Dr. Carl Stolze

VOM ERM ZUM SQL

Eine Kurzeinführung in Entity-Relationship-Model und der Umsetzung in SQL-Statements

3. Auflage (März 2021)

Aktuelle Version jeweils verfügbar unter www.tabellendoktor.de/1754-vom-erm-zum-sql/

Der TabellenDoktor
Arbeiten mit Daten – Excel, SQL, Python und alles andere

```
4   DROP TABLE IF EXISTS Studierende;
5
6   -- Tabelle Studierende erstellen
7   CREATE TABLE Studierende (
8       Matrikelnummer INTEGER, -- gar
9       StudName VARCHAR(30) NOT NULL,
        Zeichenfolge mit 30 Stellen, N
        angegeben werden
10      PRIMARY KEY (Matrikelnummer) -
        Identifikation jeder Zeile
11  )
12
13  -- Tabelle Dozent erstellen
14  CREATE TABLE Dozent (
15      Dozentennummer INTEGER,
16      DozName VARCHAR(30) NOT NULL,
17      PRIMARY KEY (Dozentennummer)
```

VORWORT & ANMERKUNGEN

Willkommen bei „Vom ERM zum SQL“!

Mit dieser praktischen Übung möchte ich erklären wie aus einem abstrakten Entity-Relationship-Model eine „richtige“ Datenbank wird. Dazu werden einfache SQL-Statements vorgeschlagen.

Bevor es losgeht noch ein paar Anmerkungen und Hinweise zu diesem Werk:

I. Übung macht den Meister. Einmal lesen ist nett. Einmal Ausprobieren schon gut. Mehrmals durcharbeiten und selber Alternativen entwickeln noch besser.

II. Nachdenken vor dem Enter-Drücken: Haftung für Schäden an Datenbanken und/oder Datenständen übernehme ich keine. Daher erst denken, noch einmal drüber schauen und dann ein Kommando auslösen. Oder noch besser: Zum Lernen und Ausprobieren ein getrenntes System verwenden.

III. Geschrieben habe ich alles nach bestem Wissen und Gewissen. Nur macht jeder Mensch auch Fehler – und damit ich auch. Daher die Bitte etwaige Ungenauigkeiten und Verbesserungsvorschläge an mich zu senden, damit ich das korrigieren kann.

IV. Die Beispiele sind angelehnt an die Darstellungen in *Database System Concepts* (7. Auflage, siehe Quellen)

So genug der Worte: Viel Vergnügen beim Modellieren, Ausführen und Arbeiten von Datenstrukturen!

Dr. Carl „Tabellendoktor“ Stolze
team@tabellendoktor.de

Umgebung einrichten

LOS GEHT'S

- Um die Übungen nachzuvollziehen braucht es eine „echte“ Datenbank (DB) bzw. ein Datenbankmanagementsystem (DBMS)
- Fürs Üben geht auch ein Online-„Spielplatz“ (Sandbox)

NICHTS ÜBT BESSER ALS DIE PRAXIS…

…daher gibt es unter https://www.tabellendoktor.de/1940-sql-lernen-ausfuehren/ eine Übersicht über verschiedene Möglichkeiten schlank und schnell eine Umgebung zum Üben zu erhalten.

https://www.tabellendoktor.de/1940-sql-lernen-ausfuehren/

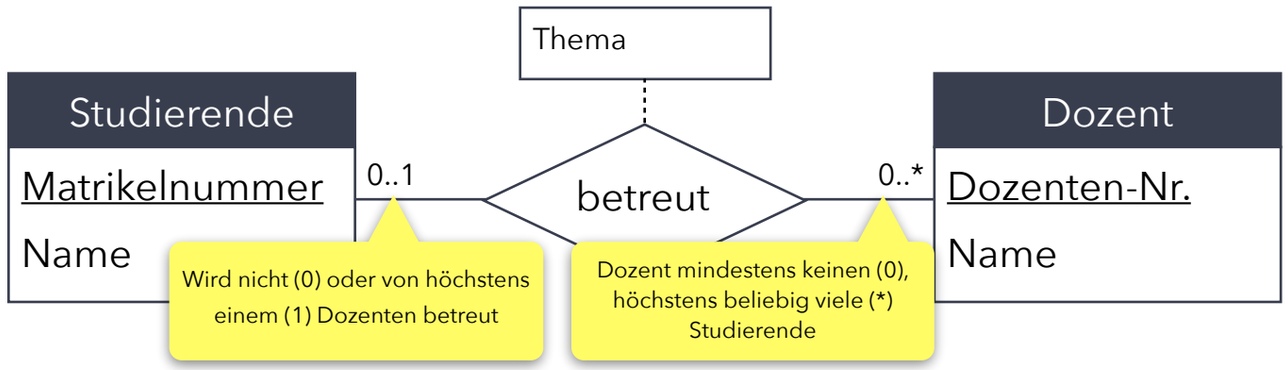
Hintergrund

ERM ⇢ SQL

- Ein Entity-Relationship-Model (ERM) zeigt eine fachliche bzw. logische Sicht auf Entitäten
- Aus Entitäten und Relationen werden physisch Tabellen in einer Datenbank
- „Übersetzung“ von ERM in Tabellen = bewußte Entscheidung wie etwas logisch Beschriebenes in einer Datenbank wirklich umgesetzt wird
- Umsetzung erfolgt dann mit der Structured Query Language (SQL)

FACHLICHE SICHT: ENTITÄTEN IN BEZIEHUNG

In der fachlichen bzw. logischen Sicht gibt es bestimmte Dinge (Entitäten) die in einer bestimmten Beziehung (Relation) zueinander stehen. Beispielsweise werden Studierende von Dozenten bei Abschlussarbeiten betreut. Studierende werden über ihre Matrikelnummer identifiziert, Dozenten über eine Dozentennummer.



UMSETZUNG IN DATENBANK: TABELLEN & SQL

In *relationalen* Datenbanksystemen werden aus Entitäten und Relationen Tabellen erzeugt. Über SQL-Statements wie „CREATE TABLE“ können die Tabellen angelegt, „SELECT […] FROM“ erlaubt die Abfrage von Daten , usw. – Details in den Folgekapiteln.

Studierende

Matrikel-nummer	StudName
19001	Michael Meier
19002	Andreas Altmann
19003	Beate Blume
19004	Claudia Cool
…	…

Betreut

Matrikel-nummer	Dozenten-Nr.	Thema	Note
19001	**2000**	SQL 1	1,0
19002	**2000**	ERM IV	2,3
19004	**2002**	SQL 5	1,0
…	**…**		

Dozent

Dozenten-Nr.	DozName
2000	Frieda Freu
2001	Günther Grau
2002	Holger Heu
2003	Ingrid Insel
…	…

SQL-Grundlagen

SQL, YEAH!

- **SQL erlaubt die Definition, Manipulation und Abfrage von Daten**
- **SQL ist bewährte Sprache für eine Vielzahl von Datenbanksystemen**
- **SQL unterscheidet im Standard nicht zwischen Groß- und Kleinschreibung**
- **SQL „hört" auf bestimmte Befehlsworte**

SQL IST MEHR ALS DATEN ABFRAGEN

Die Structured Query Language (SQL) ist – trotz ihres Namens – mehr als eine „strukturierte Abfragesprache" (mit `SELECT`):

- Datenstrukturen definieren (Data Definition Language, DDL) – Befehle wie `CREATE`, `ALTER`
- Einfügen, Ändern und Löschen von Daten (Data Manipulation Language, DML) – Befehle dazu `INSERT`, `UPDATE` und `DELETE`

AUFBAU VON SQL-BEFEHLEN

SQL-Befehle folgen unabhängig vom verwendeten Datenbanksystem einer immer gleichen Syntax. Dabei wird nicht zwischen Groß- und Kleinschreibung unterschieden. Wichtig sind Befehlsworte wie `SELECT` oder `JOIN` sowie die Einhaltung der Reihenfolge von Befehlen, Parametern und Optionen.

Schauen wir uns einen einfachen SQL-Befehl an, um alle Einträge aus einer Tabelle abzufragen:

```
SELECT * FROM Studierende
```

`SELECT` = selektiere/hole Daten
`*` = alle Spalten ohne Einschränkung (das gilt dann auch wenn mehrere Tabellen verknüpft abgefragt werden → `JOIN`)
`FROM Studierende` = aus der Tabelle Studierende

SQL-Grundlagen

SQL ≠ SQL

- **Datenbank(managementsysteme) verstehen SQL manchmal etwas unterschiedlich**
- **Verschiedene Datenbanken sprechen unterschiedliche Dialekte – so wie im echten Leben auch Menschen unterschiedlich bspw. Deutsch sprechen**
- **Grundbefehle und Aufbau sind bei allen gleich („Der Teufel steckt im Detail“)**

SQL ALS SPRACHE MIT DIALEKTEN

Trotz der Standardisierung von SQL haben sich ein paar Besonderheiten je nach verwendeter Datenbank bzw. verwendetem Datenbankmanagementsystem herausgebildet:

- Bei der Massenausführung von Befehlen in einem Rutsch reichen Zeilenumbrüche zum Trennen der Befehle oder es muss ein Semikolon (MySQL) verwendet werden
- Microsoft SQL Server „verpackt“ Spaltennamen gerne in eckige Klammern ([]) – MySQL/MariaDB hingegen nicht
- Arten von Anführungszeichen werden je nach System unterschiedlich verwendet (einfache oder doppelte, teils für Bezeichner, teils nur für Texte)

UMGANG DAMIT

Die Beispiele in diesem Buch sollten recht allgemein funktionieren, sofern sie Schritt für Schritt einzeln ausgeführt werden.

Bei der Massenausführung mehrerer Befehle in einem Rutsch, bitte schauen was das jeweilige System sich an Trennzeichen zwischen Befehlen wünscht.

Erzeugen von Struktur

CREATE

- **Um Daten in einer Datenbank abzulegen müssen Tabellen mit definierten Spalten angelegt werden – Werte innerhalb einer Spalte haben immer den gleichen festgelegten Datentyp (~Domäne)**
- **Mit `CREATE` können „echte“ Tabellen oder auch Sichten (Views) auf Daten erzeugt werden**
- **Dank `CREATE` ist SQL auch eine Data Definition Language**

SPALTEN GEBEN STRUKTUR

Eine Tabelle besteht immer aus mindestens einer Spalte für jeden Datensatz (=Zeil). Die maximale Anzahl an Spalten hängt vom konkret verwendeten System ab. In der Praxis sind häufig rund 1000 bzw. 1024 Spalten das Limit.

Neben dem Datentypen kann auch festgelegt werden, ob eine oder mehrere Spalte einen Datensatz als Primärschlüssel identifizieren. Auch können wir der Datenbank sagen, dass die Werte in einer oder mehrere Spalten immer denen aus einer anderen Tabelle entsprechen müssen (bspw. für Relation *Betreut*).

GRUNDAUFBAU CREATE BEFEHL

```
CREATE TABLE Studierende (
    Matrikelnummer INTEGER,
    StudName VARCHAR(30) NOT NULL,
    PRIMARY KEY (Matrikelnummer)
)
```

`CREATE TABLE` = erstelle eine neue Tabelle, in diesem Bespiel mit dem Namen *Studierende*

`(...)` = Definition der Spalten – jeweils durch ein Komma getrennt

INTEGER, VARCHAR(30) = jeweiliger Datentyp/Domäne für die davor neu definierte Spalte

NOT NULL = darf nicht leer bleiben

PRIMARY KEY = vor definierte Spalte(n) in der Klammer identifizieren jeweils eindeutig eine Zeile in dieser Tabelle

BEISPIEL: ERSTELLUNG DER TABELLEN STUDIERENDE, DOZENT UND BETREUT

Rechts sind die Befehle um unsere Beispielrelationen in einer SQL-Datenbank als Tabellen abzubilden. Der grüne Text sind Kommentare, welche von der Datenbank ignoriert werden und nur dem Menschen helfen sollen. 🤓

Frischer Start

`DROP TABLE IF EXISTS` = löscht Tabellen mit dem jeweiligen Namen. ❗ Alle Daten in den Tabellen werde auch gelöscht dabei.

Tabelle Studierende

Tabelle Studierende wird mit zwei Spalten (Matrikelnummer und StudName) erzeugt. Eine davon (Matrikelnummer) wird als Primärschlüssel (`PRIMARY KEY`) festgelegt.

Tabelle Dozent

Für Dozent das gleiche noch einmal – nur mit anderen Spaltennamen

Tabelle Betreut

Hier wird die Verknüpfung (Relation) als Tabelle „Betreut“ hergestellt.
Als Fremdschlüssel (`FOREIGN KEY`) werden die Primärschlüssel der beiden anderen Tabellen referenziert. Dadurch können hier nur Einträge für eine Kombination aus existierenden Einträgen aus Studierende und Dozent erstellt werden.

```sql
 -- Tabellen aus Datenbank löschen (sofern bereits vorhanden)
DROP TABLE IF EXISTS betreut;
DROP TABLE IF EXISTS Studierende;
DROP TABLE IF EXISTS Dozent;

-- Tabelle Studierende erstellen
CREATE TABLE Studierende (
    Matrikelnummer INTEGER, -- ganzzahliger Wert
    StudName VARCHAR(30) NOT NULL, -- belibige Zeichenfolge mit 30 Stellen, NOT NULL = muss angegeben werden
    PRIMARY KEY (Matrikelnummer) -- eindeutige Identifikation jeder Zeile
);

-- Tabelle Dozent erstellen
CREATE TABLE Dozent (
    Dozentennummer INTEGER,
    DozName VARCHAR(30) NOT NULL,
    PRIMARY KEY (Dozentennummer)
);

-- Tabelle Betreut erstellen
CREATE TABLE Betreut (
    Matrikelnummer INTEGER UNIQUE, -- UNIQUE damit jeder Studierende nur einmal betreut werden kann
    Dozentennummer INTEGER,
    Thema VARCHAR(160), -- Thema kann auch leer sein bei Anlage Datensatz
    Note NUMERIC(2,1), -- Note ist numerisch, hat zwei Stellen, davon eine Nachkommastelle
    PRIMARY KEY (Matrikelnummer, Dozentennummer),
    FOREIGN KEY (Matrikelnummer) REFERENCES Studierende(Matrikelnummer),
    FOREIGN KEY (Dozentennummer) REFERENCES Dozent(Dozentennummer)
)
```

Semikolon am Ende braucht MySQL bei der Verwendung von phpMyAdmin

Mit zwei Bindestrichen beginnt ein Kommentar – der wird nicht von der Datenbank verarbeitet

Daten reinbekommen

INSERT

- Daten können mit `INSERT INTO` jeweils in eine bestimmte Tabelle geschrieben werden
- Die Datenbank prüft, ob die einzufügenden Daten technisch passend sind – d.h. einerseits dem definierten Wertebereich der jeweiligen Spalte entsprechen und andererseits auch allen weiteren Bedingungen wie Primär-/ Fremdschlüssel, nicht NULL, etc.

AUFBAU AN BEISPIELEN FÜR INSERT ERKLÄRT

```
-- Einträge in Studierende einfügen
INSERT INTO Studierende VALUES (19001, 'Michael Meier'); -- ohne Angabe Spalten: Werte sind in der Reihenfolge im CREATE TABLE anzugeben
INSERT INTO Studierende(Matrikelnummer, StudName) VALUES (19002, 'Andreas Altmann'); -- mit expliziter Angabe entsprechende Reihenfolge
INSERT INTO Studierende(StudName, Matrikelnummer) VALUES ('Beate Blume', 19003); -- getauschte Reihenfolge
INSERT INTO Studierende(Matrikelnummer,StudName) VALUES (19004, 'Claudia Cool');
```

Explizit angegebene Spaltennamen für die Werte

VALUES = es werden die einzufügenden Werte explizit angegeben, zwischen den Klammern folgen dann die Werte durch Kommata getrennt
Alternativ könnte hier auch ein SELECT-Befehl statt VALUES und Werten die notwendigen Daten liefern

```
-- Einträge in Dozent einfügen
INSERT INTO Dozent(Dozentennummer, DozName) VALUES (2000, 'Frieda Freu');
INSERT INTO Dozent(Dozentennummer, DozName) VALUES (2001, 'Günther Grau');
INSERT INTO Dozent(Dozentennummer, DozName) VALUES (2002, 'Holger Heu');
INSERT INTO Dozent(Dozentennummer, DozName) VALUES (2003, 'Ingrid Insel');
```

Name der Tabelle in die eingefügt werden soll

```
-- Einträge in Betreut einfügen
INSERT INTO Betreut(Matrikelnummer, Dozentennummer) VALUES(19004, 2002); -- Betreuung noch ohne Thema und Note
INSERT INTO Betreut(Matrikelnummer, Dozentennummer, Thema) VALUES(19002, 2000, 'ERM IV'); -- Betreuung mit Thema aber ohne Note
INSERT INTO Betreut(Matrikelnummer, Dozentennummer, Thema, Note) VALUES(19001, 2000, 'SQL 1', 1.0); -- einmal alles
```

Daten abfragen

SELECT

- Mit `SELECT` können Daten aus der Datenbank wieder abgefragt werden
- Dabei können Daten aus einer Tabelle kommen. Oder auch mehrere Tabellen „verknüpft“ (`JOIN`) werden.
- Daneben kann auch die Datenbank bereits Daten zusammenfassen/aggregieren (`GROUP BY`)

ABFRAGEN LEICHT GEMACHT – SELECT

Eine der großen Stärken von SQL ist die strukturierte Abfrage:

```
SELECT * FROM Studierende; -- alle Spalten aus Studierende
SELECT StudName FROM Studierende; -- nur Namen in Spalte StudName
```

Abfragen können auch eingeschränkt werden mit Hilfe von `WHERE`

```
SELECT * FROM Studierende WHERE Matrikelnummer = 19001;
SELECT * FROM Studierende WHERE Matrikelnummer > 19003;
SELECT * FROM Studierende WHERE Matrikelnummer BETWEEN 19002 AND
19003;
SELECT * FROM Studierende WHERE StudName LIKE ,%e%'; -- irgendwo im
Namen ein e
```

Für das Verknüpfen von Tabellen wird der `JOIN`-Befehl verwendet:

```
-- Erzeugung einer Tabelle in der alle Betreuungsverhältnisse
enthalten sind – zusammen mit den Namen der Studierenden und
Dozenten
SELECT *
    FROM Betreut
    INNER JOIN Studierende
        ON Betreut.Matrikelnummer = Studierende.Matrikelnummer
    INNER JOIN Dozent
        ON Betreut.Dozentennummer = Dozent.Dozentennummer
```

Hier werden die Tabellen zusammengefügt – wenn die Matrikelnummer gleich ist, kommt es in eine Zeile

Und hier für die Dozentennummer

Auf den folgenden Seiten sind noch Beispiele für mehr Abfragen und die jeweiligen Ergebnisse sowie erweiterte Befehle wie `GROUP BY`

```sql
-- Erzeugung einer Tabelle in der alle
Betreuungsverhältnisse enthalten sind – zusammen mit
den Namen der Studierenden und Dozenten
SELECT *
    FROM Betreut
    INNER JOIN Studierende
        ON Betreut.Matrikelnummer =
Studierende.Matrikelnummer
    INNER JOIN Dozent
        ON Betreut.Dozentennummer =
Dozent.Dozentennummer

-- gefiltert auf nicht leere Themen
SELECT *
    FROM Betreut
    INNER JOIN Studierende
        ON Betreut.Matrikelnummer =
Studierende.Matrikelnummer
    INNER JOIN Dozent
        ON Betreut.Dozentennummer =
Dozent.Dozentennummer
    WHERE Betreut.Thema IS NOT NULL
```

Matrikelnummer	Dozentennummer	Thema	Note	Matrikelnummer	StudName	Dozentennummer	DozName
19001	2000	SQL 1	1.0	19001	Michael Meier	2000	Frieda Freu
19002	2000	ERM IV	*NULL*	19002	Andreas Altmann	2000	Frieda Freu
19004	2002	*NULL*	*NULL*	19004	Claudia Cool	2002	Holger Heu

NULL in dern Ergebnissen bedeutet da ist nichts – also ein „leerer Eintrag“ in der Datenbank

Matrikelnummer	Dozentennummer	Thema	Note	Matrikelnummer	StudName	Dozentennummer	DozName
19001	2000	SQL 1	1.0	19001	Michael Meier	2000	Frieda Freu
19002	2000	ERM IV	*NULL*	19002	Andreas Altmann	2000	Frieda Freu

```sql
-- Wie viele Studierende betreut ein Dozent?
SELECT Dozent.*, COUNT(Betreut.Matrikelnummer) AS
betreuteStudierende
    FROM Betreut
    INNER JOIN Dozent
        ON Betreut.Dozentennummer = Dozent.Dozentennummer
    GROUP BY Dozent.Dozentennummer
```

COUNT zählt die Anzahl von Datensätzen

Zusammenfassung nach dieser Spalte (Dozentennummer)

Dozentennummer	DozName	betreuteStudierende
2000	Frieda Freu	2
2002	Holger Heu	1

```sql
-- Wie viele Studierende betreut ein Dozent? (inklusive
Dozenten ohne Betreuungen)
SELECT Dozent.*, COUNT(Betreut.Matrikelnummer) AS
betreuteStudierende
    FROM Betreut
    RIGHT JOIN Dozent
        ON Betreut.Dozentennummer =
Dozent.Dozentennummer
    GROUP BY Dozent.Dozentennummer
```

RIGHT JOIN (gibt auch LEFT JOIN) = die Tabelle rechts davon wird komplett betrachtet – auch für die Fälle in deine kein Ergebnis für das „ON“ herauskommt)

Dozentennummer	DozName	betreuteStudierende
2000	Frieda Freu	2
2001	Günther Grau	0
2002	Holger Heu	1
2003	Ingrid Insel	0

```sql
-- Welche Studierende werden durch einen Dozenten mit mindestens 2 Betreuungen
betreut?
SELECT Studierende.Matrikelnummer, Studierende.StudName
    FROM Betreut
    INNER JOIN Studierende
    ON Betreut.Matrikelnummer = Studierende.Matrikelnummer
    WHERE Betreut.Dozentennummer IN (
        SELECT Betreut.Dozentennummer
        FROM Betreut
        GROUP BY Betreut.Dozentennummer
        HAVING COUNT(Betreut.Matrikelnummer) >= 2
    )
```

Nur ausgewählte Spalten ausgegeben

Verschachtelte Abfrage, d.h. das Ergebnis einer Abfrage ist wieder Parameter in einer anderen

HAVING = Filtern auf einem Aggregationsergebnis wie COUNT

Matrikelnummer	StudName
19001	Michael Meier
19002	Andreas Altmann

Die innere Abfrage

```sql
SELECT Betreut.Dozentennummer
    FROM Betreut
    GROUP BY Betreut.Dozentennummer
    HAVING COUNT(Betreut.Matrikelnummer) >= 2
```

So wird exakt eine Spalte ausgegeben – beim Entwickeln der Abfrage sollte man sich auch das Ergebnis vom COUNT anzeigen lassen, damit man eine Kontrolle hat

Dozentennummer
2000

QUELLEN

Lehrbuch

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *Database System Concepts* (7th ed.). McGraw-Hill.

Weitere Ressourcen zu diesem Lehrbuch wurden von den Autoren auf www.db-book.com veröffentlicht (u.a. SQL-Beispiele)

Wissenschaftliche Abhandlungen

Fettke, P. (2009). Ansätze der Informationsmodellierung und ihre betriebswirtschaftliche Bedeutung: Eine Untersuchung der Modellierungspraxis in Deutschland. *Zfbf*, *61*, 550–580.

Krcmar, H. (2005). *Informationsmanagement* (4. Auflage). Berlin: Springer.

Titelbild und Abbildungen in diesem Werk

Abbildungen in diesem Werk wurden durch den Autor selbst erstellt.

Titelbild: Screenshot eines SQL-Statements geöffnet in Visual Studio Code

Der TabellenDoktor

Arbeiten mit Daten – Excel, SQL, Python und alles andere

www.tabellendoktor.de

Telefon: +49 89 215290760

E-Mail: team@tabellendoktor.de

TabellenDoktor

ist ein Projekt der drcs82 UG (haftungsbeschränkt), Eching