelne Transaktion wird in 3 Subtransaktionen aufgelöst, die alle eigene ACID Eigenschaften haben.

1. **Subtransaktion 1 nimmt die Eingabenachricht entgegen, stellt sie in die Eingabe Queue, und commits.**
2. **Subtransaktion 2 dequeues die Nachricht, verarbeitet sie, stellt das Ergebnis in die Ausgabe-Queue, löscht den Eintrag in der Eingabequeue und commits.**
3. **Subtransaktion 3 übernimmt das Ergebnis von der Ausgabequeue, übergibt das Ergebnis an den Klienten, löscht den Eintrag in der Ausgabequeue und commits.**

Wenn der Klient abstürzt (Waise), stellt der Server das Ergebnis der Verarbeitung in die Ausgabe Queue. Die Transaktion ist damit vom Standpunkt des Servers abgeschlossen. Das Verarbeitungsergebnis kann beim Wiederanlauf des Klienten aus der Ausgabe-Queue abgeholt werden.

Getrennte Warteschlangen (Queues) für eingehende und ausgehende Nachrichten lösen viele Probleme. Die Queues sind auf einem RAID Plattenspeicher persistent gespeichert. Für den TP Monitor ist die Transaktion abgeschlossen, wenn die Ergebnisse in der Ausgabe Queue festgehalten wurden, auch wenn der Klient zwischenzeitlich ausgefallen ist.

Die Queues nehmen weiterhin eine Prioritätensteuerung und eine Lastverteilung auf mehrere Server vor.

7.4.8 Sicherheitssystem

Unter z/OS werden die Basis-Sicherheitserfordernisse durch den z/OS Security Server abgedeckt, z.B. mit Hilfe von RACF oder einer äquivalenten Software Komponente, z.B. ACS von der Firma Computer Associates. Diese ist für den Login-Process zuständig und regelt weiterhin Zugriffsrechte, z.B. auf bestimmte Daten.

In vielen Fällen ist es zusätzlich erforderlich, eine weitergehende Klienten-Autentifizierung durchzuführen und/oder Daten kryptografisch verschlüsselt zu übertragen. Hierfür werden Sicherheitsprotokolle wie SSL (Secure Socket Layer), TLS (Transport Layer Security), IPsec oder Kerberos eingesetzt.

Abb. 7.4.9
Herkules kämpft mit dem Höllenhund Kerberos

Kerberos, ursprünglich vom Massachusetts Institute of Technology (MIT) entwickelt, gehört zum z/OS Lieferumfang und ist in Mainframe Installationen weit verbreitet. Daneben werden SSL (TLS) und IPsec unter z/OS eingesetzt. Kerberos hat gegenüber SSL den Vorteil, dass ausschließlich symmetrische kryptografische Schlüssel einsetzt werden, was besonders bei kurzen Transaktionen die Performance verbessert.

Kerberos (Κέρβερος) ist der Name des dreiköpfigen Höllenhundes, der in der antiken griechischen Mythologie den Hades (die Unterwelt) bewacht.

7.4.9 Zustand von Prozessen

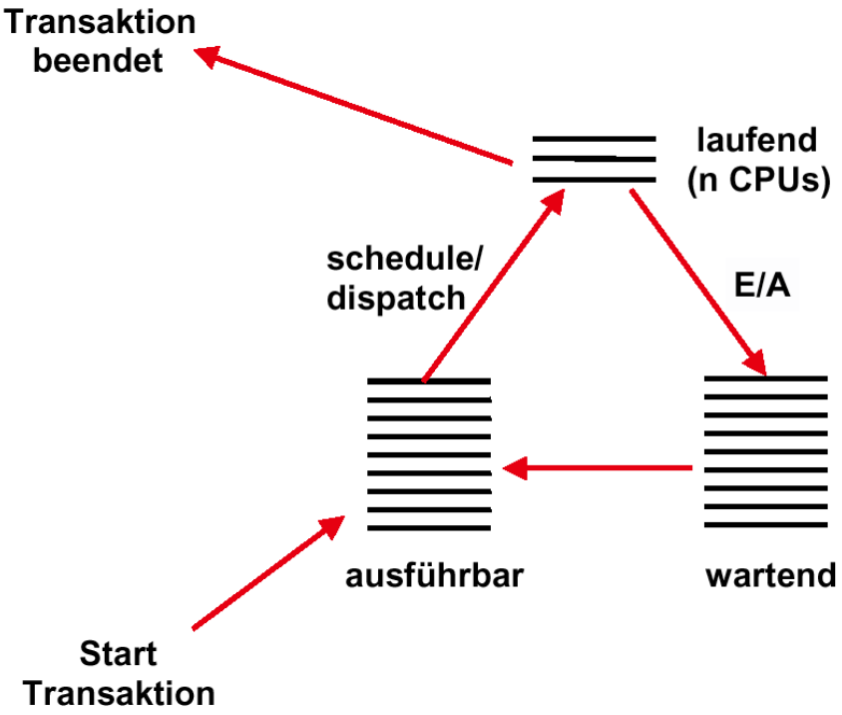


Abb. 7.4.10
Zustandswechsel für aktive Transaktionen

Jede Transaktion rotiert wiederholt durch die Zustände laufend, wartend und ausführbar. Der Scheduler des TP Monitors selektiert aus der Queue ausführbare Transaktionen jeweils einen Kandidaten wenn immer eine CPU frei wird.

In einem Rechner laufen immer zahlreiche Prozesse gleichzeitig ab. Die Prozesse befinden sich immer in einem von drei Zuständen. Hierfür werden drei Wartschlangen benutzt, in denen sich die TCBs der Prozesse befinden.

Ein Prozess befindet sich im Zustand laufend, wenn er von einer der verfügbaren CPUs ausgeführt wird. (Die allermeisten Mainframes sind Mehrfachrechner und verfügen über mehrere CPUs, bis zu 196 bei einem zEC12Rechner).

Ein Prozess befindet sich im Zustand wartend, wenn er auf den Abschluss einer I/O Operation wartet.

Ein Prozess befindet sich im Zustand ausführbar, wenn er darauf wartet, dass die Scheduler/Dispatcher Komponente ihn auf einer frei geworden CPU in den Zustand laufend versetzt.

7.4.10 TP Monitor Repository

Das Repository des TP Monitors speichert alle Daten, die für die interne Ablaufsteuerung benötigt werden. Andere Bezeichnungen sind Katalog oder Dictionary.

Im Repository werden katalogisiert:

- **Benutzer Rechte**
- **Workstation IDs (Terminal IDs)**
- **Screens**
- **Anwendungen**
- **Prozeduren**
- **Tabellen**
- **Rollen (Wer kann/darf/soll wann was machen)**
- ●
- ●
- ●
- ●

Z.B. Hinzufügen eines Feldes in einen Screen ändert Client Software, Server Schnittstelle, fügt eine weitere Spalte in eine SQL Tabelle ein.

7.4.11 Log File Konzept

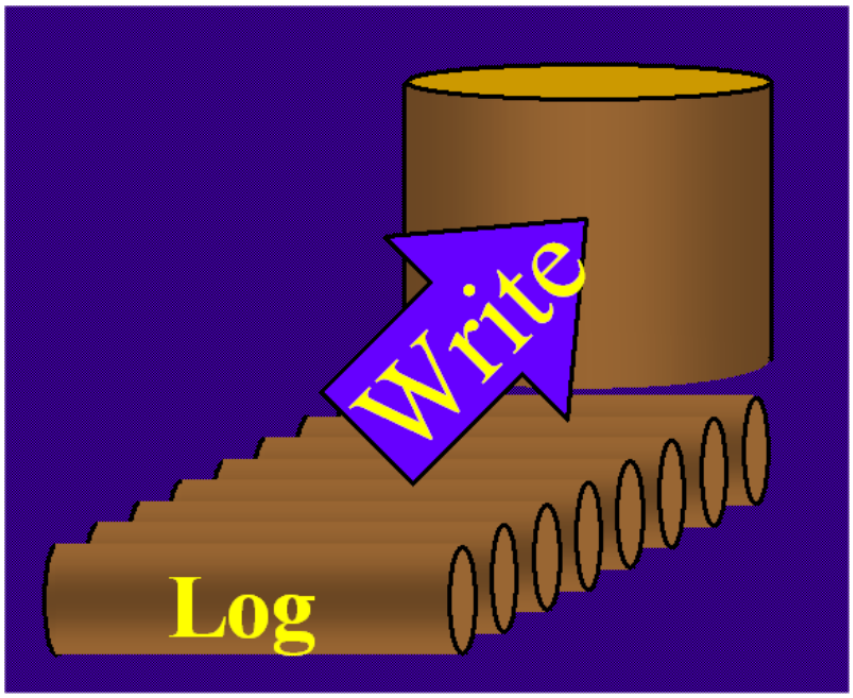


Abb. 7.4.11
Änderungen in der Datenbank werden in einem Log festgehalten

Das Log ist eine historische Aufzeichnung aller Änderungen des Zustands (state) des TP Systems. Log plus alter Zustand ergibt neuen Zustand. Das Log ist eine sequentielle Datei. Das vollständige Log enthält die historische Entwicklung aller Datenbankänderungen.

Beim „Commit“ einer Transaktion wird das Log persistent gespeichert (z.B. auf einer gespiegelten oder einer RAID Festplatte).

Das Log wird zur Wiederherstellung einer Transaktion im Falle eines Fehlers benutzt:

- **System failure: lost in-memory updates**
- **Media failure (lost disk)**

Bewirkt die Dauerhaftigkeit (Durability) einer Transaktion.

Es existiert ein statisches Log, welches erlaubt, alle historisch abgelaufenen Transaktionsabläufe wiederherzustellen (reconstruct), sowie ein dynamisches Log, dessen Einträge nach erfolgreicher Durchführung einer Transaktion wieder gelöscht werden können.

Im Fall, dass eine Datenbank zerstört wird, kann mit Hilfe einer (nicht den neuesten Stand enthaltenden) Backup Kopie sowie des statischen Logs die zerstörte Datenbank wieder hergestellt werden, indem man gegen die Backup Kopie alle seither stattgefundene Transaktionen laufen lässt.

7.4.12 Backward Recovery

Andere Bezeichnungen sind: backout, roll back oder abort.

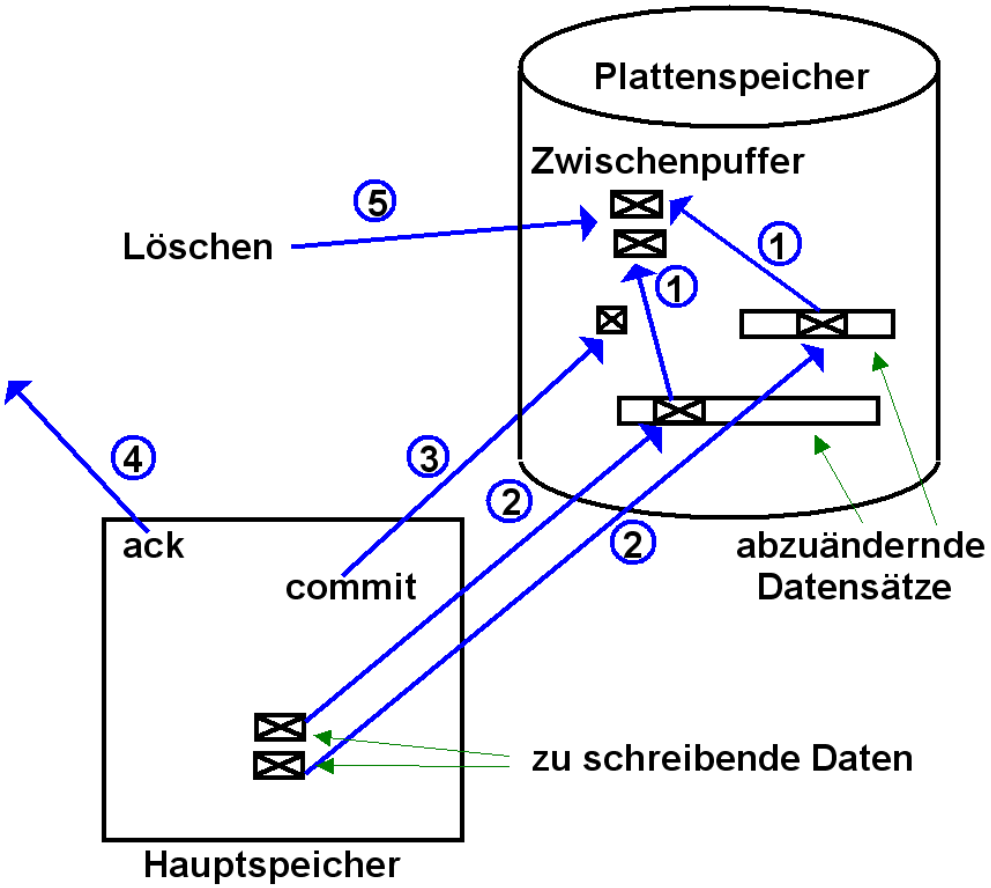


Abb. 7.4.12
Abändern von 2 Datensätzen

Zur Ermöglichung einer eventuellen Recovery finden bei jedem Write Vorgang die folgenden Schritte statt:

1. **abzuändernde Information in Puffer zwischenspeichern**
2. **Datensätze überschreiben**
3. **Commit Status festschreiben. Damit ist der ACID-relevante Teil der Transaktion abgeschlossen.**
4. **erfolgreiches Commit dem Benutzer mitteilen**
5. **Zwischenpuffer löschen**

Der Zwischenpuffer ist Teil eines dynamischen Logs.

Gewisse Daten werden von einem Transaction Processing Monitor als recoverable resource behandelt. Hierzu gehören:

- **Alle von Anwendungsprogrammen verwendeten Dateien und Datenbanken**
- **Gewisse transiente Daten und temporary storage queues**

Für diese Daten wird recovery information vom Transaction Monitor logged oder aufgezeichnet (recorded). Hierfür dient ein dynamisches Log. Wenn eine Transaktion erfolgreich abgeschlossen wird, wird die recovery Information in dem dynamischen Log gelöscht. Unabhängig davon existiert ein statisches Log, in dem alle transaktional vorgenommenen Datenänderungen durch erfolgreich durchgeführte Transaktionen festgehalten werden.

Wenn eine Transaktion nicht ausgeführt werden kann (z.B. Fehler im Anwendungsprogramm, im Datenzugriff oder aus anderen Gründen), dann führt der Transaction Monitor ein dynamic transaction backout (DTB) oder Rollback aus.

Dies macht alle bisherigen, vor dem Transaktionsabbruch durchgeführten Updates (Datenänderungen) rückgängig. DTB verhindert, dass corrupted Data von anderen Tasks oder Transaktionen benutzt werden.

Der Abbruch einer Transaktionsverarbeitung wird in der z/OS Terminologie als „abnormal end“, oder abgekürzt “abend“ bezeichnet.

7.4.13 Distributed Transaction

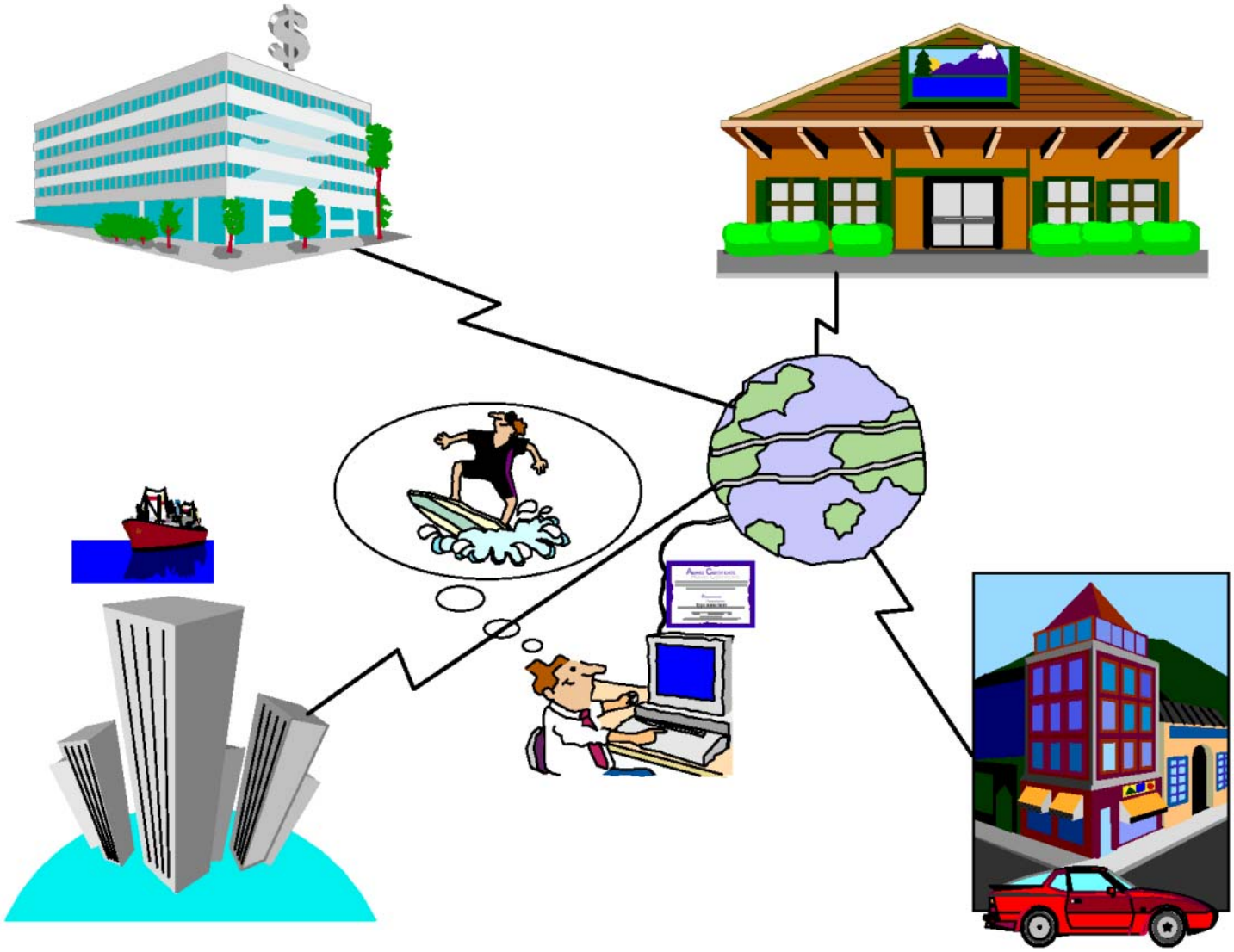


Abb. 7.4.13
Urlaubsbuchung mittels einer Distributed Transaction

Angenommen, ein Benutzer geht zu einem Reisebüro um dort eine Urlaubsreise zu buchen. Nachdem er den Flug, das Hotel und den Mietwagen selektiert hat, beschließt das Reisebüro, die Reise über das Internet zu buchen

Hierzu startet das Reisebüro eine Transaktion, welche

- **den Interessenten authentifiziert und in der Datenbank des Reisebüros speichert,**
- **die Flugreservierung im Datenbanksystem der Fluglinie aufzeichnet,**
- **die Raumreservierung im Computer des Hotels vornimmt,**
- **das Gleiche im Datenbanksystem der Autovermietung vornimmt,**
- **den gesamten Preis vom Bankkonto des Interessenten abbucht,**
- **Überweisungen auf die Bankkonten der anderen Teilnehmer der Transaktion vornimmt,**
- **für den Benutzer eine Multimedia File mit Information über das Paket an Reservierungen erstellt, und**
- **eine Bestätigung über die erfolgreiche Durchführung an den Interessenten sendet.**

Offensichtlich sind hier mehrere TP Monitore auf geographisch verstreuten Systemen involviert.

Wichtig ist, dass die gesamte Datenverarbeitung als eine atomare Transaktion durchgeführt wird. Der Interessent wird nicht zufrieden sein, wenn Hotel und Mietwagen gebucht wurden, das Geld vom Konto abgebucht wurde, aber der Flug wegen Überbuchung nicht verfügbar ist. Deshalb muss jeder Teil der Transaktion erfolgreich durchgeführt werden, oder die Transaktion insgesamt darf nicht ausgeführt werden.

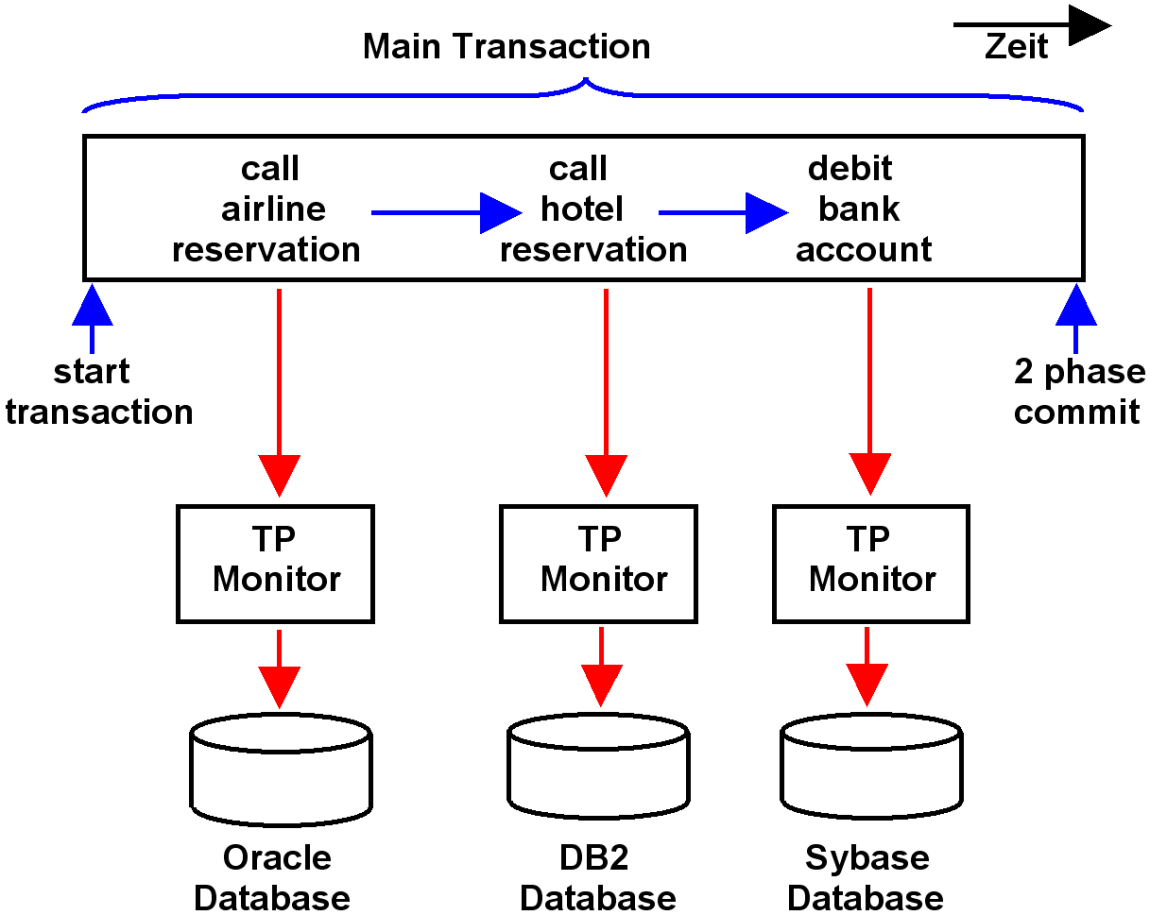


Abb. 7.4.14
Verteilte Transaktion auf mehreren Transaktionsmonitoren

In dem in Abb. 7.4.14 gezeigten Beispiel beinhaltet die Transaktion für die Urlaubsreservierung (Main Transaction) Aufrufe für drei unterschiedliche TP Monitore und drei verschiedene Datenbanken.

Dieser Fall wäre mit einer nested Transaktion lösbar. Nested Transaktionen werden in der Praxis aber nicht eingesetzt.

An deren Stelle tritt das Two-Phase Commit Protokoll, welches Bestandteil fast aller TP Monitore ist, besonders auch der unter z/OS verfügbaren TP Monitore.

Die Two-Phase Commit Management Komponente wird auch als Sync-Point Manager bezeichnet.

7.4.14 Two-phase Commit Protokoll

Das 2-Phase Commit Protokoll steuert die gleichzeitige Änderung mehrerer Datenbanken, z.B. bei der Urlaubsreise-Buchung die Änderung der Datenbanken der Fluggesellschaft, des Hotels und des Bankkontos von dem die Bezahlung der Reise abgebucht wird. Das Update der drei Datenbanken erfolgt durch unabhängige TP Monitore.

Ein Problem tritt auf, wenn eines der Updates nicht erfolgen kann, z.B. weil das Hotel ausgebucht ist. Daher sind atomare Transaktionen erforderlich

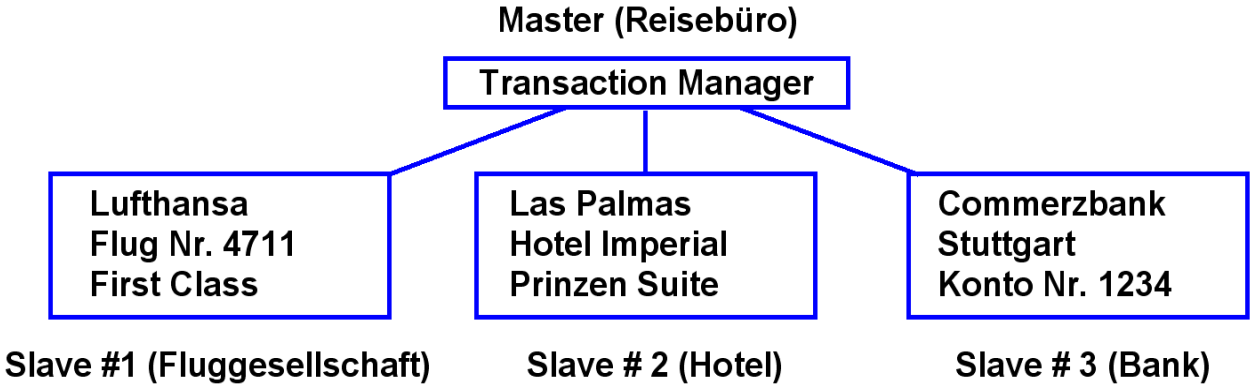


Abb. 7.4.
Master steuert Slaves

Die Konsistenz wird erreicht durch einen „Master“, der die Arbeit von „Slaves“ überwacht. Der TP Monitor des Reisebüros übernimmt die Rolle des Masters, die TP Monitore der Fluggesellschaft, des Hotels und der Bank sind die Slaves.

Der Master sendet Nachrichten an die drei Slaves. Jeder Slave markiert die Buchung als vorläufig und antwortet mit einer Bestätigung (Phase 1).

Wenn alle Slaves ok sagen sendet der Master eine Commit Nachricht, ansonsten eine Rollback Nachricht (Phase 2).

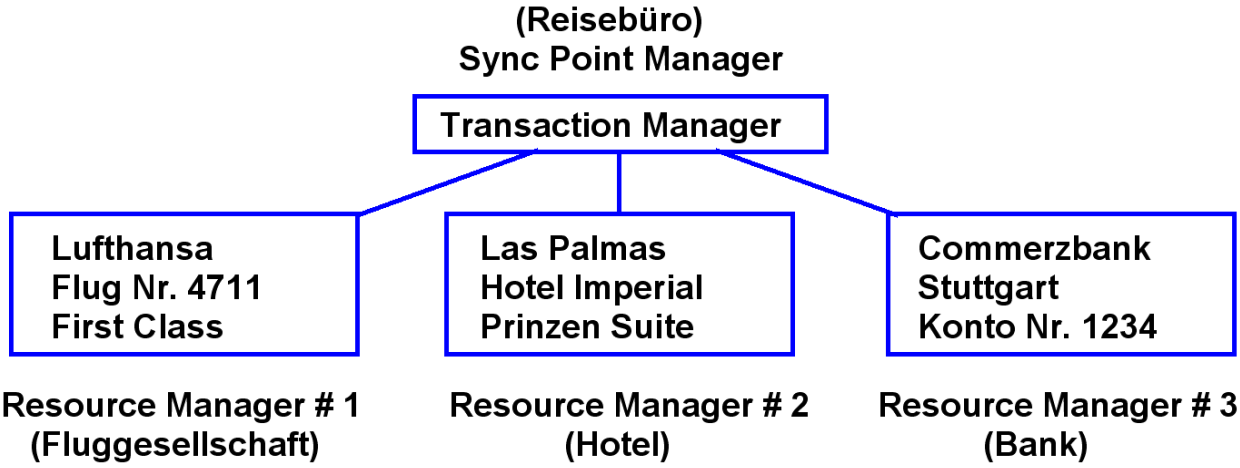


Abb. 7.4.
X/Open Terminologie

Der X/Open Standard verwendet eine andere Terminologie.

Der Master wird auch als Transaction Manager, Recovery Manager oder Sync Point Manager bezeichnet. Slaves werden als Resource Manager bezeichnet.

IBM bezeichnet den Master häufig als Sync Point Manager.

Master	Slave
Begin atomic action Send Request 1 ..... Send Request n Send „ Prepare to commit“	
	if action can be performed then begin Lock data Store initial state on disk Store requests on disk Send „OK“ end else Send „Failure“
Ende Phase 1	
if all slaves said „OK“ then send „Commit“ else send „Rollback“ Wait for acknowledgements	
	if master said commt then begin Do work Unlock data end Send „Acknowledgement“
Ende Phase 2	

Abb. 7.4.
2-Phase Commit Code Snipplet

Der Master fragt bei allen beteiligten Slaves an, ob die gewünschte Aktion machbar ist. Jeder Slave markiert die Anforderung als vorläufig und schickt eine Bestätigung an den Master (Phase 1).

Wenn alle Slaves positiv antworten sendet der Master eine commit Aufforderung, worauf die Slaves die vorläufige Änderung permanent machen. Wenn einer der Slaves seine Transaktion nicht durchführen kann sendet der Master an die Slaves eine Nachricht die vorläufige Änderung wieder rückgängig zu machen (Phase 2).

Alle Slaves bestätigen den Abschluss der 2-Phase Commit Transaktion.

In Phase 1 übergibt das Anwendungsprogramm die 2-Phase Aufforderung an den Syncpoint Manager. Dieser sendet einen PREPARE-Befehl an allen Ressource-Manager. In Reaktion auf den Befehl PREPARE, antwortet jeder an der Transaktion beteiligte Ressource-Manager an den Syncpoint Koordinator er ist bereit oder auch nicht.

Wenn der Syncpoint Manager Antworten von allen Resource Managern erhalten hat wird die Phase 2 eingeleitet. Der Syncpoint Manager sendet einen Commit oder Rollback-Befehl auf der Grundlage der vorherigen Antworten. Wenn auch nur einer der Resource Manager eine negativen Antwort gesendet hat, veranlasst der Syncpoint Manager ein Rollback der vorläufigen Änderungen in allen Resource Managern.

CICS enthält einen eigenen Syncpoint Manager, der mit dem EXEC CICS SYNCPOINT Command aufgerufen wird. Daneben und alternativ kann CICS einen generischen Recovery Manager „z/OS Resource Recovery Services“ (RRS), der von z/OS Resource Managern wie WebSphere, IMS, DB2, aber auch von CICS (an Stelle des CICS Syncpoint Managers) benutzt werden kann.

7.4.15 z/OS Resource Recovery Services

Ein Prozess, der transaktionale Anwendungen ausführt, wird als Resource Manager bezeichnet. Unter z/OS können mehrere Resource Manager gleichzeitig tätig sein. Beispiele sind die CICS, IMS und WebSphere Transaktionsmonitore, sowie Stored Procedures mehrerer Datenbanken wie IMS und DB2.

z/OS Resource Recovery Services (RRS) stellt einen Recovery Manager zur Verfügung, der das Two-Phase Commit Protokoll beinhaltet, und den jeder Resource Manager innerhalb von z/OS nutzen kann. Es ermöglicht Transaktionen, ein Update geschützter (protected) Resourcen vorzunehmen, die von unterschiedlichen Resource Managern (z.B. unterschiedlichen TP Monitoren) verwaltet werden.

RRS wird von neuen Resource Managern eingesetzt, sowie für die Nutzung neuartiger Funktionen durch existierende Resource Managers. Neu entwickelte Resource Manager Produkte unter z/OS (z.B. WebSphere Enterprise Java Beans) verwenden RRS, an Stelle einer nicht vorhandenen eigenen Two-phase Commit Protocol Komponente.

Existierende Tansaction Server wie CICS verfügen bereits über viele Funktionen, die in RRS enthalten sind, können stattdessen aber auch RRS benutzen. Vermutlich wird sich RRS in der Zukunft zu einer zentralen z/OS Komponente für die Transaktionssteuerung entwickeln.

Ursprünglich besaß jede Komponente von z/OS, die Transaktionen ausführen konnte (z.B. CICS, IMS/DC, andere) ihren eigenen Two-Phase Commit Recovery Manager. Dieser kann durch RRS ersetzt werden.

http://www.redbooks.ibm.com/abstracts/sg246980.html

7.5 Weiterführende Information

7.5.1 Die wichtigste Literatur zum Thema Transaktionen

J. Gray, A. Reuter:
"Transaction Processing".
Morgan Kaufmann, 1993.

fast 20 Jahre alt – immer noch das Standard Werk - 1070 Seiten !

P. A. Bernstein:
"Principles of Transaction Processing".
Morgan Kaufmann, 1997.

Wesentlich kürzer

J. Horswill:
"Designing and Programming CICS Applications". O'Reilly, 2000

Das beste CICS-spezifische Lehrbuch

Mark Little, Jon Maron, Greg Pavlik:
Java Transaction Processing :
Design and Implementation
Prentice Hall 2004, ISBN 0-13-035290-X

Transaktionale Anwendungen werden in Zukunft in Java entwickelt - vielleicht

7.5.2 Weitere Informationen

Das Internet enthält Tonnen von Material zu Themen wie Datenbanken, Stored Procedures und Transactionsverarbeitung.

Eine DB2 Tutorial Introduction ist verfügbar unter

http://www.youtube.com/watch?v=6noAMY3PcB8

Das folgende Video

http://www.youtube.com/watch?v=LwhkvIdombM

enthält eine detaillierte DB2 Installatonsanweisung

Videos der „Independent DB2 Users Group (idug) sind hier zu finden

http://www.idug.org/p/cm/ld/fid=85

und hier

http://www.idug.org/p/cm/ld/fid=86

Ein DB2 Stored Procedures Tutorial ist verfügbar unter

http://solutions.devx.com/ibm/skillbuilding/archives/introduction-to-db2-stored-procedures.html

oder sehr detailliert in

http://publib.boulder.ibm.com/infocenter/idm/v2r1/index.jsp?topic=/com.ibm.datatools.routines.tutorial.doc/topics/routines_introduction.html

Eine Einführung in die Transactionsverarbeitung ist hier zu finden

http://www.codeproject.com/Articles/522039/A-Beginners-Tutorial-for-Understanding-Transaction

Herr Kolja Treutlein, ehem. Student an der Uni Tübingen, erstellte ein DB2 Stored Procedures Tutorial im Rahmen seiner Diplom Arbeit. Download unter

http://www-ti.informatik.uni-tuebingen.de/~spruth/DiplArb/Treutlein.pdf

Eine Einführung in Two Phase Commit ist hier zu finden

http://docs.mongodb.org/manual/tutorial/perform-two-phase-commits/