

Datenbanken

In diesem Teil wird in die Thematik der Datenbanksysteme eingeführt. Neben den theoretischen Grundlagen, wird als reales Datenbanksystem MySQL zur Verdeutlichung verwendet. Den Abschluss bildet eine Einführung in die Datenbankprogrammierung.

Inhalt

1	Überblick.....	5
2	Konzeptioneller Entwurf.....	7
2.1	Entity Relationship Modell.....	8
2.1.1	Entity.....	8
2.1.2	Attribute.....	9
2.1.3	Relationship	12
3	Logischer Entwurf.....	19
3.1	Relationenmodell.....	19
4	MySQL und SQL.....	27

4.1	Zugriff auf eine Datenbank.....	27
4.2	Anlegen einer Datenbank	28
4.3	Anlegen einer Tabelle	29
4.4	Select, insert, update, delete.....	30
4.5	User anlegen.....	31
4.6	Datenbank-Operationen.....	34
4.6.1	Anlegen der Tabellen	34
4.6.2	Einfügen von Werten.....	36
4.6.3	Selektion.....	37
4.6.4	Projektion.....	37
4.6.5	Verbund.....	38
5	Datenmodell für eine Anwendung.....	41
5.1	SQL Elemente zum Ausdruck von Integrität.....	44
6	Normalisierung von Datenbanken.....	48
6.1	Normalisierung (informal).....	53

6.2	Erste Normalform.....	53
6.3	Zweite Normalform	55
6.4	Normalisierung (formaler).....	58
7	Abbildung von ERMen auf Relationenmodelle.....	62
8	ACID Konzept.....	70
8.1	Transaktion in MySQL.....	70
9	Datenbankgestützte Anwendungsentwicklung.....	73
9.1	Stored Procedures.....	74
9.1.1	Anlegen und Löschen von Prozeduren und Funktionen.....	74
9.1.2	Lokale Variable.....	77
9.1.3	Kontrollstrukturen	78
9.1.4	Ausnahmebehandlung.....	86
9.1.5	Zusammenfassendes Beispiel (Buchen).....	89
9.2	Cursor.....	93
9.3	Datenbank-Trigger.....	95

9.4Dynamisches SQL	96
9.5Historisieren.....	96

1 Überblick

Bei der Entwicklung von Datenbank-gestützter Anwendungen ist das Datenbankmodell besonders wichtig:

Das **Datenmodell „überlebt“** die zugreifende **Anwendung**, da mehrere Anwendungen die Datenbank verwenden (im Bankenbereich sind Datenmodelle mit einer „Lebenszeit“ von 20 Jahren keine Seltenheit, die Anwendungen selbst, leben oft kürzer).

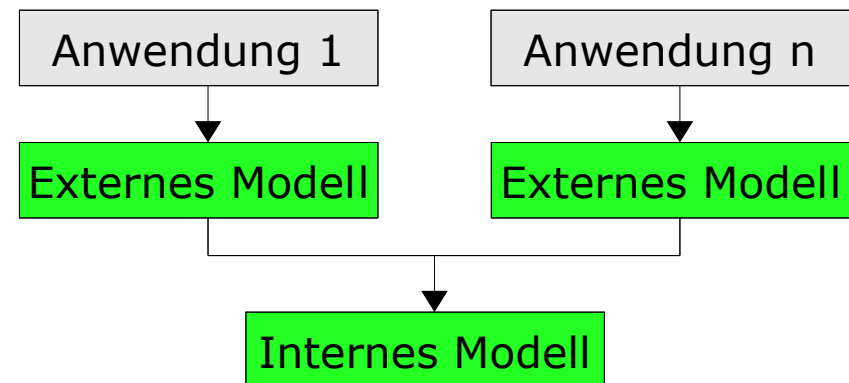
Die Wartbarkeit einer Datenbankanwendung wird maßgeblich von der Güte des Datenmodells beeinflusst.

Um ein Datenmodell herleiten zu können, sind unterschiedliche Sichtweisen auf den Datenbestand zu unterscheiden:

externe Sicht enthält die Sicht des Anwendungsbenutzers

Daten und Operationen, die der Benutzer sieht

interne Sicht sieht die Daten mit ihrer physischen Ablage in der Datenbank

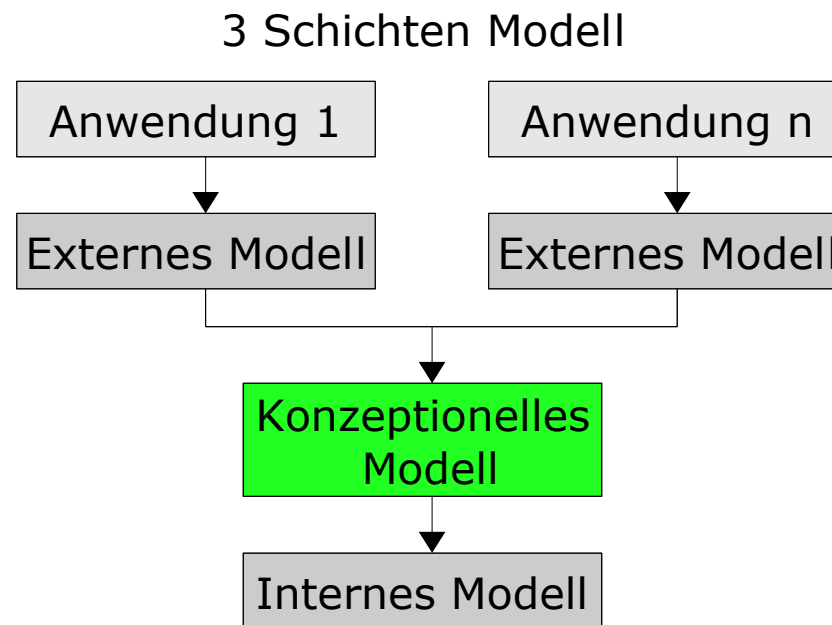


Werden Anwendungen so modelliert, müssen alle **Anwendungen angepasst** werden, wenn Änderungen an der Datenbank vorgenommen werden. Solche Änderungen werden z.B. wegen Optimierungen und **Performance-Verbesserungen** erforderlich.

Deshalb wird eine **konzeptionelle Sicht** eingeführt und so **logische Datenunabhängigkeit** erreicht:

konzeptionelle Sicht

Von **allen Benutzern** gemeinsam akzeptierte, **einheitliche Darstellung** der **realen Welt** bzw. des Ausschnittes der realen Welt, der durch die Datenbank dargestellt werden soll.



2 Konzeptioneller Entwurf

Der konzeptionelle Entwurf ist der **Vorgang**, bei dem das **konzeptionelle Modell** aus der **Realität** abgeleitet wird. Er mündet im **logischen (konzeptionellen) Datenmodell**.

Er verläuft in der Regel in den folgenden Schritten:

1. **Benennen**: Benennung eines jeden **Objektes** der realen Welt und jeder **Beziehungen** zwischen diesen Objekten
2. **Auswahl**: **Reduzierung** der betrachteten Objekte auf eine **handhabbare Anzahl**, d.h. Wahl des relevanten Ausschnitts der Realität
3. **Klassifikation**: Zuordnung der Objekte und Beziehungen zu Klassen

Die Ergebnisse des Modellierungsvorganges sind **nicht eindeutig!**

2.1 Entity Relationship Modell

Als Methode für die konzeptionelle Phase hat sich das ER-Modell (Chen, 1976) bewährt.

Es ist ein **Beschreibungsmodell** für **Datenbanken** und wird in erster Linie in frühen Entwurfsphasen eingesetzt (wenn möglich, in der Recherchephase Ihres Projektes).

Das ER-Modell hat eine **graphische Darstellungsweise** und eine klare Definition.

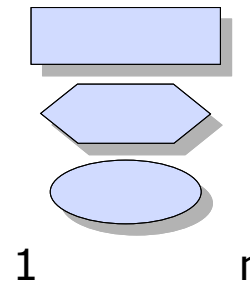
Standardelemente des ERMs und deren Darstellung sind:

Entity (Objektyp): Rechteck

Relationship (Beziehungstyp): Raute oder 6 Eck¹

Attribute: Ovale

Beziehungsart



2.1.1.Entity

Entitäten sind **Objekt** der realen oder **Vorstellungswelt**, über die Informationen zu speichern sind.

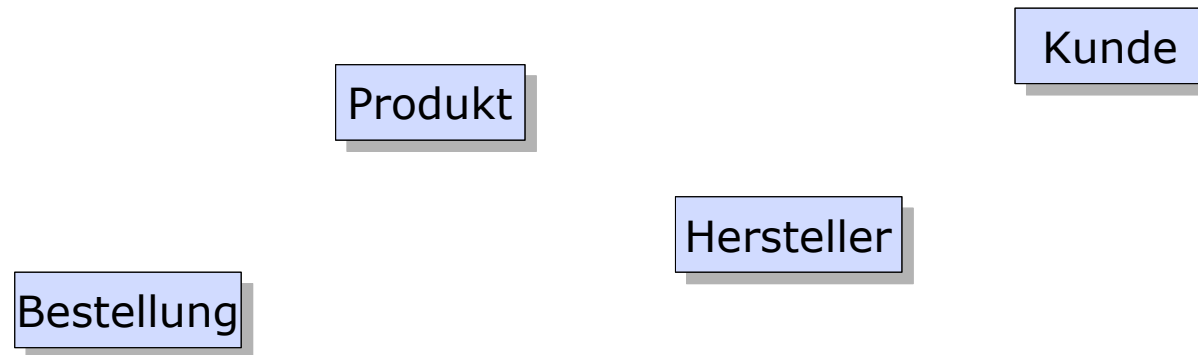
Eine Entität

ist eine in einer DB zu repräsentierende **Informationseinheiten**,

¹ normalerweise nur Raute, 6 Eck spart Platz

ist nicht direkt darstellbar, sondern nur über ihre **Attribute beobachtbar** und kann einen Typ haben.

Beispiel (Shop):



2.1.2. Attribute

Attribute sind Eigenschaft von

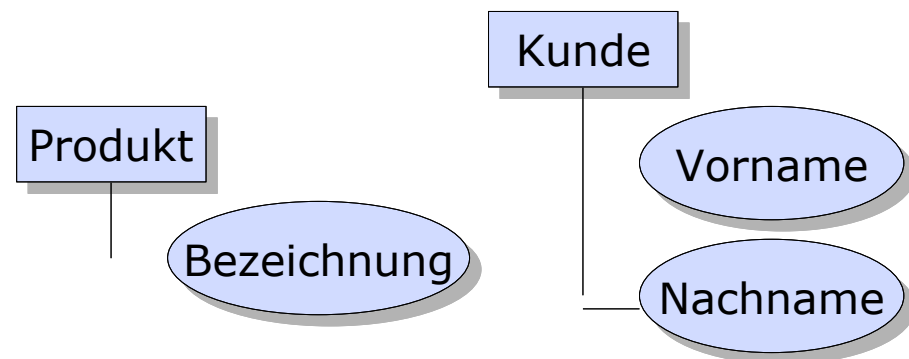
Entities

Relationships

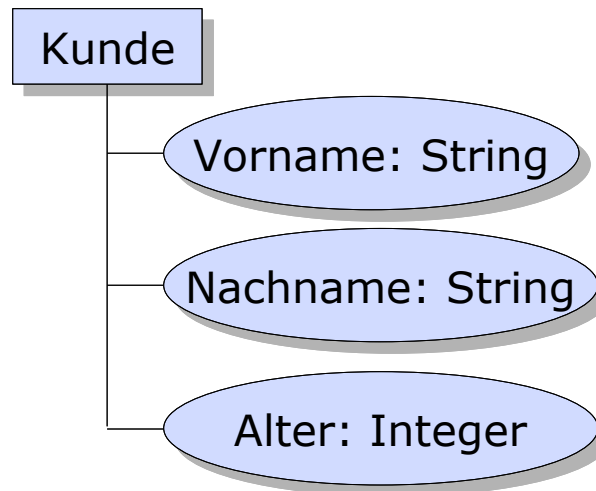
Beispiel:

Name des Kunden

Bezeichnung des Produktes



Attribute modellieren Eigenschaften, die oft einen **Typ** haben, der mit den Typen von Programmiersprachen oder Datenbankfeldern assoziiert ist.



Die Attribute werden nur für **Entity-Typen** modelliert, da Entities gleichen Typs alle die gleichen Attribute besitzen müssen. Der Definitionsbereich D definiert den Wertebereich, z.B. für Entitäten:

$$E(A_1:D_1, A_2:D_2, \dots, A_n:D_n)$$

Kunde(Vorname:String, Nachname:String, Alter:Integer)

Schlüssel sind

Attribute, die eine **eindeutige Identifizierung** von Entities ermöglichen

Stellen Kandidaten zur Wahl des Primärschlüssels dar

Die Festlegung auf einen identifizierenden Schlüssel stellt eine Modellierungsentscheidung für eine konkrete DB-Anwendung dar

Schreibweise:

Schlüsselattribute werden unterstrichen (auch in der grafischen Darstellung)

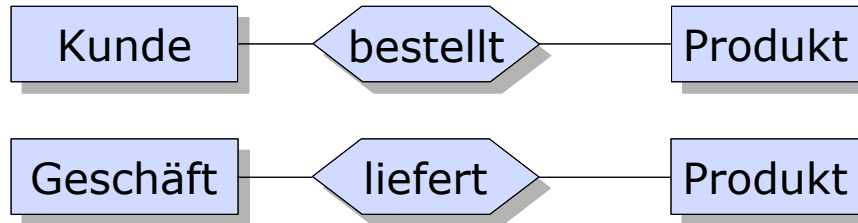
$E (\underline{A}_1:D_1, \underline{A}_2:D_2, \dots, A_n:D_n)$

Kunde (Vorname:String, Nachname:String, Alter:Integer)

2.1.3. Relationship

Relationships definieren **Beziehung** zwischen Entities.

Beispiel:



Beziehungen können unterschiedliche **Beziehungsarten** haben:

1:1

Je einem Objekt des ersten Objekttyps wird genau ein Objekt des zweiten Objekttyps zugeordnet und umgekehrt

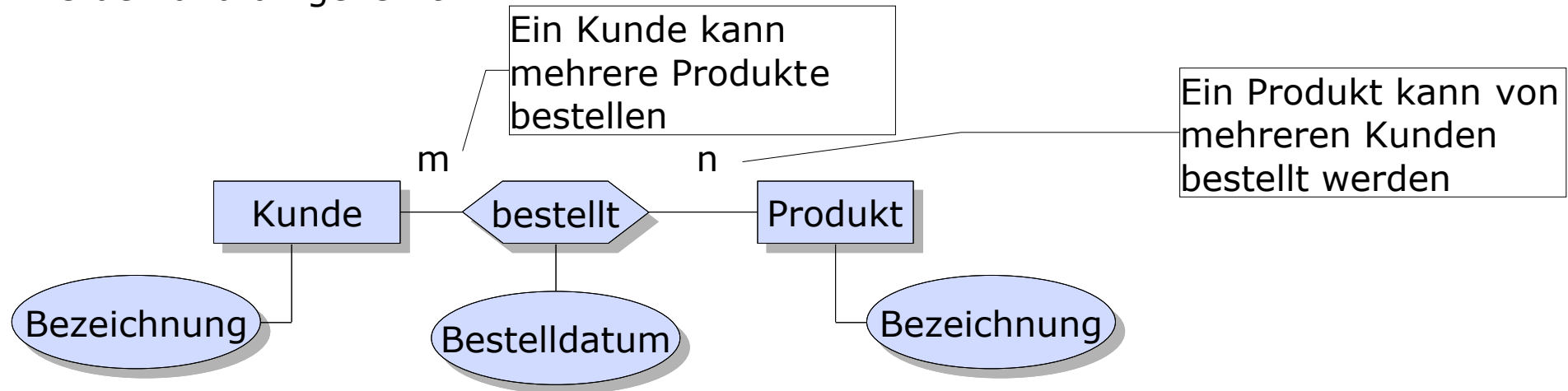
1:n

Einem Objekt des ersten Objekttyps können beliebig viele (auch null) Objekte des zweiten Objekttyps zugeordnet werden

Einem Objekt des zweiten Objekttyps wird genau ein Objekt des ersten Objekttyps zugeordnet

m:n

Einem Objekt des ersten Objekt können n Objekte des zweiten Objekttyps zugeordnet werden und umgekehrt



Beziehungen haben immer eine **Kardinalität** (Anzahl beteiligter Entitäten):

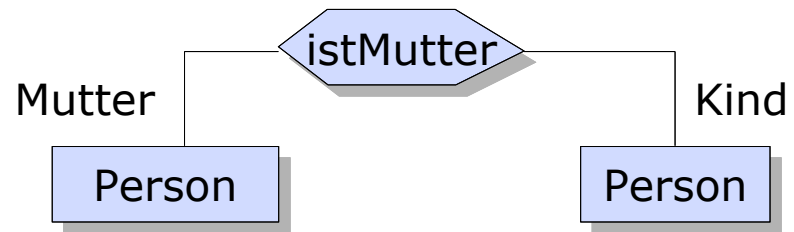
2 stellige Beziehung:

bestellt(Kunde, Produkt)

3 stellige Beziehung:

liefern(Geschäft, Kunde, Produkt)

Rollennamen können in Beziehungen vergeben werden, wenn eine Entität mehrfach vorkommt:



```
istMutter(Person:Mutter, Person:Kind)
```

Hörsaalübung:

1. Formulieren Sie grafisch mit Beziehungs- und Entitätstypen:

Bücher haben Autoren und werden verlegt von einem Verleger

Studenten besuchen Kurse, Kurse haben einen Dozenten

Verwenden Sie dazu Beziehungstypen mit namentlichen Beziehungen und typisierte Attribute

2. Überlegen Sie sich einen sinnvollen Anwendungsfall für eine 3-stellige Beziehung und formulieren Sie diesen grafisch.

Verschiedene Beziehungsmuster treten bei der Entity-Relationship-Modellierung häufiger auf:

Funktionale Beziehungen

Ist-Beziehung

Funktionale Beziehungen sind eindeutige Zuordnung eines Entities zu einem anderem Entity:

Zweistellige Beziehungen

Jedem Entity eines Entity-Typs E_1 wird maximal ein Entity eines Entity-Typs E_2 zugeordnet

Schreibweise: $R: E_1 \rightarrow E_2$

Darstellung: Pfeil

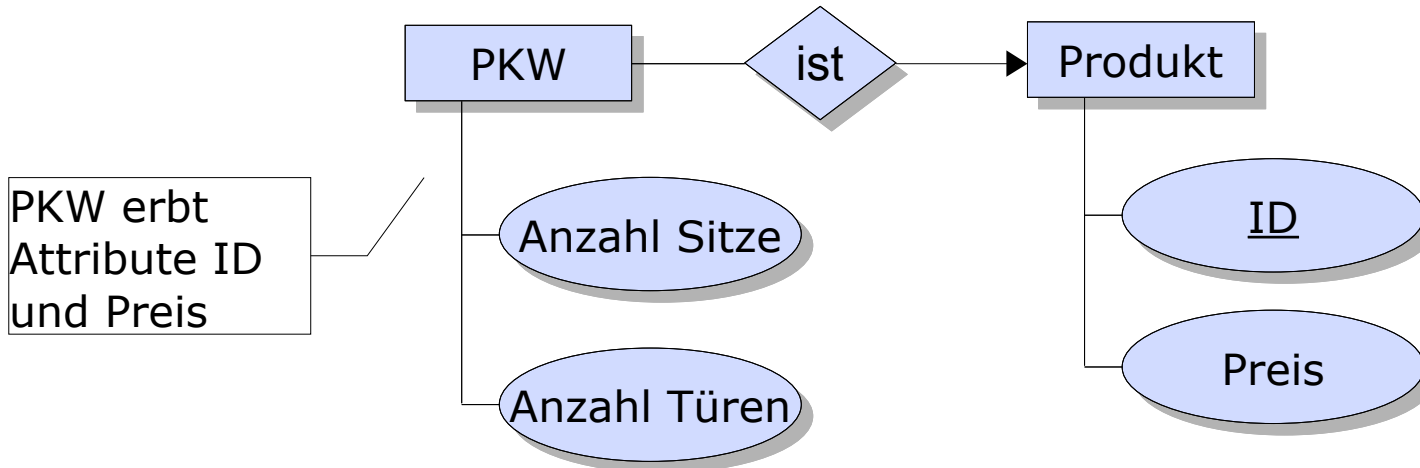


Ist Beziehungen drücken Spezialisierungen/Generalisierungen aus:

Spezialisierung erbt Attribute der Generalisierung

Alle Attribute der Generalisierung sind automatisch auch in der Spezialisierung vorhanden

Darstellung:



oder kurz auch ohne Raute



Hörsaalübung

Skizzieren Sie ein ER-Modell für einen kleinen Web-Shop.

3 Logischer Entwurf

Der **erste Teilschritt** (konzeptionelle Entwurf) des DB-Entwurfs liefert eine **abstrakte Beschreibung** der DB im **logischen Modell** (hier: **ER-Modell**).

Das **logische Modell** muss nun zur Realisierung auf das **Zielmodell** der jeweiligen DB **abgebildet** werden:

Hierarchische Modelle

Relationenmodelle

Objektorientierte Modelle

...

Relationale Datenbanken basieren dabei auf dem **Relationenmodell**.

Der folgende Schritt umfasst somit die Abbildung des ER-Modells in das Relationenmodell

3.1 Relationenmodell

Das Relationenmodell (Codd 1970) hat folgende Eigenschaften:

Es basiert auf einer bekannten Grundlage: Algebra

Daten werden in Tabellenform dargestellt (nicht in Form eines Graphen)

Es existiert keine Unterscheidung zwischen Entities und Relationships

Viele kommerzielle Datenbanksysteme basieren auf dem Relationenmodell. (DB2, Oracle, Ingres, Informix, Access, MySQL)

Im Relationenmodell sind folgende **Begriffe** zu unterscheiden:

Attribut:

Merkmalsausprägung eines Objekttyps (z.B. id, Alter, ...)

Bezeichnet eine Spalte einer Tabelle

Wertebereich:

Spezifiziert alle möglichen Merkmalsausprägungen eines bestimmten Attributes

Tupel:

Merkmalsausprägungen eines Objekts

Zeile der Tabelle

Schlüssel:

Teilmenge der Attribute, die einen Tabelleneintrag eindeutig identifiziert

Relation Person

<i>id</i>	<i>Name</i>	<i>Alter</i>
18	pitt	33
19	jenni	29

Relation (Tabelle):

Menge von Tupel

Reihenfolge der Tupel ist unwesentlich

Keine zwei gleiche Tupel

Relationenname:

Bezeichnung der Relation

Relationenschema (Tabellengerüst):

Relations- und Attributnamen

Erste Zeile der Tabelle

Relationales Datenbankschema:

Kombination verschiedener Relationenschemata

Auf den Relationen sind **Operationen** erlaubt:

Selektion

Projektion

Verbund

Relationenschema Bank

Relationen- name	Attribute		
Person	id	Name	Alter
Kunde	nr	Art	Addr.
Konto	nr	Saldo	Datum

Mengenoperationen

Selektion:

Wählt alle jene Tupel aus einer Relation aus, die eine bestimmte Bedingung erfüllen

Eigenschaften:

Anwendung auf eine Relation

Relationenschema (Anzahl der Attribute) bleibt gleich

Zahl der Tupel kann reduziert werden

Relation A

A_1	...	A_n

Selektion auf A

A_1	...	A_n

Projektion:

Ergibt als Ergebnis eine Relation mit denjenigen Attributwerten, auf deren Attribute die Projektion angewandt wurde

Eigenschaften:

Anwendung auf eine Relation

Relationenschema (Anzahl der Attribute) kann sich verringern

Zahl der Tupel kann reduziert werden, falls es in der verbleibenden Relation identische Tupel gibt.

Relation A

A_1	A_k	...	A_o	A_n

Projektion auf A

A_k	...	A_o

Verbund:

Zwei Relationen mit einem **gemeinsamen Attribut** werden zu einer neuen Relation verbunden

Das neue Relationenschema enthält alle Attribute der beiden Ursprungs-Relationen

Die Ergebnisrelation enthält all jene Tupel, für die das **Verbundattribut identisch** ist.

Eigenschaften:

Anwendung auf mehrere Relationen

Anzahl der Attribute kann sich vergrößern

Zahl der Tupel kann sowohl zu- als auch abnehmen

Relation A

X	A ₁	...	A _k

Relation B

X	B ₁	...	B _n

Verbund von A und B

X	A ₁	...	A _k	B ₁	...	B _n

Zusammenhang zwischen Verbund-Operation im Relationenmodell und Beziehungstyp im Entity-Relationship-Modell: Im relationalen Modell ist ein Verbund zwischen zwei Relationen in

der Regel nur dann möglich, wenn im ERM zwischen diesen beiden Objekttypen ein Beziehungstyp existiert.

Mengenoperationen:

Mengenoperationen werden auf zwei (oder mehrere) Relationen mit gleicher Attributmenge angewandt

Das Ergebnis ist wieder eine Relation mit gleicher Attributmenge, lediglich Anzahl der Tupel verändert sich:

Vereinigung

Die Ergebnisrelation enthält alle Tupel beider Relationen

Durchschnitt

Die Ergebnisrelation enthält nur jene Tupel, die sowohl in der ersten als auch in der zweiten Relation vorkommen

Differenz

Die Differenz der Relationen R_1 und R_2 ergibt als Ergebnis eine Relation, die alle diejenigen Tupel der Relation R_1 enthält, die nicht in der Relation R_2 vorkommen

Hörsaalübung

Entwickeln Sie ein relationales Datenbankschema für einen kleinen Web-Shop.

4 MySQL und SQL

Damit der physische Entwurf stattfinden kann, muss ein DBMS eine Sprache zur Verfügung stellen, um die Relationen der Konzeptphase in Datenbanktabellen abbilden zu können. Dazu wurde die **Structured Query Language** (SQL) definiert.

Mittels SQL sind u.a.

- neben Tabellenerzeugung und -löschung auch

- Abfragen,

- Einfügen und

- Ändern von Inhalten

möglich.

Wir werden nur einen Ausschnitt aus SQL behandeln und am Beispiel der freien DBMS MySQL diskutieren.

4.1 Zugriff auf eine Datenbank

Um auf eine MySQL-Datenbank zugreifen zu können, ist ein MySQL-Client erforderlich. Es existieren grafische Clients, hier wird ein einfacher textbasierter Client verwendet.

4.2 Anlegen einer Datenbank

```
$ mysql -uas -p mysql
```

```
Enter password:
```

```
Reading table information for completion of table and column names
```

```
You can turn off this feature to get a quicker startup with -A
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

```
Your MySQL connection id is 2 to server version: 3.23.58
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

```
mysql> create database msd;
```

```
Query OK, 1 row affected (0.05 sec)
```

```
mysql> connect msd;
```

```
Connection id:      3
```

```
Current database:  msd
```

```
mysql> exit
```

```
$
```

4.3 Anlegen einer Tabelle

```
$ mysql -u as -p msd
```

```
mysql> create table event (id integer, edate date, description varchar(200));  
Query OK, 0 rows affected (0.28 sec)
```

```
mysql> mysql> show tables;
```

```
+-----+  
| Tables_in_tutorial |  
+-----+  
| event              |  
+-----+  
1 row in set (0.00 sec)
```

```
mysql> desc event;
```

```
+-----+-----+-----+-----+-----+-----+  
| Field          | Type          | Null | Key | Default | Extra |  
+-----+-----+-----+-----+-----+-----+  
| id             | int(11)       | YES  |     | NULL    |       |  
| edate          | date          | YES  |     | NULL    |       |  
| description    | varchar(200)  | YES  |     | NULL    |       |  
+-----+-----+-----+-----+-----+-----+  
3 rows in set (0.00 sec)
```

```
mysql>
```

4.4 Select, insert, update, delete

```
$ mysql> insert into event values (1, sysdate(), 'Rolling Stones in concert');
```

```
Query OK, 1 row affected (0.10 sec)
```

```
mysql> select * from event;
```

```
+-----+-----+-----+
| id    | edate      | description          |
+-----+-----+-----+
|      1 | 2004-02-02 | Rolling Stones in concert |
+-----+-----+-----+
```

```
1 row in set (0.04 sec)
```

```
mysql>
```

```
mysql> update event set edate = edate +1 where id = 1;
```

```
Query OK, 1 row affected (0.00 sec)
```

```
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> select * from event;
```

```
+-----+-----+-----+
| id    | edate      | description          |
+-----+-----+-----+
|     1 | 2004-02-03 | Rolling Stones in concert |
+-----+-----+-----+
1 row in set (0.00 sec)
```

```
mysql>
```

4.5 User anlegen

```
$ mysql> grant all privileges on msd.* to jenni@localhost
        identified by 'lopez' with grant option;
```

```
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> flush privileges;
```

```
$
```

```
$ mysql -u jenni -p msd
```

```
mysql> show tables;
```

```
+-----+
| Tables_in_msd |
+-----+
```

MySQL und SQL

```
+-----+
| event |
+-----+
1 row in set (0.00 sec)
mysql>
```

Achtung:

Beim Wechsel von MySQL 4.* auf 5.* ist folgende SQL-Anweisung erforderlich, da sich die Authentifizierungsalgorithmen geändert haben:

```
mysql> set password for jenni@localhost = old_password('lopez');
```

Hörsaalübung

1. Legen Sie eine mySql-Datenbank mit Namen „bank“ an.
2. Geben Sie dem Benutzer mit Namen „admin“ Rechte zu Zugriff auf die angelegte Datenbank.

4.6 Datenbank-Operationen

Die vorgestellten Operationen auf den Relationen werden nun am Beispiel von den folgenden Relationen in SQL demonstriert:

Kunde	KuNr	Name	Adresse	
Konto	KoNr	KuNr	Valuta	Saldo

4.6.1. Anlegen der Tabellen

```
mysql> create table kunde (kunar integer,  
                           name varchar(20),  
                           adresse varchar(200),  
                           primary key (kunar)  
                           ) type=innodb;  
  
mysql> create table konto (konr integer,  
                           kunr integer,  
                           valuta date,  
                           saldo float ,  
                           primary key (konr),  
                           foreign key (kunar) references kunde (kunar)  
                           );
```

MySQL und SQL

```
mysql> desc kunde;
```

Field	Type	Null	Key	Default	Extra
kunr	int(11)		PRI	0	
name	varchar(20)	YES		NULL	
adresse	varchar(200)	YES		NULL	

3 rows in set (0.00 sec)

```
mysql> desc konto;
```

Field	Type	Null	Key	Default	Extra
konr	int(11)		PRI	0	
kunr	int(11)	YES		NULL	
valuta	date	YES		NULL	
saldo	float	YES		NULL	

4 rows in set (0.00 sec)

4.6.2. Einfügen von Werten

```
mysql> insert into kunde values(1,'as',NULL);
mysql> insert into konto values(1,1,sysdate(),100);
mysql> select * from kunde;
+-----+-----+-----+
| kunr | name | adresse |
+-----+-----+-----+
|    1 | as   | NULL    |
|    2 | jl   | NULL    |
|    3 | bp   | NULL    |
+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> select * from konto;
+-----+-----+-----+-----+
| konr | kunr | valuta      | saldo |
+-----+-----+-----+-----+
|    1 |    1 | 2004-07-12 |   100 |
|    2 |    2 | 2004-07-12 |    10 |
|    3 |    3 | 2004-07-12 |  1000 |
|    4 |    1 | 2004-07-12 |     0 |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

4.6.3. Selektion

```
Selektiere alle Kunden, die mehr als 50 EUR auf dem Konto haben:mysql> select *
from konto where saldo > 50;
```

```
+-----+-----+-----+-----+
| konr | kunr | valuta      | saldo |
+-----+-----+-----+-----+
|     1 |     1 | 2004-07-12 |    100 |
|     3 |     3 | 2004-07-12 |   1000 |
+-----+-----+-----+-----+
2 rows in set (0.02 sec)
```

4.6.4. Projektion

```
mysql> select konr, saldo from konto;
```

```
+-----+-----+
| konr | saldo |
+-----+-----+
|     1 |    100 |
|     2 |     10 |
|     3 |   1000 |
|     4 |     0  |
+-----+-----+
4 rows in set (0.00 sec)
```

4.6.5. Verbund

```
mysql> select a.name, k.konr, k.saldo
        from kunde a, konto k
        where a.kunr = k.kunr;
```

```
+-----+-----+-----+
| name | konr | saldo |
+-----+-----+-----+
| as   | 1    | 100   |
| jl   | 2    | 10    |
| bp   | 3    | 1000  |
| as   | 4    | 0     |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

Hörsaalübung:

1. Legen Sie die Beispieltabellen Konto und Kunde an und vollziehen Sie die gezeigten Beispiele nach.
2. Was passiert in der o.a. Abfrage ohne die where-Klausel?

Und alle Operationen zusammen:

```
mysql> select a.name, k.konr, k.saldo  
       from kunde a, konto k  
       where a.kunr = k.kunr  
       and k.saldo > 0;
```

```
+-----+-----+-----+  
| name  | konr  | saldo  |  
+-----+-----+-----+  
| as    | 1     | 100    |  
| jl    | 2     | 10     |  
| bp    | 3     | 1000   |  
+-----+-----+-----+  
3 rows in set (0.00 sec)
```

Hörsaalübung

Welche SQL-Anweisungen sind erforderlich, um eine mySQL Datenbank für einen kleinen Web-Shop anzulegen?

5 Datenmodell für eine Anwendung

Der logische Entwurf mündet in einem Datenmodell, das häufig Grundlage für die Angebotserstellung für Softwarehäuser ist.

Ein **Datenmodell** besteht im wesentlichen aus drei Komponenten:

1. Menge der logischen Datenstrukturen (**Strukturkomponente**)
2. Menge von Operatoren (**Manipulationskomponente**)
3. Menge von statischen und dynamischen Integritätsbedingungen (**Integritätskomponente**)

Die Strukturkomponente umfasst die Bestandteile des relationalen Modells:

Attribute

Schlüssel

Primärschlüssel

Wertebereiche

Relationen

Die **Manipulationskomponente** beschreibt alle:

Selektionen

Verbunde

Projektionen

Mengenoperationen

Die **Integritätskomponente** beschreibt Integritätsbedingungen/Konsistenzforderungen (**Constraints**), die die Integrität des DBS garantieren:

Entitätsintegrität

Primärschlüssel darf in einer Relation niemals unbekannt sein

Referentielle Integrität (Verweisintegrität)

Fremdschlüssel bezeichnen Schlüssel, die ein Entity in einer fremden Relation referenzieren

Entity, das in der DB referenziert wird, muss in der DB existieren

Hörsaalübung

Stellen Sie ein Datenmodell für einen kleinen Web-Shop auf.

5.1 SQL Elemente zum Ausdruck von Integrität

Im Beispiel der Kunden/Konten gab es in der Kontentabelle eine Referenz auf die Kundentabelle:

```
mysql> desc kunde;
+-----+-----+-----+-----+-----+-----+
| Field  | Type          | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| ▶ kunr | int(11)       |      | PRI  | 0        |       |
| name   | varchar(20)   | YES  |      | NULL     |       |
| adresse | varchar(200) | YES  |      | NULL     |       |
+-----+-----+-----+-----+-----+-----+

mysql> desc konto;
+-----+-----+-----+-----+-----+-----+
| Field  | Type          | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| konr   | int(11)       |      | PRI  | 0        |       |
| ▶ kunr | int(11)       | YES  |      | NULL     |       |
| valuta | date          | YES  |      | NULL     |       |
| saldo  | float         | YES  |      | NULL     |       |
+-----+-----+-----+-----+-----+-----+

foreign key (kunr) references kunde (kunr)
```

Datenmodell für eine Anwendung

```
mysql> select * from kunde;
```

```
+-----+-----+-----+
| kunr | name | adresse |
+-----+-----+-----+
```

```
| 1 | as | NULL |
| 2 | jl | NULL |
| 3 | bp | NULL |
```

```
+-----+-----+-----+
```

```
3 rows in set (0.00 sec)
```

```
mysql> select * from konto;
```

```
+-----+-----+-----+-----+-----+
| konr | kunr | valuta | saldo |
+-----+-----+-----+-----+-----+
```

```
| 1 | 1 | 2004-07-12 | 100 |
| 2 | 2 | 2004-07-12 | 10 |
| 3 | 3 | 2004-07-12 | 1000 |
| 4 | 1 | 2004-07-12 | 0 |
```

```
+-----+-----+-----+-----+-----+
```

```
4 rows in set (0.00 sec)
```

Was passiert, wenn eine Zeile in der Tabelle Kunde gelöscht wird, mit den entsprechenden Konten?

SQL sieht dafür vor, dass man Constraints definieren kann:

Datenmodell für eine Anwendung

```
[CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)  
REFERENCES tbl_name (index_col_name, ...)  
[ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]  
[ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

Bei `ON DELETE CASCADE` werden die Zeilen der Tabelle gelöscht, wenn in der referenzierten Tabelle eine Zeile mit referenziertem Schlüssel gelöscht wird.

(->**Konten werden gelöscht**, wenn Kunde gelöscht wurde)

Bei `ON DELETE SET NULL` werden die Schlüsselattribute der Zeilen der Tabelle auf `NULL` gesetzt, wenn in der referenzierten Tabelle eine Zeile mit referenziertem Schlüssel gelöscht wird.

(->**In Konten wird Kunde auf NULL gesetzt**, wenn Kunde gelöscht wurde)

Bei `ON DELETE RESTRICT` wird das Löschen der Zeile in der referenzierten Tabelle verboten.

(->**Kunde kann nicht gelöscht werden** solange Konten für ihn vorhanden sind)

Die `ON UPDATE` Option funktioniert analog bei Updates.

Beispiel:

```
mysql> create table kunde (kunr integer,  
                           name varchar(20),  
                           adresse varchar(200),  
                           primary key (kunr)  
                           ) type=innodb;
```

Datenmodell für eine Anwendung

```
mysql> create table konto (konr integer,  
                           kunr integer,  
                           valuta date,  
                           saldo float ,  
                           primary key (konr),  
                           foreign key (kunr) references kunde (kunr)  
                           on delete restrict  
                           );
```

6 Normalisierung von Datenbanken

Der logischer Entwurf beinhaltet i.A. unnötige Redundanzen, die zu verschiedenen Problemen führen können:

Speicheranomalie

Redundante Informationen benötigen unnötigen Speicherplatz

Löschanomalie (Delete-Anomalie)

Redundante Informationen erhöhen den Aufwand beim Löschen von Einträgen

Änderungsanomalie (Update-/Modify-Anomalie)

Redundante Informationen erhöhen den Aufwand zur Änderung und zur Integritätssicherung

Anomalien werden an folgendem Beispiel diskutiert:

Jeder Mitarbeiter einer Bank ist genau einer Abteilung zugeordnet.

Ein Mitarbeiter kann mehrere Aufgabenbereiche innerhalb seiner Abteilung haben.

Die Bank sei durch folgende Struktur repräsentiert (unvollständig)

Aufgaben	Name	Vorname	Abteilung	Aufgabenbereich
	Lopez	Jennifer	Privatkunden	Aquise
	Brat	Pitt	Firmenkunden	Beratung
	Gates	Bill	EDV	Office-Betreuung
	Gates	Bill	EDV	Programmierung
	Duck	Dagobert	Handel	Geldhändler

Speicheranomalie:

Aufgaben	Name	Vorname	Abteilung	Aufgabenbereich
	Lopez	Jennifer	Privatkunden	Aquise
	Brat	Pitt	Firmenkunden	Beratung
	Gates	Bill	EDV	Office-Betreuung
	Gates	Bill	EDV	Programmierung
	Duck	Dagobert	Handel	Geldhändler
	Kagermann	Henning	EDV	

Ein **neuer Mitarbeiter** Herr Kagermann kommt in die Abteilung EDV

Bislang gibt es aber noch **keine** Zuteilung eines Aufgabengebiets

Die **Speicherung** zum jetzigen Zeitpunkt ist **unmöglich**, da in diesem Fall die Relation unvollständig wäre, d.h. die Datenbank wäre in keinem konsistenten Zustand

Löschanomalie

Aufgaben	Name	Vorname	Abteilung	Aufgabenbereich
	Lopez	Jennifer	Privatkunden	Aquise
	Brat	Pitt	Firmenkunden	Beratung
	Gates	Bill	EDV	Office-Betreuung
	Gates	Bill	EDV	Programmierung
	Duck	Dagobert	Handel	Geldhändler
	Kagermann	Henning	EDV	SAP-Betreuung

Herr Kagermann gibt Aufgabenbereich SAP-Betreuung ab, bleibt aber nach wie vor im Unternehmen und in Abteilung EDV

Löschen unmöglich, da sonst die Information, dass Kagermann Mitarbeiter von EDV ist, verloren ginge

Änderungsanomalie

Aufgaben	Name	Vorname	Abteilung	Aufgabenbereich
	Lopez	Jennifer	Privatkunden	Aquise
	Brat	Pitt	Firmenkunden	Beratung
	Gates	Bill	EDV	Office-Betreuung
	Gates	Bill	EDV	Programmierung
	Duck	Dagobert	Handel	Geldhändler
	Kagermann	Henning	EDV	SAP-Betreuung

Herr Bill Gates ändert Namen in Bill Microsoft

Problem: alle Tupel mit Eintrag Gates und Bill müssen gesucht und geändert werden

Eine Lösung dieser Probleme sind spezielle Formen, die das Relationenschema erfüllen muss.

6.1 Normalisierung (informal)

Normalformen dienen der **Verfeinerung des logischen Entwurfs**. Sie stellen Einschränkungen bestimmter Art dar, denen Relationen genügen müssen, um **Anomalien** zu **vermeiden**

Ziel: Vermeidung von Redundanzen durch Aufspalten des Relationenschemas

6.2 Erste Normalform

Ein Relationenschema ist in **erster Normalform** (1NF), wenn
die Attribute nicht mehr weiter zerlegbar (= atomar) sind und
es keine Wiederholungsgruppen gibt

Erste Normalform eliminiert nicht-atomare oder komplexe Attribute.

Problem:

Die erste Normalform führt oft zu neuen, mehrfachen Redundanzen, deshalb gibt es weitere Normalformen.

Beispiel:

Bestellung	BestellNr	Datum	KundenNr	VersandId	ProduktId
	1011	02.11.2003	101	2056	306
	1013	03.11.2003	104	2022	306, 307, 119
	1014	12.11.2003	111	2019	411,112

Das Schema ist nicht in 1NF, da eine Bestellung (2. und 3. Zeile) mehrere ProduktId's haben.

Lösung:

Bestellung	BestellNr	Datum	KundenNr	VersandId	ProduktId
	1011	02.11.2003	101	2056	306
	1013	03.11.2003	104	2022	306
	1013	03.11.2003	104	2022	307
	1013	03.11.2003	104	2022	119
	1014	12.11.2003	111	2019	411
	1014	12.11.2003	111	2019	112

6.3 Zweite Normalform

Weitere Normalformen versuchen, funktionale Abhängigkeiten zu reduzieren.

Funktionale Abhängigkeit:

Eine Attributmengung X bestimmt die Attributmengung Y funktional ($X \rightarrow Y$, Y ist von X funktional abhängig), genau dann, wenn es für jeden X -Wert genau einen Y -Wert gibt.

Volle funktionale Abhängigkeit:

Ein Attribut A ist von einer Attributmengung X voll funktional abhängig, genau dann, wenn es keine Teilmenge von X gibt, die A funktional bestimmt.

Schlüssel:

Minimale Attributmengung, die jedes Tupel einer Relation eindeutig bestimmt

Teilschlüssel:

Teilmenge des Schlüssels

Nichtschlüsselattribut:

Attribut, das kein Teilschlüsselattribut ist

Ein Relationenschema ist in 2. Normalform (**2NF**), wenn

Das Schema in der 1. Normalform ist und

zusätzlich jedes Nichtschlüsselattribut vom Schlüssel voll funktional (bzw. von keinen Teilschlüssel) abhängig ist.

Wenn ein **Schlüssel nur aus einem Attribut** besteht, dann ist es in **2. Normalform**, wenn jedes Nichtschlüsselattribut vom Schlüssel funktional abhängig ist.

Beispiel:

Ein Dozent hält pro Semester nur eine Vorlesung, kann diese aber in verschiedenen Semestern anbieten.

Veranstaltungen	Vorlesung	Dozent	Semester	Fachbereich
	Verteilte Systeme	Schütte	WS2003	Informatik
	Betriebssysteme	Zimmerling	WS2003	Informatik
	Media System Design	Krajewski	WS2003	MSD
	Verteilte Systeme	Schütte	WS2004	Informatik

Funktionale Abhängigkeiten:

{Dozent} -> {Fachbereich}

{Dozent, Semester} -> {Vorlesung}

Schlüssel:

{Dozent, Semester}

Die Relation Veranstaltungen ist **nicht in 2. Normalform**, weil Fachbereich (Nichtschlüsselattribut) nur von einem Teil des Schlüssels funktional abhängig ist .

Die Relation Veranstaltungen muss in zwei Relationenschemata

Dozenten(Dozent, Fachbereich) und

Vorlesungen(Dozent, Semester, Vorlesung)

aufgespaltet werden, dann ist 2NF erfüllt.

6.4 Normalisierung (formaler)

Im Folgenden wird eine formale Erklärung von Normalformen beschrieben.

Das Schlüssel-Konzept

Sei EM Entitätsmenge mit den Attributen A_1, A_2, \dots, A_n . Ein Attribut A aus $\{A_1, A_2, \dots, A_n\}$ heißt **Schlüsselattribut** auf EM genau dann, wenn für alle Entitäten E, E' aus EM gilt: wenn $E = E'$, so auch $A(E) = A(E')$.

Eine Teilmenge $\{A_{s1}, \dots, A_{sk}\}$ von $\{A_1, A_2, \dots, A_n\}$ heißt **Schlüsselkandidat** genau dann, wenn für alle $E = E'$ auch $(A_{s1}(E), A_{s2}(E), \dots, A_{sk}(E)) = (A_{s1}(E'), A_{s2}(E'), \dots, A_{sk}(E'))$.

$\{A_{s1}, \dots, A_{sk}\}$ heißt **Schlüssel** für EM genau dann, wenn a) $\{A_{s1}, \dots, A_{sk}\}$ ist Schlüsselkandidat und b) kein weiterer Schlüsselkandidat existiert mit weniger als k Attributen (Minimalität).

Abhängigkeiten

R sei Relationenschema über den Attributen A_1, A_2, \dots, A_n . X, Y und Z seien Attributkombinationen von $\{A_1, A_2, \dots, A_n\}$ (d. h. Teilmengen dieser Menge), r sei Relation über R.

Y ist **funktional abhängig** von X (Schreibweise: $X \rightarrow Y$), genau dann, wenn für alle Entitäten E, E' aus r gilt: $X(E) = X(E') \Rightarrow Y(E) = Y(E')$.

Z heißt **transitiv abhängig** von X über Y genau dann, wenn $(X \rightarrow Y)$ und nicht $(Y \rightarrow X)$ und $(Y \rightarrow Z)$.

Normalformen

1. Normalform (1NF)

Eine Relation (Tabelle) befindet sich in 1NF, wenn im Kreuzungspunkt einer jeden Zeile und Spalte genau ein nicht zusammengesetzter (atomarer) Attributwert steht.

2. Normalform (2NF)

Eine Relation befindet sich in 2NF, wenn sie in 1NF ist und alle Nichtschlüsselattribute vom gesamten Schlüssel funktional abhängig (nicht aber von Teilschlüsselattributen) sind.

3. Normalform (3NF)

Eine Relation in 1NF befindet sich auch in 3NF, wenn kein Nichtschlüsselattribut von einem Schlüsselattribut transitiv abhängig ist.

Algorithmus zur Erlangung der 2NF

1. Schlüssel für Relation r festlegen, falls dieser nur aus einem Attribut besteht, so liegt bereits 2NF vor
2. Relation auf Teilschlüsselabhängigkeiten untersuchen
3. Neue Relation r' bilden, die einen Teilschlüssel (dieser ist nun Schlüssel von r') und die allein von ihm funktional abhängigen Nichtschlüsselattribute enthält
4. Streichen der Nichtschlüsselattribute aus der Relation r
5. Vorgang ab dem 2. Punkt wiederholen bis alle Nichtschlüsselattribute von r vom gesamten Schlüssel funktional abhängig sind.

Hörsaalübung:

Bei der Realisierung eines Eventkalenders wurde ein einfaches Datenmodell implementiert. Besucher sollen Events angezeigt bekommen und Events buchen können.

Realisierung: eine Tabelle „event“ mit folgendem Aufbau:

<i>event _id</i>	<i>event _datum</i>	<i>event _beschreibung</i>	<i>besucher _id</i>	<i>besucher _name</i>	<i>besucher _vorname</i>	<i>buchungs _datum</i>	<i>anzahl _tickets</i>
1	01.02.04	Genisis Concert	123	Schütte	Alois	-	0
2	11.02.04	Rolling Stones	123	Schütte	Alois	15.01.04	3

Wie ist das gewählte **Datenmodell** zu **bewerten** (Vorteile, Nachteile)?

Wie würde es in **2NF** aussehen?

7 Abbildung von ERMen auf Relationenmodelle

Wenn der konzeptionelle Entwurf in Form eines ER-Modells vorliegt, ist die Aufgabe, das logische Modell herzuleiten. Dabei kann man folgende Vorgehensweise anwenden:

Verfahren: ER-Modell -> Relationenmodell:

Abbildung von Beziehungstypen auf Relationenschemata

Abbildung von Entity-Typen auf Relationenschematas

Attribute werden zu Attributen des Relationenschemas

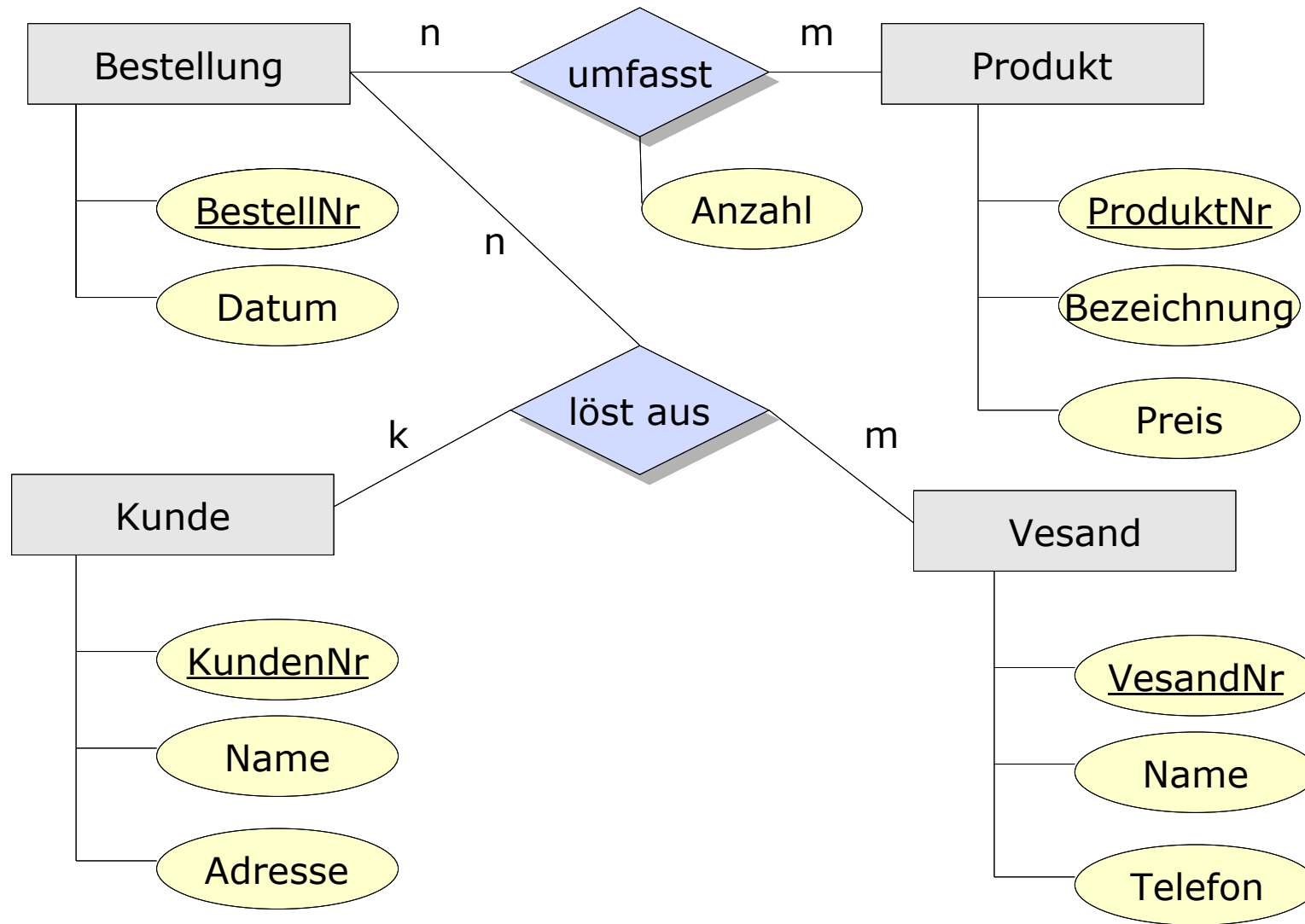
Schlüssel werden übernommen

Kardinalitäten der Beziehungen werden durch Wahl der Schlüssel bei den zugehörigen Relationenschemata ausgedrückt

Wenn möglich: Verschmelzung von Relationenschemata von Entity- und Beziehungstypen

Einführung von Fremdschlüsseln in verbleibenden Relationenschemata zur Referenzierung der Entities

Dieses Verfahren wird am folgenden ER-Modell eines **Online-Shops** demonstriert.



Das Grundprinzip der Abbildung kann folgendermassen beschrieben werden:

Abbildung von **Beziehungstypen** auf **Relationenschemata**

Abbildung von **Entity-Typen** auf **Relationenschematas**

Attribute werden zu Attributen des Relationenschemas

Schlüssel werden übernommen

Kardinalitäten der Beziehungen werden durch Wahl der Schlüssel bei den zugehörigen Relationenschemata ausgedrückt

Wenn möglich: Verschmelzung von Relationenschemata von Entity- und Beziehungstypen

Einführung von Fremdschlüsseln in verbleibenden Relationenschemata zur Referenzierung der Entities

Beziehungstypen -> Relationenschemata:

Abbildung als Relationenschema mit allen Attributen der Relation

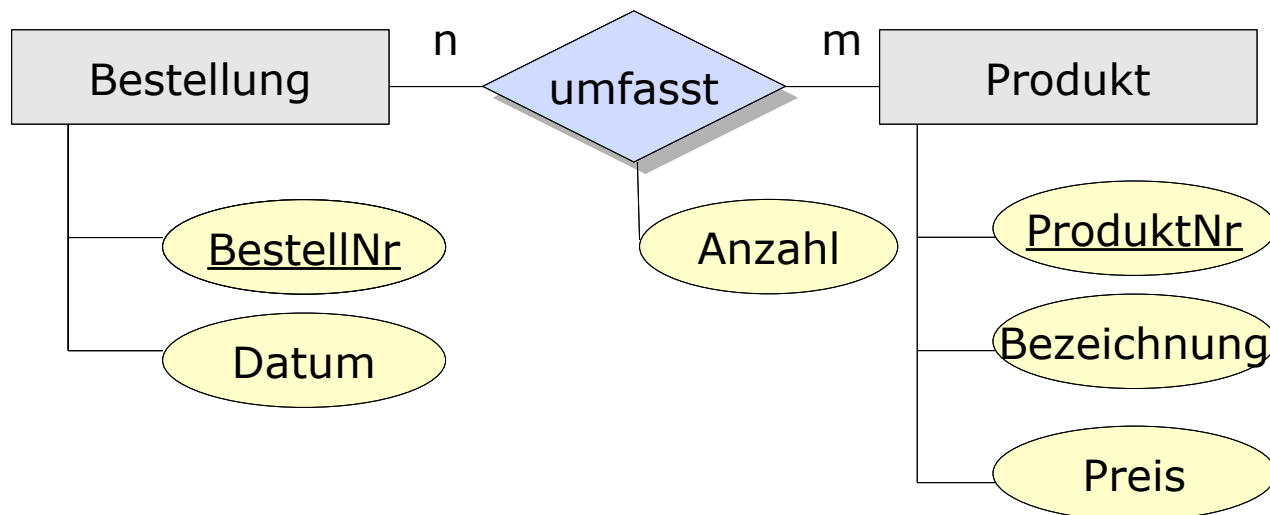
Übernahme aller Primärschlüssel der beteiligten Entities als Fremdschlüssel

Wahl des Primärschlüssels für die Beziehung auf Basis der Kardinalitäten

m:n Beide Primärschlüssel der Entities werden zusammen Schlüssel der Relation

1:n Primärschlüssel der n-Seite wird Schlüssel

1:1 Beide können als Schlüssel fungieren; Wahl eines von beiden



umfasst

<i>BestellNr</i>	<i>ProduktNr</i>	<i>Anzahl</i>
101	707	1
101	709	30
103	707	4

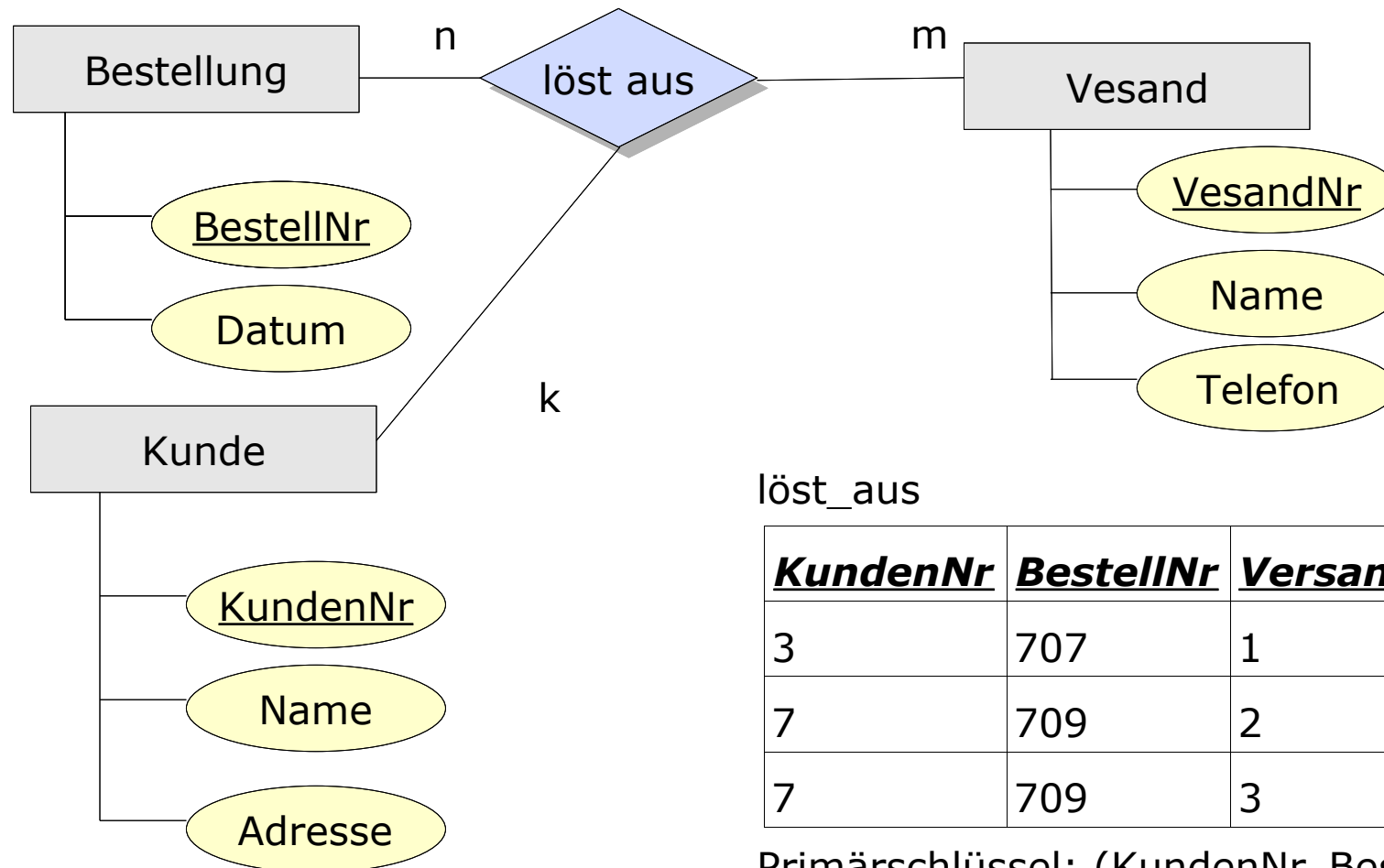
Primärschlüssel: (BestellNr, ProduktNr)

Beziehungstypen -> Relationenschemata (mehrstellige Relationen):

Wahl des Primärschlüssel

Betrachtung jedes beteiligten Entity-Typs nach den beschriebenen Regeln

Ggf. Konkatenation der jeweils notwendigen Schlüssel



löst_aus

<u>KundenNr</u>	<u>BestellNr</u>	<u>VersandNr</u>
3	707	1
7	709	2
7	709	3

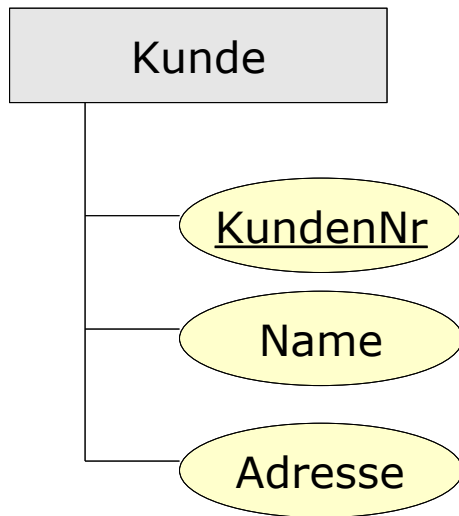
Primärschlüssel: (KundenNr, BestellNr, VersandNr)

Entity-Typen -> Relationenschematas

Relationenschema mit allen Attributen des Entity-Typs

Übernahme aller Schlüssel

Wahl eines geeigneten Primärschlüssels



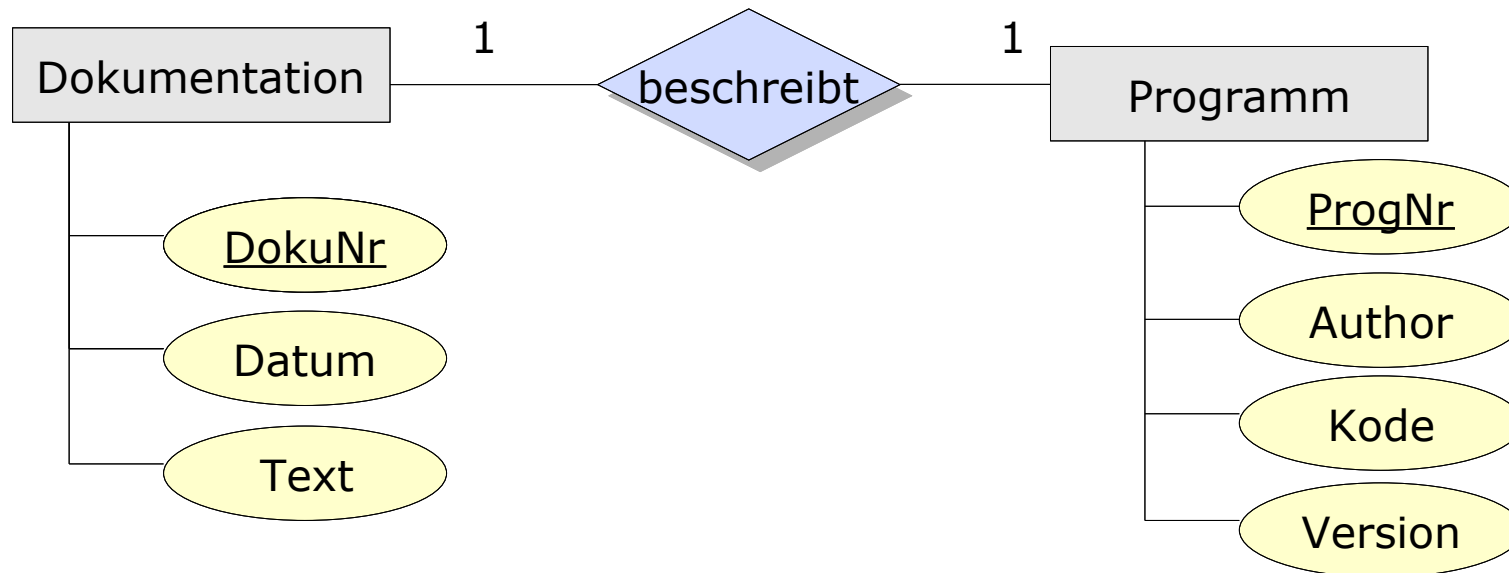
Kunde

<i>KundenNr</i>	<i>Name</i>	<i>Adresse</i>
2	as	Konz, ...
3	jl	
5	bp	

Primärschlüssel: (KundenNr)

Abbildung spezieller Relationstypen – 1:1 Beziehungen

Bei 1:1 Relationen zwischen Entities können Relation und Entities auch zu einem gemeinsamen Relationenschema zusammengefasst werden.



Beschreibung

<i>ProgNr</i>	<i>Author</i>	<i>Kode</i>	<i>Version</i>	<i>DokuNr</i>	<i>Datum</i>	<i>Text</i>

Hörsaalübung

Eine Datenbank mit Projektdokumenten soll erstellt werden, die folgende Eigenschaften aufweist:

1. Projektdaten sollen persistent abgelegt werden können.
2. Dokumente (Text-, Film- und Audiodateien) mit Metadaten sollen pro Projekt gespeichert werden können.
3. Pro Projekt soll festgelegt werden können, wer für die Pflege der Projektdokumente zuständig ist.

Aufgabe:

1. Erzeugen Sie ein ER-Diagramm für die Aufgabenstellung.
2. Leiten Sie aus dem ER-Diagramm ein Datenmodell ab.
3. Legen Sie eine eigene mysql-Datenbank an und erzeugen darin die in 2) definierten Tabellen.
4. Legen Sie in den in 3) erzeugten Tabellen Datensätze für 2 Projekte mit jeweils 3 zugehörigen Dokumenten an.
5. Erzeugen Sie eine Liste (select ...) von Projekten mit Projektverantwortlichen und zugehörigen Dokumenten.

8 ACID Konzept

Grundlegend für Datenbankvorgänge ist der Begriff der **Transaktion**. Für Transaktionen und ihre Verarbeitung gelten folgende Regeln:

Sie sind **atomar**, werden also **ganz oder gar nicht** ausgeführt.

Sie stellen in sich **konsistente Verarbeitungseinheiten** dar und lassen eine Datenbasis in einem **konsistenten Zustand** zurück.

Sie laufen quasi **isoliert** ab, d.h. sie bekommen die Wirkung paralleler Transaktionen nicht zu Gesicht.

Dauerhaftigkeit: Erfolgreiche, durch **commit** abgeschlossene Transaktionen werden durchgesetzt, d.h. auf dem Plattenspeicher "verewigt".

Die Anfangsbuchstaben der vier Regeln ergeben im Englischen das Merkwort: **"ACID"**.

8.1 Transaktion in MySQL

Kommerzielle Datenbanksysteme (wie Oracle) unterstützen das ACID-Konzept. Mit MySQL wird die Transaktionsverarbeitung in den neueren Versionen unterstützt.

Zunächst muss für eine Session die Autocommit-Verhalten ausgeschaltet werden. Dann können Manipulationen sessionlokal durchgeführt werden und ein explizites commit macht die Wirkung für andere Sessions sichtbar.

ACID Konzept

```
as@hal as> mysql -uas -p msd
Enter password:
mysql> select * from kunde;
+-----+-----+-----+
| kunr | name | adresse |
+-----+-----+-----+
|     1 | as   | NULL    |
|     2 | jl   | NULL    |
|     3 | bp   | NULL    |
+-----+-----+-----+
3 rows in set (0.03 sec)
```

```
mysql> set autocommit=off;
```

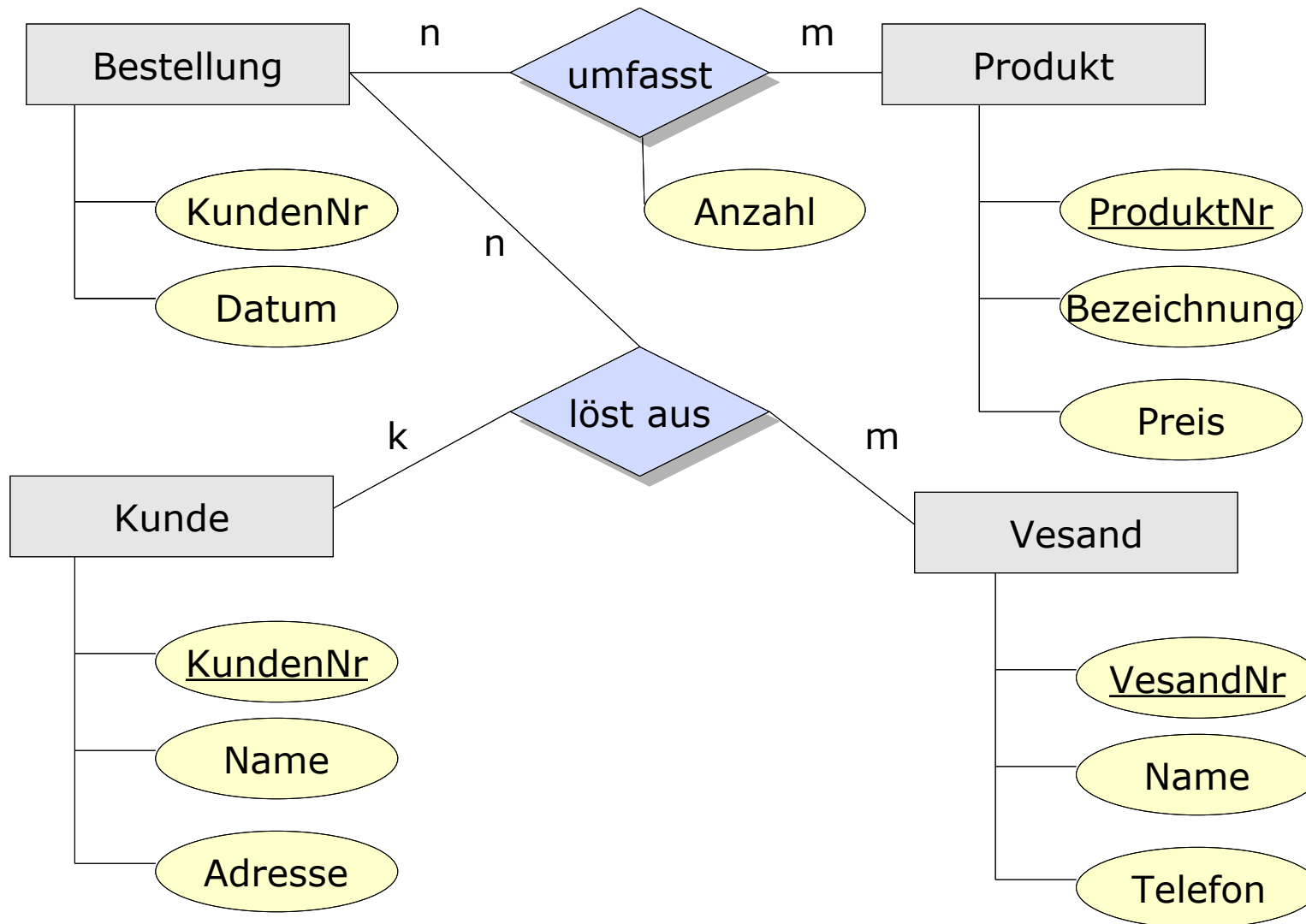
```
Query OK, 0 rows affected (0.00 sec)
```

```
mysql>
```

commit
rollback
demonstrieren

Hörsaalübung:

Realisieren Sie die aus dem ER-Modell des Onlineshop abgeleiteten Relationen als mySQL-Tabellen in der mySQL Datenbank msd.



9 Datenbankgestützte Anwendungsentwicklung

In Bereichen, in denen grosse Datenbestände zeitnah verwaltet werden müssen (Banken, Versicherungen) wird häufig **Anwendungslogik** in der **Datenbank** abgelegt.

Der Vorteil ist

dass die den Datenbestand verarbeitenden Programme die Anwendungslogik nicht stets neu implementieren müssen und

so stets eine konsistente Verarbeitung der Daten gewährleistet ist.

Beispiel (Buchen, einfach):

Der Buchungsvorgang in einer Bank muss immer auf die selbe Art und Weise vollzogen werden.

Unabhängig von der Anwendung, soll stets ein und die selbe Prozedur ausgeführt werden, die das Buchen durchführt.

Benötigt wird also ein Mechanismus, der den Buchungsalgorithmus so abbildet, dass die Datenbanktabellen **Kunde**, **Konto** und **Umsatz** immer in einem konsistenten Zustand hinterlässt.

Abhängigkeiten:

1. Ein Konto muss immer zu einem Kunden gehören
Sichergestellt durch referentielle Integrität.
2. Eine Buchung muss immer einen Eintrag in der Tabelle Umsatz erzeugen und in der Tabelle Konto das Attribut Saldo entsprechend verändern.
3. Eine Buchung darf nicht ausgeführt werden, wenn der Kontensaldo das Überziehungslimit überschreiten würde.
- 4....

Die Punkte 2 und folgende sind mit den bisher gezeigten Mitteln nicht abbildbar. SQL stellt dazu **Stored Procedures** bereit.

Hier soll **nur** ein **Überblick** über die **Datenbankprogrammierung** gegeben werden. Dies wäre eigentlich Inhalt für eine eigene Veranstaltung.

9.1 Stored Procedures

Eine Stored Procedure ist ein **Menge von SQL-Anweisungen**, die in der **Datenbank gespeichert** ist.

9.1.1. Anlegen und Löschen von Prozeduren und Funktionen

Da das Semikolon Delimiter von mysql ist, muss es von dem Erzeugen der Prozedur innerhalb des mysql-Client verändert werden.

Dann kann eine Stored Procedure angelegt werden, die einen OUT-Parameter hat, der das Ergebnis zum Aufrufer „transportiert“.

Ist die Prozedur erzeugt, ist sie in der Datenbank gespeichert und verwendet werden.

```
mysql> delimiter //
mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
  -> BEGIN
  -> select count(*) into param1 from konto;
  -> END
  -> //
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> delimiter ;
```

```
mysql> call simpleproc(@x);
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> select @x;
```

```
+-----+
| @x    |
+-----+
| 4     |
+-----+
```

1 row in set (0.00 sec)

```
mysql>
```

Neben Prozeduren sind auch Funktionen möglich. Der Code von Stored Procedures wird in der Datenbank abgelegt, die create-Statements aber sollten im File-System gespeichert werden.

```
$ cat simplfunc.sql
CREATE FUNCTION hello (s CHAR(20)) RETURNS CHAR(50)
BEGIN
    RETURN CONCAT('Hello, ',s,'!');
END
$
$ mysql ..
mysql> delimiter //
mysql> source simplfunc.sql
Query OK, 0 rows affected (0.01 sec)

mysql> delimiter ;
mysql>
mysql> select hello('MSD');
+-----+
| hello('MSD') |
+-----+
| Hello, MSD!   |
+-----+
1 row in set (0.00 sec)

mysql>
```

Prozeduren und Funktionen können mit dem „drop-Statement“ gelöscht und dem Statement „show create function/procedure“ angezeigt werden.

```
mysql> show create function hello;
...
mysql>
```

9.1.2. Lokale Variable

Mittels `declare` wird eine lokale Variable deklariert, dabei wird neben dem Name, Typ und optional ein Default-Wert festgelegt.

```
DECLARE var_name[,...] type [DEFAULT value]
```

Als elementare Datentypen stehen `int`, `char`, `date` und `bool` zur Verfügung.

```
$ cat declare.sql
create procedure anzahlKonten(out zahl int)
begin
    declare anzint default 0;
    select count(*) into anz from konto;
    set zahl = anz;
end
$
```

9.1.3.Kontrollstrukturen

Im wesentlichen können als Kontrollstrukturen

if

case

loop

leave

iterate

repeat und

while

verwendet werden.

if

```
IF search_condition THEN statement(s)
[ELSEIF search_condition THEN statement(s)]
...
[ELSE statement(s)]
END IF
```

isGerade() testet, ob die eingegebene Zahl ohne Rest durch 2 teilbar ist.

```
$ cat istGerade.sql
create function istGerade(n int) returns int
begin
    if n % 2 = 0 then
        return 1;
    else
        return 0;
    end if;
end
$
```

Eine komplexeres Beispiel ist die Schaltjahrberechnung:

```
$ cat istSchaltjahr.sql
create function istSchaltjahr(jahr int) returns int
begin
    if jahr % 400 = 0 then
        return 1;
    elseif jahr % 4 = 0 and jahr % 100 != 0 then
        return 1;
    else
        return 0;
    end if;
end
$
```

case

Entscheidungen unter mehreren Bedingungen lassen sich mit dem case-Statement realisieren, wenn die Bedingungen konstanten Charakter haben.

```
CASE case_value
  WHEN when_value THEN statement
  [WHEN when_value THEN statement ...]
  [ELSE statement]
END CASE
```

```
$ cat case.sql
create function tag(tagNr int) returns char(16)
begin
  case tagNr
    when 0 then return 'Sonntag';
    when 1 then return 'Montag';
    when 2 then return 'Dienstag';
    ...
    when 6 then return 'Samstag';
    else return 'ungültiger Tag';
  end case;
end
$
```

loop

Die einfachste Form einer Schleife ist das loop-Statement.

```
[begin_label:] LOOP  
    statement(s)  
END LOOP [end_label]
```

```
$ cat loop.sql  
create function facLoop(i int) returns int  
begin  
    declare res int default 1;  
    if i = 0 then return res; end if;  
    loop  
        set res = res * i;  
        set i = i - 1;  
        if i = 0 then return res;  
        end if;  
    end loop;  
end  
$
```

iterate

Die Iterate-Anweisung ist mit der goto-Anweisung für Schleifen anderer Programmiersprachen vergleichbar. Durch leave wird die Schleife dabei verlassen.

```
ITERATE label
```

```
$ cat iterate.sql
create function facIterate(i int) returns int
begin
  declare res int default 1;
  if i = 0 then return res; end if;
  label1: loop
    set res = res * i;
    set i = i - 1;
    if i != 0 then iterate label1;
    end if;
  leave label1;
end loop label1;
  return res;
end
$
```

repeat

In der Repeat-Anweisung kann die Abbruch-Bedingung am Ende formuliert werden.

```
[begin_label:] REPEAT  
  statement(s)  
UNTIL search_condition  
END REPEAT [end_label]
```

```
$ cat repeat.sql  
create function facRepeat(i int) returns int  
begin  
  declare res int default 1;  
  if i = 0 then return res; end if;  
  repeat  
    set res = res * i;  
    set i = i - 1;  
  until i = 0  
  end repeat;  
  return res;  
end  
$
```

while

Im While-Statement wird die Abbruchbedingung am Anfang geprüft.

```
[begin_label:] WHILE search_condition DO
    statement(s)
END WHILE [end_label]
```

```
$ cat while.sql
create function facWhile(i int) returns int
begin
    declare res int default 1;
    while i > 0 do
        set res = res * i;
        set i = i - 1;
    end while;
    return res;
end
$
```

9.1.4. Ausnahmebehandlung

Wenn beim **Zugriff auf Datensätze** bestimmte **Bedingungen** erfüllt sind, soll mit definiertem Verhalten **reagiert** werden. Z.B. soll eine Aktion ausgelöst werden, wenn keine Datensätze gefunden werden. Dazu dienen **conditions** und **handler**.

```
DECLARE condition_name CONDITION FOR condition_value
```

```
condition_value:  
    SQLSTATE [VALUE] sqlstate_value  
    | mysql_error_code
```

```
DECLARE handler_type HANDLER FOR condition_value[,...] sp_statement
```

```
handler_type:  
    CONTINUE  
    | EXIT  
    | UNDO  
  
condition_value:  
    SQLSTATE [VALUE] sqlstate_value  
    | condition_name  
    | SQLWARNING  
    | NOT FOUND  
    | SQLEXCEPTION  
    | mysql_error_code
```

SQLWARNING ist Abkürzung für alle SQLSTATE Codes die mit 01 beginnen.

NOT FOUND ist Abkürzung für alle SQLSTATE Codes die mit 02 beginnen.

SQLException ist Abkürzung für alle SQLSTATE Codes, die nicht durch die beiden ersten abgefangen sind.

Beispiel:

Definiertes Reagieren, wenn versucht wird, einen **Satz** in eine Tabelle **einzu**fügen, wobei es schon einen Satz mit dem **selben Schlüssel gibt**.

```
mysql> CREATE TABLE t (s1 int,primary key (s1));  
Query OK, 0 rows affected (0.05 sec)  
  
mysql> insert into t values (18);  
Query OK, 1 row affected (0.03 sec)  
  
mysql> insert into t values (18);  
ERROR 1062 (23000): Duplicate entry '18' for key 1  
mysql>
```

Durch das nachstehende Programm wird eine solche Situation abgefangen.

```
$ cat handler.sql
CREATE PROCEDURE handlerdemo ()
BEGIN
  DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
  SET @x = 1;
  INSERT INTO t VALUES (1);
  SET @x = 2;
  INSERT INTO t VALUES (1);
  SET @x = 3;
END;
$
```

```
mysql> call handlerdemo() /
Query OK, 0 rows affected (0.03 sec)

mysql> select * from t /
+-----+
| s1 |
+-----+
| 1 |
| 18 |
+-----+
2 rows in set (0.00 sec)
```

```
mysql> select @x;  
      -> /  
+-----+  
| @x    |  
+-----+  
| 3     |  
+-----+  
1 row in set (0.00 sec)  
  
mysql>
```

Der Datensatz ist **einmal** vorhanden, ohne den Handler wäre die Procedure abgebrochen und es wäre auch der **erste Satz nicht** in der Tabelle, der Wert von @x wäre dann 2, d.h. ein Abbruch inmitten des Kodes.

9.1.5.Zusammenfassendes Beispiel (Buchen)

Das folgende Beispiel soll einige der Sprachelement in einem grösseren Zusammenhang darstellen.

Wir betrachten unsere **Bank** mit **Kunden** und **Konten**. Eine weitere **Tabelle „Umsatz“** soll die Ein- und Auszahlungen speichern. Eine Stored Procedure soll dabei das **Buchen** der Zahlungen übernehmen, d.h.

Prüfen, ob das Konto existiert,

Prüfen, ob das Überziehungslimit nicht überschritten wird,
den Umsatz speichern und
den Kontensaldo ändern.

```
mysql> desc konto;
+-----+-----+-----+-----+-----+-----+
| Field  | Type      | Null  | Key  | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| konr   | int(11)   |       | PRI  | 0        |       |
| kunr   | int(11)   | YES   | MUL  | NULL     |       |
| valuta | date      | YES   |      | NULL     |       |
| saldo  | float     | YES   |      | NULL     |       |
| linie  | float     | YES   |      | NULL     |       |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

```
mysql> desc umsatz;
+-----+-----+-----+-----+-----+-----+
| Field  | Type      | Null  | Key  | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| umnr   | int(11)   |       | PRI  | NULL     | auto_increment |
| konr   | int(11)   | YES   | MUL  | NULL     |       |
| valuta | date      | YES   |      | NULL     |       |
| betrag | float     | YES   |      | NULL     |       |
+-----+-----+-----+-----+-----+-----+
```

```
4 rows in set (0.00 sec)
```

```
mysql>
```

```
$ cat buchen.sql
```

```
create procedure buchen(in pkonr int, in pbetrag float,  
                        in pvaluta date, out status int)
```

```
# buchen eines Umsatzes
```

```
# ToDo: Fehler abfragen, ob insert/update erfolgreich
```

```
begin
```

```
    declare n int;
```

```
    declare b float;
```

```
    declare s float;
```

```
    l1: begin
```

```
        # existiert Konto?
```

```
        select count(*) into n from konto where konr = pkonr;
```

```
        if n = 0 then
```

```
            set status = 1;
```

```
            leave l1;
```

```
        end if;
```

```
    # wird Limit überzogen
```

```
    select saldo, linie into s, b from konto where konr = pkonr;
```

```
    if s + pbetrag < b then
        set status = 2;
        leave l1;
    end if;
    # wird rückvalutarisch gebucht
    if pvaluta < curdate() then
        set status = 3;
        leave l1;
    end if;
    # Einfügen des Umsatzes
    insert into umsatz values(null,pkonr,pvaluta,pbetrag);
    # Aktualisieren das Kontensaldo
    update konto set valuta = pvaluta, saldo = saldo +pbetrag
        where konr = pkonr;
    set status = 0;
end l1;
end
$
```

9.2 Cursor

Cursor sind **Datenstrukturen im Arbeitsspeicher**, die den Zustand der Abarbeitung einer SQL-Anweisung verwalten.

Werden bei einer Select-Anweisung **mehrere Datensätze** in die **Ergebnismenge** eingefügt, muss diese Menge verwaltet werden. Dazu dient ein **Cursor**, der stets **auf einen** dieses **Datensätze zeigt**.

Folgende Anweisungen sind für die Handhabung eines Cursors notwendig:

DECLARE

Im Declare-Statement wird dem **Cursor** ein **Name** gegeben und das betreffende **Select-Statement** definiert.

OPEN

Durch Open wird das **Select-Statement ausgeführt** und der **Cursor** auf den **ersten Satz** positioniert.

FETCH

Mit dem Fetch-Statement werden die einzelnen Sätze der Ergebnismenge sequentiell eingelesen. Das erste Fetch liest den ersten Satz, das zweite den Zweiten usw.

CLOSE

Close schliesst den Cursor. Danach sind keine weiteren Fetches mehr möglich.

Das folgende Beispiel zeigt eine **Stored Procedure** zur **Ermittlung des Kontos** mit dem **grössten Saldo**.

```
$ cat cursor.sql
CREATE PROCEDURE maxSaldo(OUT konto int)
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;
    DECLARE cur1 CURSOR FOR SELECT konr, saldo FROM konto;
    DECLARE k INT;
    DECLARE s, max FLOAT;
    OPEN cur1;
    set max = 0;
    REPEAT
        FETCH cur1 INTO k, s;
        IF NOT done THEN
            IF s > max THEN
                set max = s;
                set konto = k;
            END IF;
        END IF;
    UNTIL done END REPEAT;
    CLOSE cur1;
END
$
```

9.3 Datenbank-Trigger

Datenbank-Trigger sind in der Datenbank gespeicherte **Programme**, die nicht explizite aufgerufen werden, sondern **automatisch aufgerufen werden**, wenn eine Tabelle, mit der der Trigger verbunden ist manipuliert wird.

Trigger werden wie Stored Procedures geschrieben und dann mit einer Tabelle verbunden. Dabei wird bei der Deklaration neben der Tabelle auch angegeben, wann die Stored Procedure aufgerufen werden soll: BEFORE oder AFTER einem INSERT, UPDATE oder DELETE.

Beispiel (Oracle)

```
CREATE TRIGGER hist
AFTER UPDATE or INSERT on konto
BEGIN
    ... /* Trigger Kode */
END
```

Wird im o.a. Beispiel die Tabelle konto durch eine Update- oder Insert-Operation verändert, so wird der Trigger-Kode ausgeführt.

MySQL unterstützt Trigger erst in der nächsten Version (5.1).

Trigger werden in folgenden Fällen eingesetzt:

- wenn referentielle Integrität auch über Tabellen einer verteilten Datenbank sichergestellt werden soll,

wenn das Löschen eines Primärschlüsselwertes dazu führen soll, dass der entsprechende Fremdschlüsselwert mit einem Defaultwert besetzt werden soll,
zur Sicherung der Datenintegrität, wenn Datensätze in der Datenbank erst nach der Validierung einer beabsichtigten Änderung tatsächlich geändert werden soll,
wenn Replikate einer Tabelle auf unterschiedliche Knoten einer verteilten Datenbank zu kopieren und
vor allem, wenn identischer Programmcode von unterschiedlichen Anwendungen immer ausgeführt werden muss.

9.4 Dynamisches SQL

In Arbeit!

9.5 Historisieren

In Arbeit!