Zusammenfassung Modul 104

Datenmodell implementieren

Version: 01.00
Freigabe: 20.05.11

Janik von Rotz
Hoheneich 4, 6064 Kerns
Internet www.janikvonrotz.ch
Email contact@janikvonrotz.ch

Inhaltsverzeichnis

Änderungskontrolle

Version	Datum	Autor	Beschreibung der Änderung	Status
<<#>>	<<Datum>>	<<Name>>		

Referenzierte Dokumente

Nr.	Dok-ID	Titel des Dokumentes / Bemerkungen
<<#>>	<<#>>	<<Titel/Name des Dokumentes>>

Titel:	Zusammenfassung Modul 104	**Typ:**	Hanbuch	**Version:**	01.00
Thema:	Datenmodell implementieren	**Klasse:**	öffentlich	**Freigabe:**	20.05.11
Autor:	Janik von Rotz	**Status:**	Freigegeben	**PrtDat./gültig bis:**	20.05.11 / Mai 11
Ablage/Name:	c:\Dokumente und Einstellungen\ILZ32\Eigene Dateien\Dropbox\exchange\teil_abschluss_prüfungen\zusammenfassung\m104\modul104_zusammenfassung.docx			**Registratur:**	.

1. Datenbankgrundlagen

1.1 Datenbankmodell

Grundlage für die Strukturierung der Daten und ihrer Beziehungen zueinander ist das Datenbankmodell, das durch den DBMS-Hersteller festgelegt wird. Je nach Datenbankmodell muss das Datenbankschema an bestimmte Strukturierungsmöglichkeiten angepasst werden:

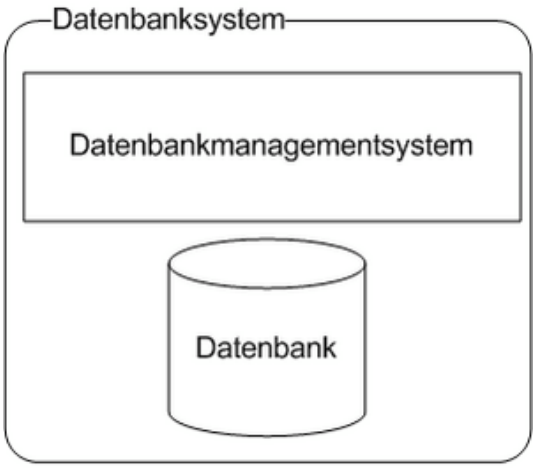
- **hierarchisch:** Die Datenobjekte können ausschließlich in einer Eltern-Kind-Beziehung zueinander stehen.
- **netzwerkartig:** Die Datenobjekte werden miteinander in Netzen verbunden.
- **relational:** Die Daten werden zeilenweise in Tabellen verwaltet. Es kann beliebige Beziehungen zwischen Daten geben. Sie werden durch Werte bestimmter Tabellenspalten festgelegt.
- **objektorientiert:** Die Beziehungen zwischen Datenobjekten werden vom Datenbanksystem selbst verwaltet. Objekte können Eigenschaften und Daten von anderen Objekten erben.

Es existiert eine Vielzahl von Misch- und Nebenformen, wie zum Beispiel das objektrelationale Modell.

1.2 Funktionen eines DBMS

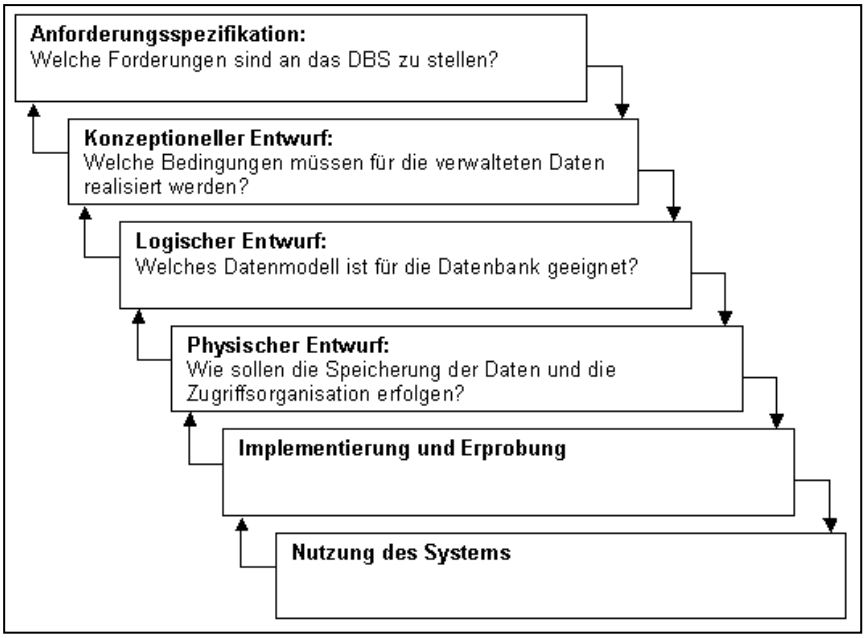
Die wesentlichen Funktionen von heutigen Datenbankmanagementsystemen sind:

- Speicherung, Überschreibung und Löschung von Daten
- Verwaltung der Metadaten
- Vorkehrungen zur Datensicherheit
- Vorkehrungen zum Datenschutz
- Vorkehrungen zur Datenintegrität
- Ermöglichung des Mehrbenutzerbetriebs durch das Transaktionskonzept
- Optimierung von Anfragen
- Ermöglichung von Triggern und Stored Procedures
- Bereitstellung von Kennzahlen über Technik und Betrieb des DBMS



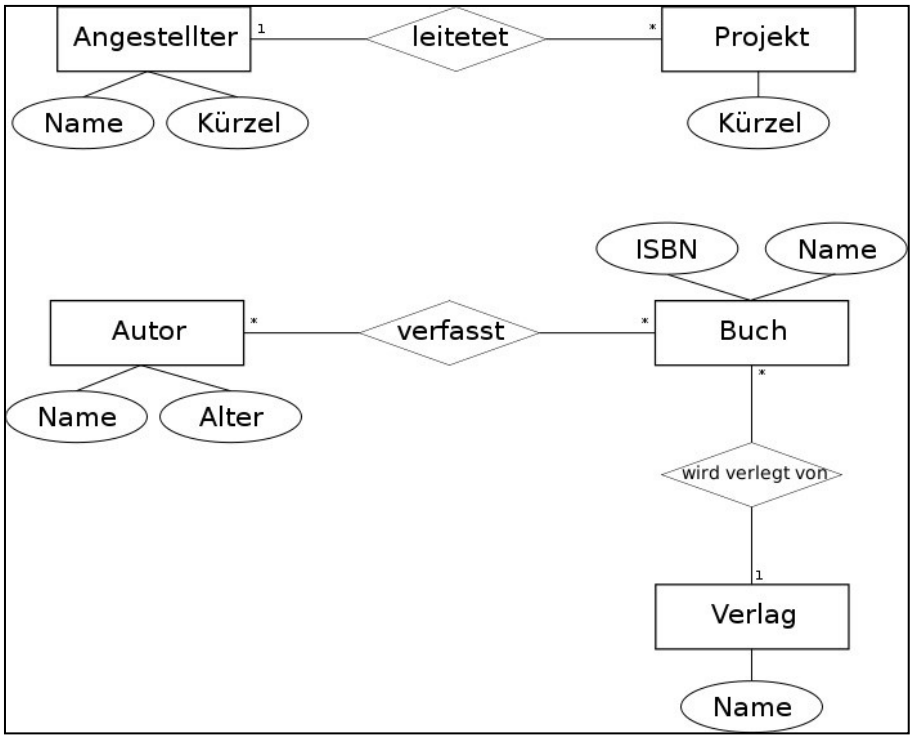
1.2.1 Datenbankentwurf

- Anforderungsanalyse und -spezifikation (Herausfinden was gewollt wird)
- Konzeptueller Entwurf (ER-Diagramm, Beschreibung etc.)
- Logischer Entwurf (Umsetzung in ein Daten-Modell wie z.B. das Relationelle)
- Datendefinition (Implementierung des Schemas mit DDL)
- Physischer Entwurf (Tuning, Indexe u.s.w)
- Wartung und Dokumentation



1.3 ER-Modell

Ein einfaches Modell als Beispiel



1.4 Normalisierungsprozess

1. Normalform: Eine Relation befindet sich in der ersten Normalform, wenn keines ihrer Attribute eine untergeordnete Relation darstellt und wenn alle Attribute nur atomare Werte beinhalten.

Studenten

Vorname	Nachname	Informatikkentnisse
Thomas	Müller	Java, C++, PHP
Ursula	Meier	PHP, Java
Igor	Müller	C++, Java

Ausgangslage

Resultat nach Normalisierung

Studenten

Vorname	Nachname	Informatikkentnisse
Thomas	Müller	C++
Thomas	Müller	PHP
Thomas	Müller	Java
Ursula	Meier	Java
Ursula	Meier	PHP
Igor	Müller	Java
Igor	Müller	C++

Beispiel 1. Normalform

2. Normalform: Laut Definition muß die Datenbank immer zuerst in die erste Normalform versetzt werden, bevor man diese in die 2. Normalform versetzen kann. Hierbei müssen alle nicht zum Schlüssel gehörenden Attribute von diesem voll funktional abhängig sein. Besteht ein Schlüssel aus mehreren Teilschlüsseln, so ist das Element aus dem Datensatz herauszuziehen, welches nur von einem Teilschlüssel abhängt.

Studenten

IDSt	Nachname	IDProf	Professor	Note
1	Müller	3	Schmid	5
2	Meier	2	Borner	4
3	Tobler	1	Bernasconi	6

Ausgangslage

Resultat nach Normalisierung

Studenten

ID	Nachname
1	Müller
2	Meier
3	Tobler

Professoren

IDProf	Professor
1	Bernasconi
2	Borner
3	Schmid

Noten

IDST	IDProf	Note
1	3	5
2	2	4
3	1	6

3. Normalform: Zusätzlich zur 2. Normalform gilt für jeden Schlüssel: Alle nicht zum Schlüssel gehörende Attribute sind nicht von diesem transitiv abhängig. Das bedeutet, daß alle Attribute nur vom Schlüsselattribut, nicht aber von anderen Attributen abhängig sein. Eine Abhängigkeit zwischen den Attributen muß aufgelöst werden.

Lieferant

ID	Name	Konto_Nr	Bank_Clearing_Nr	Bank

Ausgangslage

Resultat nach Normalisierung

Lieferant

ID	Name	Konto_Nr	Bank_Clearing_Nr

Bank

Bank_Clearing_Nr	Bank

Beispiel 3. Normalform

1.4.1 1. Normalform

Eine Relation R ist (1NF), wenn alle Attribute nur atomare Werte enthalten.

Studenten

Vorname	Nachname	Informatikkentnisse
Thomas	Müller	Java, C++, PHP
Ursula	Meier	PHP, Java
Igor	Müller	C++, Java

Ausgangslage

Resultat nach Normalisierung

Studenten

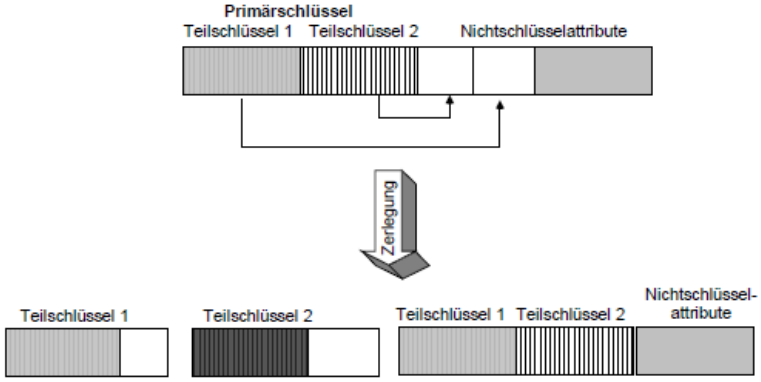
Vorname	Nachname	Informatikkentnisse
Thomas	Müller	C++
Thomas	Müller	PHP
Thomas	Müller	Java
Ursula	Meier	Java
Ursula	Meier	PHP
Igor	Müller	Java
Igor	Müller	C++

Beispiel 1. Normalform

Lösen Sie alle Tupel, die mehrwertige Attribute enthalten, auf, in dem Sie Tupel erzeugen, die nur einfache Attribute enthalte,
durch Duplizieren der ursprünglichen Tupel

1.4.2 2. Normalform

Eine Relation R mit Primärschlüssel S ist (2NF), wenn sie (1NF) ist und jedes Nicht-Schlüssel-Attribut voll funktional abhängig vom Primärschlüssel S ist.

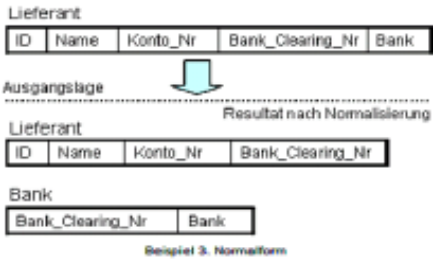


Funktionale Abhängigkeit

Bestimmen einige dieser Attribute eindeutig die Werte anderer Attribute, so spricht man von funktionaler Abhängigkeit. So könnte man sich etwa eine Kundendatenbank vorstellen, in der die Anschrift und die Telefonnummer eines Kunden eindeutig durch seinen Namen zusammen mit seinem Geburtsdatum bestimmt sind. Hier wären also Anschrift und Telefonnummer funktional abhängig von Name und Geburtsdatum.

1.4.3 3. Normalform

Zusätzlich zur 2. NF gilt für jeden Schlüssel: Alle nicht zum Schlüssel gehörende Attribute sind nicht von diesem transitiv abhängig. Das bedeutet, dass alle Attribute nur vom Schlüsselattribut, nicht aber von anderen Attributen abhängig sind. Eine Abhängigkeit zwischen den Attributen muss aufgelöst werden



KNR	Kurs	Bereich	BereichsNR
44	Deutsch	Sprache	1
45	Chemie	Naturwissenschaften	6
46	Sport	Gesellschaftswissenschaften	2
47	Mathematik	Naturwissenschaften	6
48	Informatik	Naturwissenschaften	6
49	Englisch	Sprache	1

In Tabelle ist der Bereich abhängig von der Bereichsnummer, da jedem Bereich immer nur eine bestimmte Nummer zugewiesen wird, es besteht also eine transitive Abhängigkeit zwischen den beiden Attributen.

Lösung: Um diese Relation in die dritte Normalform umzuwandeln, muß also diese transitive Abhängigkeit beseitigt werden. Das geschieht ebenfalls durch eine Auslagerung der Attribute BereichsNR und Bereich in eine eigene Relation / Tabelle.

KNR	Kurs	BereichsNR
44	Deutsch	1
45	Chemie	6
46	Sport	2

BereichsNR	Bereich
1	Sprache
2	Gesellschaft
3	Kunst

Hier die neu gebildeten 2 Relationen.

2. Relationales Datenbankmodell

Das Relationales Datenbankmodell ist ein Datenbankmodell, in dem Daten als Relationen dargestellt werden.

2.1 Relationen und Tupel

ine Relation kann mit einer Tabelle verglichen werden, deren Spalten einzelne Datenattribute darstellen. Ein Datenelement eines relationalen Datenbankmodells - vergleichbar mit einer Zeile einer Tabelle - wird als Tupel bezeichnet. Den Relationen und Tupeln liegt die Relationenalgebra zugrunde, wird aber durch neue Konzepte (z.B. Primärschlüssel und Fremdschlüssel) erweitert.

2.2 Integritätsbedingungen

Folgende vier Integritätsbedingungen gelten für das Relationales Datenbankmodell:

- Es müssen fest definierte Wertebereiche existieren (z.B. String, Integer, Float, Date, etc.)
- Die Datentypen eines Wertebereiches sind stets gleich (sog. Domain Integrity)
- Pro Zelle (Spalte und Zeile) ist genau ein Wert eines Wertebereiches zugelassen
- Alle Werter einer Spalte sind vom gleichen Wertetyp (sog. Column Integrity)

2.3 Gegenüberstellung von Grundbegriffen

Relationale Datenbank	Relationen-Modell	Entity-Relationship-Modell (ERM)	Unified Modeling Language (UML)
Wertebereich (Domäne, Domain)	Wertebereich (Domäne, Domain)	Wertebereich (Domäne, Domain)	Wertebereich (Domäne, Domain)
Kopfzeile	Relationstyp/Relationsformat	Entitätstyp	Klasse
Spaltenüberschrift	Attribut	Attribut	Attribut
Inhalt	Relation	Entitätsmenge(-set)	Objektmenge, Instanz-

			menge, Klasse
Inhalt	Fremdschlüssel	Beziehung (Relations-hip)	Assoziation
Zeile	Tupel	Entität	Objekt, Instanz
Zelle	Attributwert	Attributwert	Attributwert

3. DDL (Data Definition Language)

Erstellen von Datenbanken, Tabellen (Relationen) und Indizes

3.1 Create Database

`Create Database uebung;`	erstellt eine Datenbank
`Create Database testdb;`	erstellt noch eine Datenbank
`Use testdb;`	erstellt eine Verbindung zur Datenbank testdb
`drop Database testdb;`	löschte die Datenbank testdb

3.2 Create Table

`create table t_person(`
`id integer not null,`
`name varchar(100) not null,`
`vorname varchar(100) not null`
`primary key(id));`
	Erstellt eine Tabelle
	Id ist der Attributname, integer der Datentyp und not null, erfordert eine Eingabe.
	Varchar ist ein String für 100 Zeichen
	Primary Key ist die id.

`alter table t_person` `add lebenslauf text;`	Zum ändern eine Tabelle. Attribut hinzufügen mit Datentyp text, welcher eine variable Länge an Strings speichern kann.
`alter table t_person` `add beschaeftigt_seit date;`	Neues Attribut mit date Datentyp.
`alter table t_person` `drop beschaeftigt_seit;`	Entfernen eines Objektes einer Tabelle
`drop table t_person;`	Löschen einer Tabelle.

3.3 PRIMARY KEY

`produkt_nr integer PRIMARY KEY,`	PrimKey bei Attributerstellung
`PRIMARY KEY (a, c),`	Primkey als Kombination

3.4 Check (Eingabe-Einschränkungen)

`Name Datentyp CHECK (Name [between x and y][>z]),`	Check bei Attributerstellung
`CONSTRAINT constraintbezeichner CHECK (name [between x and y][>z]),`	Check als Constraint

3.5 Unique (Kandidatenschlüssel):

`produkt_nr integer UNIQUE,`	Unique bei Attributerstellung
`UNIQUE (a, c),`	Unique als Kombination
`produkt_nr integer CONSTRAINT müssen_verschieden_sein UNIQUE,`	Unique als Constraint

3.6 Foreign Key (Fremdschlüssel):

`produkt_nr integer REFERENCES produkte (produkt_nr)`	Referenzierung bei der Attributerstellung
`FOREIGN KEY (b, c) REFERENCES andere_tabelle (c1, c2)`	Referenzierung als Befehl

`CONSTRAINT FOREIGN KEY (fk_Eltern) REFERENCES Personen (PersNr)`
`ON DELETE [CASCADE][SET NULL] [NO ACTION] [DEFAULT] [RESTRICT]`
`ON UPDATE [CASCADE] [SET NULL] [NO ACTION] [DEFAULT] [RESTRICT]`
	Referenzierung mit Auslöser (nicht nur für Foreign Keys verwendbar)
	Cascade: alle aktualisieren
	Set Null: alle Ref. Auf Null setzen
	No Action: Gar nichts machen
	Restrict: verbieten

4. DML (Data Manipulation Language)

Anlegen, Ändern und löschen von Datenätzen

`INSERT INTO tabelleAutor ( Nr, NachName, VorName, GebJahr ) VALUES ( 1, 'Böll', 'Heinrich', 1917 );`	Daten in bestehende Tabelle einfügen. Zahlenwerte ohne Hochkommas und Datentext-strings mit einfachen Hochkommas angeben. Soll der Wert eines Feldes nicht gesetzt werden, kann der entsprechende Feldname weggelassen werden oder alternativ als Datenelement NULL angegeben werden.
`UPDATE tabelleAutor SET Name = Otto, GebJahr = 1954, Beruf = NULL WHERE Nr = 10;`	Daten in Tabelle ändern. Auf NULL setzen bedeutet Feld löschen.
`DELETE FROM tabelleAutor WHERE Datum < (SYSDATE - 3650);`	Zeilen löschen (hier alle älter als 10 Jahre alten Einträge).

5. DQL (Data Query Language)

Abfragen von Daten

5.1 Select Syntax

`1. SELECT [DISTINCT] * | Datenfelder FROM Tabellenname`
`[WHERE Bedingung]`
`2.[GROUP BY Datenfelder [HAVING Bedingung] ]`
`3. [ORDER BY Datenfelder [ASC | DESC] ]`
`4. [LIMIT [Start, ] Anzahl];`
	1. Mit Select (auswählen) leitet man eine Abfrage ein. Mit Distinct werden keine gleiche Datensätze angegeben. Ansonsten ist die Syntax nicht anders.
	2. Datensätze werden nach einer optionalen Bedingung eines Datenfeldes gruppiert.
	3. Datensätze werden entweder aufsteigend (ASC) oder absteigend (DESC) nach einem Datenfeld sortiert.
	4. Limitiert die Anzahl der Datensätze.

5.2 Beispiele

`SELECT * FROM meineTabelle;`	Alle Daten einer Tabelle lesen.
`SELECT * FROM "meine Tabelle";`	Normalerweise unterscheidet SQL nicht zwischen Groß-/Kleinschreibung. Wird der Tabellenname in Anführungszeichen gesetzt, muss Groß-/Kleinschreibung exakt stimmen und der Tabellenname immer genau so geschrieben werden. Einige Datenbanken akzeptieren dann auch Leerzeichen im Tabellennamen (was eigentlich nicht erlaubt ist).
`SELECT feldName1, feldName2 FROM meineTabelle;`	Bestimmte Felder (Spalten) einer Tabelle lesen.
`SELECT feldName1, feldName2 FROM meineTabelle ORDER BY feldName2, feldName1 DESC;`	Spalte(n) zur Sortierung vorgeben, entweder per Feldnamen oder auch per Spaltennummern. Bei Spaltennummern beachten: Die erste Spalte ist 1 (und nicht 0). Ohne DESC aufsteigend, mit absteigend.
`SELECT * FROM meineTabelle WHERE feldName1 = 'xy' AND feldName2 < 100 AND feldName3 BETWEEN 1 AND 10;`	Die Zeilen der Tabelle lesen, deren Elemente die Bedingung erfüllen. '=' testet auf Gleichheit, '<>' auf Ungleichheit und '<', '<=', '>' und '>=' vergleichen. Textstrings werden z.B. für Oracle, MySQL und MS Access mit einfachen Hochkommas, aber z.B. für InterBase mit doppelten Hochkommas eingeschlossen.
`SELECT * FROM meineTabelle WHERE UPPER(feldName1) = UPPER('xy');`	Vergleich mit Ignorierung von Groß-/Kleinschreibung. Kommandos sind unterschiedlich je nach Datenbank. Großschreibung wird z.B. bei Oracle mit UPPER() und bei MS-Access mit UCASE() erreicht.
`SELECT * FROM meineTabelle WHERE feldName1 LIKE 'B%';`	Die Zeilen der Tabelle lesen, deren Element in der Spalte feldName1 mit einem großen B beginnt (oder mit '%abc%' den Teilstring 'abc' enthält). '_' ist Platzhalter für genau einen Zeichen, '__' für zwei Zeichen und '%' für eins oder mehrere Zeichen.
`SELECT * FROM meineTabelle WHERE feldName1 IN( 11, 13, 17 );`	Selektiere Zeilen, wo feldName1 in angegebener Menge enthalten ist.
`SELECT * FROM meineTabelle1 WHERE feldName1 IN( SELECT feldName2 FROM meineTabelle2 );`	Wie vorher, aber angegebene Menge ist Resultat von weiterer Abfrage (mit einspaltigem Ergebnis).
`SELECT meineTabelle1.feldName3, meineTabelle2.feldName4 FROM meineTabelle1, meineTabelle2 WHERE meineTabelle1.fremdSchlüsselFeld = meineTabelle2.primärSchlüsselFeld;`	Join zweier Tabellen. Leider ist die Syntax nicht bei allen Datenbanken gleich. Die gezeigte Schreibweise gilt z.B. für Oracle, MySQL und MS Access. Primärschlüsselspalte und Fremdschlüsselspalte können in der Datenbank entsprechend definiert werden.
`SELECT Autor.Name, Autor.Vorname, Buch.Titel, Gebiet.Bez, Verlag.Name_Kurz FROM Autor, Buch, Gebiet, Verlag WHERE Buch.Autor_Nr = Autor.Nr AND Buch.Gebiet_Abk = Gebiet.Abk AND Buch.Verlag_Nr = Verlag.Nr;`	Join vierer Tabellen. Bei Verknüpfung von n Tabellen sind n-1 Join-Kriterien erforderlich.
`SELECT * FROM Kunde K JOIN Bestellung B ON K.kdkey=B.kdkey;`	Join zweier Tabellen in einer für die Datenbank InterBase verständlichen Syntax.
`SELECT feldName1 "Nachname", feldName2 "Vorname" FROM meineTabelle;`	Aliasnamen: Für Feldnamen andere Bezeichnungen vorgeben.
`SELECT Nachname || ', ' || Vorname "Name" FROM meineTabelle;`	Konkatenation mit ||: Zwei Spalten werden zu einer Ausgabespalte (mit dem neuen Namen

	"Name") verbunden.
`SELECT SUBSTR( Name, 1, 1 )` `FROM meineTabelle;`	Teilstring extrahieren. Parameter: String, Startposition, Länge.
`SELECT DISTINCT feldName1 FROM meineTabelle;`	DISTINCT bedeutet Zusammenfassung gleicher Elemente zu einer Zeile.
`SELECT COUNT(*) "Anzahl" FROM meineTabelle;`	Eingebaute Aggregatfunktionen: COUNT() (Anzahl), MIN(), MAX(), AVG() (Durchschnitt), SUM().
`SELECT ZahlungsEmpfaenger, SUM(Betrag) FROM Rechnungen GROUP BY ZahlungsEmpfaenger;`	GROUP BY reduziert die returnierten Reihen pro Group-Wert auf eine Reihe. GROUP BY normalerweise zusammen mit Aggregatfunktionen (z.B. SUM, AVG ...).
`SELECT TO_CHAR( Datum, 'YYYY' )` `FROM meineTabelle;`	Datentypkonvertierung: TO_CHAR() (String), TO_NUMBER() (Zahl), TO_DATE() (Datum).
`SELECT * FROM meineTabelle where date = TO_DATE( '2002-01-23_14:51', 'yyyy-MM-dd_HH24:mi' );`	Datumsformatkonvertierung mit TO_DATE() (z.B. bei Oracle).
`SELECT SYSDATE FROM DUAL;`	SYSDATE ist das aktuelle System-Datum. DUAL ist ein Dummy-Name als Platzhalter für eine Tabelle, wo eigentlich keine Tabelle benötigt wird. SYSDATE und DUAL werden nicht von allen Datenbanken unterstützt (aber z.B. von Oracle).
`SELECT * FROM meineTabelle WHERE Datum >= (SYSDATE - 28);`	Die Zeilen der Tabelle lesen, deren Eintrag im Datumsfeld nicht älter als vier Wochen ist. Datums-Kommando ist unterschiedlich je nach Datenbank, z.B. SYSDATE bei Oracle und NOW() bei MS-Access.
`SELECT 1 FROM DUAL WHERE EXISTS ( SELECT 1 FROM MeineTabelle WHERE ... );`	EXISTS prüft Existenz.
`SELECT * FROM meineTabelle WHERE feldName1 IS NULL AND feldName2 IS NOT NULL;`	SQL returniert NULL, wenn ein Feld leer ist. Es gibt normalerweise keine Leerstrings. NULL kann nicht mit Vergleichsoperatoren geprüft werden, sondern mit IS NULL bzw. IS NOT NULL.
`SELECT Name, NVL( TO_CHAR(GebJahr), '?' )` `FROM meineTabelle;`	NVL() ersetzt NULL Values durch etwas anderes.
`SELECT title, text FROM books` `WHERE CONTAINS( text, '!door' ) > 0;`	Ausrufezeichenoperator für phonetische Suche mit 'soundex' (nicht in allen Datenbanken implementiert, aber z.B. in Oracle).

5.3 Operatoren Übersicht

Operator/Element	Beschreibung
-	unäres Minus
^	Potenzierung
* / %	Multiplikation, Division, Modulus
+ -	Addition, Subtraktion
IS	IS TRUE, IS FALSE, IS UNKNOWN, IS NULL
ISNULL	Test für NULL
NOTNULL	Test für nicht NULL
(alle anderen)	alle anderen eingebauten und benutzerdefinierten Operatoren
IN	Mengenmitgliedschaft
BETWEEN	Vergleich
OVERLAPS	Zeitinterval überlappt
LIKE ILIKE SIMILAR	Mustervergleich von Zeichenketten
< >	kleiner als, größer als
=	Gleichheit, Wertzuweisung
NOT	logische Negierung
AND	logische Konjunktion
OR	logische Disjunktion

5.4 Platzhalter

Platzhalter	Erklärung	Beispiel	Mögliches Ergebnis
%	Platzhalter steht für kein, ein oder mehrere beliebige Zeichen	name LIKE „F&“ name LIKE „%son“ name LIKE „&ill“	Funke, Franz Benson, Jenson Miller, Filler, Ofillson
_ (Unterlinde)	Platzhalter steht für exakt ein beliebiges Zeichen	name LIKE „M_ller“ name LIKE „____“	Müller, Möller Adas, Funk, Tier

6. DCL (SQL Data Control Language)

Anlegen von Benutzern und Vergabe von Zugriffsrechten

`GRANT SELECT, DELETE, UPDATE, REFERENCES(Nr) ON meineTabelle TO Mueller;`	Rechte vergeben.
`REVOKE DELETE ON meineTabelle FROM Mueller;`	Rechte entziehen.

7. Glossar

Begriff	Beschreibung
Datenbankmanagementsystem	Das Datenbankmanagementsystem (DBMS) ist die eingesetzte Software, die für das Datenbanksystem installiert und konfiguriert wird. Das DBMS legt das Datenbankmodell fest, hat einen Großteil der unten angeführten Anforderungen zu sichern und entscheidet maßgeblich über Funktionalität und Geschwindigkeit des Systems. Datenbankmanagementsysteme selbst sind hochkomplexe Softwaresysteme.
Datenbank	In der Theorie versteht man unter Datenbank (engl. database) einen logisch zusammengehörigen Datenbestand. Dieser Datenbestand wird von einem laufenden DBMS verwaltet und für Anwendungssysteme und Benutzer unsichtbar auf nichtflüchtigen Speichermedien abgelegt. Um einen effizienten Zugriff auf die Datenbank zu gewährleisten, verwaltet das DBMS in der Regel eine Speicherhierarchie, die insbesondere auch einen schnellen Zwischenspeicher (Pufferpool) umfasst. Zur Wahrung der Konsistenz des Datenbestandes müssen sich alle Anwendungssysteme an das DBMS wenden, um die Datenbank nutzen zu können. Allein administrativen Tätigkeiten, wie zum Beispiel der Datensicherung, ist der direkte Zugriff auf den Speicher erlaubt.
Attribut	Spalte einer Relation, Eigenschaft einer Entität oder einer Beziehung im ER-Modell
Datenbanksicht	Unterschiedliche Betrachtungsweise einer Datenbank
ER-Modell	Entity-Relationship-Modell: Hilfsmittel zur grafischen Darstellung eines Datenbankentwurfes
Entity	Entität, Datensatz, Tupel, Zeile einer Tabelle
Fremdschlüssel (foreign Key)	Ein Fremdschlüssel ist ein Abttribut einer Relation, das einer anderen Relation als Primärschlüssel dient. Mithilfe von Fremschlüsseln lassen sich Beziehungen zwischen Relationen herstellen
Index	Ein Attribut oder eine Kombination von Attributen einer Relation wird als Index gespeichert, um z.B. Sortierfunktion schneller durchzuführen können.
Normalisierung	Methode zur Erreichung einer redundanzfreien Datenspeicherung in den Relationen eines Datenbankschemas
Redundanz	Mehrfaches Speichern einer Information am gleichen Ort
Relationship	Beziehung zwischen Entities
Relationale Datenbank	Daten werden in Relationen organisiert, die mit Daten anderer Relationen in Beziehung stehen könne, Eine Relation kann sowohl Objekte (Entitäten)als auch Beziehungen beinhalten
Primärschlüssel, Primary Key	Ein Attribut oder eine Kombination von Attributen, welche einen Tupel eindeutig kenzeichnen
Tabelle	Zweidimensionale Konstruktion aus Spalten und Zeilen zum Speichern von Datensätzen (Tupel).
Sekundärschlüssel	Zusätzlicher Schlüssel, um Redundanzen in den Datensätzen der Tabelle zu vermeiden
Data Dictionary	Speichert Informationen über die Datenbank, die physisch zusammenhängen auf einem externen permanenten Speichermedium abgelegt sind.
Datenmodell	Hilfsmittel zur Abstraktion der Daten aus der realen Welt. Es wird eine Struktur aus den relevanten Daten, deren Beziehungen und Bedingungen erzeugt.
Integrität	Liegt vor, wenn Daten in sich richtig (stimmig), widerspruchsfrei und vollständig sind (logische Integrität). Referenzielle Integrität => siehe Konsistenz
Integrität	Integritätsbedingungen sind Bestimmungen, die eingehalten werden müssen, um die Korrektheit und die logische Richtigkeit der Daten zu sichern.
Konsistenz/ Inkonsistenz	Konsistenz (referenzielle Integrität) ist die Übereinstimmung von mehrfach gespeicherten Daten. Werden bei Änderungen nicht alle mehrfach gespeicherten Daten geändert, ist der Datenbestand inkonsistent, d.h., es existieren unterschiedliche Versionsstände der gleichen Daten.