

Lernzettel Informatik 2

Quick

Grundlage

Datensatz: Mehrere Datenfelder einer Datenbanktabelle, die Informationen zu einem Datenobjekt (z.B. eine Person, ein Artikel) enthalten

Eine Zeile einer Datenbanktabelle

Primärschlüssel: Ein Attribut einer Datenbanktabelle, das jeder in dieser Tabelle gespeicherten Datensatz eindeutig identifiziert.

Attribut: Eine konkrete Eigenschaft, die einem Datenobjekt zugeordnet werden kann. Eine Spalte einer Datenbanktabelle.

Attributwert: Ein konkreter Wert, der einem Attribut eines Datensatzes zugewiesen werden kann. Schnittpunkt zwischen Zeile und Spalten einer Datenbanktabelle.

Entwurfsraster einer Tabelle

Tabellenname:								
Attributname								
Datentyp								
max. Zeichenanzahl								

Datentypen

Typ	Bedeutung	Speicherplatz	Beispiel	Hinweis
INT	Integer = Ganze Zahlen von -2.147.483.648 bis + 2.147.483.647	4 Bytes	Blutspenden: 15	
DOUBLE	Kommazahlen von $-1,798 \times 10^{308}$ bis $+1,798 \times 10^{308}$	8 Bytes	Preis: 17.99	Dezimalpunkt (nicht Komma)
VARCHAR(n)	Zeichenkette mit variabler Länge bis zu n Zeichen	n + 1 Byte	Nachname: Häberle	n legt die max. Zeichenanzahl des Textes fest.
DATE	Datum von 01.01.1000 bis 31.12.9999	3 Bytes	Geburts- Datum: 1991-11-30	YYYY-MM-DD
TIME	Zeit	3 Bytes	Zielankunft: 03:56:04	hh:mm:ss
BOOLEAN TINYINT(1)	Wahrheitswert	1 Bytes	Anmeldung: true / false Bestanden: 1 / 0	Die MySQL Work- bench verwendet als Wahrheitswert den Typ TINYINT(1)

ER-Modell / Relationenmodell

Entity-Relationship-Diagramm

Ein Entity-Relationship-Diagramm (ERD) stellt einen Standard für die Modellierung von Datenbankbeziehungen dar. Mit Hilfe des Modells wird u.a. der logische Aufbau einer Datenbank dargestellt. Es bildet die Datenstrukturen ab und dient zum einen in der konzeptionellen Phase der Anwendungsentwicklung der Verständigung zwischen Anwendern und Entwicklern. Zum anderen ist das ER-Modell in der Implementierungsphase die Grundlage für das Design der Datenbank.

Die Modellierung beschreibt in erster Linie die benötigten Tabellen (Fachbegriff: Entitätstypen) und deren Beziehungen zueinander.

Für die Schreibweise der Entitätstypen gilt folgende Konvention:

Bezeichnung des Entitätstyps: Großbuchstabe als erstes Zeichen(Singular).

Bezeichnung der Attribute: Kleinbuchstabe als erstes Zeichen

Relationenmodell erstellen

Die Modellierung eines Entity-Relationship-Diagramms dient als Grundlage für einen Datenbankentwurf und beschreibt in erster Linie die benötigten Tabellen (Entitätstypen) und deren Beziehung untereinander.

Nach der Modellierung des ER-Diagramms muss dieses Modell in ein Relationenschema (Relationenmodell) überführt werden, das in einer konkreten Datenbank implementiert werden kann.

Für die Schreibweise der Relationen gilt folgende Konvention:

Bezeichnung der Relation: Kleinbuchstabe als erstes Zeichen (Plural)

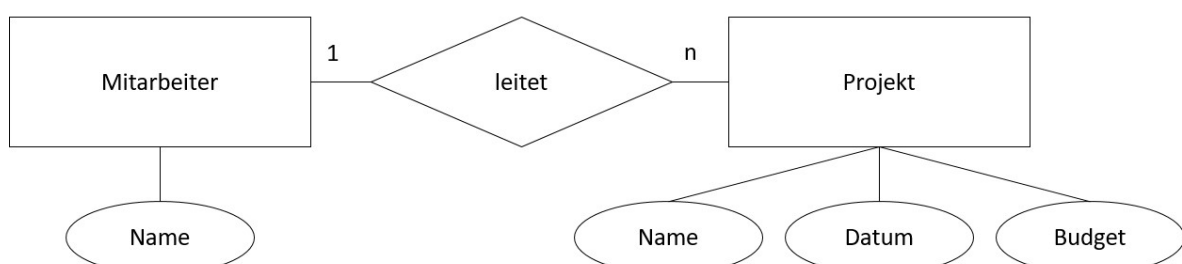
Bezeichnung der Attribute: Kleinbuchstabe als erstes Zeichen

1.Welche Aufgaben hat ein Entity-Relationship-Diagramm?

Ein ERD ist die Grundlage für das Design einer Datenbank.

Es beschreibt in erster Linie die Benötigten Tabellen (Entitätstypen) und deren Beziehungen zueinander

2.Erstellen Sie ein Entity-Relationship-Diagramm für die Daten der Fahrschüler.



3. Welche Aufgaben erfüllt ein Relationenmodell?

Ein Relationenmodell beschreibt also die in einer relationalen Datenbank zu implementierenden Relationen (Tabellen), deren Attribute, die festgelegten Datentypen sowie die verwendeten Schlüssel.

4. Erstellen Sie auf der Grundlage des entwickelten Entity-Relationship-Diagramms ein entsprechendes Relationenmodell.

fahrschueler(schuelernr INT, nachname VARCHAR, vorname VARCHAR, telefon VARCHAR, email VARCHAR, strasse VARCHAR, hausnr VARCHAR, plz VARCHAR, ort VARCHAR, geburtsdatum DATE, fahrstundenzahl INT)

Redundanzfreiheit

Primärschlüssel:

Mit dem Attribut *schuelernr* ist die eindeutige Identifizierung jedes Fahrschülers gewährleistet.

Atomare Attributwerte:

In jedem Datenfeld ist nur ein Wert enthalten.

Name getrennt in die Attribute *vorname* und *nachname*. Adresse getrennt in die Attribute *strasse*, *hausnr*, *plz* und *ort*.

Redundanzfreiheit:

In der Datenbanktabelle sind redundante Daten enthalten.

Dies kann zu Widersprüchen (Inkonsistenz) in der Datenbank führen.

Aufteilung einer Tabelle

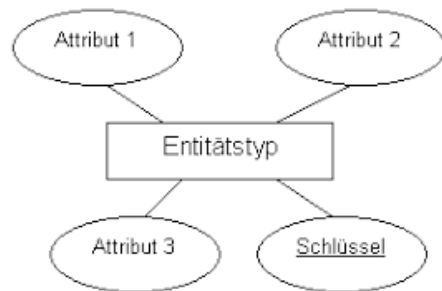
Tabellenname: fahrschueler						
Attributname	schuelernr	vorname	nachname	strasse	hausnr	Email
Datentyp	int	varchar	varchar	varchar	varchar	varchar
max. Zeichenanzahl		45	45	45	4	45
Primärschlüssel	X					

Tabellenname: ort			
Attributname	ortnr	plz	ort
Datentyp	int	varchar	varchar
max. Zeichenanzahl			
Primärschlüssel	x		

Datenbankmodell mit 2 Tabellen

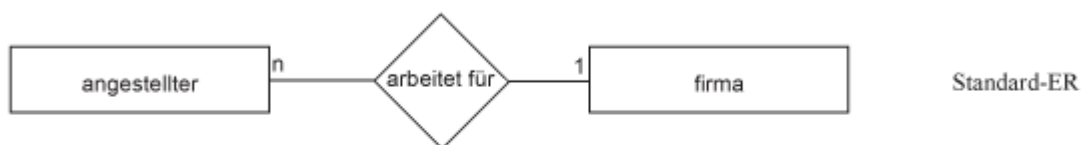
Entität: Entitäten stellen eindeutig bestimmbar, von anderen unterscheidbare Objekte der realen Welt oder unserer Vorstellung dar. Eine Entität wird durch Eigenschaften (Attribute) näher beschrieben.

Entitätstyp: Die Entitäten, die durch dieselben Eigenschaften beschrieben werden, werden zu einem Entitätstyp zusammengefasst.

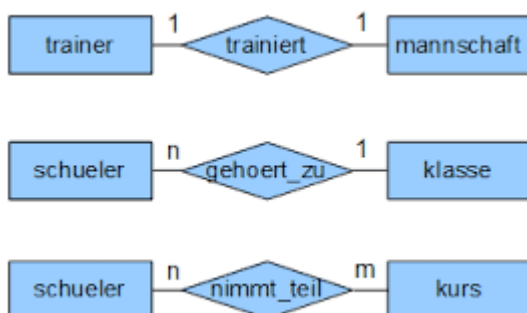


Beziehungstyp: Beziehungstypen stellen gleichartige Beziehungen zwischen den Entitäten zweier Entitätstypen dar.

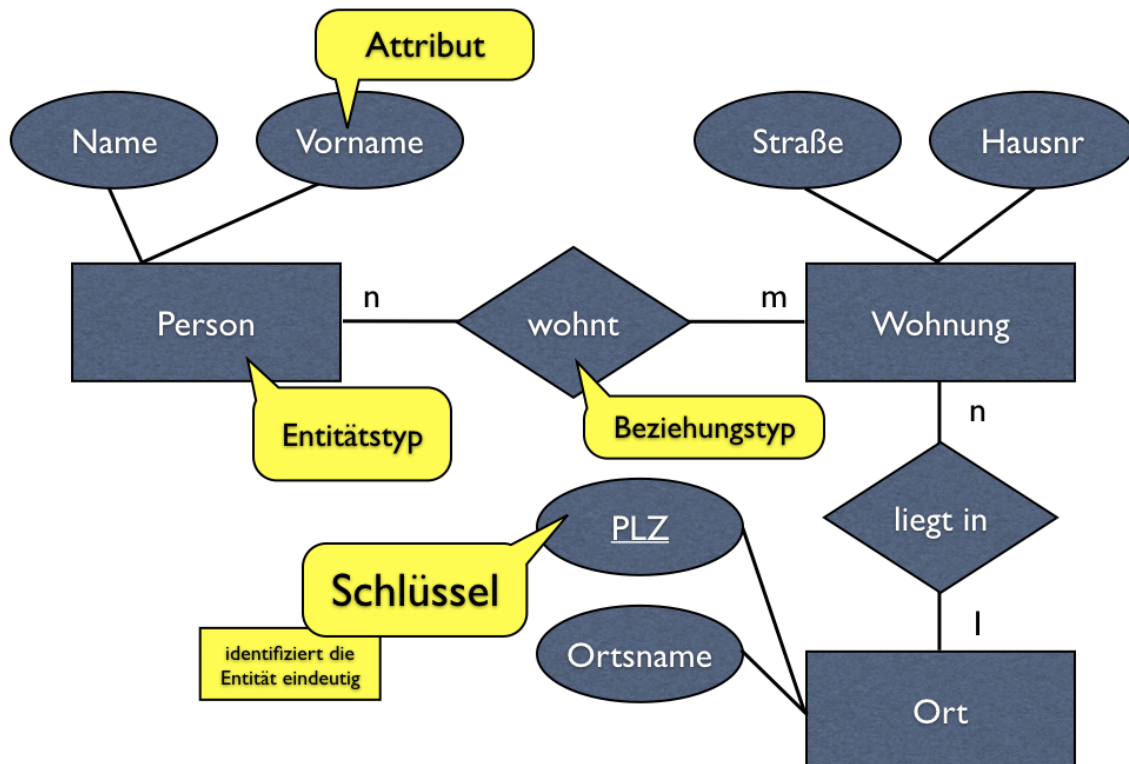
1:n-Beziehung



Kardinalität: Die Kardinalität einer Beziehung definiert wie viele Entitäten eines Entitätstyps mit genau einer Entität des anderen am Beziehungstyp beteiligten Entitätstyps (und umgekehrt) in Beziehung stehen können.



Umsetzung in ER-Modell



Umsetzung in Relationsmodell

autos(autonr INT, kennzeichen VARCHAR, kaufpreis DOUBLE, ..., ↑haendlernr INT)
 haendler(↑haendlernr INT, firma VARCHAR, telefon VARCHAR, ...)

Fremdschlüssel

Ein **Fremdschlüssel** kann Bestandteil einer Tabelle in einer relationalen Datenbank sein. Er wird benötigt, um Beziehungen zwischen Tabellen in der Datenbank zu realisieren. Dabei handelt es sich um ein Schlüsselattribut, das auf deinen Primärschlüssel in einer anderen Tabelle verweist. Er stellt sicher, dass von einer Tabelle zu einer in Beziehung stehende Tabelle navigiert werden kann.

Projektion:

SELECT attributname1, attributname2...

FROM tabellenname

[**ORDER BY** attributname1, attributname2, ... [ASC|DESC]]

Bsp:

Alle Fahrschüler mit Schülernummer sowie Vor- und Nachnamen sollen aufgelistet werden. Die Liste soll alphabetisch nach den Nachname geordnet (sortiert) sein. Bei gleichen Nachnamen soll nach den Vornamen sortiert werden.

Formulieren Sie einen entsprechenden SQL-Befehl.

```
select schuelernr, nachname, vorname from fahrschueler  
order by nachname ASC, vorname ASC;
```

Selektion:

```
SELECT attributname1, attributname2 ...
```

```
FROM tabellenname
```

```
[WHERE bedingung ]
```

```
[ORDER BY attributname1, ... [ASC|DESC]..]
```

Übersicht Operatoren

Operator	Name	Bedeutung
=	gleich	wahr, wenn die zu vergleichenden Werte identisch sind
!= oder <>	ungleich	wahr, wenn die zu vergleichenden Werte verschieden sind
<	kleiner als	wahr, wenn der links stehende Vergleichswert kleiner ist
>	größer als	wahr, wenn der links stehende Vergleichswert größer ist
<=	kleiner gleich	wahr, wenn der links stehende Vergleichswert kleiner oder gleich ist
>=	größer gleich	wahr, wenn der links stehende Vergleichswert größer oder gleich ist
BETWEEN	zwischen	wahr, wenn der links stehende Vergleichswert im angegebenen Wertebereich liegt (siehe unten)

Bsp:

Alle Fahrschüler, die vor 2001 geboren sind, sollen nach dem Geburtsdatum absteigend mit allen Informationen aufgelistet werden.

```
select * from fahrschueler  
where geburtsdatum < '2001-01-01'  
order by geburtsdatum DESC;
```

-In vielen Fällen soll nach Textmustern gesucht werden, die eine bestimmte Zeichenkette enthalten. Hierzu kann der Operator **LIKE** verwendet werden, der zwei Platzhalter zur Verfügung stellt:

% - für beliebig viele Zeichen z.B.: WHERE plz LIKE '70%'

(alle Orte, deren Postleitzahl mit den Ziffern 70 beginnt.)

_ - für ein bestimmtes Zeichen z.B.: WHERE nachname LIKE 'M__er'

(alle Personen die Maier, Meier oder Mayer heißen.)

-Eine WHERE – Klausel kann aus mehreren Bedingungssteilen bestehen. Sie werden entsprechend der Aufgabenstellung mit AND (bzw. &&) oder OR (bzw. ||) verknüpft.

AND - sowohl die erste als auch die zweite Bedingung muss erfüllt sein

z.B.: WHERE ort = 'Stuttgart'

AND nachname = 'Huber'

(alle Personen, die Huber heißen und aus Stuttgart kommen.)

OR - entweder die erste oder die zweite Bedingung muss erfüllt sein

z.B.: WHERE plz = '78150'

OR plz = '79599'

(Alle die in einem Ort mit der Postleitzahl 78150 oder mit der Postleitzahl 79599 wohnen.)

NOT - Der Operator NOT verneint die Bedingung. Somit werden alle Datensätze ausgeschlossen, die die Bedingung erfüllen, z.B.: WHERE not plz = '78150' (Es werden alle Postleitzahlen mit dem Wert '78150' ausgeschlossen)

Funktionen / Berechnungen:

Aggregatfunktionen dienen zur Zusammenfassung und Verdichtung von Rechenergebnissen. Sie arbeiten über alle Zeilen der entsprechenden Tabelle bzw. der selektierten Untergruppe. Mit ihnen werden Rechenergebnisse ermittelt, bei denen genau ein Ergebniswert aus einer Vielzahl von Eingabewerten gebildet wird.

Diese Funktionen können verwendet werden, um

-die Anzahl der selektierten Datensätze zu zählen

COUNT(PrimaryKey) (oder *)

-den Maximalwert einer Spalte zu bestimmen

MAX(Attributname)

-den Minimalwert einer Spalte zu bestimmen

MIN(Attributname)

-den Durchschnittswert einer Spalte zu bestimmen

AVG(Attributname)

-die Werte einer Spalte zu addieren

SUM(Attributname)

Am ende immer **AS Spaltenname**

Um das Ergebnis einer Berechnung oder einer Funktion in einem gewünschten Zahlenformat zu erhalten, muss die FORMAT-Funktion verwendet werden.

FORMAT(zahl, dezimalstellen)

Der Ausdruck

FORMAT(17.896578101,2) ergibt das Ergebnis: 17.90

Bsp:

Wie hoch ist die durchschnittliche Anzahl an Fahrstunden?

Das Ergebnis soll mit zwei Dezimalstellen angezeigt werden.

```
select format(avg(fahrstundenzahl),2) as durchschnittliche_stundenzahl  
from fahrschueler;
```

Redundanzen:

Mit dem SQL-Schlüsselwort **DISTINCT** können doppelte Werte bei einem Abfrageergebnis verhindert werden. Es sorgt dafür, dass Datensätze, die bereits im Abfrageergebnis vorhanden sind, nicht wiederholt werden. (z.B bei Ort, plz).

Dazu muss das Schlüsselwort **DISTINCT** direkt hinter dem **SELECT** platziert werden.

Bsp:

Es sollen die Vornamen der Fahrschüler angezeigt werden. Jeder Vorname soll nur einmal angezeigt werden.

```
select distinct vorname  
from fahrschueler;
```

Gruppierung:

```
SELECT ort, COUNT(*) AS Anzahl_der_Fahrschueler  
FROM fahrschueler  
GROUP BY ort;
```

Die **GROUP BY** – Klausel kann mit der **HAVING** – Klausel erweitert werden. Sie ermöglicht es, eine gruppierte Ergebnismenge einzuschränken.

```
SELECT ort, COUNT(*) AS Anzahl_der_Fahrschueler  
FROM fahrschueler  
GROUP BY ort  
HAVING COUNT(*) > 2;
```

Nur werte größer als 2

Bsp:

Es soll ermittelt werden, wie viele Fahrstunden insgesamt je Wohnort der Fahrschüler genommen wurden. Angezeigt werden sollen nur die Orte, deren Fahrschüler insgesamt mehr als 20 Fahrstunden genommen haben.

```
select ort, sum(fahrstundenzahl) as Anzahl_der_Fahrstunden
from fahrschueler
group by ort
having sum(fahrstundenzahl) > 20;
```

Einfügen von Daten:

```
INSERT INTO tabellenname
[attributname1, attributname2, ....]
VALUES
(wert1, wert2, ..... );
```

Bsp:

```
INSERT INTO fahrschueler
(schuelernr, nachname, vorname, telefon, email, strasse, hausnr, plz, ort,
geburtsdatum, fahrstundenzahl)
VALUES
(13, 'Cramer', 'Susanne', '01763734572', 'susi.cramer@web.de', 'Buchenweg ',
'8', '73614', 'Schorndorf', '1999-02-18', 1);
```

Aktualisieren von Daten:

```
UPDATE tabellenname
SET attributname1 = attributwert1, attributname2 = attributwert2, ...
WHERE bedingung
```

Bsp:

Für die E-Mail-Adresse des Fahrschüler Hakan Öztürk ergibt sich somit folgende Anweisung:

```
UPDATE fahrschueler
SET email = 'hakan.oeztuerk@web.de '
WHERE schuelernr = 14
```

Löschen von Daten:

DELETE FROM tabellenname

WHERE bedingung

Bsp:

Für die E-Mail-Adresse des Fahrschüler Hakan Öztürk ergibt sich somit folgende Anweisung:

DELETE FROM fahrschueler

WHERE schuelernr = 14