Prof. DI Dr. Erich Gams

Datendefinition in SQL

Einführung und Anwendung in

informationssysteme htl-wels

To start….

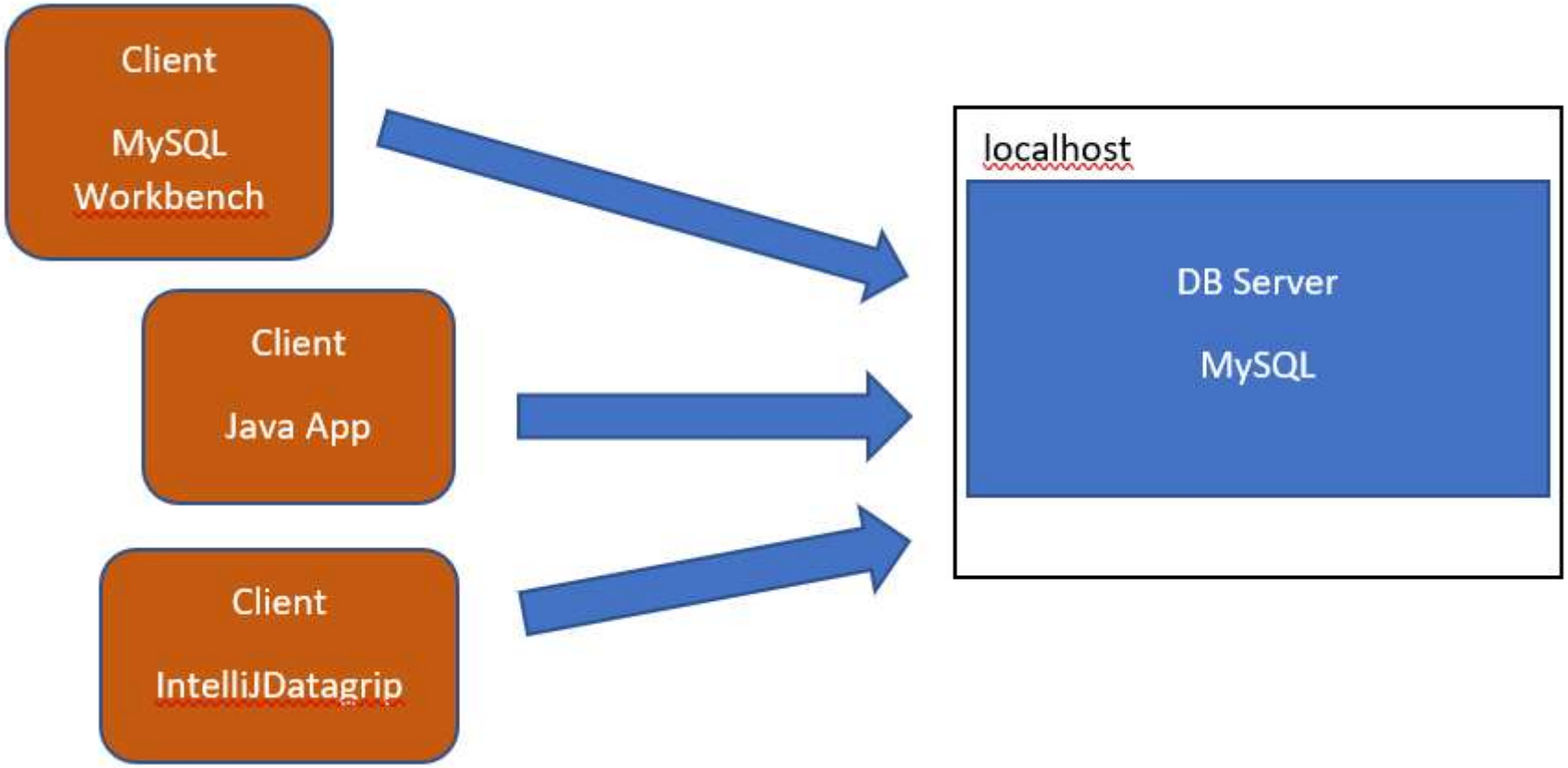
Please, close your laptops

and just

Übersicht ☛ Was lernen wir?

- SQL Definition und Geschichte
- Grundlagen und Datentypen
- Tabellenerstellung/veränderung
- Daten befüllen
- Constraints

MySQL Aufbau



SQL Einführung

- „SQL (Structured Query Language) ist Datenbanksprache zur Definition, Abfrage und Manipulation von Daten in relationalen Datenbanken.“ (Wikipedia)
- Abgeleitet von SEQUEL (Structured English Query Language) basierend auf der Originalsprache SQUARE (Specifying Queries As Relational Expressions).
- SEQUEL wurde dann in SQL umbenannt.

SQL Einführung

- **SQL ist ein ISO- und ANSI-Standard.**
- Da sich SQL im Laufe ihrer Geschichte weiterentwickelt hat, kann man jedoch nicht von einem einzigen Standard sprechen.
- Die meisten Datenbankmanagementsysteme unterstützen **SQL92**.

SQL Einführung

- sogenannte “Sprache der vierten Generation”
- deskriptiv nicht wie etwas berechnet wird, sondern was das Ergebnis sein soll.
- **SQL ist einer der Hauptgründe für den kommerziellen Erfolg von relationalen Datenbanken in der Geschäftswelt**

SQL Standard

- Anwendungsprogramme sollen vom **verwendeten Datenbanksystem unabhängig** sein.
- **Heutige Datenbanksysteme implementieren** mehr oder **weniger große Teile des Sprachstandards**.
- Darüber hinaus stellen sie oftmals **herstellerspezifische Erweiterungen** bereit, die nicht dem Standard-Sprachumfang entsprechen

SQL Einführung

- SQL-86 oder SQL1
  - Standard-Version von SQL (ANSI 1986).
- SQL2 oder SQL-92
  - 90er Jahren
- SQL3 oder SQL-99
  - im Moment gültige Standard, aber noch nicht von allen DBMS unterstützt
- SQL 2003
  - bietet erweiterte Unterst ützung von Nested Tables, von Merge Operationen und auf XML bezogene Eigenschaften
- SQL 2006
  - Brücke zu XML, XQuery, nicht in DBMS realisiert
- Vereinfachung der Migration zwischen einzelnen DBMS-Produkten (die den Standard implementiert haben).
- Egal welches DBMS-Produkt, die Schnittstelle (SQL) zu den einzelnen Systemen bleibt immer die gleiche.
- **Wir halten uns an den "traditionellen" SQL92 Standard**

SQL - Grundlagen

- Die Grundlage für die Datenmanipulation im Relationenmodell bildet die relationale Algebra.
- Bei SQL handelt es um eine deskriptive, mengenorientierte Sprache.
- Sie ist sowohl selbständig, als auch eingebettet in eine Wirtssprache (C, C++, Java, PHP, ASP, u.v.m.) verfügbar.

SQL Grundlagen

- SQL kann in drei konzeptionelle Einheiten aufgeteilt werden.

  - Datendefinition (DDL – Data Definition Language)
    - zuständig für die Erstellung und Veränderung der Struktur der Datenbank
  - Datenmanipulation (DML – Data Manipulation Language)
    - zuständig für den (Daten-) Inhalt der Datenbank (Daten hinzufügen, ändern, löschen, abfragen,...)
  - Datenkontrolle / -steuerung (DCL – Data Controlling Language)
    - zuständig für die Sicherheit der Datenbank

SQL – Programmierschnittstellen

- Interaktives ("stand-alone") SQL für direkte Befehle an die Datenbank, auch über eine grafisch orientierte Benutzerschnittstelle (GUI).
  - Herkömmliche Programmierschnittstellen erlauben die direkte Übergabe von SQL-Befehlen an Datenbanksysteme über Funktionsaufrufe. Beispiele: ODBC, JDBC
- Eingebettetes ("embedded") SQL für die gängigen Programmiersprachen
  - SQL-Anweisungen können im Quelltext eines Programms, typischerweise in C, C++, COBOL, Ada, Pascal o. Ä. geschrieben, eingebettet werden, Beispiel SQLJ

embedded SQL - SQLJ

- SQLJ erfordert eine Vorkompilierung.
- Das SQLJ-Umsetzungsprogramm konvertiert Code, der in Java-Code eingebettete SQLJ-Klauseln enthält, in reinen Java-Code, der die SQLJ-Laufzeitbibliothek aufruft.
- Vorteile:
  - SQLJ-Umsetzungsprogramm überprüft die Syntax von SQL-Anweisungen während der Umsetzung.
  - SQLJ überprüft den Typ von statischem SQL-Code mithilfe von Datenbankverbindungen
  - Man kann Java-Variablen in SQL-Anweisungen einbetten.

- Beispiel:
- #sql [ctx] cursor1 = {SELECT EMP_ACT.EMPNO FROM EMP_ACT WHERE EMP_ACT.PROJNO = :strProjNo};

SQL – Programmierschnittstellen

- Integration von SQL in "4th Generation Language"
  z.B. Oracle PL/SQL, Postgres PL/pgSQL
- Persistenz-Frameworks wie etwa Hibernate
  abstrahieren vom Datenbankzugriff und ermöglichen objektorientierte Verarbeitung von relationalen Datenbanken in einer objektorientierten Programmiersprache (z. B. Java oder C#)

Transaktion

- **Ein Bündel von Aktionen, die in der Datenbank durchgeführt werden,**
  - um diese von einem konsistenten Zustand wieder in einen konsistenten (widerspruchsfreien) Zustand zu überführen.
- Dazwischen sind die Daten zum Teil zwangsläufig inkonsistent.

- **Eine Transaktion ist atomar, d. h. nicht weiter zerlegbar.**
- Innerhalb einer Transaktion werden entweder alle Aktionen oder keine durchgeführt.
- Nur ein Teil der Aktionen würde zu einem inkonsistenten Datenbankzustand führen.

MySQL Datentypen

Ganzzahlen

Typ	vorzeichenbehaftet		vorzeichenlos	
	Minimalwert	Maximalwert	Minimalwert	Maximalwert
TINYINT	-128	+127	0	255
SMALLINT	-32.768	+32.767	0	65.535
MEDIUMINT	-8.388.608	+8.388.607	0	16.777.215
INT/INTEGER	-2.147.483.647	+2.147.483.646	0	4.294.967.295
BIGINT (*)	-2^{63}	$+2^{63} - 1$	0	$2^{64} - 1$

Fließkommazahlen

Typ	vorzeichenbehaftet		vorzeichenlos	
	Minimalwert	Maximalwert	Minimalwert	Maximalwert
FLOAT	$-3{,}402823466 \times 10^{38}$	$-1{,}175494351 \times 10^{-38}$	$1{,}175494351 \times 10^{-38}$	$3{,}402823466 \times 10^{38}$
DOUBLE	$-1{,}798 \times 10^{308}$	$-2{,}225 \times 10^{-308}$	$2{,}225 \times 10^{-308}$	$1{,}798 \times 10^{308}$

MySQL Datentypen

Typ	Format	Beispiel
DATE	YYYY-MM-DD	2008-07-10
TIME	hh:mm:ss	12:35:17
DATETIME/TIMESTAMP	YYYY-MM-DD hh:mm:ss	2008-07-10 12:35:17
YEAR	YYYY	1966

Zeichenketten

Typ	Wertebereich	Anmerkung
CHAR	0 - 255 Zeichen	feste Länge *
VARCHAR	0 - 255 Zeichen (**)	variable Länge
TINYTEXT	0 - 255	feste Länge *
TEXT	0 - 65535	feste Länge *
MEDIUMTEXT	0 - 16.777.215	feste Länge *
LONGTEXT	0 - 4.294.967.295	feste Länge *

Binärdaten

Typ	Wertebereich	Anmerkung
TINYBLOB	bis 2^8 Byte	feste Länge *
BLOB	bis 2^{16} Byte	feste Länge *
MEDIUMBLOB	bis 2^{24} Byte	feste Länge *
LONGBLOB	bis 2^{32} Byte	feste Länge *

Datentypen

Beschreibung	Datentyp	Kurzform
Zeichen(ketten):		
Ein einzelnes Zeichen	CHARACTER	CHAR
Kette fester Länge	CHARACTER (n)	CHAR (n) *)
Kette variabler Länge	CHARACTER VARYING (n)	VARCHAR (n)
Bit-Datentyp:		
Ein Bit	BIT	
Bitfolge fester Länge	BIT (n)	
Bitfolge variabler Länge	BIT VARYING (n)	
Exakte Zahlen:		
mit Nachkommastellen	DECIMAL [(p,[,s])]	DEC(p,s)
dto.	NUMERIC [(p,[,s])]	
Ganzzahl	INTEGER	INT
kleine Ganzzahl	SMALLINT	
Gleitkommazahlen:		
hohe Genauigkeit	DOUBLE PRECISION	
benutzerdefinierte Gen.	FLOAT (n)	
geringere Genauigkeit	REAL	

*) n=integer, p, s ebenso.

Datentypen

Beschreibung	Datentyp	Kurzform
Zeit, Datum, Zeitintervall:		
Datum (y,m,d)	DATE	
Zeitpunkt (h,min,sec)	TIME	
Zeitpunkt (y,m,d,h, min, sec)	TIMESTAMP	
Zeitintervall	INTERVAL f INTERVAL sf to ef f, sf, ef ∈ {YEAR, MONTH, DAY, HOUR, MINUTE, SECOND} („sf>ef“)	

Neue Datenbank/Schema anlegen

```
CREATE DATABASE [IF NOT EXISTS] database_name
```

`CREATE SCHEMA` statement command has the same effect

Datendefinition – Anlegen von Tabellen

- In SQL werden die Begriffe:
  - TABLE, ROW und COLUMN synonym für
  - Relation, Tupel und Attribute gebraucht.

- Mittels des CREATE TABLE-Befehls wird ein Relationenschema in der Datenbank definiert.

- Das Relationenschema muss genau spezifiziert werden, oder in anderen Worten, die zur Relation gehörenden Attribute sowie deren Domänen müssen angegeben werden.

- Zusätzlich sind noch eine Reihe weiterer Deklarationen möglich wie z. B. Wertebeschränkungen (CHECK-Klausel), Standardwerte oder Primär- und Fremdschlüsseldeklarationen.

Schemadefinition und -veränderung

```
create table Professoren
    (PersNr integer primary key,
     Name varchar(10) not null,
     Rang character(2));
```

```
drop table Professoren;
```

Anlegen & Löschen von Tabellen in MySQL

```
CREATE TABLE `test4`.`person`
(       `idPerson` INT NOT NULL,
        `Vorname` VARCHAR(45) NOT NULL,
        `Nachname` VARCHAR(45) NOT NULL,
        PRIMARY KEY (`idPerson`)
);

DROP TABLE ‘Person‘;
```

Anlegen & Löschen von Tabellen in MySQL

```
CREATE TABLE IF NOT EXISTS tasks (
    task_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    start_date DATE,
    due_date DATE,
    status TINYINT NOT NULL,
    priority TINYINT NOT NULL,
    description TEXT,
    created_at TIMESTAMP
)  ENGINE=INNODB;
```

Ändern von Tabellen

- Mit ALTER TABLE kann die Struktur einer Tabelle geändert werden.
- Es können somit die mit CREATE TABLE erzeugten Attribute und Beschränkungen geändert, neue hinzugefügt oder vorhandene gelöscht werden.
- Der Befehl hat folgende Syntax:

  ALTER TABLE <Tabellenname> <Änderung>;

Alter Table <Änderung>

- ADD [COLUMN] <Attributdefintion>
  - Attribut hinzufügen (Attributdefiniton wie bei CREATE)
- ALTER [COLUMN] <Attributname> SET DEFAULT <Standardwert>
  - neuer Standardwert festlegen
- ALTER [COLUMN] <Attributname> DROP DEFAULT
  - aktuellen Standardwert löschen
- DROP [COLUMN] <Attributname> {RESTRICT | CASCADE}
  - löschen eines Attributes
- ADD <Tabellenbeschränkung>
  - neue Tabellenbeschränkung hinzufügen (Tabellenbeschränkung wie bei CREATE)
- DROP CONSTRAINT <Tabellenbeschränkung>
  - löschen einer Tabellenbeschränkung

Ändern von Tabellen in MySQL

```sql
CREATE TABLE vehicles (
    vehicleId INT,
    year INT NOT NULL,
    make VARCHAR(100) NOT NULL,
    PRIMARY KEY(vehicleId)
);
```

```sql
ALTER TABLE vehicles
ADD model VARCHAR(100) NOT NULL;
```

Ändern von Tabellen in MySQL

- Add Column

```
ALTER TABLE vehicles
ADD model VARCHAR(100) NOT NULL;
```

- Modify Column

```
ALTER TABLE vehicles
MODIFY note VARCHAR(100) NOT NULL;
```

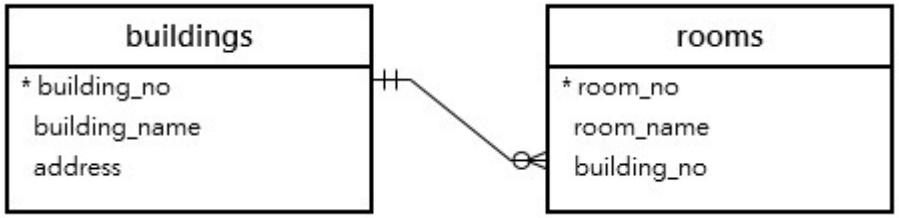
- Change Column Name

```
ALTER TABLE vehicles
CHANGE COLUMN note vehicleCondition VARCHAR(100) NOT NULL;
```

RESTRICT und CASCADE

- Das Schlüsselwort RESTRICT bewirkt, dass nur Attribute gelöscht werden können, die nicht mit anderen Tabellen verbunden sind (Fremdschüssel).
- Alternativ kann das Schlüsselwort CASCADE verwendet werden. Dabei wird nicht nur die gewünschte Spalte, sondern auch die verbundene Spalte in der anderen Tabelle gelöscht.

MySQL ON DELETE CASCADE



- When you delete a row from the buildings table, you also want to delete all rows in the rooms table that references to the row in the buildings table.

MySQL ON DELETE CASCADE

```
CREATE TABLE buildings (
    building_no INT PRIMARY KEY AUTO_INCREMENT,
    building_name VARCHAR(255) NOT NULL,
    address VARCHAR(255) NOT NULL
);
```

```
CREATE TABLE rooms (
    room_no INT PRIMARY KEY AUTO_INCREMENT,
    room_name VARCHAR(255) NOT NULL,
    building_no INT NOT NULL,
    FOREIGN KEY (building_no)
        REFERENCES buildings (building_no)
        ON DELETE CASCADE
);
```

Find Tables that use CASCADE Delete rule

```
USE information_schema;


SELECT
    table_name
FROM
    referential_constraints
WHERE
    constraint_schema = 'classicmodels'
        AND referenced_table_name = 'buildings'
        AND delete_rule = 'CASCADE'
```

Einfügen von Tupeln/Zeilen

› Um eine neue Zeile einzufügen (englisch *insert*), verwende den Befehl INSERT.

```
INSERT INTO tasks(title,priority)
VALUES('Learn MySQL INSERT Statement',1);
```

```
INSERT INTO tasks(title,priority)
VALUES('Understanding DEFAULT keyword in INSERT statement',DEFAULT);
```

Einfügen von Tupeln/Zeilen

- Datum einfügen

```
INSERT INTO tasks(title, start_date, due_date)
VALUES('Insert date into table','2018-01-09','2018-09-15');
```

```
INSERT INTO tasks(title,start_date,due_date)
VALUES('Use current date for the task',CURRENT_DATE(),CURRENT_DATE())
```

- Mehr Zeilen einfügen

```
INSERT INTO tasks(title, priority)
VALUES
    ('My first task', 1),
    ('It is the second task',2),
    ('This is the third task of the week',3);
```

Insert into Select Statement

- Es kommt auch häufig vor, dass die Werte die eingefügt werden müssen, als Resultatrelation einer Anfrage vorhanden sind.

```
INSERT INTO table_name(column_list)
SELECT
   select_list
FROM
   another_table
WHERE
   condition;
```

```
INSERT INTO stats(totalProduct, totalCustomer, totalOrder)
VALUES(
    (SELECT COUNT(*) FROM products),
    (SELECT COUNT(*) FROM customers),
    (SELECT COUNT(*) FROM orders)
);
```

Fehler ignorieren

- Alle Daten, die möglich sind, werden insertiert!
- Es erfolgt kein Abbruch des Statements!

```
INSERT IGNORE INTO table(column_list)
VALUES( value_list),
      ( value_list),
      ...
```

Löschen von Tupeln

- Das Kommando „Löschen“ (DELETE FROM) entfernt ein oder mehrere Tupel aus einer Relation.
- Die WHERE-Klausel spezifiziert, welche Tupel betroffen sind. Bei einer fehlenden WHERE-Klausel werden alle Tupel aus der angegebenen Relation gelöscht.

```
DELETE FROM <Relationenname>

WHERE <Bedingung>;
```

```
TRUNCATE TABLE <Relationenname>;
```

Löschen von Tupeln in MySQL

```
DELETE FROM employees
WHERE
    officeCode = 4;
```

```
DELETE FROM employees;
```

```
DELETE FROM table_name
ORDER BY c1, c2, ...
LIMIT row_count;
```

MySQL Delete Join Beispiel

```sql
DROP TABLE IF EXISTS t1, t2;

CREATE TABLE t1 (
    id INT PRIMARY KEY AUTO_INCREMENT
);

CREATE TABLE t2 (
    id VARCHAR(20) PRIMARY KEY,
    ref INT NOT NULL
);

INSERT INTO t1 VALUES (1),(2),(3);

INSERT INTO t2(id,ref) VALUES('A',1),('B',2),('C',3);
```

MySQL Delete Join Beispiel

id
1
2
3

id	Ref
A	1
B	2
C	3

```
DELETE
      t1 , t2
FROM
      t1
      INNER JOIN t2
            ON t2.ref = t1.id
WHERE
      t1.id = 1;
```

Ändern von Tupeln

- Das Kommando „Ändern“ (UPDATE) wird gebraucht, um Werte von Attributen von einem oder mehreren Tupeln zu ändern.
- Die WHERE-Klausel spezifiziert, welche Tupel von der Änderung betroffen sind.
- Die SET-Klausel spezifiziert die Attribute, die geändert werden sollen und bestimmt deren neue Werte.

```
UPDATE [LOW_PRIORITY] [IGNORE] table_name
SET
    column_name1 = expr1,
    column_name2 = expr2,
    ...
[WHERE
    condition];
```

Ändern von Tupeln - MySQL

```
UPDATE employees
SET
    lastname = 'Hill',
    email = 'mary.hill@classicmodelcars.com'
WHERE
    employeeNumber = 1056;
```

```
UPDATE employees
SET email = REPLACE(email,'@classicmodelcars.com','@mysqltutorial.org')
WHERE
   jobTitle = 'Sales Rep' AND
   officeCode = 6;
```

Update & Select (Fortgeschritten)

```
UPDATE customers
SET
    salesRepEmployeeNumber = (SELECT
            employeeNumber
        FROM
            employees
        WHERE
            jobtitle = 'Sales Rep'
        ORDER BY RAND()
        LIMIT 1)
WHERE
    salesRepEmployeeNumber IS NULL;
```

Vorgabewerte

- Einer Spalte kann ein Vorgabewert (englisch *default value*) zugewiesen werden.
- Wenn eine neue Zeile erzeugt wird und für einige Spalten keine Werte angegeben sind, dann werden die Spalten mit ihren entsprechenden Vorgabewerten gefüllt.

```
CREATE TABLE produkte (
        produkt_nr integer,
        name text,
        preis numeric DEFAULT 9.99 );
```

- Wenn kein Vorgabewert ausdrücklich definiert wurde, wird der NULL-Wert als Vorgabe angenommen

SQL Autoincrement

```
CREATE TABLE Persons (
    ID Integer PRIMARY KEY AUTOINCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);

AUTOINCREMENT(10,5)
```

SQL Autoincrement - Beispiel

```
CREATE TABLE animals (
     id MEDIUMINT NOT NULL AUTO_INCREMENT,
     name CHAR(30) NOT NULL,
     PRIMARY KEY (id)
);

INSERT INTO animals (name) VALUES
    ('dog'),('cat'),('penguin'),
    ('lax'),('whale'),('ostrich');

SELECT * FROM animals;
```

```
+----+---------+
| id | name    |
+----+---------+
|  1 | dog     |
|  2 | cat     |
|  3 | penguin |
|  4 | lax     |
|  5 | whale   |
|  6 | ostrich |
+----+---------+
```

SQL Autoincrement - Beispiel

```
INSERT INTO animals (id,name) VALUES(0,'groundhog');
INSERT INTO animals (id,name) VALUES(NULL,'squirrel');
INSERT INTO animals (id,name) VALUES(100,'rabbit');
INSERT INTO animals (id,name) VALUES(NULL,'mouse');
SELECT * FROM animals;
+-----+-----------+
| id  | name      |
+-----+-----------+
|   1 | dog       |
|   2 | cat       |
|   3 | penguin   |
|   4 | lax       |
|   5 | whale     |
|   6 | ostrich   |
|   7 | groundhog |
|   8 | squirrel  |
| 100 | rabbit    |
| 101 | mouse     |
+-----+-----------+
```

Constraints

- In SQL können benutzerdefinierte Datentypen *als Einschränkung der vorgegebenen Datentypen definiert werden*
  - *CHECK*
  - *NOT-NULL*
  - *PRIMARY KEY*
  - *UNIQUE*
  - *FOREIGN KEY/REFERENCES*

Constraints - CHECK

- Ein Check-Constraint ist der allgemeinste Constraint-Typ. Er erlaubt es Ihnen, einen beliebigen Ausdruck anzugeben, den jeder Wert einer Spalte erfüllen muss.
- Um zum Beispiel positive Produktpreise zu erzwingen, könnten Sie Folgendes tun:

```
CREATE TABLE produkte
      ( produkt_nr integer,
         name text,
         preis numeric CHECK (preis > 0) );
```

```
CREATE TABLE produkte
      (   produkt_nr integer,
         name text,
         preis numeric CONSTRAINT positiver_preis CHECK (preis > 0) );
```

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
```

Constraints - CHECK

- Ein Check-Constraint kann sich auch auf mehrere Spalten beziehen.

```
CREATE TABLE produkte
        (       produkt_nr integer,
                name text,
                preis numeric,
                CHECK (preis > 0),
                rabattpreis numeric,
                CHECK (rabattpreis > 0),
                CHECK (preis > rabattpreis) );
```

- Man sollte beachten, dass der Check-Constraint passiert wird, wenn das Ergebnis des Check-Ausdrucks logisch wahr oder der NULL-Wert ist.

Constraints - CHECK

```
CREATE TABLE t1
(
  CHECK (c1 <> c2),
  c1 INT CHECK (c1 > 10),
  c2 INT CONSTRAINT c2_positive CHECK (c2 > 0),
  c3 INT CHECK (c3 < 100),
  CONSTRAINT c1_nonzero CHECK (c1 <> 0),
  CHECK (c1 > c3)
);
```

Constraints – NOT NULL

- Ein NOT-NULL-Constraint gibt ganz einfach an, dass eine Spalte den NULL-Wert nicht aufnehmen darf.

In den meisten Datenbankentwürfen sollte die Mehrzahl der Spalten als NOT NULL markiert sein.

Constraints – NOT NULL

```
CREATE TABLE produkte
    (   produkt_nr integer NOT NULL,
        name text NOT NULL,
        preis numeric );
```

› Natürlich kann eine Spalte mehrere Constraints haben.

```
CREATE TABLE produkte
(       produkt_nr integer NOT NULL,
        name text NOT NULL,
        preis numeric NOT NULL CHECK (preis > 0) );
```

Constraint - UNIQUE

- Unique Constraints stellen sicher, dass die Daten einer Spalte oder einer Gruppe von Spalten von allen anderen Zeilen der Tabelle voneinander verschieden sind.

```
CREATE TABLE produkte (
        produkt_nr integer UNIQUE,
        name text,
        preis numeric );
```

Constraint – PRIMARY KEY

- Technisch gesehen ist ein Primärschlüssel-Constraint (englisch *primary key*) einfach eine Kombination aus Unique Constraint und NOT-NULL-Constraint.

```
CREATE TABLE produkte (
        produkt_nr integer PRIMARY KEY,
        name text,
        preis numeric );
```

- Primärschlüssel können auch über mehrere Spalten gehen; die Syntax ist ähnlich der des Unique Constraints:

```
CREATE TABLE beispiel (
        a integer,
        b integer,
        c integer, PRIMARY KEY (a, c) );
```

Constraint – PRIMARY KEY

- Ein Primärschlüssel zeigt an, dass eine Spalte oder eine Gruppe von Spalten als eindeutige Identifikation einer Zeile in der Tabelle verwendet werden kann.
- Das ist eine direkte Folge aus der Definition des Primärschlüssels.

Beachte, dass ein Unique Constraint keine eindeutige Identifikation erlaubt, da er NULL-Werte nicht ausschließt.

Constraint - REFERENCES

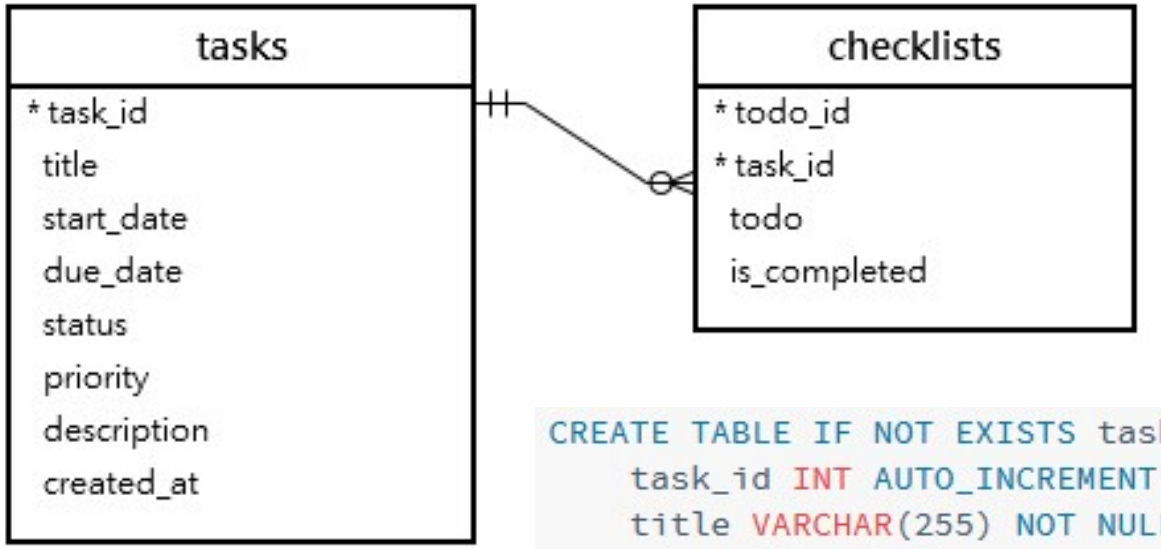
- Ein Fremdschlüssel-Constraint (englisch *foreign key*) gibt an, dass die Werte in einer Spalte (oder einer Gruppe von Spalten) mit den Werten in irgendeiner Zeile in einer anderen Tabelle übereinstimmen müssen.
- Sicherung der *referentiellen Integrität* zwischen zwei zusammenhängenden Tabellen.

Fremdschlüssel-Constraint

```
[CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (col_name, ...)
    REFERENCES tbl_name (col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
```

Fremdschlüssel-Constraint




```
CREATE TABLE IF NOT EXISTS tasks (
    task_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    start_date DATE,
    due_date DATE,
    status TINYINT NOT NULL,
    priority TINYINT NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)  ENGINE=INNODB;
```

Fremdschlüssel-Constraint

```
CREATE TABLE IF NOT EXISTS tasks (
    task_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    start_date DATE,
    due_date DATE,
    status TINYINT NOT NULL,
    priority TINYINT NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)  ENGINE=INNODB;
```

```
CREATE TABLE IF NOT EXISTS checklists (
    todo_id INT AUTO_INCREMENT,
    task_id INT,
    todo VARCHAR(255) NOT NULL,
    is_completed BOOLEAN NOT NULL DEFAULT FALSE,
    PRIMARY KEY (todo_id, task_id),
    FOREIGN KEY (task_id)
        REFERENCES tasks (task_id)
        ON UPDATE RESTRICT ON DELETE CASCADE
);
```

Constraint - REFERENCES

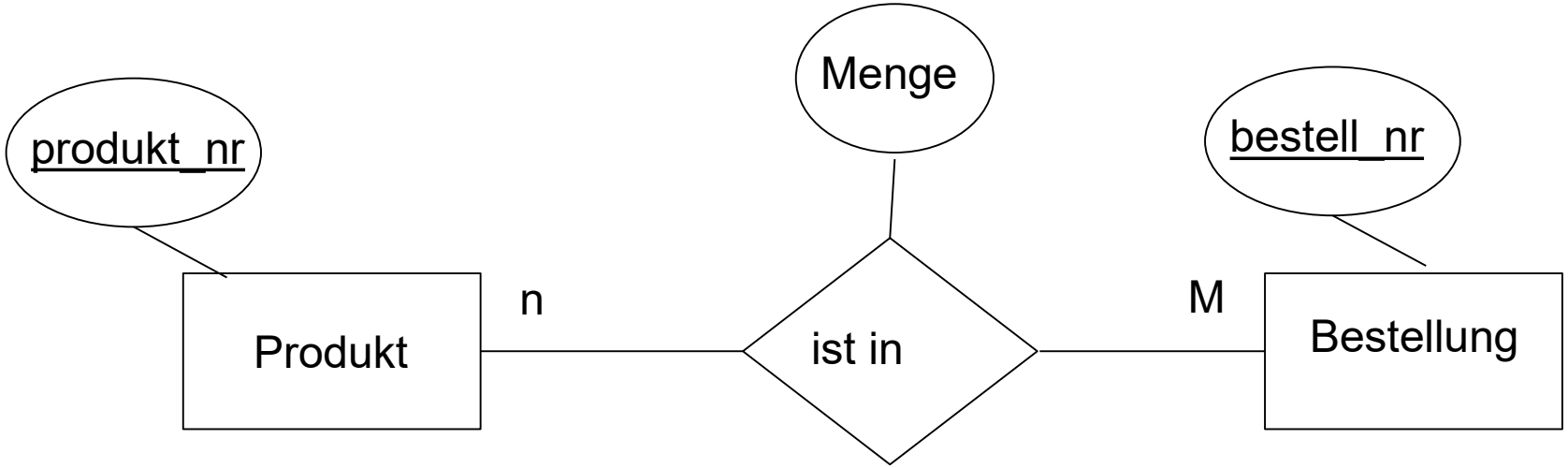
```
CREATE TABLE produkte
        (       produkt_nr integer PRIMARY KEY,
                name text,
                preis numeric );
```

- Nehmen wir an, dass Sie eine Tabelle haben, die Bestellungen dieser Produkte speichert.
- Wir wollen sichergehen, dass die Bestellungstabelle nur Bestellungen von tatsächlich existierenden Produkten aufnimmt.

Umsetzung einer m:n Beziehung mittels Constraints

- Eine Tabelle kann mehrere Fremdschlüssel-Constraints enthalten.
- Beispiel:
  - Tabellen mit Produkten und mit Bestellungen
  - Eine Bestellung kann mehrere Produkte aufnehmen.

Umsetzung einer m:n Beziehung mittels Constraints



Constraint - REFERENCES

- Also definieren wir einen Fremdschlüssel-Constraint in der Bestellungstabelle, der sich auf die Produkttabelle bezieht:

```
CREATE TABLE Produkt_Bestellung (
    bestell_nr int,
    produkt_nr int,
    menge int,
    PRIMARY KEY(bestell_nr,produkt_nr),
    FOREIGN KEY (bestell_nr) REFERENCES Bestellung(bestell_nr),
    FOREIGN KEY (produkt_nr) REFERENCES Produkt(produkt_nr)
);
```

- Jetzt ist es unmöglich, Bestellungen mit produkt_nr-Einträgen zu erzeugen, die nirgendwo in der Produkttabelle auftauchen.

Umsetzung einer m:n Beziehung mittels Constraints

- Wir haben jetzt erreicht, dass die Fremdschlüssel das Erzeugen von Bestellungen, die zu keinen Produkten passen, verhindern.
- Was passiert, wenn ein Produkt gelöscht wird, während noch eine Bestellung darauf verweist?
- In SQL können Sie auch das angeben. Intuitiv haben wir einige Möglichkeiten:
  - Das Löschen des Produktes verhindern
  - Die Bestellungen ebenso löschen
  - Noch etwas?

Umsetzung einer m:n Beziehung mittels Constraints

- Folgende Einschränkung definieren:
  - Wenn jemand ein Produkt entfernen möchte, das noch in einer Bestellung verwendet wird (über Produkt_Bestellung), wird das verboten.
  - Wenn jemand eine Bestellung entfernt, dann werden die bestellte Produkte (in Produkt_Bestellung) auch gelöscht.

On Delete/On Update Cascade

- ON DELETE CASCADE
  - It specifies that the child data is deleted when the parent data is deleted.

```
ALTER TABLE child_table
ADD CONSTRAINT fk_name
    FOREIGN KEY (child_col1, child_col2, ... child_col_n)
    REFERENCES parent_table (parent_col1, parent_col2, ... parent_col_n)
    ON DELETE CASCADE;
```

On Delete/ On Update Restrict

- RESTRICT
  - Rejects the delete or update operation for the parent table. (Wenn ein Primärschlüssel in der Primärtabelle geändert bzw. gelöscht werden soll und abhängige Sätze in der Detailtabelle existieren, dann wird die Änderung/Löschung verweigert.)

Umsetzung einer m:n Beziehung mittels Constraints

```
CREATE TABLE Produkt_Bestellung (
    bestell_nr int,
    produkt_nr int,
    menge int,
    PRIMARY KEY(bestell_nr,produkt_nr),
    FOREIGN KEY (bestell_nr) REFERENCES Bestellung(bestell_nr)
            ON DELETE CASCADE,
    FOREIGN KEY (produkt_nr) REFERENCES Produkt(produkt_nr)
            ON DELETE RESTRICT
);
```

- RESTRICT steht für „einschränken“, also faktisch verbieten;
- CASCADE bedeutet, dass die jeweilige Aktion auch in den anderen Tabellen durchgeführt wird.

Aufgabe

- Ergänze das Rechnungswesenbeispiel um Constraints:
  - Ein Artikel muss einen Lieferanten haben.
  - Ein Artikel muss zu einer Kategorie gehören.
  - Lege sinnvoll fest, welche Attribute nicht leer sein sollen. (Liefereinheit soll mit 0 initialisiert werden)
  - Eine Bestellung muss einem Kunden, einem Bearbeiter und einer Versandfirma zugeordnet sein.
  - Einzelpreis eines Artikels soll größer 0 sein.
  - Rabatt soll zwischen 0 und 100 (Prozent liegen)
  - Versandfirmennamen sollen nur einmal existieren.

Aufgabe

- Zusatz

  m:n Beziehungen realisieren.

  Wenn ein Artikel gelöscht werden soll, zu dem eine Bestellung existiert, soll das verhindert werden.

  Wenn ein Artikel gelöscht werden soll, zu dem eine Bestellung existiert, soll das verhindert werden.

Auf los geht‘s los ;-)

Quellen

- https://www.mysqltutorial.org/mysql-sequence/
- https://www.techonthenet.com/sql_server/foreign_keys/foreign_delete.php