1. Allgemeines Datenbankmodell

1.1 Datenbankmanagementsysteme als Weiterentwicklung der Dateisystemfunktionalität eines Betriebssystems

Standardbetriebssysteme (Windows, iOS, UNIX/Linux, …) bieten dem Anwendungsprogrammierer und Benutzer wichtige Kernfunktionalitäten[1] an:

- das **Prozesssystem** zur Verwaltung zur Ausführung gebrachter Programme (z.B. um nebenläufige Programme (Multitasking / Multiprogramming) zu unterstützen),
- das **Dateisystem**, um alle **Dateien** eines Rechners für den lesenden bzw. schreibenden Zugriff unabhängig von der Hardware des Hauptspeichers (RAM) bzw. der Sekundärspeichermedien (Festplatte, USB-Stick, CD-ROM, …) zu verwalten.

Def.1: Eine **Datei** ist eine nach sachlichen Gesichtspunkten aufgebaute Sammlung von Daten. Dateien werden vom Betriebssystem in Verzeichnissen zusammengefasst und unter einem **Dateinamen**, eventuellen Hinweisen auf die **Organisation** der Datei (z.B. über die Endung des Dateinamens (.txt, .doc, .xml[2], .htm / .html, …)), dem **Schutzsta-**

[1] vgl. Carsten Vogt: „Betriebssysteme“, Heidelberg (Spektrum Akademischer Verlag) 2001, ISBN 3-8274-1117-3, S. 47ff., S. 171ff..

[2] XML := Extensible Markup Language.

tus (z.B. in Linux: **r**, **w**, **x** : der Benutzer darf die Datei lesen (**read**), darf in die Datei schreiben (**write**) bzw. darf die Datei ausführen (**execute**)) und dem **Adressbereich** (Startadresse, Länge), worunter sich die Datei auf dem Speichermedium befindet, verwaltet. Die Startadresse einer Datei wird vom Betriebssystem gekapselt und ist für den Benutzer bzw. das Anwendungsprogramm nicht sichtbar.

Dateien sind in der Regel höchst unterschiedlich strukturiert bzw. organisiert: Nach ihrem **inneren Aufbau** können Dateien dahin gehend unterschieden werden, ob sie aus **Datensätzen** aufgebaut sind (**semistrukturiert**, z.B. CSV-Dateien[3], XML-Dateien mit DTD, …) oder es nicht sind (**unstrukturiert**, z.B. Pixeldateien für Bilder oder Audiodaten, Fließtextdateien, …). Bestehen alle Datensätze einer semistrukturierten Datei aus einer gleichen Folge von **Attributen** und kann jedem Attribut ein Datentyp zugeordnet werden, dann spricht man von **strukturierten Dateien**. Für eine strukturierte Datei hat man vereinfacht folgenden Aufbau:

→ Eine strukturierte Datei besteht aus **n** Datensätzen. ($n \in \mathbb{N}$)

→ → Ein Datensatz besteht aus **m** Attributen ($m \in \mathbb{N}$)

BSP.1: Zur Veranschaulichung der Begriffe betrachten wir folgende Textdatei PERS.CSV, mit der einfache Personaldaten einer Firma verwaltet werden. Diese Datei ist als CSV-Datei organisiert. Ein Datensatz PERSDS besteht aus den Attributen PNR

[3] CSV:= Comma Separated Value.

(Personalnummer), PNAM (Name der Person), GEHALT (Monatsgehalt), EJAHR (Jahr des Eintritts in die Firma):

```
1023;Müller;4380.20;2005
1056;Mayer;3741.90;2009
1015;Huber;5134.15;2003
1044,Schmidt;4590.30;2011
```

Eine solche Datei würde in einer Firma typischerweise im Mehrprogrammbetrieb verarbeitet, wobei die Programme teilweise nacheinander und teilweise nebenläufig arbeiten (s. Abb.1):

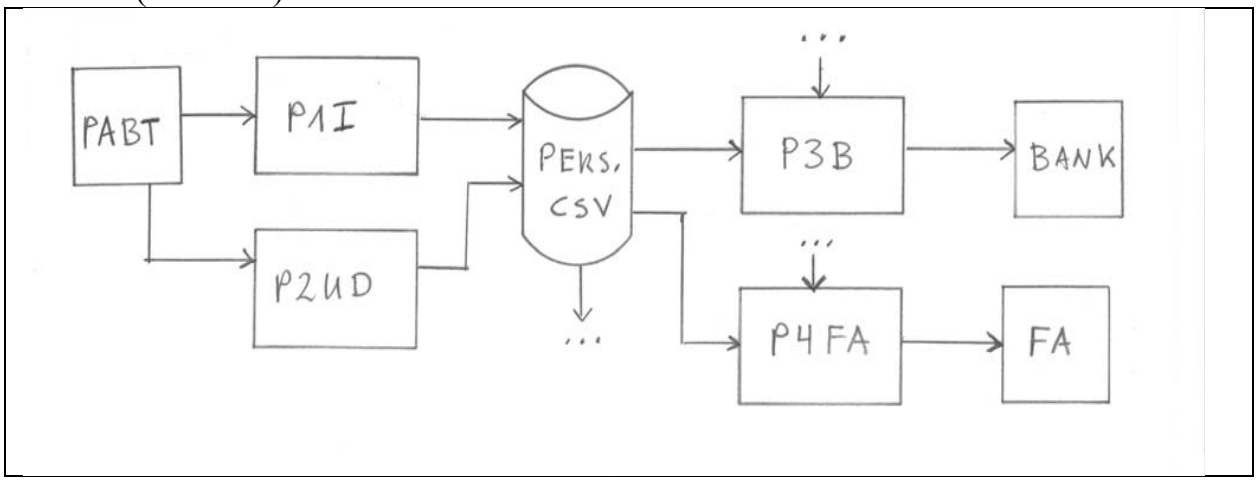


Abb.1: Vereinfachtes Systemübersichtsdiagramm eines Personalverwaltungssystems.

Legende zur Abb.1:

```
PABT  := Personalabteilung
FA    := Finanzamt
PI1   := Programm zur Einstellung eines Mitarbeiters
PI2UD := Programm zur Änderung von Personaldaten
P3B   := Programm zur Gehaltsüberweisung
P4FA  := Programm zur Abführung der Einkommensteuer
```

Die Struktur (Grammatik) eines Personaldatensatzes PERSDS kann formal z.B. als Backus-Naur Form (BNF)[4] beschrieben werden:

<PERSDS>::=<PNR>;< PNAM>;<GEHALT>;< EJAHR>

Def.2: Die Angaben zur Strukturbeschreibung einer Datei, die insbesondere den Aufbau von Datensätzen beschreibt und alle Attribute von Datensätzen benennt, heißen **Metadaten** einer Datei.

Ein **zentrales Problem** von Betriebssystemen ist, dass Dateisysteme standardmäßig nicht mit einer Verwaltung von Metadaten für strukturierte Dateien ausgestattet ist. Neben anderen Gründen hat dieses in der Geschichte der Informatik zur Entwicklung

[4] Vgl. Gregor Büchel: „Praktische Informatik – Eine Einführung, Lehr- und Arbeitsbuch mit Tafelbildern“, Wiesbaden (Springer Vieweg) 2012, ISBN 978-3-8348-1874-4. Kap.13: Beschreibung einer Grammatik durch eine BNF, S.203ff.

von Datenbankmanagementsystemen als Weiterentwicklung der Dateisystem - funktionalität eines Betriebssystems geführt.

Zur Funktionsfähigkeit eines Programmsystems, das Dateibestände gemeinsam im Mehrprogrammbetrieb nutzt, ist es notwendig, dass alle Anwendungsprogramme die gleiche Strukturbeschreibung der Dateibestände nutzen können und dass dafür der Metadatenbestand auch maschinell verfügbar ist.

1.2 Anforderungen an die Funktionalität eines Datenbankmanagementsystems

Def.2: Eine **Datenbank** (DB) ist eine nach sachlichen Gesichtspunkten aufgebaute Sammlung von strukturierten Daten, die unabhängig von Anwendungsprogrammen verwaltet werden[5]. Ein **Datenbankmanagementsystem** (DBMS) ist ein System von Software zur Verwaltung von Datenbanken. Ein **Datenbanksystem** (DBS) ist der Verbund von einem DBS mit einer Vereinigung von Datenbanken.

In Kurzschreibweise: Datenbanksystem = DB + DBMS.

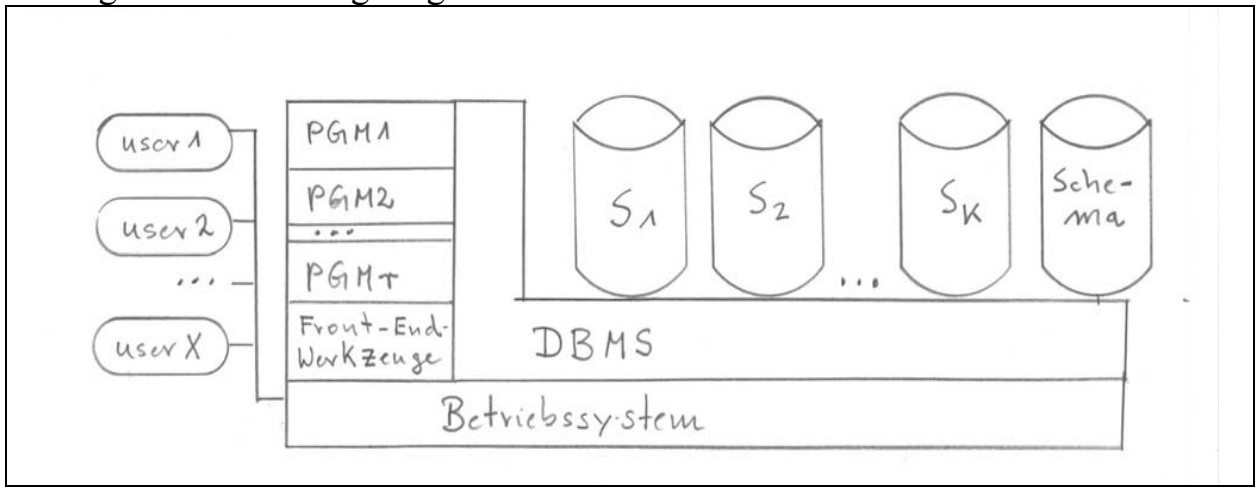
Eine Datenbank DB besteht in der Regel aus mehreren Datenbanksegmenten Si ($1 \leq i \leq k$). Je nach Datenbankmodell können die Segmente unterschiedlich aufgebaut sein:

---

[5] Die Anwendungsunabhängigkeit (Kapselung) einer Sammlung strukturierter Daten wird auch **logische Datenunabhängigkeit** genannt.

Datenbankmodell	**Segmentart**
relational	Tabelle
objektorientiert	Extent einer persistenten Klasse (= Sammlung der serialisierten Objekte einer persistenten Klasse)
fileorientiert (z.B. bei XML-Datenbanken)	Datei

Ein allgemeines Datenbankmodell, das den Mehrprogrammbetrieb unterstützt, ist in nachfolgender Abbildung dargestellt:



Die Anforderungen an die **Funktionalität** eines DBMS können in folgenden zehn Funktionen zusammengefasst werden:

1. Persistenz: Unter Persistenz versteht man die **dauerhafte und strukturtreue Speicherung von Datenbeständen**. Dauerhafte Speicherung wird auf Sekundärspeichermedien (z.B. Festplatten) ausgeführt. Die strukturtreue Speicherung von Datensätzen erfordert, dass komplexe Datentypen, die solche Datensätze beschreiben ohne besonderen Programmieraufwand gespeichert und gelesen werden können.

BSP.2: Ein JAVA-Datentyp, um die Inhalte eines Personaldatensatzes PERDS, wie er in BSP.1 dargestellt wurde, ohne besonderen Programmieraufwand strukturtreu zu speichern, ist folgende serialisierte Klasse:

```
class PERSDS implements Serializable
{ int pnr;          /* PNR: Personalnummer   */
  String pnam;      /* PNAM: Name der Person */
  double gehalt;    /* GEHALT: Monatsgehalt  */
  int ejahr;        /* EJAHR: Jahr des Eintritts in die Firma */
}
```

Das **strukturtreue** Schreiben bzw. Lesen in **einem Schreibvorgang** bzw. in **einem Lesevorgang** pro Instanz (eine Instanz entspricht einem Datensatz) wird für serialisierte Klassen in JAVA mit den Methoden **writeObject()** bzw. **readObject()** der Klassen **ObjectOutputStream** bzw. **ObjectInputStream** implementiert.

Ohne DBMS müssen Instanzen dieser Klasse, wenn man sie in eine Datei schreibt bzw. sie aus einer Datei liest, durch einen **besonderen Programmieraufwand** verwaltet werden. Im Mehrprogrammbetrieb hat man hier das Problem der **Datenabhängigkeit**: Alle lesenden Programme und alle weiterverarbeitende Programme (Update- und Lösch-Programme) müssen die innere Funktionsweise des ursprünglichen schreibenden Programms kennen, um korrekt auf die gespeicherten Daten zugreifen zu können. Ziel ist die **Datenunabhängigkeit:** Hierbei braucht kein Programm die innere Funktionsweise des ursprünglichen schreibenden Programms zu kennen.

Bei einem DBMS wird die dauerhafte und strukturtreue Speicherung von Datenbeständen durch eine Metadatenverwaltung gesteuert. Hierzu genügt es, den Datentyp der Daten, die dauerhaft gespeichert werden sollen, in einen Datentyp zu transformieren, der im Data-Dictionary des DBMS eingetragen wird (siehe Funktionalität 3. Verwaltung eines Schema-Katalogs (Data-Dictionary)).

2. Sekundärspeicherverwaltung: Die Sekundärspeicherverwaltung implementiert Direktzugriffe. **Direktzugriff** bedeutet, dass neben gewissen RAM-Operationen für den lesenden Zugriff (z.B. binäre Suche in einer doppelt verketteten Liste oder Suche in einem flachen Suchbaum) nur **ein** lesender Zugriff auf einen Nutzdatensatz, der auf einem **Sekundärspeichermedium** liegt, erforderlich ist.

Anm.1: In der Regel wird eine Sekundärspeicherverwaltung mit Direktzugriff über eine **Indexverwaltung** implementiert. Hierbei wird neben der Nutzdatendatei (z.B. PERS.DAT) eine **Indexdatei** (z.B. PERS.IDX) aufgebaut. Um eine Indexdatei aufzubauen benötigt man die Offsets der einzelnen Nutzdatensätze. Ein **Offset** ist die Position des ersten Bytes eines Datensatzes relativ zum Dateianfang.

PERS.DAT:

Offset	PNR	PNAM	GEHALT	EJAHR	…
0	1023	Müller	4380.20	2005	…
100	1056	Mayer	3741.90	2009	…
170	1015	Huber	5134.15	2003	…
230	1051	Mayer	3880.95	2011	…
300	1044	Schmidt	4590.30	2011	…

In der Nutzdatendatei wird der Offset nicht gespeichert, er kann durch Betriebssystemfunktionen des Dateisystems oder des DBMS ermittelt werden. Eine Indexdatei besteht aus Indizes. Ein Index ist ein Paar der Form **(Schlüsselwert, Offset)**. Der Schlüsselwert ist bezogen auf ein Attribut des Nutzdatensatzes. Z.B. **PNR** (eindeutig) oder **PNAM** (mehrdeutig). In einer Indexdatei stehen die Paare in einer nach Schlüsselwerten sortierten Folge. Z.B. PERS.IDX ist nach PNR-Werten aufsteigend sortiert. Zugriffe mittels einer sortierten Indexfolge nennt man **ISAM** (**i**ndex **s**equential **a**ccess **m**ethod).

PERS.IDX:

PNR	Offset
1015	170
1023	0
1044	300
1051	230
1056	100

Um beim Lesen Direktzugriffe ausführen zu können, ist beim Einfügen ein erhöhter Aufwand an RAM-Operationen erforderlich, der aber gegenüber Leseoperationen auf einem Sekundärspeicher zeitlich sehr klein ist.

BSP.3: Die Indexdatei wird für eine Arbeitssitzung mit mehreren Einfüge- und Leseoperationen in den RAM als doppelt verkettete Liste geladen. Die Liste wird mit **binärer Suche** durchsucht.

Suche: Person mit PNR = 1051.
=> 2 Vergleichsschritte in der Indexliste (RAM-Operationen).
=> 1 Direktzugriff auf Nutzdatensatz (hier Offset = 230) (Operation auf Sekundärspeicher).

Einfügen eines neuen Datensatzes:

=> 1 Datensatz ans Ende der Nutzdatendatei anhängen (Datensatz: PNR:1027; PNAM:Schmitz; … wird bei OFFSET 360 eingefügt): 1 Operation auf Sekundärspeicher.
=> Indexdatensatz einfügen (Das Paar (1017,360) in PERS.IDX): Lineare Suche in doppelt verketteter Liste: Aufwand ~ n/2 RAM-Operationen.

Anm.2: Als Praktikumsversuch wird eine Sekundärspeicherverwaltung mit einer ISAM-Verwaltung aufgebaut. Hierbei wird die JAVA-Klasse **RandomAccessFile** aus dem Paket **java.io** verwendet.

Anm.3: Je nach DBMS kann die Indexverwaltung im RAM unterschiedlich organisiert sein: Bei **relationalen Datenbanken** ist der Index im RAM als **Suchbaum** (Varianten von Bayer-Bäumen (≅ (2m+1)-Bäumen)) organisiert. Bei bestimmten NoSQL-datenbanken (z.B. Cassandra) ist er als Hash-Tabelle organisiert, d.h. die Position, an der das Paar (Schlüsselwert, Offset) in der Tabelle einsortiert ist, wird als Funktion des Schlüsselwerts berechnet.

3. Verwaltung eines Schema-Katalogs (Data-Dictionary): Der Schema-Katalog enthält sämtliche Metadaten einer DB. Abhängig vom Datenmodell, das dem DBMS zu Grunde liegt, werden die **Datenbanksegmente**, aus der die DB aufgebaut ist, die ihnen zugehörigen **Entitäten** und die **Attribute**, aus denen die Entitäten aufgebaut sind, definiert.

BSP.4: Tabellarische Übersicht zu den Schemaelementen in Abhängigkeit vom Datenmodell des DBMS:

DBMS-Daten-modell	**Segmenttyp**	**Entität**	**Attributtyp**
relational	Tabellentyp	Zeile	Datentyp einer Spalte
objektorientiert	Persistente Klasse	Instanz einer persistenten Klasse	Datentyp eines Klassenattributs
XML-Datenbank	Dokumenttyp eines XML-Wurzelelements	Ein von der Wurzel abhängiges XML-Element	Elementdefinition und Attributedefini-tion des abhängigen XML-Elements
fileorientiert (z.B. bei hierar-chischen Daten-banken)	Dateityp	Datensatz	Datentyp eines Datensatzattributs
graphorientiert	Knotentyp und Kantentyp	ein Knoten bzw. eine Kante	Datentyp einer Knoten- bzw. einer Kantenfärbung

Für alle Segmente einer DB werden im Schemakatalog der Segmentaufbau (d.h. der Segmenttyp bzw. das **Segmentschema**) definiert. Zum Segmentschema gehören die folgenden Angaben:

a) Der **Segmentname**.
b) Eine Strukturbeschreibung, wie die **Entitäten** der Segmente aus **Attributen** aufgebaut sind.
c) Die **Datentypen** aller **Attribute**, aus denen die Entitäten bestehen und Einschränkungen ihrer Wertebereiche (Integritätsbedingungen).
d) **Angaben,** wie verschiedene Segmente einer Datenbank untereinander logisch **verknüpft** sind.

BSP.5: Die Instanzen der Klasse **Artikel** sollen in einer Datenbank DB1 gespeichert werden. Die Datenbank DB1 soll ein relationales Datenmodell haben. Jeder Artikel, d.h. jede Instanz der Klasse **Artikel** besteht aus **n** Zutaten **(n ∈ℕ)**. Jede Zutat ist Instanz einer Klasse **Zutat**. Beide Klassen sind nachfolgend gegeben:

```
class Artikel
{ int artnr;      /* Artikelnummer */
  String artname;
  double preis;
  LinkedList<Zutat> zutaten;
}
```

```
class Zutaten
{int zutnr;       /* Zutatennummer        */
 String zubez;    /* Name der Zutat       */
 double quant;    /* Menge der Zutat      */
 String einheit;  /* Maßeinheit der Zutat */
}
```

Anm.4: Logische Verknüpfungen zwischen Segmenten (d.h. Tabellen) einer relationalen Datenbank werden durch spezielle Attribute, sog. **Schlüsselattribute** (engl. **key**) hergestellt. Um eine logische Verknüpfung der Form „ zu einer Entität vom Typ A gehören **n** Entitäten vom Typ B“ herzustellen geht man zweistufig vor:

(1) Man definiert unter der Menge der Attribute von A und B jeweils ein Attribut mit eindeutiger Wertefolge. Dieses ist das **Primärschlüsselattribut** (primary key =: PRIK).

(2) Man definiert in dem Entitätentyp B ein Attribut **x**, dass nur Werte des PRIK-Attributs von A annehmen kann. Hierbei können in **n** Entitäten von B ein gleicher PRIK-Wert angenommen werden. Ein solches Attribut **x** heißt **Fremdschlüsselattribut** (foreign key =: FKEY).

Eintragungen ins Data-Dictionary der DB1:

E0) Name von DB1 = ARTIKELDB

E1.1) Name der Artikeltabelle = ARTTAB

E1.2) Eintragungen für die Attribute von ARTTAB:

Attribut	**Datentyp**[6]	**Bedingung**
artnr	int	PRIK
artname	char(50)	Ø
preis	float[7]	Ø

E2.1) Name der Zutatentabelle = ZUTTAB

E2.2) Eintragungen für die Attribute von ZUTTAB

Attribut	**Datentyp**	**Bedingung**
zutnr	int	PRIK
zubez	char(50)	Ø
quant	float	Ø
einheit	char(10)	Ø
zartnr	int	FKEY(ARTTAB.artnr)

[6] Angabe des Datentyps bezogen auf das relationale Datenmodell.

[7] Hier ist das Attribut **preis** als Gleitpunktzahl modelliert. In der Datenbankwelt werden für Anwendungen in der Finanzwirtschaft bevorzugt **Festpunktzahlen** verwendet. Diese können mit einem Datentyp **decimal(p,q)** modelliert werden. Hierbei ist **p** die Anzahl aller Dezimalstellen und **q** die Anzahl der Nachkommastellen. In Java kann dieses mit dem Datentyp **java.math.BigDecimal** modelliert werden.

4. Integritätskontrolle: Integrität bedeutet, dass schreibende Operationen eine DB von einem korrektem Zustand der Daten nur in einen neuen korrekten Zustand überführen dürfen. Generell werden drei Arten von schreibenden Operationen unterschieden:

(1) Einfügen von neuen Entitäten (=: INSERT).
(2) Ändern von vorhandenen Entitäten (=: UPDATE)
(3) Löschen von vorhandenen Entitäten (=: DELETE)

Um die Integrität zu sichern, gibt es im Data-Dictionary die Möglichkeit, **Integritätsbedingungen** festzulegen: Man unterscheidet drei Arten von Integritätsbedingungen:

(1) **Typintegrität:** Bei den schreibenden Operationen INSERT und UPDATE dürfen in ein Attribut **A** mit Datentyp **dtA** nur Werte vom Typ **dtA** geschrieben werden. Längenbedingungen, wie z.B.: **n** beim **dtA = char(n)** oder **p,q** beim **dtA = decimal(p,q)** (Datentyp für rationale Festpunktzahlen) gehören zur Datentypangabe
(2) **Referentielle Integrität:** Wenn zwei Entitäten aus zwei Segmenten durch eine Fremdschlüsselbedingung (FKEY) aufeinander Bezug nehmen, darf bei schreibenden Operationen die Fremdschlüsselbeziehungen nicht zerstört werden. **BSP.** (s. BSP.5): Beim INSERT bzw. UPDATE einer Zeile in ZUTTAB wird geprüft, ob der Wert des Attributs **zartnr** in der Spalte ARTTAB.artnr vorkommt. Falls nein, wir die schreibende Operation abgewiesen. Beim

DELETE einer Zeile in ARTTAB wird geprüft, der **artnr**-Wert als **zartnr**-Wert einer Zeile von ZUTTAB vorkommt. Falls ja, wird der DELETE-Versuch abgewiesen.

(3) **NULL Integrität:** Bei Attributen wird unterschieden, ob es MUSS-Attribute oder KANN-Attribute sind. Bei einem MUSS-Attribut muss das Attribut mit einem zulässigen Wert gemäß Wertebereich gefüllt sein. Bei einem KANN-Attribut, kann das Attribut leer bleiben. D.h. es wird in diesem Fall mit einem „leeren Wert“ (NULL-Wert) gefüllt. Der NULL-Wert kann mathematisch mit der leeren Menge verglichen werden. Er kann je nach Datentyp **nicht** mit der Zahl 0 oder dem leeren String (““) verglichen werden.

(4) **Sachliche Integrität** (Wertintegrität): Mit einer sachlichen Integritätsbedingung **IBed(A)** kann eine anwendungsbezogene **Einschränkung** des Wertebereichs **WdtA** des gegebenen Datentyps **dtA** des Attributs **A** durchgesetzt werden. Der durch **IBed(A)** eingeschränkte Wertebereich **WA** ist mathematische Teilmenge von **WdtA:** $\mathbf{WA \subseteq WdtA}$

BSP.6: Gegeben ist das Attribut PLZ (deutsche Postleitzahl) einer Tabelle KUNDEN. Der Datentyp von PLZ ist **dtPLZ = int**. Der Wertebereich des Datentyps **int** ist:
$\mathbf{Wint = \{ z \in \mathbb{Z} \mid -2^{31} \leq z \leq 2^{31}-1\}}$ **(2^{31} ≈ 2,1 Milliarden).**

Die sachliche Integritätsbedingung für PLZ ist **IBed(PLZ)** : $\mathbf{1000 \leq z \leq 99999}$. Damit gilt für den Wertebereich des Attributs PLZ $\mathbf{WPLZ = \{ z \in Wint \mid 1000 \leq z \leq 99999\}}$.

Daran ist auch ersichtlich, dass die Integritätsbedingung den Wertebereich des Datentyps für den Wertebereich des Attributs einschränkt. Es gilt nämlich:

WPLZ ⊆ Wint

Anm.5: Allgemein kann jedes Attribut in einen Schema-Katalog mittels eines Tripels **(A, dtA, IBed(A))**, wobei **A** das Attribut, **dtA** der Datentyp von **A** und **IBed(A)** die Integritätsbedingung von A ist, eingetragen werden. Liegt keine Integritätsbedingung vor, wird die leere Bedingung **Ø** eingetragen.

BSP.7: In einer Tabelle KUNDE sollen die Entitäten, die in einem Anwendungsprogramm als Instanzen einer Klasse **Kunde** auftreten, persistent gespeichert werden.

```
class Kunde
{ int knr;      /* Kundennummer */
  String name;  /* Kundenname   */
  int plz;      /* Kunden-PLZ   */
}
```

Die Einträge in den Schema-Katalog können für die Attribute der Tabelle KUNDE analog zu E1.2) und E2.2) tabellarisch spezifiziert werden:

E3.2) Einträge in den Schema-Katalog für die Attribute der Tabelle KUNDE

Attribut	**Datentyp**	**Bedingung**
knr	int	PRIK
name	char(50)	Ø

plz	int	$1000 \leq plz \leq 99999$

Oder sie können in Tripelnotation angegeben werden[8]:
AF(KUNDE)={ (knr,int, PRIK), (name, char(30), Ø), (plz, int, $1000 \leq plz \leq 99999$)**}**

5. Interpretation einer Anfragesprache: Um auf Daten einer DB lesend oder schreibend zugreifen zu können, stellt das DBMS eine Anfragesprache zur Verfügung. Das DBMS enthält einen Interpreter für Kommandos der Anfragesprache. Die Anfragesprache ist abhängig vom Datenmodell des DBMS.

DBMS-Daten-modell	**Kürzel**	**Anfragesprache**
relational	SQL	Structured Query Language
objektorientiert	OQL	Object Query Language
XML nativ	XQUERY	XQUERY
hierarchisch	DLI	Data Language Interface
graphorientiert	GQL	Graph Query Language

[8] AF steht hier für den Eintrag der **A**ttribut**f**olge in das Schema der Tabelle KUNDE.

Anm.6: Für Anfragesprachen gibt es Einbettungen in höhere Programmiersprachen:
a) für SQL: ESQL/C (in C), ODBC (in C++) <*Open Database Connectivity*>, JDBC (in Java) <*Java Database Connectivity*> (vgl. Paket: java.sql).
b) für OQL: Z.B. db4O/OQL.

6 . Verwalten von Benutzersichten (VIEW): Jede DB hat in der Regel verschiedene Benutzergruppen. Jede Benutzergruppe hat eventuell unterschiedliche Anforderungen für lesende und schreibende Zugriffe auf Segmente bzw. einzelne Attribute. Weiterhin kann für bestimmte Benutzergruppen festgelegt sein, dass sie nur verdichtete Attributwerte lesen. können. Diese unterschiedlichen Formen des Zugriffs nennt man Benutzersicht.

BSP.8: Wir betrachten die Tabellen ARTTAB und KUNDE. Weiterhin eine Tabelle UMSATZ. Die Tabelle ARTTAB ist noch um das Attribut **gfken** (Gefahrgutkennzeichen) ergänzt. Die Tabelle UMSATZ hat folgende Schema-Einträge:

Attribut	**Datentyp**	**Bedingung**
umnr	int	PRIK
uartnr	int	FKEY(ARTTAB.artnr)
uknr	int	FKEY(KUNDE.knr)
umenge	float	Ø
wert	decimal(11,2)	Ø
uwoche	int	$1 \leq$ uwoche ≤ 52

In Bezug auf diese drei Tabellen kann es in einer Firma unterschiedliche Benutzersichten bei lesenden und schreibenden Zugriffen geben. Z.B.:

Benutzergruppe	**Sichten**
System	Insert (umnr; knr, artnr)
Vertrieb	Read(umnr, knr, artnr), Write(umenge, wert)
Kundenbeauftragter	Read(umnr, knr, sum(wert) über alle Artikel für einen Kunden), Write(name, plz)
Transportplaner	Read(umnr, artnr, gfken, sum(umenge) pro plz)
Werksfeuerwehr	Read(artnr), Write(gfken)
Produktmanager	Read(artnr, sum(wert) über alle Kunden für einen Artikel) Write(preis, artname)

7. Datenschutz: Datenschutz bedeutet nach Bundesdatenschutz den Schutz der Daten in der Datenbank vor dem Zugriff unbefugter Dritter. D.h. das DBMS muss über Sicherungsmechanismen, wie z.B. Authentifizierung verfügen. Während im Dateisystem eines Betriebssystems der Zugriffsschutz nur auf eine **gesamte** Datei funktioniert (z.B. ein Benutzer ist der einzige, der auf eine Datei Schreibrechte hat), ist es im DBMS möglich, Schreibrechte (I, U, D) und Leserechte (S) auf **einzelne Attribute** Benutzer bzw. Benutzergruppen festzulegen.

8. Transaktionsverwaltung: Eine Transaktion ist eine Folge von schreibenden DB-Zugriffen, die zu einer Gruppe zusammen gefasst werden. Für diese Gruppe gilt: Entweder werden alle Zugriffe ausgeführt oder es wird kein Zugriff ausgeführt.

BSP.9: Eine Transaktion T für eine Löschoperationen auf die obige Artikel-Tabelle und die zugehörigen Zeilen in der Zutaten-Tabelle hat folgenden Aufbau:
T = [dz_1, dz_2, ..., dz_N, dA] Hierbei ist **dz_i** ein DELETE auf eine Zutatenzeile ($1 \leq i \leq N$) und **dA** ein DELETE auf die zugehörige Artikelzeile. Falls eine dieser Löschoperationen scheitern würde, wäre die DB in einem inkonsistenten Zustand. Daher können alle Löschoperationen von T nur gemeinsam ausgeführt werden oder die DB bleibt in einem Störungsfall (z.B. Stromausfall) im alten Zustand.

Die Realisierung der Transaktionsverwaltung verlangt, dass alle Schreib-Operationen in einer LOG-Datei notiert werden, mit der im Falle eines fehlerhaften Transaktionsendes die DB in ihren ursprünglichen Zustand zurückversetzt werden kann (**Rollback**). Die Ausführung einer ganzen Transaktion nennt man **commit**.

Wikipedia charakterisiert Transaktionen durch die sog. ACID-Eigenschaften: „Bei der Ausführung von Transaktionen muss das Transaktionssystem die ACID-Eigenschaften garantieren:

- *Atomarität* (**A**tomicity): Eine Transaktion wird entweder ganz oder gar nicht ausgeführt. Transaktionen sind also „unteilbar“. Wenn eine atomare Transaktion abgebrochen wird, ist das System unverändert.

- *Konsistenz* (**C**onsistency): Nach Ausführung der Transaktion muss der Datenbestand in einer konsistenten Form sein, wenn er es bereits zu Beginn der Transaktion war.
- *Isolation* (**I**solation): Bei gleichzeitiger Ausführung mehrerer Transaktionen dürfen sich diese nicht gegenseitig beeinflussen.
- *Dauerhaftigkeit* (**D**urability): Die Auswirkungen einer Transaktion müssen im Datenbestand dauerhaft bestehen bleiben. Die Effekte von Transaktionen dürfen also nicht „mit der Zeit verblassen“ oder „aus Versehen verloren gehen“. Eine Verschachtelung von Transaktionen ist wegen dieser Eigenschaft streng genommen nicht möglich, da ein Zurücksetzen (Rollback) einer äußeren die Dauerhaftigkeit einer inneren, bereits ausgeführten Transaktion verletzen würde.“ [Wikipedia-Seite: Transaktion_(Informatik), http://de.wikipedia.org/wiki/Transaktion_(Informatik) , letzter Besuch: 9.10.2013].

9. Synchronisation: An das DBMS besteht die Anforderung, zeitlich konkurrierende Zugriffe (Delete, Insert, Update, Select) auf eine Entität (z.B. eine Tabellenzeile) zu steuern. Für diese Steuerung müssen Regelmechanismen eingerichtet sein (z.B.: „Schreiben geht vor Lesen“). Diese Mechanismen sind in ihren Algorithmen ähnlich zu entsprechenden Mechanismen in einem Betriebssystem (z.B. Verfahren zum wechselseitigen Ausschluss von Programmzugriffen auf eine gemeinsame Datei). Diese

Mechanismen werden in einem DBMS zusammen mit den Methoden der Sekundärspeicherverwaltung implementiert.

10. Datensicherung (Recovery): In festgelegten Arbeitsperioden (Arbeitstagen, Stunden, Minuten) werden bestimmte oder alle Segmente einer DB auf schnellschreibenden Sekundärspeichermedien gesichert. Dafür verfügt das DBMS über Import- und Exportmechanismen. Die Datensicherung ist die Grundlage für die Recoveryfähigkeit des DBMS nach Systemausfall. Die Import- / Exportmechanismen unterstützen die strukturtreue Speicherung von DB-Segmenten in sequentiellen Dateien. Hierbei werden sowohl die Nutz- als auch die Metadaten der DB-Segmente gesichert. Für diese Mechanismen werden typische Austauschformate verwendet (Z.B. .CSV, .XML, .SQL).

Lernziele zu Kap.1: Allgemeines Datenbankmodell

1. **Unterschiede zwischen der Datenverwaltung in einem Dateisystem eines Betriebssystem und der Datenverwaltung in einem DBMS benennen können.**
2. **Die zehn Anforderungen an die Funktionalität eines DBMS erläutern können.**