Datenbankdesign Grundlagen

https://www.datenbanken-verstehen.de/

Damit eine **erfolgreiche Datenbankentwicklung** durchgeführt werden kann, müssen alle kritischen Faktoren vorher in einem sauberen **Datenbankdesign** geklärt werden.

Im Bereich **Datenbankdesign Grundlagen** werden euch die Grundlagen gezeigt wie man ein **professionelles Datenbankdesign aufbauen kann**.

Ein **gutes Datenbankdesign** zeichnet sich durch eine saubere Trennung der zwei Schichten (Datenbankschicht/Anwendungsschicht) aus.

Oft wird dabei zwischen dem **klassischen Datenbankdesign** auf der einen Seite und dem **Anwendungsdesign** auf der anderen Seite unterschieden.

Beide Komponenten müssen nahtlos ineinandergreifen, damit eine **Datenbankapplikation erfolgreich konzipiert und implementiert werden** kann.

Lebensstadien einer Datenbankanwendung

Damit ein **Datenbankprojekt erfolgreich erstellt** werden kann, werden die einzelnen **Lebensstadien einer Datenbankanwendung** festgelegt, um eine erfolgreiche Datenbankentwicklung zu gewährleisten.

Die **Lebensstadien einer Datenbankanwendung** beziehen sich nicht nur auf das klassische Datenbankdesign. **Es beinhaltet auch das Anwendungsdesign**, welches später auf das Datenbankdesign aufgesetzt wird.

Somit wird der **gesamte Ablauf einer Datenbankentwicklung festgelegt**: Von der Planung der Datenbank bis zu Datenerfassung und dem sogenannten „Rollout“ einer Datenbankanwendung.

Erstellung eines neuen Datenbankprojekts

Bei der **Erstellung eines neuen Datenbankprojekts** muss folgende Checkliste vorab geführt werden, damit ein **Datenbankprojekt erfolgreich durchgeführt** werden kann:

- Planung der Datenbank
- Definition von Benutzergruppen
- Sammlung und Analyse der Daten
- Datenbank Design
- Auswahl eines geeigneten DBMS
- Anwendungsdesign
- Erstellung eines Prototyps
- Test des Prototyps und eventuelle Korrekturmaßnahmen
- Implementierung der gesamten Datenbank
- Datenerfassung, Test und Korrekturen

Datenbankdesign im Detail

In der **Anforderungsanalyse des Datenbankdesigns** wird festgelegt, was die Datenbank jetzt und in Zukunft leisten soll. Das **zentrale Problem der Anforderungsanalyse** ist, dass der Entwickler einer Datenbank alle Informationen sammeln, analysieren und in den richtigen Kontext bringen muss.

„Was macht ein gutes Datenbankdesign aus?“

Ein **guten Datenbankdesign** zeichnet sich dadurch aus, dass der **Datenbankentwickler ein wartbares und leicht erweiterbares Datenbankmodell erstellt**.

Um dieses Ziel zu realisieren, ist es wichtig, dass der Datenbankentwickler die einzelnen **Designschichten einer Datenbank** kennt und auch optimal ausnutzt.

Folgende Fragen muss der **Datenbankentwickler in der Anforderungsanalyse klären**, um einen ersten Rahmen zu schaffen:

- Was sind meine Objekte und deren Attribute?
- Wie sehen die Beziehungen zwischen den Objekten aus?
- Wie viele Objekte werden in der Datenbank benötigt?
- Was sind meine typischen Operationen?
- Laufzeit und Bedeutung dieser Operationen?

Dieser **Fragen sind kein Patentrezept**, da jedes Projekt unterschiedlich ist und immer wieder auf das Projekt neu zugeschnitten werden muss.

Konzeptionelles Datenbankdesign

Im **Konzeptionellen Datenbankdesign** werden die Datenbankstrukturen in einer formalen Sprache, auf Basis eines konzeptionellen Datenmodells mit hohem Abstraktionsgrad, beschrieben.

Das bekannteste konzeptionelle Designmodell ist das Entity-Relationship Modell (ER-Modell). Die Daten einer Datenbank werden im **konzeptionellen Datenbankdesign** nicht berücksichtigt.

Es wird nur das Schema der Datenbank festgehalten, um eine Grundlage für eine Erstellung einer Datenbank zu gewährleisten.

Besonders kritisch wird es, wenn die Anforderungsanalyse in ein konzeptionelles Datenbankdesign umgewandelt werden muss. Viele Entwickler und Benutzer verwenden verschiedene Bezeichnungen für den gleichen Objekttyp.

Des Weiteren verwenden viele Benutzer den gleichen Bezeichner für verschiedene Objekttypen. Der Entwickler muss auch dafür sorgen, unnötige Strukturen zu entfernen, um die Abstraktion der Objekte zu gewährleisten.

Konzeptionelles Datenbankdesign Definition

Das **konzeptionelle Datenbankdesign** bzw. Datenmodell ist der erste Schritt für die Erstellung einer Datenbank zur Verwendung in Anwendungen. Der konzeptionelle Entwurf basiert auf einer Analyse der Realwelt-Umgebung, in der die Anwendung später eingesetzt werden soll.

In dieser wurden relevante (Geschäfts-)Objekte und (Geschäfts-)Prozesse und deren Beziehung untereinander sowie der eigentliche Zweck der Anwendung und der Datenbank ermittelt. Mit dem konzeptionellen Modell werden diese Ergebnisse festgehalten.

Bestandteile eines konzeptionellen Datenmodells

Das Modell wird mit Hilfe grafischer und textueller Notationen (ER-Diagramme) entwickelt und enthält die Objekte und Objektbeziehungen. Es werden folgende Sachverhalte dargestellt:

- (Geschäfts-)Objekte -> Realweltobjekte z. B. Bestellungen, Kunden
- Attribute von Objekten -> z. B. Bestell- und Kundennummer
- Eindeutiger Schlüssel -> Für die Identifikation eines Objektes
- Beziehungen -> z. B. zwischen Kunde und Bestellung
- Beziehungstypen -> z. B. 1:1, 1:N, N:M
- Generalisierung -> hierarchische Aufteilung in Ober- und Unterklassen z. B. Kunde unterteilt in Neukunde, Stammkunde

Konzeptionelles Datenmodell Beispiel

Im Folgenden wird ein ER-Diagramm zur Darstellung eines einfachen Sachverhaltes erstellt. Angenommen wird, dass eine Analyse ergeben hat, dass die Vertriebsabteilung einer Firma eine neue Software zur Verwaltung von Kundenverträgen einführen möchte. Hierfür muss zunächst analysiert werden, welche Geschäftsobjekte und -prozesse in der Abteilung existieren und wie diese in Beziehung stehen.

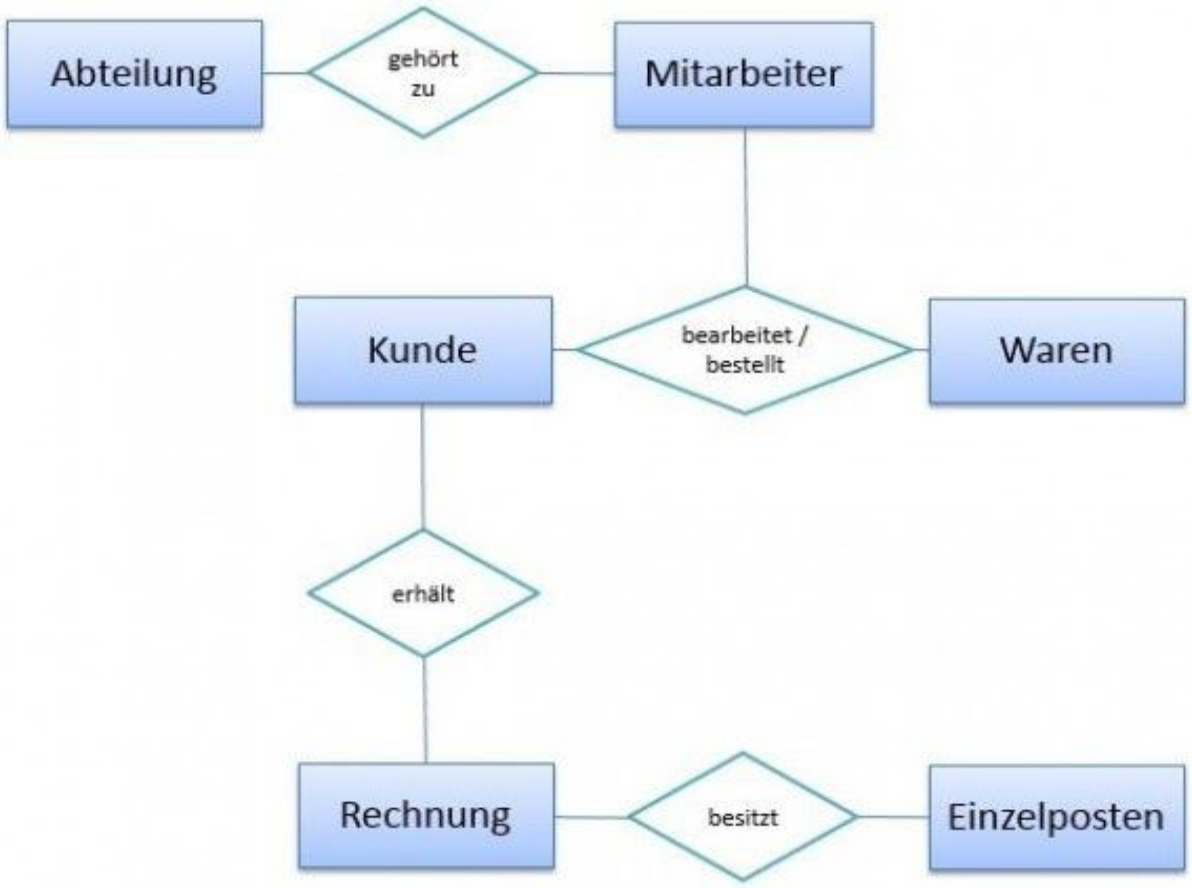
Die Analyse ergibt folgende Objekte:

- Waren
- Kunden
- Rechnung
- Rechnungsposten
- Mitarbeiter
- Abteilung

Es bestehen folgende Beziehungen zwischen den Objekten:

- Kunde bestellt Waren
- Kunde erhält Rechnung
- Rechnung besitzt Einzelposten
- Mitarbeiter gehört zu einer Abteilung
- Mitarbeiter bearbeitet Bestellungen

Nach der Analyse sieht das **konzeptionelle Datenmodell** wie folgt aus:



Das hier entstandene **konzeptionelle Datenbankmodell** ist die **Grundlage für die Entwicklung eines logischen Datenbankdesigns**.

Logisches Datenbankdesign

Im **logischen Datenbankdesign** wird die Umwandlung des konzeptionellen Designs auf die Datenstrukturen der Datenbank angepasst. Ein wichtiges Ziel im logischen Datenbankdesign ist die **Eliminierung von Redundanzen und die einmalige Speicherung der Daten vorzubereiten**.

Das **logische Datenbankdesign** stellt den zu implementierenden **Entwurf der Datenbank** dar. Ausgangsbasis des logischen Entwurfs ist das **konzeptionelle Datenbankdesign**, das die fachlichen Anforderungen eines Realwelt-Prozesses erfasst.

Mit dem logischen Modell wird eine Beschreibung erstellt, die vom letztendlich eingesetzten relationalen **Datenbank-Management-System** (**RDBMS**) unabhängig ist. Es stellt „nur“ die logischen Tabellenstrukturen und Beziehungen zwischen den Tabellen dar.

Bestandteile eines logischen Datenbankmodells

Das in der konzeptionellen Phase entwickelte **ER-Diagramm** wird in eine Tabellenform überführt. Jede Entität, also jedes Objekt, des **ER-Modells** wird zunächst zu einer Tabelle zugeordnet. Um die Beziehungen zwischen den Entitäten bzw. den Tabellen abzubilden, werden **Primär-Fremdschlüssel-Beziehungen** in Form zusätzlicher Attribute in die jeweilige Tabelle eingefügt. Es werden folgende Sachverhalte festgelegt:

- Tabellen -> deren Bezeichnung und Aufbau
- Primär- und Fremdschlüssel / Referenzielle Integrität
- Zusätzliche Attribute
- Integritätsbedingungen
- Konsistenzbedingungen

Überführung des konzeptionellen Modells in ein Relationenmodell – Beispiel

Das **ER-Diagramm** wird mit Hilfe des logischen Datenbankdesigns in ein **Relationenmodell** überführt. Um diesen Übergang zu beschreiben, wird das Beispiel aus dem konzeptionellen Datenbankdesign wieder aufgegriffen.

FIR_PER	
PERNR	Personalschlüssel
FIRNR	Firmenschlüssel
WERNR	Werksschlüssel
ABTNR	Abteilungsschlüssel
PERUSER	Personalbezeichnung
AKTIV	Aktiv
TITEL	Titel
GENUS	Geschlecht
VORNAME	Vorname
NAME	Nachname
STRASSE	Straße
PLZ	Postleitzahl
ORT	Ort
LAND	Land
TELEFON	Telefon
FAX	Fax
MOBIL	Mobiltelefon
EMAIL	E-Mail Adresse
BEM	Bemerkung
GEBDATUM	Geburtsdatum
EINTRITT	Eintrittsdatum
AUSTRITT	Austrittsdatum

FIR_ABT	
ABTNR	Abteilungsschlüssel
FIRNR	Firmenschlüssel
ABTUSER	Abteilungsbezeichnung

Das **Relationenmodell** wird folgendermaßen dargestellt:

- **Tabelle Personal** für eine Firma – FIR_PER (PERNR, FIRNR, WERNR, ABTNR, PERUSER, AKTIV, TITEL, GENUS, VORNAME, NAME, STRASSE, PLZ, ORT, LAND, TELEFON, FAX, MOBIL, EMAIL, BEM, GEBDATUM, EINTRITT, AUSTRITT)
- **Tabelle Abteilung** für eine Firma – FIR_ABT (ABTNR, FIRNR, ABTUSER)

In diesem Fall gibt es zwei Beziehungsrichtungen. **Ein Mitarbeiter ist einer Abteilung zugeordnet und enthält die Referenz auf die Abteilungsnummer.** Zudem enthält die Relation Abteilung die Information, wer die Abteilung leitet. Diese wird aus der Mitarbeitertabelle referenziert, wodurch die **Relation Abteilung einen Fremdschlüssel der Mitarbeitertabelle enthält**.

Im **Relationenschema** muss darauf geachtet werden, dass die **Normalisierung** eingehalten wird, sonst könnte es zu **Anomalien und fehlender Integrität** kommen. **Welche Normalisierungsstuf**e angewendet werden muss, **hängt stark vom Zweck der Datenbank ab** und muss bereits in der Analyse des Einsatzzwecks ermittelt und festgelegt werden.

Mit der Erstellung des logischen Datenbankmodells wird die Grundlage für die Implementierung auf einem Ziel-DBMS geschaffen, dieses wird über das **physische Datenbankdesign** beschrieben.

Physisches Datenbankdesign

Das **Physische Datenbankdesign** basiert auf der Grundlage des logischen Datenbankdesigns. Hier ist besonders zu beachten, **welche Indexstrukturen entworfen werden müssen**, um effiziente **SQL-Anfragen zu gewährleisten**.

Werden zum Beispiel zu viele Indizes erstellt, werden SQL UPDATES-Statements auf eine Datenbank sehr teuer, da die Speicherung länger dauert und die Indizes neu berechnet werden müssen.

Werden zu wenige Indizes in einer Datenbank gesetzt, so werden die Suchoperationen nicht effizient unterstützt und es kommt zu längeren Suchanfragen.

Physisches Datenbankdesign Definition

Das im Schritt des **logischen Datenbankdesigns** erstellte **Relationenmodell** wird im physischen Datenbankdesign auf ein konkretes relationales Datenbank-Management-System (RDBMS) angewandt und implementiert.

Für die Anlage von Tabellen und die spätere Speicherung von Daten, verwendet man eine **Datenbanksprache**. Bei den etablierten **relationalen Datenbanksystemen** basierte diese fast ausschließlich auf **SQL**. Wobei die Datenbankhersteller meist einen Zusatz entwickelt haben, auch als SQL-Dialekte bekannt, um Elemente der Programmierung zu integrieren (z. B. PL/SQL von Oracle, T-SQL von Microsoft).

Bestandteile eines physischen Datenbankdesigns

Das **physische Datenbankdesign** beschreibt die Art und Weise wie Tabellen und Daten strukturiert, verwaltet und zugreifbar gemacht werden. Bei der **Überführung des logischen in das physische Modell** werden zunächst die Datentypen aller Attribute festgelegt und anschließend das Datenbankskript erstellt.

Datentypen beschreiben welche Attribute einer Tabelle welche physische Repräsentation auf dem Speichermedium besitzt. Ein **Datentyp stellt eine Struktur dar**, die in einer bestimmten Art und Weise gespeichert wird. Gängige Datentypen sind:

Datentyp	Eigenschaften	Verwendung
Integer	Ganzzahlen	Primärschlüssel
Float	Fließkommazahlen	Gehalt
Datum	Berechnungen von Datumsangaben	Kaufdatum
Char	Zeichenketten	Beschreibung
Boolean	Wahr-/Falsch-Werte	Markierung einer Gültigkeit

Nach der Festlegung der Datentypen werden die Tabellen angelegt und deren Beziehungen hinterlegt. Zudem werden die **Integritätsbedingungen**, die im logischen Modell festgelegt wurden, mittels Datenbanksprachen (SQL) angelegt.

Überführung eines Relationenmodells in ein SQL-Skript

Für die **Umsetzung des physischen Datenbankmodells wird das Relationenmodell verwendet**. Die Attribute, Bezeichnungen und Relationen werden in SQL-Syntax überführt.

Tabelle Personal für eine Firma – FIR_PER

FIR_PER (PERNR, FIRNR, WERNR, ABTNR, PERUSER, AKTIV, TITEL, GENUS, VORNAME, NAME, STRASSE, PLZ, ORT, LAND, TELEFON, FAX, MOBIL, EMAIL, BEM, GEBDATUM, EINTRITT, AUSTRITT)

Das **Relationenschema wird in das folgende SQL-Statement** zur Erstellung der Tabelle Personal für eine Firma – FIR_PER **umgewandelt**:

```
1. CREATE TABLE FIR_PER (
2. PERNR int not null,
3. FIRNR int not null,
4. WERNR int not null,
5. ABTNR int not null,
6. PERUSER varchar(20) not null,
7. AKTIV tinyint not null,
8. TITEL varchar(30),
9. GENUS tinyint not null,
10. VORNAME varchar(40),
11. NAME varchar(60),
12. STRASSE varchar(60),
13. PLZ varchar(15),
14. ORT varchar(50),
15. LAND varchar(50),
16. TELEFON varchar(30),
17. FAX varchar(30),
18. MOBIL varchar(30),
19. EMAIL varchar(30),
20. BEM varchar(100),
21. GEBDATUM date not null,
22. EINTRITT date not null,
23. AUSTRITT date
24. );
```

Tabelle Abteilung für eine Firma – FIR_ABT

FIR_ABT (ABTNR, FIRNR, ABTUSER)

Das **Relationenschema wird in das folgende SQL-Statement** zur Erstellung der Tabelle Abteilung für eine Firma – FIR_ABT **umgewandelt**:

```
1. CREATE TABLE FIR_ABT (
2. ABTNR int not null,
3. FIRNR int not null,
4. ABTUSER varchar(50) not null
5. );
```

SQL-Statement zur Ergänzung von Fremdschlüsseln

Da beide Tabellen nicht gleichzeitig, sondern sequentiell erstellt werden, müssen die **Fremdschlüssel separat mittels ALTER TABLE** erstellt werden. Sie stehen in direkter Abhängigkeit zur Tabelle für eine Firma – FIRMA und zur Tabelle für ein Werk – _WERK.

Beim **physischen Datenbankdesign** können neben strukturellen Informationen auch **Zugriffsstrukturen** angelegt werden. Beispiele dafür sind **Indizes und Partitionen**, die einen schnelleren Datenzugriff bei der Abfrage ermöglichen

Datenmodellierung

Im Bereich **Datenmodellierung** lernt ihr die **Grundlagen**, um komplexe Geschäftsmodelle in einem **Entity Relationship Modell abzubilden**. Um erfolgreich **Daten** zu **modellieren**, ist es besonders wichtig, die Grundlagen und Zusammenhänge zwischen den einzelnen Themen zu verstehen und anhand von Beispielen nachzubilden.

„Was macht eine gute Datenmodellierung aus?“

Viele Datenbankentwickler sprechen von einer guten Datenmodellierung, wenn folgende Punkte erreicht sind: **Redundanzfreie Datenspeicherung** und **hohe Datenkonsistenz**.

Eine **redundanzfreie Datenspeicherung** liegt dann vor, wenn jede Information in einer Datenbank genau einmal vorkommt. Des Weiteren muss eine **hohe Datenkonsistenz** verfolgt werden, so dass Daten eindeutige Informationen darstellen.

Die **Datenmodellierung** verwendet eine Menge neuer Grundbegriffe, welche vor allem im Zusammenhang mit dem Thema Datenbanken verwendet werden. Um sich mit einem echten Datenbankspezialisten unterhalten zu können, solltet man die **nachfolgenden Kapitel sorgfältig durcharbeiten**.

Grundbegriffe der Datenmodellierung

Die **Grundbegriffe der Datenmodellierung** lassen sich aufteilen in **Begriffe der Datenbankgrundlagen** und der **Informatik**.

Um die **Begriffserklärungen** noch näher an das Thema Datenbanken zu legen, werden zuerst die **Bezeichnungen im konzeptionellen Datenbankdesign** und dann die **Bezeichnungen in Klammen** für das physische Datenbankdesign aufgeführt.

Besonders bei der Überführung vom **konzeptionellen Datenbankdesign**, über das logische Datenbankdesign, in das **physische Datenbankdesign** kann es immer wieder zu Problemen

kommen. Daher werden **Methoden** wie **Bottom-up** oder **Top-down** verwendet, um diese **Probleme zu beseitigen**.

Datenmodellierung – Grundlagen und Begriffe

Entität (Tabellenname): Eine Entität stellt ein Objekt eines Themenkreis dar, welches Elemente mit gleichen Merkmalen beinhaltet. Beispiele für mögliche Entitäten sind: Firma, Student, Kurs und Professor.

Entitätsmenge (Alle Datensätze einer Entität): Eine Entitätsmenge repräsentiert alle Datensätze, die zu einer Entität gehören.

Relation (Tabelle): Eine Relation umfasst eine Entität inklusive der dazugehörigen Entitätsmenge. Eine komplette Relation besteht aus einer Entitätsbezeichnung, deren Attributen und Tupeln.

Wichtiger Hinweis zum Begriff Relation und Tabelle:

Heutzutage werden die Begriffe Relation und Tabelle gleichgesetzt. Das ist leider sehr problematisch, da das Wort „Relation“ im Englischen „Beziehung“ bedeutet. Eine Beziehung hat aber in den Datenbank-Grundlagen eine ganz andere Bedeutung und sagt aus, in welcher Abhängigkeit zwei Tabellen zueinander stehen.

Tupel (Datensatz): Ein Tupel repräsentiert alle Merkmalswerte einer Entität einer Entitätsmenge. Alle Tupel einer Entität bilden die Entitätsmenge.

Attribut (Spaltenname): Ein Attribut beschreibt genau ein Merkmal eines Tupels in einer Entitätsmenge, zum Beispiel den Namen eines Kunden.

Attributwert (Wert): Der Attributwert repräsentiert den Wert, den ein Attribut in einem Tupel annehmen kann, zum Beispiel: Attribut => Name = „Meier“ <= Attributwert

Entity-Relationship-Modell (ER-Modell / ERM)

Das **Entity-Relationship-Modell** – abgekürzt mit ER-Modell oder **ERM** – dient als Grundlage für einen Datenbankentwurf. Bevor mittels SQL angefangen wird, Tabellen und Beziehungen anzulegen, wird erst mal mittels ER-Modell geplant, wie die Datenbankstruktur aufgebaut und funktionieren soll.

„Entity-Relationship-Modell erstellen – aber warum?“

Der Einsatz von **ER-Modellen** ist in der Praxis ein gängiger **Standard für die Datenmodellierung**, auch wenn es **unterschiedliche grafische Darstellungsformen** von Datenbankmodellen gibt.

Mithilfe des **Entity Relationship Modells** soll eine Typisierung von **Objekten, ihrer relationalen Beziehungen** untereinander und der zu überführenden Attribute, stattfinden.

Gute Datenbankentwickler können in kurzer Zeit sehr komplexe **Entity-Relationship-Modelle** verstehen und umsetzen.

Entitäten, Attribute & Beziehungen – Entity-Relationship-Modell

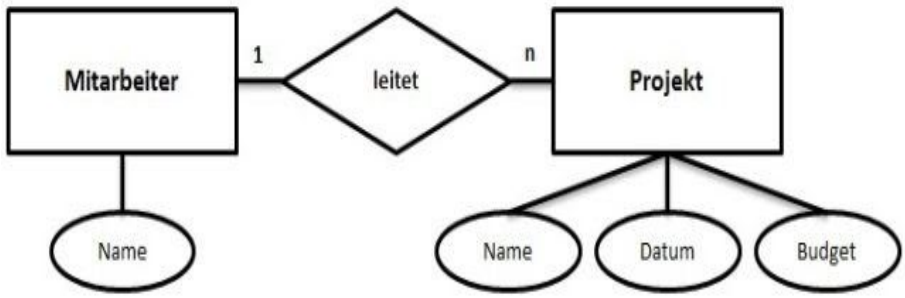
Die **Grundelemente eines jeden Entity-Relationship-Modells** bilden: Entitäten, Beziehungen und Attribute. Diese werden grafisch folgendermaßen dargestellt:

Um was genau es sich bei diesen Elementen handelt, klären die folgenden Punkte:

- Eine Entität ist ein individuell identifizierbares Objekt der Wirklichkeit.
- Eine Beziehung ist eine Verknüpfung / Zusammenhang zwischen zwei oder mehreren Entitäten.
- Ein Attribut ist eine Eigenschaft, die im Kontext zu einer Entität steht.

Entity-Relationship-Modell Beispiel

Das folgende **Beispiel eines Entity-Relationship-Modells** soll zeigen, wie leicht ein Modell anhand eines realen Beispiels zu erstellen ist:



Erklärung zum ER-Modell: Ein Mitarbeiter hat einen Namen. Ein Projekt hat einen Namen, ein Datum und ein Budget. Ein Mitarbeiter **kann mehrere** Projekte leiten, aber nur ein Projekt **kann von genau einem** Mitarbeiter geleitet werden. Diese **Notation** nennt man **Chen-Notation** und ist ein gängiger Standard in der **Praxis der Datenmodellierung**. Diese Notation beinhaltet die **Kardinalität**, die näher im Kapitel Beziehungen in Datenbanken behandelt wird.

Entitäten in einer Datenbank

In einer **Datenbank ist eine Entität** ein **konkretes Objekt bzw. ein konkreter Sachverhalt,** der sich eindeutig von anderen Entitäten des gleichen Entitätstyps unterscheidet.

Ein **Entität-Typ** beschreibt die **Ausprägungen eines Objektes** oder Sachverhaltes durch die Angabe von Attributen.

Er gibt demnach an, welche Eigenschaften eine konkrete Entität aufweist. Übertragen auf eine Datenbank ist eine Entität ein Tupel (Datensatz) einer Relation (Tabelle).

Die **Relation** stellt den **Entität-Typ** dar und deren **Spalten die Attribute**. Eine Eigenschaft entspricht dem konkreten Attributwert.

Die **einzelnen Entitäten** werden in den **unterschiedlichen Tabellen erfasst**, sodass diese einen für den Anwender definierten Ausschnitt aus der realen Welt darstellen.

Die Tabelle „FIRMA“ enthält demnach Entitäten (= Tupel bzw. Datensätze) mit den Attributen Name und weiteren Angaben, die wiederum unterschiedliche Attributwerte annehmen.

Attribute in einer Entität

Jede Entität besitzt eine **bestimmbare Anzahl an Attributen** (Ausprägungen bzw. Eigenschaften), die sich eindeutig von anderen Entitäten des gleichen Entitätstyps abgrenzen. Eine Eigenschaft ist ein konkreter Attributwert, den ein zuvor definiertes Attribut annehmen kann. Die Attribute stellen einen „Bauplan“ dar, der eine abstrakte Abbildung der Wirklichkeit ist.

„Welche Arten von Attributen gibt es?“

Die **Attribute in einer Entität** können unterschiedlich aufgebaut sein. Man **unterscheidet** zwischen **zusammengesetzte, mehrwertige und abgeleitete Attribute**.

Wichtig ist, wenn man eine **Entität modelliert**, dass schon immer vorab die erste Normalform der Normalisierung befolgt wird. So umgeht man spätere Modellierungsprobleme, wenn die zweite und dritte Normalform anstehen.

Zusammengesetzte Attribute

Die **zusammengesetzten Attribute** bestehen aus der Kombination mehrerer Attribute, die inhaltlich zusammengehören. Anhand einer Firmenadresse wird dies deutlich. Die **Firmenadresse selbst ist ein Attribut der Firma**, sie enthält aber die Attribute Straße, Hausnummer, Postleitzahl und Ort.

Mehrwertige Attribute

Mehrwertige Attribute können **einen oder mehrere Attributwerte aufnehmen**. So könnte ein Student gleichzeitig in zwei Studiengänge eingeschrieben sein.

Abgeleitete Attribute

Abgeleitete Attribute werden aus anderen Attributen oder aus Entitäten berechnet. Bezogen auf eine Datenbank wäre das z.B. die Bildung einer **Summe aus mehreren Spalten einer Tabelle**.

Attribute und Datentypen

Attribute können jeden von der Datenbank unterstützen **Datentyp** annehmen. Dabei muss bei der Konzeption der Datenbank, anhand einer Analyse ein allgemeingültiger Datentyp für jedes einzelne Attribut festgelegt werden. Treten bei den Entitäten unterschiedliche Datentypen für ein

gemeinsames Attribut auf, so muss vor der Speicherung eine Transformation in den Zieldatentyp durchgeführt werden.

Datenbank Index

Der **Datenbankindex** ist eine Datenstruktur mit deren Hilfe die Abfrageoptimierung gesteigert werden kann. Mittels einer Indextabelle werden die Daten sortiert auf dem Datenträger abgelegt.

Der **Index** selbst stellt einen **Zeiger** dar, der entweder auf einen weiteren Index oder auf einen Datensatz zeigt. Dadurch findet eine Trennung von Daten- und Index-Strukturen statt.

„Welche Arten von Datenbank-Indizes existieren?“

Bei **Datenbanken** unterscheidet man generell zwei Arten von Indizes. Zum einen gibt es **gruppierte Indizes** (Clustered Index). Zum anderen gibt es **nicht-gruppierte Indizes** (Nonclustered Index).

Ohne Indizes auf einer Tabelle müsste die Datenbank die Informationen (Datensatz) **sequentiell suchen**, was selbst mit modernster Hardware und Software **viel Zeit beanspruchen** kann.

Gruppierte Indizes (Clustered Index)

Bei der Verwendung eines gruppierten Index werden die Datensätze entsprechend der **Sortierreihenfolge ihres Index-Schlüssels gespeichert**. Wird für eine Tabelle beispielsweise eine Primärschlüssel-Spalte „NR“ angelegt, so stellt diese den Index-Schlüssel dar. Pro Tabelle kann **nur ein gruppierter Index erstellt** werden. Dieser kann jedoch aus mehreren Spalten zusammengesetzt sein.

Nicht-gruppierte Indizes (Nonclustered Index)

Besitzt eine **Tabelle einen gruppierten Index**, so können weitere nicht-gruppierte Indizes angelegt werden. Dabei zeigen die Einträge des Index auf den Speicherbereich des gesamten Datensatzes. Die **Verwendung eines nicht-gruppierten Index bietet sich an, wenn regelmäßig nach bestimmten Werten in einer Spalte gesucht** wird z.B. dem Namen eines Kunden.

Bei einer Abfrage wird nun zuerst nach dem Namen gesucht. Werden weitere Daten zum Kunden benötigt, so können diese über den gruppierten Index, der mit dem Namen abgelegt

wurde, abgerufen werden. Bei einem nicht-gruppierten Index ist es nicht notwendig, dass dessen Werte eindeutig sein müssen. Zudem kann auch dieser aus mehreren Spalten zusammengesetzt sein.

Darüber hinaus gibt es noch weitere sehr spezielle und zum Teil proprietäre Indizes, die in bestimmten Datenbanken verwendet werden. Beispielsweise der **Bitmap-Index, der im Data Warehouse eingesetzt wird**.

Vorteile von Datenbank Indizes

Der **Einsatz von Indizes empfiehlt sich für Datenbanken die große Datenmengen speichern** und **sehr häufig abgefragt werden**. Hier kommt es darauf an welche Informationen dabei eine zentrale Rolle spielen.

Welcher Index bei einer Abfrage tatsächlich verwendet wird, entscheidet in letzter Instanz der Abfrageoptimierer der Datenbank. Dieser erstellt für eine Abfrage mehrere Ausführungspläne, um die Kosten für die Abfrage zu ermitteln. Wird diese nun ausgeführt, wählt er den kostengünstigsten Ausführungsplan. Dieser berücksichtigt nicht nur Indizes, sondern auch die Systemauslastung.

Nachteile eines Datenbank Index

Das **Anlegen von Indexstrukturen führt zur Belegung von Plattenspeicher** und kann bei einer großen Anzahl von Indizes einen nicht unerheblichen Speicherverbrauch verursachen.

Ein weiterer Nachteil ist, dass der Einsatz von Indizes zu einem **größeren Aufwand beim Schreiben von Datensätzen führt**. Das Datenbankmanagementsystem muss in diesem Fall auch den Index berücksichtigen und diesen entsprechend laden. Hier gilt, **je mehr Indizes eine Tabelle hat**, desto **größer ist der Performance-Verlust beim Speichern** neuer Datensätze.

Primärschlüssel (Primary Key)

Der **Primärschlüssel** kommt in **relationalen Datenbanken** zum Einsatz und wird zur **eindeutigen Identifizierung eines Datensatzes** verwendet. In einer normalisierten Datenbank besitzen alle Tabellen einen Primärschlüssel.

Der **Wert** eines Primärschlüssels muss in einer Tabelle **einmalig sein**, da er jeden Datensatz **eindeutig kennzeichnet**. Des Weiteren wird er häufig als Datenbank-Index verwendet, um die Daten auf der Festplatte abzulegen.

„Welche Arten von Primärschlüsseln gibt es?“

Der **Primärschlüssel einer Relation** kann unterschiedlich aufgebaut sein. Man unterscheidet zwischen **eindeutige, zusammengesetzte und künstliche Primärschlüssel**.

Eindeutiger Primärschlüssel

Hierbei handelt es sich um einen **eindeutigen Schlüssel der in einer Spalte der Tabelle gespeichert wird**. Als Spalte kann ein Attribut des Datensatzes verwendet werden, das für jeden Eintrag in der Tabelle einen einmaligen Wert annimmt. Als eindeutiges Primärschlüsselattribut könnte beispielsweise die Sozialversicherungsnummer in einer Mitarbeitertabelle verwendet werden.

Zusammengesetzter Primärschlüssel

Ist ein Datensatz anhand eines Attributes nicht eindeutig identifizierbar, so kann der Primärschlüssel auch aus einer Kombination mehrerer Attribute bestehen. Dabei muss sichergestellt werden, dass jede dieser Kombinationen nur einmalig auftritt. Ein **zusammengesetzter Primärschlüssel** kann z.B. der Vor- und Nachname, sowie das Geburtsdatum sein.

Künstlicher Primärschlüssel

Gibt es in einer Tabelle keine eindeutigen Spalten bzw. Kombinationen aus Spalten, so kann auch auf einen **künstlichen Schlüssel** zurückgegriffen werden. Dieser ist auch als Surrogate Key bekannt und wird als zusätzliche Spalte in einer Tabelle eingefügt. In der Praxis wird häufig eine fortlaufende Ganzzahlenfolge verwendet, um einen Datensatz eindeutig identifizieren zu können.

Fremdschlüssel (Foreign Key)

Der **Fremdschlüssel kann Bestandteil einer Tabelle** in einer relationalen Datenbank sein. Dabei handelt es sich um eine Schlüsselspalte, die auf einen Primärschlüssel einer anderen oder aber derselben Tabelle verweist.

„Welche Fremdschlüsselarten gibt es?“

Es kann sich dabei um einen **einfachen oder zusammengesetzten Schlüssel** handeln. Das hängt davon ab, wie der Primärschlüssel der referenzierten Tabelle aufgebaut ist.

Aufgrund der referentiellen Integrität kann der Fremdschlüssel nur Werte annehmen, die in der Referenztabelle vorhanden sind. Zudem kann eine beliebige Anzahl von Datensätzen den gleichen Fremdschlüsselwert aufweisen.

Beispiel für den Einsatz eines Fremdschlüssels

In einer normalisierten Tabelle die Kontakte verwaltet, kann beispielsweise zu einer Person ein Unternehmen referenziert werden. In der Tabelle „Ansprechpartner“ wird „Susi Meier“ und ihre Telefonnummer angelegt. Ihr Unternehmen wird aus der Tabelle „Unternehmen“ referenziert, das wäre dann beispielsweise die „ABC GmbH“. Über diese Referenz kann bei einer Abfrage die Anschrift und andere Fakten zum Unternehmen aus der Tabelle „Unternehmen“ abgerufen werden.

Beziehungen in Datenbanken

Zwischen **Relationen** (Tabellen/Entitäten) können **Beziehungen in einer Datenbank** bestehen. Angenommen man hat eine Relation „Mutter“ und eine Relation „Kind“ – denkbar wären nun vier Möglichkeiten von Assoziationen/Beziehungen zwischen den Tabellen.

„Beziehungen zwischen Tabellen erstellen – so geht’s!“

In einem **Datenbankmodell** können folgende Beziehungen auftreten:

Jede Mutter hat **exakt ein** Kind

Jede Mutter hat **ein oder kein** Kind

Jede Mutter hat **mindestens ein** Kind

Jede Mutter hat eine **beliebige Anzahl von Kindern** (Mehr als 1, dann spricht man von Geschwistern)

Kardinalität von Beziehungen in relationalen Datenbanken

Die **Kardinalität von Beziehungen** definiert, wie viele Entitäten eines Entitätstyps mit genau einer Entität des anderen am Beziehungstyp beteiligten Entitätstyps (und umgekehrt) in Relation (**Beziehung**) stehen können oder müssen. Die Kardinalität von Beziehungen ist in relationalen Datenbanken in folgenden Formen vorhanden: **1:1 Beziehung**, **1:n Beziehung** und **m:n Beziehung**.

1:1 Beziehung in relationalen Datenbanken

In einer **„eins zu eins“-Beziehung in relationalen Datenbanken** ist jeder Datensatz in Tabelle A genau einem Datensatz in Tabelle B zugeordnet und umgekehrt. Diese Art von Beziehung sollte in der Modellierung vermieden werden, weil die meisten Informationen, die auf diese Weise in Beziehung stehen, sich in einer Tabelle befinden können. Eine **1:1-Beziehung** verwendet man nur, um eine Tabelle aufgrund ihrer Komplexität zu teilen oder um einen Teil der Tabelle aus Gründen der Zugriffsrechte zu isolieren.

1:n Beziehung in relationalen Datenbanken

Eine **„eins zu viele“-Beziehung relationalen Datenbanken** ist der häufigste Beziehungstyp in einer Datenbank. In einer **1:n-Beziehung** können einem Datensatz in Tabelle A mehrere passende Datensätze in Tabelle B zugeordnet sein, aber einem Datensatz in Tabelle B ist nie mehr als ein Datensatz in Tabelle A zugeordnet.

m:n Beziehung in relationalen Datenbanken

Bei **„viele zu viele“-Beziehung in relationalen Datenbanken** können jedem Datensatz in Tabelle A mehrere passende Datensätze in Tabelle B zugeordnet sein und umgekehrt. Diese Beziehungen können nur über eine dritte Tabelle, eine Verbindungstabelle C, realisiert werden. Die Verbindungstabelle C enthält in der Regel nur die Fremdschlüssel der beiden anderen Tabellen (A/B). Der Primärschlüssel der Verbindungstabelle wird aus diesen beiden Fremdschlüsseln gebildet. Daraus folgt, dass eine **m:n Beziehung in Wirklichkeit zwei 1:n Beziehungen sind**.

Normalisierung von Datenbanken

Unter **Normalisierung** eines **relationalen Datenbankmodells** versteht man die Aufteilung von Attributen in mehrere Relationen (Tabellen) mithilfe der **Normalisierungsregeln und deren Normalformen**, sodass eine Form entsteht, die keine vermeidbaren Redundanzen mehr enthält.

„Warum wird eine Normalisierung durchgeführt?“

Ziel der **Normalisierung** ist eine **redundanzfreie Datenspeicherung zu erstellen**. Redundanzfrei bedeutet, dass Daten entfernt werden können, ohne dass es zu Informationsverlusten kommt.

Weiterhin soll die Normalisierung Anomalien entfernen. Im **Normalisierungsprozess** gibt es fünf Normalformen, welche im Folgenden genauer erklärt werden.

In der **Datenbanktentwicklung** ist die **Dritte Normalform** oft ausreichend, um die **perfekte Balance aus Redundanz, Performance und Flexibilität für eine Datenbank zu gewährleisten**. Natürlich gibt es auch Sonderfälle, z.B. im wissenschaftlichen Bereich, wo eine Datenbank bis zur 5. Normalform normalisiert werden kann bzw muss.

Ziele der Datenbank-Normalisierung

- Beseitigung von Redundanzen
- Vermeidung von Anomalien (funktionelle und transitive Abhängigkeiten)
- Erstellung eines klar strukturierten Datenbankmodells

Grundlagenwissen zur Normalisierung von Datenbanken

- Was sind Redundanzen?
- Was sind Anomalien in einer Datenbank?
- Welche Abhängigkeiten gibt es innerhalb der Normalisierung?

Artikel im Bereich Normalisierung von Datenbanken

- Nullte Normalform
- Erste Normalform (1NF)
- Zweite Normalform (2NF)
- Dritte Normalform (3NF)
- Boyce Codd Normalform (BCNF)
- Vierte Normalform (4NF)
- Fünfte Normalform (5NF)

Referentielle Datenintegrität

Im Bereich der **relationalen Datenbanken** wird die **referentielle Integrität** dazu verwendet die Konsistenz und die Integrität der Daten sicherzustellen. Dazu werden **Regeln aufgestellt**, wie und unter **welchen Bedingungen ein Datensatz in die Datenbank eingetragen** wird.

Bei der **referentiellen Integrität** können Datensätze die einen Fremdschlüssel aufweisen **nur dann gespeichert werden**, wenn der Wert des **Fremdschlüssels einmalig in der referenzierten Tabelle existiert**. Im Falle, dass ein referenzierter Wert nicht vorhanden ist, kann der Datensatz nicht gespeichert werden.

„Warum wird die Referentielle Integrität benötigt?“

Eine **Datenbank kann schnell in einen inkonsistenten Zustand geraten**. Im ungünstigsten Fall liegt eine nicht-normalisierte Datenbank vor, die starke Redundanzen aufweist.

Dabei können **Anomalien im Datenbestand auftreten**, die verschiedene Formen annehmen. Man spricht hier von **Einfüge-, Lösch- und Änderungsanomalien**. Tritt eine oder mehrerer dieser Anomalien auf, kann das zur Verfälschung oder Löschung von Informationen führen.

Einfüge-Anomalien in einer Datenbank

Eine **Einfüge-Anomalie tritt auf**, wenn ein **Datensatz** gespeichert werden soll und dieser **keine oder kein eindeutigen Primärschlüsselwerte aufweist**.

Das Einfügen in eine Tabelle ist somit nicht möglich. Informationen können nicht gespeichert werden und gehen womöglich verloren.

Das kann zum Beispiel der Fall sein, wenn für die Speicherung der Kundendaten zu Verifizierungszwecken die Personalausweisnummer als Primärschlüssel verwendet wird, diese aber leider vom Sachbearbeiter nicht erfasst werden konnte. Der Datensatz des Kunden kann nicht gespeichert werden.

Änderungs-Anomalien in einer Datenbank

Man spricht von einer **Änderungs-Anomalie**, wenn eine **Entität redundant** in einer oder sogar in mehreren Tabellen enthalten ist und bei einer Aktualisierung nicht alle berücksichtigt werden. Dadurch kommt es zur Inkonsistenz im Datenbestand. Es **kann möglicherweise nicht mehr nachvollzogen** werden welcher Wert der gültige Datensatz ist.

Dieser **Sachverhalt lässt sich gut an einer Auftragstabelle darstellen**. Diese speichert neben der Auftragsnummer auch den Namen eines Kunden und dessen Bestellung. Ein Kunde kann mehrere Bestellungen aufgegeben haben, wobei jede Bestellung in einem Datensatz erfasst wird. Wird nun aufgrund eines Schreibfehlers nachträglich der Name des Kunden „Reiher“ in „Reier“ bei einem Datensatz geändert, führt dies zu einem inkonsistenten Datenbestand. Nach der Änderung liegen demnach Aufträge für scheinbar zwei verschiedene Kunden vor und zwar für einen Kunden „Reiher“ und einen Kunden „Reier“.

Lösch-Anomalien in einer Datenbank

Enthalten die **Datensätze einer Tabelle mehrere unabhängige Informationen**, so kann es leicht zu **Lösch-Anomalien kommen**. Da sich die Daten in einem nicht-normalisierten Zustand befinden, kann durch Löschen eines Datensatzes ein Informationsverlust entstehen. Die

Ursache liegt darin, dass in einer **Tabelle unterschiedliche Sachverhalte gespeichert werden**.

Am Beispiel einer nicht-normalisierten Mitarbeitertabelle soll dies kurz skizziert werden. In der Mitarbeitertabelle werden die Personalnummer, der Name und die Abteilung gespeichert. Der Mitarbeiter „Krause“, der als einziger in der Abteilung „Lager“ war, ist aus dem Unternehmen ausgetreten und wird daher aus der Datenbank gelöscht. Da die Abteilung in der gleichen Tabelle gespeichert wird, verschwindet das „Lager“ aus der Datenbank, da „Herr Krause“ ja als einziger dieser Abteilung zugeordnet war.

Datenbank-Anomalien auflösen

Die **beschriebenen Anomalien** treten durch ein **schlechtes Datenbank-Design auf**. Daraus ergibt sich auch die redundante Datenhaltung. Um diese zu vermeiden, müssen die Tabellen einer Datenbank normalisiert werden. Die Normalisierung umfasst in der Praxis drei Stufen und sorgt für eine redundanzfreie und nach Entitätstyp getrennte Datenhaltung.