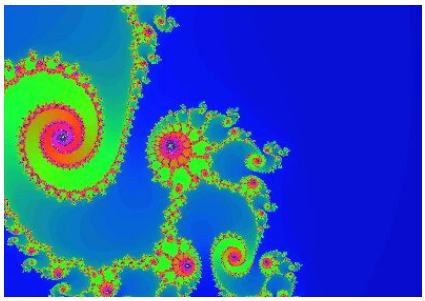
# Datenbanken

**6. Datendefinition mit SQL DDL**

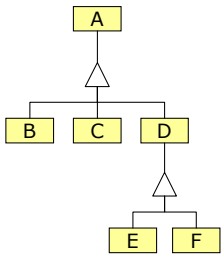


Fachhochschule Darmstadt
Fachbereich Informatik

Dr. Peter Nevermann

## Rückblick

- **Erweitertes ER-Modell**
  - *Weak Entities*
  - *Spezialisierung/Generalisierung*
- **ER-Diagramm**



## Datenbank erzeugen

- **Erzeuge eine Datenbank mit dem Namen "firma2"**

MySQL:

```
CREATE DATABASE firma2
   CHARACTER SET latin1
   COLLATE latin1_german1_ci;

USE firma2;
```



Nachdem wir mit dem ER Modell Mittel kennengelernt haben, um die relevanten Aspekte einer Miniwelt zu beschreiben, gehen wir jetzt den nächsten Schritt, nämlich die Überführung eines konkreten ER Entwurfs in ein Datenbankschema.

**SQL DDL**

Dazu brauchen wir zunächst einen weiteren SQL Bestandteil, nämlich SQL DDL (*Data Definition Language*). Bisher haben wir mit SQL DML (*Data Manipulation Language*) uns darauf beschränkt, auf bestehende Tabellen in bestehenden Datenbanken zugegriffen.

Mit SQL DDL kann man:

•Datenbanken erzeugen und löschen

•Tabellendefinitionen innerhalb einer Datenbank erzeugen, verändern und löschen

•Attributdefinitionen innerhalb einer Tabelle erzeugen, verändern und löschen

•Integritätsregeln innerhalb einer Tabelle/Datenbank erzeugen und löschen

**Datenbank erzeugen**

Mit dem Befehl CREATE DATABASE wird in vielen DBMS (z.B. MySQL) eine neue Datenbank angelegt. In MySQL hat der Befehl noch die Parameter CHARACTER SET um die Zeichencodierung festzulegen (latin1, utf8, ascii, etc.) und COLLATE um die nationale Sortierreihenfolge festzulegen (*collation*). Diese Werte legen Standardwerte für die Datenbank fest, und können auf Tabellen- und Spaltenebene umdefiniert werden.

**MySQL CHARACTER SET und COLLATE**

Mit folgenden MySQL Befehlen kann man herausfinden welche Werte es für CHARACTER SET und COLLATE gibt, und wie diese kombiniert werden können:

mysql> SHOW CHARACTER SET;

mysql> SHOW COLLATION;

Aus der MySQL Dokumentation zu den deutschen Sortierungen:

<zitat>The latin1_german1_ci and latin1_german2_ci collations are based on the DIN-1 and DIN-2 standards, where DIN stands for Deutsches Institut für Normung (that is, the German answer to ANSI). DIN-1 is called the dictionary collation and DIN-2 is called the phone-book collation.

latin1_german1_ci (dictionary) rules:

•Ä = A

•Ö = O

•Ü = U

•ß = s

latin1_german2_ci (phone-book) rules:

•Ä = AE

•Ö = OE

•Ü = UE

•ß = ss

</zitat>

In MySQL wird noch mit dem Befehl USE die Datenbank selektiert, mit der gearbeitet werden soll. Man könnte auch sagen, daß mit USE eine Datenbank geöffnet wird.

**SHOW CREATE DATABASE in MySQL**

Hat man bereits eine Datenbank in MySQL, so kann man mit folgendem Befehl den CREATE DATABASE Befehl ausgeben, der zur angegebenen Datenbank passt:

mysql> SHOW CREATE DATABASE firma2\G

*************************** 1. row ***************************

Database: firma2

Create Database: CREATE DATABASE `firma2` /*!40100 DEFAULT CHARACTER SET latin1 COLLATE latin1_german1_ci */

1 row in set (0.00 sec)

**SQL2 Schema**

Im SQL2 Standard gibt es noch CREATE SCHEMA, wobei ein "Schema" als Mittel zur Gruppierung von Tabellen dient [wird aber weder in MySQL noch in Access unterstützt].

## Tabelle erzeugen

- **Erzeuge eine Tabelle "abteilung" für die Miniwelt "Firma"**

```
CREATE TABLE abteilung (
  abtname              VARCHAR(50)   NOT NULL,
  anr                  INT           NOT NULL,
  mgrpnr               INT,
  mgrstartdat          DATE,
  PRIMARY KEY (anr),
  UNIQUE (abtname)
);
```

Datenbanken I | 5 | SS 05, 20.05.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

In einer SQL Tabellendefinition werden die Spalten (Attribute) einer Tabelle jeweils mit ihren Datentypen festgelegt. Außerdem werden innerhalb der Tabellendefinition noch Integritätsregeln festgelegt.

**Tabellendefinition**

Eine Tabellendefinition in SQL DDL hat die Form:

```
CREATE TABLE <tabellenname> (
  <attributname-1>    <datentyp-1>  [<integritätsregel-1>],
  <attributname-2>    <datentyp-2>  [<integritätsregel-2>],
  ...                 ...           ...
  [PRIMARY KEY (<schlüssel-att1, schlüssel-att2, ...>)],
  [FOREIGN KEY (<fremdschlüssel-att>) REFERENCES <tabellenname>(schlüsselatt)],
  [<weitere_integritätsregeln>]
);
```

Der Tabellenname muß eindeutig innerhalb der Datenbank sein, Attributnamen sind eindeutig innerhalb einer Tabelle.

In SQL gibt es eine Reihe von Datentypen für numerische, text- oder datumbezogene Attribute. Außerdem kommen bei jedem DBMS noch einige proprietäre Datentypen hinzu, die im Standard nicht definiert sind.

**Datentypen**

Im Standard (und/oder MySQL) definierte Datentypen sind u.a.

Numerisch

•INT oder INTEGER
Ganzzahlig -2147483648 bis 2147483647

•SMALLINT
Ganzzahlig -32768 bis 32767

•TINYINT (MySQL)
Ganzzahlig -128 to 127

•FLOAT
Gleitpunktzahl -3.402823466E+38 bis -1.175494351E-38, 0, 1.175494351E-38 bis 3.402823466E+38

•DOUBLE oder REAL
Gleitpunktzahl -1.7976931348623157E+308 to -2.2250738585072014E-308, 0, 2.2250738585072014E-308 to 1.7976931348623157E+308

•DEC oder DECIMAL oder NUMERIC oder FIXED
Fixpunktzahl

Mit Argumenten hinter dem Datentyp kann man die Stelligkeit/Länge des Attributwertes eingrenzen, z.B.

•INT(10) – ganzzahlig 10 Stellen

•DEC(5,2) – 5 Vorkomma- und 2 Nachkommazahlen.

Datum/Zeit

•DATE
Typischerweise im Format YYYY-MM-DD, aber viele andere Formate möglich

•TIME
Typischerweise im Format HH:MM:SS

•DATETIME
Typischerweise im Format YYYY-MM-DD HH:MM:SS
In MySQL: '1000-01-01 00:00:00' bis '9999-12-31 23:59:59'

•TIMESTAMP
In MySQL: ab '1970-01-01 00:00:00' bis ins Jahr 2037

String/Text

•CHAR(n) oder CHARACTER(n)
Zeichenkette der Länge n
Dabei ist CHAR äquivalent mit CHAR(1).

•VARCHAR(n) oder CHARACTER VARYING(n)
Zeichenkette variabler Länge, maximal n

•TEXT
Text mit max. 65.535 Zeichen (~64 KB)

•MEDIUMTEXT
Text mit max. 16.777.215 Zeichen (~16 MB)

•LONGTEXT
Text mit max. 4.294.967.295 Zeichen (~4 GB)

Weitere Datentypen

•BIT oder BOOL oder BOOLEAN
Boolesche Werte TRUE und FALSE
In MySQL äquivalent zu TINYINT(1) mit den Werten 1 und 0

•ENUM ('wert1', 'wert2', ..., 'wertn')
Aufzählung von String-Konstanten
Attribute diese Typs können nur die aufgezählten Werte oder NULL annehmen

**Standardwerte (Default-Werte)**

In einer Attributdefinition können Standardwerte für den Attributwert festgelegt werden. Zum Beispiel könnte ein Attribut "farbe" einer Tabelle "Fahrzeug" in einer Datenbank der Deutschen Post AG wie folgt definiert sein:

```
farbe    CHAR(10) DEFAULT 'gelb'
```

Wird ein Fahrzeug-Tupel angelegt ohne Angabe der Farbe, so bekommt dieses Attribut automatisch den Wert 'gelb'.

Wird keine Standardwertdefinition gemacht, so ist der Default-Wert NULL.

**MySQL Storage Engines**

MySQL verfügt über unterschiedliche "Storage Engines", wie z.B. MyISAM, InnoDB, BDB, usw.. Bei der Erzeugung von Tabellen mit CREATE TABLE kann man die Tabelle inividuell einer solchen Engine zuordnen, und zwar mit dem Schlüssel ENGINE (in älteren MySQL Versionen mit TYPE), z.B.:

```
CREATE TABLE mitarbeiter (
 ...
) ENGINE= InnoDB;
```

Ohne Angabe einer Engine wird standardmäßig die "MyISAM" Engine zugeordnet. Allerdings sind die unterschiedlichen Engines auch unterschiedlich mächtig und effizient. So wird z.B. *referentielle Integrität* (kommt gleich ... siehe FOREIGN KEY .. REFERENCES ..) derzeit nur von der InnoDB Engine unterstützt.

Mit folgendem MySQL Befehl bekommt man eine Liste der verfügbaren Engines:

```
mysql> SHOW ENGINES;
```

**SHOW CREATE TABLE in MySQL**

Hat man bereits eine Tabelle, kann man mit dem MySQL Befehl SHOW CREATE TABLE den SQL DDL Befehl ausgeben, der zur der Tabellendefinition führt:

```
mysql> SHOW CREATE TABLE standort\G
*************************** 1. row ***************************
       Table: standort
Create Table: CREATE TABLE `standort` (
  `abtnummer` int(11) NOT NULL default '0',
  `stadt` varchar(50) collate latin1_german1_ci NOT NULL default '',
  PRIMARY KEY  (`abtnummer`,`stadt`),
  CONSTRAINT `standort_ibfk_1` FOREIGN KEY (`abtnummer`) REFERENCES `abteilung` (`anr`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_german1_ci
1 row in set (0.00 sec)
```

## Integritätsregeln

- **Erzeuge eine Tabelle "mitarbeiter" für die Miniwelt "Firma"**

```
CREATE TABLE mitarbeiter (
  nachname            VARCHAR(50)    NOT NULL,
  vorname             VARCHAR(50)    NOT NULL,
  pnr                 INT            NOT NULL,
  gebdatum            DATE,
  adresse             VARCHAR(150),
  geschlecht          ENUM('M','W') NOT NULL,
  gehalt              INT            NOT NULL
                                     CHECK (gehalt > 0),
  vpnr                INT,
  abtnr               INT,
  PRIMARY KEY (pnr),
  FOREIGN KEY (abtnr) REFERENCES abteilung(anr)
     ON UPDATE CASCADE ON DELETE SET NULL,
  FOREIGN KEY (vpnr)  REFERENCES mitarbeiter(pnr)
     ON UPDATE CASCADE ON DELETE SET NULL
);
```

Datenbanken I | 9 | SS 05, 20.05.2005 | FH Darmstadt, FB Informatik, Dr. P. Nevermann

Integritätsregeln (IR) werden vom DBMS überwacht und können innerhalb der Tabellendefinition entweder auf Attribut- oder auf Tabellenebene festgelegt werden.

Auf Attributebene durch:

•die Zuordnung eines Datentyps (stellt implizit eine IR dar)

•Angabe von NOT NULL (Attributwerte dürfen nicht NULL sein)

•CHECK Regeln

Auf Tabellenebene (im Anschluß an die Attributdefinitionen) durch:

•Festlegen des Primärschlüssels mit PRIMARY KEY (...)

•Festlegen von Sekundärschlüsseln mit UNIQUE (...)

•Definition von Fremdschlüsseln mit FOREIGN KEY (...) REFERENCES <tabellenname> (...)

•CHECK Regeln

**Referentielle Integrität**

Durch Definition von Fremdschlüsseln wird die *referentielle Integrität* der Daten durch das Datenbanksystem überwacht. Ansich handelt es sich hierbei um IR auf Datenbankebene, da referentielle Integrität tabellenübergreifend ist. Bei der Definition eines Fremdschlüssels (*foreign key*, FK) wird eine Beziehung zum Primärschlüssel (*primary key*, PK) einer anderen Tabelle (oder auch derselben Tabelle) hergestellt:

FOREIGN KEY (fk1, ..., fkn) REFERENCES selbe_oder_andere_tabelle (pk1, ..., pkn)
  ON UPDATE <update-action>
  ON DELETE <delete-action>,

Für <update-action> und <delete-action> gibt es folgende Möglichkeiten:

•**CASCADE**
update: bei Änderung des referenzierten PK Wertes soll der FK Wert automatisch entsprechend geändert werden
delete: bei Löschung des referenzierten Tupels soll dieses Tupel automatisch gelöscht werden

•**SET NULL**
update: bei Änderung des referenzierten PK Wertes soll der FK Wert automatisch auf NULL gesetzt werden
delete: bei Löschung des referenzierten Tupels soll der FK dieses Tupels automatisch auf NULL werden

•**SET DEFAULT**
wie SET NULL aber der FK wird auf den Default-Wert gesetzt

•**RESTRICT**
verhindert die Änderung oder Löschung, falls eine Instanzen-Referenz besteht

•**NO ACTION**
keine Aktion auf FK Seite bei Änderung oder Löschung auf PK Seite

**Integritätsregeln mit CHECK**

Mit CHECK-Ausdrücken können Integritätsregeln auf Attribut- oder Tabellenebene definiert werden.

Beispiele:

•Attributebene:
Gehalt muß positiv sein
gehalt INT CHECK (gehalt > 0),

•Tabellenebene (im Anschluß an die Attributdefinitionen in der CREATE TABLE Anweisung):
Personalnummer des Vorgesetzten muß kleiner als eigene Personalnummer sein (UNSINNIG aber MÖGLICH!)
CHECK (vpnr < pnr),

**Integritätsregeln einen Namen geben (CONSTRAINT)**

Man kann einer IR einen Namen zuordnen, der eindeutig innerhalb der Tabelle sein muß:

```
CONSTRAINT pk PRIMARY KEY (pnr),
CONSTRAINT fk_abteilung FOREIGN KEY (abtnr) REFERENCES abteilung(anr)
  ON UPDATE CASCADE ON DELETE SET NULL,
CONSTRAINT fk_vorgesetzter FOREIGN KEY (vpnr)  REFERENCES mitarbeiter(pnr)
  ON UPDATE CASCADE ON DELETE SET NULL
```

Das hat den Nutzen, daß man später die IR benennen kann um sie z.B. nachträglich wieder zu löschen oder zu ersetzen (➔ ALTER TABLE ... kommt gleich!).

## Schema-Evolution

- **Die Struktur der Tabelle "abteilung" muß nachträglich geändert werden?**

```
ALTER TABLE abteilung
  ADD kostenstelle int  NOT NULL,
  DROP mrgstartdat
;
```



Mit dem SQL DDL Befehl ALTER TABLE kann man nachträglich eine Tabellendefinition verändern/erweitern:

- Attribute hinzufügen oder entfernen
- Attributdefinition verändern (z.B. anderer Datentyp, neue IR, ...)
- Integritätsregeln hinzufügen oder entfernen

In der Regel ist das auch dann möglich, wenn die Tabelle bereits Daten enthält. Man spricht dann von *Schema-Evolution*.

Der ALTER TABLE Befehl hat die folgende Form:

ALTER TABLE <tabellen-name>
*alter-anweisung-1*,
*alter-anweisung-2*,
... ;

Dabei kann eine *alter-anweisung* folgendermaßen aussehen:

•ADD <attribut-name> oder ADD COLUMN <attribut-name>

•ADD PRIMARY KEY (...) oder ADD CONSTRAINT PRIMARY KEY (...)

•ADD UNIQUE (...) oder ADD CONSTRAINT UNIQUE (...)

•ADD FOREIGN KEY (...) REFERENCES ... oder ADD CONSTRAINT FOREIGN KEY (...) REFERENCES ...

•ALTER <attribut-name> SET DEFAULT ... oder ALTER <attribut-name> DROP DEFAULT

•CHANGE <bisheriger-attributname> <neue-attributdefinition-evtl.-mit-neuem-namen>

•DROP <attribut-name>

•DROP PRIMARY KEY

•DROP FOREIGN KEY <integritätsregel-name>

•RENAME TO <neuer-tabellenname>

•... und noch eine Reihe weiterer Möglichkeiten, z.T. DBMS-spezifisch

In SQL2 ist pro ALTER TABLE Befehl allerdings nur eine *alter-anweisung* erlaubt. Die Möglichkeit mehrerer Anweisungen in einem ALTER TABLE Befehl ist eine MySQL Erweiterung des Standards.

## Tabelle oder Datenbank löschen

- **Die Tabelle "abteilung" soll gelöscht werden, danach die ganze Datenbank "firma2"**

```
DROP TABLE abteilung;
```

```
DROP DATABASE firma2;
```



Kurz und schmerzlos: mit DROP TABLE und DROP DATABASE können ganze Tabellen oder sogar ganze Datenbanken gelöscht werden.

Natürlich verschwinden mit DROP TABLE sowohl die Tabellendefinition aus dem Katalog der Datenbank wie auch die Daten der Tabelle. Mit DROP DATABASE ist die Datenbank komplett weg, samt Katalog und Daten.

Für die Befehle DROP DATABASE, DROP TABLE (wie übrigens auch für CREATE DATABASE und CREATE TABLE) kann noch IF EXISTS oder IF NOT EXISTS verwendet werden, wenn man eine Existenzbedingung an die Erzeugung oder Löschung koppeln will:

Beispiele:

```
DROP DATABASE IF EXISTS firma2;
CREATE DATABASE IF NOT EXISTS firma2
   CHARACTER SET latin1   COLLATE latin1_german1_ci;
 USE firma2;

DROP TABLE IF EXISTS mitarbeiter;
DROP TABLE IF EXISTS abteilung;

CREATE TABLE IF NOT EXISTS abteilung ();
```

## Ausblick

- **Transfomation eines ER Modells in ein Datenbankschema**