

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
- Daten**strukturen** 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/>)
 - **SQLite** (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**
 - Jeweils **lesend** oder **schreibend**

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-Account-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 / MySQL

- **SQL-Queries**

- Wir können nun **Anfragen (Queries)** stellen

- Anfragen können über mehrere Zeilen gehen
- Groß-/Kleinschreibung bei **Schlüsselwörtern** 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 / MySQL

- **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
```

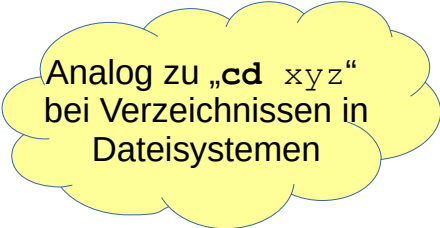
SQL / MySQL

- **SQL-Queries: Tabellen auflisten**

- Wir können nun eine der Datenbanken auswählen

- Wir schauen uns mal die System-DB „mysql“ an

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



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 / MySQL

- **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)
```

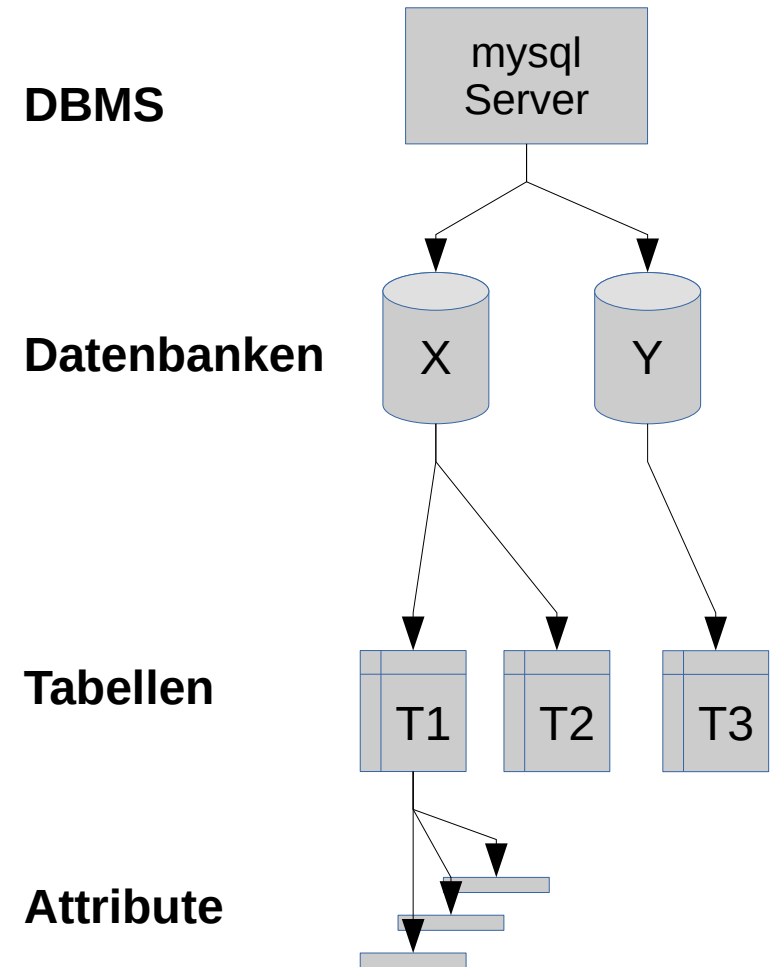
- Es gibt also 42 Spalten in mysql.user, z.B.
 - Attribut „Host“ vom Typ „char(60)“
 - Attribut „User“ vom Typ „char(16)“
 - Attribut „Password“ vom Typ „char(41)“
 - ...

SQL / MySQL

- Überblick: **Schrittweise Untersuchung** des Schemas

- **SHOW DATABASES;**
 - zeigt die Namen aller **Datenbanken** an
- **USE database;**
 - wechselt aktuelle **Datenbank** auf die angebebe
 - 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

- 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 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           | ...
| ...      | ...  | ...               | ...        | ...        | ...
+-----+-----+-----+-----+-----+
```

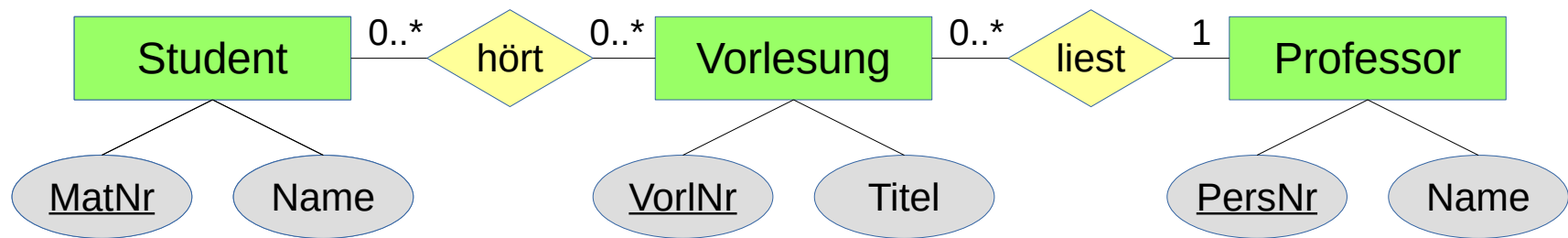
- Problem: Die Tabelle ist sehr breit (42 Attribute)
- Einen Datensatz als Zeile (also horizontal) darzustellen ist hier nicht sinnvoll

- 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		hört		Vorlesung			Professor	
MatNr	Name	MatNr	VorlNr	VorlNr	Titel	PersNr	PersNr	Name
26120	Fichte	25403	5001	5001	ET	15	12	Wirth
25403	Jonas	26120	5001	5022	IT	12	15	Tesla
27103	Fauler	26120	5045	5045	DB	12	20	Urlauber

- Quelle: Deutsche Wikipedia-Seite zu „SQL“

- <http://de.wikipedia.org/wiki/SQL>

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-Skripte 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-Skripte werden wir später erklären.

SQL / MySQL: SELECT

• 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 2;
 - Höchstens 2 Ergebnisse ausgeben

Zur Übung:
Jeweils erst überlegen,
dann ausprobieren!

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(MatrnNr) FROM hört GROUP BY VorlNr;
+-----+-----+
| VorlNr | COUNT(MatrnNr) |
+-----+-----+
| 5001   | 2               |
| 5045   | 1               |
+-----+-----+
```