

Q2 - Datenbanken

In der Q2 beschäftigen wir uns mit [Datenbanken](#). Auch in diesem Halbjahr spielt [UML](#) wieder eine Rolle, diesmal mit einer anderen Form von Diagrammen.



Inhaltsverzeichnis

Literaturhinweise.....	4
Tools zum Zeichnen von ER-Diagrammen.....	4
Das 3-Schichten- / 3-Ebenen- / 3-Sichten-Modell.....	5
Das Entity-Relationship-Modell.....	5
Entitytyp.....	5
Relationship.....	5
Grafische Darstellung eines ERM.....	6
Geschäftsregeln.....	6
Primärschlüssel.....	6
Kardinalitäten.....	6
Optionalitäten.....	7
Beispiele: ERM zur Schülerdatenbank.....	7
Übungen zu Kardinalitäten.....	8
Aufgabe zum ERM.....	9
Das relationale Datenbankmodell.....	10
Relationen.....	10
Relationale Algebra.....	11
Übertragen des ER-Modells in das relationale Datenbankmodell (RDM).....	18
Grundregeln.....	18
Normalisierung.....	20
Anomalien.....	20
Fachbegriffe zur Normalisierung.....	21
Nullte Normalform (0NF).....	21
Erste Normalform (1NF).....	21
Zweite Normalform (2NF).....	21
Dritte Normalform (3NF).....	21
Boyce-Codd-Normalform (BCNF).....	21
Beispiel 1. NF.....	22
Beispiel 2. NF.....	22
Beispiel 3. NF.....	23
Weitere Aufgaben zur Normalisierung.....	24
SQL - Structured Query Language.....	26
SELECT-Abfragen.....	26
Einfügen.....	27
Ändern.....	27
Löschen.....	28
Verwaltung.....	28
Sonstiges.....	29
Joins.....	31

Funktionen.....	32
Mathematische Funktionen.....	33
Logische Operatoren.....	33
Sonstige Funktionen.....	34
Datums-Funktionen.....	36
Gruppenfunktionen.....	37
Links.....	37
Anhang.....	38

Literaturhinweise

- Datenbank-Theorie und -Praxis: <http://reeg.junetz.de>
- SQL: <http://de.wikipedia.org/wiki/SQL>
- Normalisierung: http://de.wikipedia.org/wiki/Normalisierung_%28Datenbank%29
- [SQLite und Java](#)

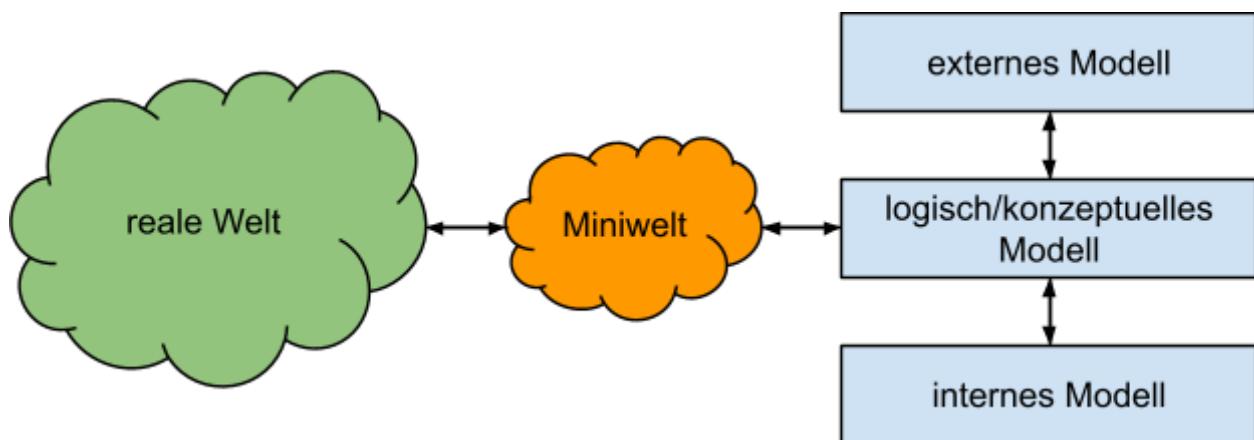
Tools zum Zeichnen von ER-Diagrammen

- Ein hilfreiches Tool zum Zeichnen von UML-Diagrammen ist "[yEd](#)".
- Das Linux-Programm "[umbrello](#)" kann auch UML-Diagramme zeichnen und sogar Quellcode daraus erzeugen.
- In Google Drive gibt es auch Extensions zum Erstellen von UML.

Das 3-Schichten- / 3-Ebenen- / 3-Sichten-Modell

Bevor man an die praktische Umsetzung einer Datenbankanwendung geht, sollte man verschiedene Stufen durchlaufen.

Von dem Problem der **realen Welt** geht man aus, erstellt daraus die sogenannte **Miniwelt**. Diese enthält nur die für die Softwareumsetzung relevante Daten und Informationen.



Aus der **Miniwelt** wird dann das **logisch/konzeptuelle Modell** aufgebaut. Dabei werden die Daten und Informationen nach ihren logischen Zusammenhängen angeordnet. Hilfsmittel zur Visualisierung ist auch hier UML.

Ausgehend von diesem Modell erstellt man dann meist zunächst das **interne Modell**. Dieses Modell beschreibt die praktische Umsetzung auf der Hardware bzw. im DBMS.

Als letzten Punkt erstellt man das **externe Modell**. Dies ist die Darstellung in der Anwendung. Dabei ist nicht die grafische Oberfläche gemeint, sondern die Abfragen, die eine spezielle Anwendung an die Datenbank hat.

Rücksprünge zwischen allen Elementen des 3-Schichten-Modells sind natürlich erlaubt.

Das Entity-Relationship-Modell

(ERM, nach P. P. Chen 1976)

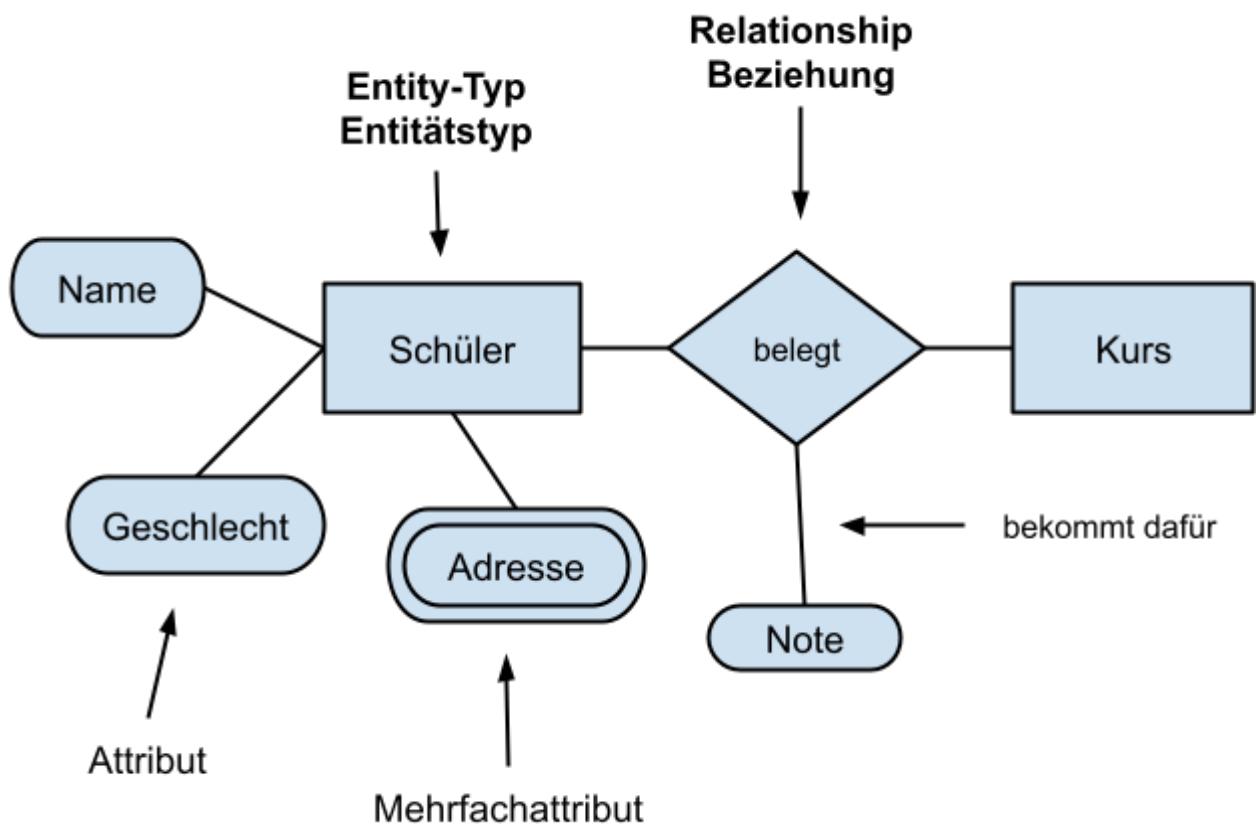
Entitytyp

Eine **Entität** ist ein real oder virtuell existierendes Objekt einer Menge, die gleichartige Entitäten enthält. Die Menge nennt man **Entitätstyp**. **Attribute** beschreiben eine Entität genauer.

Relationship

Stellt eine Beziehung zwischen zwei oder mehreren Entitäten dar. Eine Beziehung kann durch Attribute genauer definiert werden.

Grafische Darstellung eines ERM



Geschäftsregeln

- definieren die Entitäten und Beziehungen genauer
- sind z.B. für die GUI-Entwicklung wichtig
- beschreiben Regeln (z.B. müssen für einen Schüler ein Name und Geburtsdatum existieren)

Primärschlüssel

Ein oder mehrere Attribute, die die einzelnen Entitäten eines Typs eindeutig definieren.

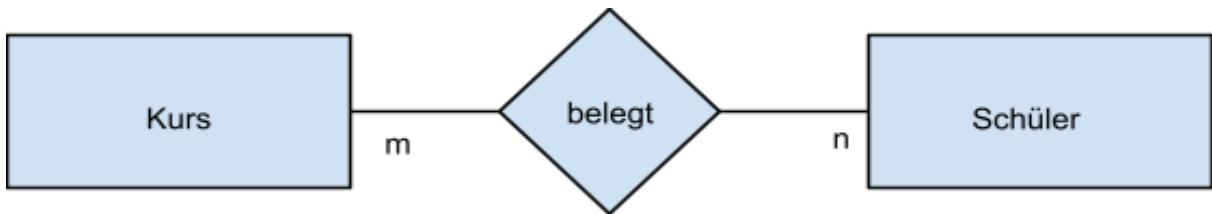
Im ERM werden Primärschlüssel-Attribute einfach unterstrichen.

Kardinalitäten

Beschreiben die Mengenverhältnisse zweier, an einer Relation beteiligten Entitäten genauer. Sie sind für die Umsetzung in die **interne Ebene** wichtig.

Die 3 Typen von Kardinalitäten

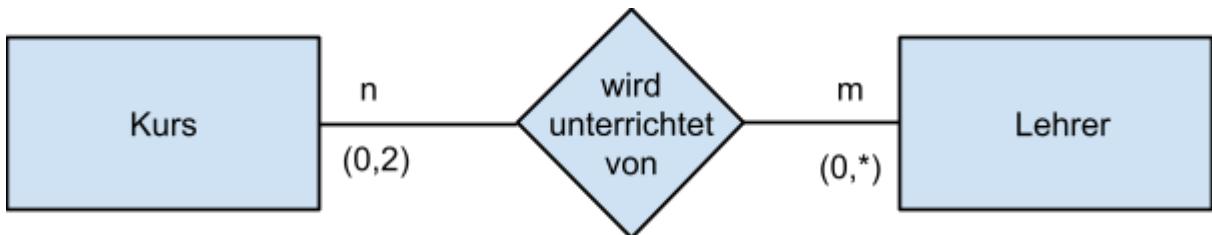
- 1-1: eins zu eins-Beziehung, Beispiel: Mannschaft hat Kapitän
- 1-n: eins zu viel-Beziehung, Beispiel: Lehrer ist Tutor von Schüler
- n-m: viele zu viele-Beziehung, Beispiel: Schüler belegt Kurs



Optionalitäten

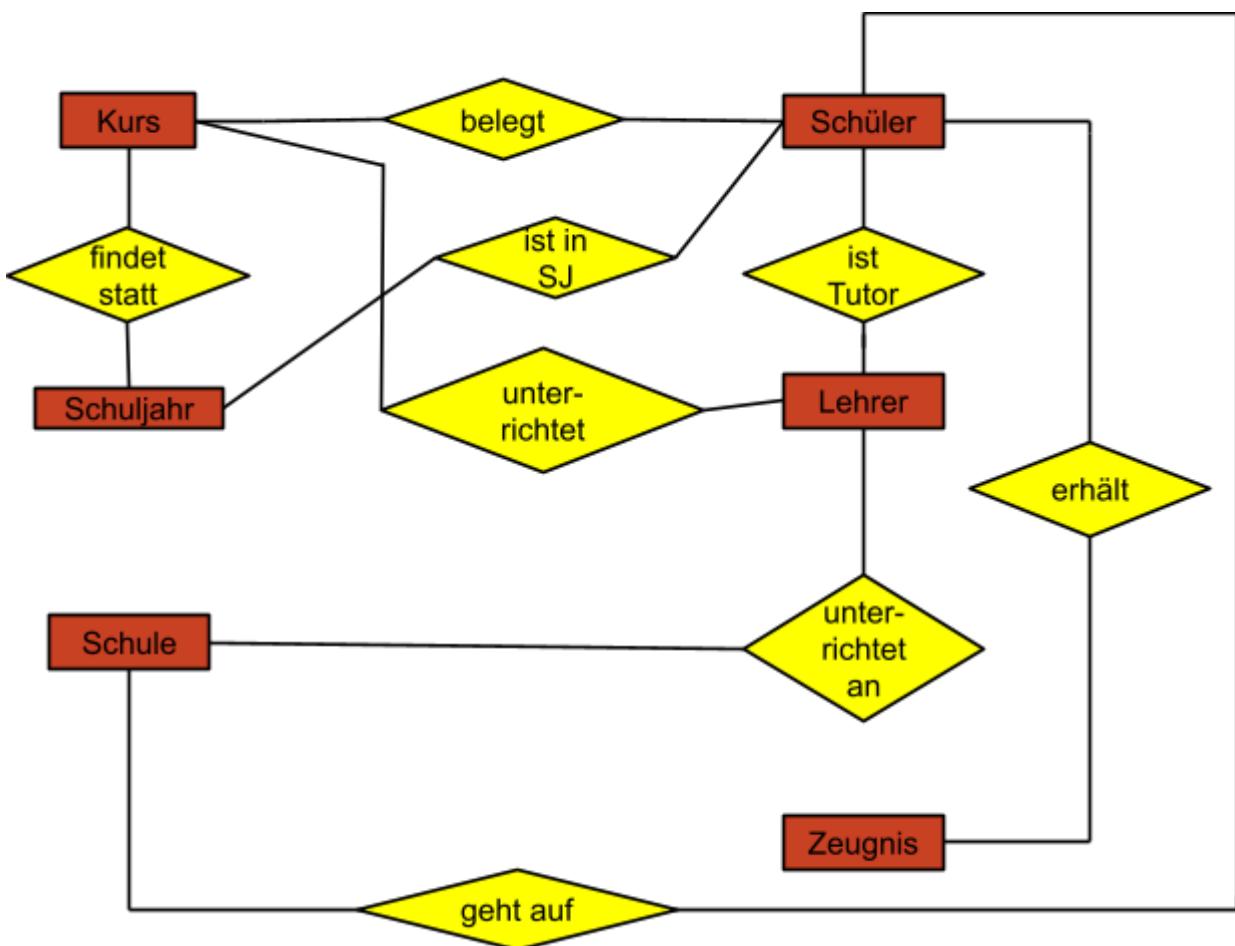
Spezifizieren die Mengenverhältnisse genauer. Sie sind für die **externe** Ebene wichtig.

Man schreibt sie oft als **min/max**-Angaben.

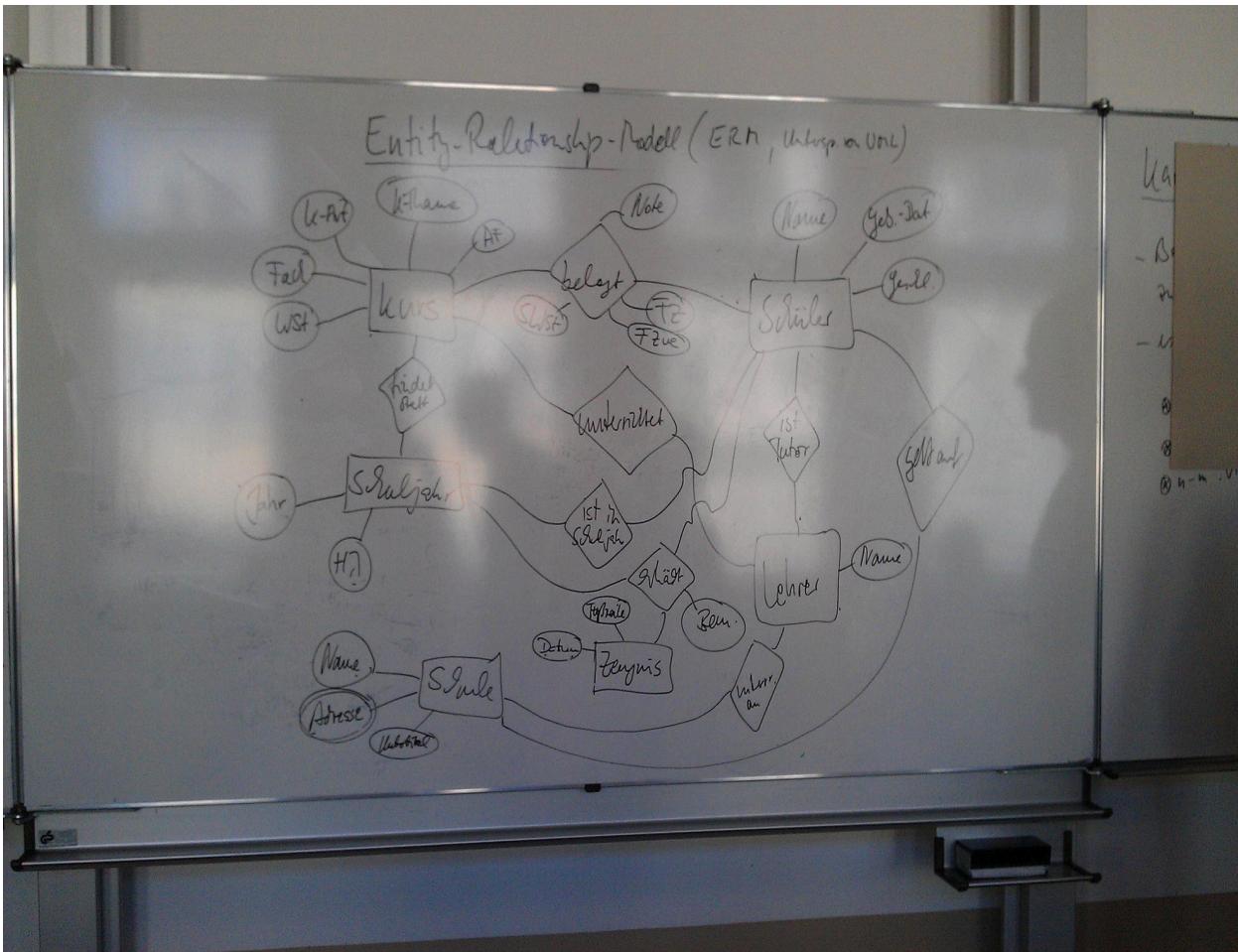


Beispiele: ERM zur Schülerdatenbank

Beispiel 1



Beispiel 2



Übungen zu Kardinalitäten

Welche Kardinalitäten haben folgende Beziehungen?

Notieren Sie bei unklaren Beziehungen eine entsprechende Geschäftsregel.

- Schüler HAT Tutor
→ n:1
- Schüler BEKOMMT HEUTE Zeugnis
→ 1:1, wenn Zeugnis individuell ist
→ n:1, wenn Zeugnis nicht individuell ist
- Schüler DARF ARBEITEN AN Computer
→ n:m
- Schüler HAT AUSGELIEHEN Buch
→ n:m, wenn Buch nicht eindeutig ist oder/und eine Historie gespeichert werden soll
→ 1:n, wenn Buch eindeutig und keine Historie
- Schüler IST BEFREUNDET MIT Schüler
→ n:m
- Vater HAT Tochter
→ 1:n
- Fuß HAT Zehe
→ 1:n

- Onkel HAT Neffe
→ n:m
- Schüler HAT UNTERRICHT BEI Lehrer
→ n:m
- Person BESITZT dt. Personalausweis
→ 1:1
- Java-Klasse BESITZT Konstruktor
→ 1:n
- Bruder HAT Schwester
→ n:m
- Ort HAT KÜRZESTE ENTFERNUNG ZU Ort
→ n:m

Aufgabe zum ERM

Erstellen Sie ein ERM zu folgender Situation:

Karteikarten des Wetenheimer Zoos

Pfleger/in

Personalnummer: 204
Name: Sabine Weidenbrück
Geb. Datum: 10.05.1961
Anschrift: Ulmenweg 5
65183 Wiesbaden
Tel.: 0611-212-35333
Eintrittsdatum: 05.11.2006
Gehalt: 2.500€

Pflegt die Tiere: Wolke, Axel
und alle Krokodile

Betreut die Gehege: Rentiergehege, Vivarium

Gehege

Name: Rentiergehege
Fläche: 250m²
Baujahr: 1912
Tiere: Dasher, Dancer, Prancer,
Vixen, Comet, Cupid,
Donner, Blitzen, Rudolph2,
Rudolph3

Tier

Name: Schneeweißchen
Art: Eisbär
Geboren: 13.10.2007
Geschlecht: weiblich
Zugangsdatum: 13.10.2007
Abgangsdatum:

Gehege: Eisbärenarena
PflegerIn: Weidenbrück, Herwig, Runkel
Futteranweisung: Milch (200ml, 15:00 Uhr)
Fleisch (5kg, 17:00 Uhr)

Art

Bezeichnung: Eisbär
Lateinische
Bezeichnung: Ursus maritimus
Lebensraum: Polarregion

[zur möglichen Lösung](#)

Das relationale Datenbankmodell

Relationen

Eine Relation ist eine Menge von Tupeln mit einer Menge von Attributen.

Identische Tupel kommen hierbei nicht vor.

(Mathematisch lässt sich das so erklären, dass die Relation sich als kartesisches Produkt der Wertebereiche der Attribute darstellen lässt. Dadurch ist jedes mögliche Tupel definiert. Die Relation definiert dann nur noch, ob das Tupel vorhanden ist oder nicht.)

Attribut

Ein Attribut definiert einen Wertebereich und den zugehörigen Attributnamen.

Bsp.: Name, Ort → String, String

Tupel

Ein Tupel ist ein Datensatz mit konkreten Attributwerten.

Bsp.: ("Müller", "Gießen")

Relationenschema

Das Schema einer Relation (also nur die Attribute, nicht die Tupel) kann so dargestellt werden:

Relation (Attribut, Attribut, ...)

Bsp.: wohntIn(Name, Ort)

Der **Grad** einer Relation ist die *Anzahl der Attribute*.

Um die Zusammenhänge zu verdeutlichen, kann man Relationen mit Tabellen vergleichen, die Tupel wären dann die Zeilen und die Attribute die Spalten.

Relationale Algebra

Durchschnitt

Der Durchschnitt $N = R \cap S$ ist die Menge an Tupeln, die sowohl in der Relation R als auch in der Relation S vorkommen.

Voraussetzung für diese Operation ist, dass die Attribute der Relationen R und S übereinstimmen, dies nennt man auch **Vereinigungsverträglichkeit**.

Beispiel:

R

A	B
1	2
2	5
3	1

S

A	B
2	5
1	4

$$N = R \cap S$$

A	B
2	5

Vereinigung

Die Vereinigung $N = R \cup S$ ist die Menge an Tupeln, die entweder in R oder in S oder in beiden vorkommen.

Voraussetzung hierfür ist wie bei einem Durchschnitt, die Vereinigungsverträglichkeit.

Beispiel:

R

A	B
1	2
2	5

S

A	B
2	6
1	4

$$N = R \cup S$$

A	B
1	2
2	5
2	6
1	4

Differenz

Eine Differenz $N = R \setminus S$ ist die Menge an Tupeln, die zwar in R , aber nicht in S vorkommt.

Die Vereinigungsverträglichkeit von R und S ist auch hier nötig.

Beispiel:

R

A	B
1	2

<u>2</u>	<u>5</u>
3	1

S

A	B
<u>2</u>	<u>5</u>
1	4

$$N = R \setminus S$$

A	B
1	2
3	1

(Kartesisches) Produkt

Das Produkt $N = R \times S$ enthält jede Kombination aller Tupel von R und S .

Das kartesische Produkt wird oft auch als *Kreuzprodukt* bezeichnet.

Beispiel:

R

A	B
1	2
2	5

S

C	D	E
2	4	6
1	4	7

$$N = R \times S$$

A	B	C	D	E
1	2	2	4	6
1	2	1	4	7
2	5	2	4	6
2	5	1	4	7

Division

Die Division $N = R \div S$ enthält alle Tupel aus R , die in jeder Kombination mit den Tupeln aus S vorkommen.

Beispiel:

R

A	B	C	D
1	2	2	5
1	2	2	4
4	5	1	4
3	4	2	5
4	5	2	5

S

C	D
2	5
1	4

$$N = R \div S$$

A	B
4	5

Selektion

Die Selektion $N = \sigma_{\text{Bedingung}}(R)$ enthält alle Tupel aus R , für die die Bedingung zutrifft.

Beispiel:

R

A	B
1	2
2	5
1	1

$$N = \sigma_{A=1}(R)$$

A	B
1	2
1	1

Projektion

Die Projektion $N = \pi_{Attribut, Attribut}(R)$ enthält alle Tupel aus R , jedoch nur deren aufgeführte Attribute.

Beispiel:

R

A	B
1	2
2	5
1	1

$$N = \pi_A(R)$$

A
1
2
1

Join

Der Join $N = R \bowtie_{\text{Bedingung}} S$ führt nacheinander die Operationen kartesisches Produkt, Selektion und Projektion aus.

Es gibt verschiedene typische Join-Arten:

- **Equi Join**

Beim Equi Join lautet die Bedingung, dass bestimmte Spalten gleichen Inhalt haben müssen.

- **Natural Join**

Der Natural Join ist ähnlich dem Equi Join, bloß dass noch eine Projektion hinzukommt, die jeweils eine der doppelten Spalten ausblendet.

- **Outer Join**

Der Outer Join führt auch die Tupel der linken (Left-Outer-Join), rechten (Right-Outer-Join) oder beiden Relationen (Outer-Join oder auch Full-Outer-Join) auf, die keine Entsprechung in der anderen Relation finden. Die fehlenden Werte werden mit Nullwerten aufgefüllt. Hierbei kann noch die Projektion des Natural Join angewendet werden (Natural Outer Join).

- **Self Join**

Die Tabelle joint sich selbst.

Beispiele:

R

A	B
1	2
2	5
3	5

S

A	C
2	4
1	4
4	2

Equi Join

$N = R \bowtie_{R.A = S.A} S$

R.A	B	S.A	C
-----	---	-----	---

1	2	1	4
2	5	2	4

Natural Join

$$N = R \bowtie_{R.A = S.A} S$$

A	B	C
1	2	4
2	5	4

Full Outer Join

$$N = R \bowtie_{R.A = S.A} S$$

R.A	B	S.A	C
1	2	1	4
2	5	2	4
3	5	Null	Null
Null	Null	4	2

Umbenennung

Die Umbenennung $N = \rho_{[neu \leftarrow alt]}(R)$ ($\rho = ro$) gibt die Relation R zurück, bloß mit einer Änderung des Namens eines Attributs alt auf neu .

Beispiel:

R

A	B
1	2
2	5
1	1

$$N = \rho_{[C \leftarrow A]}(R)$$

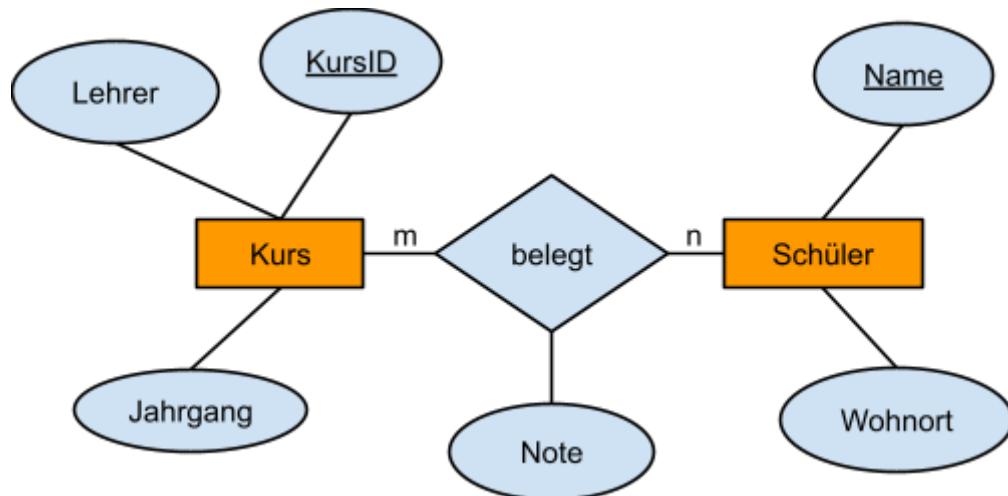
C	B
1	2
2	5
1	1

Übertragen des ER-Modells in das relationale Datenbankmodell (RDM)

Grundregeln

- Jeder Entitätstyp wird zu einer eigenen Relation.
- Jeder Beziehungstyp wird in ein eigenes Relationenschema abgebildet. Die Attribute des Relationenschemas sind die Primärschlüssel der beiden beteiligten Entitätstypen zusätzlich der beziehungseigenen Attribute.

Beispiel:



wird zu diesem Relationenschema:

- Kurs (KursID, Lehrer, Jahrgang)
- Schüler (Name, Wohnort)
- belegt (\uparrow Schüler.Name, \uparrow Kurs.KursID, Note)

Fremdschlüssel

Als Fremdschlüssel bezeichnet man aus anderen Relationen importierte Primärschlüsselelemente. Sie werden im Relationenschemata meist gestrichelt unterstrichen oder mit einem kleinen Pfeil (\uparrow) gekennzeichnet. Fremdschlüssel können in der Relation auch zum Primärschlüssel gehören.

Primärschlüssel der Beziehungsrelation

Je nach der Kardinalität der Beziehung im ERM, sind verschiedene Schlüsselelemente für die Beziehungsrelation notwendig.

Für die Beziehungsrelation $R(a, b)$ von $A(\underline{a}, c)$ und $B(\underline{b}, d)$ gilt:

Bei einer **n:m - Beziehung** müssen a und b gemeinsam der Primärschlüssel sein, da jeder Wert für sich genommen mehrfach vorkommen kann. Nur gemeinsam sind sie i.d.R. eindeutig.

Bei 1:1 und 1:n Beziehungen kann die Beziehungstabelle oder sogar eine der beiden Entitätstabellen ganz wegfallen. Die Primärschlüssel bzw. alle Attribute der einen Entitätstabelle werden dann in die andere "importiert":

Bei einer **1:1 - Beziehung** kann entweder a oder b ein Primärschlüssel sein, da für beide Attribute gilt, dass ein Wert (höchstens) einmal in der Relation vorkommt. Es hängt vom Kontext ab, in welcher Relation der Fremdschlüssel notiert wird. Man kann aber auch beide Relationen vereinen.

Beispiele für eine 1:1-Beziehung:

Person hat Personalausweis

Ausgangsrelationen:

- Person (PNr, Name, Vorname, ...)
- hat (\uparrow PNr, \uparrow PANr)
- Personalausweis (PANr, Datum, ...)

Neue Relationen:

- Person (PNr, Name, Vorname, Trikotnummer, ..., PANr, Datum, ...)

Die Zusammenfassung der Relationen gelingt dann, wenn die Entitätstypen abhängig voneinander sind. Eine Person muss einen Personalausweis haben.

Mannschaft hat Kapitän

Ausgangsrelationen:

- Mannschaft (MNr, Name, Trikotfarbe, ...)
- hatKapitaen (\uparrow MNr, \uparrow SNr)
- Spieler (SNr, Name, Vorname, Trikotnummer, ...)

Neue Relationen (Variante 1):

- Mannschaft (MNr, Name, Trikotfarbe, ..., \uparrow SNr)
- Spieler (SNr, Name, Vorname, Trikotnummer, ...)

Neue Relationen (Variante 2):

- Mannschaft (MNr, Name, Trikotfarbe, ...)
- Spieler (SNr, Name, Vorname, Trikotnummer, ..., ↑MNr)

Variante 2 hat den Nachteil, dass so bei den meisten Spielern in dem Feld MNr Nullwerte stehen würden, da nur wenige Spieler Kapitän sein können. Zudem würde es auch nicht funktionieren, wenn das DBMS keine Nullwerte für Fremdschlüssel zulässt.

Hier gelingt die Zusammenfassung der Relationen zu einer einzigen nicht, da ein Spieler kein Kapitän sein muss.

Bei einer **1:n - Beziehung** muss *a* der Primärschlüssel sein, da *b* mehrfach vorkommen kann.

Beispiel für eine 1:n-Beziehung:

Lehrer ist Tutor von Schüler

Ausgangsrelationen:

- Schüler (Schüler-Nr, Name, Vorname, Geburtsdatum, Adresse, Schuleintrittsdatum)
- istTutor (↑Schüler-Nr, ↑Lehrer-Nr, Tutorgruppe) → Tutorengruppe müsste evtl. auch ein Schlüssel sein, falls ein Lehrer mehrere Tutorengruppen haben kann, wir gehen aber mal von n:1 aus.
- Lehrer (Lehrer-Nr, Name, Vorname, Titel, Fächer)

Neue Relationen:

- Schüler-Tutor(Schüler-Nr, Name, Vorname, Geburtsdatum, Adresse, Schuleintrittsdatum, ↑Lehrer-Nr, **Tutorgruppe**)
- Lehrer(Lehrer-Nr, Name, Vorname, Titel, Fächer)

[Video dazu](#)

Normalisierung

([Präsentation zu Normalisierung](#))

Die Normalisierung ist die Änderung eines Schemas, so dass bestimmte Regeln erfüllt werden.

Ziele sind

- das Verhindern von **Anomalien**.
- die Vermeidung von Redundanz (mehrfach gespeicherte Informationen)
- strukturiertes und übersichtliches Design

Anomalien

Anomalien sind Fehler im Design der Datenbank, durch die Inkonsistenzen oder fehlende Informationen entstehen können.

Beispiel-Relationen:

- Mitarbeiter (PersNr, Name, Vorname, Beruf, Abteilung, AbtNr)
- Projekt (ProjNr, Projekt, Leiter)
- Arbeitet_an (ProjNr, PersNr, Telefon, Zeit)

Einfüge-Anomalie

Ein neuer Mitarbeiter ist noch keiner Abteilung zugeordnet, obwohl dies zwingend vorgesehen ist.

Löschen-Anomalie

Ein Projekt wird gelöscht. Dabei werden auch die Telefonnummern gelöscht.

Änderungs-Anomalie

Eine Projektleiterin ändert durch Heirat ihren Namen. In der Relation *Mitarbeiter* wird er geändert, im *Projekt* jedoch nicht.

Fachbegriffe zur Normalisierung

Funktionale Abhängigkeit

Ein Attribut B ist von A funktional abhängig, wenn zu jedem Wert von A genau ein Wert von B gehört.

Transitive Abhängigkeit

Ist ein Attribut B funktional abhängig von A und C ist funktional abhängig von B, dann ist auch C funktional abhängig von A, man sagt, C ist transitiv abhängig von A.

Nullte Normalform (0NF)

Jede Tabelle liegt zunächst in der nullten Normalform vor. Es ist sozusagen die Basis allen Tuns. Speziell schaut man aber nach Folgendem:

In der nullten Normalform kann es Zellen geben, in denen mehrere Werte z.B. durch Kommata getrennt stehen und somit eine eindeutige Zuordnung erschwert wird bzw. unmöglich sein kann.

Erste Normalform (1NF)

Die Bedingung für die erste Normalform lautet, dass **nur "atomare" Werte** auftreten. Das bedeutet, dass in jeder Zelle der Tabelle nur einzelne Werte, die nicht weiter gespalten werden können, und keine Wertemengen vorkommen. Weiterhin muss der **Primärschlüssel** vorhanden sein.

Zweite Normalform (2NF)

Die zweite Normalform ist gegeben, wenn die *erste Normalform* vorhanden ist und zusätzlich die Bedingung erfüllt ist, dass **jedes Attribut vom gesamten Primärschlüssel funktional abhängig ist** und nicht von nur einem Teil des Schlüssels. Das bedeutet, dass in der Relation R(A, B) für jedes A genau ein B existiert.

Dritte Normalform (3NF)

Eine Relation befindet sich in der 3NF, wenn sie sich in der 2NF befindet und zusätzlich gilt, dass **keine transitiven Abhängigkeiten** zwischen **Nicht-Schlüssel**-Attributen existieren.

Boyce-Codd-Normalform (BCNF)

Eine Relation befindet sich in der BCNF, wenn sie sich in der 3NF befindet und zusätzlich gilt, dass

keine transitiven Abhangigkeiten zwischen **Schlssel**-Attributen existieren.

Beispiel 1. NF

Keine/Nullte Normalform:

PersNr	Name	AbtNr	ProjNr
123	Mller	5	6, 3

Die erste Normalform ist nicht gegeben, da ProjNr mehrere Werte erhalten kann, sogenannte *Wiederholungsgruppen*. Deshalb mussen wir das geschachtelte Attribut in eine eigene Relation auslagern.

Erste Normalform:

PersNr	Name	AbtNr	ProjNr
123	Mller	5	6
123	Mller	5	3

Zweite Normalform (und auch direkt schon dritte NF):

PersNr	Name	AbtNr
123	Mller	5

↑PersNr	ProjNr
123	6
123	3

Beispiel 2. NF

Erste Normalform:

PersNr	Name	AbtNr	Abteilung	ProjNr	Projekt	Zeit
123	Mller	5	EDV	6	Novell	50%
123	Mller	5	EDV	3	DV2000	50%

876	Schulze	3	Personal	3	DV2000	100%
-----	---------	---	----------	---	--------	------

Name, AbtNr und Abteilung sind funktional abhängig von **PersNr**.

Für eine **PersNr** können allerdings mehrere **ProjNr** und **Projekt** vorhanden sein, diese müssen also ausgegliedert werden.

Projekt ist dagegen funktional abhängig von **ProjNr**, diese Attribute bieten sich also für eine eigene Relation an.

Zeit ist sowohl von **PersNr** als auch von **ProjNr** abhängig, hierfür wird also ebenfalls eine eigene Relation benötigt.

Zweite Normalform:

Mitarbeiter (PersNr, Name, AbtNr, Abteilung)

Projekt (ProjNr, Projekt)

Arbeitet an (↑PersNr, ↑ProjNr, Zeit)

Beispiel 3. NF

Zweite Normalform:

Mitarbeiter (PersNr, Name, AbtNr, Abteilung)

Abteilung ist von dem Attribut *AbtNr* abhängig, welches kein Schlüsselattribut ist. Dies widerspricht der 3NF.

Dritte Normalform:

Mitarbeiter (PersNr, Name, ↑AbtNr)

Abteilungen (AbtNr, Abteilung)

Abteilung wurde in eine eigene Relation ausgegliedert und hängt dort nur noch vom Schlüssel ab. Die dritte Normalform ist somit erreicht.

Weitere Aufgaben zur Normalisierung

Aufgaben zur Normalisierung

11.

Ein Hochschulberater berät Studenten, die alle im Wohnheim der Universität leben und alle das gleiche Hauptfach studieren. Aus Besprechungen mit Studenten will der Berater eine kleine Datenbank zur Unterstützung der Beratung entwickeln. er legt folgende Attribute und Regeln fest:

SNr	Studentennummer, ganze Zahl, eindeutiger Schlüssel für Studenten
SName	Name des Studenten, nicht eindeutig
ZNr	Zimmernummer ganzzahlig; jeder Student hat ein Zimmer, das aber von mehreren Studenten bewohnt werden kann
TNr	Telefonnummer des Studenten; sie ist für alle Zimmerbewohner gleich
Kurs	Identifikationsnummer der Kurse, die ein Student belegt oder bereits abgeschlossen hat
Semest	Semester, in dem ein Kurs abgeschlossen wurde; ein Student darf den gleichen Kurs in einem späteren Semester wiederholen
Note	Note eines abgeschlossenen Kurses

Der Berater möchte folgende Daten in der Datenbank speichern (Auszug):

SNr	SName	ZNr	TNr	Kurs	Semest	Note
3215	Jonas Mike	120DH	2136	MAT122	W88	1,4
				PHY120	S88	2,5
				WIW330	W89	3,1
				MAT122	S89	1,2
3456	Schmidt Klaus	237VH	3127	MAT122	W87	3,2
				MAT130	S87	2,9
4576	Neider Paul	120DH	2136	PHY230	W88	2,8
				MAT120	S88	2,1

- Erstellen Sie ein Diagramm zum ER-Modell für diese Daten!
- Machen Sie aus der obigen Tabelle eine gültige Relation in der 1. Normalform. Welche Anomalien können dabei auftreten?
- Erstellen Sie anhand des ER-Modells und der Umsetzungsregeln das relationale Modell! Begründen Sie, inwieweit die Tabellen die 3. Normalform erfüllen!
- Geben Sie die relationalen Operationen an, um Kurslisten auszugeben!

CD Lieder			
CD_ID	Album	Titelliste	Gründungsjahr
4811	Anastacia - Not That Kind	{1. Not That Kind, 2. I'm Outta Love, 3. Cowboys & Kisses}	1999
4712	Pink Floyd - Wish You Were Here	{1. Shine On You Crazy Diamond}	1965

SQL - Structured Query Language

SQL ist eine standardisierte Sprache zum Umgang mit relationalen Datenbanken.

Typische DBMS, die SQL implementieren, sind MariaDB, PostgreSQL, MySQL oder SQLite.

Ein gutes, interaktives Tutorial zu SQL gibt es hier: <https://sql-tutorial.de/home/start.php>

SELECT-Abfragen

*SELECT * FROM tabelle;*

Gibt alle Tupel aus der Tabelle zurück.

SELECT spalte1, spalte2 FROM tabelle;

Gibt *spalte1* und *spalte2* aller Tupel der Tabelle zurück. (→ Projektion)

SELECT DISTINCT spalte1 FROM tabelle;

Gibt *spalte1* aller Tupel der Tabelle zurück, doppelte Einträge werden hierbei nur einfach angezeigt.

SELECT spalte1 AS name, spalte2 AS vorname FROM tabelle;

Gibt *spalte1* und *spalte2* aller Tupel der Tabelle zurück, allerdings heißen die ausgegebenen Attribute nicht mehr *spalte1* und *spalte2* sondern *name* und *vorname*. (→ Umbenennung)

SELECT t1.spalte1, t1.spalte2 FROM tabelle t1 WHERE t1.spalte1=5;

Tabellen kann man auch einen Aliasnamen vergeben. Diesen schreibt man einfach im FROM-Teil hinter die entsprechende Tabelle. Dann kann man im SELECT- und WHERE-Teil damit arbeiten.

*SELECT * FROM tabelle ORDER BY attribut ASC;*

Gibt alle Tupel der Tabelle, nach *attribut* sortiert, zurück. ASC gibt vor, dass aufsteigend sortiert wird.

Das Gegenstück wäre *DESC*.

*SELECT * FROM tabelle WHERE bedingung;*

Gibt alle Tupel der Tabelle zurück, bei denen die Bedingung erfüllt ist. (→ Selektion)

SELECT tabelle1.attributID, tabelle1.attribut1, tabelle2.attribut1 FROM tabelle1, tabelle2 WHERE tabelle1.attributID = tabelle2.attributID;

Gibt die angegebenen Spalten aus den Tabellen *tabelle1* und *tabelle2* zurück, für die das Attribut *attributID* gleich ist. (→ Join)

`SELECT * FROM tabelle LIMIT [offset, rows];`

Gibt eine selbst ausgewählte Anzahl an Zeilen einer Tabelle wieder. Mit offset gibt man die Startzeile an (Bsp: Möchte man bei der 6 Zeile beginnen, so muss offset gleich 5 gewählt werden.) Mit rows gibt man die Anzahl der ausgegebenen Zeilen an.

Einfügen

Syntax

`INSERT INTO tabelle (attribut1Name, attribut3Name, attribut2Name) VALUES (attribut1Wert, attribut3Wert, attribut2Wert);`

Fügt in *tabelle* einen Datensatz ein. Die Attributwerte werden parallel zur Aufzählung eingefügt.

- Feldnamen können auch weggelassen werden, wenn in alle Felder etwas eingefügt werden soll. Dabei müssen jedoch dann die Werte auch in der Reihenfolge eingeben werden wie die Felder vorher definiert wurden.
- Bei den Werten müssen Zeichenketten und Daten in Hochkommata (Anführungszeichen) stehen, nur für Zahlen gilt das nicht.

Beispiel:

Es soll folgende Werte in eine Tabelle eingefügt werden (die Felder der Tabelle wurden schon vorher definiert):

Mitarbeiter

Name	GebDat	Telefon
junetz.de	5.3.1998	0815

Folgender SQL-Befehl erledigt diese Aufgabe:

`INSERT INTO Mitarbeiter (Name, GebDat, Telefon) VALUES ('junetz.de', '1998-3-5', 0815);`

Ändern

Syntax

`UPDATE tabelle SET attribut = wert [WHERE bedingung];`

Setzt in der Tabelle für alle Tupel[, bei denen die *bedingung* erfüllt ist,] für *attribut* den *wert* ein.
Wenn **keine** `where_definition` angegeben wird, werden **alle** Tupel geändert

Beispiel:

`UPDATE Student SET Raumnummer=521 WHERE sname='Leonard'`

Leonard wohnt jetzt in Zimmer 521

Löschen

Syntax

`DELETE FROM tabelle WHERE bedingung;`

Löscht alle Tupel aus *tabelle*, bei denen die *bedingung* erfüllt ist.

Die Bedingung ist nicht nötig, dann wird der gesamte Inhalt der Tabelle gelöscht.

Verwaltung

Hinweis: Die folgenden Befehle hängen teilweise vom eingesetzten DBMS ab und müssen nicht in allen genauso funktionieren!

Datenbanken

`SHOW DATABASES;`

Gibt alle vorhandenen Datenbanken aus.

`CREATE DATABASE db;`

Erstellt die Datenbank *db*.

`DROP DATABASE [IF EXISTS] db;`

Löscht die Datenbank *db*. [Überprüft vorher, ob die Datenbank vorhanden ist]

Tabellen

`SHOW TABLES FROM db;`

Gibt alle Tabellen in der Datenbank *db* zurück.

`CREATE TABLE tabelle (attribut1Name attribut1Typ, attribut2Name attribut2Typ);`

Erstellt die Tabelle "*tabelle*" mit den Attributen "*attribut1Name*" und "*attribut2Name*" von den jeweiligen Datentypen.

`DROP TABLE [IF EXISTS] tabelle;`

Löscht die Tabelle. [Falls sie existiert]

TRUNCATE TABLE "Tabellen_Name"

Löscht alle Zeilen in der angegebenen Tabelle.

ALTER TABLE tabelle ADD COLUMN (attributName attributTyp);

Fügt der Tabelle das Attribut vom entsprechenden Typ hinzu.

Beispiel: **ALTER TABLE Kunde ADD COLUMN (alter INT, registrierung DATE, bemerkung TEXT);**

ALTER TABLE tabelle MODIFY (attributName attributTyp); (Von Oracle eingeführt. Wahrscheinlich kein Standard.)

Ändert den Datentyp eines vorhandenen Attributs *attributName* in *attributTyp*.

Beispiel: **ALTER TABLE Kunde MODIFY bemerkung MEDIUMTEXT;**

ALTER TABLE tabelle DROP COLUMN attribut;

Löscht ein Attribut aus der Tabelle.

Beispiel: **ALTER TABLE Kunde DROP COLUMN alter, registrierung;**

ALTER TABLE tabelle CHANGE (attribut1 attribut2 attribut2Typ);

Ersetzt ein Attribut durch ein zweites Attribut.

Beispiel: **ALTER TABLE Kunde CHANGE alter geburtsdatum DATE NOT NULL;**

ALTER TABLE tabelle ADD PRIMARY KEY attribut;

Setzt ein Attribut als Primärschlüssel. Das Attribut muss **NOT NULL** sein.

Beispiel: **ALTER TABLE Kunde ADD PRIMARY KEY alter;**

Sonstiges

Ähnlich wie in Java oder höheren Programmiersprachen gibt es in DBMS auch Datentypen. Hier sind einige genannt.

Attributtypen

int

Integer

char(n)

Zeichen mit fester Länge *n*. (Quasi ein String mit einer festgelegten Länge).

varchar [(n)]

Zeichen mit nicht festgelegter Länge. [Mit maximaler Länge *n*]

bit

Ein Bit, 1 oder 0.

numeric[(p[, s])]

Ein Datentyp mit Kommastellen. [p gibt die maximale insgesamte Anzahl der Stellen an, s die maximale Anzahl nach dem Komma]

Attributparameter

NOT NULL

Der Wert für dieses Attribut muss für jedes Tupel gesetzt sein.

UNIQUE

Jeder Wert für dieses Attribut darf nur bei einem Tupel vorhanden sein.

AUTO INCREMENT

Bedingungen und Vergleichsoperatoren

>	größer als
<	kleiner als
=	gleich
<>	ungleich
>=	größer oder gleich
<=	kleiner oder gleich

a LIKE b

LIKE ist ein Zeichenoperator, das heißt, er bearbeitet *char* und *varchar*.

Hierbei gibt es zwei Sonderzeichen: "%" und "_". Das Prozentzeichen steht für beliebig viele beliebige Zeichen, der Unterstrich für genau eines.

Bsp.:

"%def" *LIKE* "abcdef" ist wahr, das Prozentzeichen steht für abc.

"%def%" *LIKE* "abcdefghi" ist ebenfalls wahr. Prinzipiell kann man sagen, dass ein "%abc%" für alle Zeichenketten gilt, in denen "abc" vorkommt.

Sonderfunktionen

Die folgenden Funktionen kann man wie normale SELECT-Funktionen verwenden, das heißt, hier sind z.B. auch Bedingungen möglich.

SELECT COUNT() FROM tabelle;*

Liefert die Anzahl der Tupel zurück, die die Abfrage ergibt.

SELECT SUM(attributName) FROM tabelle;

Gibt die Summe aller Werte für *attributName* aller Tupel zurück, die die Abfrage ergibt.

SELECT AVG(attributName) FROM tabelle;

Gibt den Durchschnittswert von *attributName* zurück.

SELECT MAX(attributName) [/MIN(attributName)] FROM tabelle;

Gibt den maximalen [minimalen] Attributwert zurück.

Joins

Equi-Join

Der Equi-Join ist gleich dem bekannten Natural-Join, nur dass beim Natural-Join gleiche Spalten weg projiziert werden.

Self-Join

Der Self-Join ist ein Natural-Join, nur dass in dem Join eine Tabelle mit sich selbst verbunden wird:

```
SELECT
    KK1.KndNr,
    KK1.Firma,
    KK2.KndNr,
    KK2.Firma
FROM Kreditkarten AS KK1, Kreditkarten AS KK2
WHERE KK1.KndNr = KK2.KndNr
```

INNER JOIN

Nach einem Schlüssel werden die Daten/Tabellen verbunden (bekannter Join).

Tabelle Persons:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Tabelle Orders:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Die Anfrage...

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
INNER JOIN Orders
ON Persons.P_Id=Orders.P_Id
ORDER BY Persons.LastName
```

...ergibt:

LastName	FirstName	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678

OUTER JOIN

Der Unterschied zwischen dem Inner und Outer Join ist, dass der Inner Join sich nur auf Elemente bezieht, die in beiden Tabellen vorhanden sind.

Hingegen der Outer Join: Alle Elemente der einen Tabelle sind in der Ergebnismenge vorhanden, dabei sind von den Feldern aus der outer join Tabelle bei nicht vorhandenen Elementen der Wert Null (gesprochen "null", also nichts), gesetzt .

Funktionen

Bei select_expression und where_expression können neben Konstanten und Spaltenwerten auch

Funktionen verwendet werden. Es gibt zwei Arten von Funktionen, zum einen die sog. „singlerow``-Funktionen und zum anderen die Gruppenfunktionen. Singlerow-Funktionen werden auf jede Zeile angewendet, während die Gruppenfunktionen immer auf eine Gruppe von Zeilen angewendet werden.

Es wäre zum Beispiel Unsinn, zu versuchen, den Betrag von allen Stellenanzahlen auf einmal zu berechnen; bei der Summe paßt das schon eher.

Mathematische Funktionen

Es können nicht nur Spaltennamen angegeben werden, sondern auch mathematische Rechenoperationen mit Spalten und/oder Konstanten.

Hinweis: Nicht alle der hier genannten Funktionen werden von allen DBMS unterstützt.

+ - * /	addieren/subtrahieren/multiplizieren/dividieren
%	modulo (ganzzahliger Rest)
ABS()	Betrag von
COS()	Cosinus in rad
DEGREES()	Umrechnung von rad in deg (Grad)
MOD()	Modulo (ganzzahliger Rest)
PI()	die Zahl Pi
POW(X,Y)	rechnet X hoch Y aus
RAND()	liefert eine Zufallszahl zwischen 0 und 1
ROUND()	rundet Wert
ROUND(x,n)	rundet Wert von x auf n Stellen
SQRT()	Wurzel (2. Grades)
TRUNCATE(x,n)	schneidet nach n Kommastellen von x ab

Logische Operatoren

NOT	logisches NOT. Wird genau dann wahr (gibt 1 zurück), wenn
!	das Argument 0 ist (sonst Rückgabe 0). NOT NULL gibt NULL zurück.
AND	logisches UND. Wird genau dann wahr (gibt 1 zurück), wenn
&&	alle Argumente weder 0 noch NULL sind (sonst Rückgabe 0).
OR	logisches ODER. Wird genau dann wahr (gibt 1 zurück), wenn
	eines der Argumente weder 0 noch NULL ist (sonst Rückgabe 0).

Sonstige Funktionen

In Tabelle 6.6 und 6.7 sind sonstige, teilweise praktische Funktionen aufgeführt. Die Tabellen sind nicht vollständig!

I	bitweises ODER
&	bitweises UND

CONCAT(str1, str2, ...)	Gibt den String zurück, der durch Zusammenführen der Argumente entstanden ist. Sobald ein Argument NULL ist, wird NULL zurückgegeben.
-------------------------	---

LEFT(str,n)	schneidet n Buchstaben von `str` ab und gibt diese zurück
LTRIM(str)	löscht alle Leerzeichen am Anfang von `str`
PASSWORD(str)	verschlüsselt den Klartext `str`
REVERSE(str)	dreht `str` um, d.h. letzter Buchstabe ist dann am Anfang
LCASE(str)	Wandelt `str` in Kleinbuchstaben
LOWER(str)	und gibt das Ergebnis zurück
UCASE(str)	Wandelt `str` in Großbuchstaben
UPPER(str)	und gibt das Ergebnis zurück

DAYOFWEEK(date)	Gibt den Wochentag-Index des Datums zurück (1 = Sonntag, 2 = Montag, ..., 7 = Samstag)
DAYOFMONTH(date)	Gibt den Tag des Monats zurück
DAYOFYEAR(date)	Gibt den Tag im Jahr zurück
WEEK(date)	Gibt die Woche des Datums zurück.
WEEK(date,first)	Wenn `first` nicht angegeben wird bzw. 0 ist, fängt die Woche mit Sonntag an. Ist `first` z.B. 1, fängt die Woche mit Montag an.
MONTH(date)	Gibt den Monat zurück
YEAR(date)	Gibt das Jahr zurück
DATE_FORMAT(date,format)	Formatiert das Datum date

	entsprechend dem übergebenen format String.
UNIX_TIMESTAMP(date)	Gibt den Unix-Timestamp (Sekunden seit dem 1.1.1970) des Datums date zurück.

Datums-Funktionen

Mit Hilfe von DATE_FORMAT kann man Datumswerte aus Tabellen so formatieren, wie man sie gerne hätte. Die Funktion erwarten zwei Parameter. Zum einen das Datumsfeld, zum anderen den Formatierungs-String. Die Formatierungszeichen werden durch die entsprechenden Werte ersetzt. Alle anderen Zeichen werden so wie sie sind ausgegeben.

%W	Wochentag
%w	Tag in der Woche (0 = Sonntag, ..., 6=Samstag)
%d	Tag des Monats (00 - 31)
%e	Tag des Monats (0 - 31)
%j	Tag im Jahr (001 - 366)
%U	Woche, mit Sonntag als 1. Tag der Woche (00 - 52)
%u	Woche, mit Montag als 1. Tag der Woche (00 - 52)
%M	Monatsname
%m	Monat, numerisch (01 - 12)
%c	Monat, numerisch (1 - 12)
%Y	Jahr (4stellig)
%y	Jahr (2stellig)
%T	Uhrzeit (24 Std.) (hh:mm:ss)

%S	Sekunden (00 - 59)
%S	Sekunden (00 - 59)
%i	Minuten (00 - 59)
%H	Stunde (00 - 23)
%k	Stunde (0 - 23)
%h	Stunde (00 - 12)
%l	Stunde (00 - 12)
%l	Stunde (0 - 12)
%%	%

Gruppenfunktionen

Es können aber auch die sogenannten Gruppenfunktionen verwendet werden. Gruppenfunktionen heißen so, weil sie immer auf eine Gruppe von Tupeln (Datensätzen) angewendet werden.

In Verbindung mit Gruppenfunktionen darf streng genommen kein Spaltenname mehr mit in der select_expression stehen.

COUNT(expr)	zählt die Zeilen, deren Werte ungleich NULL sind
AVG(expr)	durchschnittlicher Wert
MIN(expr)	kleinster Wert
MAX(expr)	größter Wert
SUM(expr)	Summe

Links

- [Relationale Algebra @ Wikipedia](#)
- <http://www.sql-und-xml.de/sql-tutorial/>
- [Kleine SQL-Referenz](#)

- [Übersicht über die Datentypen in SQL](#)
- [DSP: Datenbank, MySQL und PHP \(<http://reeq.junetz.de/DSP/>\)](#)
- [SQL Tutorial](#)

Anhang

Mögliche Lösung zur Aufgabe **Wetenheimer Zoo**.

