Datenbanken Kurzübersicht

Andy Dunkel

Inhaltsverzeichnis

1 Einführung Datenbanken

Grundsätzlich ist eine Datenbank eine geordnete Sammlung von Informationen, welche in irgendeiner Weise miteinander in Beziehung stehen.

1.1 Grundbegriffe des ER-Modells

1.1.1 Entität

Als Entität wird eine eigenständige Einheit, die im Rahmen des betrachteten Modells **eindeutig** identifiziert werden kann, bezeichnet. Dieses Identifizierungsmerkmal wird als "Schlüssel" bezeichnen. Entitäten bezeichnen Dinge der realen Welt, z.B. Personen, Firmen, Objekte etc.
Beispiele:

- Kunde Hubert = Entität, Kundennummer = Schlüssel
- Artikel XYZ = Entität, Artikelnummer = Schlüssel

1.1.2 Abhängige Entität

Entität ist abhängig, wenn die Existenz der Entität von einer anderen Entität abhängt und nicht allein durch eigene Attribute identifiziert werden kann.
Beispiele:

- Position einer Bestellung die entfallen kann, wenn Bestellung storniert wird
- Bankverbindung eines Kunden, wenn dieser gelöscht wird

1.1.3 Entitätentypen und Entitätenmengen

Der **Entitätentyp** ist eine abstrakte Zusammenfassung von Entitäten, welche in dem betrachteten Modell durch die selben Eigenschaften beschrieben werden können. Jede Entität ist durch einen Entitätentyp beschreibbar.
Eine **Entitätenmenge** ist eine Menge von Entitäten des gleichen Typs (z.B. die Menge aller Studenten an der Berufsakademie).
Beispiele:

Entitäten	Student Lehmann, Getränk Bier, Projekt Assembler
Entitätenmengen	Alle Studenten, Alle Getränke, Alle Projekte
Entitätentypen	Mitarbeiter, Abteilung, Projekte

Abbildung 1: Darstellung von Entitätentypen

Eigenschaften von Entitäten werden durch **Attribute** beschrieben. Ein Attribut besteht aus dem Attributbezeichner und einen Wert. Die Domaine beschreibt den Datentypen und den Wertebereich. Grafisch werden Attribute als Ellipsen dargestellt, welche durch eine ungerichtete Kante mit dem Entityp verbunden sind. Der Schlüssel wird unterstrichen dargestellt.

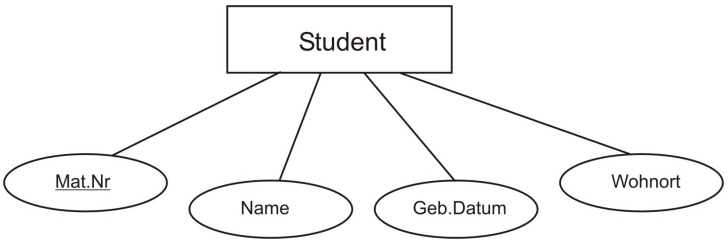

Abbildung 2: Darstellung von Entitätentypen

Alternative Schreibweise: *Student = (Mat.Nr, Name, Geb.Datum, Wohnort)*

1.1.4 Schlüssel

Jede Entität muss durch ihre Attribute eindeutig identifizierbar sein (Ausnahme sind abhängige Entitäten, hier reichen Attribute die von der jweiligen Vaterentität unterschiedlich oder nicht enthalten sind). Daher muss für jeden Entitätentyp ein Schlüssel definiert werden, der aus einem oder mehreren Attributen besteht. Es gibt keine zwei Entitäten, die in allen Attributen des Schlüssels übereinstimmen.
Notwendige Eigenschaften von Schlüsseln:

- **Eindeutigkeit:**
  Für jede Entität gibt es genau einen Identifikationsschlüsselwert, der nicht nochmal vorkommt.
- **Laufende Zuteilbarkeit:**
  Eine neu auftretende Entität erhält ihren Schlüssel sofort.
- **Kürze/Schreibbarkeit**
  So kurz wie möglich, so lang wie nötig.
- **Keine NULL-Werte**
  Ein Schlüssel muss immer einen gültigen Wert haben, NULL-Werte sind daher nicht zulässig.

Ist unter den Attributen kein Schlüsselkandidat vorhanden, welcher als Schlüssel in Frage kommen kann, so wird ein "künstliches" Schlüsselattribut, wie z.B. Integer, Autowert, eingefügt.

1.1.5 Relationale Beziehungen (Relationships)

Entitäten können miteinander in Beziehung stehen. Der Beziehungstyp ist ähnlich dem Entitätentyp eine Abstraktion gleichartiger Beziehungen. Grafisch werden Relationships durch Rauten dargestellt, welche durch ungerichtete Kanten mit den Entitätentypen verbunden sind.

Beispiel: Student B. Lehmann arbeitet an Projekt Vokabeltrainer

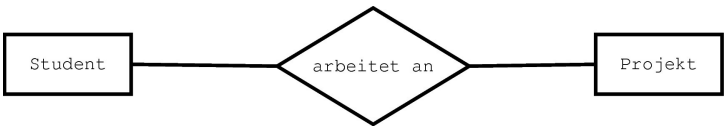

Abbildung 3: Darstellung von Beziehungen

Beziehungen können ebenfalls eigene Attribute besitzen:

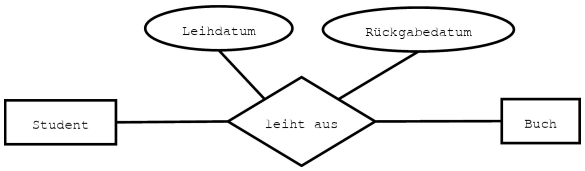

Abbildung 4: Relationship mit Attributen

Rekursive Beziehungen sind ebenfalls möglich:

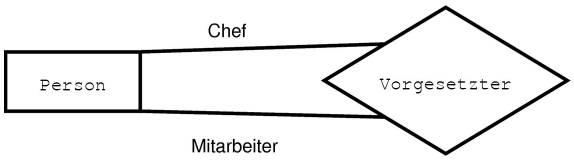

Abbildung 5: Rollennamen in Diagrammen

2 Beziehungstypen

2.1 1:1-Beziehung

Die 1:1-Beziehung ist ein Sonderfall, weil es möglich ist die Attribute der beiden Entitäten in eine Entität zu vereinen. Dennoch gibt es Fälle bei denen eine Aufteilung sinnvoll ist. Jeder Entität einer Entitymenge wird bei dieser Beziehung genau eine Entität einer anderen Entitymenge zugeordnet.

Abbildung 6: 1:1 Beziehung

Logisches Modell (ER-Modell)

Abbildung 7: 1:1 Beziehung als ER-Modell

2.1.1 1:n-Beziehung

Bei der 1:n-Beziehung steht eine Entität eines Entitätstyps mit beliebig vielen anderen einer anderen Entitätenmenge in Beziehung. Damit diese Verbindung hergestellt werden kann, muss in beiden Tabellen ein gemeinsames Attribut definiert sein (Primärschlüssen, Fremdschlüssel).

Beispiel: Jeder Kunde kann beliebig viele Rechnungen erhalten.

Abbildung 8: 1:n Beziehung

Logisches Modell (ER-Modell)

Abbildung 9: 1:n Beziehung als ER-Modell

2.1.2 n:m-Beziehung

Bei der n:m-Beziehung kann jeder Entität beliebig vielen anderen Entitäten eines anderen Entitätentyps zugeordnet werden. Dies gilt auch umgekehrt.

Abbildung 10: n:m Beziehung

ER-Modell Eine n:m-Beziehung kann nicht direkt als ER-Modell dargestellt werden. Hier ist das Anlegen einer Verknüpfungsrelation notwendig.

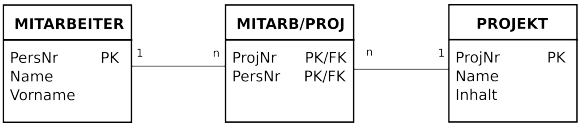

Abbildung 11: n:m Beziehung als ER-Modell

2.1.3 Die (min,max) Notation

Beziehungen können auch präziser angegeben werden (Unter- und Obergrenze).

Abbildung 12: Angabe von Kardinalitätsrestriktionen

2.1.4 Schwache Entitäten

Schwache Entitäten können nicht als eigenständiges Gebilde exitisieren, sie benötigen eine “starke” Entität zur eindeutigen Identifizierung. Eine starke und eine schwache Entität gehen entweder eine 1:n- oder eine 1:1-Beziehung ein. Eine n:m-Beziehung ist nicht möglich. Schwache Entitäten haben im Normalfall keinen eigenständigen Primärschlüssel, der alle Entitäten eindeutig identifiziert, diese Indentifizierung geschieht über die übergeordnete Entität.

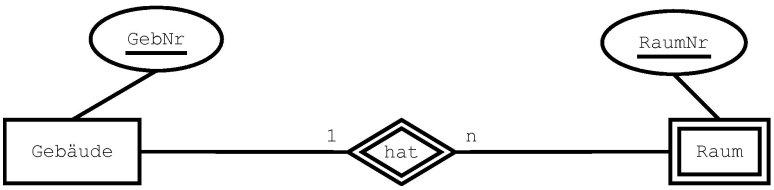

Abbildung 13: Darstellung von schwachen Entitäten

3 Generalisierung

Wie in objektorientierten Programmiersprachen können Attribute von einem Obertyp an einen Untertyp vererbt werden. Der Untertyp erbt alle Eigenschaften des Obertyps. Grafisch wird durch “is-a” an den ungerichteten Kanten dargestellt. Alternative Darstellung mit einem Dreieck als Symbol ist möglich.

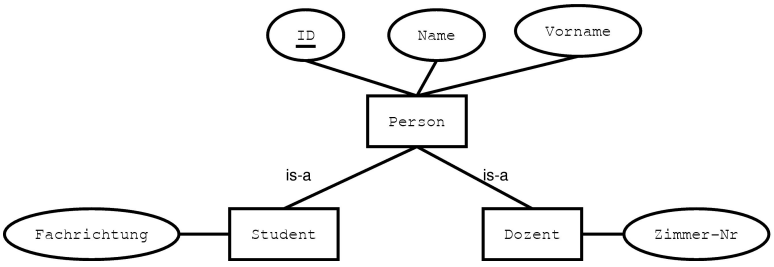

Abbildung 14: Beispiel einer Generalisierung

4 Aggregation

Die Aggregation wird zur Darstellung von “ist ein Teil von” Beziehungen verwendet. Die Beziehung wird mit einem “part-of” oder einer Raute grafisch gekenntzeichnet. Objekte können gleichzeitig Elemente verschiedener Komponenten sein, bzw. Subkomponenten in mehreren Oberkomponenten.

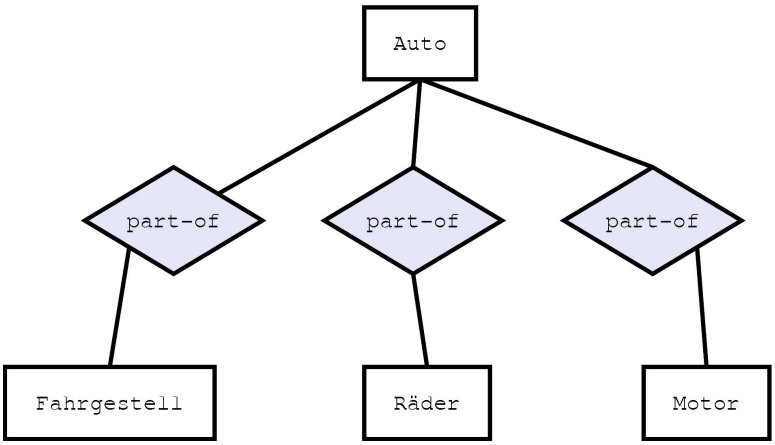

Abbildung 15: Beispiel einer Aggregation

5 Das relationale Datenmodell

5.1 Grundbegriffe

Eine relationale Datenbank besteht aus Relationen (Tabellen). Diese Relationen enthalten eine beliebige Anzahl von Tupeln (Zeilen) und eine fixe Anzahl von Attributen (Spalten).

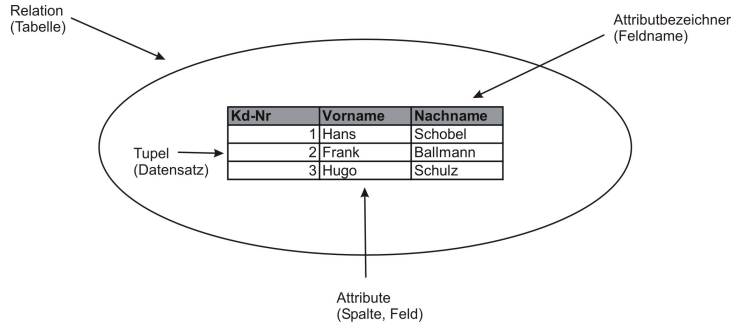

Abbildung 16: Grundbegriffe Datenbank

5.2 Primär- und Fremdschlüssel

Ein wichtiges Leistungsmerkmal relationaler Datenanken ist es, dass Tabellen untereinander verknüpft werden können. Die Informationen können so auf mehrere Tabellen verteilt werden. Die Verknüpfung erfolgt dadurch, dass Datensätze über ein bestimmtes Feld einander zugeordnet werden. Es wird hierbei zwischen Haupttabelle und Detailtabelle unterschieden.
Jede Tabelle benötigt grundsätzlich einen **Primärschlüssel (primary key)**, welcher einen Datensatz eindeutig kennzeichnet. Dieser darf nie den Wert NULL annehmen. Primärschlüssel werden durch unterstreichen der Attribute gekennzeichnet:

Beispiele

- Person(PersNr, Name, Nachname, Strasse, PLZ, Ort)
- Auftrag(AuftragNr, Datum, Sachbearbeiter)

Jede verknüpfte Tabelle benötigt zudem einen **Fremdschlüssel (foreign key)**, über den sich die Datensätze zuordnen lassen.

- Ein Fremdschlüssel referenziert immer auf einen Primärschlüssel, er hat die gleich Anzahl Spalten und ist vom gleichen Datentyp
- In einer Tabelle können mehrere Fremdschlüssel vorkommen.
- Eine Spalte kann gleichzeitig Fremd- und Primärschlüssel sein.
- Der Fremdschlüssel kann auch auf die eigene Tabelle referenzieren, z.B. ein Mitarbeiter referenziert über einen Fremdschlüssel auf seinen Vorgesetzten.
- Der Fremdschlüssel ist zusammengesetzt, wenn auch der Primärschlüssel zusammengesetzt ist.

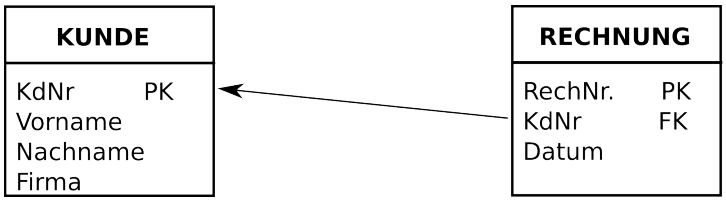

Abbildung 17: Beispiel für Primärschlüssel - Fremschlüssel

- Kunde(KdKr, Vorname, Nachname, Firma)
- Rechnung(RechNr, KdNr REFERENCES Kunde, Datum)

5.3 Integritätsregeln

In einer relationalen Datenbank gibt es eine Reihe von Integritätsregeln.

Beispiele:

- Eine Kundennummer muss eindeutig sein.
- Kontostand darf Kreditlimit nicht unterschreiten
- Es darf keine Rechnung mit einem Artikel existieren, welcher nicht im Artikelstamm vorhanden ist.
- Keine doppelte Kundennummer.
- Keine Bestellung ohne einen Kunden.

Es ist Aufgabe des Datenbankverwaltungssystems die Integritätsbedingungen dafür zu sorgen, dass diese Regeln eingehalten werden. Dies geschieht **unabhängig** von den Anwendungsprogrammen.

Das relationale Modell kennt folgende Arten von Integritätsregeln:

- **Domain oder Wertebereitsbedindungen**: Einschränkung von möglichen Werte für Attribute.
- **Primärschlüssel:** Jeder Datensatz ist eindeutig identifizierbar.
- **Weitere Schlüssel (Unique Key):** Unique Key hat alle Eigenschaften eines Primary Key, stellt aber lediglich die Eindeutigkeit sicher, dient aber nicht zur Indentifizierung des Tupels.
- **Referentielle Integritätsbedingung:** Diese Integrität stellt sicher, dass ein Feldwert einer untergeordneten Detailtabelle mit den Werten im Schlüsselfeld einer übergeordneten Haupttabelle übereinstimmt. Z.B. Benutzer kann Rechnung nur für einen Kunden anlegen, der Bereits in der Haupttabelle angelegt ist. Bei der referentiellen Integrität gibt es mehrere Verhaltensweisen, welche beim Löschen/Ändern eines Datensatzes in der Haupttabelle eintreten können:
  - **restricted (nicht zulässig) / no action**
    Datensatz darf nicht gelöscht werden, sofern noch eine Referenz auf diesen in einer Detailstabelle existiert. Dies ist die Vorgabeeinstellung.
  - **cascade (weitergeben)**
    Alle Datensätze in der Detailtabelle die auf den Datensatz in der Haupttabelle referenzieren werden mit gelöscht.
  - **set null (auf NULL setzen) / set default (auf Vorgabe setzen)**
    Die Fremdschlüssen die auf den Datensatz der Haupttabelle referenzieren werden auf NULL oder auf einen Vorgabewert gesetzt.

5.4 Überführung von ER-Diagrammen in Relationenmodell

5.4.1 1:n-Beziehungen

Abbildung 18: Umsetzung einer 1:n Beziehnung

Bei einer 1:n-Beziehung kann die Beziehung in eine eigene Tabelle überführt werden:

- Kunde(KdNr, Name...)
- Rechnung(RechNr, Datum ...)
- Erhalten(KdNr REFERENCES Kunde, RechNr REFENRENCES Rechnung)

Es ist allerdings auch eine kompakte Überführung mit zwei Relationen möglich, die Relation die maximal einen Beziehungspartner hat, wird hierbei die Detailstabelle.

- Kunde(KdNr, Name...)
- Rechnung(RechNr, KdNr REFERENCES Kunde,Datum ...)

5.4.2 1:1-Beziehungen

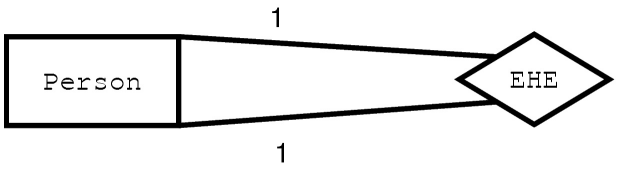

Abbildung 19: Umsetzung einer 1:1 Beziehnung

Eine 1:1-Beziehung kann analog umgesetzt werden:

- Pers (PNR, Name ...)
- Ehe (PNR-Mann REFERENCES Pers, PNR-Frau REFERENCES PERS)

Oder alternativ in eine Tabelle:

- Pers(PNR, Name, ... , PNR-Gatte REFERENCES Pers)

5.4.3 n:m-Bezieungen

Bei einer n:m-Beziehnung kann jeder Partner mit mehreren Partnern in Beziehung stehen, daher ist ein direkter Verweis auf den anderen Partner nicht möglich. Daher muss eine Zwischentabelle eingefügt werden. Diese wird mit 1:n und n:1 zu den jeweiligen Partnern verknüpft. Attribute der Relation wandern ebenfalls in diese Zwischentabelle.

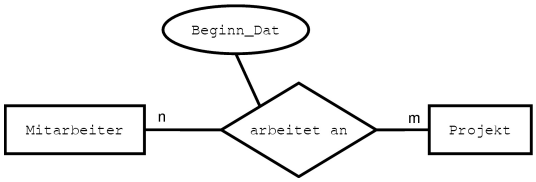

Abbildung 20: Umsetzung einer n:m Beziehnung

Umsetzung in das Relationenmodell:

- Mitarbeiter(PersNr, MName, ...)
- Projekt(ProjektNr, PName, ...)
- MitarbeiterProjekt(PersNr REFERENCES Mitarbeiter, ProjektNr REFERENCES Projekt, Beginn_Dat)

5.4.4 Beziehungen mit Kardinalität größer zwei

Eine Relationship die eine Beziehung zwischen mehr als zwei Entitäten herstellt, wird wie eine n:m-Beziehung behandelt. Es wird analog eine Zwischentabelle angelegt, welche die Primärschlüssel der referenzierten Entitätenmenge enthält. Diese bilden in der Zwischentabelle einen gemeinsamen Primärschlüssel.

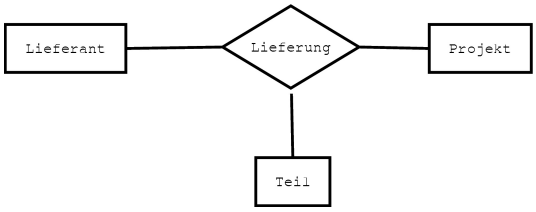

Abbildung 21: Umsetzung von Beziehungen mit Kardinalität > 2

Umsetzung in das Relationenmodell:

- Lieferant(LNr, LName, ...)
- Projekt(PNr, PName, ...)
- Teil(TNr, TName, ...)
- Lieferung(LNr REFERENCES Lieferant, PNr REFERENCES Projekt, TNr REFERENCES Teil, Anzahl, Datum)

5.4.5 Umsetzung schwacher Entitätentypen

Schwache Entitätentypen sind ein Spezialfall einer 1:n Beziehung, der Schlüssel der 1-Entität wandert als Fremdschlüssel in die Detailtabelle. Dort bildet er mit dem Schlüssel der Detailstabelle einen zusammengesetzten Primärschlüssel.

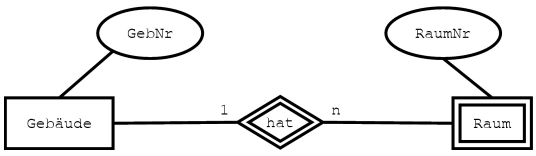

Abbildung 22: Umsetzung von schwachen Entitäten

Umsetzung in das Relationenmodell:

- Gebäude(GebNr, GName, ...)
- Raum(RaumNr, GebNr REFERENCES Gebäude, RName, ...)

5.4.6 Zusammenfassung der Umsetzungsregeln

- Entitätentypen bilden immer eine eigene Relation.
- Mehrwertige Attribute[1] werden durch eine eigene Relation dargestellt.

5.5 Abbildung von Generalisierung im Relationenmodell

Das Relationenmodell sieht keine Unterstützung der Abstraktionsvorgänge und Vererbung vor. Die Simulation ist nur eingeschränkt möglich und die verschiedenen Methoden haben alle ihre Vor- und Nachteile.

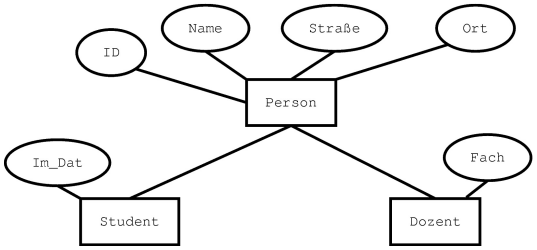

Abbildung 23: Beispiel einer Generalisierung

Attribute der einzelnen Entitätentypen sind:

- Person (ID, Name, Straße, Ort)
- Student (Im_Dat)
- Dozent (Fach)

5.5.1 Hausklassenmodell

Beim Hausklassenmodell wird für jeden Entitätentyp eine Relation erzeugt, welche alle Attribute der Oberklassen beinhaltet. Jede Instanz ist daher komplett und von anderen unabhängig, dies erspart Redundanzen erschwert aber die Suche, da mehrere Tabellen durchsucht werden müssen, z.B. bei der Suche nach der Person mit der ID “5”.
Die Basisklasse kann hierbei entfallen.

- Person (ID, Name, Straße, Ort)
- Student (ID, Name, Straße, Ort, Im_Dat)
- Dozent (ID, Name, Straße, Ort, Fach)

Beispiel: Wird ein Student angelegt, so werden die Eigenschaften nur in die Studenttabelle eingetragen.

[1] Ein Attribut, das für eine einzelne Entität mehrere Werte gleichzeitig haben kann (z.B. "E-Mail-Adresse": eine Person kann mehrere haben).

5.5.2 Partitionierungsmodell

Beim Partitionierungsmodell werden nur die (auf der Vererbungsstufe) neu hinzukommenden Attribute mit in die Relationen aufgenommen sowie der Primärschlüssel. Soll eine Person in die Datenbank eingetragen werden, so werden die Attribute auf die Relationen aufgeteilt. Dadurch entsteht ein erhöhter Aufwand, wenn alle Eigenschaften einer Person abgefragt werden sollen, da hierzu in verschiedenen Relationen nachgeschaut werden muss.

- Person (ID, Name, Straße, Ort)
- Student (ID, Im_Dat)
- Dozent (ID, Fach)

Beispiel: Wird ein Student angelegt, so werden die Eigenschaften zur Person in die Tabelle "Person" eingetragen, die zusätzlichen Informationen werden in "Student" eingetragen.

5.5.3 Volle Redundanz

Bei der vollen Redundanz werden die Relationen wie beim Hausklassenmodell angelegt, beim Abspeichern von Daten werden diese jedoch in alle Relationen eingetragen. Dies hat Vorteile bei der Abfrage der Daten, allerdings müssen bei Änderungen diese u.U. an mehreren Stellen durchgeführt werden, was Inkonstenzen zur Folge haben kann.

- Person (ID, Name, Straße, Ort)
- Student (ID, Name, Straße, Ort, Im_Dat)
- Dozent (ID, Name, Straße, Ort, Fach)

Beispiel: Wird ein Student angelegt, so werden die Eigenschaften komplett in die Person- und Studenttabelle eingetragen.

5.5.4 Überrelation

Alle Datensätze werden hier, mit ihren Attributen, in einer Relation abgelegt, was die Nachteile der bisherigen Lösungen beseitigt. Zusätzlich wird jedoch noch ein Attribut "Typ" mit in die Relation aufgenommen, was eine Zuordnung des Entitätentyps ermöglicht.
Nachteilig wirkt sich hier aus, dass alle Attribute in einer Relation landen, was bei vielen Attributen zu Unübersichtlichkeit führen kann. Attribute die nicht zum jeweiligen Typ gehören werden mit NULL belegt. Der Benutzer muss immer wissen, welche Attribute zu welchem Typ gehören.

- Person (ID, Typ, Name, Straße, Ort, Im_Dat, Fach)

Beispiel: Beim Anlegen eines Studenten wird das Feld "Fach" mit NULL belegt und im Feld Typ der Typ "Student" hinterlegt.

6 Normalformen

Um bei relationalen Datenbanken Redundanzen und Fehler so gering wie möglich zu halten, müssen die Relationen (Tabellen) nach den Regel der "Normalisierung" erstellt werden.

6.1 Die 1. Normalform

Jede Spalte enthält unteilbare (atomare) Informationen. Die Datensätze verwenden keine sich wiederholenden Informationen, die nicht auch zu einer separaten Gruppe zusammengefasst werden können.

PNR	NAME	FACH	STUDENT
300	Fahr	INF	(4384, Späth, …) (4759 Ginter, …)
500	Zinsi	SWENG	(5736 Stack, …) (5739 Dunkel, …)

Abbildung 24: Unnormalisierte Relation "Diplomprüfungen"

Die Spalte Student verstößt gegen die 1. Normalform, da die Werte nicht atomar sind, diese werden daher in eine eigene Relation ausgelagert.

Relation Prüfer

PNR	NAME	FACH
300	Fahr	INF
500	Zinsi	SWENG

Relation Prüfung

PNR	MATNR	NAME	GEB	FR	FNAME	FACHLEITER	PDAT	Note
300	4384	Späth	12.06.2000	INF	Informatik	Fahr	04.02.2007	2.0
300	4759	Ginter	27.02.1978	INF	Informatik	Fahr	05.02.2007	3.0
500	5736	Stack	10.10.1967	BWL	Betriebswirt.	Zinsi	06.02.2007	5.0
500	5739	Dunkel	03.04.1983	BWL	Betriebswirt.	Zinsi	03.01.2008	1.0

Abbildung 25: Relationen in 1. Normalform

6.2 Die 2. Normalform

Es wird die 1. Normalform eingehalten und alle Informationen in den Nicht -Schlüsselfeldern hängen nur vom kompletten Primärschlüssel ab, nicht jedoch von einzelnen Schlüsselteilen.

Alle Tabellen in der ersten Normalform, die einen einfachen (aus nur einer Spalte bestehenden) Primärschlüssel haben, sind so automatisch auch in der zweiten Normalform. Ist aber der Primärschlüssel zusammengesetzt, ist für die zweite Normalform zu prüfen, ob es Attribute gibt, die in Wirklichkeit nur von einer oder einem Teil der Spalten des Primärschlüssels abhängen.

Wie man in der Relation Prüfung erkennt, hängen die Attribute eines Studenten nur vom Teilschlüssel "MATNR" ab. Daher werden diese Attribute in eine eigenständige Studentenrelation ausgelagert.

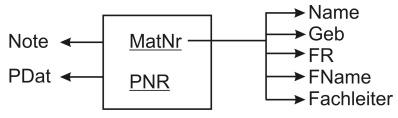

Abbildung 26: Darstellung der funktionalen Abhängigkeiten

Relation Prüfer

PNR	NAME	FACH
300	Fahr	INF
500	Zinsi	SWENG

Relation Prüfung

PNR	MATNR	PDAT	Note
300	4384	04.02.2007	2.0
300	4759	05.02.2007	3.0
500	5736	06.02.2007	5.0
500	5739	03.01.2008	1.0

Relation Student

MATNR	NAME	GEB	FR	FNAME	FACHLEITER
4384	Späth	12.06.2000	INF	Informatik	Fahr
4759	Ginter	27.02.1978	INF	Informatik	Fahr
5736	Stack	10.10.1967	BWL	Betriebswirt.	Zinsi
5739	Dunkel	03.04.1983	BWL	Betriebswirt.	Zinsi

Abbildung 27: Relationen in 2. Normalform

Man kann allerdings bereits erkennen das in den Relationen noch Redundanzen vorhanden sind.

6.3 Die 3. Normalform

Es wird die 2. Normalform eingehalten und alle Informationen in den Nicht -Schlüsselfeldern der Tabelle sind untereinander vollständig unabhängig. Es dürfen keine transitiven Abhängigkeiten vorliegen.
In der Relation "Student" hängen die Attribute "FNAME" und "FACHLEITER" transitiv von "MATNR" und auch von "FR" ab. Diese müssen daher in eine extra Relation ausgelagert werden.

Relation Prüfer

PNR	NAME	FACH
300	Fahr	INF
500	Zinsi	SWENG

Relation Prüfung

PNR	MATNR	PDAT	Note
300	4384	04.02.2007	2.0
300	4759	05.02.2007	3.0
500	5736	06.02.2007	5.0
500	5739	03.01.2008	1.0

Relation Fachrichtung

FR	FNAME	FACHLEITER
INF	Informatik	Fahr
BWL	Betriebswirt.	Zinsi

Relation Student

MATNR	NAME	GEB	FR
4384	Späth	12.06.2000	INF
4759	Ginter	27.02.1978	INF
5736	Stack	10.10.1967	BWL
5739	Dunkel	03.04.1983	BWL

Abbildung 28: Relationen in 3. Normalform

6.4 Boyce-Codd-Normalform

Eine Relation ist in Boyce-Codd Normalform, wenn jeder Determinant ein Schlüsselkandidat ist.
Die Boyce-Codd-Normalform ist eine Weiterentwicklung der 3. Normalform. In dieser kann es vorkommen, dass ein Teil eines (zusammengesetzten) Schlüsselkandidaten funktional abhängig ist von einem Teil eines anderen Schlüsselkandidaten.

PERSNR	PROJNAME	TELNR	AUFGABE
1	Dokumentation	110	Vorlagen erstellen
2	Roboter	112	CNC-Funktionen
3	CMS	129	MySQL-Funktionen
1	DMS	110	Dokumente einstellen
4	SPS-Erstellung	374	Programmieren

Abbildung 29: Relation nicht in BCNF

In dieser Relation sind "PERSNR" und "PROJNAME" Primärschlüssel jedoch ist auch "TELNR" ein Schlüsselkandidat. Diese ist von "PERSNR" funktional abhängig und wird daher in eine extra Relation ausgelagert. Da es sich bei "TELNR" um einen Schlüsselkandidaten handelt, gibt es verschiedene Möglichkeiten der Aufteilung.

PERSNR	PROJNAME	AUFGABE
1	Dokumentation	Vorlagen erstellen
2	Roboter	CNC-Funktionen
3	CMS	MySQL-Funktionen
1	DMS	Dokumente einstellen
4	SPS-Erstellung	Programmieren

PERSNR	TELNR
1	110
2	112
3	129
4	374

Abbildung 30: Relationen in BCNF

6.5 4. Normalform

Die 4. Normalform beschreibt die mehrwertige Abhängigkeit (MWAs). Eine Datenbank ist dann in der 4. Normalform, wenn sie nur noch triviale mehrwertige Abhängigkeiten enthält.

PERSNR	LFARBE	FAHRZEUG
1	Rot	Golf
1	Rot	Ferrari
1	Blau	Golf
1	Blau	Ferrari
2	Hund	Porsche

Abbildung 31: Relation nicht in 4. Normalform

Wie man erkennen kann gibt es in den Attributen "LFARBE" und "FAHRZEUG" Redundanzen. "LFARBE" und "FAHRZEUG" sind jedoch unabhängig voneinander und werden daher in zwei Relationen aufgeteilt.

PERSNR	LFARBE
1	Rot
1	Blau
2	Hund

PERSNR	FAHRZEUG
1	Golf
1	Ferrari
2	Porsche

Abbildung 32: Relation in 4. Normalform

7 Relationenalgebra

7.1 Selektion

Bei der Selektion werden die Datensätze einer Relation ausgewählt auf welche die Selektionsbedingung zutrifft.

Beispiele: Relation: Pers(PNR, Name, Gehalt, Provision, Wohnort)
Selektion 1, alle Mitarbeiter mit Namen "Mueller" und Gehalt größer 2000:

$$\sigma_{Name='Mueller' \wedge Gehalt>2000}(Pers)$$

Selektion 2, alle Miarbeiter mit Wohnort "Krauchewiese" oder Namen Lehmann:

$$\sigma_{Name='Lehmann' \vee Wohnort='Krauchewies'}(Pers)$$

7.2 Projektion

Bei der Projektion werden einzelne Spalten einer Tabelle ausgewählt und dargestellt. Das Ergebnis ist eine Menge, d.h. doppelte Tupel werden aus dem Ergebnis entfernt.

Beispiele: Relation: Pers(PNR, Name, Gehalt, Provision, Wohnort)
Projektion, die Spalten Name und Gehalt werden selektiert:

$$\pi_{Name,Gehalt}(Pers)$$

Natürlich können Selektion und Projekttion auch kombiniert werden, da die Operationen immer eine Menge zurückgeben:

$$\pi_{Name,Gehalt}(\sigma_{Name='Lehmann' \vee Wohnort='Krauchewies'}(Pers))$$

7.3 Mengenoperationen

Auf Relationen die vereinigungsverträglich sind, können die klassischen Operationen aus der Mengenlehre angewendet werden[2].

Vereinigung (alle Elemente von R und alle Elemente von S):

$$R \cup S$$

Differenz (alle Elemente von R abzgl. derer die auch in S vorkommen):

$$R - S$$

Durchschnitt (alle Elemente die in R und S gemeinsam vorkommen):

$$R \cap S$$

Symmetrische Differenz (alle Elemente auf R und S die nicht gemeinsam sind):

$$R \Delta S = (R \cup S) - (R \cap S)$$

Beispiele:
Alle Namen der Studenten und Dozenten an der BA:

$$\pi_{Name}(Studenten) \cup \pi_{Name}(Dozenten)$$

MatNr der Studenten die noch kein Projekt ausgewählt haben:

$$\pi_{MatNr}(\text{Studenten}) - \pi_{MatNr}(\text{Projekte})$$

[2]Zwei Relationen R und S sind vereinigungsverträglich, wenn Sie die gleiche Anzahl an Attributen und die Wertebereiche der Attribute gleich sind.

7.4 Erweitertes kartesisches Produkt (Kreuzprodukt)

Das Kreuzprodukt enthält alle möglichen Paare von Tupeln zweier Relationen R und S.

Relation R

A	B	C
a	b	c
b	c	d

Relation S

D	E
d	e
e	f

$R \times S$:

A	B	C	D	E
a	b	c	d	e
a	b	c	e	f
b	c	d	d	e
b	c	d	e	f

Abbildung 33: Beispiel Kreuzprodukt

7.5 Verbund

Beim Kartesischen Produkt erhält man dadurch, dass alle möglichlichen Kombinationen gebildet werden eine extrem aufgeblähte Ergebnismenge. Beim Verbundoperator (Join) wird gleichzeitig eine Filterung vorgenommen.

7.5.1 Der Θ-Join

Beim Θ-Join werden die Datensätze miteinander verknüpft auf denen die Bedingung die unter dem Operatorsymbol ⋈ stehen. **Definition:**

$$R \bowtie_{A\Theta B} S = \sigma_{A\Theta B}(R \times S) \; mit \; \Theta \in \{<, =, >, \neq, \leq, \geq\}$$

Beispiel 1, alle Kunden mit ihren Bestellungen:

$$R \bowtie_{R.KdNr=B.KdNr} R$$

Beispiel 2, alle Stundenten, die in der Matheprüfung besser als 2 waren:

$$S \bowtie_{R.MatNr=P.MatNr \wedge P.Fach='Mathe' \wedge P.Note<2} P$$

7.5.2 Der natürliche Verbund

Beim natürlichen Verbund werden die Spalten die in beiden Tabellen den gleichen Namen haben zur Verknüpfung verwendet, die doppelte Spalte wird in der Ergebnismenge nicht angezeigt. Gibt es in den verknüpften Tabellen keine gleichnamigen Spalten, so ist die Ergebnismenge das kartesische Produkt.
Beispiel Relationen Stundent und Prüfung:

- S (MatNr, Name, ...)
- P (PNr, MatNr, Note, Fach, ...)

Die Verknüpfung der beiden Tabellen wird beim natürlichen Verbund $S \bowtie P$ über die gleichnamige Spalte MatNr durchgeführt.

Beispiel, alle MatNr. der Studenten die eine 1 in einer Prüfung haben:

$$\pi_{MatNr}(\sigma_{Note=1}(S \bowtie P))$$

7.5.3 Offene Joins

Bei den bisherigen Joins wurden Tupel die keinen Joinpartner haben nicht angezeigt.

(Full) Outer Join Beim Outer-Join bleiben die Tupel beider verknüpften Relationen erhalten.

$$R \rtimes S$$

Beispiel:

Studienrichtung

FachNr	FName
1	Informatik
2	Sport
3	Physik
4	BWL

Student

MatNr	FachNr	Name
1	3	Lehmann
2		Ginter
3	2	Späth
4	1	Dunkel

Ergebnismenge

FachNr	FName	MatNr	FachNr	Name
1	Informatik	1	1	Dunkel
2	Sport	3	2	Späth
3	Physik	1	3	Lehmann
4	BWL			
		2		Ginter

Abbildung 34: Beispiel Outer-Join

*Beim Outer-Join werden hier auch die Datensätze auf **beiden** Seiten berücksichtigt die keinen Join-Partner haben. Die Attribute die nicht belegt sind in der Ergebnismenge enthalten NULL.*

Left Join Beim Left-Join bleiben die Tupel der linken Relation erhalten, die keinen Joinpartner in der rechten Relation haben.

$$R \rtimes S$$

Beispiel:

Studienrichtung

FachNr	FName
1	Informatik
2	Sport
3	Physik
4	BWL

Student

MatNr	FachNr	Name
1	3	Lehmann
2		Ginter
3	2	Späth
4	1	Dunkel

Ergebnismenge

FachNr	FName	MatNr	FachNr	Name
1	Informatik	1	1	Dunkel
2	Sport	3	2	Späth
3	Physik	1	3	Lehmann
4	BWL			

Abbildung 35: Beispiel Left-Join

Beim Left-Join der beiden Relationen wird der Datensatz der linken Relation mit angezeigt, der Datensatz der rechten Relation der keinen Joinpartner hat wird hingegen nicht in der Ergebnismenge auftauchen.

Right-Join Der Right-Join funktioniert analog zum Left-Join, nur das hier die Tupel des rechten Joinpartners mit auftauchen, welche keinen Join-Partner haben.

$$R \ltimes S$$

Beispiel:

Studienrichtung

FachNr	FName
1	Informatik
2	Sport
3	Physik
4	BWL

Student

MatNr	FachNr	Name
1	3	Lehmann
2		Ginter
3	2	Späth
4	1	Dunkel

Ergebnismenge

FachNr	FName	MatNr	FachNr	Name
1	Informatik	1	1	Dunkel
2	Sport	3	2	Späth
3	Physik	1	3	Lehmann
		2		Ginter

Abbildung 36: Beispiel Right-Join

Der Student der keinen Studiengang besucht[3]*, taucht in der Ergebnismenge mit auf.*

8 Die Sprache SQL

8.1 Anlegen von Tabellen

Relationen werden in SQL mit dem Befehl **CREATE TABLE** angelegt.

```
CREATE TABLE Tabellenname (
     Spaltenname {Datentyp | Domänenname}
        [{NOT NULL | UNIQUE | PRIMARY KEY |
          REFERENCES Tabellenname [(Spalten)] |
          CHECK (Bedindung)}]
        [ DEFAULT {Wert | USER | NULL} ]
)
```

Jede Spalte muss mit Namen und einem Datentyp angegeben werden. Zusätzlich können weitere Parameter angegeben werden:

NOT NULL: Spalte darf nicht leer sein.

PRIMARY KEY Spalte ist Primärschlüssel, kann nur bei einer Spalte angegeben werden, zusammengesetzte Primärschlüssel müssen im Bereich der Tabellenabhängigkeiten angegeben werden.

UNIQUE: Werte der Spalte müssen eindeutig sein, keine Tupel der Tabelle weisen bei dem Attribut den gleichen Wert auf.

CHECK: Einschränkung des Wertebereiches der Spalte.

REFERENCES: Fremdschlüsselspalte und Referenz zu einer Tabelle wird festgelegt. Wird keine Spalte angegeben so wird auf den Primärschlüssel der anderen Tabelle referenziert. Datentypen beider Spalten muss übereinstimmen.

DEFAULT: Wert der als Vorgabewert für die Spalte verwendet wird. Wird beim Einfügen eines Datensatzes kein Wert angegeben, so wird dieser eingetragen.

Beispiele:

[3] ein fauler Student also

```
CREATE TABLE Fachrichtung (
    FNr     INT        PRIMARY KEY,
    NAME    CHAR(20)   NOT NULL,
)

CREATE TABLE Student (
  MatNr   INT         PRIMARY KEY,
  NAME    CHAR(20)    NOT NULL,
  EMAIL   VARCHAR(50) NOT NULL,
  FNr     INT         REFERENCES Fachrichtung(FNR)
)
```

Bei der Tabellendefinition können weitere Tabellenabhängigkeiten definiert werden, dies erfolgt nach der letzten Felddefinition der Tabelle:

```
UNIQUE | PRIMARY KEY (Spaltenname [,Spaltename]...) |
   {FOREIGN KEY (Spaltenname [,Spaltename]...)
      REFERENCES Tabelle [(Spaltenname [,Spaltename]...)]
      [ON {DELETE [UPDATE] | UPDATE [DELETE]}
          {CASCADE | SET NULL | SET DEFAULT | NO ACTION}]}|
      CHECK (Bedingung)
```

PRIMARY KEY: Bei zusammengesetzten Primärschlüsseln, können hier die Attribute als Liste angegeben werden. Auch einwertige Primärschlüssel können hier angelegt werden.

UNIQUE: Wie beim Primärschlüssel können andere Schlüsselkanditaten als Liste angegeben werden.

CHECK: Angabe von Bedingungen für ein Attribut, z.B. CHECK (Gehalt > 100)

FOREIGN KEY: Festlegung der Fremdschlüssel einer Tabelle, z.B. FOREIGN KEY FNr REFERENCES Fach (FNr). Außerdem kann angegeben werden wie beim Löschen/Ändern des referenzierten Datensatzes reagiert werden soll. Dies geschieht durch die "ON DELETE | UPDATE" Option. Siehe Seite 9.

Beispiele:

```
CREATE TABLE Gebaeude (
    GNr      INT           PRIMARY KEY,
    Name     VARCHAR(32)   NOT NULL
)

CREATE TABLE Student (
    RNr    INT,
    GNr    INT,
    NAME   CHAR(10),
    PRIMARY KEY (RNr, GNr),
    FOREIGN KEY (GNr) REFERENCES ON Gebauede(GNr) ON DELETE CASCADE
```

8.2 Änderungen an Tabellendefinitionen

Für Änderungen an der Struktur einer Tabelle dient der ALTER Befehl.

```
ALTER TABLE Tabellenname
   { ADD [COLUMN] Spaltendefinition
   | MODIFY [COLUMN] Sp {DEFAULT WERT | DROP DEFAULT}
   | DROP [COLUMN] Sp [RESTRICT | CASCADE]
   | ADD Tabellenconstraint
   | DROP CONSTRAINT constraint }
```

Beispiele:

```
ALTER TABLE Student ADD email VARCHAR(100) UNIQUE
```

Die Spalte "email" wird der Stundententabelle hinzugefügt. Vorhandene Tupel erhalten bei diesem Attribut den Wert NULL.

```
ALTER TABLE BAStudent ADD
FOREIGN KEY (FirmaID) REFERENCES Firma (ID)
```

Es wird die Spalte "FirmaID" zu einer Fremdschlüsselspalte gemacht.

```
ALTER TABLE BAStudent MODIFY COLUMN Firma DEFAULT 'EADS'
```

Die Spalte "Firma" der Tabelle "BAStudent" erhält einen neuen Defaultwert[4]

8.3 Löschen von Objekten

Zum Löschen von Objekten die mit dem CREATE-Befehl erzeugt wurden, steht der Befehl DROP zur Verfügung.

```
DROP
  { TABLE Tabellenname
  | VIEW Viewname
  | DOMAIN Domainname
  | INDEX Indexname
  | SCHEMA Schemaname}
  [ RESTRICT | CASCADE]
```

Bei der CASCADE-Option werden refenzierte Objekte mit gelöscht, z.B. ein View auf eine Tabelle. RESTRICT verhindert das Löschen, wenn das Objekt noch referenziert wird.

Beispiele:

```
DROP TABLE Studenten
DROP VIEW Stunden_EADS
```

8.4 Datenbankabfragen

Der Hauptzweck einer Datenbank besteht darin Informationen zu speichern und diese jederzeit wieder zur Verfügung zu Stellen. Der SQL-Befehl zum Abfragen von Daten aus einer Datenbank lautet SELECT.

```
SELECT [ DISTINCT | ALL]
 { *
  | { Ausdruck [[AS] SpAlias]
  | { Tabname | Viewname | TabAlias}.*
  | { Tabname | Viewname | TabAlias}.Spalte [[AS} SpAlias]}
 [, Ausdruck [[AS] SpAlias] ]
  | { Tabname | Viewname | TabAlias}.*
  ...
FROM {Tabname | Viewname} [TabAlias]
     [, {Tabname | Viewname } [TabAlias]]...
```

[4]Das Feld Firma verstößt in dieser Tabelle natürlich (je nach Aufbau) gegen die 2. oder 3. Normalform.

```
[WHERE Bedinung]
[GROUP BY Ausdruck [, Ausdruck] ... [HAVING Bedinung]]
[ORDER BY {Spalte | Position} [ASC |DESC]
    [, {Spalte | Position} [ASC | DESC]] ...]
[FOR UPDATE [OF UpdateSpalten]]
```

8.4.1 Einfache Abfragen mit SELECT und FROM

Beispiele:

```
SELECT * FROM Studenten
```

Alle Datensätze der Tabelle Studenten werden angezeigt.

```
SELECT MatNr FROM Studenten
```

Alle Matrikelnummern der Tabelle Studenten werden angezeigt.

```
SELECT DISCTINCT Name FROM Studenten
```

Das Attribut "Name" der Studententabelle wird angezeigt. Tupel der Ergebnismenge die den gleichen Namen haben werden durch das Schlüsselwort DISTINCT nicht mit angezeigt.

8.4.2 Filterung der Daten mit WHERE

Bisher wurden immer alle Datensätze einer Tabelle angezeigt, mit dem Schlüsselwort WHERE können die Datensätze der Ergebnistabelle anhang einer oder mehrere Bedingungen gefiltert werden.

Ein einfacher Vergleich hat folgenden Aufbau:

```
Operand1 Operator Operand2
```

Folgende Operatoren sind möglich:

- Spaltennamen von Tabellen/Views, welche in der FROM-Klausel vorkommen
- Konstanten
- Artithmetische Ausdrücke (Operatoren +,-,*,/ mit Klammerung, außerdem diverse Funktionen, auch für Zeichenketten und Datumsberechnungen)
- Unterabfragen

Vergleichsoperatoren sind:

- =: gleich
- >: größer
- <: kleiner
- <=: kleiner gleich
- >=: größer gleich
- !=,<>: ungleich

Beispiele:

Welche Studenten kommen aus Ulm:

```
SELECT * FROM Studenten WHERE Ort = 'ULM'
```

Welche Mat-Nr hat Student Lehmann:

```
SELECT MatNr FROM Studenten WHERE Name = 'Lehmann'
```

Welcher Mitarbeiter verdient mehr als 4000 Euro im Monat:

```
SELECT * FROM Mitarbeiter WHERE Gehalt >= 4000
```

In der WHERE-Bedingung können auch mehrere Suchkriterien vorkommen, diese können mit logischen Verknüpfungen AND, OR und NOT verbunden werden. Die höchste Priorität hat das NOT, gefolgt vom AND.

Beispiel: Welche Studenten kommen aus Krauchewies und heißen Lehmann:

```
SELECT * FROM Studenten WHERE Ort = 'ULM' AND Name = 'Lehmann
```

8.4.3 Der Bereichs-Operator BETWEEN

Der Operator BETWEEN dient zum Abfragen von Wertebereichen in SQL. Der Bereich ist inklusive der Grenzen definiert.

```
Spaltenname [NOT] BETWEEN {Spalte | Wert | Ausdruck} AND {SPALTE | Wert | Ausdruck}
```

Beispiel:
Welche Mitarbeiter verdienen zwischen 5000 und 10000 Euro.

```
SELECT * FROM Mitarbeiter
WHERE Gehalt BETWEEN 5000 AND 10000
```

Diese Abfrage ist gleichbedeutend zu:

```
SELECT * FROM Mitarbeiter
WHERE Gehalt >= 5000 AND Gehalt <=10000
```

8.4.4 Der Operator IN

Mit dem IN-Operator ist es möglich mittels einer Wertliste zu filtern. Dies entspricht einer kompakteren OR-Verknüpfung.

```
Spaltenname [NOT] IN (Wert [,Wert]...)
```

Beispiel:
Welche Studenten kommen aus Krauchewies, Schemmerhofen, Wilflingen oder Kaufbeuren:

```
SELECT * FROM Studenten
WHERE Ort IN ('Krauchewies', 'Schemmerhofen', 'Wilflingen', 'Kaufbeuren')
```

8.4.5 Zeichenkettenähnlichkeiten

Werte einer Zeichenkette können mit dem LIKE-Operator gemäß einem Muster, welche in der Zeichenkette mit Platzhaltern angegeben werden. Die Platzhalter sind hier "%" für eine beliebige (auch leere) Zeichenkette sowie "_" für ein beliebiges Zeichen.

```
Spaltenname [NOT] LIKE 'Maske' [ESCAPE 'EscapeZeichen']
```

Das Escapezeichen muss angegeben werden, wenn ein Platzhalterzeichen in der Maske verwendet werden soll. Das Escapezeichen vor einem Platzhaltersymbol gibt an, dass dieser nicht als Platzhalter verwendet werden soll.

LIKE-Klausel	**Trifft zu auf**
Name LIKE '%Lehmann%'	'Beni Lehmann', 'Lehmann', 'R. Lehmannle'
TelNr LIKE '_3%'	Telefonnummern, die an zweiter Stelle eine 0 haben (03, 1320)
Name NOT LIKE '%eli%'	Namen ohne 'eli', z.B. Felix wird nicht angezeigt.
Fach LIKE '%*_%' ESCAPE '*'	Alle Fächer mit einem Unterstrich.

8.4.6 Sortieren

Die Ergebnistabelle kann mit ORDER BY sortiert werden. Die Spalten nach denen sortiert werden soll werden nach dem Schlüsselwort angegeben. Standardmäßig wird aufsteigend sortiert, durch Angabe von DESC kann man eine absteigende Sortierung veranlassen.
Werden mehrere Spalten zur Sortierung angegeben so wird zuerst nach der ersten angegebenen Spalte sortiert, danach wird nach der zweiten Spalte sortiert (bei gleichen Werten innerhalb der ersten Sortierstufe.

```
ORDER BY {Spalte | Spaltenummer} [ASC | DESC]
           [, {Spalte | Spaltennummer} [ASC | DESC]] ...
```

Beispiele:
Abfrage welche der Fehlzeiten von Stunden, absteigende Sortierung:

```
SELECT * FROM Studenten
ORDER BY Fehlzeit DESC
```

Abfrage der Mitarbeiter mit einem Gehalt größer 3000, Sortierung Gehalt (absteigend) und Namen (aufsteigend).

```
SELECT PersNr, Name FROM Personal
WHERE Gehalt > 3000
ORDER BY Gehalt DESC, Name
```

8.4.7 Umbennung von Spalten

Spalten der Ergebnistabelle können umbenannt werden. Dazu wird nach der Spalte ein AS gefolgt vom gewünschten Namen angegeben.

Beispiel:

```
SELECT ID AS Personalnummer, VNAME as Vorname, NNAME as Nachname
FROM Personal
```

8.4.8 Verküpfung von Tabellen (Joins)

Tabellen sind oftmals über mehrere Tabellen verteilt, diese kann man gemeinsam Abfrage und zu einer Ergebnistalle zusammenfassen indem man in der FROM-Klausel mehrere Tabellen angibt. Hier kann es vorkommen, dass in den Tabellen gleichnamige Attribute vorkommen. In diesem Fall ist es notwendig bei der Attributauswahl den Tabellennamen anzugeben.

Beispiel:

```
SELECT Student.MatNr, Student.Name, Fach.Name
FROM Student, Fach
```

Durch Aliasnamen kann die Schreibweise verkürzt werden:

```
SELECT S.MatNr, S.Name, F.Name
FROM Student S, Fach F
```

Werden mehrere Tabellen, wie oben beschrieben, miteinander verknüpft so wird das kartesische Produkt gebildet, d.h. jedes Element der Tabelle mit jedem Element der anderen Tabelle verknüpft. Von daher ist die Angabe eines Verknüpfungskriteriums sinnvoll. Dies sind in der Regel Spalten die den Primär/Fremdschlüssel enthalten. Diese Bedingung wird in der WHERE-Klausel angegeben.

Beispiel:

```
SELECT S.MatNr S.Name, F.Name
FROM Student S, Fach F
WHERE S.FachNr = F.FachNr
```

Bei dieser Abfrage werden alle Stundenten mit dem ihnen zugehörigen Fachnamen angezeigt.

Weitere Beispiele:
Alle Studenten der Fachrichtung Informatik:

```
SELECT S.*, F.Name
FROM Studenten S, Fach F
WHERE S.FachNr = F.FachNr AND F.Name = 'Informatik'
```

Alle Studenten bei den der Studienort, gleich dem Sitz des Unternehmens ist:

```
SELECT S.*, U.Name
FROM Studenten S, Unternehmen U
WHERE S.UNr = U.UNr AND S.WOrt = U.Ort
```

8.4.9 Alternative Join-Formulierungen

In SQL2 gibt es alternative Join-Formulierungen.

Equi-Join

```
SELECT S.MatNr S.Name, F.Name
FROM Student S, Fach F
WHERE S.FachNr = F.FachNr
```

entspricht:

```
SELECT S.MatNr S.Name, F.Name
FROM Student S INNER JOIN Fach S ON S.FachNr = F.FachNr
```

Weiterhin sind OUTER JOINS (LEFT JOIN, RIGHT JOIN, FULL JOIN) möglich. Die Funktionalität entspricht denen der Relationenalgebra (siehe Seite 19).

8.4.10 Gruppenfunktionen

Gruppenfunktionen in SQL führen Standardberechnungen über eine Menge von Datensätzen durch. Folgende Funktionen stehen in SQL zur Verfügung:

Funktion	Beschreibung
AVG	Mittelwert
COUNT	Anzahl
MAX	Maximum
MIN	Minimum
SUM	Summe

```
{AVG | SUM} ({[DISTINCT] Spaltename} | Math.Ausdruck) |
   COUNT (*| [DISTINCT] Spaltename) |
   {MAX | MIN} (Spaltenname | Math.Ausdruck)
```

Die Funktionen dürfen nicht in der WHERE-Klausel verwendet werden, AVG und SUM sind nur auf numerische Attribute anwendbar, MIN und MAX auf alle Grunddatentypen. COUNT(*) zählt alle Zeilen, COUNT(DISTINCT Spalte) filtert Werte heraus, welche mehrfach vorkommen und zählt diese nur einmal. Hat ein Attribut eine NULL Belegung, so wird dieser Datensatz nicht mitgezählt.

Beispiele:
Durchschnittsgehalt aller Mitarbeiter:

```
SELECT AVG(Gehalt) FROM Personal
```

Anzahl aller Stunden:

```
SELECT COUNT(*) FROM Studenten
```

Anzahl aller Studenten mit unterschiedlichen E-Mail-Adresse:

```
SELECT COUNT(DISCTINCT email) FROM Studenten
```

Das höchste Gehalt ermitteln:

```
SELECT MAX(gehalt) AS HoechstesGehalt FROM Personal
```

8.4.11 Gruppenbildung

Mit der GROUP BY-Klausel erfolgt in SQL eine Gruppierung von Daten, welche gleiche Werte in einer oder mehreren Spalten haben. Mit der HAVING-Klausel kann eine Bedingung für die Gruppierung festgelegt werden.

```
GROUP BY Ausdruck [,Ausdruck]... [HAVING Bedingung]
```

Die direkt hinter der SELECT-Anweisung aufgeführen Spalten müssen auch in der GROUP BY-Klausel erscheinen. Im SELECT- und im HAVING-Teil können die beschriebenen Gruppenfunktionen eingesetzt werden.

Beispiele:
Alle Benutzer des Gästebuchdienstes mit gleichen Nachnamen:

```
SELECT nachname, count(*) as ANZAHL
FROM gb_user
GROUP BY nachname
ORDER BY ANZAHL DESC
```

Anzahl der Einträge des Gästebuchdienstes mit gleicher IP:

```
SELECT userip, Count(*) as Anzahl
FROM gb_posts
GROUP by userip
ORDER BY Anzahl DESC
```

Gästebücher mit mehr als 1000 Einträgen:

```
SELECT u.username, COUNT(*) AS Anzahl
FROM gb_user u, gb_posts p
WHERE u.id = p.uid
GROUP BY u.username
HAVING COUNT(*) > 1000
ORDER BY Anzahl DESC
```

8.4.12 Unterabfragen

In die WHERE-Klausel können neben Attributen und Konstanten auch Unterabfragen verwendet werden, welche eine Ergebnistabelle zurückliefern.

Beispiele:
Welche Studenten wohnen im gleichen Ort wie der Dozent mit der ID 7:

```
SELECT S.Name, S.MatNr
FROM Studenten S
WHERE S.Ort = (SELECT D.Ort
               FROM Dozent D
               WHERE D.ID = 7 )
```

Welche Studenten haben die beste Prüfung geschrieben:

```
SELECT S.Name, S.MatNr, S.Note
FROM Studenten S
WHERE S.Note = (SELECT MIN(P.Note)
                FROM Pruefung P)
```

Bei diesen Beispielen ist es erforderlich, dass die Unterabfrage maximal einen Ergebniszeile hat.

Liefert eine Unterabfrage mehrere Ergebniszeilen zurück, so kann ein Mengentest mit IN erfolgen. Es erfolgt dann ein Vergleich mit allen Ergebniszeilen.

Beispiel:
Welche Gästebuchbesitzer haben einen Eintrag unter der gleiche E-Mail-Adresse in Gästebüchern getätigt:

```
SELECT U.email, U.username
FROM gb_user U
WHERE U.email IN (SELECT P.email FROM gb_posts P)
```

8.4.13 Quantifizierte Bedingungen (ALL/ANY)

Der IN-Operator lässt nur Vergleiche auf Gleichheit zu. Mit den Operatoren ALL und ANY kann ein Vergleich mit den Standardoperatoren durchgeführt werden. Bei ALL muss die Bedingung auf alle Ergebniszeilen zutreffen, bei ANY reicht es, wenn die Bedindung mindestens einen Datensatz zutrifft.

Beispiel:
Finde die Manager, die mehr verdienen als alle ihre Angestellten:

```
SELECT P.PNr
FROM Pers P
WHERE P.Gehalt > ALL (SELECT M.Gehalt FROM Pers M
                      WHERE P.PNr = M.PNr)
```

8.4.14 Existenztests

Wenn eine Unterabfrage irgendwelche Datensätze zurückgibt, ist EXISTS wahr und NOT EXISTS ist falsch. In der Unterabfrage wird in der Regel SELECT * verwendet.

Beispiele:
Listet alle Dozenten auf, die keine Vorlesungen geben.

```
SELECT D.Name
FROM Dozent D
WHERE NOT EXISTS (SELECT * FROM Vorlesung
                  WHERE PersNr = D.PersNr)
```

Alle Studenten die bereits an Prüfungen teilgenommen haben:

```
SELECT S.Name
FROM Studenten S
WHERE EXISTS (SELECT * FROM Pruefungen P
              WHERE S.MatNr = P.MatNr)
```

8.4.15 NULL-Werte abfragen

Zum Abprüfen von NULL-Werten gibt es eine spezielle SQL-Anweisung:

```
IS [NOT] NULL
```

Beispiel:
Alle Studenten ohne E-Mail-Adresse:

```
SELECT S.Name, S.MatNr
FROM Studenten S
WHERE S.email IS NULL
```

8.4.16 Mengenfunktionen

Die Mengenfunktionen erlauben es zwei Ergebnistabellen miteinander gemäß der Mengenlehre zu verknüpfen. Hierfür müssen die beiden Tabellen allerdings über die gleiche Anzahl an Spalten und die Datentypen der Spalten müssen übereinstimmen. Folgende Operatoren stehen zur Verfügung:

- **UNION [ALL]:** Vereinigung der Ergebnistabellen, Duplikate werden entfernt sofern kein ALL angegeben wurde.
- **INTERSEC [ALL]:** Schnittmenge der Ergebnistabellen.
- **EXCEPT | MINUS [ALL]:** Differenzmenge der Ergebnistabellen.

Beispiel:
Welche BA-Studenten und normale Studenten kommen aus Kaufbeuren:

```
SELECT Name, MatNr FROM BAStudent WHERE Ort = 'Kaufbeuren'
UNION
SELECT Name, MatNr FROM Student WHERE Ort = 'Kaufbeuren'
```

8.5 Datenmanipulation

8.5.1 Zeile einfügen (INSERT)

Neue Datensätze werden mit dem Befehl INSERT in eine Tabelle eingefügt. Vor dem Eintragen werden die Integritätsbedingungen überprüft, nur wenn diese Prüfung erfolgreich erfolgt ist, wird der Datensatz eingetragen.

```
INSERT INTO {Tabellenname | Viewname}
    [(Spaltenname [,Spaltenname]...)]
    VALUES (Wert [,Wert]...)
```

Die Angabe der Spaltennamen ist optional, fehlen diese so müssen die Werte in Reihenfolge der Tabellendefinition übergeben werden. Werden nicht alle Attribute einer Tabelle mit Werten versehen so wird in der Tabelle bei diesen Attributen ein NULL oder Default-Wert gesetzt. Die Werte die gespeichert werden sollen, werden in der Wertliste angegeben.

Beispiel:

```
INSERT INTO Student (MatNr, Name, Ort)
VALUES (1,'Benjamin Lehmann','Friedrichshafen')

INSERT INTO Student
VALUES (1,'Benjamin Lehmann',NULL, NULL,'Friedrichshafen')
```

8.5.2 Mengen-Insert

Der Menge-Insert eignet sich zum Kopieren von Daten von einer Tabelle in eine andere.

```
INSERT INTO {Tabellenname | Viewname}
   [(Spaltenname [,Spaltenname]...)]
   SELECT-Anweisung
```

Beispiel:

```
INSERT INTO BAStudent (MatNr, Name, Ort)
  SELECT MatNr, Namr, Ort FROM Student WHERE Typ = 'BA'
```

8.5.3 Löschen von Datensätzen (DELETE)

Der Löschbefehl DELETE arbeitet Mengenorientiert, die Auswahl der zu löschenden Datensätze erfolgt mit der WHERE Bedingung[5].

```
DELETE {Tabellenname | Viewname} [Aliasname]
     [WHERE Bedingung]
```

Beispiel:
Löschen aller Datensätze in der Studententabelle:

```
DELETE FROM Studenten
```

Löschen der Studenten mit dem Namen “Benjamin Lehmann”:

```
DELETE FROM Studenten
 WHERE vname = 'Benjamin' AND nname = 'Lehmann'
```

[5]Es stehen alle Möglichkeiten wie bei SELECT zur Verfügung, inkl. Unterabfragen.

Löschen aller Studenten die keinen Kurs belegen:

```
DELETE FROM Studenten S
  WHERE NOT EXISTS (SELECT * FROM Kursbelegung K
                    WHERE K.MatNr = S.MatNr)
```

8.5.4 Ändern von Datensätzen (UPDATE)

Wie der DELETE-Befehl arbeitet auch der UPDATE-Befehl mengenorientiert. Die Angabe einer WHERE-Bedingung hier analog möglich. Mittels SET können die Spalten, die geändert werden sollen angegeben werden.

```
UPDATE {Tabellenname | Viewname} [Aliasname]
  SET (Spaltenname = Wert,
      [, Spaltenname = Wert]...)
  [WHERE Bedingung]
```

Beispiele:
Gehaltskürzung bei den BA-Studenten um 20%:

```
UPDATE BAStudent S
   SET Gehalt = Gehalt * 0.8
```

Student mit MatNr. 17 studiert jetzt BWL:

```
UPDATE Student
  SET Fach = 'BWL'
  WHERE MatNr = 17
```

8.5.5 TRUNCATE

Der TRUNCATE-Befehl löscht alle Einträge einer Tabelle. Mit DROP STORAGE wird der verwendete Speicher freigegeben. TRUNCATE ist nicht transaktionsfähig, d.h. ein TRUNCATE kann nicht mehr rückgangig gemacht werden.

```
{}
TRUNCATE TABLE Tabellenname [{DROP | REUSE} STORAGE]
```

9 Datenkontrolle

9.1 Sichten

Eine Sicht ist keine wirkliche Tabelle, sondern nur eine Definition eines Blickwinkels auf Daten, z.B. nur bestimmte Spalten einer Tabelle. Eine Sicht enthält selbst keine Daten.
Gründe für Views:

- Benutzer bekommen nur die Daten angezeigt, welche sie für eine bestimmte Tätigkeit benötigen
- Oft genutzte oder komplexe Abfragen müssen nicht immer wieder eingegeben werden, sondern werden einmal als View definiert.
- Datenschutz, bestimmte Spalten sollen nicht für jeden Benutzer angezeigt werden, z.B. Gehalt in Personaltabelle.

- Datenunabhängigkeit, auch wenn die Tabellenstruktur in der Datenbank geändert wurde, kann die Sicht so angepasst werden, dass z.B. Anwendungsprogramme weiterhin ohne Änderungen am Quellcode lauffähig sind.

```
CREATE VIEW Viewname [(Spaltenliste)]
   AS SELECT Anweisung
   [WITH CHECK-OPTION]
```

Die Spaltenliste enthält die Spalten, welche in der Sicht angezeigt werden sollen, es können auch eigene Spalten definiert werden, z.B. für arithmetische Berechnungen. Ist eine Spalte nicht in der Spaltenliste angegeben so wird der Spaltenname aus der SELECT-Anweisung übernommen.
Die SELECT-Anweisung bietet alle bisher bekannten Möglichkeiten, bis auf ORDER BY und UNION.
CHECK OPTION bewirkt, dass INSERT- und UPDATE-Operationen nur dann zugelassen sind, wenn diese die Bedingung des Views erfüllen (d.h. nach dem Einfügen/Ändern innerhalb des Views angezeigt werden würden). INSERT- und UPDATE-Operationen sind bei Sichten nur zulässig wenn:

- sich der View nur über eine Tabelle erstreckt, d.h. kein JOIN zulässig
- die SELECT-Anweisung keine Unterabfrage enthält.
- DISTINCT, GROUP BY und HAVING in der Sichtdefinition nicht vorkommen
- sich nur Spalten in der Spaltenliste befinden, die ihren Ursprung in der Basistabelle haben
- die SELECT-Anweisung kein DISTINCT enthält

Beispiel:

```
CREATE VIEW Durchgefallene_Studenten
   (MatNr, Name, Fach) AS
   SELECT S.MatNr, S.Name, S.Fach FROM Studenten S
   INNER JOIN Pruefung P ON S.MatNr = P.MatNr
   WHERE P.Note > 4.0
```

Löschen der Sicht:

```
DROP VIEW Durchgefallene_Studenten
```