Datenbanken

Datenbanken sind simpel aufgebaut.

Innerhalb der Datenbank (database) sind die Informationen in Tabellen (tables) gespeichert. Die Daten (dataset, entity) werden zeilenweise in die Tabellen eingetragen, die Spalten (attribute) bezeichnen die Informationsteile.

database “Werkstattverwaltung“

table “Mitarbeiter“

Attribute

Nummer	Name	Abteilung	Telefon
123	Maier	EDV	12212
124	Müller	Werkstatt	456645

Einträge, Datensätze

Zwischen den Einträgen bestehen Beziehungen (Relationen), deshalb werden solche Datenbanken als relationale Datenbanken bezeichnet. Die Sprache, mit der diese Datenbanken bearbeitet werden können, heißt „structured query language“ (sql), man spricht im Umkehrschluß auch von sql-Datenbanken.

Für diese Beziehungen und den Aufbau der Tabellen (Attributstruktur) gibt es nun Regeln, die dafür sorgen sollen, daß die Datenbank Informationen so speichert, daß sie jederzeit eindeutig zu finden sind und widerspruchsfrei auch bei jeder Frage die gleiche Antwort kommt.

Zwei grundliegende Forderungen an den Aufbau der Tabellen sind die Existenz eines Primaryx Key und die Redundanzfreiheit.

Primary key

Jede Tabelle benötigt ein Attribut, das eindeutig die Suche nach einem Eintrag ermöglicht. Hierzu ein paar Fachbegriffe : Ein Schlüsselkandidat ist ein Attribut, das die eindeutige Identifizierung jedes Eintrags ermöglicht. Davon kann es mehrere geben. Ein Schlüsselkandidat kann sich auch aus der Kombination von mehreren Attributen ergeben (Beispiel : Eine Personalausweisnummer kann z.b. in Österreich und Kanada gleich sein. Dann kombiniert man einfach Personal-ausweisnummer und Land, und erhält damit Eindeutigkeit). Das ist dann ein zusammengesetzter Schlüssel.

Aus den Schlüsselkandidaten muß einer gewählt werden, der dannPrimärschlüssel oder primary key genannt wird. Oft wird er Einfachheit halber eine fortlaufende Nummer hierfür benutzt.

Redundanzfreiheit

Eine Information, und das ist hier immer der Zusammenhang zwischenzwei Einträgen : also zum Beispiel daß Frau Maier in der EDV arbeitet (Maier - EDV), darf in einer database nur einmal vorkommen.

Steht zum Beispiel in der table Mitarbeiter ein Attribut Name und ein Attribut Abteilung, und in der table Personalräte steht wieder Name und Abteilung :

Mitarbeiter

Nummer	Name	Telefon	Abteilung
1332	Maier	asffsd	EDV

Personalräte

Nummer	Name	Abteilung	e-mail
12	Maier	EDV	a@w.d

Bei Redundanz kann folgendes passieren :

Frau Maier wird befördert, und steigt von der EDV in die Firmenleitung auf. Die Sekretärin macht die Datenbank auf, und speichert dies, indem sie in der table Mitarbeiter EDV nach Leitung ändert.

Nun fragen zwei verschiedene Mitarbeiter irgendwann die Datenbank ab, und die eine Frage geht nach Mitarbeiter, die andere nach Personalräte.

Einmal kommt raus, daß Frau Maier in der EDV arbeitet, einmal, daß sie Mitarbeiter der Geschäftsleitung ist. Man nennt dies eine Inkonsistenz, die dann nicht vorkommen kann, wenn die Beziehung Name <-> Abteilung nur einmal gespeichert ist.

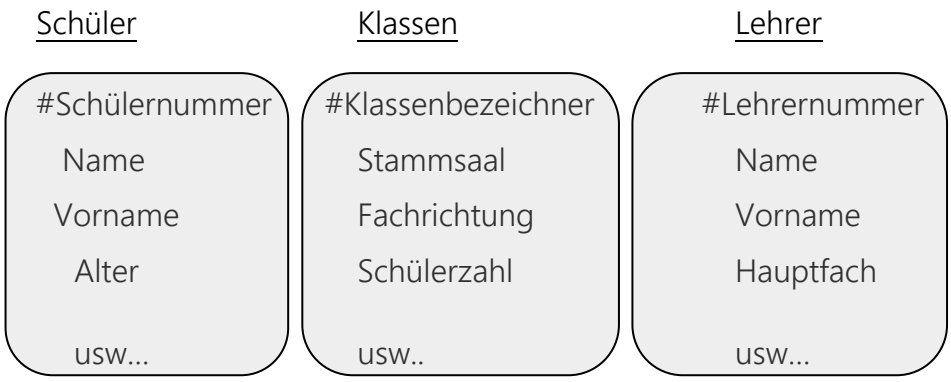
Es gibt darüber hinaus eine ganze Reihe von Anforderungen, die Datenbankstrukturen erfüllen müssen, um effizient und fehlerfrei arbeiten zu können. Hier gibt es Regelwerke, die eingehalten werden sollen : Normalformen

z.b. : http://de.wikipedia.org/wiki/Normalisierung_(Datenbank)
...das ist, um es optimistisch zu sagen, nicht unkompliziert ;-)

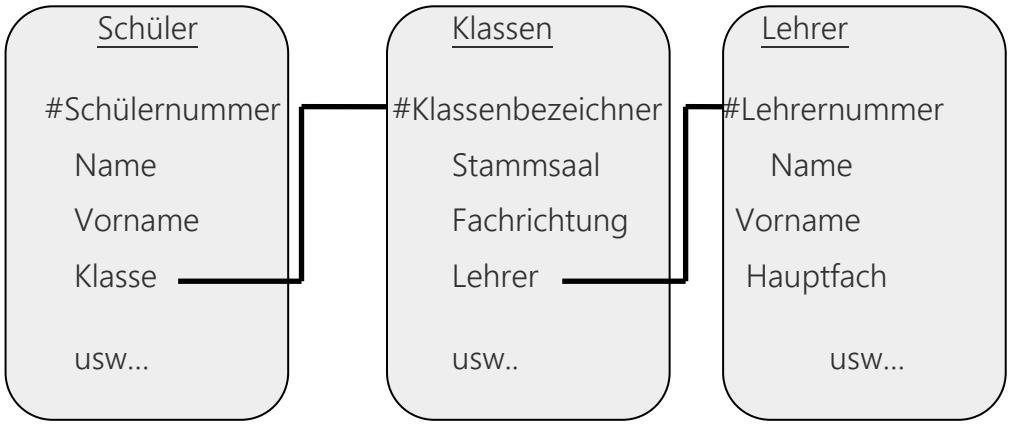
Nun wollen wir aber nicht große Datenbanksysteme für Unternehmen entwickeln, sondern nur eine kleine SQL-Datenbank benutzten, um unseren Webserver zu unterstützen. Meist wird sogar eine table genügen. Deshalb genügt hier ein einfacheres Entwurfssystem, das einige Forderungen der Normalisierung relativ einfach erfüllt, das ER-Modell.

ER – Modell (entity-relationship)

Gleich am Beispiel einer Schuldatenbank gezeigt : Die Database heißtSchule, der Entwurf beinhaltet drei tables : Lehrer, Schüler, Klassen. Die tables werden als Rechteck oder so gezeichnet, der primary key jeweils mit einem # davor als erster Eintrag und dann alle Attribute (wichtig : Attribute, keine Daten !!) :



hier werden nun die Beziehungen (Relationen) eingetragen, und zwar als Linien zwischen den jeweiligen Attributen :

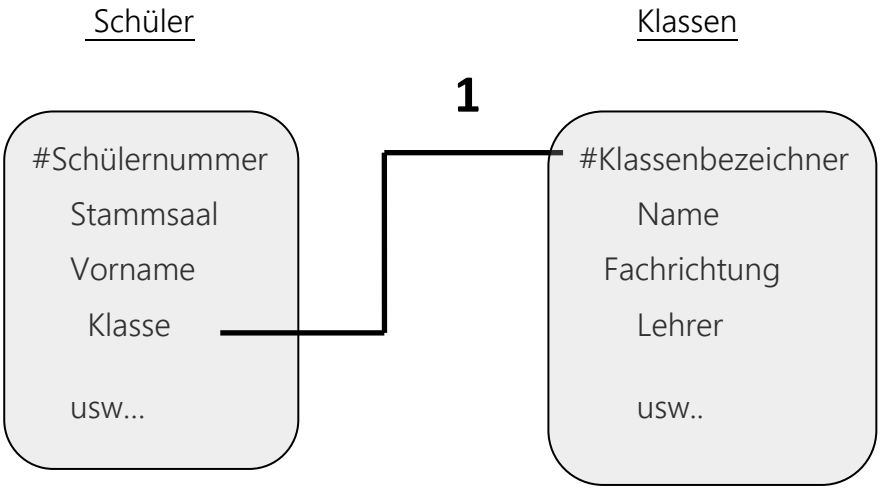


und dann werden diese Beziehungen untersucht :

1. Frage :

Ich setze mich in die table Schüler, und blicke in Richtung Klassen. Nun frage ich : kommt ein Element hier (= ein Schüler) in der Zieltabelle Klassen einmal oder öfter vor ? (Ist ein Schüler in einer oder in mehreren Klassen ?) -> Antwort : Einmal !

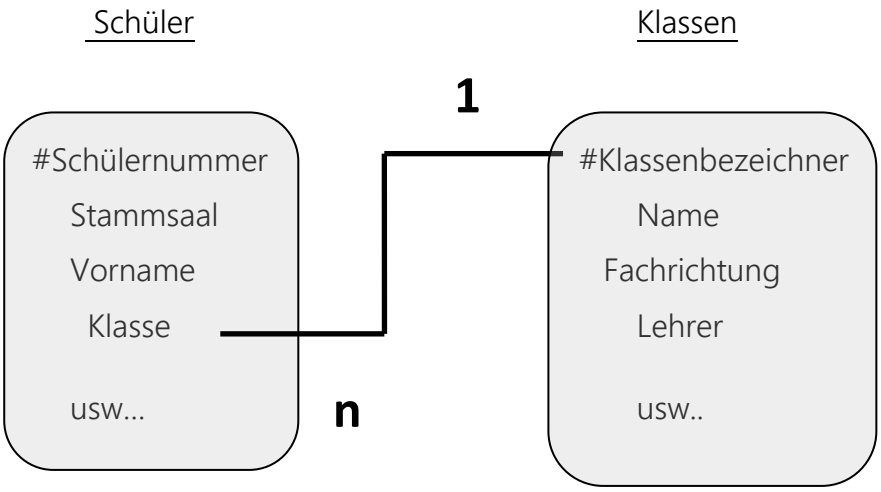
Diese 1 wird jetzt an der Linie am Zielelement angeschrieben :



2. Frage :

Nun setze ich mich in die table Klassen, und blicke in Richtung Schüler.Kommt ein Element aus meiner table in der Zieltable einmal oder öfter vor ? Antwort : öfter ! (eine Klasse steht bei mehreren Schülern dabei)

Dieses „öfter“ wird als n (oder m) an der Zieltable angeschrieben :



Diese Aussage nennt man nun Kardinalität, oben besteht eine 1:n –Kardinalität.

Man macht diese Untersuchung nun für alle Beziehungen, die zwischen den tables einer database bestehen. Dabei können folgende Ergebnisse auftreten :

1:n-Kardinalität

Alles prima, das kann so bleiben !

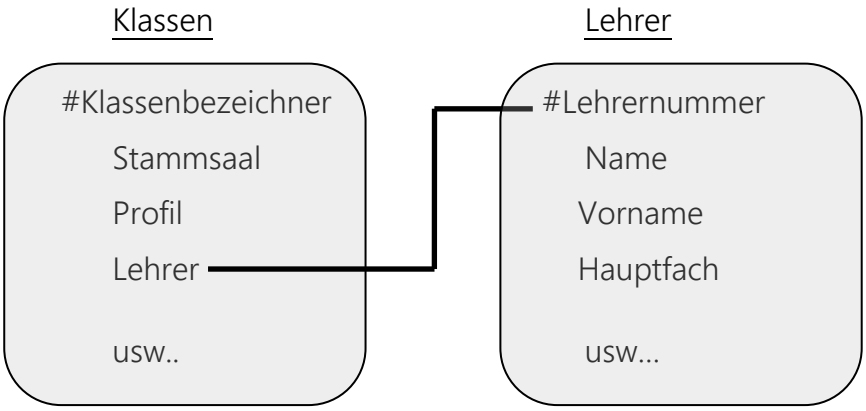
1:1 – Kardinalität

Schlecht, ist verboten. Hier kann aber immer aus den beiden tables eine einzige gemacht werden.

n:m – Kardinalität

Auch schlecht, auch verboten. Hier muß in der Regel eine Zwischentablelle gefunden werden, die für eine Entkoppelung der unerlaubten Kardinalitäten sorgt.

Beispiel für n:m

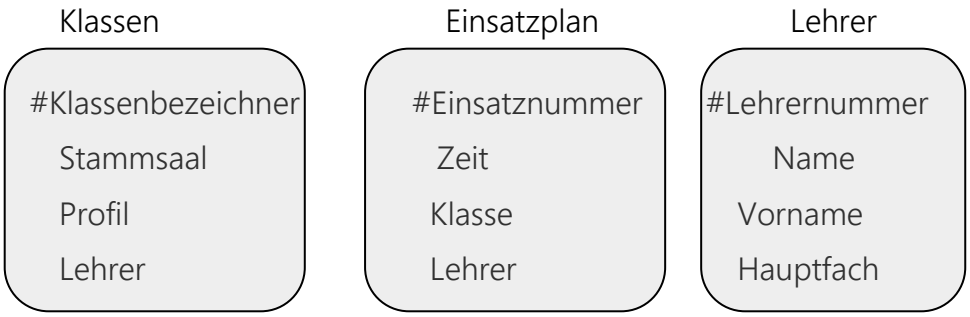


1. Frage : Aus Sicht von Klassen : kommt ein Element aus Klassen (die TE2b z.b) bei einem oder bei mehreren Lehrern vor ? Klar, es unterrichten mehrere Lehrer in der Te2b -> n !

2. Frage Aus Sicht von Lehrer : kommt ein Element aus Lehrer (z.b. der Doll) in einem oder mehreren Klassen vor ? Wieder klar, Doll hat Unterricht in mehreren Klassen -> m !

Das ist also eine n:m – Kardinalität, und die ist verboten !

Überlegen wir also eine (möglichst sinnvolle) Zwischentable, die hier für Ordnung sorgt. Pptimal wäre hier eine Liste, die irgendwie in jeder Zeile eine Beziehung zwischen den beiden primary keys aus den ursprünglichen tables herstellt. Ich nenne das mal „Einsatzplan", man könnte auch einen „Stundenplan" draus machen :



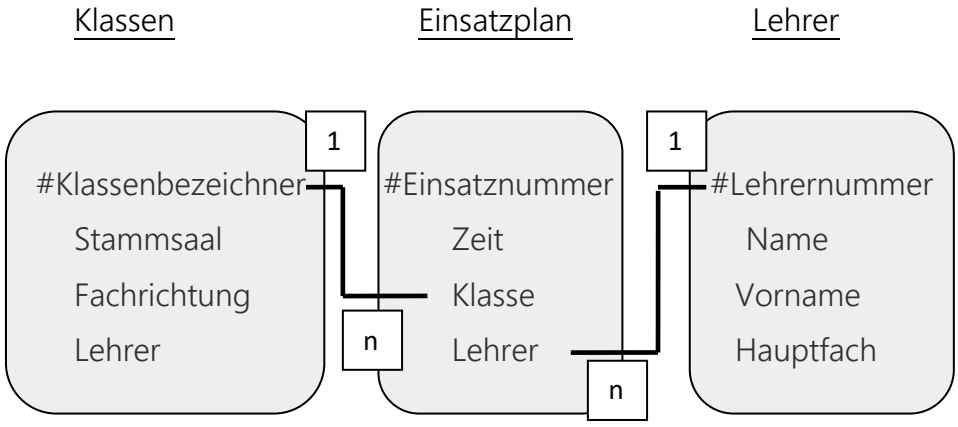
Wieder die zwei Fragen, jeweils an den Beziehungen gestellt :

Klassen -> Einsatzplan : Kommt eine Klasse im Einsatzplan öfter vor ? Ja : n !

Einsatzplan -> Klassen : Kommen in einem Einsatz eine oder mehrere Klassen vor ? Eine : 1 !

Einsatzplan -> Lehrer : Kommen in einem Einsatz ein oder mehrere Lehrer vor ? Einer : 1 !

Lehrer -> Einsatzplan : Kommt ein Lehrer in einem oder in mehreren Einsätzen vor ? n !



Nun ist alles in Ordnung, das ER-Modell ist korrekt ausgeführt !