Aufbau von Datenbanken

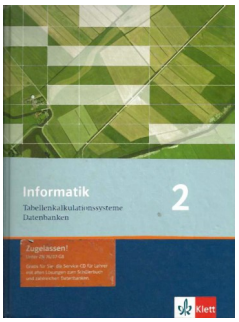
Aufbau einer Datenbanktabelle (Wiederholung)

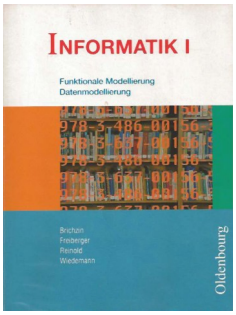
Eine ***Tabelle*** stellt einen ***einen Bauplan (eine Klasse)*** für gleichartige Objekte dar.

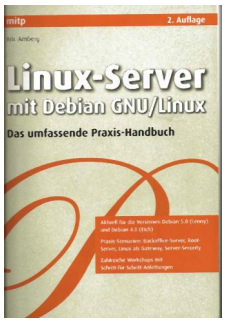
Bei Tabellen entspricht eine Zeile einem ***Datensatz (ein Objekt)***.

Die Spalten sind die ***Attribute*** der Objekte. In der Zelle steht der ***Attributwert***.

reale Objekte







buch_1	**buch_2**	**buch_3**
Titel = **Informatik 2**	Titel = **Informatik I**	Titel = **Debian Linux**
Autor = **Hubwieser**	Autor = **Brichzin**	Autor = **Amberg**
Seitenzahl = **179**	Seitenzahl = **159**	Seitenzahl = **893**
Verlag = **Klett**	Verlag = **Oldenbourg**	Verlag = **Mitpress**

Objektkarten

B U C H
Titel
Autor
Seitenzahl
Verlag

einfache Klassenkarte

B U C H			
Titel	**Autor**	**Seitenzahl**	**Verlag**
Informatik 2	Hubwieser	179	Klett
Informatik I	Brichzin	159	Oldenbourg
Debian Linux	Amberg	893	Mitpress

Primärschlüssel (Wiederholung)

Bei einer Tabelle wählt man ***ein Attribut*** oder eine ***möglichst kleine Menge von Attributwerten*** aus, um einen **Datensatz eindeutig zu identifizieren**. Dieses Attribut bzw. die kleine Menge von Attributen kann man dann zum ***Primärschlüssel*** der Tabelle wählen.

Falls es ***keine*** sinnvolle Teilmenge der Attribute in der Tabelle existieren, die als Primärschlüssel geeignet sind, so erzeugt man eine zusätzliche Spalte, die dann als ***künstlicher Primärschlüssel*** dient.

Beispiele: Kundennummer, ISBN, ID, … (Gegenteil: ***natürlicher Primärschlüssel***)

Schema einer Datenbanktabelle

Für jedes Attribut muss ein geeigneter ***Datentyp*** gewählt werden. Gängige Datentypen sind:

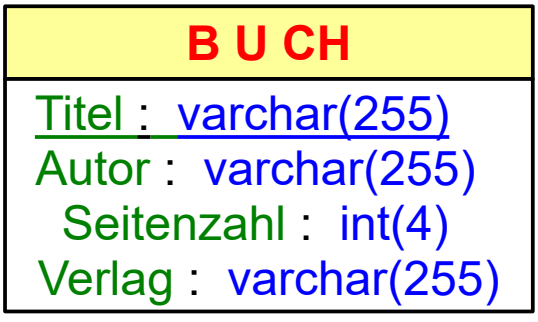
Datentyp	**Attributswertform**	**„Länge“ des Datentyps**
Ganze Zahl		***int(*** Anzahl der Stellen ***)*** z.B. int(6) → größte Zahl 999999
Zeichenfolge	Anführungszeichen: “Hallo“	***varchar(*** Anzahl der Zeichen ***)*** z.B. varchar(255)
Kommazahl	Dezimalpunkt: 1.36 nicht 1,36	***decimal(*** Anzahl der gesamten Stellen ***,*** Anzahl/Anteil der Nachkommastellen an den gesamten Stellen ***)*** z.B. decimal(6,2) → 0000,00
Datum	“JJJJ-MM-TT“	***date*** → feste Länge
Uhrzeit	“hh:mm“ oder “hh:mm:ss“	***time*** → feste Länge

Die ***Primärschlüssel*** werden bei Schemata von Datenbank bzw. bei Klassendiagrammen durch ***Unterstreichung*** gekennzeichnet.

Somit ergibt sich das ***Schema einer Tabelle***:

BUCH (Titel : varchar(255) ;
Autor : varchar(255) ;
Seitenzahl : int(4) ;
Verlag : varchar(255))

Schema einer Tabelle



erweiterte Klassenkarte

Mit dem Schema modelliert man die logische Struktur der Tabelle, bevor man sie im DBS anlegt.

Aufgabe 1:

Eine Autovermietung möchte ihre Autos in einer Datenbank speichern. Dazu braucht man die Marke (VW, Audi, ...), Modell (Golf, Passat, …), Kennzeichen, Kilometerstand, Ausleihdatum, Ausleihuhrzeit, Ausleiher, PreisproStunde (z.B. 19,49€) .

Gib das Schema der nötigen Tabelle an.

Aufgabe 2:

Für einen Sportverein sollen die angebotenen Trainings in einer Datenbanktabelle gespeichert werden. Dazu braucht man die Sportart, den Wochentag1 des Trainings, die Uhrzeit1, den Wochentag2, die Uhrzeit2, den Jahresbeitrag für diese Sportart sowie die maximale Teilnehmerzahl und die aktuelle Teilnehmerzahl.

Gib das Schema der nötigen Tabelle an.

Aufgabe 3:

In der Schule möchten wir die Schüler in einer Datenbanktabelle speichern. Dazu brauchen wir deren Nachnamen, Vornamen, Konfession, Geschlecht, Geburtsdatum, Klasse.

Gib das Schema der nötigen Tabelle an.

Aufgabe 4:

Lege die Tabellen aus den Aufgaben 1, 2 und 3 in phpmyadmin an.

Aufgabe 5:

Füge in jede der drei Tabelle zwei Datensätze ein.

Anomalien in relationalen Datenbanken

Aufgabe 6:

Importiere die Tabelle WerkstattMitKennzeichen. Eine Autowerkstatt führt eine Tabelle über alle ihre Kunden (Personen und Firmen) und deren Fahrzeuge. Bei jedem Werkstattbesuch werden die Kilometerstände der Fahrzeuge angepasst.

a) *Ein neuer Kunde (EVB-G-11 Judith Schwarz Audi rot) hat angerufen und einen Termin in der Werkstatt ausgemacht. Allerdings konnte er auswendig nicht seinen aktuellen Kilometerstand nennen. Wie würdest du vorgehen, wenn du jetzt diesen Kunden in die Tabelle einfügen möchtest?*

b) *Der Name des Kunden Max Maier wurde leider falsch eingetragen. Ändere den Namen von Maier auf Meier. Ist die Datenbank jetzt noch konsistent (in sich stimmig)? Was fällt dir auf?*

c) *Der Kunde Fritz Schneider hat ein Auto zu Schrott gefahren. Das Auto existiert nicht mehr, aber der Kunde hat versichert, dass er weiterhin Kunde dieser Werkstatt bleiben möchte. Ist es möglich nur das Auto zu löschen ohne dass der Kunde gelöscht wird?*

Eine ***Anomalie*** ist ein Fehlverhalten der Datenbank aufgrund von redundanten Daten (Regel: „every information once“).

Einfüge-Anomalie: Man spricht von eine Einfüge-Anomalie, wenn ein neuer Datensatz nicht oder nur schwierig eingetragen werden kann, weil nicht zu allen Attributen Werte vorliegen.

Änderungs-Anomalie: Man spricht von einer Änderungs-Anomalie, wenn nicht alle (redundanten) Vorkommen eines Attributwerts geändert werden. Dies führt zu ***inkonsistenten Daten*** (Unstimmigkeiten).

Lösch-Anomalie: Eine Lösch-Anomalie entsteht, wenn durch das Löschen eines Datensatzes mehr als die gewünschten Informationen gelöscht werden.

Alle Anomalien lassen sich auf ***redundante Daten*** (doppeltes Vorkommen von Informationen) in der Datenbank zurückführen. Um dieses Fehlverhalten zu verhindern ist es notwendig, dass die Informationen möglichst nur einmal in der Datenbank vorkommen. ***Der Grundtrick hierbei ist die Tabelle in mehrere kleinere Tabellen aufzuteilen und diese miteinander zu verknüpfen!*** (Wissenschaftlich: Man überführt die Datenbank in eine hohe/höhere Normalform. Eine Datenbank ist einer bestimmten Normalform, wenn sie bestimmte Kriterien erfüllt.)

Mehrere Tabellen mit Fremdschlüsseln

Beim Aufteilen von Tabellen muss man Attribute nach ihrer logischen Zusammengehörigkeit aufteilen. Bei der Aufgabe 6 wäre es sinnvoll, dass man in Fahrzeugdaten und Kundendaten aufteilt. Nun muss für jedes Attribut der Tabelle die Zugehörigkeit zu den beiden neuen Datengruppen bestimmt werden.

Aufgabe 7.1: Projekt Bier

Importiere die Tabelle BierVorlage. Hier wurden Biere mit ihren (möglichen) Eigenschaften in einer Tabelle angelegt. Das Benutzen von nur einer Tabelle hat bereits zu einigen Anomalien geführt.

Überlege dir auf Papier eine sinnvolle Aufteilung der Daten in mehrere Tabellen und bestimme den Primärschlüssel der Teiltabellen.

Tabellen verknüpfen mit Fremdschlüsseln

Ein ***Fremdschlüssel*** wird benötigt, wenn zwei Tabellen miteinander verknüpft – ***in Beziehung gesetzt*** – werden. Der Fremdschlüssel ***referenziert*** den ***Primärschlüssel der anderen Tabelle***.

Der Fremdschlüssel referenziert/übernimmt hierbei den Wert des Primärschlüssels des anderen Datensatzes und ermöglicht dadurch die ***eindeutige Verknüpfung*** der Datensätze der beiden Tabellen.

Vorsicht: Hier entstehen keine redundanten Daten, denn der Fremdschlüssel besitzt ***keinen eigenen Attributwert***, sondern zeigt nur den bereits vorhandenen Wert auch in der eigenen Tabelle an (***Referenz***: Verweis auf einen Wert, vgl. (Hyper-)Link im Internet).

Hinweis: Um auf eine andere Tabelle mit Fremdschlüsseln verweisen zu können, müssen diese Tabellen bereits vorhanden sein. Analog dazu können beim Einfügen von Daten Fremdschlüssel nur Werte annehmen, die in der anderen Tabelle bereits als Primärschlüssel vorhanden sind!

Aufgabe 7.2: Projekt Bier

a) *Verknüpfe durch Hinzufügen von Fremdschlüsseln auf geeignete Art die aufteilten Tabellen. Zeichne ein Datenbankschema zu deinen Tabellen inklusive Datentypen, Unter-/Überstreichung der Schlüssel und passenden Referenzierungspfeilen.*

b) *Implementiere die Tabellen gemäß deines Schemas. (Fremdschlüssel → Index, sodass silberner Schlüssel erscheint, Designer → erzeuge Verknüpfung)*

c) *Füge dieselben Daten aus der Ursprungstabelle BierVorlage auch in deine neuen Tabellen ein. Nutze dafür diese Befehle: (**Achte auf die Reihenfolge deiner Spalten**)*

INSERT INTO Biersorte SELECT DISTINCT Biersorte, Hefetyp FROM BierVorlage

INSERT INTO Stadt SELECT DISTINCT Stadt, Bundesland FROM BierVorlage WHERE Stadt IS NOT NULL

INSERT INTO Brauerei SELECT DISTINCT Name, Gründungsjahr, Stadt FROM BierVorlage WHERE Name IS NOT NULL

INSERT INTO Bier SELECT DISTINCT BierNr, Alkoholgehalt, Brauerei, Biersorte FROM BierVorlage

Constraints in Datenbanksystemen

Contraints definieren, wie sich die Fremdschlüssel verhalten sollen, falls der Primärschlüssel – der referenziert wird – ***verändert*** oder ***gelöscht*** wird.

Für beide Fälle gibt es vier verschiedene Modi (***RESTRICT***, ***NO ACTION***, ***SET NULL***, ***CASCADE***).

Aufgabe 8:

Die Aufgabe ist herauszufinden, wie das Datenbanksystem bei den vier Modi auf Änderungen oder Löschungen reagiert.

Importiere dir aus der Alle-Datenbank die beiden Tabellen KundeConstraint und RechnungConstraint. Klicke in der Tabelle RechnungContraint auf Struktur -> Beziehungsübersicht.

Hier wird festgelegt, wie in der Tabelle RechnungContraint verfahren wird, wenn sich Einträge in der referenzierten Spalte (Primärschlüssel der Spalte KundeConstraint) geändert (ON UPDATE) oder gelöscht (ON DELETE) werden.

Gehe dazu folgendermaßen vor:

a) *Setze beide Felder auf RESTRICT und speichere diese Einstellung. Wechsel in die Tabelle KundeConstraint und führe einen beliebigen UPDATE-Befehl auf einen Primärschlüsselwert aus (Ändere einen Primärschlüsseleintrag). Notiere die Reaktion des DBS bzw. die Auswirkung auf die Tabelle Rechnung. Führe jetzt einen DELETE-Befehl auf einen Primärschlüssel der Tabelle Kunde aus. Notiere ebenfalls die Reaktion des DBS bzw. die Auswirkung auf die Tabelle Rechnung.*

Die Reaktion kann auch eine Fehlerausgabe sein!

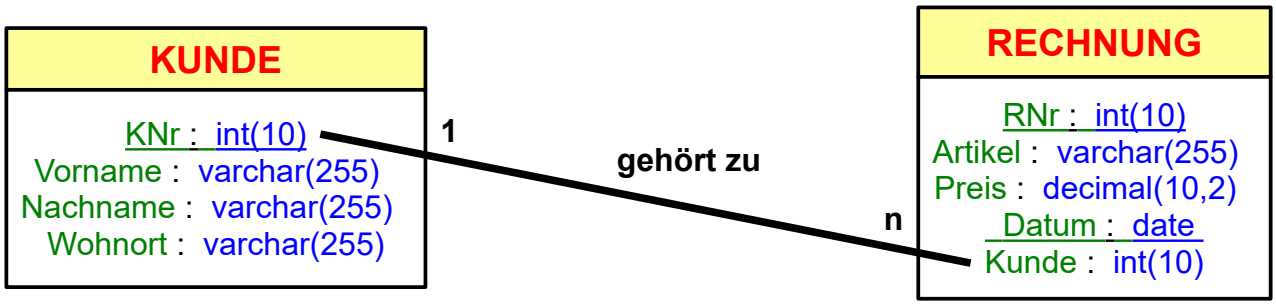
b) *Verfahre analog zum RESTRICT bei Aufgabe **a)** auch mit den anderen drei Modi in der folgenden Reihenfolge: NO ACTION, SET NULL, CASCADE*

Falls die Tabellen "zerstört" sein sollten, lösche die beiden Tabellen (DROP TABLE Tabellenname) und importiere diese beiden erneut.

c) *Überlege dir aus deinen Notizen eine sinnvolle Standardeinstellung für die Fremdschlüsselbeziehung (Constraint).*

Kardinalitäten in Klassendiagrammen mit Beziehungen

Aus der Aufgabe 7 ist bereits bekannt, dass ***einem*** Kunden ***mehrere*** Rechnungen durch mehrere Einkäufe zugeordnet werden. Diese Beziehung wird in einem Klassendiagramm folgendermaßen dargestellt:



Häufig interessieren nur die Beziehungen zwischen den Klassen. Hier bietet sich an, dass man die einfachen Klassenkarten nutzt:

Aufgabe 7.3: Projekt Bier

Zeichne zu deinem Projekt Bier ein passendes Klassendiagramm mit einfachen Klassenkarten und trage die entsprechend die Kardinalitäten ein. Finde eine passende Beschriftung für die jeweilige Beziehung.

Es existieren drei verschiedene Arten von ***Beziehungstypen***:

- ***n:1*** bzw. ***1:n***
- ***1:1*** (Beispiel: Land:Hauptstadt)
- ***n:m*** (Beispiel: Lehrer:Schüler)

Aufgabe 9:

Erstelle ein Klassendiagramm zunächst mit einfachen Klassenkarten zu den Klassen Schüler, Lehrer, Klassen, (Klassen-)Zimmer. (Nimm viel Platz zum Zeichnen; diese Aufgabe wird noch erweitert.)

Überlege dir den passenden Beziehungstyp und trage die Kardinalitäten inklusive Beschriftung ein.

Aufgabe 10:

Vervollständige dein Klassendiagramm aus der Aufgabe 9, indem du jeder Tabelle einen Primärschlüssel und gemäß der Kardinalitäten Fremdschlüssel hinzufügst.

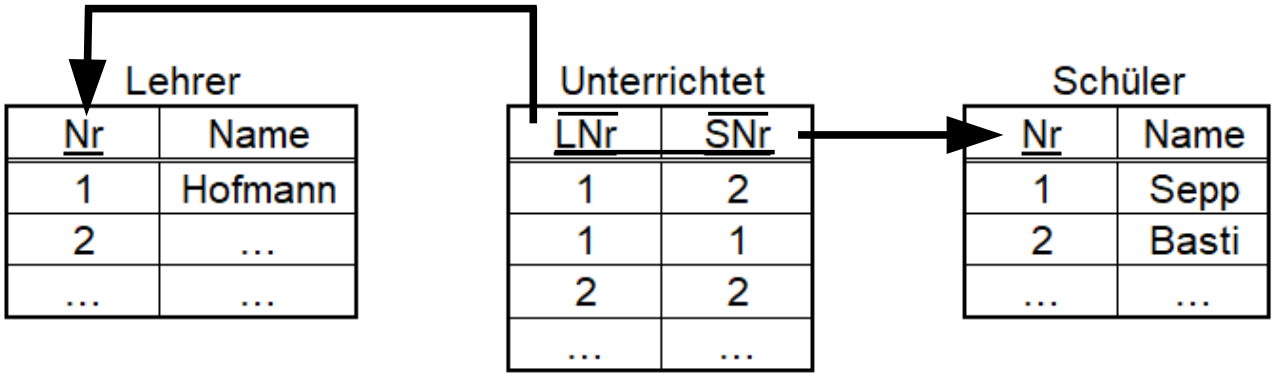
Überlege dir dabei, in welchen Klassen die Fremdschlüssel eingefügt werden müssen.

Bei welchem Beziehungstyp klappt dies nicht bzw. nicht eindeutig?

Beziehungstabellen für n:m-Beziehungen

Bei der n:m-Beziehung ist es notwendig eine eigene Beziehungstabelle anzulegen, damit dort die Kombinationen eingetragen werden können. Dies könnte folgendermaßen aussehen:

Das Beispiel ist stellt die kompletten Tabellen mit eingetragenen Datensätzen dar (**kein Klassendiagramm!**).



Die Attribute LNr und SNr sind ***Fremdschlüssel*** und ***gleichzeitig zusammen*** der ***Primärschlüssel*** der Tabellen Unterrichtet.

Aufgabe 11:

Setze die n:m-Beziehung zwischen Lehrern und Schülern in phpmyadmin um.

Abfragen über mehrere Tabellen

Da die Daten aufgrund von Redundanzen und den daraus resultieren Anomalien jetzt auf mehrere Tabellen verteilt sind, müssen wir bei SELECT-Abfragen die Informationen aus mehreren Tabellen wieder zusammensetzen.

Datenbankabfragen mit SQL (Wiederholung)

SELECT Spalten oder ******* für alle Spalten oder ***DISTINCT*** Spalte (ohne Mehrfachnennung), Aggregatsfunktion ***MAX(Spalte)***, ***MIN(Spalte)***, ***AVG(Spalte)***, ***COUNT(*)***, ***SUM(Spalte)***

FROM Tabellenname

WHERE Bedingung mit **=, <**, **>**, **<=**, **>=**, **!=** und weiteren Operatoren ***AND***, ***OR***, ***IN***, ***BETWEEN***, ***LIKE*** / ***NOT LIKE*** mit ggf. ***%***

GROUP BY Spalte

Kreuzprodukt von Tabellen

Wenn bei einer SELECT-Abfrage beim Schlüsselwort FROM ***mehrere Tabellen angegeben*** werden, wird das Kreuzprodukt auf diesen Tabellen angewendet.

Beim ***Kreuzprodukt*** wird jeder Datensatz der einen Tabelle mit jedem Datensatz der anderen Tabelle verknüpft.

Beispiel:

```
SELECT *
FROM Brauerei, Stadt
```

Aufgabe 7.4: Projekt Bier

Führe dieses oben angegebene Beispiel auf deinen Tabellen des Projekts Bier aus. Sieh dir die Arbeitsweise des Kreuzprodukts an.

Nicht jede Verknüpfung beim Kreuzprojekt ergibt Sinn. Identifiziere die Datensätze/Verknüpfungen, die für eine Abfrage zu gebrauchen sind.

Join von Tabellen

Die Join-Bedingung sortiert die unsinnigen Datensätze heraus.

```
SELECT  …
FROM Tabelle1, Tabelle2
WHERE Tabelle1.Fremdschlüssel = Tabelle2.Primärschlüssel
    AND …
```

Zur Schreibersparnis können die Tabellen umbenannt werden. In der Regel nutzt man für den neuen Namen lediglich den Anfangsbuchstaben der Tabelle:

```
SELECT  …
FROM Tabelle1 as NeuerName1, Tabelle2 as NeuerName2
WHERE NeuerName1.Fremdschlüssel = NeuerName2.Primärschlüssel
    AND …
```