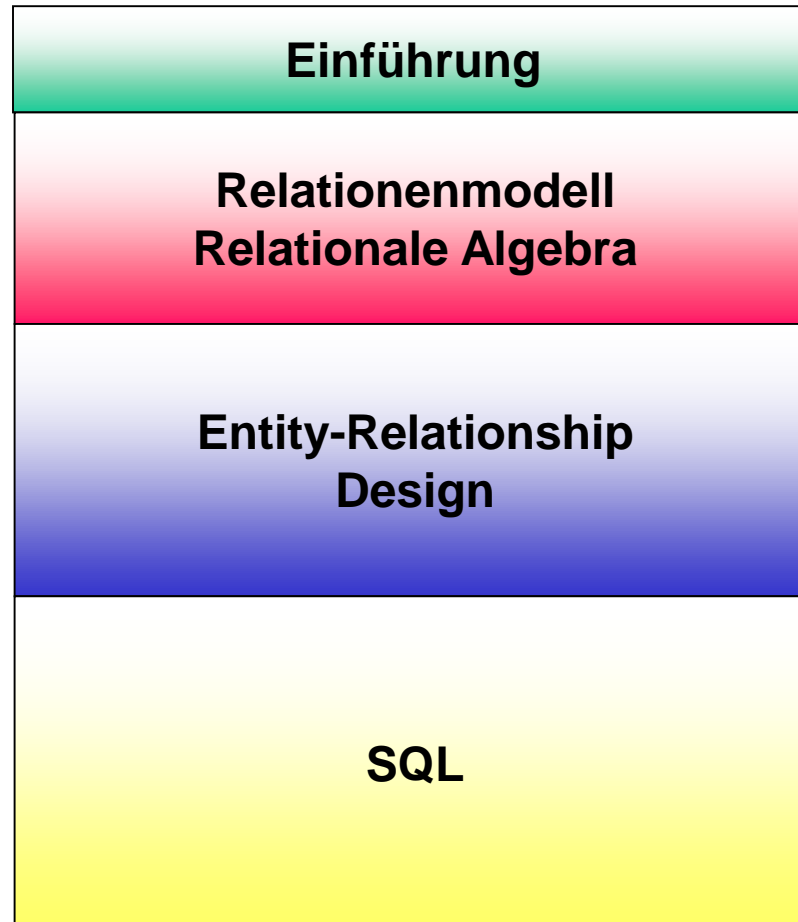


DAB1 – Datenbanken 1

Dr. Daniel Aebi (aebd@zhaw.ch)

Lektion 9: SQL – DDL

Wo stehen wir?

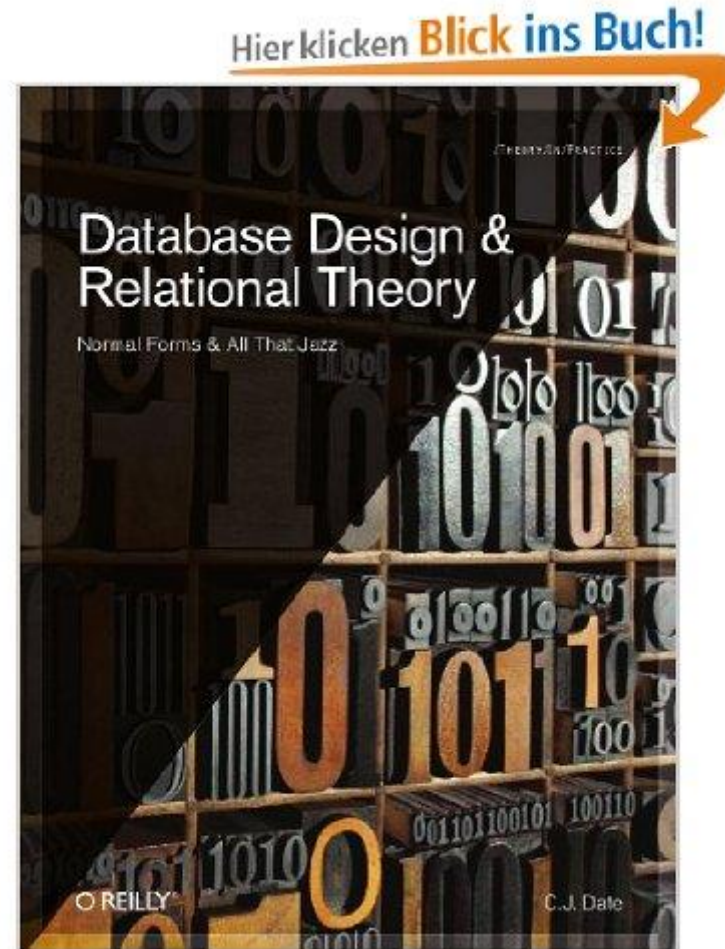


← "You are here"

Diskutiert im Unterricht. Machen Sie Ihre eigenen Notizen.

Rückblick

- Buchempfehlung zum Thema «Normalisierung»:
- C.J. Date: Database Design and Relational Theory: Normal Forms and All That Jazz
- O'Reilly Media; 1. Auflage 2012
ISBN 978-1449328016



Lernziele Lektion 9

- Entwicklung von SQL kennen
- Grundstruktur einzelner Anweisungen verstehen
- Elementare SQL DDL-Befehle anwenden können

SQL – Geschichte

- IBM Research: Structured English QUERy Language)
 - SEQUEL-XRM (1974)
 - SEQUEL-2 in System R (1976)
- RSI (heute: Oracle): Oracle V2 mit SQL (1980)
- IBM: SQL/DS (1981), DB2 für MVS Betriebssystem (1983)
- Ingres: QUEL, PostgreSQL
- SQL-86 und SQL-89 Standards: ca. 150 Seiten
- SQL-92 (SQL2) Standard: ca. 600 Seiten
 - Konsistenzbedingungen
 - CLI (Call Level Interface)
 - Schemas

SQL – Geschichte

- SQL:1999 Standard: ca. 3000 Seiten
 - Objekt-orientierte Erweiterungen
 - Rekursive Abfragen
 - „OLAP“-Erweiterungen
 - Trigger
- SQL:2003 Standard
 - XML-features, auto-generated-values, ..., siehe auch:
 - <http://www.acm.org/sigmod/record/issues/0403/E.JimAndrew-standard.pdf>
 - <http://www.sigmod.org/record/issues/0409/11.JimMelton.pdf>

SQL – Geschichte

- Aktuell: SQL:2008/2011, siehe auch
 - <http://www.iso.org/iso/search.htm?qt=9075&searchSubmit=Search&sort=rel&type=simple&published=true>
- Heute: Viele proprietäre Spracherweiterungen im Einsatz:
 - Transact SQL (Microsoft)
 - PL/SQL (Oracle)
 - ...

SQL \neq SQL

Wer hat's erfunden?

- Don Chamberlin, Ph. D. (Stanford University)
- Erst-Designer von SQL
- Arbeitete im IBM Almaden Research Center
- Mitglied des "System R"-Forschungsteams
- Erhielt 2005 ein Ehrendoktorat der Universität Zürich für seine Arbeiten zu SQL
- <http://researcher.ibm.com/researcher/view.php?person=us-dchamber>



SQL – Bemerkungen

- SQL ist eine – eher schlechte – Umsetzung der Ideen des relationalen Modelles von Codd.
- Was ist schief gelaufen?
 - Mangelnde Performanz (in den siebziger Jahren) führte zu **Kompromissen**, z.B. der Verzicht auf eine rein mengenmässige Verarbeitung, SQL lässt **Duplikate** zu.
 - Unklare, teils **widersprüchliche Bedeutung** einzelner Anweisungen, z.B. die Behandlung von NULL (siehe später).
 - Trotz Standardisierungsbemühungen ist eine **Vielzahl von Dialekten** entstanden. Die Sprache enthält sehr viel **Redundanz**.
 - ...
- Aber: Es gibt nichts Anderes, machen wir das Beste draus!

SQL – Überblick

- Data Definition Language (**DDL**)
 - Erzeugen, Ändern, Löschen von „Datenbank-Objekten“ (Schemas, Tabellen, Sichten, Integritätsbedingungen, ...)
 - Anweisungen: **CREATE, ALTER, DROP**, ...
- Data Manipulation Language (**DML**)
 - Daten ändern (Einfügen, Ändern, Löschen)
 - Anweisungen: **INSERT, UPDATE, DELETE**, ...

SQL – Überblick

- Data Query Language (**DQL**)
 - Daten lesen (Anfragen an Datenbank stellen)
 - Anweisung: **SELECT – FROM – WHERE**
 - Bemerkung: Die SELECT-Anweisung wird gelegentlich auch zu den DML-Anweisungen gezählt.
- Data Control Language (**DCL**)
 - Rechtevergabe, Datensicherung, ...
 - Anweisungen: **GRANT, REVOKE**, ...

SQL – Überblick

- SQL: Sprache für die Bearbeitung von **relationalen** Datenbanken.
- **Keine** vollumfängliche **Programmiersprache** wie z.B. Java; man kann nicht alle denkbaren Aufgaben lösen → meistens von anderen Programmiersprachen aus aufgerufen.
- Dafür einfach und sehr mächtig für die **Behandlung von Mengen**.
- Die meisten Programmiersprachen behandeln Mengen «one record at the time», nicht so SQL.

SQL – Terminologie

- SQL basiert zwar auf den Konzepten des Relationenmodelles, verwendet aber in vielen Bereichen eine **eigene Terminologie**, z.B.:

| | Relationenmodell | SQL | |
|----------|----------------------------|------------------|------------------|
| | | D | E |
| Struktur | Relation | Tabelle | table |
| | Attribut | Spalte, Kolonne | column |
| | Tupel | Zeile | row |
| Auswahl | Relationenalgebra Ausdruck | SELECT Anweisung | SELECT statement |
| | Projektion | SELECT Klausel | SELECT clause |
| | Selektion | WHERE Klausel | WHERE clause |

SQL – Bemerkungen

- Wir sprechen in SQL von **Tabellen**, nicht von Relationen.
- Tabellen implizieren eine bestimmte Reihenfolge der Spalten, im Gegensatz zu einer «idealen» Relation.
- Streng betrachtet ist SQL keine Mengensprache, sondern eine Sprache für den Umgang mit relationalen Bags (= Multimengen), resp. Tabellen (da Duplikate erlaubt sind).
- SQL ist «formatfrei», d.h. es gibt keine Regeln für die Anordnung der Anweisungen (z.B. Einrücken von Code). Es lohnt sich jedoch sehr, hier eigene Konventionen aufzustellen und konsequent einzuhalten (→ «Programmierrichtlinien»).

SQL – Bemerkungen

- In SQL spielt Gross-/Kleinschreibung nur innerhalb von Text-Konstanten eine Rolle!
- Die folgenden Schreibweisen von Namen werden als absolut identisch angesehen: **FAMILYNAME**, **familyname**, **FamilyName** und **familyName**.
- **Nicht identisch** sind jedoch die Text-Konstanten **'SWISS'**, **'Swiss'** und **'swiss'**.
- Konvention zur Lesbarkeit:
 - Schlüsselworte gross schreiben (z.B. CREATE TABLE)
 - Schemaelemente klein schreiben (ausser am Wortanfang)

SQL \neq SQL

- Trotz Standardisierung unterscheiden sich die Sprachimplementationen der verschiedenen RDBMS-Hersteller teils erheblich.
- Es gibt **kein** relationales Datenbankverwaltungssystem, das den **Standard vollständig** umgesetzt hat.
 - Warum ist das so?
- In den Kernfunktionalitäten besteht jedoch eine weitgehende Übereinstimmung. Mit Abweichungen muss man aber leben!
- In dieser Vorlesung beziehen wir uns auf den SQL-92-Standard! In der Praxis wird man sich meist auf einen Herstellerdialekt festlegen. Warum?

SQL – Bemerkungen

- Ein Name (TableName, TableAlias, ColumnName, ColumnAlias, ...) muss **mit einem Buchstaben beginnen**, gefolgt von Buchstaben, Ziffern oder der Unterlänge (underscore) _
- Eine Konstante (hier Literal genannt) hat, je nach Datentyp, folgende Form:
 - Eine Folge von Ziffern mit oder ohne Vorzeichen, z.B. 123 , -45
 - Eine Fix- oder Gleitkommazahl, z.B. 6.789 , -1.23E4
 - Ein Text der Form 'Dies ist ein Text' oder 'John''s Home'
- Jede Anweisung ist mit einem ; (Semikolon) abzuschliessen.

SQL – wie lernen?

- Wie lernt man eine (technische) Sprache? Was braucht man?
 - Syntaxregeln: Regeln, was in der Sprache X formal zulässig ist
 - Wie dürfen Dinge heissen?
 - In welcher Reihenfolge müssen Anweisungen erfolgen?
 - ...
 - ACHTUNG: Eine Korrekte Syntax ist lediglich eine **Voraussetzung** und heisst noch lange nicht, dass das Richtige getan wird!
 - Semantik: Aussagen, was die einzelnen Anweisungen machen
 - Ein System um zu **üben** (→ MySQL)

Erweiterte Backus-Naur Form (EBNF)

- Sprachen werden oft in EBNF (formale Syntaxbeschreibung) dargestellt.

| Symbol | Bedeutung |
|--------|---|
| " " | Bezeichnen Symbole der Sprache, die wörtlich zu übernehmen sind |
| | Alternativen werden mit einem senkrechten Strich getrennt |
| () | Runde Klammern dienen lediglich der Gruppierung. |
| [] | Eckige Klammern stehen für einen optionalen Inhalt, der Null oder einmal vorkommt. |
| { } | Geschweifte Klammern stehen für eine beliebige Wiederholung des Inhalts: 0-mal, 1-mal, 2-mal, ... |
| <> | Spitze Klammern stehen für Nichtterminale/Variablen. |
| ::= | Definition / Produktionsregel (z.b. $a ::= b$) |

SQL – Syntaxbeschreibung

- EBNF-Darstellung: Formal präzise, aber schwierig zu lesen

```

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [( ) LIKE old_tbl_name ( )];

create_definition:
    column_definition
  | [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)
  | KEY [index_name] [index_type] (index_col_name,...)
  | INDEX [index_name] [index_type] (index_col_name,...)
  | [CONSTRAINT [symbol]] UNIQUE [INDEX]
      [index_name] [index_type] (index_col_name,...)
  | FULLTEXT [INDEX] [index_name] (index_col_name,...)
      [WITH PARSER parser_name]
  | SPATIAL [INDEX] [index_name] (index_col_name,...)
  | [CONSTRAINT [symbol]] FOREIGN KEY
      [index_name] (index_col_name,...) [reference_definition]
  | CHECK (expr)

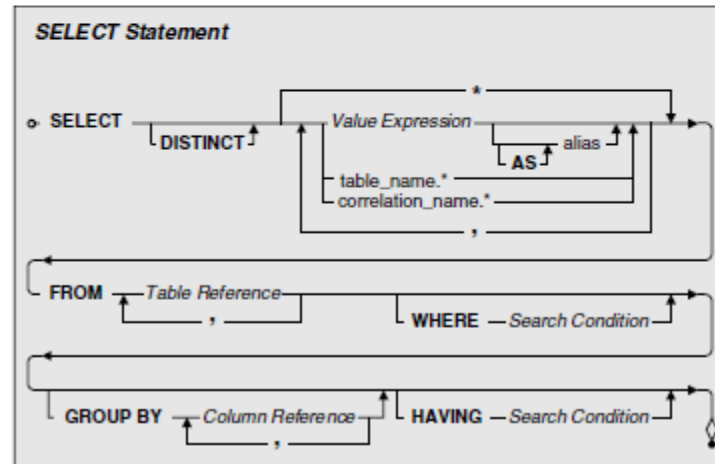
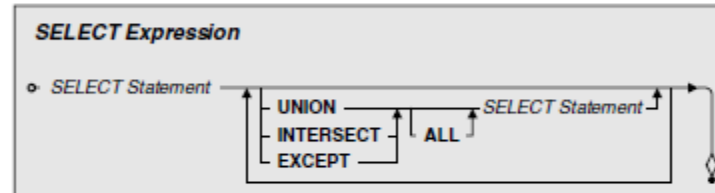
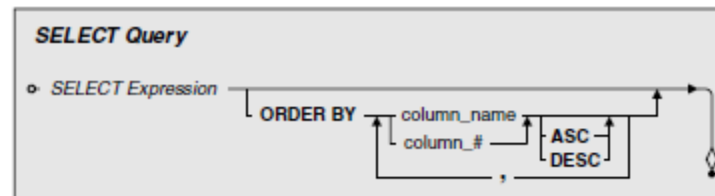
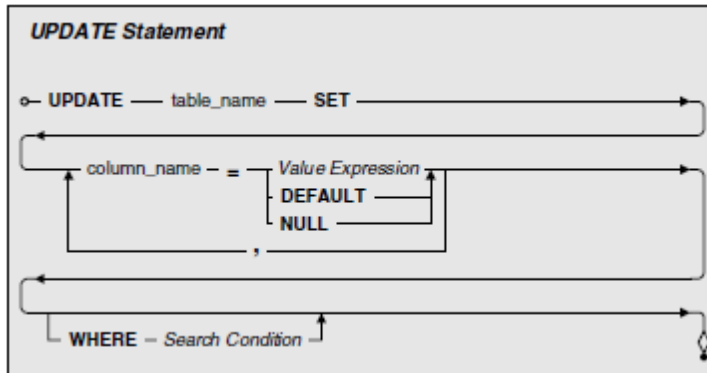
column_definition:
    col_name type [NOT NULL | NULL] [DEFAULT default_value]
      [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
      [COMMENT 'string'] [reference_definition]

type:

```

SQL – Syntaxbeschreibung

- "EBNF-grafisch": etwas einfacher zu lesen und zu verstehen



Erweiterte Backus-Naur Form (EBNF)

- Beispiel in EBNF (Anweisung zur Erzeugung einer Tabelle in SQL, Auszug):

```
"CREATE TABLE" <tableName>
```

```
"(" <tableElementDef> {, <tableElementDef> } ") ;"
```

```
tableElementDef ::= <columnDef> | <tableConstraintDef>
```

```
columnDef ::= <attributeName> <dataType>["(" <domain> ")"]  
[attributeConstraintDef]
```

```
attributeConstraintDef ::= ["CONSTRAINT" <constraintName>] {"DEFAULT"  
<defaultClause> | "NOT NULL" | "CHECK" "("checkCondition")"}
```

```
defaultClause ::= "NULL" | <constant> | <systemVariable>
```

SQL – Syntaxbeschreibung

- Was wir hier machen: "Learning by example", d.h. versuchen, Probleme durch **Analogie** zu lösen:
 - Vorteil: Viel einfacher
 - Nachteil: Unvollständig, muss durch weitere Unterlagen (Handbücher, Fachliteratur, Web-Recherchen, ...) ergänzt werden
 - Beispiel: Erstellen einer Tabelle

```
"CREATE TABLE" <tableName>
 "(" <tableElementDef> {, <tableElementDef> } ";

tableElementDef ::= <columnDef> | <tableConstraintDef>

columnDef ::= <attributeName> <dataType>["(" <domain> ")"]
[attributeConstraintDef]

attributeConstraintDef ::= ["CONSTRAINT" <constraintName>]
{"DEFAULT" <defaultClause> | "NOT NULL" | "CHECK"
 " ("checkCondition")"}

defaultClause ::= "NULL" | <constant> | <systemVariable>
```

```
create table salary_mst (
    id int not null primary key auto_increment,
    name varchar(50),
    salary double not null default 0
);
```


Handbücher / on-line-Dokumentationen

- Je nach Produkt sieht die Sprachbeschreibung anders aus (Bsp. MySQL):

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [( ) LIKE old_tbl_name ( )];

create_definition:
    column_definition
| [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)
| KEY [index_name] [index_type] (index_col_name,...)
| INDEX [index_name] [index_type] (index_col_name,...)
| [CONSTRAINT [symbol]] UNIQUE [INDEX]
    [index_name] [index_type] (index_col_name,...)
| FULLTEXT [INDEX] [index_name] (index_col_name,...)
    [WITH PARSER parser_name]
| SPATIAL [INDEX] [index_name] (index_col_name,...)
| [CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (index_col_name,...) [reference_definition]
| CHECK (expr)

column_definition:
    col_name type [NOT NULL | NULL] [DEFAULT default_value]
    [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
    [COMMENT 'string'] [reference_definition]

type:
```

DDL: Datendefinition

- DDL = Data Definition Language
- Mit Hilfe von DDL-Anweisungen werden **Datenbankobjekte erzeugt, geändert und gelöscht.**
- Es gibt viele verschiedene Datenbankobjekte:
 - Datenbanken
 - Tabellen
 - Sichten («virtuelle» Tabellen)
 - Constraints («Einschränkungen», dazu zählen auch Schlüssel)
 - Indizes
 - Stored Procedures, Triggers
 - ...

DDL: Datendefinition

- Was brauchen wir um ein ER-Schema zu implementieren?
 - Eine – zu Beginn – leere **Datenbank**
 - **Tabellen**
 - **Attribute** (Bezeichnung, Datentyp)
 - **Schlüssel:**
 - Primärschlüssel
 - Fremdschlüssel
 - Schlüssel
 - Ev. weitere Möglichkeiten, um Konsistenzbedingungen zu formulieren.

DDL: Datenbank

- Erzeugen einer leeren Datenbank (in MySQL):

```
DROP DATABASE IF EXISTS Verwaltung; -- Gut für Übungen
CREATE DATABASE Verwaltung;          -- Erzeugen
USE Verwaltung;                      -- Auswählen
```

- Ein RDBMS kann verschiedene Datenbanken verwalten.
- Bei anderen Systemen lautet die Anweisung anders!
- Achtung: In der Praxis ist die Sache komplizierter! Wir verzichten hier auf Aspekte der Sicherheit, Performanz, Datenverteilung u.a.
→ physischer Entwurf

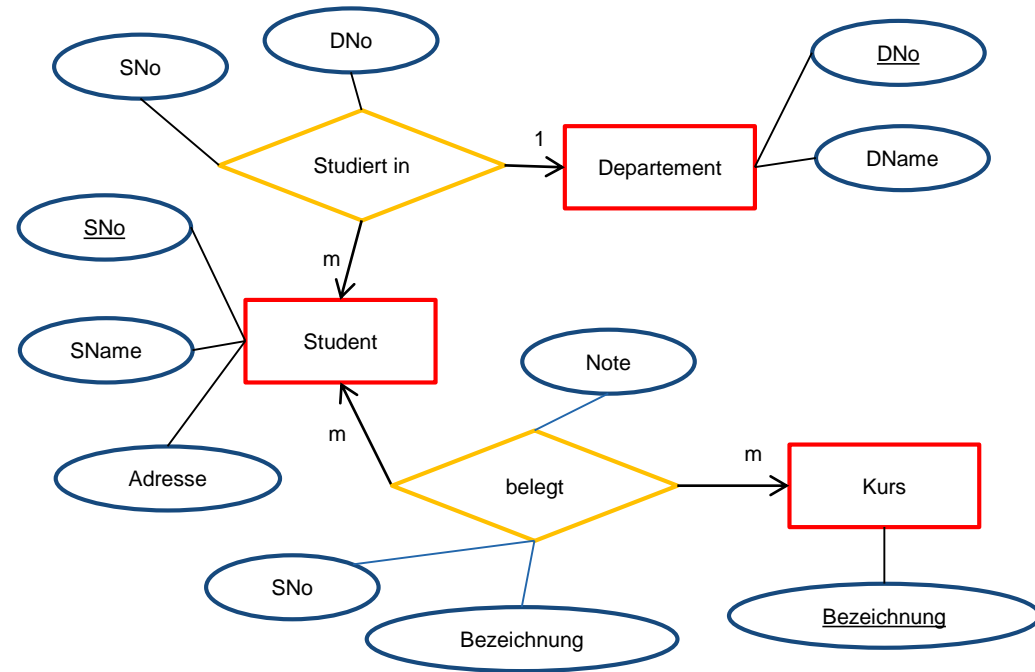
DDL: Tabelle

- Die folgenden Beispiele beziehen sich auf folgendes Schema:

- Erzeugen** einer Tabelle:

```
CREATE TABLE Departement  
(  
...  
);
```

- Departement: Name der Tabelle. Muss innerhalb der Datenbank eindeutig sein



DDL: Tabelle

- **Erzeugen** einer Tabelle:

```
CREATE TABLE Departement  
(  
    DNo char(4),  
    DName varchar(100)  
);
```

- DNo, DName: **Namen der Attribute**
- char(4), varchar(100): **Datentypen (Domänen)**

DDL: Tabelle

- **Erzeugen** einer Tabelle:

```
CREATE TABLE Departement  
(  
    DNo char(4) NOT NULL,  
    DName varchar(100) NOT NULL  
);
```

- NOT NULL: Attributwert muss immer vorhanden sein (das Tupel kann sonst nicht eingefügt werden)
- **Alternative:** DName varchar(100) NULL DEFAULT 'unbekannt'

DDL: Datentypen

- Je nach System stehen unterschiedliche **Datentypen** zur Verfügung.
- Grunddatentypen:
 - `char (n)`: String fester Länge (n)
 - `varchar (n)`: String variabler Länge, maximal n Zeichen
 - `integer`: Ganze Zahlen
 - `float, real`: Fließkommazahlen
 - `decimal (n,d)`: Festkommazahlen (n = Anzahl Stellen, d = Anzahl Nachkommastellen)
 - ...
- Dokumentation des Systems muss beigezogen werden!
- Viele Datentypen sind proprietär!

DDL: Tabelle

- **Erzeugen** einer Tabelle:

```
CREATE TABLE Departement  
(  
    DNo char(4) NOT NULL PRIMARY KEY,  
    DName varchar(100) NOT NULL  
);
```

- **PRIMARY KEY**: Attribut DNo ist Primärschlüssel (d.h. es wird von einer anderen Tabelle aus darauf verwiesen mit einem Fremdschlüssel).
- Ein Primärschlüssel kann auch aus mehreren Attributen bestehen (→ Formulierung als Constraint).

DDL: Tabelle

- **Erzeugen** einer Tabelle:

```
CREATE TABLE Departement  
(  
    DNo char(4) NOT NULL PRIMARY KEY,  
    DName varchar(100) NOT NULL UNIQUE  
);
```

- **UNIQUE**: Schlüssel (d.h. es gibt keine zwei Departemente, die die gleiche Bezeichnung haben).
- Ein Schlüssel kann auch aus mehreren Attributen bestehen (→ Formulierung als Constraint).

DDL: Tabelle

- **Erzeugen** einer Tabelle:

```
CREATE TABLE StudiertIn
(
    SNo char(8) NOT NULL,
    DNo char(4) NOT NULL,
    FOREIGN KEY (SNo) REFERENCES Student (SNo),
    FOREIGN KEY (DNo) REFERENCES Departement (DNo)
);
```

- **FOREIGN KEY...:** Fremdschlüssel der auf eine andere Tabelle verweist (diese muss bereits existieren, d.h. die Reihenfolge der CREATE-Anweisungen ist wichtig).

DDL: Tabelle

- **Erzeugen** einer Tabelle:

```
CREATE TABLE Belegt
(
  SNo char(8) NOT NULL,
  Bezeichnung varchar(100) NOT NULL,
  Note decimal(5,2) NOT NULL CHECK (Note >= 1.0),
  FOREIGN KEY (SNo) REFERENCES Student (SNo),
  FOREIGN KEY (Bezeichnung) REFERENCES Kurs (Bezeichnung)
);
```

- **CHECK**: Integritätsbedingung. Einschränkung, d.h. im Beispiel kann eine Note nicht kleiner als 1.0 sein.
- In MySQL nicht implementiert (aber syntaktisch möglich)!

DDL: Tabelle

- **Ändern** einer Tabelle:

```
ALTER TABLE Student ADD Geburtstag date NOT NULL;
```

- **ALTER**: Anweisung um etwas zu ändern.
- **ADD**: Hinzufügen eines Attributes.
- Wenn eine Datenbank sauber implementiert (und «richtig» genutzt) wird, kann sie im **laufenden Betrieb erweitert** werden (d.h. OHNE Anpassungen an bestehenden Anwendungsprogrammen)!

→ Logische Datenunabhängigkeit!

DDL: Tabelle

- **Ändern** einer Tabelle:

```
ALTER TABLE Belegt ADD CONSTRAINT  
FK_Belegt_Student FOREIGN KEY (SNo) REFERENCES Student (SNo) ;
```

```
ALTER TABLE Belegt ADD CONSTRAINT  
CHK_Note CHECK (Note >= 0.0) ;
```

- FK_Belegt_Student, CHK_Note: Einschränkungen können benannt werden.
- Man kann auch später noch weitere Einschränkungen hinzufügen.

DDL: Tabelle

- **Löschen** einer Tabelle:

```
DROP TABLE Belegt;
```

- **DROP**: Anweisung um etwas zu löschen.

- Gute Praxis (insbesondere für Übungen):

```
DROP TABLE IF EXISTS Belegt;  
CREATE TABLE Belegt...;
```

DDL: Tabelle

- UNIQUE: **mehrere** Unique-Klauseln **pro Tabelle** möglich (→ Schlüssel!)
- Die Werte der Attributsmenge jeder Unique-Klausel müssen in jedem Tupel verschieden sein, wird bei der Dateneingabe vom DBMS überprüft
- Primary Key (Primärschlüssel): es kann **höchstens einen PK** pro Tabelle geben
- Ein PK ist dann nötig, wenn die Tabelle von einer anderen (mit Foreign Key) referenziert wird. Ansonsten genügt eine Unique-Klausel.
Gängige Praxis: IMMER einen PK definieren!
- Beispiel CD-Shop, Tabelle, BestellPosition
`CONSTRAINT BestPosPK PRIMARY KEY(bestNr, posNr)`

DDL: Foreign Key

```
"FOREIGN KEY" "(" <attributeName>{, <attributeName>}  
")" "REFERENCES" <tableName> ["(" <attributeName>{,  
<attributeName>} ")"]  
[impliziter Foreign Key-Trigger {impliziter Foreign  
Key-Trigger}]
```

```
"FOREIGN KEY" "(" <attributeName>{, <attributeName>}  
")" "REFERENCES" <tableName>
```

→ ist die minimale Angabe

DDL: Foreign Key

- Falls in Table <tableName> keine Attribute genannt werden, wird automatisch der Primary Key von <tableName> referenziert
- D.h.: durch explizite Angabe können auch andere Attribute als die des PK referenziert werden. Davon ist Anfängern **abzuraten!**
- Anhand dieser Referenz sichert das DBMS bei Dateneingabe, aber auch bei Löschen von Tabellen, die **referentielle Integrität**

- Beispiel CD-Shop, Tabelle ‚HatStil‘ (2 Fremdschlüssel!):

```
CONSTRAINT HatStFKStil FOREIGN KEY (stil) REFERENCES Musikstil,  
CONSTRAINT HatStFKCD FOREIGN KEY (eanNr) REFERENCES CD
```

- Besser:

```
CONSTRAINT HatStFKStil FOREIGN KEY (stil)  
REFERENCES Musikstil(stil),  
CONSTRAINT HatStFKCD FOREIGN KEY (eanNr) REFERENCES CD(eanNr)
```

DDL: Foreign Key

- Implizite Foreign Key-Trigger sind Aktionen, die vom DBMS automatisch bei Dateneingabe ausgeführt werden

FK-Trigger ::= ("ON DELETE" | "ON UPDATE") ("NO ACTION" | "SET NULL" | "SET DEFAULT" | "CASCADE")

DDL: Tabelle

- ON UPDATE: wenn ein Tupel **in der referenzierten Tabelle** geändert oder eingefügt wird
- ON DELETE: wenn ein Tupel **in der referenzierten Tabelle** gelöscht wird
- NO ACTION: falls der Wert des Fremdschlüssels nach der Aktion keinem gültigen Primärschlüsselwert der referenzierten Tabelle mehr entsprechen würde, wird die Aktion verboten
- SET NULL, SET DEFAULT: selbsterklärend (SET NULL = **pfui**)

DDL: Tabelle


- CASCADE: Werte des Fremdschlüssels werden bei Ändern des PK-Werts der referenzierten Tabelle automatisch angepasst
- **ACHTUNG**: wird das referenzierte Tupel gelöscht, werden alle Tupel *dieser* Tabelle mit diesem FK-Wert ohne Warnung gelöscht (ON DELETE CASCADE)

Beispiel CD-Shop: Bestellung löschen → alle zugehörigen BestellPositionen werden auch gelöscht

DDL: Tabelle

- Beispiel CD-Shop, Tabelle ‚HatStil‘ (betrachten nur Fremdschlüssel zu ‚Musikstil‘) :

```
CONSTRAINT HatStFKStil FOREIGN KEY (stil) REFERENCES Musikstil  
ON UPDATE CASCADE,          FK-Element erst hier fertig  
ON DELETE CASCADE;
```



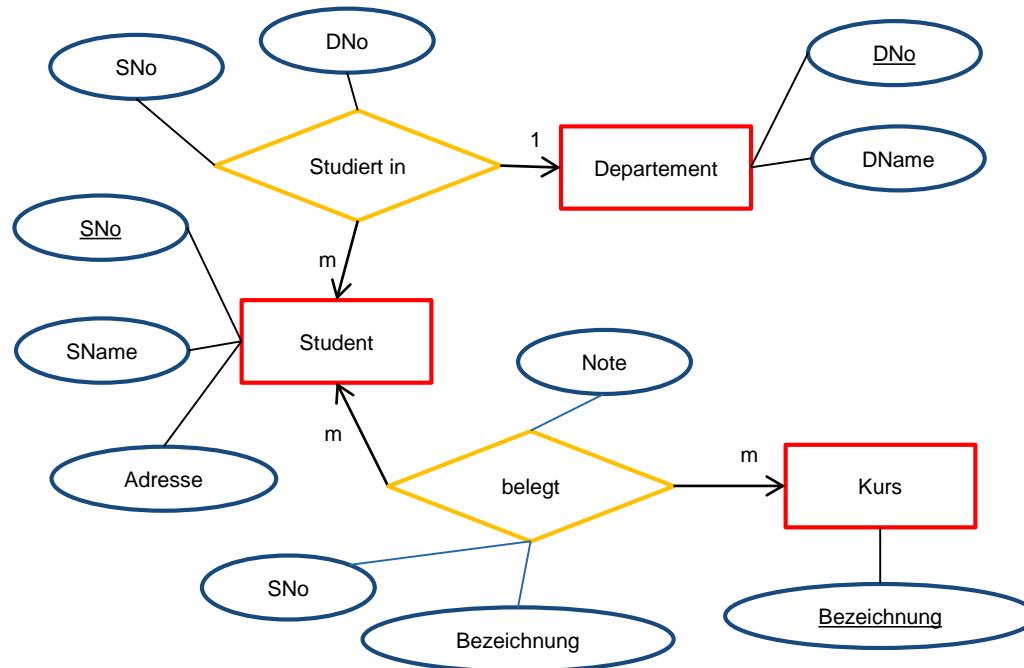
- Hat folgende Auswirkungen:
 - wird in ‚Musikstil‘ ein Stil geändert (z.B. von ‚HippHopp‘ auf ‚HipHop‘), so werden alle Tupel von ‚HatStil‘, deren Attributswerte von ‚stil‘ ‚HippHopp‘ enthalten, auf ‚HipHop‘ geändert
 - wird in ‚Musikstil‘ der Stil ‚Blasmusik‘ gelöscht, so werden in ‚HatStil‘ alle **Tupel** mit ‚Blasmusik‘ ebenfalls **gelöscht**! Die entsprechenden CDs sind danach keinem Stil mehr zugeordnet.

DDL: Zusammenfassung

- Zweck:
 - Objekte erzeugen, ändern, löschen
- Wichtigste Anweisungen:
 - CREATE ...
 - ALTER ...
 - DROP ...
- DDL ist das wesentliche Hilfsmittel für:
 - DatenbankAUFBAU
 - Spätere strukturelle (nicht inhaltliche!) Anpassungen an den Objekten
- Viele, produktspezifische Syntax-Erweiterungen im Gebrauch

DDL: Hörsaalübung

- Aufgabe: Schreiben Sie ein SQL-Skript, das untenstehendes ER-Schema vollständig implementiert. Nutzen Sie dazu MySQL-Workbench (wesentliche Teile davon können aus den Vorlesungsslides direkt übernommen werden). Wählen Sie die Datentypen geeignet.



Und weiter...

- Das nächste Mal: SQL (DML, Queries)

