1. Einleitung

Datenbanksysteme (DBS)

DBS ermöglicht die anwendungsübergreifende Nutzung von Daten.

DBS isoliert Anwendungen von Hardware und Betriebssystem

DBS= DB + DBMS (Datenbank + Datenbankmanagementsystem)

Datenbank (DB)

Persistente Speicherung von Daten

Zentraler Ansatz (Integrierter Datenbestand)

Datendefinition (Datenstruktur, Integritätsregeln) und Zugriff auf die Daten (Operationen) erfolgen nur durch das DBMS

Datenbankmanagementsystem (DBMS)

DBMS ermöglicht einfache Datendefinition und Zugriff auf die Daten

DBMS garantiert Korrektheit der Daten (Einhaltung der Integritätsregeln)

DBMS garantiert Korrektheit der Daten und der Sicht auf die Daten im Mehrbenutzerbetrieb (Concurrency Control)

DBMS garantiert Datensicherheit, d.h., Korrektheit bei fehlerhaften Abläufen und bei Systemabsturz

DBMS bietet Mechanismen zur Zugriffskontrolle (Datenschutz)

Transaktionen

ACID Prinzip

- **Atomicity (Atomizität)**
- **Consistency (Konsistenzerhaltung, Konsistente Sicht auf Daten)**
- **Isolation**
- **Durability (Dauerhaftigkeit)**

Umsetzung durch

- **Logging**
- **Locking**

ACID

A - Atomicity:

Eine Transaktion wird entweder ganz oder gar nicht ausgeführt

- **Logging (Protokollierung von before und after images von Änderungen) ermöglicht Rollback**
- **Locking (Synchronisation paralleler Zugriffe durch Sperren) verhindert parallele Änderungen**

C - Consistency:

Sind die von einer Transaktion betroffenen Daten vor der Transaktion konsistent, so sind sie es auch nach der Transaktion (Konsistenzerhaltung).

Innerhalb einer Transaktion hat man eine konsistente Sicht auf die Daten.

(Diese Anforderung wird durch die Einführung von Isolation Levels aufgeweicht!)

- **Logging sorgt für die Konsistenz der Daten auch im Fehlerfall**
- **Locking sorgt für die Konsistenz der Daten auch bei konkurrierendem Zugriff**
- **Locking (oder Logging) sorgt für die konsistente Sicht auf die Daten**

I - Isolation:

Innerhalb einer Transaktion greift man auf die Daten zu, als ob es keine konkurrierenden Zugriffe gäbe.

- **Locking sorgt für die Abschirmung konkurrierender Transaktionen voneinander**

D-Durabiliy:

Änderungen, die in einer Transaktion erfolgt sind, sind in der Datenbank festgeschrieben und stehen allen zur Verfügung.

- **Logging ermöglicht es, dass nach einem Systemausfall sämtliche bestätigten Transaktionen in der Datenbank festgeschrieben werden und sämtliche offenen Transaktionen zurückgerollt werden können**
- **Locking verhindert(e) parallele Änderungen**

Logging

- **Grundsätzlich arbeiten DBS mit einem Buffer Cache, in dem sich die Datenblöcke befinden, auf die häufig zugegriffen wird (aus Performancegründen).**
- **Änderungen werden grundsätzlich im Buffer Cache durchgeführt. Sie werden auch beim Commit nicht zwingend in die Datenbank zurückgeschrieben (Fast Commit).**
- **Das Schreiben von Log Einträgen, das beim Commit zwingend erforderlich ist (siehe unten), wird gebündelt (Group Commit).**
- **Um einen Rollback zu ermöglichen, müssen Before Images zu allen Änderungen zur Verfügung stehen.**

Um im Falle eines Systemausfalls (bei dem der Buffer Cache verloren geht), eine Crash Recovery durchführen zu können, ist folgendes zu berücksichtigen:

- **Die Before Images müssen persistent gespeichert werden, bevor der entsprechende Block in die Datenbank geschrieben wird. (Write Ahead Logging)**
- **Die After Images müssen spätestens beim Commit persistent gespeichert werden.**
- **Es muß persistent festgehalten werden, bis zu welchem Zeitpunkt Änderungen in die Datenbank zurückgeschrieben worden sind (Checkpoint)**
- **Im Rahmen der Crash Recovery werden zunächst die Änderungen aller (bestätigter) Transaktionen nach dem Checkpoint nachgefahren (Redo Recovery), anschließend erfolgt ein Rollback aller offenen Transaktionen.**

(Widerstrebende) Anforderungen zur Konfiguration von Checkpoints:

- **Checkpoints sollten den laufenden Betrieb nicht stören!**
- **Recovery sind abhängig von der Position des letzten Checkpoints!**

Media Recovery

bei Ausfall der DB (nicht nur der Instanz):

Voraussetzungen:

- **Backup der DB (bzw. von Teilen der DB, die wiederherzustellen sind)**
  - **Full backup**
  - **Incremental backup (Level 0,1,2,...)**

- **Archivierung der Protokolle seit dem (Beginn des) Backup(s)**

Offline Backup ist konsistent und kann alleine zur Media Recovery verwendet werden.

Online Backup ist nicht notwendig konsistent und kann daher nur zur Media Recovery herangezogen werden, wenn die Änderungen während des Backup protokolliert werden!

Durchführung der Media Recovery (durch DBA):

- **Zurückspielen des Backup (Restore)**
- **Anwendung der Protokolle (Recovery)**
  - **bis zur letzten abgeschlossenen Transaktion**
  - **bis zu einem gewissen Zeitpunkt (Point in Time – Recovery)**

Minimierung der MTTR (Mean Time to Repair):

- **Selektiver bzw. Paralleler Restore**

- **kleines Recovery Fenster (kritisch!)**

Ergänzende Konzepte zur Datensicherheit:

- **Replikation (im weitesten Sinn!)**

Locking

Schreibende Zugriffe sind i.a. mit exklusiven Sperren (X locks) auf die betroffenen Zeilen verbunden, die bis zum Transaktionsende gehalten werden. Diese Sperren sind grundsätzlich nicht konfigurierbar!

Lese Sperren sind konfigurierbar, indem man das Isolation Level auf Transaktionsebene setzt:

read uncommitted	**Bereitschaft, nicht bestätigte Daten zu lesen**
read committed	**Nur bestätigte Daten werden gelesen**
repeatable read	**Wiederholtes Lesen einer Zeile innerhalb einer TX führt zum gleichen Ergebnis**
serializable	**Wiederholtes Ausführen einer Abfrage innerhalb einer TX führt zum gleichen Ergebnis (keine Phantom Inserts!)**

- **Es gibt DBS, die nicht alle Isolation Level implementieren (z.B. Oracle)**
- **Die Implementierung der Isolation Level ist von DBS zu DBS unterschiedlich!**

Weitere Möglichkeiten, Locking zu konfigurieren:

- **Update / Exclusive Locks für "Read for Update" - Situationen**
- **Table Locks**
- **Timeouts**

Locking Strategien für Read for Update Situationen

Optimistisches Locking

- **Keine bzw. kurzfristige Sperren für lesende Zugriffe**
- **Kann prinzipiell zu Lost Updates führen**
- **Lösung in Applikation:**
    - **Falls Daten in der Datenbank mit den gelesenen Daten nicht mehr übereinstimmen, Konflikt feststellen (durch wiederholtes Lesen oder intelligenten Update) und behandeln (Wiedervorlage)**

- **Problem: Zu viele Konflikte werden i.a. nicht akzeptiert!**

Pessimistisches Locking

- **Lesende Zugriffe mit Update / Exclusive Locks**
- **Führt zu Serialisierung der Read for Update Zugriffe**

- **Problem: Wartezeiten!**

Datenunabhängigkeit

Immunity of applications to change in storage and access strategy

Datenunabhängigkeit wird erreicht durch die Einführung von Indirektion und Abstraktion

3 Ebenen Architektur

Anwendungsschicht

Logische Schicht

Physische Schicht

Physische Datenunabhängigkeit

Abschirmung der Anwendungsschicht von der physischen Sicht durch die Existenz der logischen Schicht

Logische Datenunabhängigkeit

Abschirmung der Anwendungsschicht von der logischen Schicht durch Transformationsregeln zwischen diesen Schichten

Vorteile von DBS

Integrierter Datenbestand => Redundanzfreiheit

Zentrale Verwaltung der Daten durch DBMS => Integrität => höhere Datenqualität

Einheitliche Mechanismen für Datenschutz, Datensicherheit und konkurrierenden Zugriff

Datenunabhängigkeit

Einfachere Entwicklung

Nachteile eines DBS

Effizienz

Höhere Kosten (Beschaffung, Administration)

Datenmodell (Metamodell)

Datenstruktur

Integritätsregeln

Operationen auf den Daten

Konzeptuelle Datenmodelle

Entity-Relationship Modell

UML

Logische Datenmodelle

Hierarchisch

Netzwerk

Objektorientiert

Relational

Objektrelational

2. Entity Relationship Modell (ERM)

P. P. Chen (1976)

Konstrukte des ERM

Objekttyp (Entity):

Darstellung als Rechteck

Attributtypen:

Darstellung als Oval, mit dem Objekttyp durch Linien verbunden

Beziehungstypen (Relationship):

Darstellung als Raute, mit den in Beziehung gesetzten Objekttypen durch Linien verbunden

Attribute zulässig

Grad eines Beziehungstyps: Anzahl der beteiligten Objekttypen

Binäre Beziehungstypen: Beziehungstypen vom Grad 2

Beziehungen höheren Grades lassen sich nicht immer verlustfrei in binäre Beziehungstypen aufbrechen!

Beziehungstypen können auch reflexiv sein, d.h., Objekte des gleichen Objekttyps in Beziehung setzen!

Kardinalität eines Beziehungstyps:

Bei Betrachtung eines Tupels von Objekten der in Beziehung stehenden Objekttypen, wie viele Objekte des betrachteten Objekttyps stehen zu diesem Tupel in Beziehung?

Komplexität eines Beziehungstyps (z.T. auch als Kardinalität bezeichnet):

Wie oft kommen Objekte eines Objekttyps in Beziehungen eines Beziehungstyps vor?

1	**bzw.**	**(1,1)**	**genau 1**
c	**bzw.**	**(0,1)**	**höchstens 1**
n	**bzw.**	**(1,*)**	**mindestens 1**
nc	**bzw.**	**(0,*)**	**mindestens 0**
		(m,n),	**wobei für n der Stern erlaubt ist**

Die Klassifikation der Rolle von Objekttypen im Rahmen eines Beziehungstyps als optional (höchstens 1, mindestens 0) / mandatorisch (genau 1, mindestens 1) ist für die weitere Modellierung sehr wichtig. Im Zweifelsfall sollte die Klassifikation als optional erfolgen, da das Datenmodell dadurch mehr Flexibilität aufweist.

Erweiterungen des ERM:

Existenzabhängige Objekttypen

Objekte zu diesen Objekttypen hängen von der Existenz eines Objekts von einem übergeordneten Objekttyp ab (Beispiel: Auftragsposition kann als existenzabhängig von Auftrag betrachtet werden).

Darstellung: Beziehungstyp und existenzabhängiger Objekttyp werden mit gedoppeltem Rand versehen.

Mengenwertige Attributtypen werden häufig als existenzabhängige Objekttypen modelliert.

Im weiteren Verlauf der Modellierung ist die Existenzabhängigkeit nur noch bedingt wichtig!

Aggregation / Komposition

Beziehungstypen der Form “besteht aus”

Die Komposition zeichnet sich dadurch aus, daß die Teile zu genau einem Ganzen gehören und von diesem existenzabhängig sind, was bei der Aggregation nicht zwingend ist.

Häufig wird Komposition per Existenzabhängigkeit modelliert, Aggregation als “normaler” Beziehungstyp dargestellt.

Vererbung (Generalisierung / Spezialisierung) / Partitionierung

Vererbung wird im ERM als spezieller Beziehungstyp, IS_A, i.a. in Form eines Hexagons dargestellt, wobei ein Pfeil auf den generalisierten Objekttyp zeigt und zu jedem spezialisierten Objekttyp eine Linie besteht.

Partitionierung zeichnet sich dadurch aus, daß die spezialisierten Objekttypen disjunkt sind und die Vereinigung dem generalisierten Objekttyp entspricht. Für die Partitionierung gibt es keine allgemeine grafische Notation, es ist eine verbale Beschreibung üblich!

Modellierung im Hinblick auf das relational Datenmodell

Der Schlüsselbegiff (Key):

- **Eine Attributkombination, durch die ein Objekt eines Objekttyps identifiziert wird und die in dieser Eigenschaft minimal ist, wird als Schlüsselkandidat bezeichnet (candidate key).**
- **Im Rahmen der Modellierung wird ein Schlüsselkandiadat als Primärschlüssel ausgezeichnet (primary key).**
- **Die übrigen Schlüsselkandidaten werden als Alternativschlüssel bezeichnet (alternate key).**
- **Eine Attributkombination, über die die Zuordnung zwischen Objekten des untergeordneten und Objekten des übergeordneten Objekttyps (via Schlüsselkandidat) bei einem n:1 – Beziehungstyp vorgenommen wird, wird als Fremdschlüssel bezeichnet (foreign key).**

Notation:

Für die Darstellung von Primärschlüsseln, Alternativschlüsseln und Fremdschlüsseln in einem ERM werden unterschiedliche Notationen verwendet.

Wir werden Primärschlüssel mit PK, Alternativschlüssel mit AK1, AK2,... und Fremdschlüssel mit FK1, FK2,... bezeichnen, wobei für Fremdschlüssel der Bezug zum zugehörigen Schlüsselkandidaten hergestellt wird.

Schlüssel dienen der Darstellung von Integritätsbedingungen im relationalen Datenmodell (Objektintegrität, referentielle Integrität).

Modellierung im Hinblick auf das relationale Datenmodell

Assoziative Objekttypen

Im Hinblick auf die Modellierung für ein relationales Datenbanksystem werden i.a. n:m, nc:m, n:mc, nc:mc, n:c, c:m und c:c – Beziehungstypen und Beziehungstypen, deren Grad größer als 2 ist, aufgelöst und durch sog. assoziative Objekttypen ersetzt.

Hieraus resultiert ein ERM, das nur noch Objekttypen und 1:n – Beziehungstypen enthält (abgesehen von Vererbung)! Jeder dieser Objekttypen wird als Tabelle dargestellt, die Beziehungen werden durch Fremdschlüssel und Schlüsselkandidaten hergestellt.

Für assoziative Objekttypen gilt:

Ein Schlüsselkandidat jedes aus Sicht des Beziehungstyps zugehörigen Objekttyps ist als Fremdschlüssel – Attribut zu übernehmen!

Es gilt häufig, aber nicht in jedem Fall, daß die Kombination der Fremdschlüssel einen Schlüsselkandidaten für den assoziativen Objekttyp darstellt.

Vererbung

Zur Darstellung der Vererbung wird in relationalen Datenbanksystemen i.a. eine der folgenden Varianten gewählt:

- **Eine Tabelle mit allen Attributen des generalisierten Objekttyps.**
- **Eine Tabelle mit den gemeinsamen Attribute aller spezialisierten Objekttypen, pro spezialisiertem Objekttyp eine Tabelle mit den nicht gemeinsamen Attributen (und dem Primärschlüssel des generalisierten Objekttyps).**
- **Pro spezialisiertem Objekttyp eine Tabelle.**

Beispiel (Universitäts – Datenmodell)

- ***Vorlesungen (Vorlesungsnummer, Titel) setzen Vorlesungen voraus***
- ***Professoren (Personalnummer, Name, Rang) lesen Vorlesungen***
- ***Vorlesungen werden jeweils von einem Professor gelesen***
- ***Studenten (Matrikelnummer, Name, Semester) hören Vorlesungen***
- ***Vorlesungen werden von Studenten gehört***
- ***Professoren prüfen Studenten über Vorlesungen (pro Student und Vorlesung nur eine Prüfung, pro Prüfung eine Note!)***
- ***Professoren beschäftigen Assistenten (Personalnummer, Name, Fachgebiet)***
- ***Assistenten arbeiten jeweils für einen Professor***

3. Relationales Datenmodell

3.1 Datendefinition (DDL)

- **Einzige Datenstruktur für Daten (inklusive Metadaten) ist die Relation (Tabelle)**
- **Relation: Teilmenge des kartesischen Produkts der Attribut Domänen (Wertebereiche der Spalten).**
- **Relationale DBMS unterstützen das Domänenkonzept nur teilweise (check constraints)**

3.2 Datenintegrität (DDL)

- **NULL Werte stehen für nicht vorhandene Information.**
- **Objekt Integrität (Entity Integrity): Kein Teil eines Schlüsselkandidaten darf NULL sein.**
- **Referentielle Integrität (Referential Integrity): Ein Fremdschlüsselwert kommt im zugehörigen Schlüsselkandidaten vor, oder sämtliche Komponenten sind NULL. Wird die referentielle Integrität (durch Löschen oder Änderung des Werts für einen Schlüsselkandidaten des übergeordneten Objekts) verletzt, so kann definiert werden, ob das DBMS mit Ablehnung der Aktion (reject), Kaskadierung (cascade) oder Setzen des Fremdschlüsselwerts auf NULL (set null) reagiert. Im Prinzip besteht die gleiche Möglichkeit, auf Verletzung der referentiellen Integrität bei Einfügen eines untergeordneten Objekts oder Ändern des Werts für einen Fremdschlüssel zu reagieren.**
- **Business Rules werden wie Objekt Integrität und Referentielle Integrität vom relationalen DBMS überwacht. (Primary Key und Unique Constraints zur Implementierung von Schlüsselkandidaten, Foreign Key Constraints zur Implementierung von Fremdschlüsseln, Check Constraints und Trigger zur Implementierung von Business Rules)**

3.3 Datenmanipulation (DML)

3.3.1 Relationale Algebra:

- **Restriktion: Auswahl von Zeilen aus einer Tabelle**
- **Projektion: Auswahl von Spalten aus einer Tabelle (Duplikate?)**
- **Vereinigung**
- **Durchschnitt**
- **Differenz**
- **Kartesisches Produkt**
- **Join**
  - **Equi Join: Verknüpfung aufgrund der Gleichheit von Attributwerten**
  - **Natural Join: Verknüpfung aufgrund der Gleichheit von Fremdschlüsselwerten und zugehörigen Schlüsselkandidatwerten (ggf Elimination der Fremdschlüssel-attribute oder der Schlüsselkandidatattribute)**
  - **Semi Join: Projektion des Natural Join auf eine Tabelle**
  - **Inner Join: alternativer Begriff zum Natural Join**
  - **Left Outer Join, Right Outer Join, Full Outer Join: Ergänzung des natural / inner Joins durch Berücksichtigung von Masterzeilen ohne passende Details, Detailzeilen ohne passenden Master sowie beidem**
  - **Theta Join: Verknüpfung aufgrund von Vergleichen von Attributwerten**
  - **Auto / Self Join: Verknüpfung einer Tabelle mit sich selbst**
- **Quotient**

3.3.2 Relationales Tupelkalkül

Eine Abfrage ist im relationalen Tupelkalkül durch eine sog. Formel beschrieben:

Bestimme die Menge aller Tupelvariablen t, die die Formel F(t) erfüllen

Die Grundbausteine von Formeln sind die sog. Atome:

- **t in T, wobei T eine Tabelle ist und t eine Tupelvariable**
- **Vergleich von t1.A1 und t2.A2 wobei t1 und t2 Tupelvariablen, A1 und A2 Attribute sind**
- **Vergleich von t.A und c, wobei wieder t eine Tupelvariable und A ein Attributname sind und c eine Konstante**

Formeln sind wie folgt aufgebaut:

- **Alle Atome sind Formeln**
- **Falls F eine Formel ist, so sind es auch die Negation von F und (F)**
- **Falls F1 und F2 Formeln sind, so sind es auch F1 und F2, F1 oder F2, F1 impliziert F2**
- **Falls F eine Formel ist, so sind auch "Es existiert eine Tupelvariable t, so daß F(t) gilt" und "Für alle Tupelvariablen t gilt F(t)" Formeln**

Sichere Ausdrücke des Tupelkalküls:

Die Komponenten der Ergebnistupel sind enthalten im Wertebereich der zugehörigen Attribute der beteiligten Tabellen.

Bemerkungen:

- **Es läßt sich zeigen, daß das relationale Tupelkalkül, reduziert auf sichere Ausdrücke, die gleiche Mächtigkeit besitzt wie die Relationenalgebra!**
- **Dies gilt auch für eine dritte relationale Abfragesprache, das relationale Domänenkalkül, das hier allerdings nicht behandelt wird.**

4. Normalformen:

Aufeinander aufbauendes Regelwerk, daß einen guten Datenbankentwurf verifizieren soll.

Ziel ist die Verifikation, nicht die Konstruktion eines Datenmodells, auch wenn das in der Literatur zu finden ist!

Die Normalformen lassen sich in drei Gruppen unterteilen,

- **die erste Normalform, die sich mit nicht-atomaren Attributen beschäftigt**
- **die zweite und dritte Normalform und die Boyce-Codd Normalform, die sich mit funktionalen Abhängigkeiten beschäftigen**
- **die vierte und fünfte Normalform, die sich mit mehrwertigen Abhängigkeiten und der Möglichkeit der verlustfreien Zerlegung beschäftigen**

Erste Normalform:

Eine Relation ist in erster Normalform (1NF), wenn jedes Attribut atomar ist.

Bemerkungen:

- **Es ist interpretationsfähig, ob ein Attribut atomar ist!**
- **Die objektrelationalen Datenbanksysteme, die als Weiterentwicklung der relationalen Datenbanksysteme zu sehen sind, lassen gerade strukturierte Datentypen zu, d.h., es wird bewußt gegen die 1NF verstoßen.**
- **Auch bei Verwendung rein relationaler Datenbanksysteme können Verstöße gegen die 1NF aus Performance Aspekten sinnvoll sein.**

Für die folgende Gruppe der Normalformen ist der Begriff der funktionalen Abhängigkeit wichtig:

Funktionale Abhängigkeit:

Ein Attribut A heißt funktional abhängig von einer Attributkombination {A1,...,An} ({A1,...,An}→A), wenn es eine Funktion f gibt mit

- **f(a1,...,an) = a für alle (a1,...,an,a) für alle Einträge in der Relation mit ai in Ai, i=1...,n, und a in A.**

Im Kontext der funktionalen Abhängigkeit werden häufig zwei Definitionen verwendet:

Ein Attribut A heißt voll funktional abhängig von einer Attributkombination (A1,...,An), wenn

- **A funktional abhängig ist von {A1,...,An}**
- **A nicht funktional abhängig ist von einer echten Teilmenge von {A1,...,An}**

Ein Attribut A heißt transitiv funktional abhängig von einer Attributkombination {A1,...,An} via einer Attributkombination {B1,...,Bm), wenn

- **Bi funktional abhängig ist von {A1,...,An} für alle i=1,...,m**
- **A funktional abhängig ist von der Attributkombination {B1,...,Bm}**

Mit Hilfe der oben eingeführten Begriffe lassen sich die zweite und dritte Normalform einfach definieren:

Zweite Normalform:

Eine Relation ist in zweiter Normalform (2NF), wenn sie in 1NF ist und jedes Nichtschlüsselattribut (Attribut, das zu keinem Schlüsselkandidaten gehört) voll funktional abhängig von jedem Schlüsselkandidaten ist. (Trivial, falls alle Schlüsselkandidaten aus einem Attribut bestehen!)

Dritte Normalform:

Eine Relation ist in dritter Normalform (3NF), wenn sie in 2NF ist und kein Nichtschlüsselattribut transitiv funktional abhängig von einem Schlüsselkandidaten via einer Kombination von Nichtschlüsselattributen ist. (D.h., kein Nichtschlüsselattribut darf von einer Kombination anderer Nichtschlüsselattribute funktional abhängig sein!)

Probleme bei nicht normalisierten Relationen:

- **Update – Anomalie**
- **Insert – Anomalie**
- **Delete – Anomalie**

Lösung:

Zerlegung der Relation in Relationen, die normalisiert sind. Diese Zerlegung sollte folgende Eigenschaften haben:

- **Verlustfrei (Der Inner Join der aus der Zerlegung resultierenden Relationen ergibt die ursprüngliche Relation.)**
- **Abhängigkeitserhaltend (Jede Funktionale Abhängigkeit in der ursprünglichen Relation ist eine funktionale Abhängigkeit in einer der resultierenden Relationen.)**

Jede Relation kann verlustfrei und abhängigkeitserhaltend in Relationen in 3NF zerlegt werden!

Da die "kanonische" Zerlegung "entlang" der funktionalen Abhängigkeiten erfolgt, ist diese Zerlegung verlustfrei und abhängigkeitserhaltend.

Die 3NF löst noch nicht alle Probleme im Zusammenhang mit funktionalen Abhängigkeiten. Es werden nämlich nur funktionale Abhängigkeiten von Nichtschlüsselattribute (von Schlüsselkandidaten) betrachtet, nicht funktionale Abhängigkeiten von Attributen in einem Schlüsselkandidaten von Attributen in einem anderen Schlüsselkandidaten. (Derartige Probleme treten natürlich nur auf, wenn es mehrere Schlüsselkandidaten gibt!)

Um auch diese funktionalen Abhängigkeiten zu behandeln, wird die Boyce-Codd Normalform eingeführt:

Boyce-Codd Normalform:

Eine Relation ist in Boyce-Codd Normalform (BCNF), wenn jede Determinante (Quelle einer funktionalen Abhängigkeit) ein Schlüsselkandidat ist.

- **Ist eine Tabelle in BCNF, so ist sie aus der Sicht der funktionalen Abhängigkeiten "sauber"**
- **Es gibt allerdings Relationen, die sich nicht verlustfrei und abhängigkeitserhaltend in Relationen in BCNF zerlegen lassen!**

Für die folgende Gruppe der Normalformen ist der Begriff der mehrwertigen Abhängigkeit wichtig:

Mehrwertige Abhängigkeit (Multi Valued Dependency):

Eine Relation R bestehe aus drei paarweise disjunkten Attributkombinationen A,B,C. Dann heißt B mehrwertig abhängig von A (A →→B), wenn die Menge der B-Werte, die es zu einem Wertepaar (A,C) gibt, nur von A abhängig ist, nicht aber von C.

Ausgehend vom Begriff der mehrwertigen Abhängigkeit wird der Begriff der vierten Normalform eingeführt:

Vierte Normalform:

Eine Relation ist in vierter Normalform (4NF), wenn sie in BCNF ist und für jede mehrwertige Abhängigkeit A→→B diese entweder trivial ist (B ist Teilmenge von A oder die Relation ist die Vereinigung von A und B) oder A Schlüsselkandidat ist.

Ist eine Relation in 4NF, so gibt es keine Möglichkeit, diese Relation verlustfrei in 2 Relationen zu zerlegen!

Es ist aber noch nicht gesagt, daß es keine Möglichkeit gibt, diese Relation verlustfrei in mehr als zwei Relationen zu zerlegen. Um auch diese Fälle (die eher von theoretischem Interesse sind) zu behandeln, wird der Begriff der fünften Normalform eingeführt.