M153

Zusammenfassung

Flavio De Roni
Uferweg 27
6014 Littau

http://www.mrf2thed.ch

27.12.2009

Inhaltsverzeichnis

1 DB-Entwurf

1.1 Allgemein/Geschichte

1.1.1 Vorteil einer Datenbank-basierten Datenhaltung?

- Mehrbenutzerunterstützung (concurrency access)
- Datenintegrität (DB nimmt der Applikation Integritätsaufgaben ab)
- Transaktionsunterstützung (Roll-Back, falls Transaction fehlschlägt)
- Datensicherheit und Datenschutz (Verschlüsselung)
- Logische Datenunabhängigkeit (Reihenfolge, Erweiterungen)

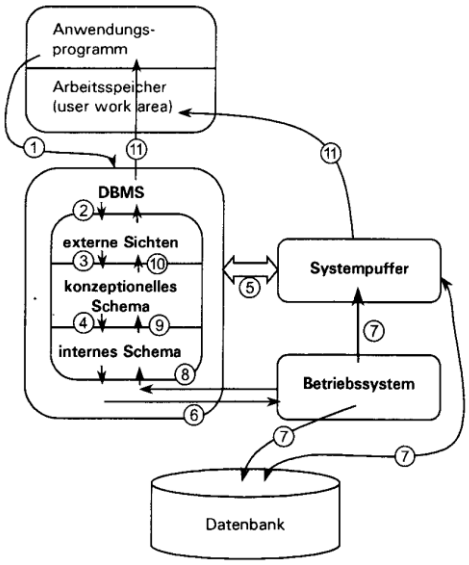
1.2 Relationales DBMS

1.2.1 Aufbau

Das Hauptmerkmal einer relationalen Datenbank ist, dass die Daten in Tabellen (tables), akademisch Relationen (relations) genannt, angeordnet sind, die weitgehend voneinander unabhängig sind. Eine Tabelle ist ein zweispaltiges Gebilde aus Zeilen (rows) und Spalten (columns).

1.2.2 Ablauf bei SQL-Anfrage

1. DBMS empfängt alle Anfragen, in denen Daten von einer externen Schicht angefordert werden
2. Syntax der Abfrage wird überprüft
3. Rechte des Benutzers werden überprüft, ob er auf die Daten zugreifen darf
4. die benötigten Datenobjekte werden ermittelt
5. die physischen Datenobjekte und Zugriffspfade werden ermittelt
6. DBMS beauftragt das Betriebssystem, die ermittelten Speicherbereiche zu lesen
7. Betriebssystem legt die gelesenen Blöcke im Systempuffer des DBMS ab
8. gewünschte Auswahl der Daten wird zusammengestellt
9. Daten werden für andere Benutzer so lange gesperrt, bis die Bearbeitung beendet wird
10. Daten werden ans Anwendungsprogramm oder den Benutzer übergeben



1.3 DBS

1.3.1 Grundaufbau

Ein DBS besteht aus den beiden Komponenten DBMS sowie der Datenbank (DB)

Datenbank:
In der Datenbank kommen die eigentlichen Daten zu liegen. Die Datenbank ist auch die Location für Index-Dateien, Log-Dateien, Transaction Logs, Meta-Daten (Informationen über die Daten).

DBMS:
Softwarekomponente, die zum DBS gehört. Liefert diverse Unterstützungsdienste

1.3.2 Aufgaben DBMS

- Zugriff auf Daten ermöglichen (Zutrittsberechtigungen)
- Verwaltung der Index-Dateien
- Transaktionsverwaltung (Begin Trans, Commit, Rollback)
- Korrektheit der Daten bei Mehrbenutzerbetrieb
  bewusste Redundanz (Replikation)
- Sicherheit (autom. Sicherung von Daten und Transaction Logs)

Was soll ein DBMS dem Programmierer bringen?

- kürzere Entwicklungszeiten durch einfachere Zugriffsmöglichkeiten (Abstraktion)
- kostengünstige Entwicklung
- stabile Anwendung durch DB-seitige Kontrolle der Integritätsregeln.

Ein DBMS muss das entsprechende Datenmodell (hierarchisch, relational, etc) unterstützen.

1.4 Vergleich der verschiedenen Datenmodelle

1.4.1 Hierarchisches Datenbankmodell

1.4.1.1 Vorteile

- schneller Datenzugriff

1.4.1.2 Nachteile

- Redundanzprobleme, da ein Eintrag unter Umständen mehrfach erscheinen muss.
- Änderungen sind schwierig zu bewerkstelligen
- Applikation muss angepasst werden, falls Änderungen an der Datenbank durchgeführt werden

1.4.2 Netzwerk Datenbankmodell

1.4.2.1 Vorteile

- jeder Datenknoten hat unmittelbaren Zugang zu jedem anderen.
- kein Knoten muss mehrfach existieren.

1.4.2.2 Nachteile

- Struktur wird schnell ziemlich undurchsichtig (komplex)
- Applikation muss angepasst werden, falls Änderungen an der Datenbank durchgeführt werden

1.4.3 Relationales Datenbankmodell

1.4.3.1 Vorteile

- Applikation muss NICHT angepasst werden, falls Änderungen an der Datenbank durchgeführt werden
- schneller Datenzugriff
- beliebig viele Sichtweisen (Views)
- virtuelle Tabellen

1.4.3.2 Nachteile

- Impedance Mismatch: In der heutigen Zeit mit den objektorientierten Programmiersprochen müssen die Daten, welche aus dem Relationalen Datenmodell stammen, immer transformiert werden.
- Mapping verlangsamt den Entwicklungsprozess

1.4.4 Objektorientiertes Datenbankmodell

1.4.4.1 Vorteile

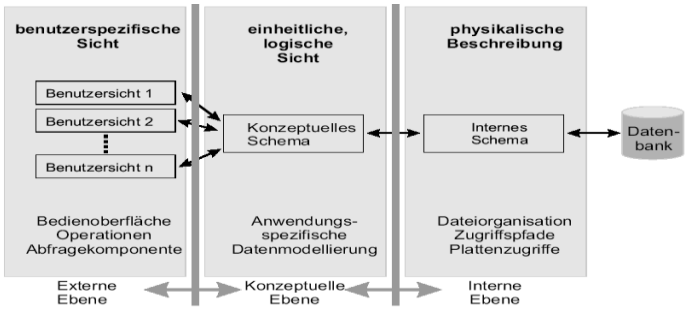
- beides ist objektorientiert, passt perfekt zueinander

1.4.4.2 Nachteile

- keine schnelle und einfache Abfragesprache wie bei SQL

1.5 Datenbankarchitektur

1.5.1 Schichten



1. **Externe Ebene (benutzerspezifische Schicht):**
   Bedienoberfläche, Operationen, Abfragekomponente

2. **konzeptuelle Ebene (einheitliche, logische Schicht):**
   Anwendungsspezifische Datenmodellierung, SQL

3. **Interne Ebene (physikalische Beschreibung):**
   Dateiorganisation, Plattenzugriffe

1.5.2 Transaktion

Stapel von SQL-Anweisungen, die als ganzes behandelt und abgearbeitet wird. Falls die Transaktion nicht korrekt ausgeführt wird, wird die Transaktion in den Ursprungszustand zurückgesetzt (Roll-Back). Transaktionen werden im Transaction Log gespeichert/protokolliert.

1.5.3 Puffer-Manager

der Puffer Manager verwaltet die im Speicher befindlichen Versionen aller physischen Festplattenseiten und bietet allen anderen Modulen Zugriff darauf.

Vorteil: Performance (Arbeitsspeicher-Zugriff ist schneller als Zugriff auf Festplatte)

1.6 Anomalien

1.6.1 Update-Anomalie

Änderungen müssen an mehreren Orten gemacht werden.

1.6.2 Lösch-Anomalie

Wenn Informationen verloren gehen

1.6.3 Einfüge-Anomalie

Wenn leere Felder entstehen, die nicht ausgefüllt werden.

2 SQL

2.1 Was ist SQL?

SQL ist eine Spezialsprache, die für den Entwurf und die Verwaltung von relationalen Datenbanken sowie die Manipulation der darin enthaltenen Daten entwickelt wurde. SQL wird durch einen ANSI-Standard definiert. Der Sprachumfang von SQL ist einer permanenten Weiterentwicklung und Standardisierung unterworfen. Derzeit relevant sind SQL-92, SQL-1999 sowie SQL-2003.
Alle Anbieter von relationalen DBMS haben ihre eigene Implementation von SQL, die sich mehr oder weniger vom Standard SQL-92 unterscheiden.
SQL-Server nennt sein erweitertes SQL „Transact SQL“, bei Oracle spricht man von „PL/SQL“.
Wenn man die Datenbank auf unterschiedlichen Datenbanksystemen verwenden will, sollte man auf die Verwendung von erweiterten Komponenten „z. B. Transact-SQL) verzichten. **Eine enge Anlehnung an den ANSI-Standard ist besonders dann wichtig, wenn die DB-Applikation auf unterschiedlichen Datenbanksystemen funktionieren soll.**
Obwohl SQL keine Allzweckprogrammiersprache ist, enthält sie alles, was man benötigt, um relationale Datenbanken zu erstellen, zu verwalten, zu sichern und zu schützen

SQL (Structured Query Language) besteht aus folgenden Hauptbestandteilen:

- **DDL (Data Definition Language)**
  DDL stellt alles zur Verfügung, was benötigt wird, um eine Datenbank und deren Elemente wie Relationen und Beziehungen zu definieren, zu ändern und zu löschen.
- **DML (Data Manipulation Language)**
  DML verfügt über Befehle, mit welchen man Daten (Informationen) in eine Relation einfügen und löschen kann.
- **DQL (Data Query Language)**
  DQL verfügt über Befehle, mit welchen man Informationen nach den verschiedensten Kriterien aus einer Datenbank abrufen (betrachten) kann.
- **DCL (Data Control Language)**
  DCL verfügt über Befehle, mit denen man die Datenbank vor unerwünschten Einflüssen schützen kann (Berechtigungen)

2.2 Entwicklungsgeschichte von SQL

- **ca. 1975**
  SEQUEL = Structured English Query Language, Vorläufer von SQL wird für das Projekt System R von IBM entwickelt.
- **1981**
  SQL gelangt mit SQL/Data Systems erstmals durch IBM auf den Markt.
- **1986**
  SQL1 wird von ANSI als Standard verabschiedet.
- **1987**
  SQL1 wird jetzt auch von ISO als Standard verabschiedet und 1989 nochmals überarbeitet.
- **1992**
  Der Standard SQL2 bzw. SQL-92 wird von der ISO verabschiedet.
- **1999**
  SQL3 bzw. SQL-1999 wird verabschiedet.
- **2003**
  SQL-2003 wird von der ISO als Nachfolger des SQL-1999 Standards verabschiedet.

2.3 SQL-DDL

2.3.1 SQL-Datentypen

Numerische Datentypen (Ganzzahlen)
TINYINT < SMALLINT < INTEGER

Numerische Datentypen (Fliesskommazahlen)
FLOAT < DOUBLE PRECISION

Numerische Datentypen (Festkommazahlen)
NUMERIC / DECIMAL (Präzision, Skalierung)

Datumswerte

- DATE (MySQL)

- DATETIME
- SMALLDATETIME (SQLServer)

Zeichen
CHAR(Länge), VARCHAR(Länge), TEXT, BLOB

2.3.2 Datenbank erstellen

CREATE DATABASE DbName;

2.3.3 Datenbank aktivieren

USE DbName;

2.3.4 Datenbank löschen

DROP DATABASE DbName;

2.3.5 Tabellen erstellen

```
CREATE Table TKunde(
KNr INTEGER NOT NULL
AUTO_INCREMENT,
Nachname VARCHAR(30),
u_KNr(KNr),
PRIMARY KEY(KNr)
);
```

Nach SQL-92 können folgende Konsistenzbedingungen für eine Tabelle festgelegt werden:

- PRIMARY KEY
- UNIQUE (Kandidatenschlüssel)
- FOREIGN KEY
- CHECK (Einschränkungen des Wertebereichs)
- NULL/NOT NULL (Verbot von Nullmarken in Spalten)
- CHECK (Spaltenübergreifende Integritätsbedingungen)
- ASSERTION (Tabellenübergreifende Integritätsbedingungen)

Bestehende Tabellen ändern
ALTER TABLE Tabellenname ADD Test VARCHAR(9) DEFAULT('Unbekannt')

Foreign Key (Fremdschlüssel)

- Die Veränderung kann ganz verboten werden → NO ACTION
- Die Veränderung kann an den Fremdschlüssel weitergegeben werden → CASCADE
- Die Veränderung kann den Fremdschlüsselwert auf NULL setzen → SET NULL
- Die Veränderung kann den Fremdschlüsselwert auf einen Defaultwert setzen → SET DEFAULT

Referenzielle Datenintegrität
Unter Datenintegrität versteht man die **Fehlerfreiheit, Genauigkeit und Zuverlässigkeit,** d.h. die **Qualität** von Daten. Insbesondere erkennt man die Genauigkeit bei Änderung von Daten.
Die referenzielle Datenintegrität definiert nun die Beziehungen zwischen Datenzeilen über mehrere Tabellen. In SQL-Server basiert diese Durchsetzung der referenziellen Integrität auf Beziehungen zischen PK und FK bzw zwischen FK und KK.
→ FOREIGN KEY-Einschränkungen

2.3.6 Tabellenstruktur bearbeiten

Spalten hinzufügen
ALTER TABLE Kunden ADD COLUMN Titel VARCHAR(10);

Spalten löschen
ALTER TABLE Kunden DROP COLUMN Titel;

2.3.7 Tabellen löschen

DROP TABLE Kunde;

2.3.8 Referenzielle Datenintegrität

Ein Fremdschlüssel muss immer auf einen gültigen Primärschlüssel verweisen, oder muss NULL sein.

Definition von MS TechNet:
Die referenzielle Integrität stellt die tabellenübergreifende Konsistenz von Schlüsseln sicher. Eine solche Konsistenz erfordert, dass keine Verweise auf nicht vorhandene Werte bestehen und dass beim Ändern eines Schlüsselwertes alle Verweise auf diesen Wert in der Datenbank konsistent geändert werden.

2.4 SQL-DQL

M153_SQL.sql

3 Transaktionen, Concurrency, Sperrmechanismen

3.1 Transaktionen

Sobald ein Anwender auf eine Datenbank (lesend oder schreibend) zugreift, wird eine Transaktion gestartet. Eine Transaktion kann aus einem simplen Select-Statement oder einer Reihe von komplexen Update-Befehlen bestehen.
Eine Transaktion ist nur dann erfolgreich, wenn alle Einzelschritte erfolgreich waren.

Unter einer Transaktion versteht man eine logische Operation, die entweder in ihrer Gesamtheit erfolgreich verlaufen muss, oder in ihrer Gesamtheit scheitert. Eine teilweise Ausführung der Transaktion ist nicht zulässig (Inkonsistenzen).

➔ Eine Transaktion versetzt die Datenbank von einem konsistenten Zustand in einen anderen konsistenten Zustand.

➔ DBMS muss Transaktionen kontrollieren!

3.1.1 Transaktionen detaillierter betrachtet

Man kann eine Datenbank nur durch Befehle, die Daten ändern, in einen inkonsistenten Zustand bringen (also nur durch die Befehle UPDATE, INSERT oder DELETE).

Wenn man von einem Konto auf ein anderes 100000 Franken überweisen würde, und dazwischen der Strom ausfällt, wären ohne Kontrolle die 100000 Franken verloren. Dies darf natürlich nicht passieren.
Daher kann das DBMS, wenn es gestartet wird, offene Transaktionen erkennen. Gibt es während des Datenbankstarts offene Transaktionen, so werden die Änderungen, die im Rahmen der Transaktion durchgeführt wurden, zurückgenommen und die Datenbank wird in den Zustand vor der offenen Transaktion versetzt.

3.1.2 Eigenschaften einer Transaktion

Wie bereits erwähnt, überführt eine Transaktion eine Datenbank von einem konsistenten Zustand in einen anderen konsistenten Zustand.

ACID

- **Atomarität (Atomicity)**
  Unter Atomarität versteht man, dass eine Transaktion stets als Ganzes betrachtet werden muss, das heisst, es ist nicht zulässig, dass einige Teiloperationen ausgeführt werden und andere nicht. Sobald eine einzige Teiloperation fehlschlägt, schlägt die gesamte Transaktion fehl.

- **Konsistenz (Consistency)**
  Nach dem Abschluss einer Transaktion, unabhängig davon, ob sie erfolgreich war oder nicht, muss die Datenbank in einem konsistenten Zustand vorliegen.

- **Isolation (Isolation)**
  Transaktionen müssen isoliert voneinander stattfinden. Sobald eine Transaktion Daten verändert, darf keine andere Transaktion auf diese Daten zugreifen, bis die erste Transaktion beendet wurde.

- **Dauerhaftigkeit (Durability)**
  Wurde eine Transaktion erfolgreich durchgeführt, so werden alle in dieser Transaktion an den Daten vorgenommenen Änderungen dauerhaft in der Datenbank gespeichert.

3.1.3 Transaktionsverwaltung mit SQL

in SQL kann man mit Hilfe der Befehle COMMIT und ROLLBACK Transaktionen steuern.
Die Änderungen, die man in dieser neuen Transaktion getätigt hat, werden dann entweder mit COMMIT bestätigt, das heisst, die Transaktion wird beendet und die Änderungen werden in der Datenbank festgeschrieben, oder man kann mit Hilfe des Befehls ROLLBACK die getätigten Änderungen wieder zurücknehmen.

3.1.4 Das Transaktionsprotokoll

Damit das Datenbanksystem den Überblick über all diese Transaktionen behält, führt es intern ein Transaktionsprotokoll mit, in dem festgehalten wird, welche Transaktionen welche Änderung an der Datenbank vorgenommen hat.

ACHTUNG:
Das Transaktionsprotokoll ist eine der kritischsten Komponenten des ganzen Datenbanksystems. Wird das Transaktionsprotokoll zerstört oder beschädigt, so kann dies dazu führen, dass die ganze Datenbank nicht mehr verwendet werden kann. Aus Geschwindigkeitsgründen lagern manche Datenbanksysteme das Transaktionsprotokoll in den Hauptspeicher des Rechners aus, was das ganze System extrem empfindlich gegen Stromunterbrechungen macht.

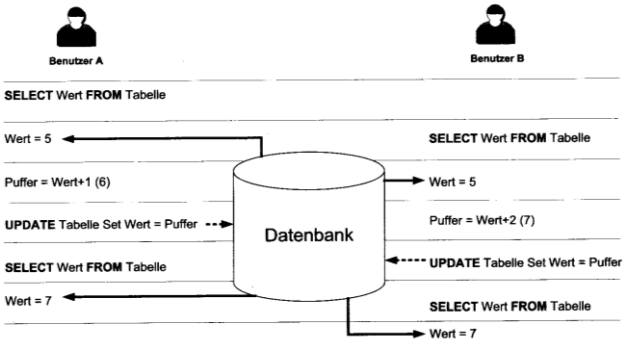
3.2 konkurrenzierende Zugriffe

Probleme (Synchronisationsprobleme) im Mehrbenutzerbetrieb sind:

1. Lost Updates
2. Dirty Reads
3. Nonrepeatable Reads
4. Phantome

3.2.1 Lost Updates

Das Lost-Update-Problem tritt auf, wenn zwei Benutzer in engen zeitlichen Abständen auf die Datenbank zugreifen und dort Änderungen vornehmen.

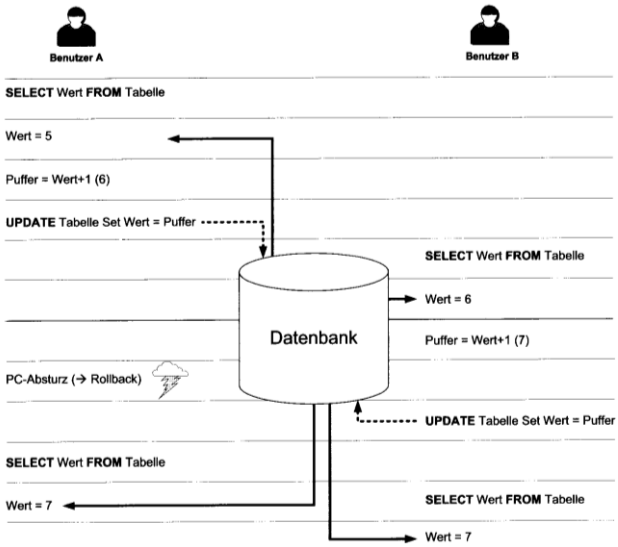


Die Änderung von Benutzer A wurde verworfen (verloren → lost update). Dieses Phänomen ist auch als „wer zuletzt speichert, gewinnt“ bekannt.

Damit ein derartiges Verhalten der Datenbank verhindert wird, muss der Update-Befehl einen Sperrmechanismus auslösen, der verhindert, dass es für den Benutzer B möglich ist, den Wert in der Tabelle zu ändern, nachdem Benutzer A ihn bereits geändert hat.

3.2.2 Dirty Reads

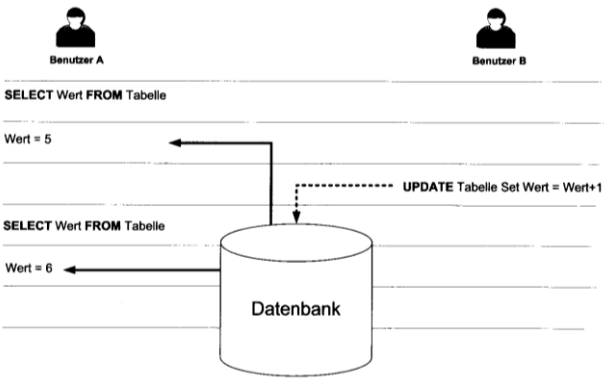
Das Dirty-Read-Problem tritt dann auf, wenn auch unbestätigte Transaktionen berücksichtigt werden und zwischenzeitlich ein Rollback (z.B. durch einen Rechnerabsturz) durchgeführt wurde.



Das Dirty-Read-Problem kann nur umgangen werden, wenn das Datenbanksystem beim Zurückliefern von Werten niemals unbestätigte Transaktionen berücksichtigt.

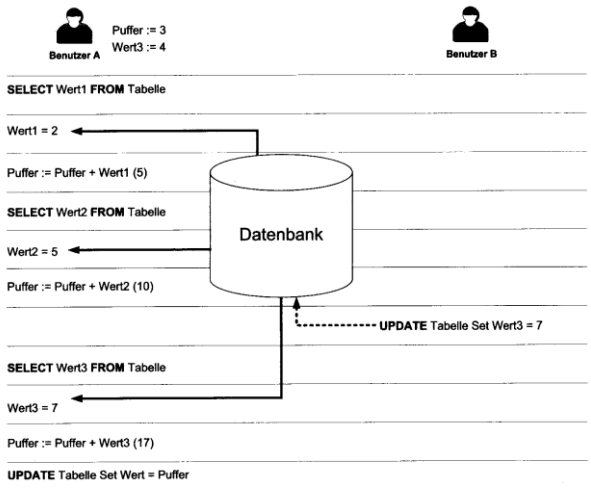
3.2.3 Nonrepeatable Reads

Wird innerhalb einer Transaktion ein Wert mehrfach aus der Datenbank ausgelesen, so muss dieser Wert gleich sein, auch wenn er inzwischen von anderen Benutzern verändert wurde.



3.2.4 Phantome

Das Problem der Phantome tritt auf, wenn nacheinander Werte eingelesen werden, die verrechnet werden, und wenn zwischenzeitlich einer dieser Werte von einem anderen Benutzer geändert wurde.



3.2.5 Zusammenfassend

Für ein Datenbankmanagementsystem ist es wichtig, die einzelnen Benutzer über Transaktionen voneinander zu trennen, damit die vier vorgestellten Synchronisationsprobleme nicht auftreten können. Ausserdem muss ein Datenbank-managementsystem Probleme, wie z.B. dem Absturz eines Clientrechners erkennen, und die nicht bestätigten Aktionen wieder rückgängig machen.

3.3 Sperrmechanismen

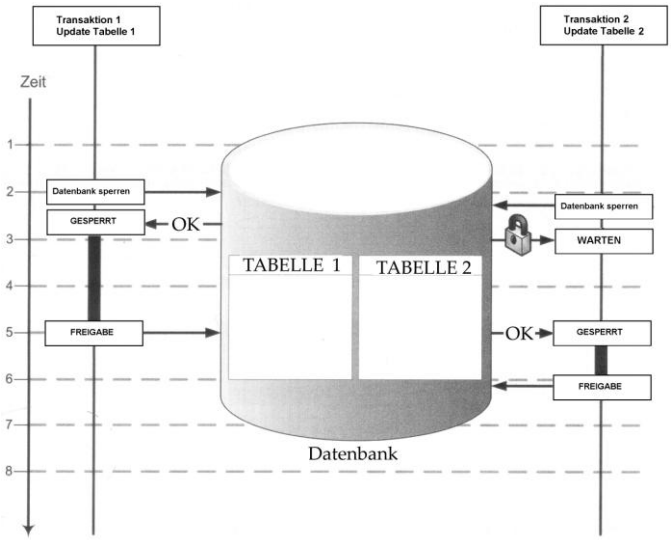
Sperren (engl. Locks) garantieren auf eine Transaktion den exklusiven Zugriff auf einen bestimmten Teil der Datenbank, das heisst, wenn ein Teil der Datenbank durch eine Transaktion gesperrt ist, so kann keine andere Transaktion auf diesen Teil der Daten zugreifen.
Moderne Datenbanksysteme verwalten Sperren von selbst, so dass sich der Anwendungsprogrammierer oder Datenbankdesigner überhaupt nicht manuell um das Einrichten und Löschen von Datenbanksperren kümmern muss.

3.4 Granularität

Unter der Granularität von Sperren versteht man, auf welcher Ebene eine Sperre die Datenbank sperrt.

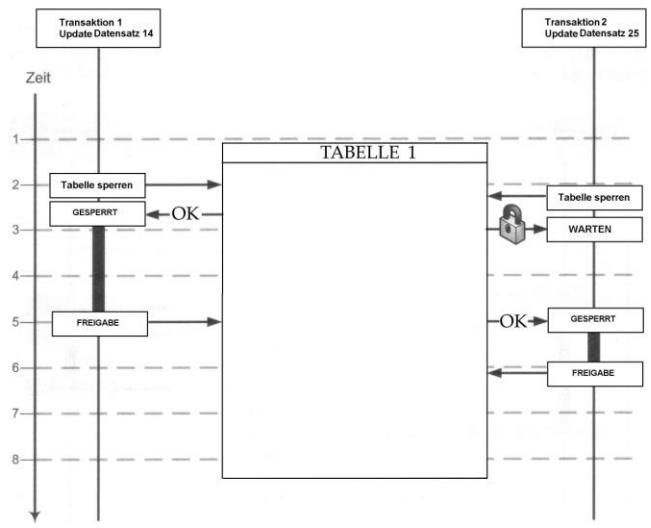
3.4.1 Sperren auf Datenbankebene

Wird eine Sperre auf Datenbankebene errichtet, so ist die gesamte Datenbank gesperrt, das heisst, das andere Transaktionen gar nicht auf die Datenbank zugreifen können, selbst wenn sie ganz andere Daten bearbeiten möchten als die Transaktion, welche die Datenbank gesperrt hat.



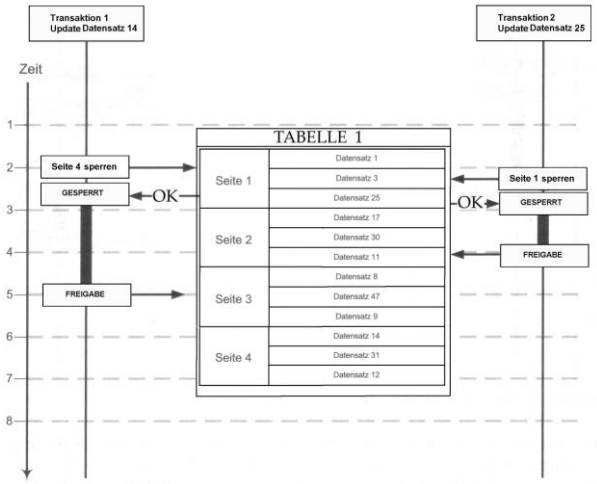
3.4.2 Sperren auf Tabellenebene

Im Gegensatz zur Sperre auf Datenbankebene wird bei einer Sperrung auf Tabellenebene nur eine komplette Tabelle gesperrt. Der Zugriff auf verschiedene Tabellen so wie in der vorherigen Abbildung ist hier problemlos möglich.



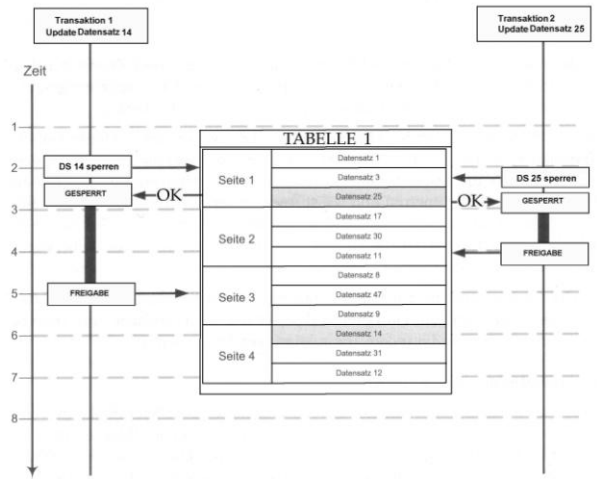
3.4.3 Sperren auf Seitenebene

Die Speicherverwaltung einer Datenbank ist ähnlich aufgebaut wie die Speicherverwaltung einer Festplatte, d.h., der Gesamtspeicher ist in so genannte Seiten (Pages) von einheitlicher Grösse (z.B. 4,8,16 KByte) aufgeteilt.



3.4.4 Sperren auf Datensatzebene

Noch flexibler als die Sperrung auf Seitenebene ist natürlich die Sperrung auf Datensatzebene, das heisst, Transaktionen kommen sich nur noch dann in die Quere, wenn sie gleichzeitig auf denselben Datensatz zugreifen möchten.



3.4.5 Sperren auf Feldebene

Die flexibelste aller Sperren stellt die Sperre auf Feldebene dar. Hier stören sich gleichzeitige Transaktionen nur noch dann, wenn sie gleichzeitig auf dasselbe Feld desselben Datensatzes zugreifen möchten. Allerdings ist auch der Administrationsaufwand des DBMS höher, da für jedes Feld jedes Datensatzes jeder Tabelle eine Sperre verwaltet werden muss. Darum hat die Sperre auf Feldebene auch keine grosse Bedeutung in professionellen Datenbanksystemen

3.5 Sperrtypen

Unabhängig von der Granularität der Sperre ist der Typ der Sperre, der bestimmt, wie die Sperre funktioniert. Über den Sperrtyp wird festgelegt, was eine andere Transaktion mit einem gesperrten Teil der Datenbank machen darf. Man unterscheidet.

- Binäre Sperren
- Exklusive / Nichtexklusive Sperren

3.5.1 Binäre Sperren

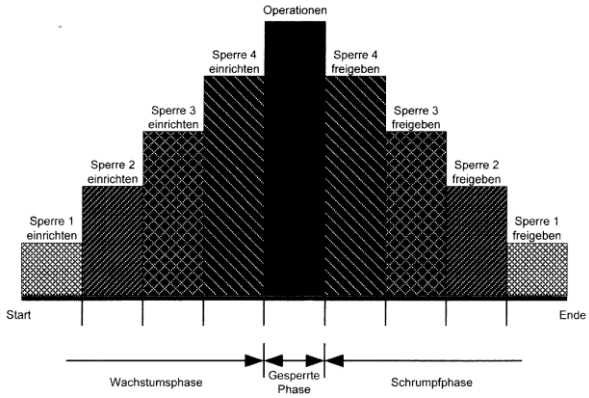
Unter einer binären Sperre versteht man eine Sperre, die zwei Zustände haben kann: gesperrt oder offen.
Werden binäre Sperren verwendet, so kann ein gesperrtes Objekt von einer anderen Transaktion gar nicht verwendet werden.

3.5.2 Exklusive/nicht exklusive Sperren

Fordert eine Transaktion eine exklusive Sperre an, so wird das zu sperrende Objekt exklusiv für diese eine Transaktion gesperrt und keine andere Transaktion kann auf das Objekt zugreifen.
Eine Nichtexklusive Sperre lässt den Lesezugriff auf Das Datenbankobjekt parallel zu, da ja hier kein Konfliktpotenzial gegeben ist.

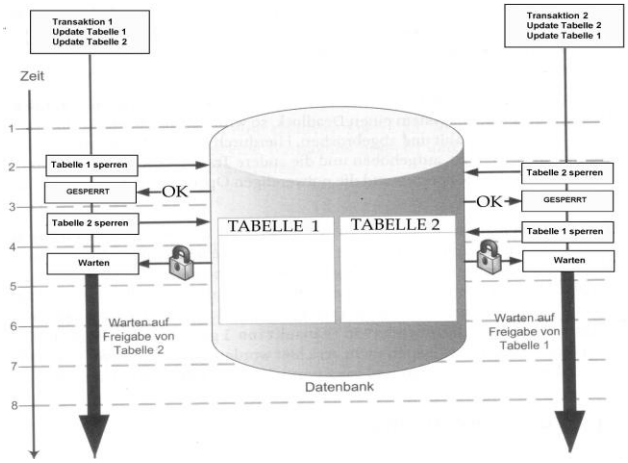
3.5.3 Zwei-Phasen-Locking

Um zu verhindern, dass Transaktionen nicht mehr seralisierbar sind, wurd das Zwei-Phasen-Locking-Protokoll entwickelt. Hierbei gibt es eine Wachstumsphase, in der alle notwendigen Sperren errichtet werden, und eine Schrumpfphase, in der diese Sperren wieder gelöscht werden.



3.5.4 Deadlocks

Unter einem Deadlock versteht man eine Situation, die auftritt, wenn zwei Transaktionen dieselben Ressourcen benötigen, diese aber in verschiedener Reihenfolge belegen und somit jede der beiden Transaktionen darauf wartet, dass die jeweils andere Transaktion die belegte Ressource wieder freigibt.



3.6 Single point of failure

Unter einem Single point of failure versteht man diejenigen Komponenten eines Systems, die nach einem Ausfall den Komplettausfall eines Systems nach sich ziehen.

4 Trigger

4.1 Übersicht

Trigger sind Funktionen, die das Datenbankmanagement automatisch, beim Eintreffen von definierten Ereignissen, ausführt.

Ein Trigger besteht aus SQL-Anweisungen und gehört immer zu genau einer Tabelle. Pro Tabelle kann es aber mehrere Trigger geben.

Trigger sind viel flexibler als die automatischen Lösch- und Aktualisierungsfunktionen.
Mit Trigger könnte man z.B. eine Beziehung so erstellen, dass das Löschen eines Kunden nur erlaubt ist, wenn der Kunde in den letzten zwei Jahren weniger als 10 Artikel bestellt hat. Wird ein Kunde gelöscht, sollen aber trotzdem alle Bestelldetails gelöscht werden.

Trigger werden in den folgenden Fällen angewendet:

- Zur Wahrung der Datenintegrität, die über die einfach Aktualisierung von Referenzen hinausgeht.
- Zur Aktualisierung von Summen
- Zur Aktualisierung von Spalten, die Rechenergebnisse beinhalten.
- Zur Pflege von Revisionsdatensätzen
- Zur Durchführung von refrenziellen Operationen wie kaskadierendes Löschen etc.
- Zur Auslösung externer Aktionen (E-Mail auslösen bei Unterschreitung des Mindestbestandes.

Ein Trigger gehört immer zu einer Tabelle und reagiert auf eines oder mehrere der folgenden Ereignisse:

- INSERT
- UPDATE
- DELETE

Pro Tabelle und Ereignis kann man mehrere Trigger programmieren. Die Ablauf-Reihenfolge ist jedoch nicht voraussagbar. ➔ Es kann immer wieder anders herauskommen. Bei zeitkritischen Aktionen sollte man nur ein Trigger verwenden!

4.2 Kopf eines Triggers

```
CREATE  TRIGGER PersonMutation ON  Personen
FOR  INSERT, UPDATE
AS
Begin
        .....
        (SQL- BEFEHLE ODER TRANSACT-SQL-BEFEHLE)
        .....
END
```

4.3 Die zwei „Schattentabellen“ INSERTED und DELETED

Jede Tabelle besitzt zwei Schattentabellen, welche die Bezeichnungen inserted und deleted tragen. Die Schattentabellen sind von der Struktur her identisch aufgebaut wie die „Original“-Tabelle, nur sind sie normalerweise leer.

4.3.1 Einfügevorgang

INSERT INTO Personen VALUES (5, „Decker“, „Harald“)

Bevor die Daten effektiv eingefügt werden, werden sie in die Inserted-Schattentabelle der Tabelle Personen geschrieben.

4.3.2 Löschvorgang

DELETE Personen WHERE PNr = 2

Bevor die Daten in der Tabelle Personen effektiv gelöscht werden, wird eine Kopie der Daten in die deleted-Schattentabelle geschrieben.

4.3.3 Änderungsvorgang

UPDATE Personen SET Name = „Feldermann“ WHERE PNr = 1

Bevor die Daten in der Tabelle Personen effektiv geändert werden, wird der alte Zustand der betroffenen Daten (Zeilen) in die Schattentabelle DELETED kopiert. Der neue Zustand der Daten wird in die Schattentabelle INSERTED kopiert.

4.3.4 grundsätzliche Idee

Bevor eine Aktion definitiv ausgeführt wird, werden die zu löschenden, die einzufügenden oder zu ändernden Informationen in die inserted oder deleted Schattentabellen geschrieben. Die Originaltabelle wird zu diesem Zeitpunkt noch nicht geändert. Nun wird, falls existent ein entsprechender Trigger aufgerufen. Wenn der Trigger kein Abbruch (ROLLBACK) befiehlt, können die Änderungen ausgeführt werden und die inserted und deleted Schattentabellen werden wieder geleert.

4.4 Code


Block 4_A5.sql


Block 4_A5_Trigger.sql


RepTrigger_CreateTable.sql


RepTrigger_TRIGGERS.sql

5 Verteilte Datenbanken

5.1 Einstieg

In den 70er-Jahren wurden Datenbanksysteme generell so aufgebaut, dass es ein zentrales System gab (meist ein Mainframe-Rechner), der das Datenbankmanagement-System ausführte und die Daten verwaltet hat. Auf diese zentrale Datenbank wurde mit vielen Terminal-Arbeitsplätzen zugegriffen.

Ein Datenbanksystem, das so aufgebaut ist, hat 2 entscheidende Nachteile:

1. „single point of failure“. Es gibt einen zentralen Punkt, und wenn der ausfällt, legt er das ganze System lahm.
2. Informationen können bei Bedarf nicht schnell geliefert werden.

Die ändernden Marktbedingungen in den 80er Jahren stellten zwei neue Hauptforderungen an moderne Datenbanksysteme:

1. Ad-hoc-Abfragen, so dass zeitnahe Entscheidungen getroffen werden können
2. DBS mussten den dezentralen Unternehmensstrukturen Rechnung tragen.

5.2 Vor- und Nachteile verteilter Transaktionen

Durch eine Verteilung der Gesamtdatenmenge auf verschiedene Standorte kann man erreichen, dass die benötigten Daten näher am Anwender sind und somit die Datenzugriffszeiten erheblich reduzieren.

Wird an jedem Standort ein Datenbankserver aufgestellt, der lokale Anfragen beantworten kann, und müssen nur Anfragen nach Kunden anderer Filialen an die Zentrale geschickt werden, so reduziert sich der Netzwerk-Traffic über die Langstrecke erheblich.

Durch die lokale Speicherung der Daten an den einzelnen Standorten werden die Abfragen schneller ausgeführt.

Ausserdem verringert sich auch die zu durchsuchende Datenmenge, da nur noch die relevanten Daten am jeweiligen Standort in den dortigen Datenbanken vorhanden sind. Grössere Datenverarbeitungsaufgaben werden parallel mit einer kleineren Datenmenge in den einzelnen Standorten abgewickelt, was die Zeit drastisch reduzieren kann.

Jedoch hat ein verteiltes Datenbanksystem auch einige Nachteile:

- komplexer in der Planung, Implementierung, Wartung
- Daten an den verschiedenen Orten müssen in der Zentrale zusammengebracht werden. → Konflikte müssen gelöst werden.

5.3 verteilte Datenhaltung vs. verteilte Datenbanken

Man muss sehr genau zwischen verteilter Datenverarbeitung und verteilten Datenbanken unterscheiden.

5.3.1 verteilte Datenverarbeitung

Bei der verteilten Datenverarbeitung verteilt sich der gesamte Datenverarbeitungsprozess über mehrere Rechner, d.h., bestimmte Teilaufgaben der Datenverarbeitung werden auf Rechnern an verschiedenen Orten erledigt.

5.3.2 verteilte Datenbanken

Bei der verteilten Datenbank hingegen werden Teile der Datenbank an verschiedenen Standorten gespeichert. Die Datenbank selbst präsentiert sich dem Benutzer aber als Einheit, d.h., er muss nicht wissen, welche Daten auf welchem Server gespeichert sind (Transparenz)

5.4 Komponenten eines verteilten Datenbanksystems

Ein verteiltes Datenbanksystem wird von einem Distributed Data Management System (DDBMS) verwaltet. Das DDBMS steuert sowohl die verteilte Datenverarbeitung als auch die verteilte Datenhaltung der verteilten Datenbank.
DDBMS muss in der Lage sein:

- zu erkennen, welche Teile der DB-Abfrage lokal bearbeitet werden können und welche über das Netzwerk ausgeführt werden müssen.
- Optimierung der Datenbankabfrage durchführen, um den optimalen Zugriffsweg zu ermitteln.
- physikalische Standorte der Daten ermitteln und herausfinden, ob die Daten auf dem lokalen Server gespeichert sind oder auf einem entfernten System,
- Sicherheitsfunktionen implementiert
- Backup and Recovery

Das DDBMS sorgt dafür, dass eine verteilte Datenbank, die sich aus mehreren Fragmenten zusammensetzt, für einen Client wie eine einzige logische Datenbank aussieht.

5.4.1 verteilte Datenverarbeitung und zentrale Datenhaltung

Bei der verteilten Datenverarbeitung mit zentraler Datenhaltung werden die Daten zentral auf einem Server gespeichert, während die Datenverarbeitung parallel und verteilt auf mehreren Computern abläuft, die mit dem Server über ein Netzwerk verbunden sind- Es gibt zwei Arten der zentralen Datenhaltung, einerseits die zentrale Datenhaltung im Dateisystem oder die zentrale Datenhaltung auf einem Datenbankserver.

5.4.2 Vollständig verteiltes Datenbanksystem

Beim vollständig verteilten Datenbanksystem ist, wie der Name schon andeutet, sowohl die Datenhaltung als auch die Datenverarbeitung verteilt. Man unterscheidet zwischen homogenen und heterogenen verteilten Datenbanksystemen.

- homogen:
  jeder Computer, der zur Datenhaltung dient, führt dasselbe DBMS aus

- heterogen:
  auf verschiedenen Datenbanksystemen werden DBMS unterschiedlicher Hersteller betrieben

5.5 Transparenz beim Datenzugriff

Das verteilte Datenbanksystem wird dem Benutzer als eine logische Datenbank präsentiert (Transparenz → Verbergen der phys. Aufbaus.)

5.5.1 Transparente Datenverteilung

Die physikalische Datenverteilung des verteilten Datenbanksystems soll dem Anwender transparent sein, das heisst, der Datenbankanwender weiss gar nicht, wo sich die Daten, mit denen er arbeitet, befinden. Leider ist eine vollständige Transparenz nicht immer gegeben.

5.5.2 Transparentes Transaktionsmanagement

Es ist wichtig, dass bei verteilten Datenbanksystemen das Transaktionsmanagement transparent ist. Werden verteilte Transaktionen durchgeführt (eine Transaktion ändert Daten an mehreren Standorten), so muss gewährleistet sein, dass durch diese Änderung die Integrität der Datenbank nicht verletzt wird.

→ Ein verteilte Transaktion kann nur dann erfolgreich sein, wenn die Änderungen an allen Standorten erfolgreich durchgeführt werden konnten!!!

5.6 Datenfragmentierung

einzelne Tabellen können über mehrere Standorte verteilt sein. In diesem Zusammenhang spricht man von Datenfragmentierung. Ein Teilstück dieser Tabelle wird als Datenfragment bezeichnet. Die Information darüber, welche Tabelle wie fragmentiert ist und welche Datenfragmente sich wo befinden, wird im „Verteilten Datenbank Dictionary“ gespeichert.

5.6.1 horizontale Fragmentierung

Bei der horizontalen Fragmentierung wird die Tabelle anhand der Datensätze aufgeteilt, das heisst, an jedem Standort ist die Tabellenstruktur gleich – die Tabellenfragmente unterscheiden sich nur durch dien an den Standorten gespeicherten Datensätze.

➔ **Jede Datenfragment hat dieselbe Anzahl Spalten**

5.6.2 vertikale Fragmentierung

Jedes Tabellensegment enthält alle Datensätze, aber nicht alle Felder jedes Datensatzes.

Bei einer vertikalen Fragmentierung dürfen die Datenfragmente unterschiedliche Felder besitzen, es müssen aber gleich viele Datensätze in beiden Tabellen enthalten sein. Zwischen den Datenbanksegmenten besteht eine 1:1-Beziehung.

5.6.3 gemischte Fragmentierung

sowohl vertikale als auch horizontale Fragmentierung…

5.7 Replikation

Unter dem Begriff Datenbankreplikation versteht man die Speicherung von Kopien der Daten an verschiedenen Standorten. Durch diese redundante Speicherung kann die Effizienz des verteilten Datenbanksystems erheblich gesteigert werden, da Daten lokal an einem Standort bereitgestellte werden können, die ansonsten über WAN-Verbindungen abgerufen werden müssten.

6 Glossar

6.1 Normalformen

Um Anomalien zu beseitigen, bedient man sich der Normalisierung. Es gibt verschiedene Regeln (Normalformen), welche man einhalten muss. Bekannt sind die Normalformen 1,2,3, BC, 4, 5. Wir verwenden nur die Normalformen 1,2 und 3.
Die Beispiele orientieren sich an dieser nicht normalisierten Datenstruktur:

6.1.1 nicht normalisierte Datenstruktur

Projekttätigkeiten							
PersNr	Name	Vorname	AbtNr	Abtname	**ProjNr**	Projektname	Tätigkeit
0004	Richter	Hans	3	Verkauf	1,2,3	Amazonas, Colorado, Buzibach	Sachbearbeiter, Projektleiter, Sachbearbeiter

6.1.2 1. Normalform

Eine Relation(Tabelle) befindet sich in der 1. Normalform (1NF), wenn:

- Sie ein zweidimensionales Gebilde aus Zeilen und Spalten ist
- Für jedes Attribut nur ein Wert eingetragen ist
- Sich in jeder Spalte nur Daten befinden, die einem Attribut entsprechen und das Attribut nur einmal in der Relation vorkommt
- Sich in jedem Datensatz nur Daten befinden, die zu einem Objekt gehören und jeder Datensatz nur einmal vorkommt

Projekttätigkeiten							
PersNr	Name	Vorname	AbtNr	Abtname	**ProjNr**	Projektname	Tätigkeit
0004	Richter	Hans	3	Verkauf	1	Amazonas	Sachbearbeiter
0004	Richter	Hans	3	Verkauf	2	Colorado	Projektleiter,
0004	Richter	Hans	3	Verkauf	3	Buzibach	Sachbearbeiter

[1. NF]

6.1.3 2. Normalform

Eine Relation(Tabelle) befindet sich in der 2. Normalform (2-NF), wenn die 1. Normalform erfüllt ist und keine teilweisen funktionalen Abhängigkeiten innerhalb einer Relation existieren (jedes Nicht-Schlüsselfeld vom ganzen Primärschlüssel abhängig ist).

teilweise funktionale Abhängigkeiten:

- PersNr→ Name, Vorname, AbtNr, AbtBezeichnung (==>eigene Relation)
- ProjNr → Projektname (==>eigene Relation)

Personen				
PersNr	Name	Vorname	AbtNr	Abtname
0004	Richter	Hans	3	Verkauf

Projekte	
ProjNr	Projektname
1	Amazonas
2	Colorado
3	Buzibach

Projekttätigkeiten		
PersNr	**ProjNr**	Tätigkeit
0004	1	Sachbearbeiter
0004	2	Projektleiter,
0004	3	Sachbearbeiter

[2. NF]

6.1.4 3. Normalform

Eine Tabelle befindet sich in der 3. Normalform (3-NF), wenn die 2. Normalform erfüllt ist und keine Abhängigkeit unter den Nicht-Schlüsselfeldern existieren.
→keine transitiven Abhängigkeiten innerhalb einer Relation.

transitive Abhängigkeiten:

- AbtNr→ Abt.-Name (==>eigene Relation)

Abteilungen	
AbtNr	Abt.-Name
3	Verkauf

Personen				
PersNr	Name	Vorname	fk_AbtNr	~~Abtname~~
0004	Richter	Hans	3	~~Verkauf~~

Projekte	
ProjNr	Projektname
1	Amazonas
2	Colorado
3	Buzibach

Projekttätigkeiten		
PersNr	**ProjNr**	Tätigkeit
0004	1	Sachbearbeiter
0004	2	Projektleiter,
0004	3	Sachbearbeiter

[3. NF]

6.1.5 Übernormalisierung

Eine Übernormalisierung kann sich negativ auf die DB auswirken (Performance und Informationsverlust). Man könnte das ganze soweit treiben, dass wir nur noch Relationen besitzen mit einem Key und einer Information

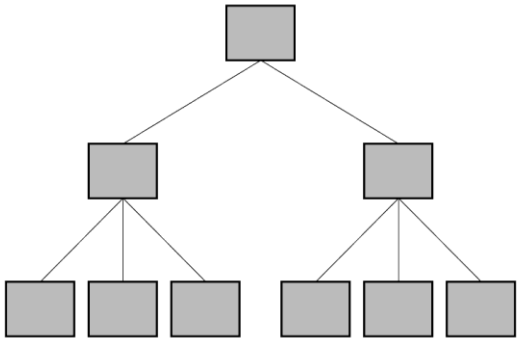
- Tabelle1 [PersNr, Name]
- Tabelle2 [PersNr, Vorname]
- Tabelle3 [PersNr, Strasse]

Sämtliche NF wären hierbei erfüllt. Sinn macht es aber keinen (Performance & Komplexität).
Es kommt also bei dem Normalisierungsvorgang auch auf den gesunden Menschenverstand an.

6.2 DB-Systeme

6.2.1 Hierarchisches Datenbankmodell

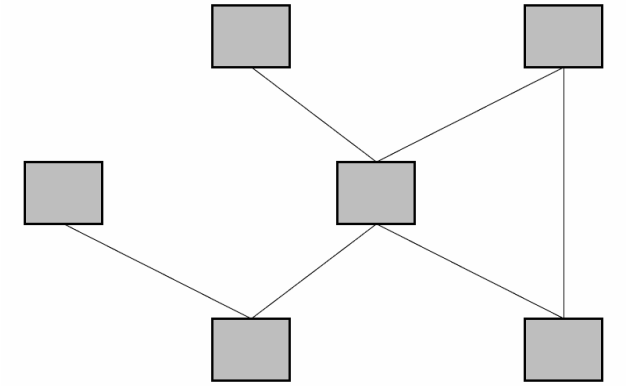
Ein Hierarchisches Datenbankmodell ist das älteste Datenbankmodell, es bildet die Realwelt durch eine hierarchische Baumstruktur ab. Jeder Satz (Record) hat also genau einen Vorgänger, mit Ausnahme genau eines Satzes, nämlich der Wurzel der so entstehenden Baumstruktur.
Die Daten werden in einer Reihe von Datensätzen gespeichert, mit denen verschiedene Felder verknüpft sind. Die Instanzen eines bestimmten Datensatzes werden als Datensatzabbild zusammengefasst. Diese Datensatzabbilder sind vergleichbar mit den Tabellen einer relationalen Datenbank.
Verknüpfungen zwischen den Datensatzabbildern werden in hierarchischen Datenbanken als Eltern-Kind-Beziehungen (Parent-Child Relationships, PCR) realisiert, die in einer Baumstruktur abgebildet werden. Der Nachteil von hierarchischen Datenbanken ist, dass sie nur mit einem solchen Baum umgehen können. Verknüpfungen zwischen verschiedenen Bäumen oder über mehrere Ebenen innerhalb eines Baumes sind nicht möglich.



6.2.2 Netzwerkdatenbankmodell

Das Netzwerkdatenbankmodell wurde von der Data Base Task Group (DBTG) des Programming Language Committee (später COBOL Committee) der Conference on Data Systems Language (CODASYL) vorgeschlagen, der Organisation die auch für die Definition der Programmiersprache COBOL verantwortlich war. Es ist auch unter den Namen "CODASYL Datenbankmodell" oder "DBTG Datenbankmodell" bekannt und entsprechend stark von Cobol beeinflusst. Der fertige DBTG-Bericht wurde 1971, etwa zur gleichen Zeit wie die ersten Veröffentlichungen über das relationale Datenbankmodell, vorgestellt. Er enthielt Vorschläge für drei verschiedene Datenbanksprachen: Eine Schema Data Description Language oder Schema-Datenbeschreibungssprache, eine Subschema Data Description Language oder Subschema-Datenbeschreibungssprache und eine Data Manipulation Language oder Datenmanipulationssprache.

Das Netzwerk-Modell fordert keine strenge Hierarchie sondern kann auch m:n-Beziehungen abbilden, d. h. es kann ein Datensatz mehrere Vorgänger haben. Auch können mehrere Datensätze an oberster Stelle stehen. Es existieren meist unterschiedliche Suchwege, um zu einem bestimmten Datensatz zu kommen. Man kann es als eine Verallgemeinerung des hierarchischen Datenbankmodells sehen.



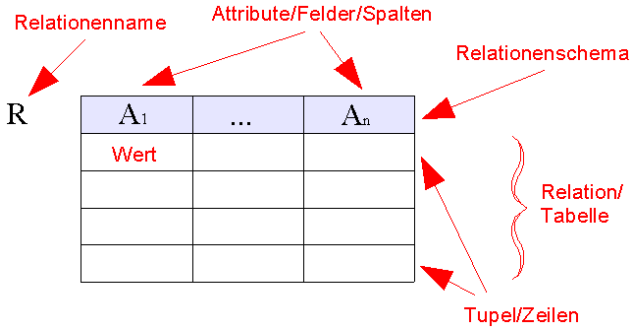
6.2.3 Relationales Datenbankmodell

Eine relationale Datenbank ist eine Datenbank, die auf dem relationalen Datenbankmodell basiert. Das Datenbankmodell wurde von Edgar F. Codd 1970 erstmals vorgeschlagen und ist heute, trotz einiger Kritikpunkte, ein etablierter Standard zum Speichern von Daten. Das zugehörige Datenbankmanagementsystem wird als das relationale Datenbankmanagementsystem (RDBMS) bezeichnet. Bekannt im Zusammenhang mit relationalen Datenbanken ist die Datenbanksprache SQL, zum Abfragen und Manipulieren der Daten in der Datenbank.

Grundlage des Konzeptes relationaler Datenbanken ist die Relation, ein im mathematischen Sinn wohldefinierter Begriff. Dabei handelt es sich im Wesentlichen um eine mathematische Beschreibung für eine Tabelle (siehe dazu Datenbank Relation). Operationen auf diesen Relationen werden durch die Relationale Algebra bestimmt, diese

Operationen finden im Sprachumfang von SQL Berücksichtigung.

Trotz der mathematischen, abstrakten Definition des Datenbankmodells, sind Relationale Datenbanken vergleichsweise einfach und flexibel zu handhaben. Dies hatte großen Einfluss auf den Erfolg dieser Datenbanktechnik.



6.2.4 Objektorientiertes Datenbankmodell

Eine objektorientierte Datenbank ist eine Datenbank, deren Inhalt Objekte im Sinn der Objektorientierung sind. Als ein Objekt wird die Zusammenfassung von zugehörigen Attributen bezeichnet, also gehört zum Beispiel die Farbe und das Gewicht eines Autos zu dem Objekt Auto. Attribute beschreiben ein Objekt näher. Daten und Methoden werden nicht getrennt gespeichert.
Der Vorteil einer objektorientierten Datenbank liegt in der Möglichkeit, Objekte ineinander zu schachteln, um Strukturen abbilden zu können, wie zum Beispiel Firma(Abteilung(Mitarbeiter)). Im englischen und auch im deutschen Sprachgebrauch ist anstelle der Bezeichnung objektorientierte Datenbank auch die Bezeichnung Objektdatenbank (engl. object database) gebräuchlich. Diese Bezeichnung ist kürzer und genauer, denn die Datenbank selbst ist nicht objektorientiert, sondern speichert nur Objekte.

6.3 Transaktionen, konkurrenzierende Zugriffe, Sperrmechanismen

6.3.1 Binäre Sperre

Eine binäre Sperre ist eine Sperre, die genau 2 Zustände haben kann: gesperrt oder offen. Werden binäre Sperren verwendet, so können andere Transaktionen das gesperrte Objekt gar nicht verwenden.

6.3.2 COMMIT

Befehl für die Transaktionssteuerung
Alle Änderungen der Transaktion werden bestätigt und in der Datenbank festgeschrieben.

6.3.3 Deadlock

Situation, die entsteht, wenn zwei Transaktionen die selben Ressourcen benötigen, diese aber in verschiedener Reihenfolge belegen und somit jede der beiden Transaktionen darauf wartet, dass die jeweils andere Transaktion die belegte Ressource wieder freigibt.

6.3.4 Dirty Read

Dieses Problem tritt dann auf, wenn auch unbestätigte Transaktionen berücksichtigt werden und zwischenzeitlich ein Rollback (z.B. durch einen Rechnerabsturz) durchgeführt wurde.

6.3.5 Exklusive Sperre

Nur die Transaktion, welche die exklusive Sperre errichtet hat, hat auch Lese- & Schreibrechte darauf. Alle anderen Transaktionen müssen warten.

6.3.6 konkurrenzierend Zugriffe

Gleichzeitiger, paralleler Zugriff auf Daten

6.3.7 Lost-Update

Wenn ein Update überschrieben wird und so verloren geht. ...

6.3.8 Nichtexklusive Sperre

Die Transaktion, welche die nicht-exklusive Sperre errichtet hat, hat Lese- & Schreibrechte. Die anderen Transaktionen haben nur Leserechte, da ja so kein "Schadenspotential" vorhanden ist.

6.3.9 Nonrepeatable Read

innerhalb einer Transaktion muss der Wert immer gleich sein (obwohl er inzwischen durch andere Benutzer verändert wurde), obwohl er mehrfach ausgelesen wird

6.3.10 Phantome

Das Problem der Phantome tritt auf, wenn nacheinander Werte eingelesen werden, die verrechnet werden, und wenn zwischenzeitlich einer dieser Werte von einem anderen Benutzer geändert wurde.

6.3.11 ROLLBACK

Zurücksetzen einer Transaktion. Der Ursprungszustand wird somit wiederhergestellt.

6.3.12 Sperre (Lock)

Unter Locking (engl. für Sperren) versteht man in der Informatik das Sperren des Zugriffs auf eine Ressource. Eine solche Sperre ermöglicht den exklusiven Zugriff eines Prozesses auf eine Ressource d.h. mit der Garantie dass kein anderer Prozess diese Ressource liest oder verändert solange die Sperre besteht. Locking wird häufig bei Prozesssynchronisation sowie in Datei- und Datenbanksystemen verwendet um atomare und konsistente Lese- und Schreibanforderungen zu gewährleisten.

6.3.13 Sperren auf Datenbankebene

→ vgl. Granularität von Sperren
Wenn eine Transaktion Daten verändert, wird die ganze Datenbank gesperrt. Eine zweite Transaktion kann somit nicht auf die Datenbank zugreifen, obwohl sie eigentlich ganz andere Daten ändern will.

6.3.14 Sperren auf Datensatzebene

→ vgl. Granularität von Sperren
Wenn eine Transaktion Daten verändert, wird der ganze Datensatz gesperrt. Eine zweite Transaktion kann somit nicht auf den gleichen Datensatz zugreifen und diesen ändern.

6.3.15 Sperren auf Feldeben

→ vgl. Granularität von Sperren
Wenn eine Transaktion Daten verändert, werden nur die bearbeiteten Felder gesperrt. Eine zweite Transaktion kann somit auf den gleichen Datensatz, nicht aber auf die gleichen Felder zugreifen und diesen ändern.

6.3.16 Sperren auf Seitenebene

→ vgl. Granularität von Sperren
Wenn eine Transaktion Daten verändert, wird der ganze Seite (Page) gesperrt. ...

6.3.17 Sperren auf Tabellenebene

→ vgl. Granularität von Sperren
Wenn eine Transaktion Daten verändert, wird die ganze Tabelle gesperrt. Eine zweite Transaktion kann somit nicht auf die gleiche Tabelle zugreifen und diesen ändern.

6.3.18 Transaktion

Eine Menge von Befehlen, die nur als ganzes oder gar nicht ausgeführt wird. Schlägt nur ein Teil der Transaktion fehl, wird die Transaktion als fehlgeschlagen angesehen und der Ursprungszustand wiederhergestellt.

6.3.19 Transaktionsprotokoll

Interne Tabelle, die vom DBMS geführt wird und alle Transaktionen festhält. Kommt es zu einem Fehler oder wird ein ROLLBACK-Befehl an die DB abgesetzt, so werden die im Transaktionsprotokoll vorhandenen Informationen genutzt, um den Ursprungszustand wiederherzustellen.

6.4 Verteilte Datenbanken

6.4.1 Datenfragment

Unter einem Datenfragment versteht man einen Teil einer Tabelle oder einer Datenbank, der durch Fragmentierung der gesamten Tabelle entstanden ist und an einem bestimmten Standort gespeichert wird.

6.4.2 Datenfragmentierung

Wird eine einzelne Tabellen über mehrere Standorte verteilt, so spricht man von Datenfragmentierung.

6.4.3 Datenmanager

Der Datenmanager ist eine Software-Komponente eines verteilten Datenbanksystems, der auf den einzelnen Datenbankservern läuft und den Zugriff der Transaktionsmanager auf die Datenbank steuert, die auf den Clientrechnern des Datenbanksystems ausgeführt werden.

6.4.4 Datenreplikation

Bei der Datenreplikation werden Kopien der Daten an verschiedenen Standorten gehalten, um den Zugriff auf diese Daten zu beschleunigen.

6.4.5 Distributed Database Management System (DDBMS)

Das DDBMS ist das Datenbankmanagement-System der verteilten Datenbank. Es kümmert sich darum, dass die verteilte Umgebung alle Funktionen unterstützt, die bei einem zentralen Datenbanksystem implementiert sind.

6.4.6 DO

Der Befehl DO schreibt eine Datenänderung in das Transaktionsprotokoll. Hierbei wird sowohl der neue als auch der alte Werte des Datenfeldes gespeichert, damit die Änderung

gegebenenfalls wieder rückgängig gemacht werden kann.

6.4.7 Gemischte Fragmentierung

Bei der gemischten Fragmentierung werden Tabellen sowohl horizontal als auch vertikal fragmentiert.

6.4.8 heterogenes verteiltes Datenbanksystem

Bei einem heterogenen verteilten Datenbanksystem sind die DBMS verschiedener Hersteller an der verteilten Datenbank beteiligt.

6.4.9 homogenes verteiltes Datenbanksystem

Bei einem homogenen verteilten Datenbanksystem sind die auf den verschiedenen Servern installierten DBMS vom selben Hersteller.

6.4.10 Horizontale Fragmentierung

Bei der horizontalen Fragmentierung wird eine Tabelle datensatzweise fragmentiert, das heißt, jedes Tabellenfragment enthält dieselben Felder, es ist nur die Anzahl der gespeicherten Datensätze unterschiedlich.

6.4.11 Ortstransparenz

Unter Ortstransparenz versteht man, dass der Anwender, der auf ein Tabellenfragment zugreifen will, nicht wissen muss, an welchem Ort sich dieses Tabellenfragment befindet.

6.4.12 REDO

REDO führt eine mittels UNDO zurückgenommene Änderung erneut aus.

6.4.13 single point of failure

Der Single point of failure stellt die Komponente eines Systems dar, die bei ihrem Ausfall bewirkt, dass das gesamte System nicht mehr funktionieft

6.4.14 Transaktionsmanager

Der Transaktionsmanager ist ein Programm, das auf jedem Computer ausgeführt wird, der auf das verteilte Datenbanksystem zugreifen muss. Er kümmert sich darum, dass die Datenbankabfragen des Clients an die richtigen Datenbankserver verteilt werden und fügt die von verschiedenen Servern erhaltenen Daten auf dem Client lokal zusammen.

6.4.15 Transparenz

Unter Transparenz versteht man, dass die physikalische Verteilung des Datenbanksystems für den Anwender nicht sichtbar ist.

6.4.16 UNDO

Mit Hilfe des Befehls UNDO kann man die durch eine Transaktion an der Datenbank vorgenommenen Änderungen wieder zurücknehmen.

6.4.17 Verteilte Datenbank

Bei einer verteilten Datenbank ist die Datenhaltung über mehrere Standorte verteilt.

6.4.18 Verteilte Datenverarbeitung

Bei einer verteilten Datenverarbeitung ist die Verarbeitung der Daten über mehrere Standorte verteilt, so wie das z.B. bei der Client-Server-Architektur üblich ist. Die verteilte Datenverarbeitung impliziert nicht, dass die Datenhaltung auch verteilt ist. Verteilte Datenverarbeitung kann auch mit einer zentralen Datenbank stattfinden.

6.4.19 Verteilte Transaktion

Eine verteilte Transaktion ändert Daten an mehreren Standorten einer verteilten Datenbank. Es muss bei verteilten Transaktionen sichergestellt sein, dass die verteilte Transaktion nur dann erfolgreich ausgeführt werden kann, wenn alle Teilaktionen an allen Standorten erfolgreich beendet werden konnten.

6.4.20 »Verteiltes Datenbank Dictionary«

Das »Verteilte Datenbank Dictionary« verwaltet, welche Daten sich an welchem Standort befinden. Es ist selbst eine verteilte Datenbank, die auf jeden Standort vollständig repliziert wird.

6.4.21 Vertikale Fragmentierung

Bei der vertikalen Fragmentierung werden Tabellen entsprechend ihrer Felder fragmentiert, das heißt, die Struktur der Tabellen der einzelnen Tabellenfragmente ist unterschiedlich. Wichtig ist allerdings, das s die Anzahl der Datensätze in beiden Tabellen gleich sein muss.

6.4.22 Vollständig verteiltes Datenbanksystern

Ein vollständig verteiltes Datenbanksystem besitzt eine verteilte Datenverarbeitung und eine verteilte Datenhaltung.

6.4.23 Vollständige Transparenz

Besitzt eine verteilte Datenbank vollständige Transparenz, so stellt sich dieses System dem Benutzer gegenüber wie ein zentrales Datenbanksystem dar.

6.4.24 Write-ahead-Modus

Beim Write-ahead-Modus wird das Transaktionsprotokoll auf die Festplatte geschrieben, bevor Änderungen an der Datenbank selbst durchgeführt werden.

6.4.25 Zwei-Phasen-Commit-Protokoll

Das Zwei-Phasen-Commit-Protokoll wird dazu verwendet, verteilte Transaktionen zu steuern und um zu garantieren, dass die Daten entweder an allen Standorten geändert werden konnten oder dass alle Änderungen zurückgenommen werden.

6.5 Fragen

6.5.1 Was sind die Vor- und Nachteile verteilter Datenbanksysteme?

Die Vorteile verteilter Datenbanksysteme liegen darin, dass die Gesamtmenge der Daten auf verschiedene Standorte verteilt werden kann und hierdurch Datenzugriffszeiten erheblich reduziert werden können. Des Weiteren bietet ein verteiltes Datenbanksystem im Gegensatz zum zentralen System keinen single-point-of-failure aus. Die Nachteile des verteilten Datenbanksysterns liegen hauptsächlich in der erheblich komplexeren Struktur.

6.5.2 Was ist der Unterschied zwischen verteilter Datenverarbeitung und verteilten Datenbanken

Bei der verteilten Datenverarbeitung wird lediglich die Verarbeitung der Daten auf andere Systeme ausgelagert, wie Sie das ja bereits von der Client-ServerArchitektur her kennen. Bei verteilten Datenbanken geht es wirklich darum, dass sich die Datenhaltung über verschiedene physikalische Standorte verteilt.

6.5.3 Was sind die Komponenten eines verteilten Datenbanksystems?

Ein verteiltes Datenbanksystem wird von einem DDBMS kontrolliert. Auf den Clients befindet sich ein Transaktionsmanager, der sich darum kümmert dass die Daten auf den einzelnen Standorten richtig angesprochen und die Datenmenge, die von verschiedenen Standorten zurückgeliefert wurde, wieder richtig zusammengesetzt wird. Auf den Datenbankservern befinden sich Datenmanager, die auf die verteilten Datenbankabfragen der einzelnen Clients antworten.

6.5.4 Was bedeutet Transparenz beim Datenzugriff?

Transparenz beim Datenzugriff bedeutet, dass es für den Benutzer nicht erkennbar sein soll, dass er statt mit einem zentralen Datenbanksystern mit einer verteilten Datenbank arbeitet. Das DDBMS muss alle Funktionen anbieten, die ein zentrales Datenbanksystem auch anbietet, darf aber die zusätzliche Komplexität den Benutzer nicht spüren lassen.

6.5.5 Welche Arten von Transparenz beim Datenzugriff gibt es?

Beim Datenzugriff gibt es die totale Transparenz, bei der verteilte Tabellen genau so angesprochen werden wie zentrale Tabellen auf einem zentralen Datenbankserver. Des Weiteren gibt es die Ortstransparenz, das heißt, der Anwender muss wissen, dass die Tabelle, die er abfragen möchte, aus mehreren Teilen besteht, er muss aber nicht wissen, wo sich diese Teile befinden. Zuletzt gibt es noch Systeme ohne Ortstransparenz, das heißt, hier müssen die Anwender sowohl wissen, dass die Datenbank in mehrere Teile aufgeteilt ist als auch, wo sich diese Teile befinden.

6.5.6 Warum ist ein transparentes Transaktionsmanagement notwendig?

Transparentes Transaktionsmanagement ist notwendig, da ein SQL-Befehl in einer verteilten Umgebung Daten an mehr als einem Standort ändern kann. Kommt es nun an einem Standort zu Problemen, so müssen die Änderungen an allen anderen Standorten auch wieder rückgängig gemacht werden. Daher ist hier ein verteiltes, transparentes Transaktionsmanagement notwendig.

6.5.7 Wie funktionieren verteilte Transaktionen?

Bei einer verteilten Transaktion wird das Zwei-Phasen-Commit-Protokoll eingesetzt. Ein Transaktionskoordinator sendet an alle an der Transaktion beteiligten Stationen den Befehl READY TO COMMIT. Liefert eine Station eine negative Antwort zurück, so wird die verteilte Transaktion abgebrochen. Haben alle Stationen bestätigt, so wird nun ein COMMIT an diese gesendet. Teilt eine Station dem Transaktionskoordinator mit, dass sie das COMMIT nicht durchführen konnte, so wird die gesamte Transaktion abgebrochen und der Ursprungszustand auf allen beteiligten Systemen wieder hergestellt.

6.5.8 Was versteht man unter Datenfragmentierung?

Unter Datenfragmentierung versteht man die Aufteilung logisch zusammenhängender Daten, also beispielsweise einer Tabelle, in mehrere physikalische Einheiten.

6.5.9 Welche Arten der Datenfragmentierung gibt es?

Es gibt die horizontale Fragmentierung, bei der die Tabellen datensatzweise fragmentiert werden, die vertikale Fragmentierung, bei der die Tabellen feldweise fragmentiert werden, und die gemischte Fragmentierung, bei der die Datensätze sowohl feldweise als auch tabellenweise fragmentiert werden.

6.5.10 Was versteht man unter Replikation?

Bei der Replikation werden Datenbestände physikalisch an andere Standorte kopiert. Hiermit kann ein schnellerer Datenzugriff gewährleistet werden. Diesen Geschwindigkeitsvorteil erkauft man sich aber durch einen erhöhten Verwaltungsaufwand.

6.6 JOIN: Erklärung

6.6.1 INNER JOIN

Die Datensätze aus beiden Tabellen werden verbunden, wenn ein oder mehrere gemeinsame Felder den gleichen Wert haben.

6.6.2 LEFT OUTER JOIN

(Linke Inklusionsverknüpfung)
Von der ersten (linken) Tabelle werden alle Datensätze in die Ergebnismenge übernommen. Von der zweiten (rechten) Tabelle werden nur die dazugehörigen Datensätze übernommen. Die Felder der zweiten Tabelle bleiben leer, wenn kein passender Datensatz vorhanden ist.

6.6.3 RIGHT OUTER JOIN

(Rechte Inklusionsverknüpfung)
Von der zweiten (rechten) Tabelle werden alle Datensätze in die Ergebnismenge aufgenommen. Von der ersten (linken) Tabelle werden nur die dazugehörigen Datensätze übernommen. Die Felder der zweiten Tabelle bleiben leer, wenn kein passender Datensatz vorhanden ist.

6.6.4 FULL OUTER JOIN

= FULL JOIN
Der Full-Join ist eine Kombination aus dem Left-Outer-Join und dem Right-Outer-Join. Alle Datensätze beider Tabellen werden in die Ergebnismenge übernommen. Passen Datensätze aus beiden Tabellen laut Vergleichsoperation zusammen, so werden sie verbunden.