1 Redundanzfreiheit als oberste Regel

Ein sauber definiertes Datenmodell muss neben der korrekten Abbildung der realen Situation vor allem frei von allen Redundanzen sein. Dies bedeutet, dass jede Information auch nur einmal abgelegt werden darf.

~~Blöde~~ Frage:
Es gibt aber doch Datenmodelle, in denen bewusst Redundanzen eingebaut werden, oder?
Antwort: Wir müssen hier zwischen der „reinen Lehre“ und der Praxis unterscheiden. Grundsätzlich ist die Redundanzfreiheit ein Muss. Wenn jedoch aufgrund von äußeren Umständen dies zu Problemen führt, können kontrolliert Redundanzen eingeführt werden. Meist hat dies Performancegründe (im Regelfall werden hierdurch Joins gespart). Im „Business Intelligence“ Bereich (also OLAP) werden diese Redundanzen oftmals massiv eingebaut um möglichst schnell zu Ergebnissen zu kommen. In Transaktionalen Datenbanken ist dies aber möglichst zu vermeiden.

2 Atomares Attribut

Bevor wir uns mit der Redundanz beschäftigen, noch ein wichtiger Einschub. Um den Zugriff auf die Daten möglichst strukturiert zu ermöglichen, müssen wir die Attribute atomar halten. Ein Attribut ist dann atomar, wenn sich in dem Feld (also der Tabellenspalte) immer nur eine Information befindet. Gehen wir davon aus, dass wir eine Tabelle für Kunden einer Autowerkstatt haben, in der neben dem Kennzeichen auch der Fahrzeugbesitzer steht:

Name	**Kennzeichen**
Michael Mayer	A-BC 1234

Sollte sich Herr Mayer aber nun ein zweites Auto kaufen, so könnte man auf die Idee kommen, das zweite Kennzeichen ebenfalls in das Feld „Kennzeichen“ zu schreiben:

Name	**Kennzeichen**
Michael Mayer	A-BC 1234, A-DE 4321

Wir hätten im Datensatz von Herrn Mayer also beide Kennzeichen in dem dafür vorgesehenen Feld abgelegt. Dies würde der atomaren Datenhaltung also wiedersprechen. Die Lösung wäre, pro Kennzeichen einen Datensatz anzulegen, was aber wieder die Redundanzen erhöht (wir hätten den Namen „Michael Mayer“ somit zweimal in der Datenbank).

3 Anomalien

Basierend auf der Forderung der atomaren Datenhaltung müssen wir nun die Redundanzen und deren Konsequenzen analysieren. Neben unnötiger Speicherbelastung führen Redundanzen oft zu Dateninkonsistenzen oder „Anomalien“. Hier unterscheiden wir zwischen:

- Einfüge-Anomalien
- Änderungs-Anomalien
- Lösch-Anomalien

3.1 Einfüge-Anomalie

Von einer Einfüge-Anomalie (oder auch „Insert Anomalie“) spricht man, wenn in bestimmten Situationen beim Einfügen der Daten nicht alle notwendigen Daten abgelegt werden können. Dies soll an der unten stehenden Tabelle veranschaulicht werden. Wie wir sehen, haben wir hier zu viele Daten in eine Tabelle gesteckt – wenn ein Kunde bspw. zwei Fahrzeuge hat, dann müssten wir den Kunden zweimal mit verschiedenen Ids anlegen. Insofern laufen wir in Gefahr Redundanzen zu erzeugen.

Kunde ID	Name	Geb-Dat	Telefon	Kennzeichen	Fabrikat	Farbe
1234	Michael Mayer	23.04.1976	01234567	A-BC 1234	Wartburg	Grün
1235	Eva Huber	24.05.1977	02345678	D-EF 5678	Lada	Grau

Die Einfüge Anomalie würde wiederum entstehen, wenn wir einen Kunden anlegen wollen, aber noch keine genauen Informationen über das Fahrzeug haben. Hierdurch würden wesentliche Informationen der Tabelle leer bleiben:

Kunde ID	Name	Geb-Dat	Telefon	Kennzeichen	Fabrikat	Farbe
1234	Michael Mayer	23.04.1976	01234567	A-BC 1234	Wartburg	Grün
1235	Eva Huber	24.05.1977	02345678	D-EF 5678	Lada	Grau
12346	Paul Schmidt	26.06.1978	03456789			

Schlimmer noch – wenn die Tabelle eigentlich für das Abspeichern von Fahrzeuginformationen gedacht wäre, dann würden Spalten wie bspw. „Kennzeichen“ als Pflichtfelder formuliert werden und wir wären erst dann in der Lage Kundeninformationen abzulegen, wenn wir die Fahrzeugdaten hätten.

3.2 Änderungs-Anomalie

Die Änderungs-Anomalie (oder auch „Update Anomalie“) tritt auf, wenn wir bei der Änderung einer Information mehrere Zeilen einer Tabelle anpassen müssten. Um dies zu verdeutlichen erweitern wir unsere „fehlerhaft“ designte Tabelle von oben um einen weiteren Datensatz:

Kunde ID	Name	Geb-Dat	Telefon	Kennzeichen	Fabrikat	Farbe
1234	Michael Mayer	23.04.1976	01234567	A-BC 1234	Wartburg	Grün
1235	Eva Huber	24.05.1977	02345678	D-EF 5678	Lada	Grau
1237	Eva Huber	24.05.1977	02345678	G-HI 2345	Trabant	Blau

Wir sehen, dass Eva Huber sich ein zweites Fahrzeug gekauft hat. Ihre personenbezogenen Daten liegen somit redundant vor. Wenn sie nun umzieht und somit ihre Telefonnummer sich ändert, so muss diese in der zweiten und dritten Zeile geändert werden. Dies führt zu Inkonsistenzen in der Datenhaltung, was bei großen Tabellen zu unüberschaubaren Konsequenzen führen würde.

3.3 Lösch-Anomalie

Die dritte Anomalie im Bunde ist die Lösch-Anomalie (oder auch „Delete Anomalie“). Hierunter verstehen wir die Situation, dass wir beim Löschen von obsoleten Daten diejenigen mitlöschen müssen, welche wiederum noch benötigt werden. Bemühen wir hierbei wieder unsere Tabelle von oben:

Kunde ID	Name	Geb-Dat	Telefon	Kennzeichen	Fabrikat	Farbe
1234	Michael Mayer	23.04.1976	01234567	A-BC 1234	Wartburg	Grün
1235	Eva Huber	24.05.1977	02345678	D-EF 5678	Lada	Grau

Gehen wir davon aus, dass das Auto von Michael Mayer einen Totalschaden hat – wir also den Datensatz der ersten Zeile löschen. Dadurch verlieren wir aber auch jegliche Information zu dem Kunden Michael Mayer, wodurch er komplett aus dem System verschwindet.

4 Abhängigkeiten

Wie wir bei der Erstellung des ER Modells bereits gelernt haben, gibt es zwischen den einzelnen Informationen „Beziehungen“. Für eine saubere Modellierung müssen diese erstmal Kategorisiert werden. Grundsätzlich unterscheiden wir zwischen:

- Funktionalen Abhängigkeiten
- Transitiven Abhängigkeiten

4.1 Funktionale Abhängigkeit

Eine funktionale Abhängigkeit zwischen zwei Attributen X und Y liegt dann vor, wenn es zu einer Attributsausprägung in X genau eine Attributsausprägung Y gibt. Somit ist Y von X funktional abhängig. Sehen wir uns das auch mal in einem Beispiel an:

Zeile	Attribut X	Attribut Y	Attribut Z
1	1	10	1234
2	2	12	2345
3	2	12	3456
4	3	10	4567

Wie wir in diesem Beispiel sehen, finden wir zur jeden Ausprägung von X immer die gleiche Attributsausprägung Y. Die wichtigen Zeilen sind die Zeilen 2 und 3. Hier hat das Attribut X jeweils den Wert „2“ und passend dazu das Attribut Y immer den Wert „12“. Y ist von X also funktional abhängig. Z ist wiederum nicht funktional von X abhängig, da für die X – Ausprägung „2“ zwei verschiedene Z Werte existieren.

Überschreiben wir die Tabelle nun mal mit „realistischen“ Spaltennamen, um den Zusammenhang besser nachvollziehbar zu machen.

Personalnummer	Arbeitsplatz ID	Raumnummer	Telefonnummer
1	1	10	1234
2	2	12	2345
3	2	12	3456
4	3	10	4567

Wie wir sehen, teilen sich zwei Mitarbeiter den Arbeitsplatz mit der ID 2, der sich im Raum „12“ befindet. Die Mitarbeiter mit der Personalnummer 1 und 4 haben jeweils einen eigenen Arbeitsplatz, wobei beide sich im Raum 10 befinden. Die funktionale Abhängigkeit besteht also darin, dass der Raum funktional vom Arbeitsplatz abhängt. Das ist zwar auf den ersten Blick verwirrend, da wir eigentlich davon ausgehen, dass der Arbeitsplatz vom Raum abhängt, die funktionale Beziehung ist aber genau anders herum.
Man kann es sich so vorstellen, dass wir lediglich die Arbeitsplatz ID benötigen, um den Raum bestimmen zu können. Die Notation einer funktionalen Abhängigkeit wird mit einem Pfeil realisiert:

„Y ist von X funktional abhängig“ wird geschrieben als: **X -> Y**

Bei der Funktionalen Abhängigkeit unterscheidet man noch zwischen „funktional abhängig“ und „voll funktional abhängig“. Um dies zu verdeutlichen, wird die Tabelle nochmals erweitert:

Zeile	Attribut W	Attribut X	Attribut Y	Attribut Z
1	1	1	10	1234
2	1	2	12	2345
3	1	2	12	3456
4	1	3	10	4567

Wir haben nun ein weiteres Attribut (W) eingeführt. Die funktionale Abhängigkeit kann nun wie folgt formuliert werden:

W,X -> Y

Y ist also funktional abhängig von der Kombination aus W und X. Wir können also von der Kombination von W und X Ausprägungen eindeutig auf eine Ausprägung von Y schließen. Entscheidend hier ist und bleibt aber die Ausprägung von X, da diese ausreichend für die Bestimmung von Y ist. Insofern spricht man von einer vollen funktionalen Abhängigkeit:

X => Y

4.2 Transitive Abhängigkeit

Eine weitere Form der Abhängigkeit ist die „transitive Abhängigkeit“. Hier kommt nun ein drittes Attribut ins Spiel. Um das zu verstehen, erweitern wir unsere Beispieltabelle erneut, diesmal um eine Gebäudekennung:

Personalnummer	Arbeitsplatz ID	Raumnummer	Gebäude	Telefonnummer
1	1	10	A	1234
2	2	12	A	2345
3	2	12	A	3456
4	3	10	A	4567
5	10	110	B	5678

Gehen wir davon aus, dass alle Raumnummern über 99 zum Gebäude B gehören. Wir haben nun eine Abhängigkeit von Raumnummer zu Arbeitsplatz ID und eine Abhängigkeit von Gebäude zu Raumnummer. Dadurch ergibt sich die transitive Abhängigkeit von Gebäude zur Arbeitsplatz ID.

4.3 Schlüssel

Grundsätzlich ist ein Schlüssel eine Menge von 1 bis mehreren Attributen, welche einen Datensatz („Tupel“) in einer Tabelle eindeutig identifizieren. Basierend auf den Abhängigkeiten können wir nun die Schlüssel suchen. Hierbei unterscheidet man zwischen drei verschiedenen Schlüsselbegriffen:

Begriff	Bedeutung	Ist Teilmenge von:
Superschlüssel	Alle (beliebigen) Kombinationen von Attributen, welche ein Tupel eindeutig identifizieren. Im einfachsten Fall alle Attribute zusammen.	-
Schlüsselkandidat	Die Menge der minimalen Schlüssel, welche jedes Tupel eindeutig identifizieren.	Superschlüssel
Primärschlüssel	Der Schlüssel, der zur eindeutigen Identifizierung der Tupel verwendet wird.	Schlüsselkandidaten

Anmerkung: In der technischen Realisierung wird im Regelfall ein künstlicher Schlüssel verwendet, um bei Fremdschlüsseln keine fachlichen Informationen redundant zu halten.

5 Normalformen

Um nun von einem Konzept in ein Datenmodell zu gelangen, müssen die Normalformen bestimmt werden. Hierzu wenden wir unser Wissen aus den vorausgegangenen Kapiteln anwenden. Beginnen wir mit der ersten Normalform.

5.1 Erste Normalform (1NF)

Die erste Normalform ist dann gegeben, wenn alle Attributsausprägungen atomar vorliegen, also in jeder Spalte immer nur eine Information vorliegt. Sehen wir uns hierfür zuerst eine Tabelle an, welche nicht in der 1NF vorliegt:

RechnungID	Artikelname	Artikelnummer	KundeID	Name
1	USB Stick	12345	24332	Michael Mayer
2	CD Box	34221	23432	Eva Müller

Im Feld „Name“ befindet sich Vor- und Nachname. Hier können wir nun eine Anpassung vornehmen:

RechnungID	Artikelname	Artikelnummer	KundeID	Nachname	Vorname
1	USB Stick	12345	24332	Mayer	Michael
2	CD Box	34221	23432	Müller	Eva

Nun liegen die beiden Informationen Vor- und Nachname atomar vor. Die Zerlegung von Daten in mehrere Tabellen nennt man „***Decomposition***“.

5.2 Zweite Normalform (2NF)

Bei der zweiten Normalform muss das Datenmodell bereits in der 1NF vorliegen und jedes Nichtschlüsselattribut von jedem Schlüsselkandidaten voll funktional abhängig sein. Sehen wir uns dies wiederum an dem Beispiel der vorausgegangenen Tabelle an, welche ja in der 1NF vorliegt.

Die Kundeninformationen (letzten drei Spalten) haben keine voll funktionale Abhängigkeit zu der RechnungID, was ohnehin fachlich nachvollziehbar sein sollte. Es ist also sinnvoll, die Kundendaten und die Rechnungsdaten zu trennen:

RechnungID	Artikelname	Artikelnummer
1	USB Stick	12345
2	CD Box	34221

KundeID	Nachname	Vorname
24332	Mayer	Michael
23432	Müller	Eva

Nun sind die Informationen getrennt worden und es existieren nur noch funktionale und transitive Abhängigkeiten.

5.3 Dritte Normalform (3NF)

Die (für uns) finale Normalform stellt die dritte Normalform dar. Hier gehen wir von der 2NF wiederum aus und entfernen alle transitiven Abhängigkeiten. Eine solche finden wir in der Tabelle mit den Rechnungsinformationen aus dem vorausgegangen Kapitel. Hier haben wir eine transitive Abhängigkeit des Artikelnamens über die Artikelnummer zur RechnungID. Diese können wir wiederum durch eine Aufteilung der Daten in zwei Tabellen eliminieren:

RechnungID	Artikelnummer
1	12345
2	34221

Artikelnummer	Artikelname
12345	USB Stick
34221	CD Box

6 Die Definition der Tabellen „from scratch[1]“

Soweit die Theorie! Grundsätzlich können wir aber davon ausgehen, dass wir ohne diese dedizierten Überlegungen die Datenbank ohnehin intuitiv richtig designen. Niemand würde auf die Idee kommen, Kundendaten und Rechnungsdaten in eine Entitätsklasse zu packen. Um nun schematisch, aber einfach vorzugehen, können wir uns ein kleines Regelwerk überlegen:

6.1 Entitätsklassen

Wir überlegen uns alle notwendigen Entitätsklassen. Eine Entitätsklasse klassifiziert etwas, was wir in unserer Datenbank abbilden wollen. Hierbei gilt:

Alles, was gemeinsame Eigenschaftsmerkmale aufweist, sollte in einer eigenen Entitätsklasse abgebildet werden.

6.2 Verteilung der Informationen

Jede (atomare) Information die in dem Datenmodell benötigt wird, muss eindeutig einer Entitätsklasse zugeordnet werden können. Ist dies nicht der Fall, so muss für diese Information eine neue Entitätsklasse geschaffen werden.

Kann eine Information nicht einer Entitätsklasse zugeordnet werden, so wurde sehr wahrscheinlich eine Entitätsklasse vergessen.

[1] „from scratch“ = von Grund auf

6.3 Beziehungen und Kardinalitäten

Die Beziehungen und die Kardinalitäten werden basierend auf den Anwendungsfall formuliert. Es ist durchaus möglich, dass die reale Welt „anders tickt“, als unser Anwendungsfall (Beispiel: in der realen Welt können pro Adresse mehrere Kunden wohnen, wir modellieren im Regelfall pro Adresse immer nur einen Kunden). Es sollten stets „sprechende“ Namen für die Beziehungen gewählt werden um die Kardinalität einfach feststellen zu können. Hierbei wird jededer beiden Kardinalitätszahlen getrennt bestimmt:

Man frägt aus Sicht der einen Entität die mögliche Anzahl der anderen Entität einer Beziehung ab.

Beispiel:
„Wie viele Bestellungen können von einem einzelnen Kunden erstellt werden?“
und
„Wie viele Kunden haben eine einzelne Bestellung erstellt?“

6.4 Tabellen aus Entitäten

Ab jetzt wird das physikalische Datenmodell erstellt. Hierbei gilt:

Jede Entitätsklasse wird zu einer eigenen Tabelle.

6.5 Attribute

Die atomaren Informationen werden nun ebenfalls zugeordnet:

Jedes Attribut wird zu einer eigenen Tabellenspalte.

6.6 Abbildung der Beziehungen

Bei den Beziehungen kennen wir die drei grundlegenden Typen:
1:1 Beziehung
1:n Beziehung
n:m Beziehung

Diese werden unterschiedlich modelliert. Beginnen wir mit der 1:1 Beziehung:

6.6.1 1:1 Beziehung

Hier gilt, dass wir rein technisch zwar für beide Entitäten zusammen nur eine Tabelle benötigen würden, dies aber aufgrund von folgenden Gründen meist nicht zusammengeführt wird:

- Verletzung der 3NF (transiente Abhängigkeiten werden nicht aufgelöst)
- Nicht für jedes Tupel in Tabelle 1 wird ein Tupel in Tabelle 2 benötigt und somit kann durch die Auslagerung in eine eigene Entität Speicher gespart werden.
- In Tabelle 2 liegen sensitive Daten, welche über Zugriffsmechanismen geschützt werden müssen.
- Das Datenmodell sollte übersichtlich gestaltet werden.

Grundsätzlich wird eine 1:1 Beziehung wie folgt umgesetzt:

Der Primärschlüssel von Tabelle 1 wird als Primärschlüssel der Tabelle 2 genutzt (oder alternativ als eindeutiger Fremdschlüssel, sofern Tabelle 2 einen eigenen Primärschlüssel hat).

In der Chen Notation werden eindeutige Attribute (also Attribute, in denen pro Entitätsklasse eine Ausprägung nur einmal vorkommen darf) doppelt unterstrichen.

6.6.2 1:n Beziehung

Die 1:n Beziehung ist die einzige „asymmetrische“ im Bunde. Wenn die Tabellen 1 und 2 eine 1:n Beziehung aufweisen, so kann es zu einem Tupel aus Tabelle 1 mehrere Tupel aus Tabelle 2 geben, nicht aber umgekehrt. Dies führt zu folgender Regel:

Der Primärschlüssel von Tabelle 1 ist ein Fremdschlüssel in Tabelle 2.

6.6.3 n:m Beziehung

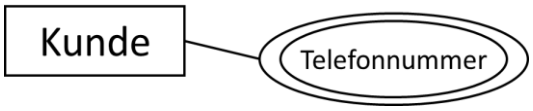
Zum Schluss kommen die n:m Beziehungen. In relationalen Datenbanken können zwischen zwei Tabellen keine n:m Beziehungen abgebildet werden. Man benötigt hierfür eine Zwischentabelle (engl. „intetsection table“). Hierbei gilt:

Haben die Tabellen 1 und 2 eine n:m Beziehung, entsteht eine neue (Zwischen-) Tabelle. Diese hat die Primärschlüssel der Tabellen 1 und 2 als Fremdschlüsselattribute.

Technisch werden in der Zwischentabelle die beiden Fremdschlüssel aus Tabelle 1 und 2 entweder als Primärschlüssel oder als eindeutige Attribute modelliert.

6.7 Mehrwertige Attribute

Ein mehrwertiges Attribut ist ein Attribut mit einer Menge an abzählbaren Werten mit unbekannter Anzahl. Beispiele hierfür wären Telefonnummern (also wenn ein Kunde mehrere Kontaktnummern angeben kann), Autorennamen eines Buches, eMail Adressen eines Kunden etc. In der Chen Notation werden solche mehrwertigen Attribute in einem doppelten Oval notiert.

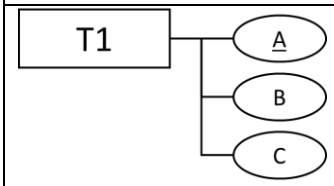
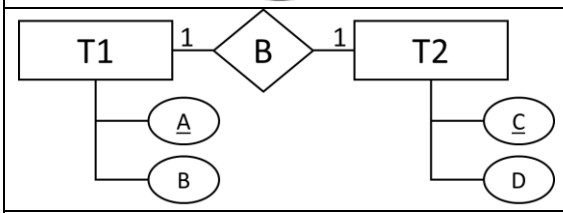
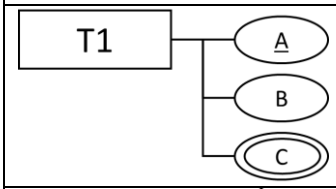
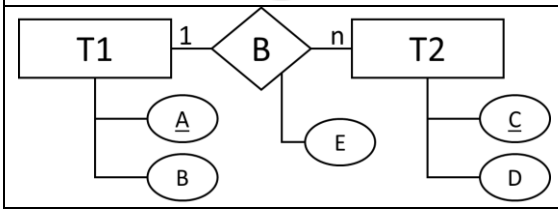


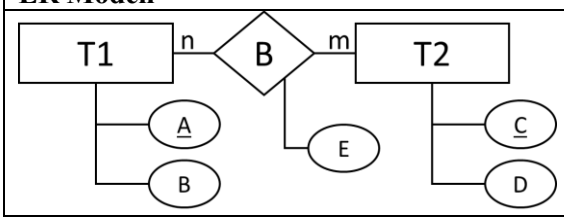
Im physikalischen Datenmodell gilt:

Mehrwertige Attribute werden als eigenständige Tabellen abgebildet.

7 Relationsschema

Das Relationsschema zeigt das physikalische Datenmodell in einer Kurzschreibweise an. Hierbei werden die ER Modelle wie folgt umgesetzt:

ER Modell	Bemerkung	Schema
	Die Notation beginnt mit dem Tabellennamen, gefolgt von den Attributen. Die Primärschlüssel werden unterstrichen dargestellt.	T1{A, B, C}
	1:1 Beziehungen werden über eine Schlüsselübernahme modelliert. Entweder wandert der Schlüssel von T2 als eindeutiges Attribut nach T1 oder der Schlüssel von T2 wird eindeutiges Attribut in T1.	T1{A, B} T2{C, D, A*(FK)*} oder T1{A, B, C*(FK)*} T2{C, D}
	Mehrwertige Attribute werden in eigenen Tabellen ausgelagert.	T1{A, B} C {C, A*(FK)*}
	Bein 1:n Beziehungen wird der Schlüssel der 1 – Tabelle als Fremdschlüssel der n – Tabelle verwendet. Eventuelle Attribute der Beziehung werden der ***n*** – Tabelle zugeordnet.	T1{A, B} T2{C, D, A*(FK)*, E}

ER Modell	Bemerkung	Schema
	n:m Beziehungen müssen in einer eigenen Tabelle abgelegt werden. Hierzu werden die Schlüssel der beiden Entitätsklassen übernommen. Eventuelle Attribute der Beziehung werden hier ebenfalls übernommen.	T1{<u>A</u>,B} B{A*(FK)*,C*(FK)*,E} T2{<u>C</u>,D}