SQL

- **SQL = Structured Query Language**
  - Standard-Anfrage-Sprache für relationale Datenbanken
  - Anfragen und Daten werden als Text (Text-String) übergeben
    - Beispiel:

      SELECT Nachname, Vorname **FROM** Student **WHERE** MatrNr = 123456 **;**

  - Anfragen können ...
    - Daten abfragen
      - Datensätze selektieren, Attribute auswählen, Daten verknüpfen
    - Daten ändern
      - einfügen, löschen, ändern
    - Datenstrukturen ändern
      - Tabellenstruktur, Attributtypen, Restriktionen

SQL

- **Es gibt viele relationale DBMS** (Software-Lösungen)
  - *Open Source*, z.B.
    - **MySQL** (siehe https://dev.mysql.com/doc/refman/8.0/en/)
    - **MariaDB** (siehe https://mariadb.org/) ← Ein „Fork“ vom MySQL
    - **PostgreSQL** (siehe https://www.postgresql.org/docs/)
    - **SQLight** (siehe https://www.sqlite.org/docs.html)
      ← *keine separate DBMS-Instanz, sondern nur Client-Bibliothek*
  - *Kommerziell* (proprietär, meist Closed Source), z.B.
    - **Oracle RDBMS**
    - **DB2** (IBM)
    - **Microsoft SQL Server**
- **Wir betrachten hier MySQL als Beispiel**
  - Typisch: **LAMP-Server** = Linux + Apache + MySQL + PHP
  - Beliebte Lösung (kostengünstig, ressourcensparsam, relativ einfach)
    - Andere DBMS als MySQL können aber oft mehr

SQL

- **DBMS bieten ihren Dienst anderen Programmen**
  - Zugriff in Form von SQL-Queries über **Netzwerk-Schnittstelle**
    - Via TCP → zugreifbar über das Internet
    - Kann aber auch auf Server-interne Zugriffe beschränkt werden (Isolation)
  - Der Zugreifer muss sich beim DBMS **authentifizieren**
    - Bei MySQL: Benutzername + Passwort
    - Auch Programme (z.B. PHP-Scripte) müssen das tun
  - Den einzelnen Nutzern können unterschiedliche Rechte gewährt werden (**Autorisierung**)
    - Zugriff auf bestimmte Datenbanken eines DBMS
    - Zugriff auf bestimmte Tabellen in einer Datenbank
    - Zugriff auf bestimmte Attribute einer Tabelle
    - Zugriff auf bestimmte Datensätze

SQL

- **Zum Zugriff gibt es auf Client-Seite Hilfsmittel**
  - SQL-Client-Tools (Grafisch / Webfrontend)
    - z.B. Web-Admin-Oberfläche PHPmyAdmin
  - SQL-Client-Bibliotheken (Connector)
    - z.B. für Zugriffe aus PHP heraus: http://php.net/manual/de/set.mysqlinfo.php
  - SQL-Client-Tools (Kommandozeile)
    - https://dev.mysql.com/doc/refman/8.0/en/programs-client.html
    - z.B. das **Client-Kommandozeilen-Programm** „**mysql**“
      - damit können SQL-Queries von Hand oder aus Dateien eingegeben werden

SQL

- **Plattform für die Übungen**
  - Übungsserver sind LAMP-Server (scilab-*nnnn*.informatik.uni-kl.de)
    - Jede Übungsgruppe erhält einen eigenen Server
  - MySQL ist nur Server-intern zugreifbar
    - Der Client muss also auf dem Server betrieben werden
  - Zugriff auf den Server per SSH
    - Zugangsdaten für SSH und Datenbank werden ausgegeben
    - richten Sie sich aber gleich einen Public-Key-Authentifizierung ein
  - Wir gehen im Folgenden von einer bestehenden SSH-Sitzung auf den LAMP-Server aus

SQL / MySQL

- **Kommandozeilen-Tool „mysql“ - erste Schritte**

```
[~] mysql
ERROR 1045 (28000): Access denied for user 'lamp'@'localhost'
(using password: NO)
```

  - Nach kurzem Lesen von „`man mysql`“ …

```
[~] mysql -p
Enter password: _
```

  - Der Login in die Datenbank funktioniert nun

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Type 'help;' or '\h' for help.
Type '\c' to clear the current input statement.

mysql> _
```

  - Man kann bei Bedarf Server (`-h`, default ist „localhost“) und Username (`-u`, default ist der Login-Acccount-Name) angeben

```
[~] mysql -h localhost -u lamp -p
Enter password: *******
```

SQL / MySQL

- **Konfigurationsdatei ~/.my.cnf**
  - Hier kann man z.B. das DB-Passwort ablegen

```
[client]
   user = lamp
   password = **********
```

  - Und vielleicht auch den Eingabe-Prompt erweitern

```
[mysql]
   prompt = (\\u@\\h) [\\d]>\\_
```

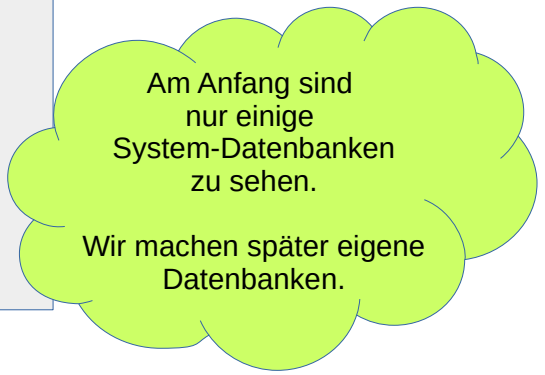
  - Danach funktioniert der DB-Login ohne weitere Angaben

```
[~] mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Type 'help;' or '\h' for help.
Type '\c' to clear the current input statement.

(lamp@localhost) [(none)]> _
```

- **SQL-Queries**
  - Wir können nun **Anfragen** (**Queries**) stellen
    - Anfragen können über mehrere Zeilen gehen
    - Groß-/Kleinschreibung bei **Schlüsselworten** beliebig (**Konvention**: groß), bei **Namen** (Tabellen, Attribute) aber genau wie bei ihrer Definition.
    - Anfragen an den Server enden mit einem **Semikolon**

```
[(none)]> SHOW DATABASES;
+----------------------+
| Database             |
+----------------------+
| information_schema   |
| mysql                |
| performance_schema   |
+----------------------+
```



    - Die Ausgabe zeigt eine **Tabelle**
      - Nur ein Attribut („Database“) als Spalte
      - Drei Datensätze als Zeilen
    - Wir sehen hier also **die Namen von** drei **Datenbanken**
      - Diese speziellen Tabellen enthalten Verwaltungsinformationen (*Metadaten*) des DBMS

- **SQL-Queries**
  - Client-bezogene **Anfragen** (**Queries**)
    - Anfragen, die der Client selbst beantwortet, brauchen kein Semikolon

```
[(none)]> HELP SHOW DATABASES
Name: 'SHOW DATABASES'
Description:
Syntax:
SHOW {DATABASES | SCHEMAS}
    [LIKE 'pattern' | WHERE expr]

SHOW DATABASES lists the databases on the
MySQL server host. [...]
```

    - Die Client-Abfrage „STATUS“ liefert Informationen zu Client und Verbindung

```
[(none)]> STATUS
mysql  Ver 14.14 Distrib 5.5.43

Current database:
Current user:          lamp@localhost
Server characterset:    utf8
Uptime:                    1 day 6 hours 12 min 53 sec
```

- **SQL-Queries: Tabellen auflisten**
  - Wir können nun eine der Datenbanken auswählen
    - Wir schauen uns als Beispiel mal die System-DB „mysql“ an

```
[(none)]> USE mysql
Database changed
[mysql]>
```

Die System-Datenbank “mysql“ ist nur ein erstes Beispiel

Analog zu „**cd** xyz“ bei Verzeichnissen in Dateisystemen

    - Ab jetzt ist in dieser Sitzung „mysql“ die **Default-DB**

```
[mysql]> SHOW TABLES;
+---------------------------+
| Tables_in_mysql           |
+---------------------------+
| columns_priv              |
| db                        |
| event                     |
| …                         |
| user                      |
+---------------------------+
```

    - Man kann die Datenbank auch direkt beim `mysql`-Aufruf übergeben
      - **mysql** *datenbankname*

- **SQL-Queries: Tabellenstruktur anzeigen**
  - Mit „DESCRIBE“ erhält man Informationen zur **Tabellenstruktur**

```
[mysql]> DESCRIBE user;
+------------------+------------------+------+-----+---------+-------+
| Field            | Type             | Null | Key | Default | Extra |
+------------------+------------------+------+-----+---------+-------+
| Host             | char(60)         | NO   | PRI |         |       |
| User             | char(16)         | NO   | PRI |         |       |
| Password         | char(41)         | NO   |     |         |       |
| Select_priv      | enum('N','Y')    | NO   |     | N       |       |
  …                  …                  …      …     …         …
+------------------+------------------+------+-----+---------+-------+
42 rows in set (0.01 sec)
```

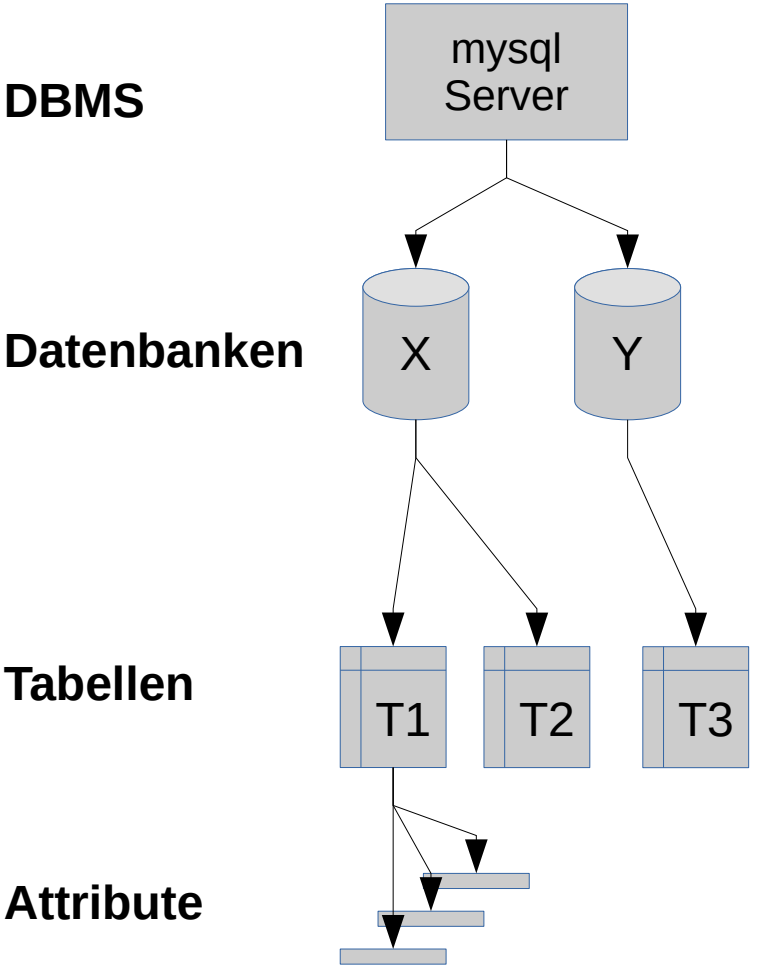
    - Es gibt also 42 Spalten in mysql.user, z.B.
      - Attribut „Host“ vom Typ „char(60)“
      - Attribut „User“ vom Typ „char(16)“
      - Attribut „Password“ vom Typ „char(41)“
      - ...

Wir schauen uns die Systemtabelle „*user*“ hier als mal Beispiel an, weil sie ja schon da ist.

SQL / MySQL

- **Überblick: Schrittweise Untersuchung des Schemas**
  - SHOW DATABASES;
    - zeigt die Namen aller **Datenbanken** an
  - USE database;
    - wechelt aktuelle **Datenbank** auf die angegebe
      - Man kann Tabellen qualifiziert angeben, z.B. „mysql.user“
  - SHOW TABLES;
    - zeigt die Namen der **Tabellen** in der aktuellen Datenbank
  - DESCRIBE tablename;
    - Zeigt die **Attribute** einer Tabelle an (z.B. „`DESCRIBE user;`“)

Hierarchie



SQL / MySQL: SELECT

- **SQL-Queries: Daten anfordern**
  - Mit „**SELECT**“ erhält man Zugriff auf die **Daten** der DB
    - z.B. die Daten der Spalten (→ Attribute) user und host aus der Tabelle user

```
> SELECT user,host FROM user;
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| debian-sys-maint | localhost |
| lamp             | localhost |
| mysql.session    | localhost |
| mysql.sys        | localhost |
| root             | localhost |
+------------------+-----------+
```

    - Mit WHERE kann man die Zeilen (→ Datensätze) selektieren

```
> SELECT user,host FROM user WHERE user = 'lamp';
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| lamp             | localhost |
+------------------+-----------+
```

SQL / MySQL: SELECT

- **SQL-Queries: Daten anfordern**
  - Spaltenwerte der Ausgabe können Duplikate enthalten
    - z.B. die Daten der Spalten user aus der Tabelle user

```
> SELECT host FROM user;
+-----------+
| host      |
+-----------+
| localhost |
| localhost |
| localhost |
| localhost |
| localhost |
+-----------+
```

    - Mit DISTINCT kann man die Duplikate entfernen

```
> SELECT DISTINCT host FROM user;
+-----------+
| host      |
+-----------+
| localhost |
+-----------+
```

SQL / MySQL: SELECT

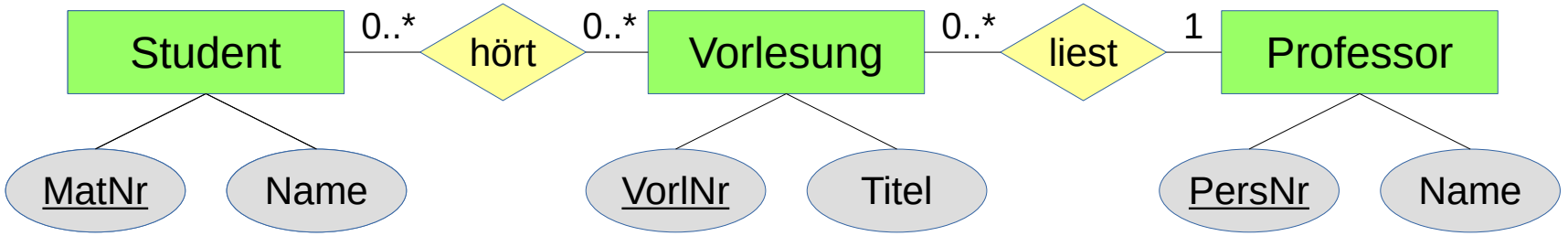
- **SQL-Queries: Daten anfordern**
  - Mit „**SELECT ***“ bekommt man alle Attribute

```
> SELECT * FROM user;
+-------------+-------+-------------------+-------------+-------------- …
| Host        | User  | Password          | Select_priv | Insert_priv  …
+-------------+-------+-------------------+-------------+-------------- …
| localhost   | root  | *E37FC...3DE8060 | Y           | Y             …
  …             …       …                   …             …
+-------------+-------+-------------------+-------------+-------------- …
```

    - Problem: Die Tabelle ist sehr breit (42 Attribute)
    - Einen Datensatz als Zeile (also horizontal) darzustellen ist hier nicht sinnvoll
  - Vertikale Darstellung: „\G“ statt Semikolon

```
> SELECT * FROM user \G
*************************** 1. row ***************************
                  Host: localhost
                  User: root
              Password: *E37FCBC917BA6E5F0EDBC5B6441E532293DE8060
           Select_priv: Y
                     …  …
```

- **Anwendungsbeispiel (SQL-Schema)**
  - ER-Schema (konzeptionelles Modell)



  - Beispiel-Daten

Student

MatNr	Name
26120	Fichte
25403	Jonas
27103	Fauler

hört

MatrNr	VorlNr
25403	5001
26120	5001
26120	5045

Vorlesung

VorlNr	Titel	PersNr
5001	ET	15
5022	IT	12
5045	DB	12

Professor

PersNr	Name
12	Wirth
15	Tesla
20	Urlauber

  - Quelle: Deutsche Wikipedia-Seite zu „**SQL**“
    - http://de.wikipedia.org/wiki/SQL

Zur Übung: Zeichnen Sie das Implementierungsmodell (ER-Diagramm ohne n:m)

Mehr dazu auf dem ersten Übungsblatt.

HOWTO: SQL-*Schema* und *Daten* zum Anwendungsbeispiel

Um mit dem o.g. Anwendungsbeispiel praktisch arbeiten zu können, stellen wir Ihnen das SQL-Schema und die Daten zur Verfügung.

Auf den **Übungsservern** ist das Schema mit den Daten bereits in der Datenbank `wikipedia_sql_example` installiert.

Falls Sie das Schema und die Daten auf **eigenen Systemen** installieren wollen, können Sie von folgender URL zwei SQL-Scripte herunterladen:

```
https://sci.cs.uni-kl.de/lv/w2t2/download/wikipedia_sql_example
```

Damit können Sie das Schema und die Daten anlegen. Rufen Sie dazu folgende Shell-Kommandos in dem Verzeichnis mit den Dateien auf.

```
mysql < create-schema.sql
mysql < create-data.sql
```

- Die Funktionsweise der SQL-Scripte werden wir später erklären.

SQL / MySQL: SELECT

Zur Übung:
Jeweils erst überlegen,
dann ausprobieren!

- **Anwendungsbeispiele**
  - SELECT * FROM Student;
  - SELECT VorlNr, Titel FROM Vorlesung;
  - SELECT DISTINCT MatrNr FROM hört;
  - SELECT VorlNr, Titel FROM Vorlesung WHERE Titel = 'ET';
  - SELECT VorlNr AS Vorlesungsnummer, Titel FROM Vorlesung;
    - Nur die beiden Spalten anzeigen, dabei die erste Spalte **umbenennen**
  - SELECT Name FROM Student WHERE Name LIKE 'F%';
    - Nur Namen die mit „F“ beginnen
  - SELECT Name FROM Student ORDER BY Name;
    - Alphabetisch sortieren (mit „ORDER BY Name DESC“ umgekehrt)
  - SELECT Name FROM Student LIMIT 5 OFFSET 10;
    - Höchstens 5 Ergebnisse ausgeben, überspringe die ersten 10 vorher
      - Anwendung z.B. Paginator: 3. Seite wenn pro Seite nur 5 angezeigt werden

SQL / MySQL: SELECT

- **Berechnungen und Funktionen in Queries**
  - Im SELECT-Statement können auch berechnete Ergebnisse ausgegeben werden

```
> SELECT 1+2*3;
+-------+
| 1+2*3 |
+-------+
|     7 |
+-------+
```

  - Hier können auch Funktionsergebnisse abgefragt werden
    - z.B. MIN(), MAX(), SUM(), AVG(), COUNT()

```
> SELECT MIN(PersNr), MAX(PersNr), SUM(PersNr), AVG(PersNr), COUNT(*)
  FROM   Vorlesung;
+-------------+-------------+-------------+-------------+---------------+
| MIN(PersNr) | MAX(PersNr) | SUM(PersNr) | AVG(PersNr) | COUNT(*)      |
+-------------+-------------+-------------+-------------+---------------+
|          12 |          15 |          39 |     13.0000 |             3 |
+-------------+-------------+-------------+-------------+---------------+
```

    - COUNT(Attributname) zählt nur Nicht-NULL-Werte, COUNT(*) zählt alle Zeilen
    - COUNT(DISTINCT Attributname) zählt unterschiedliche Nicht-NULL-Werte

SQL / MySQL: SELECT

- **Berechnungen und Funktionen in Queries**
  - Mit GROUP BY können Berechnungen auch für Gruppen von Datensätzen mit gleichen Eigenschaften ausgegeben werden
    - Bsp.: Wir wollen wissen, wie viele Studierende jeweils welche VL hören

```
> SELECT VorlNr, MatrNr FROM hört;
+--------+--------+
| VorlNr | MatrNr |
+--------+--------+
|   5001 |  25403 |
|   5001 |  26120 |
|   5045 |  26120 |
+--------+--------+
```

    - Idee: Zeilen mit gleicher VorlesungsNummer werden gruppiert

```
> SELECT VorlNr, COUNT(MatrNr) FROM hört GROUP BY VorlNr;
+--------+---------------+
| VorlNr | COUNT(MatrNr) |
+--------+---------------+
|   5001 |             2 |
|   5045 |             1 |
+--------+---------------+
```

SQL / MySQL: SELECT

- **Verknüpfung von Tabellen (Join)**
  - Wir machen ein Select über zwei Tabellen (Professor, Vorlesung)
    - Ziel: wir wollen den Dozenten-Namen zu jeder Vorlesung sehen

```
> SELECT *
  FROM   Professor , Vorlesung ;
+--------+----------+--------+-------+--------+
| PersNr | Name     | VorlNr | Titel | PersNr |
+--------+----------+--------+-------+--------+
|     12 | Wirth    |   5001 | ET    |     15 |
|     15 | Tesla    |   5001 | ET    |     15 |
|     20 | Urlauber |   5001 | ET    |     15 |
|     12 | Wirth    |   5022 | IT    |     12 |
|     15 | Tesla    |   5022 | IT    |     12 |
|     20 | Urlauber |   5022 | IT    |     12 |
|     12 | Wirth    |   5045 | DB    |     12 |
|     15 | Tesla    |   5045 | DB    |     12 |
|     20 | Urlauber |   5045 | DB    |     12 |
+--------+----------+--------+-------+--------+
```

    - Offensichtlich werden einfach alle (3*3 = 9) Kombinationen gebildet
    - Sinnvoll sind aber nur die, bei denen PersNr übereinstimmt

SQL / MySQL: SELECT

- **Verknüpfung von Tabellen** (mit Bedingungen)
  - Wir wollen ja nur bestimmte Datensätze → WHERE-Klausel
    - Idee: WHERE Professor.PersNr = Vorlesung.PersNr

```
> SELECT *
  FROM   Professor , Vorlesung
  WHERE  Professor.PersNr = Vorlesung.PersNr;
+--------+-------+--------+-------+--------+
| PersNr | Name  | VorlNr | Titel | PersNr |
+--------+-------+--------+-------+--------+
|     15 | Tesla |   5001 | ET    |     15 |
|     12 | Wirth |   5022 | IT    |     12 |
|     12 | Wirth |   5045 | DB    |     12 |
+--------+-------+--------+-------+--------+
```

    - Das sind die gewünschten Datensätze.
    - Die Spalte PersNr ist allerdings doppelt. (Frage: Warum?)

SQL / MySQL: SELECT

- **Verknüpfung von Tabellen**
  - Jetzt lassen wir noch die unwichtigen Spalten weg

```
> SELECT Vorlesung.Titel, Professor.Name
  FROM   Professor , Vorlesung
  WHERE  Professor.PersNr = Vorlesung.PersNr;
+-------+-------+
| Titel | Name  |
+-------+-------+
| ET    | Tesla |
| IT    | Wirth |
| DB    | Wirth |
+-------+-------+
```

  - Das Verknüpfungs-Attribut PersNr ist übrigens gar nicht mehr in der Ausgabe.
  - Verständnisfrage: Warum ist die Verknüpfung anhand dieses Elements trotzdem möglich?

SQL / MySQL: SELECT

- **Verknüpfung von Tabellen** (inner Join)
  - Diese Verknüpfung nennt man einen **inner Join** der Tabellen
    - Es wird das Kreuzprodukt über beide Tabellen gebildet.
    - Dann werden die Kombinationen, die die Join-Bedingung nicht erfüllen, ausgefiltert (siehe WHERE-Bedingung oben)
  - Das besondere am inner Join ist, dass Datensätze beider Tabellen die keinen passenden Join-Partner haben, nicht auftauchen
    - Beispiel: Dozent (20, „Urlauber“) hat keine Vorlesung
  - Es gibt dafür auch ein explizites Konstrukt:
    - FROM … **INNER JOIN** … **ON** …

```
> SELECT Vorlesung.Titel, Professor.Name
  FROM Professor INNER JOIN Vorlesung
       ON Professor.PersNr = Vorlesung.PersNr;
```

    - Das Ergebenis ist das selbe wie zuvor.

SQL / MySQL: SELECT

- **Verknüpfung von Tabellen** (USING)
  - Werden nur gleichnamige Attribute verglichen,
    so kann im Join anstatt ON auch **USING** verwendet werden
    - Anstatt ON …

```
> SELECT *
  FROM Professor INNER JOIN Vorlesung
       ON Professor.PersNr = Vorlesung.PersNr;
```

    - ... kann man auch USING verwenden:

```
> SELECT *
  FROM Professor INNER JOIN Vorlesung
       USING (PersNr);
```

    - Im letzteren Fall ist zudem das in USING angegebene Attribut nicht mehr doppelt vorhanden:

```
> SELECT * FROM Professor INNER JOIN Vorlesung USING (PersNr);
+--------+-------+--------+-------+
| PersNr | Name  | VorlNr | Titel |
+--------+-------+--------+-------+
|     15 | Tesla |   5001 | ET    |
     ...   .....     .....  ...
```

SQL / MySQL: SELECT

- **Verknüpfung von Tabellen** (outer Join)
  - Es gibt auch einen **outer Join** zwischen Tabellen
  - Besonders nützlich ist der **left outer Join**
    - Es wird analog zum inner Join vorgegangen
    - Datensätze der linken Tabelle, die keinen Join-Partner haben, werden mit NULL-Werten verknüpft in die Ergebnismenge aufgenommen
      - Bsp.: Professor (20, Urlauber) wird mit NULL-Vorlesungsattributen gelistet

```
> SELECT Vorlesung.Titel, Professor.Name
  FROM Professor LEFT OUTER JOIN Vorlesung
       USING (PersNr);
+-------+----------+
| Titel | Name     |
+-------+----------+
| IT    | Wirth    |
| DB    | Wirth    |
| ET    | Tesla    |
| NULL  | Urlauber |
+-------+----------+
```

    - Beim outer Join (ohne „*left*“) geschieht dies auf beiden Seiten so.

SQL / MySQL: SELECT

- **Verknüpfung von Tabellen** (Subqueries)
  - Man kann in Queries auch auf das Ergebnis von eingeschachtelten Anfragen Bezug nehmen (**Subqueries**)

```
> SELECT Titel,
         (SELECT Name FROM Professor
                      WHERE Vorlesung.PersNr = Professor.PersNr
         ) AS Name
  FROM Vorlesung;
```

    - Für jeden Datensatz der äußeren Anfrage (über Vorlesung) wird die innere Anfrage (über Professor) einmal ausgeführt.

```
+-------+-------+
| Titel | Name  |
+-------+-------+
| ET    | Tesla |
| IT    | Wirth |
| DB    | Wirth |
+-------+-------+
```

    - Da das ineffizient ist vermeidet man das, wenn auch ein Join möglich ist.
    - Frage: Ist das ein Inner- oder ein Outer Join?

SQL / MySQL: SELECT

- **Verknüpfung von Tabellen** (Subqueries)
  - **Subqueries** können auch als Bedingungen (in WHERE-Klauseln) wie Werte benutzt werden.

```
> SELECT Titel
  FROM Vorlesung
  WHERE (SELECT Name
         FROM   Professor
         WHERE  Vorlesung.PersNr = Professor.PersNr
        ) = 'Wirth';

+-------+
| Titel |
+-------+
| IT    |
| DB    |
+-------+
```

    - Frage: Was bedeutet diese Anfrage?
  - Sie dürfen dort meist nur einen Wert (Datensatz) liefern
    - Frage: Kann das hier schief gehen?

SQL / MySQL: SELECT

- **Verknüpfung von Tabellen** (Subqueries)
  - **Subqueries** können auch als Datenquellen (in FROM-Klauseln) benutzt werden.
    - Primitives Beispiel:

```
> SELECT *
  FROM  (SELECT Titel FROM Vorlesung);
+-------+
| Titel |
+-------+
| ET    |
| IT    |
| DB    |
+-------+
```

  - Das kann bei komplexeren Anfragen zur klareren Strukturierung dienen.
    - Hier darf der Subquery natürlich mehrere Datensätze liefern.
    - Übung: Geben Sie ein Beispiel an, bei dem das sinnvoll genutzt wird.

SQL / MySQL: Schemaerzeugung

- **Erzeugung des Datenschemas**
  - Wir betrachten nun, wie die Datenstrukturen der Beispiel-Datenbank aus dem obigen Beispiel in SQL erzeugt werden.
    - Vorzugsweise bereitet man die Generierung als SQL-Datei vor und importiert diese dann per Eingabeumleitung auf Kommandozeilenebene in den mysql-Client.

```
[~] mysql < create-schema.sql
```

    - Damit Erzeugung der Datenbank zum Testen immer erneut erfolgen kann, *löschen* wir zuallererst möglicherweise noch existierende frühere Fassungen

```
> DROP   DATABASE IF EXISTS wikipedia_sql_example;
> CREATE DATABASE           wikipedia_sql_example;
> USE                       wikipedia_sql_example;
```

    - Nun existiert die leere Datenbank „*wikipedia_sql_example*“ und ist aktuelle Datenbank.

SQL / MySQL: Schemaerzeugung

- **Erzeugung des Datenschemas**
  - „**`CREATE TABLE ...`**“ erzeugt eine neue Tabelle
    - **Definition der Spalten** (Name, Typ, Eigenschaften)
    - Angabe von **Tabelleneigenschaften** (Primärschlüssel, …)
  - Beispiel

```
CREATE TABLE Student (
               MatrNr     INT(10)  NOT NULL,
               Name       CHAR(64) NOT NULL,

               PRIMARY KEY (MatrNr)
 );
```

Spaltendefinitionen

Tabelleneigenschaft

    - Die Spalte MatNr ist ein Integer mit maximal 10 Stellen
    - Die Spalte Name ist ein Character-String mit max. 64 Zeichen
    - Primärschlüssel ist die Spalte MatNr
  - Siehe https://dev.mysql.com/doc/refman/8.0/en/create-table.html

SQL / MySQL: Schemaerzeugung

- **Erzeugung des Datenschemas: Datentypen (Auszug)**
  - Ganze Zahlen: **INT, INTEGER**
    - INT[(length)] [UNSIGNED]
  - Fließkommazahlen: **FLOAT**
    - FLOAT [(length,decimals)] [UNSIGNED]
  - Strings mit fester / begrenzter Länge: **CHAR, VARCHAR**
    - CHAR[(length)] (Strings werden mit Leerzeichen aufgefüllt)
    - VARCHAR(length) (Strings werden mit exakter Länge gespeichert)
    - Optional Angabe von Encoding / Collation
      - ... [CHARACTER SET charset_name] [COLLATE collation_name]
  - Strings mit variabler Länge: **TEXT**
    - Optinal mit Encoding / Collation

SQL / MySQL: Schemaerzeugung

- **Erzeugung des Datenschemas: Datentypen (Auszug)**

  Viele weitere Datentypen, z.B. ...

  - Zeit/Datum: **DATE, TIME, DATETIME**
  - Aufzählungstypen: **ENUM**
    - ENUM(value1,value2,value3,...)
      - Beispiel: ENUM('yes', 'no', 'perhaps')
  - Binäre Objekte: **BLOB**
    - „**Binary Large Object**“, werden uninterpretiert gespeichert
  - Viele **Typ-Varianten**
    - z.B. zu INT, INTEGER: **TINYINT, SMALLINT, MEDIUMINT, BIGINT**
    - Haben meist unterschiedliche Wertebereiche

SQL / MySQL: Schemaerzeugung

- **Erzeugung des Datenschemas: Spalten-Eigenschaften**

  Jede Spalte kann weitere Eigenschaften haben, z.B. ...

  - Ist ***NULL*** als Wert erlaubt *(default: ja)*: **NULL**, **NOT NULL**
  - Einen Default-Wert angeben: **DEFAULT** value
    - Beispiel: comment TEXT DEFAULT 'no comment'
  - Werte der Spalte müssen sich unterscheiden: **UNIQUE**
    - NULL-Werte (wenn erlaubt) dürfen mehrfach vorkommen
  - Spalte ist Primärschlüssel: [**PRIMARY**] **KEY**
    - Impliziert NOT NULL und UNIQUE
      - Besser trotzdem explizit angeben!
  - Automatisch neuen Wert setzen: **AUTO_INCREMENT**
    - Wenn beim Einfügen kein Wert angegeben wird, wird ein noch nie genutzter Wert (der zudem größer ist als der maximal vorhandene) eingesetzt.
    - Verständnisfrage: Wozu braucht man das?

SQL / MySQL: Schemaerzeugung

- **Erzeugung des Datenschemas: Tabellen-Eigenschaften**

  Die Tabelle kann ebenfalls Eigenschaften haben, z.B. …

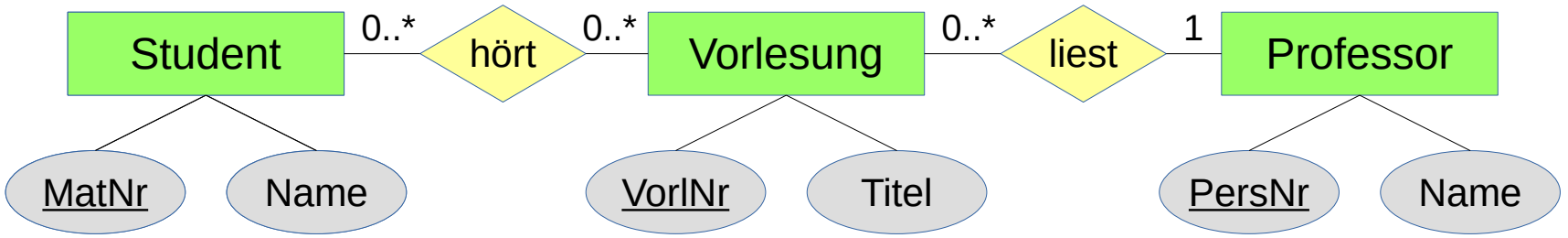
  - Mehreren Spalten müssen sich als Tupel unterscheiden: **UNIQUE**
    - UNIQUE [KEY](index_col_name, …)
    - Wenn mehrere Spalten nicht die selbe Wertkombination haben dürfen
  - Mehrere Spalten sind Primärschlüssel: **[PRIMARY] KEY**
    - PRIMARY KEY (index_col_name,...)
    - Beispiel: PRIMARY KEY (MatrNr, VorlNr)
  - Spalten sind Fremdschlüssel: **FOREIGN KEY**
    - FOREIGN KEY (index_col_name,...) *reference_definition*

SQL / MySQL: Schemaerzeugung

- **Erzeugung des Datenschemas: Tabellen-Eigenschaften**
  - **FOREIGN KEY** (index_col_name,...) reference_definition
    - **reference_definition**: REFERENCES tbl_name (index_col_name,...)
      [ON DELETE reference_option]
      [ON UPDATE reference_option]
    - **reference_option**: RESTRICT | CASCADE | SET NULL | ...
  - Zur Erinnerung: **Referentielle Integrität**
    - Referenzierte Objekte müssen existieren!
    - Was passiert, wenn der referenzierte Schlüssel **verändert** / **gelöscht** wird?
      - RESTRICT: Das ist nicht erlaubt, so lange es Referenzen gibt
      - CASCADE: Ändere den Fremdschlüssel ebenfalls ab
      - SET NULL: lösche den Fremdschlüssel (auf NULL setzen)
      - SET DEFAULT: Fremdschlüssel auf angegebenen Wert setzen
      - NO ACTION: Erst mal erlauben, am Ende der **Transaktion** (s.u.) prüfen
    - Siehe auch http://en.wikipedia.org/wiki/Foreign_key

SQL / MySQL: Schemaerzeugung

- **Anwendungsbeispiel (SQL-Schema)**
  - ER-Schema



  - Beispiel-Daten

Student

MatNr	Name
26120	Fichte
25403	Jonas
27103	Fauler

hört

MatNr	VorlNr
25403	5001
26120	5001
26120	5045

Vorlesung

VorlNr	Titel	PersNr
26120	ET	15
25403	IT	12
27103	DB	12

Professor

PersNr	Name
12	Wirth
15	Tesla
20	Urlauber

  - Quelle: Deutsche Wikipedia-Seite zu SQL
    - http://de.wikipedia.org/wiki/SQL

SQL / MySQL: Schemaerzeugung

- **Erzeugung des Datenschemas**
  - **Tabelle Student** anlegen

```
CREATE TABLE Student (
               MatrNr     INT(10) UNSIGNED
                                  UNIQUE
                                  NOT NULL
                                  AUTO_INCREMENT,
               Name       CHAR(64)   NOT NULL,

               PRIMARY KEY (MatrNr)
 );
```

- Die MatrNr ist ein vorzeichenloser Integer mit max 10 Dezimalstellen.
  - Da sie **Primärschlüssel** sein soll, ist sie
    - UNIQUE (die Werte dafür müssen verschieden sein, sofern sie nicht NULL sind)
    - NOT NULL (es müssen konkrete Werte benutzt werden, NULL ist verboten)
    - AUTO_INCREMENT (es werden beim Einfügen ggf. automatisch Werte vergeben)
- Der Name ist ein String mit max. 64 Zeichen
    - NOT NULL (es müssen konkrete Werte benutzt werden, NULL ist verboten)
- PRIMARY KEY (MatNr): MatNr ist Primärschlüssel der Tabelle

SQL / MySQL: Schemaerzeugung

- **Erzeugung des Datenschemas**
  - **Tabelle Professor** anlegen

```
CREATE TABLE Professor (
               PersNr    INT(10) UNSIGNED
                                 UNIQUE
                                 NOT NULL
                                 AUTO_INCREMENT,
               Name      CHAR(64)NOT NULL,

               PRIMARY KEY (PersNr)
 );
```

    - völlig analog zu oben

SQL / MySQL: Schemaerzeugung

- **Erzeugung des Datenschemas**
  - **Tabelle Vorlesung** anlegen

```
CREATE TABLE Vorlesung (
               VorlNr  INT(10)      UNSIGNED
                                    UNIQUE
                                    NOT NULL
                                    AUTO_INCREMENT,
               Titel   CHAR(64)     NOT NULL,
               PersNr  INT(10)      UNSIGNED
                                    NULL,

               PRIMARY KEY (VorlNr),
               FOREIGN KEY (PersNr) REFERENCES Professor(PersNr)
                       ON DELETE SET NULL
                       ON UPDATE CASCADE
);
```

  - Neu ist hier die **Fremdschlüssel-Definition**
    - PersNr ist **Fremdschlüssel** zum Attribut PersNr in der Tabelle Professor
    - Beim Löschen des referenzierten Professors wir der Verweis auf NULL gesetzt
    - Beim Ändern der Personalnummer des Professors wird die neue übernommen

SQL / MySQL: Schemaerzeugung

- **Erzeugung des Datenschemas**
  - **Beziehungs-Tabelle „hört“** anlegen

```
CREATE TABLE hört (
               MatrNr     INT(10) UNSIGNED,
               VorlNr     INT(10) UNSIGNED,

               PRIMARY KEY (MatrNr, VorlNr),
               FOREIGN KEY (MatrNr) REFERENCES Student(MatrNr)
                          ON DELETE CASCADE
                          ON UPDATE CASCADE,
               FOREIGN KEY (VorlNr) REFERENCES Vorlesung(VorlNr)
                          ON DELETE CASCADE
                          ON UPDATE CASCADE
);
```

  - Neu ist hier der **zweiteilige Primärschlüssel**
    - (MatNr, VorlNr) bilden **gemeinsam** den **Primärschlüssel**
      - Sie sind daher **gemeinsam UNIQUE**
    - MatNr und VorlNr sind einzeln Fremdschlüssel zu Student bzw. Vorlesung
    - Beim Löschen des referenzierten Professors oder der referenzierten Vorlesung wird der betroffen „hört“-Datensatz auch gelöscht.