```
HOWTO zur MySQL-Nutzung (Entwickler oder Anwender)

(C) 2006-2018 T.Birnthaler/H.Gottschalk <howtos(at)ostc.de>
              OSTC Open Source Training and Consulting GmbH
              http://www.ostc.de

$Id: mysql-user-HOWTO.txt,v 1.297 2020/02/24 06:36:18 tsbirn Exp $

Dieses Dokument beschreibt den MySQL-Einsatz auf Entwickler/Anwenderseite.

HINWEIS: MySQL-SpezialitÃ¤ten sind durch "MY!" oder "MY!N.M" gekennzeichnet
         (falls sie ab MySQL Version N.M (nicht mehr) verfÃ¼gbar sind).

HINWEIS: Der Begriff "Datenbank" wird hÃ¤ufig "schwammig" verwendet. MySQL ist
         ein "Datenbank-Managementsystem" (DBMS), das die Verwaltung vieler
         "Datenbanken" gleichzeitig erlaubt. Jede Datenbank besteht aus
         Tabellen und weiteren Datenbank-Objekten, die zur Abbildung eines
         konkreten Sachverhalts verwendet werden. Statt der (langen) Begriffe
         "Datenbank-Managementsystem" oder (abgekÃ¼rzt) "Datenbank-System" wird
         gerne der kurze Begriff "Datenbank" verwendet. Ein besserer Begriff
         fÃ¼r eine eigentliche "Datenbank" ist daher "Schema", so besteht keine
         Verwechslungsgefahr, was gemeint ist. Das DBMS Oracle z.B. vermeidet
         den Begriff "Datenbank" und verwendet "Schema". FÃ¼r "Datenbank" wird
         in diesem Skript "DB" oder <Db> als AbkÃ¼rzung verwendet.

_____________________________________________________________________________

INHALTSVERZEICHNIS
==================

1) MySQL-Clients
1a) Zusammenspiel der MySQL-Komponenten
1b) MySQL Web-Client "phpMyAdmin" einrichten und aufrufen
1c) MySQL-Clients: Konfigurations-Dateien
1d) MySQL-Clients: Allgemeine Optionen
1e) MySQL-Clients: Verbindung zum Server (SSL)
1f) Client "mysql" starten (Verbindung/Session)
1g) Client "mysql": Befehlsein- und ausgabe (Prompt, Strg-C, TEE)
1h) Client "mysql" effizient bedienen (USE, History, Strg-L, TAB, Hilfe, SOURCE)
1i) Client "mysql" verlassen
1j) Client "mysql" auf Kommandozeile nutzen (Batch-Modus)
1k) Client "mysql": Spezielle Optionen
1l) Client "mysql": Ausgabeformat-Optionen
1m) Client "mysql": Sonstige Optionen
1n) Client "mysql": Umgebungsvariablen
1o) Client "mysql": Interne Befehle (Escape-Sequenzen)
1p) Client "mysql": Hilfe anzeigen
1q) Client "mysql": Prompt-Definition
1r) Grafische MySQL-Programme (GUI)
2) Syntax
2a) Leerraum und Formatierung
2b) Zeichenketten (Strings) und Quotierung
2c) Escape-Sequenzen
2d) Zahlen
2e) Datenbank-Objekte
2f) Identifier (Bezeichner)
2g) GROSS/kleinschreibung
2h) Kommentare
3) Typische MySQL-Datenbankbefehle (Tutorial)
4) MySQL-Datentypen
4a) Datentyp-Optimierung (Performance/Speicherplatz)
4b) Fixe/Variable Record-LÃ¤nge (Row-Format)
4c) AUTO_INCREMENT
5) MySQL-Operatoren
6) Boolesche Logik
7) Der Wert NULL
7a) Eigenschaften von NULL
7b) PrÃ¼fung auf Wert NULL
7c) Vergleiche mit NULL
7d) Boolesche Logik mit NULL (ternÃ¤r/dreiwertig)
8) RegulÃ¤re AusdrÃ¼cke in MySQL (RegEx)
9) MySQL-Funktionen
9a) Aggregatfunktionen und Gruppierung (Aggregation)
10) SchlÃ¼ssel (Key) und Index
10a) Eigenschaften
10b) Erstellen/entfernen
10c) FULLTEXT-Index
10d) SPATIAL-Index
10e) Index-Nutzung
10f) Index-Optimierung
10g) FremdschlÃ¼ssel (Foreign Keys) und Referenzielle IntegritÃ¤t
```

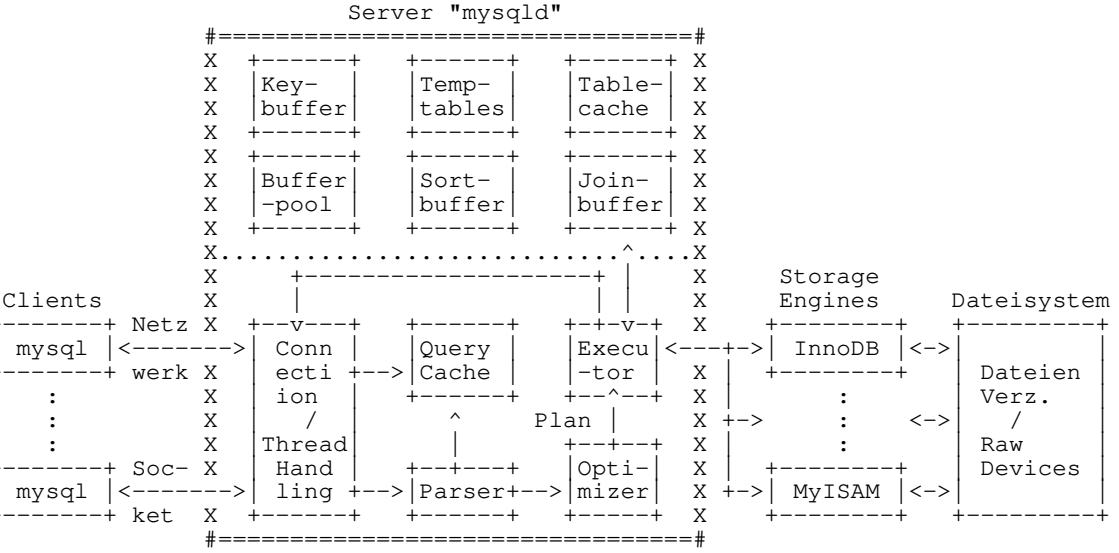
11) Joins
12) Mengen-Operationen
13) Unterabfragen (Subqueries/Subselect)
14) Transaktionen
15) Locking
16) Views (Sichten)
17) Variablen
18) Prepared Statements (Vorbereitete Anweisungen)
19) Stored Routines (Stored Procedures/Functions)
19a) Lokale Variablen
19b) Wertzuweisung an Variable
19c) Ausgabe von Variablen oder Daten
19d) Kontrollstrukturen (Block/Compound Statement)
19e) Kontrollstrukturen (Verzweigungen)
19f) Kontrollstrukturen (Schleifen)
19g) Prozeduren
19h) Funktionen
20) Trigger
21) Condition Handler (Error/Warning)
22) Cursor (Zeiger)
23) Table Handler
24) Events (Ereignisse)
25) Signale
25a) MySQL-Fehlercode / SQL-Status
26) Begrenzungen von MySQL (Limits)
27) MySQL testen

---

1) MySQL-Clients

1a) Zusammenspiel der MySQL-Komponenten

Architektur:
- N MySQL-Clients + 1 MySQL-Server (auf anderem oder gleichem Rechner)
  - Client: mysql (Kommandoschnittstelle, eine von vielen)
  - Server: mysqld (Daemon, Instanz)
- MySQL-Server-Schichten
  - Oben: Benutzer-Schnittstelle (API, SQL, DB-Management)
  - Mitte: Storage Engine
  - Unten: Dateisystem



Die Verbindung zwischen MySQL-Client und -Server erfolgt per Socket, Named Pipe, Memory oder TCP/IP (siehe Optionen "-h/--host" und "--protocol").

1b) MySQL Web-Client "phpMyAdmin" einrichten und aufrufen

A) Einrichtung per Paket "phpmyadmin" (Administrator "phpmyadmin"):

```
sudo apt-get install phpmyadmin         # Paket + abhÃ¤ngig. Pakete install.
/etc/phpmyadmin/config.inc.php          # Konfiguration
/usr/share/phpmyadmin/*                 # PHP-Code
/usr/share/doc/phpmyadmin/*             # Dokumentation
```

```
  http://localhost/phpmyadmin/index.php   # Anmeldung an Web-OberflÃ¤che

B) Apache Webserver mit PHP-UnterstÃ¼tzung installieren und phpMyAdmin-Quellcode
kopieren nach:

  /srv/www/htdocs/phpMyAdmin   # Variante A
  /var/www/htdocs/phpMyAdmin   # Variante B

Dazu vorher in Konfig-Datei (inc=include)

  /srv/www/htdocs/phpMyAdmin/config.inc.php      # Variante A
  /var/www/htdocs/phpMyAdmin/config.inc.php      # Variante B

einen gÃ¼ltigen MySQL-Benutzer mit gÃ¼ltigem Passwort eintragen,
der MySQL-Administrationsrechte (d.h. ALL PRIVILEGES + GRANT OPTION) besitzt:

  $cfg['Servers'][$i]['auth_type'] = 'config';   # Auth.meth. (config/http/cookie)
  $cfg['Servers'][$i]['user']      = 'root';     # MySQL Benutzer (Administrator)
  $cfg['Servers'][$i]['password']  = 'GEHEIM';   # MySQL Passwort (wg. 'config')

Den Apache-Webserver starten:

  /etc/init.d/apache2 start   # TemporÃ¤r (init)
  service apache2 start       # TemporÃ¤r (systemd)
  rcapache2 start             # TemporÃ¤r (OpenSUSE)
  chkconfig -a apache2        # Permanent (OpenSUSE)

Im Web-Browser dann Ã¼ber folgende URL die OberflÃ¤che "phpMyAdmin" aufrufen
und mit den in "config.inc.php" eingetragenen Benutzer-Daten anmelden:

  http://localhost/phpMyAdmin/index.php


1c) MySQL-Clients: Konfigurations-Dateien
-----------------------------------------

Die Einstellung der MySQL-Variablen und -Optionen stammen aus dem
MySQL-Kommando selbst sowie einigen Konfig-Dateien in der hier angegebenen
Reihenfolge (sofern vorhanden, die Einstellungen der zuletzt angegebenen
Ã¼berschreiben die Einstellungen der ersten):

  +---+-------------------+-----------------------------------+
  | N | Quelle (Datei)    | Bedeutung                         |
  +---+-------------------+-----------------------------------+
  | 0a| Client-Programm   | Global (zentral, fest eingebaut)  |  |
  | 0b| Server "mysqld"   | Global (zentral, fest eingebaut)  |  |
  +---+-------------------+-----------------------------------+  |
  | 1 | /etc/my.cnf       | Global (zentral)                  |  |
  | 2 | /etc/mysql/my.cnf | Global (zentral)                  |  | steigender
  | 3 | /usr/etc/my.cnf   | Global (zentral)                  |  | Vorrang
  +---+-------------------+-----------------------------------+  |
  | 4 | ~/.my.cnf         | Lokal (benutzerbezogen)           |  |
  | 5 | Umgebung          | Lokal (Umgebungsvariablen)        |  |
  | 6 | Kommandozeile     | Lokal (direkt beim Kommandoaufruf)|  v
  +---+-------------------+-----------------------------------+

HINWEIS: Optionen auf der Kommandozeile haben immer das "letzte Wort"!

Konfig-Dateien sind durch in eckige Klammern gesetzte Programmnamen in
"GRUPPEN" (Abschnitte) eingeteilt. BerÃ¼cksichtigt werden fÃ¼r den "mysql"-Client
Optionen folgender Gruppen (--> mysql-admin-HOWTO.txt --> 1a) Grundlegendes):

  [mysql]    # Programmname
  [client]   # Programmtyp

TIP: Mit folgenden Optionen wird das Einlesen der Konfig-Dateien
verÃ¤ndert bzw. der sich aus den Konfig-Dateien ergebende Default-Wert
der Optionen ausgegeben. Die ersten drei Optionen sind nur als 1. Argument nach
dem Kommandonamen (z.B. "mysql") erlaubt, --print-defaults muss direkt danach
stehen:

  +-----------------------------+-----------------------------------------------+
  | Option                      | Bedeutung                                     |
  +-----------------------------+-----------------------------------------------+
  | --no-defaults               | Keine Konf.dateien lesen                      |
  | --defaults-file=<Path>      | Default-Optionen NUR aus Datei <Path> lesen   |
  | --defaults-extra-file=<Path>| Datei <Path> ZUSÃM-^DTZL. nach glob. Konf.dat.  |
  +-----------------------------+-----------------------------------------------+
  | --print-defaults            | Programmargumente ausgeben und Abbruch        |
  +-----------------------------+-----------------------------------------------+

D.h. die in das Server-Programm "mysqld" bzw. das Client-Programm "mysql"
```

eingebauten Einstellungen erhÃ¤lt man per:

```
  mysqld --no-defaults --print-defaults   # Server
  mysql  --no-defaults --print-defaults   # Client
```

Die Einstellungen gemÃ¤Ã-^_ Konfig.Dateien von Server und Client erhÃ¤lt man per:

```
  mysqld --print-defaults   # Server
  mysql  --print-defaults   # Client
```

1d) MySQL-Clients: Allgemeine Optionen

Die Optionen der MySQL-Kommandozeilen-Programme sind in Kurzform "-X" sowie in Langform "--XXX" angebbar, bei der Kurzform ist die GROSS/kleinschreibung relevant, bei der Langform nicht. ALLE MySQL-Programme kennen folgende Optionen:

Kurzform	Langform	Bedeutung
-v	--verbose	Mehr Info ausgeben (mehrfach erlaubt)
-s	--silent	Weniger Info ausgeben (mehrfach erlaubt)
-V	--version	Versioninformation ausgeben
-?	--help	Hilfe zum Aufruf ausgeben (Syntax, Optionen)

ALLE MySQL-Kommandozeilen-Clients mit einer Verbindung zum MySQL-Server kennen folgende Optionen (legen Art und Weise der Verbindung fest):

Kurzform	Langform	Bedeutung
-h<Host>	--host=<Host>	Ziel-Host (STD: localhost)
-u<User>	--user=<User>	Benutzer festlegen (STD: OS-Anmeldung)
-p[<Pass>]	--password[=<Pass>]	Passwort abfragen bzw. direkt angeben
-D<Db>	--database=<Db>	Datenbank auswÃ¤hlen (STD: keine)
	--protocol=<Proto>	Verb.protokoll (tcp/socket/pipe/memory)
-P<Port>	--port=<Port>	Ziel-Port (STD: 3306)
-S<File>	--socket=<File>	Socket (STD: /var/lib/mysql/mysql.sock bzw. /var/run/mysqld/mysqld.sock)

Bedeutung von "<Proto>" bei Option --protocol:

<Proto>	Verbindung Client <-> Server erzwingen ...	Art
tcp	... per TCP (STD: Port 3306)	lokal+remote
socket	... per Socket (STD: /var/lib/mysql/mysql.sock)	nur lokal
pipe	... per Named Pipe (STD:	
memory	... per Shared Memory	nur lokal

HINWEIS: StandardmÃ¤Ã-^_ig wird lokal per "socket" und remote per "tcp" verbunden.

HINWEIS: Bei der Kurzform -X darf der Wert auch per Leerzeichen getrennt von der Option angegeben werden. AUSNAME: Das Passwort <Pass> ist DIREKT NACH der Option "-p" (ohne Leerzeichen dazwischen) anzugeben!

HINWEIS: Ohne Option "-p" wird eine Anmeldung OHNE Passwort versucht.

1e) MySQL-Clients: Verbindung zum Server (SSL)

Die MySQL-Clients versuchen je nach Aufruf auf folgende Arten mit dem MySQL-Server Verbindung aufzunehmen:

* Lokale Verbindung per Socket/Named Pipe/Shared Memory:
  Option "-h/--host" nicht oder mit Wert "localhost/127.0.0.1/::1" angegeben und Option "--protocol" nicht angegeben oder nicht auf Wert "tcp" gesetzt. MySQL-Client und -Server mÃ¼ssen dann natÃ¼rlich auf dem gleichen Host laufen oder per VPN verbunden sein.

* Netzwerk-Verbindung per TCP/IP:
  Option "-h/--host" angegeben und bezeichnet fremden Rechner/fremde IP-Adresse oder Option "--protocol" auf Wert "tcp" gesetzt.
  Auch dann, wenn rein lokale Verbindung mÃ¶glich wÃ¤re!

FÃ¼r eine per SSL verschlÃ¼sselte Verbindung zwischen Client und Server sind folgende Optionen anzugeben:

Option (nur Langform)	Bedeutung
--ssl	SSL fÃ¼r Verbindung wz. Cl. und S. einschalten (automatisch von anderen --ssl-Opt. aktiviert)
--skip-ssl	Abschalten der SSL-VerbindungsverschlÃ¼sselung
--ssl-ca=<File>	CA-Datei im PEM-Format (siehe OpenSSL-Doku)
--ssl-capath=<Dir>	CA-Verz. (siehe OpenSSL-Doku)
--ssl-cert=<File>	X509-Zertifikat in PEM-Format
--ssl-cipher=<Typ>	SSL-VerschlÃ¼sselungstyp (...)
--ssl-key=<File>	X509-SchlÃ¼ssel im PEM-Format
--ssl-verify-server-cert	"Common Name" des Servers in seinem Zertifikat mit dem beim Verb.aufbau benutzten Hostnamen vergleichen (STD: aus)

Ob ein Verbindung per SSL verschlÃ¼ssel werden MUSS, wird pro Benutzer beim Anlegen durch Angabe der Option REQUIRE <SSLOpt> festgelegt:

Option <SSLOpt>	Bedeutung
NONE	UnverschlÃ¼ss. Verbindung erlaubt (STD), verschl. auch
SSL	Nur SSL-verschlÃ¼sselte Verbindungen erlaubt
X509	GÃ¼ltiges Zertifikat notwendig, Issuer + Subject egal
CIPHER <Spec>	VerschlÃ¼sselung und SchlÃ¼ssel mÃ¼ssen best. Bed. genÃ¼gen (z.B. CIPHER "EDH-RSA-DES-CBC3-SHA")
ISSUER <String>	Zertifikat muss von CA <String> ausgestellt sein
SUBJECT <String>	Zertifikat muss Subject <String> enthalten

1f) Client "mysql" starten (Verbindung/Session)

Der Client "mysql" bietet eine einfache KOMMANDOZEILENBASIERTE Schnittstelle zum Absetzen von SQL-Anweisungen an einen MySQL-Server und zur Ausgabe ihrer Ergebnisse (MySQL-"Terminal/Monitor"). Trotzdem ist er sehr leistungsfÃ¤hig, weil MySQL (nahezu) vollstÃ¤ndig Ã¼ber SQL-Anweisungen steuerbar ist. Um diese Schnittstelle nutzen zu kÃ¶nnen, sind allerdings gute Kenntnisse der SQL-Anweisungen und ihrer Syntax notwendig.

Auf der Kommandozeile wird zunÃ¤chst per Client "mysql" eine VERBINDUNG mit dem MySQL-Server "mysqld" aufgenommen (eine SESSION gestartet), indem eine Anmeldung mit gÃ¼ltigem Benutzernamen + Passwort durchgefÃ¼hrt wird (-u=user, -p=password, -h=host, -D=database, -P=Port, -S=Socket). AnschlieÃM-^_end kÃ¶nnen solange SQL-Anweisungen abgesetzt werden, bis durch Verlassen des "mysql"-Clients die Verbindung zum MySQL-Server "mysqld" wieder beendet wird:

```
mysql -utom -pgeheim
```

Geht die Verbindung zum MySQL-Server verloren (z.B. durch Timeout), dann Ã¶ffnet "mysql" beim nÃ¤chsten Kommando AUTOMATISCH eine neue Verbindung (reconnect). Da dies fast unmerklich geschieht und beim Verbindungsabbruch Sitzungsdaten wie lokale Variablen und sonstige Einstellungen verloren gehen, ist dies auch verhinderbar durch:

```
mysql --skip-reconnect ...
```

Typische Aufrufe des "mysql"-Clients:

Kommando	Bedeutung
mysql	Als angemeldeter OS-Benutzer aufrufen
mysql -utom	Benutzer "tom" ohne Passwort (unsinnig!)
mysql -utom -p	Benutzer "tom" (Passwort interaktiv eingeb)
mysql -utom -pgeheim	Passwort "geheim" gleich mitgeben (ungut!)
mysql -utom -pgeheim -htest	Mit Rechner "test" verbinden (host)
mysql -utom -pgeheim -P1234	Mit Port "1234" verbinden (STD: 3306)
mysql -utom -pgeheim -S/tmp/x	Mit Socket verbinden
mysql -utom -pgeheim -Dprod	Datenbank "prod" auswÃ¤hlen (Variante A)
mysql -utom -pgeheim prod	Datenbank "prod" auswÃ¤hlen (Variante B)

Hilfe zum Aufruf des "mysql"-Client anzeigen (Optionen, Reihenfolge der Konfig-Dateien, Werte von Variablen und Optionen):

```
  mysql --help
  mysql -?

Weitere Einstellungen zur Verbindung:  !!!

  --connect-timeout=<N>      # Sek. bis autom. Verbindungstrenn. (STD: 0=nie)
  --max-allowed-packet=<N>   # Max. LÃ¤nge transf. Befehle/Ergebnisse (STD: 16MB)
  --net-buffer-length=<N>    # PuffergrÃ¶Ã-^_e TCP/IP+Socket-Komm. (STD: 16KB)
  --skip-reconnect           # Keine automat. Neuverbindung nach Trennung

Durch Setzen des Timeout wird der Client bei lÃ¤ngerer Nichtbenutzung
automatisch verlassen (STD: 0 = kein Timeout):

TIP: Aliase fÃ¼r "mysql"-Anmeldung + Umschalten auf aktuell relevante Datenbank
definieren und in "~/.alias" oder "~/.bash_aliases" ablegen (-D=Standard-DB,
-t=ASCII-Rahmen um Ausgaben zeichnen, -e=Befehl ausfÃ¼hren):

  alias my="mysql -utom -pgeheim -Dtest -t"       # Interaktiv/Batch per Umlenk.
  alias mye="mysql -utom -pgeheim -Dtest -t -e"   # SQL direkt auf Kommandozeile

Aufruf durch:

  my                          # Interaktiv-Modus starten
  my < <SqlFile>              # Batch-Datei mit SQL-Anweisungen einlesen
  mye 'SELECT * FROM pers'    # SQL-Anweisung direkt ausfÃ¼hren


1g) Client "mysql": Befehlsein- und ausgabe (Prompt, Strg-C, TEE)
-----------------------------------------------------------------

Nach der Anmeldung am "mysql-Client" erscheint ein PROMPT folgender Form, der
die interaktive Eingabe beliebiger SQL-Anweisungen (Statements) erwartet.
Abschluss der Eingabe per <RETURN> Ã¼bergibt diese dem MySQL-Server zur
AusfÃ¼hrung und zeigt das Ergebnis sowie statistische Informationen an.

  mysql> ...   # Wartet auf Kommando-Eingabe + <RETURN>

HINWEIS: Echte SQL-Anweisungen (werden an MySQL-Server geschickt) sind mit
         ";" abzuschlieÃM-^_en! Client-spezifische Befehle ("USE <Db>" oder "QUIT")
         dÃ¼rfen ohne abschlieÃM-^_enden ";" geschrieben werden (besser angeben).

Um z.B. alle Datenbanken anzuzeigen, ist folgender Dialog zu fÃ¼hren (die
SQL-Anweisung ist hier GROSS geschrieben, GROSS/kleinschreibung ist aber
eigentlich egal, siehe auch --> 2g) GROSS/kleinschreibung):

  mysql> SHOW DATABASES;<RETURN>     # Prompt + Benutzer-EINGABE (";" notwendig)
  +----------------------------+     # ASCII-Rahmen            (AUSGABE)
  | Database                   |     # SpaltenÃ¼berschrift      (AUSGABE)
  +----------------------------+     # ASCII-Rahmen            (AUSGABE)
  | INFORMATION_SCHEMA         |     # 1. Ergebniszeile        (AUSGABE)
  | mysql                      |     # 2. Ergebniszeile        (AUSGABE)
  | PERFORMANCE_SCHEMA         |     # 3. Ergebniszeile        (AUSGABE)
  | phpmyadmin                 |     # 4. Ergebniszeile        (AUSGABE)
  | ...                        |     # ...                     (AUSGABE)
  +----------------------------+     # ASCII-Rahmen            (AUSGABE)
  15 rows in set (0.01 sec)          # Statistik (15 Ergeb.zeilen + Laufzeit)
                                     # (Leerzeile)
  mysql>                             # Prompt nÃ¤chste Benutzer-EINGABE

oder (leeres Ergebnis):

  mysql> SHOW DATABASES LIKE "xyz";  # Prompt + Benutzer-EINGABE + <RETURN>
  Empty set (0.00 sec)               # Statistik (leeres Ergebnis + Laufzeit)
                                     # (Leerzeile)
  mysql>                             # Prompt nÃ¤chste Benutzer-EINGABE

oder (Fehler):

  mysql> SHOW TABLES;                        # Prompt + Benutzer-EINGABE
  ERROR 1046 (3D000): No database selected   # Fehlermeldung
  mysql>                                     # Prompt nÃ¤chste Benutzer-EINGABE

Fehlermeldungen, Warnungen und Bemerkungen (Notes) nach einer SQL-Anweisung
ausgeben (bezieht sich immer auf die vorhergehende SQL-Anweisung):

  SHOW ERRORS;     # Nur Fehler ausgeben
  SHOW WARNINGS;   # Fehler, Warnungen, Bemerkungen (Notes) ausgeben

Nach dem <RETURN> sendet der Client "mysql" das Kommando an den MySQL-Server
"mysqld", mit dem die Verbindung besteht, wartet auf das Ergebnis und gibt es
auf dem Bildschirm aus. Die Ausgabe erfolgt in tabellarischer Form (Zeilen +
```

```
Spalten). Die erste Zeile enthÃ¤lt die SpaltenÃ¼berschrift. Am Ende wird die
Anzahl der erhaltenen DatensÃ¤tze (N rows) und die Dauer der Abfrage (N.NN sec)
ausgegeben. AnschlieÃM-^_end wird erneut der Prompt angezeigt und auf die Eingabe
des nÃ¤chsten Kommandos gewartet. Hier noch ein Beispiel:

  mysql> SELECT SIN(PI()/2), (4+1)*5;   # Prompt + Benutzer-Eingabe
  +-------------+---------+             # Ausgabe (ASCII-Rahmen)
  | SIN(PI()/2) | (4+1)*5 |             # Ausgabe (ÃM-^\berschrift)
  +-------------+---------+             # Ausgabe (ASCII-Rahmen)
  |           1 |      25 |             # Ausgabe (1. Datensatz)
  +-------------+---------+             # Ausgabe (ASCII-Rahmen)
  1 row in set (0.00 sec)               # Statistik (eine Ergeb.zl. + Laufzeit)
                                        # (Leerzeile)
  mysql>                                # Prompt nÃ¤chste Benutzer-EINGABE

Fortsetzungs-Prompt "->": SQL-Kmdos mÃ¼ssen nicht auf einer Zeile stehen, sie
kÃ¶nnen mehrere Zeilen verteilt sein. MySQL erkennt das Kommando-Ende am
abschlieÃM-^_enden ";" und gibt bis zum abschlieÃM-^_enden ";" den Fortsetzungs-Prompt
"->" aus:

   mysql> SELECT user,          # Kmdo-Beginn
       ->        password,     # Kmdo-Teil
       ->        host          # Kmdo-Teil
       -> FROM mysql.user      # Kmdo-Teil
       -> ;                    # Kmdo-Ende ";"

UnvollstÃ¤ndige SQL-Anweisungen (z.B. wird der ";" gerne vergessen) fÃ¼hren zur
Anzeige eines der folgenden Fortsetzungs-Prompts (dann einfach restliche
Kommandoteile oder fehlendes ";" tippen und <RETURN> drÃ¼cken):

  +--------+------------+-------------------------------------------+
  | Prompt | Element    | Bedeutung                                 |
  +--------+------------+-------------------------------------------+
  | ->     | Kommando   | UnvollstÃ¤ndig bzw. ";" vergessen          |
  | ">     | String     | "..."   begonnen aber noch nicht beendet  |
  | '>     | String     | '...'   begonnen aber noch nicht beendet  |
  | `>     | Bezeichner | `...`   begonnen aber noch nicht beendet  |
  | /*>    | Kommentar  | /*...*/ begonnen aber noch nicht beendet  |
  +--------+------------+-------------------------------------------+

Ausgaben des "mysql"-Client in einer Datei <OutFile> aufzeichnen:

  mysql ... --tee=<OutFile>

Die Ausgabe auf Datei kann auch interaktiv im "mysql"-Client aktiviert und
wieder deaktiviert werden durch:

  mysql> TEE <OutFile>   # Aufzeichnung starten
  mysql> ...             # Ergebnisse aufzeichnen
  mysql> NOTEE           # Aufzeichnung beenden
  mysql> ...             # Ergebnisse NICHT aufzeichnen
  mysql> \T <OutFile>    # Aufzeichnung starten (Escape-Sequenz)
  mysql> ...             # Ergebnisse aufzeichnen
  mysql> \t              # Aufzeichnung beenden (Escape-Sequenz)

HINWEIS: Vorteil dieser Art der Ausgabesteuerung ist, dass die Ausgabe sowohl
auf dem Bildschirm als auch in eine Datei <OutFile> erfolgt.


1h) Client "mysql" effizient bedienen (USE, History, Strg-L, TAB, Hilfe, SOURCE)
--------------------------------------------------------------------------------

Standard-Datenbank per USE: Als 1. Kommando immer folgendes verwenden, um eine
der vorhandenen Datenbanken als DEFAULT/STANDARD-DATENBANK auszuwÃ¤hlen. Die
Objekte darin sind dann leicht Ã¼ber ihre Namen (ohne Qualifizierung mit dem
Datenbanknamen) erreichbar. Der ";" am Ende darf bei "USE" weggelassen werden
(besser erst gar nicht daran gewÃ¶hnen!):

  USE <Db>    # Variante A
  USE <Db>;   # Variante B (besser)

Name der Standard-Datenbank ausgeben ("NULL" falls noch keine ausgewÃ¤hlt):

  SELECT DATABASE();   # Variante A
  SELECT SCHEMA();     # Variante B

MySQL-History: Alte Befehle kÃ¶nnen per Cursortasten durchgeblÃ¤ttert, editiert
und erneut mit <RETURN> ausgefÃ¼hrt werden. Mehrzeilig Befehle werden dabei zu
einer (langen) Zeile zusammengezogen, die evtl. schwer zu editieren ist. Die
Befehle stehen auch nach Verlassen und erneutem Aufrufen des Clients "mysql"
wieder zur VerfÃ¼gung, da sie PRO BENUTZER in seinem Heimatverz. in folgender
Datei gespeichert werden (d.h. root <-> normaler Benutzer getrennt):
```

```
  ~/.mysql_history

Da ALLE im "mysql"-Client eingegebenen Befehle in der MySQL-History landen,
kÃ¶nnen darin evtl. Passworte oder andere sicherheitsrelevante Daten stehen. Die
Zugriffsrechte der Datei sollten daher 600 lauten ("rw" nur fÃ¼r Besitzer). Um
den Datei-Inhalt schnell zu leeren, folgendes Kommando absetzen:

  > ~/.mysql_history   # Leere Datei umlenken

Um die MySQL-History NICHT aufzuzeichen, folgendes einstellen:

  export MYSQL_HISTFILE="/dev/null"                          # Variante A
  rm ~/.mysql_histfile && ln -s /dev/null ~/.mysql_histfile  # Variante B

Befehle oder Befehlsteile kÃ¶nnen auch per MAUS (linke Taste selektiert +
mittlere Taste kopiert) in die Kommandozeile des "mysql"-Clients kopiert
werden. Wird bis zum rechten Terminalrand markiert, ist der Befehlsabschluss
<RETURN> gleich mit in der Kopie enthalten (unter Linux).

TAB-Completion: Datenbank-, Tabellen- und Spalten-Namen (aber keine anderen
SQL-Syntax-Elemente) kÃ¶nnen per <TAB>-Taste AUTOMATISCH VERVOLLSTÃM-^DNDIGT werden
(Tab-Completion). Dazu per "-D <Db>" oder "USE <Db>;" eine
Default/Standard-Datenbank auswÃ¤hlen und beim Aufruf von "mysql" die Option
"--auto-rehash" angeben oder in "mysql" den Befehl "REHASH" eingeben
(verlangsamt den Start etwas), um den Datenbankinhalt einzulesen.

  mysql -utom -pgeheim --auto-rehash mysql      # Variante A
  mysql -utom -pgeheim --auto-rehash -D mysql   # Variante B
  mysql -utom -pgeheim                          # Variante C
    USE mysql;                                  # Variante C
    REHASH                                      # Variante C

Bildschirm LEEREN im "mysql"-Client (Prompt "mysql> " steht dann ganz oben):

  <Strg-L>   # TIP: L=Leeren (eigentlich Steuerzeichen FF=FormFeed ;-)

HILFE zu Kommandos im "mysql"-Client durch folgende Befehle anzeigen (die
Hilfetexte stehen direkt in der Datenbank in den Tabellen "mysql.help_..."):

  +------------------+-------------------------------------------------------+
  | Befehl           | Bedeutung                                             |
  +------------------+-------------------------------------------------------+
  | HELP             | Liste der internen "mysql"-Client-Befehle ausgeben    |
  | HELP HELP        | Hilfe zur Hilfe selbst ausgeben                       |
  | HELP <SqlCmd>... | Hilfe zu SQL-Anweisung ausgeben (eindeut. PrÃ¤fix langt) |
  | HELP CONTENTS    | Liste der Hilfe-Themen ausgeben                       |
  | HELP <Topic>     | Hilfe zu einem Thema aus Themenliste ausgeben         |
  +------------------+-------------------------------------------------------+

TIP: UPDATE- und DELETE-Anweisungen gegen versehentliches Weglassen der WHERE-
oder LIMIT-Klausel schÃ¼tzen (d.h. versehentliches Ã M-^Dndern/LÃ¶schen ALLER
DatensÃ¤tze verhindern) und Anzahl der Ergebniszeilen bzw. VerknÃ¼pfungszeilen
beschrÃ¤nken:

  +----+--------------------------+-------------------------------------------+
  |    | Langform                 | Bedeutung                                 |
  +----+--------------------------+-------------------------------------------+
  | -U | --safe-updates           | UPDATE- und DELETE-Anweisungen gegen      |
  |    | --i-am-a-dummy           |   Weglassen von WHERE und LIMIT schÃ¼tzen  |
  +----+--------------------------+-------------------------------------------+
  |    | --select-limit=<N>       | Abbruch Erg.ausgabe ab N Zl. (STD: 1000)  |
  |    | --max-join-size=<N>      | Abbruch Erg.ausg. ab N Zl.komb. (STD: 1Mio)|
  +----+--------------------------+-------------------------------------------+

TIP: SQL-Anweisungen aus externer Datei einlesen (Include):

  SOURCE <SqlFile>;              # Kein "..." um <SqlFile>, vollst. Pfad!


1i) Client "mysql" verlassen (Strg-C)
-------------------------------------

Durch folgende Eingaben kann der "mysql"-Client verlassen werden:

  +--------+-----------------------------------------------------+
  | Befehl | Bedeutung                                           |
  +--------+-----------------------------------------------------+
  | QUIT   | Quit-Befehl                                         |
  | EXIT   | Exit-Befehl                                         |
  | \q     | Quit (Escape-Sequenz)                               |
  +--------+-----------------------------------------------------+
```

Strg-C	Abbruch (Cancel, deaktivieren mit "--sigint-ignore")
Strg-D	D=Dateiende (nur Linux)
Strg-Z	Z=Dateiende (nur Windows)

DrÃ¼cken von Strg-C zum Abbrechen der aktuellen SQL-Anweisung sollte man sich abgewÃ¶hnen. In der Shell-Kommandozeile ist man daran gewÃ¶hnt, dass die Shell dabei nicht abgebrochen und man nicht abgemeldet wird. Aus dem Client "mysql" fliegt man hingegen sofort raus und muss sich mÃ¼hsam wieder anmelden.

Ruft man den "mysql"-Client mit Option "--sigint-ignore" auf, so beendet Strg-C nur noch den aktuellen Befehl, bricht aber "mysql" nicht mehr ab:

```
mysql -utom -pgeheim --sigint-ignore -Dtest
```

TIP: Am besten in Konfig-Datei "~/.my.cnf" eintragen.

1j) Client "mysql" auf Kommandozeile nutzen (Batch-Modus)

Neben der interaktiven Nutzung kann der Client "mysql" auch im Batch-Modus benutzt werden. Dazu die SQL-Anweisung direkt nach der Option "-e/--execute" angeben. Mehrere SQL-Anweisungen durch ";" trennen, der letzte ";" darf fehlen (d.h. einzelne SQL-Anweisung muss nicht mit ";" abgeschlossen werden):

```
mysql -utom -pgeheim -e "USE mysql; SELECT user, host, password FROM user"
```

Alternativ kÃ¶nnen die SQL-Anweisungen per Eingabe-Umleitung aus einer Datei <SqlFile> eingelesen werden (analog dem Kommando "SOURCE" im "mysql"-Client). Ebenso kÃ¶nnen Ergebnisse von SQL-Anweisungen statt auf dem Bildschirm per Ausgabe-Umleitung auch auf eine Datei <OutFile> ausgegeben werden.

Kommando	Bedeutung
`mysql -u<User> -p -e "<SqlCmd>"`	execute
`... -e "USE first; SELECT * FROM pers"`	execute
`... -e "USE first; INSERT INTO pers VALUES (100, 'Hans', 'Dampf'); INSERT INTO pers VALUES (101, 'HÃ¤nschen', 'Klein')"`	execute (mehrere Kommandos durch ";" trennen!) (Statement in "..." setzen, Strings darin in '...')
`mysql -u<User> -p < <SqlFile>`	Eingabe von Datei <SqlFile>
`..... -u<User> -p -D<Db> < <SqlFile>`	(USE <Db> nicht vergessen!)
`..... -u<User> -p -D<Db> <<EOF` <Stmt>; ... EOF	Here-Dokument von EOF SQL-Statement ... bis EOF (End Of File)
`mysql ... -e "SOURCE <SqlFile>"`	Eingabe von Datei <SqlFile>
`mysql ... > <OutFile>`	Ausgabe auf Datei <Outfile>

Die Eingabedatei <SqlFile> wird oft manuell erstellt und kann neben der Definition von Tabellen auch Daten zum FÃ¼llen der Tabellen enthalten. HÃ¤ufig entsteht die Eingabedatei auch durch einen teilweisen/vollstÃ¤ndigen "Dump" einer MySQL-Datenbank per Kommando "mysqldump". Es erzeugt SQL-Anweisungen, die mit dem "mysql"-Client ausgefÃ¼hrt werden kÃ¶nnen, um Struktur + Inhalt der Datenbank wieder zu erstellen:

```
mysqldump -utom -pgeheim test > test-dump.sql   # Dump von DB "test" erzeugen
mysql -utom -pgeheim -e "DROP DATABASE test"    # DB "test" lÃ¶schen (Inhalt!)
mysql -utom -pgeheim -e "CREATE DATABASE test"  # DB "test" anlegen (leer)
mysql -utom -pgeheim -Dtest < test-dump.sql     # DB-Dump "test" einspielen
```

Alternativ (Dump-Datei editieren) folgende Anweisungen am Anfang der Dumpdatei "test-dump.sql" hinzufÃ¼gen:

```
mysqldump -utom -pgeheim test > test-dump.sql   # Dump von DB "test" erzeugen
edit test-dump.sql                              # SQL-Dumpdatei editieren:
  DROP DATABASE test;                           # - DB "test" lÃ¶schen (Inhalt!
  CREATE DATABASE IF NOT EXISTS test;           # - DB "test" anlegen (leer)
  USE test;                                     # - Auf DB "test" umschalten
  ...                                           # - Restliche SQL-Anweisungen
```

Und dann die Sicherung der Datenbank einspielen:

```
mysql -utom -pgeheim < test-dump.sql
```

HINWEIS: Bei einem SQL-Syntaxfehler wird die Verarbeitung sofort abgebrochen
(auÃM-^_er Option "-f/--force" ist gesetzt).

1k) Client "mysql": Spezielle Optionen

Die Optionen des "mysql"-Clients sind in Kurzform "-X" sowie in Langform "--XXX" angebbar, bei der Kurzform ist die GROSS/kleinschreibung relevant, bei der Langform nicht. Folgende Optionen des "mysql"-Client wurden in den vorherigen Abschnitten schon beschrieben:

Opt	Langform	Bedeutung
-e	--execute="<SqlCmd>"	SQL-Anweisung ausfÃ¼hren
	--auto-rehash	Tab.+Spaltennamen vervollst. (nach USE)
-A	--no-auto-rehash	Tab.+Spaltennamen NICHT vervollst. (alt)
-A	--skip-auto-rehash	Tab.+Spaltennamen NICHT vervollst. (alt)
-A	--disable-auto-rehash	Tab.+Spaltennamen NICHT vervollst. (neu)
	--sigint-ignore	Signal INT (Strg-C) ignorieren
	--connect-timeout=<N>	Sek. bis autom. Verbindungstrenn. (STD: 0)
	--reconnect	Bei Verb.verlust diese autom. wieder aufb.
	--skip-reconnect	Bei Verb.verlust diese NICHT autom. aufb.
	--max-allowed-packet=<N>	Max. LÃ¤nge transf. Bef./Ergeb. (STD: 16MB)
	--net-buffer-length=<N>	PuffergrÃ¶ÃM-^_e TCP/IP+Socket-Komm. (STD: 16KB)
-U	--safe-updates	UPDATE- und DELETE-Anweisungen gegen
	--i-am-a-dummy	Weglassen von WHERE und LIMIT schÃ¼tzen
	--select-limit=<N>	Abbruch Erg.ausgabe ab N Zl. (STD: 1000)
	--max-join-size=<N>	Abbruch Erg.ausg. ab N Zl.komb. (STD: 1Mio)

Folgende Optionen legen das Verhalten bei fehlerhaften SQL-Anweisungen fest (STD: Bei SQL-Syntaxfehler Verarbeitung der restlichen Anw. abbrechen):

Kurzform	Langform	Bedeutung
-f	--force	Bei einem SQL-Fehler nicht abbrechen
	--show-warnings	Warnungen nach jeder Anweisung anzeigen

1l) Client "mysql": Ausgabeformat-Optionen

Folgende Optionen legen das AUSGABEFORMAT fest:

Kurzform	Langform	Bedeutung
-t	--table	ASCII-Rahmen (STD im Interaktiv-Modus)
-B	--batch	Spalten TAB-getrennt (STD im Batch-Modus)
-E	--vertical	vErtikale-Ausgabe (nÃ¼tzlich!)
-H	--html	HTML-Format (1 Zeile ohne NL!)
-X	--xml	XML-Format (mehrzeilig + eingerÃ¼ckt)
-r	--raw	Ohne Escape-Umwandlung ausgeben (*)

(*) HINWEIS: Der Client "mysql" wandelt beim Einlesen Escape-Sequenzen wie \n \r \t \b \a \0 \\ \" \' in echte (Steuer-)Zeichen um. Bei der Ausgabe erfolgt die umgekehrte Umwandlung Steuerzeichen nach Escape-Sequenzen nur dann, wenn KEINE der Optionen -t, -X, -E, --raw angegeben wird.

Standard-Ausgabeformate:

* Interaktiver Modus (Eingabe-GerÃ¤t ist EIN Terminal) --> Option -t/--table
  Werte jeder Ausgabespalte werden auf grÃ¶sste notwendige Breite formatiert
  und Ausgabe wird mit einem "ASCII-Rahmen" versehen.

* Batch-Modus (Eingabe-GerÃ¤t ist KEIN Terminal) --> Option -B/--batch
  Werte jeder Ausgabespalte werden nicht auf einheitliche Breite
  formatiert, sondern durch je einen TABULATOR getrennt.

TIP: FÃ¼r das menschliche Auge ist das 1. Format angenehmer zu lesen,

```
     fÃ¼r den Computer ist das 2. Format besser zu verarbeiten.

Bei breiten Tabellen ist das per Option "-E" aktivierte "vErtikale Format"
sinnvoll. Die Spaltenwerte eines Datensatzes werden dann zeilenweise mit den
Spaltennamen als PrÃ¤fix ausgegeben und jeder Datensatz eingeleitet durch eine
Zeile "*** N. row ***".

  *************************** 1. row ***************************
       nr: 77
  vorname: heinz
     name: bayer
  *************************** 2. row ***************************
       nr: 88
  vorname: Andrea
     name: Bayer
  *************************** 3. row ***************************
       nr: 99
  vorname: Richard
     name: Seiler
  3 rows in set (0.00 sec)

TIP: Im "mysql"-Client durch Stmt-Abschluss "\G" (go) statt ";" auslÃ¶sen.


1m) Client "mysql": Sonstige Optionen
-------------------------------------
```

Opt	Langform	Bedeutung	
	--column-names	Spaltennamen-Ã¤M-^\berschrift anzeig. (STD)	
-N	--skip-column-names	Spaltennamen-Ã¤M-^\berschrift weglassen	
-m	--column-type-info	Spaltentyp anzeigen (Metadaten)	
-C	--compress	DatenÃ¼bertr. zw. Cl. + Server komprim.	
-#	--debug=<Opts>	Debuglogfile gemÃ¤Ã¤M-^_ <Opts> schreiben	!!!
-T	--debug-info	Einige Debuginfo am Programmende ausg.	
	--debug-check	Einige Debuginfo am Programmende ausg.	
	--default-character-set=<N>	Standard-Zeichensatz <Name>	
	--character-sets-dir=<Dir>	Verz. <DirPath> der ZeichensÃ¤tze	
-L	--line-numbers	Zeilennummer bei Fehler ausgeben (STD)	
	--skip-line-numbers	Zeilennummer bei Fehler NICHT ausgeb.	
	--tee=<OutFile>	Ausgabe auf Datei <OutFile> zusÃ¤tzlich	
	--no-tee	Ausgabe auf Datei wieder abschalten	
-n	--unbuffered	Ausgabe nach jeder Abfrage leeren	
-w	--wait	Warten+neu versuch. Verb. aufzubauen	
	--reconnect	Bei Verb.verlust diese wieder aufbauen	
	--skip-reconnect	Bei Verb.verlust Client verlassen	
-c	--comments	Kommentare zum Server senden (erhalten)	
	--skip-comments	Kommentare NICHT zum Server send (STD)	
-G	--named-commands	Interne Client-Kmdos Ã¼berall erlaubt	
-g	--disable-named-commands	Interne Client-Kmdos nur in 1.Zl erl.	-g
	--no-named-commands	Analog (veraltet)	
	--delimiter=<Text>	SQL-Kmdo-Begrenzer def. (STD: ";")	
-i	--ignore-spaces	Leerzeichen nach Fkt.name ignorieren	
	--pager=<CmdPath>	Seitenweises BlÃ¤tterkmdo. <CmdPath>	
	--disable-pager	Kein seitenweises BlÃ¤ttern	
	--no-pager	Kein seitenweises BlÃ¤ttern (veraltet)	
	--prompt=<Text>	Prompt-Definition (STD: "mysql> ")	
-o	--one-database	Nur -D<Db> zÃ¤hlt, USE <Db> ignoriert	
-q	--quick	Abfrageergebnisse nicht cachen	
	--secure-auth	Altes Protokoll (vor 4.1.1) verhindern	

```
1n) Client "mysql": Umgebungsvariablen
---------------------------------------

Folgende Umgebungsvariablen werden vom Client "mysql" ausgewertet:
```

Umgebungsvariable	Bedeutung
MYSQL_DEBUG	Debug-Trace-Optionen
MYSQL_HISTFILE	MySQL-History-Datei (STD: "$HOME/.mysql_history")
MYSQL_HISTIGNORE	MySQL-History-Datei ignorieren
MYSQL_HOME	Server-spezifische "my.cnf"-Datei
MYSQL_HOST	Hostname
MYSQL_PS1	Client-Prompt-String (STD: "mysql> ")
MYSQL_PWD	Passwort (unsicher!)
MYSQL_TCP_PORT	TCP/IP-Port (STD: 3306)
MYSQL_UNIX_PORT	Socket (bei "localhost" benutzt)
MYSQL_GROUP_SUFFIX	
DATA_VAR_MYSQL	
PAGER	Programm zum seitenweisen BlÃ¤ttern (--pager)
EDITOR	Editor fÃ¼r interaktives Bearbeiten (1. Wahl)
VISUAL	Editor fÃ¼r interaktives Bearbeiten (2. Wahl)
USER	Benutzer
HOME	Heimatverzeichnis des Benutzers
PATH	Shell-Suchpfad fÃ¼r Programme
LD_RUN_PATH	Suchpfad fÃ¼r Bibliothek "libmysqlclient.so"
UMASK	Maske fÃ¼r neu angelegte Dateien
UMASK_DIR	Maske fÃ¼r neu angelegte Verz.
TZ	Lokale Zeitzone

1o) Client "mysql": Interne Befehle (Escape-Sequenzen)

Neben SQL-Anweisungen versteht der "mysql"-Client folgende INTERNEN BEFEHLE bzw. ihre AbkÃ¼rzung in Form einer zweibuchstabigen Escape-Sequenz (ein ";" als Abschluss ist hier nicht notwendig, schadet aber auch nicht):

Befehl	ESC	Bedeutung
? [<Arg>]	\?	Synonym fÃ¼r "help" (Argument <Arg>)
CHARSET <C>	\C	Zeichensatz <C> einschalten (fÃ¼r binlog-Verarbeitung)
CLEAR	\c	SQL-Anweisung abbrechen
CONNECT [D H]	\r	Serververbindung neu aufbauen (Opt: <Db> und <Host>)
DELIMITER <S>	\d	SQL-Kmdo-Begrenzer <S> def. (Zeilenrest, STD: ";")
EDIT	\e	Letzte SQL-Anweisung per $EDITOR/$VISUAL bearbeiten
EGO	\G	SQL-Anweisung ausfÃ¼hren (vErtikale Anzeige)
EXIT	\q	"mysql"-Client verlassen (auch Strg-C, Strg-D)
GO	\g	SQL-Anweisung ausfÃ¼hren (STD: ASCII-Tabellen Anz.)
HELP	\h	Diese Hilfe anzeigen (auch \?)
NOPAGER	\n	Seitenweise Anzeige ausschalten (stdout)
NOTEE	\t	Nicht mehr in Ausgabedatei schreiben
NOWARNING	\w	Keine Warnungen nach jeder Anweisung anzeigen (STD)
PAGER [<Cmd>]	\P	Seitenweise Anzeige ein (STD: $PAGER, z.B. "less")
PRINT	\p	Aktuelle SQL-Anweisung ausgeben
PROMPT [<Str>]	\R	"mysql"-Client-Prompt <Str> festleg. (STD: "mysql> ")
QUIT	\q	"mysql"-Client verlassen (auch Strg-C, Strg-D)
REHASH	\#	TAB-Completion-Hash aufbauen
RESETCONNECTION	\x	Verb. neu aufbauen (ab MY!5.7.3)
SOURCE <File>	\.	SQL-Datei <File> einlesen und ausfÃ¼hren (Include)
STATUS	\s	Server-Status anzeigen
SYSTEM <Cmd>	\!	System-Kommando <Cmd> ausfÃ¼hren (Shell-Escape)
TEE <File>	\T	Ausgabedatei <File> setzen (alles zusÃ¤tzlich dorthin)
USE <Db>	\u	Auf Default/Standard-Datenbank <Db> umschalten
WARNING	\W	Warnungen nach jeder Anweisung anzeigen

1p) Client "mysql": Hilfe anzeigen

Im Client "mysql" ist zu den Client-internen Befehlen und den SQL-Anweisungen jederzeit Hilfe anzeigbar (die Hilfetexte stehen in der internen Verwaltungs-Datenbank "mysql" in den Tabellen "mysql.help_...").

Hilfe-Kommando	Bedeutung
?	Liste der Client-internen Befehle
\?	Liste der Client-internen Befehle
\h	Liste der Client-internen Befehle
HELP	Liste der Client-internen Befehle

HELP HELP	Hilfe zur Hilfe
HELP <SqlCmd>	Hilfe zu SQL-Anweisung (z.B. JOIN)
HELP CONTENTS	Hilfethemenliste (Topics)
HELP <Topic>	Hilfe zu Thema aus Themenliste
... Account Management	Hilfethema
... Administration	"
... Compound Statements	"
... Data Definition	"
... Data Manipulation	"
... Data Types	"
... Functions	"
... Functions and Modifiers for Use with GROUP BY	"
... Geographic Features	"
... Help Metadata	"
... Language Structure	"
... Plugins	"
... Procedures	"
... Storage Engines	"
... Table Maintenance	"
... Transactions	"
... Triggers	"
... User-Defined Functions	"
... Utility	"

1q) Client "mysql": Prompt-Definition

Das Aussehen des Prompts kann definiert werden per (STD: "mysql> "):

```
PROMPT <Text>   # Escape-Sequenzen im <Text> erlaubt (ohne Quotierung "...")
PROMPT          # RÃ¼ckkehr zum STD.-Prompt "mysql> " (Leerz. am Ende!)
```

Beispiel (mit Escape-Sequenzen \h=host, \u=user, \d=database):

```
PROMPT \u@\h:\d>\          # Leerzeichen am Ende!
```

"Escape-Sequenzen" um spezielle Zeichen oder variable Werte anzuzeigen:

Esc	Name	Beschreibung
\		Leerzeichen (dem Backslash folgt ein Leerzeichen)
_		Leerzeichen (Unterstrich!)
\"		GÃ¤nsefÃ¼Ã¶M-^_chen
\'		Hochkomma
\\		Backslash-Zeichen "\"
\S	Semicolon	Semikolon ";"
\t	tabulator	Tabulator-Zeichen
\n	newline	Zeilenvorschub
\c	count	AnweisungszÃ¤hler (pro Anweisung um 1 erhÃ¶ht)
\l	delimiter	Aktueller Kmdo-Begrenzer (STD: ";", MY!5.0.25)
\C	connection	Aktuelle Verbindungs-ID (ab MY!5.7.6)
\d	database	Default-Datenbank
\h	host	MySQL-Server
\p	port	Aktueller TCP/IP-Port oder Socket-Datei
\u	user	Benutzername
\U	User	VollstÃ¤ndiger Benutzername "user_name@host_name"
\v	version	MySQL-Server Version
\D	date	Aktuelles Datum "DD.MM.YYYY"
\O	mOnth	Aktueller Monat 3 Zeichen (Jan, Feb, ...)
\o	month	Aktueller Monat in numerischer Form (1 ... 12)
\w	weekday	Aktueller Wochentag 3 Zeichen (Mon, Tue, ...)
\Y	Year	Aktuelles Jahr (4 Ziffern)
\y	year	Aktuelles Jahr (2 Ziffern)
\m	minutes	Minuten der aktuellen Zeit (00-59)
\P	PM	Vormittag/Nachmittag der aktuellen Zeit (AM/PM)
\R	Realtime	Stunde der aktuellen Zeit (00-23)
\r	realtime	Stunde der aktuellen Zeit (00-12)
\s	seconds	Sekunden der aktuellen Zeit (00-59)
\Z		Zeichen "Z" fÃ¼r jedes hier nicht aufgefÃ¼hrte Z.

TIP: Prompt in benutzerspez. Konfig-Datei definieren (Leerzeichen am Ende!):

```
[mysql]
prompt = "\u@\h:\d>\_"
```

TIP: Prompt auf der Kmdo-Zeile definieren:

```
mysql --prompt="\u@\h:\d>\_" ...
```

1r) Grafische MySQL-Programme (GUI)

Neben den bedienungstechnisch relativ einfachen aber trotzdem leistungsfÃ¤higen Kommandozeilenprogrammen sind noch eine Reihe GUI-basierter MySQL-Clients verfÃ¼gbar, die per Maus bedienbar sind. Diese Programme mÃ¼ssen getrennt installiert werden, sie sind nicht Bestandteil des MySQL-Servers oder -Clients:

GUI-Programm	Beschreibung
MySQL Workbench	GUI-Client (frei, inkl. ER-Designer)
MySQL GUI Tools	Zusammenfassung von:
MySQL Administrator	Server-Konfiguration, -Verwaltung, -Wartung
MySQL Query Browser	Grafischer SQL-Client
MySQL Migration Toolkit	Schemata + Daten aus anderen DB importieren
phpMyAdmin	Webbasiertes DB-Management (Webserver nÃ¶tig)
mysql-admin	Webbasiertes DB-Management (Webserver nÃ¶tig)
Adminer	Webbasiertes DB-Management (Webserver nÃ¶tig)
winMySQLadmin	GUI-Client (Windows, STD., MUSS lokal laufen!)
SQL-Front	GUI-Client (Windows, kommerziell)
SQLyog	GUI-Client (Windows, kommerziell)
MySQLManager	GUI-Client (Windows, kommerziell)
Navicat for MySQL	GUI-Client (Windows, kommerziell)
EMS MySQL Manager	GUI-Client (Windows, kommerziell)
MyAdmin	GUI-Client (Windows, kommerziell)
HeidiSQL (MySQL-Front)	GUI-Client (Windows, Linux/MacOS per Wine)
SQuirrel	GUI-Client (Java: Windows, MacOS, Linux)
TOAD	GUI-Client (Windows, kommerziell, viele DB)
Knoda	GUI-Client analog Access (Linux)
Rekall	GUI-Client (Linux)
Ksql	GUI-Client (Linux)
KMySQLadmin	GUI-Client (Linux)
Emma	GUI-Client (Linux)
MySQL Control Center	Weiterentw. von MySQLGUI (mysqlcc, veraltet)
MySQLGUI	Grafischer SQL-Client (veraltet)
mysql-navigator	Grafischer SQL-Client (veraltet)

HINWEIS: Diese Liste erhebt keinen Anspruch auf AktualitÃ¤t und VollstÃ¤ndigkeit.

2) Syntax

2a) Leerraum und Formatierung

Leerraum + ZeilenvorschÃ¼be + GROSS/kleinschreibung der SQL-SchlÃ¼sselworte ist in SQL-Anweisungen (SQL-Statements) irrelevant. Allerdings empfiehlt sich die Ã¼bliche Konvention:

SQL-SchlÃ¼sselworte GROSS schreiben!

Erst ";" oder "\g" (go) bzw. "\G" (ego = Option -E = --vertical = vErtikale Ausgabe) schlieÃM-^_en eine SQL-Anweisung ab. Ausnahme: Interne "mysql"-Client Befehle wie "USE <Db>" (besser nicht daran gewÃ¶hnen ;-).

Es ist sinnvoll, SQL-Anweisungen per EinrÃ¼ckung und ZeilenvorschÃ¼be Ã¼bersichtlich zu formatieren, um eine optimale Lesbarkeit zu erreichen. Die ÃM-^\bersetzungs- und AusfÃ¼hrungsgeschwindigkeit der Anweisung verringert sich dadurch nicht. Insbesondere lange SQL-Anweisungen der ÃM-^\bersicht halber sinnvoll auf mehrere Zeilen umbrechen und einrÃ¼cken, z.B.:

```
UPDATE TABLE pers SET nr = 7, vorname = "Heinz", name = "Bayer";
```

besser so formatieren:

```
  UPDATE TABLE pers
    SET nr      = 7,
        vorname = "Heinz",
        name    = "Bayer";
```

2b) Zeichenketten (Strings) und Quotierung

Alle konstanten Werte auÃM-^_er Zahlen (d.h. Text-, Blob-, Bit-, Set-, Enum-, Datum- und Zeitwerte) sind in "..." oder '...' einzuschlieÃM-^_en (zu QUOTIEREN). Prinzipiell sind aber auch Zahlen so darstellbar, d.h. ALLE Werte dÃ¼rfen quotiert werden (einheitliches Prinzip). Um in "..." das Zeichen " und in '...' das Zeichen ' einzutragen, dieses einfach verdoppeln oder mit "\" quotieren:

```
  SELECT "Hallo", 'Welt';       # --> Hallo Welt
  SELECT Hallo, Welt;           # --> FEHLER!
  SELECT 1, 23, 456;            # --> 1 23 456
  SELECT '1', "23", '456';      # --> 1 23 456
  SELECT 'Don''t';              # --> Don't
  SELECT 'Don\'t';              # --> Don't
  SELECT "Don't";               # --> Don't
  SELECT "Er sagte: ""...""";   # --> Er sagte: "..."
  SELECT "Er sagte: \"...\"";   # --> Er sagte: "..."
  SELECT 'Er sagte: "..."';     # --> Er sagte: "..."
```

HINWEIS: Zwischen Zeichenketten in "..." und '...' gibt es KEINEN Unterschied (wie in anderen Programmiersprachen z.B. bei Escape-Sequenzen \C), allerdings ist nur '...' ANSI-SQL-kompatibel ("..." wird dort verwendet, um Bezeichner mit Sonderzeichen zu quotieren, in MySQL erfolgt dies durch Backquotes `...`).

2c) Escape-Sequenzen

Escape-Sequenzen (Maskierungszeichen) \C zur Darstellung von Sonderzeichen in Zeichenketten der Form "..." und '...':

ESC	Bedeutung
\0	ASCII 0-Zeichen (NUL)
\'	Einfaches Hochkomma in '...' (auch '' in '...')
\"	Doppeltes Hochkomma in "..." (auch "" in "...")
\\	Backslash (\)
\a	Alert (Ton)
\b	Backspace
\f	Formfeed (Seitenvorschub beim Drucker)
\n	Newline (Linefeed)
\r	Carriage Return
\t	Tabulator
\v	Vertikaler Tabulator
\Z	ASCII 26 (Control-Z, unter Windows evtl. notwendig)
\%	%-Zeichen (Zeichen "%" selbst in LIKE)
_	_-Zeichen (Zeichen "_" selbst in LIKE)

2d) Zahlen

Wird eine Zeichenkette in einem Zahlenzusammenhang verwendet, dann wird das max. AnfangsstÃ¼ck (PRÃM-^DFIX), das noch wie eine Zahl aussieht, AUTOMATISCH in diese Zahl KONVERTIERT (und eine Warnung ausgegeben).

ACHTUNG: Sieht eine Zeichenkette Ã¼berhaupt nicht wie eine Zahl aus, dann entspricht sie der Zahl 0 (Null).

```
  SELECT 123 + 0, -5 + 0, 1e+5 + 0;                  # --> 123, -5, 100000
  SELECT "123" + 0, "-5" + 0, "1e+5" + 0;            # --> 123, -5, 100000
  SELECT "123def" + 0, "-5ghi" + 0, "1e+5kjl" + 0;   # --> 123, -5, 100000
  SELECT "def" + 0, 100 + "ghi";                     # --> 0, 100
```

FlieÃM-^_kommazahlen sind (unabhÃ¤ngig von Sprach- und Collation-Einstellungen) immer mit DEZIMALPUNKT zu schreiben (Dezimalkomma und Tausendertrennzeichen sind nicht erlaubt). Die Ausgabe von Dezimalkommas ist per FORMAT(<Zahl>, <Nkst>) erreichbar. Rundung auf feste Anzahl Nachkommastellen oder ganze Zahl ist mit ROUND(<Zahl>, <Nkst>) erreichbar.

```
  SELECT 123.456 + 0, 123,456 + 0;                # --> 123.456, 123, 456
```

```
  SELECT FORMAT(123.456, 2), FORMAT(123.456, 0);   # --> 123.46, 123
  SELECT ROUND(123.456, 2), ROUND(123.456, 0);     # --> 123.46, 123

Hexadezimalzahlen bzw. -bytes folgendermaÃM-^_en schreiben:

  0xaffe...    # Variante A (0X1010 nicht erlaubt!)
  X'affe...'   # Variante B (x"1010" nicht erlaubt!)
  x'affe...'   # Variante C (X"1010" nicht erlaubt!)

BinÃ¤rzahlen und Bitfelder folgendermaÃM-^_en schreiben:

  0b1010...    # Variante A (0B1010 nicht erlaubt!)
  b'1010...'   # Variante B (b"1010" nicht erlaubt!)
  B'1010...'   # Variante C (B"1010" nicht erlaubt!)

Zahlen mit fÃ¼hrender 0 sind Dezimalzahlen (keine Oktaldarstellung):

  SELECT 0123 + 0, 0815 + 0;   # --> 123, 815


2e) Datenbank-Objekte
---------------------

MySQL kennt folgende "Datenbank-Objekte" (oder kurz "Objekte") und
"Syntax-Elemente". Die Platzhalter fÃ¼r BEZEICHNER dieser Objekte stehen in
diesem Skript in SQL-Anweisungen immer in spitzen Klammern <...> und mÃ¼ssen
durch einen konkreten Bezeichner ersetzt werden:

  +--------------------+-------------+-----------------------------------------+
  | Objekt             | Platzhalter | Bedeutung                               |
  +--------------------+-------------+-----------------------------------------+
  | Alias              | <Alias>     | Weiterer Name fÃ¼r ein Datenbank-Objekt |
  | Column             | <Col>       | Tabellenspalte                          |
  | Database           | <Db>        | Datenbank (Schema)                      |
  | Event              | <Event>     | Ereignis (einmalig oder periodisch)     |
  | Index              | <Idx>       | Index einer Tabelle                     |
  | Log File           | <LogFile>   | Logdatei                                |
  | Partition          | <Part>      | Horizontale Zerlegung einer Tabelle     |
  | Prepared Statement | <PrepStmt>  | Vorkompilierte Anweisung                |
  | Privilege          | <Priv>      | Benutzerrecht                           |
  | Statement          | <Stmt>      | SQL-Anweisung                           |
  | Stored Function    | <Func>      | Funktion (RÃ¼ckgabewert)                |
  | Stored Procedure   | <Proc>      | Prozedur (kein RÃ¼ckgabewert)           |
  | Table              | <Tbl>       | Tabelle                                 |
  | Tablespace         | <TblSpace>  | Datei fÃ¼r Tab.daten (analog Partition) |
  | Trigger            | <Trig>      | Trigger einer Tabelle                   |
  | User               | <User>      | Benutzeraccount                         |
  | View               | <View>      | Sicht (vordef. gespeicherte Abfrage)    |
  +--------------------+-------------+-----------------------------------------+
  | Datatype           | <Typ>       | Datentyp (z.B. INT, CHAR, VARCHAR, ...) |
  | Character Set      | <CharSet>   | Zeichensatz (z.B. latin1, utf8, ...)    |
  | Collation          | <Coll>      | Sortierregel (z.B. latin1_german_ic)    |
  | Directory Path     | <DirPath>   | Verz.pfad (absolut)                     |
  | Transcodierung     | <Transcode> | Zeichensatzcodierung                    |
  +--------------------+-------------+-----------------------------------------+


2f) Identifier (Bezeichner)
---------------------------

Bezeichner dÃ¼rfen standardmÃ¤Ã-^_ig aus den Zeichen A-Z, a-z, 0-9, _ und $ bestehen
(d.h. insbesondere KEINE Leerzeichen!), fÃ¼hrende Ziffern sind nicht erlaubt.
Mit der Schreibweise `...` (QUOTIERUNG mit Backquotes, NICHT '...' oder "...")
sind auch beliebige andere Zeichen verwendbar (NICHT empfohlen!).

  SELECT Name                          FROM test;   # OK (STD-Form)
  SELECT `Name`                        FROM test;   # OK (STD-Form)
  SELECT _Name_                        FROM test;   # OK (STD-Form)
  SELECT $Name                         FROM test;   # OK (STD-Form)
  SELECT Und_Ein_Name$                 FROM test;   # OK (STD-Form)
  SELECT `Name mit Leer zeichen`       FROM test;   # OK (Sonderzeichen)
  SELECT `Name mit Sonderzeichen Ã¤Ã¶Ã¼Ã-^_` FROM test;   # OK (Sonderzeichen)

Kollidiert ein BEZEICHNER (Objektname) mit einem SQL-SchlÃ¼sselwort, so kann er
in `...` (Backquotes!) gesetzt dennoch verwendet werden (QUOTIERUNG). Die
normale String-Quotierung mit "..." oder '...' funktioniert hier nicht!

  CREATE TABLE alter (...);                  # FALSCH (Kollision)!
  CREATE TABLE `alter` (...);                # OK (Quotes notwendig)
  SELECT user, host FROM mysql.user;         # OK (STD-Form ohne Quotes)
  SELECT `user`, `host` FROM `mysql`.`user`; # OK (Quotes Ã¼berflÃ¼ssig)
  SELECT `user`, `host` FROM `mysql.user`;   # FALSCH (Namen einzeln quot.)!
```

```
  SELECT "user", "host" FROM "mysql"."user";   # FALSCH (String)!
  SELECT 'user', 'host' FROM 'mysql'.'user';   # FALSCH (String)!

HINWEIS: In generierten SQL-Anweisungen (z.B. per "mysqldump") werden ALLE
BEZEICHNER grundsÃ¤tzlich prophylaktisch in `...` gesetzt (auch wenn das gar
nicht notwendig wÃ¤re).

Maximal erlaubte Bezeichner-LÃ¤nge von Datenbank-Objekten und Namen:

  +-----+------------+
  | Max | Bezeichner |
  +-----+------------+
  |  64 | Datenbank  |
  |  64 | Tabelle    |
  |  64 | Spalte     |
  |  64 | Routine    |
  | 255 | Alias      |
  +-----+------------+
  |  60 | Host       |
  |  16 | Benutzer   |
  |  41 | Passwort   |
  +-----+------------+

Zugriff auf die Objekte Datenbank, Tabelle, Spalte und Stored Procedure erfolgt
durch "relative" oder "vollqualifizierte" (full qualified) Bezeichner (im
"mysql"-Client per <TAB>-Taste = TAB-Completion automatisch vervollstÃ¤ndigbar):

  +--------------------+-----------------------------------------------------+
  | Bezeichner         | Bedeutung                                           |
  +--------------------+-----------------------------------------------------+
  | <Db>.<Tbl>         | A) VollstÃ¤ndiger Pfad (immer OK!)                   |
  | <Db>.<Tbl>.<Col>   |    Analog                                           |
  | <Db>.<Tbl>.*       |    Analog (ALLE Spalten der Tabelle)                |
  +--------------------+-----------------------------------------------------+
  | <Tbl>              | B) Relativ zu Default/Standard-Datenbank (USE <Db>) |
  | <Tbl>.<Col>        |    Analog                                           |
  | <Tbl>.*            |    Analog (ALLE Spalten der Tabelle)                |
  +--------------------+-----------------------------------------------------+
  | <Col>              | C) Analog B) in INSERT/SELECT/UPDATE/DELETE         |
  |                    |    Relativ zu EINER Tabelle immer OK!               |
  |                    |    Bei MEHREREN verknÃ¼pften Tabelle nur bei         |
  |                    |    pro Tabelle eindeutigen Spaltennamen OK.         |
  |                    |    (mit eindeutigen Spalten-Aliasen auch OK)        |
  | *                  |    Analog (ALLE Spalten einer Tabelle)              |
  +--------------------+-----------------------------------------------------+
  | <Db>.<Proc>        | D) VollstÃ¤ndiger Pfad (immer OK!)                   |
  | <Proc>             | E) Relativ zu Default/Standard-Datenbank (USE <Db>) |
  +--------------------+-----------------------------------------------------+

  SELECT test.pers.name FROM test.pers;      # A) OK (Datenbank "test")
  USE test;                                  #    Default-DB "test" auswÃ¤hlen
  SELECT pers.name FROM pers;                # B) OK (Datenbank "test")
  SELECT name FROM pers;                     # C) OK bei EINER Tabelle (DB "test")
  SELECT name FROM pers, age;                # C) Problem bei Sp.namen-Kollision
  SELECT pers.name, age.name FROM pers, age; # C) OK (auch bei Sp.Kollision)
  SELECT p.name, a.name FROM pers p, age a;  # C) OK (Aliase auch bei Sp.Koll.)

ACHTUNG: Wird im Fall C) zu einer beteiligten Tabelle nachtrÃ¤glich eine Spalte
hinzugefÃ¼gt, die GLEICHNAMIG zu einer Spalte einer anderen beteiligten Tabelle
ist, fÃ¼hrt das in bereits vorhandenen SQL-Anweisungen zu Syntaxfehlern, wenn
darin auf diese Spalte zugegriffen wird. Daher am besten mit Variante B)
arbeiten und zur AbkÃ¼rzung ALIASE einfÃ¼hren.

HINWEIS: Die Bestandteile eines relativen oder vollqualifizierten Bezeichners
mÃ¼ssen GETRENNT mit `...` quotiert werden:

  SELECT DISTINCT `mysql`.`user`.`host` FROM `mysql`.`user`;   # OK
  SELECT DISTINCT `mysql.user.host`     FROM `mysql.user`;     # FALSCH!
  SELECT DISTINCT mysql.user.host       FROM `mysql`.`user`;   # OK

Analog MÃM-^\SSEN die beiden Bestandteile "User" und "Host" eines Benutzernamens
"User@Host" GETRENNT quotiert werden (allerdings mit "..." oder '...', da es
sich nicht um Bezeichner, sondern um externe Namen handelt):

  CREATE USER "hans"@"localhost";   # OK
  CREATE USER 'hans'@'localhost';   # OK
  CREATE USER `hans`@`localhost`;   # FALSCH
  CREATE USER "hans@localhost";     # FALSCH!
  CREATE USER 'hans@localhost';     # FALSCH!
  CREATE USER `hans@localhost`;     # FALSCH!
  CREATE USER hans@localhost;       # FALSCH!
```

2g) GROSS/kleinschreibung

Beim Vergleichen und beim Sortieren von Daten in Tabellen ignoriert MySQL normalerweise die GROSS/kleinschreibung auÃM-^_er in folgenden FÃ¤llen:

* BINARY vor einem String in den Vergleichen = != <> <=> < <= > >=, BETWEEN, IN, LIKE, RLIKE/REGEX und in ORDER BY erzwingt die Beachtung der GROSS/kleinschreibung bei Vergleichen / Sortieren.

* Die Datentypen BINARY, VARBINARY, TINYBLOB, BLOB, MEDIUMBLOB und LONGBLOB erzwingen Beachtung der GROSS/kleinschreibung beim Vergleichen / Sortieren.

Die GROSS/kleinschreibung in SQL-Anweisungen ist meist egal AUSSER bei den Bezeichnern von Datenbanken, Tabellen, Views und Logfilegroups unter einem Betriebssystem mit case-sensitivem Dateisystem (da als Verz./Datei abgelegt). Die Namen von Triggern, Labeln (Marken) und Benutzern sowie PasswÃ¶rter sind immer "case-sensitive". Es gibt daher gewisse KONVENTIONEN bgzl. GROSS/kleinschreibung, an die man sich halten sollte.

Sprachelement	GROSS/kleinschreibung relevant	Konvention
Datenbank-Bezeichner	JA (Linux), NEIN (Windows)	klein
Tabellen-Bezeichner	JA (Linux), NEIN (Windows)	klein
View-Bezeichner	JA (Linux), NEIN (Windows)	klein
Dateiname/pfad	JA (Linux), NEIN (Windows)	klein
Logfilegroup-Name	JA (Linux), NEIN (Windows)	klein
Tablespace-Name	JA (Linux), NEIN (Windows)	klein
Host-Name	NEIN	klein
Benutzer-Name	JA	klein
Passwort	JA	---
Trigger-Bezeichner	JA	klein
Label (Marke)	JA	GROSS
SQL-SchlÃ¼sselwort	NEIN	GROSS
SQL-Funktions-Bezeichner	NEIN	GROSS
Spalten-Bezeichner	NEIN	klein
Index-Bezeichner	NEIN	klein
Variablen-Bezeichner	NEIN (ab MY!5.0), JA (bis MY!4.1)	klein
Prepared-Stmt-Bezeichner	NEIN	klein
Partitions-Bezeichner	NEIN	klein
Prozedur-Bezeichner	NEIN	klein
Funktions-Bezeichner	NEIN	klein
Event-Bezeichner	NEIN	klein
String-Vergleich	JA/NEIN (abhÃ¤ngig von Collation)	---
RegEx-Vergleich	JA/NEIN (abhÃ¤ngig von Collation)	---
Tabellen-Alias	NEIN	klein
Spalten-Alias	NEIN	klein

TIP: SQL-SchlÃ¼sselworte aus GrÃ¼nden der besseren Lesbarkeit und der einfacheren SuchmÃ¶glichkeit IMMER GROSS schreiben (Konvention):

```
select * from pers order by nr asc;     # Schlecht lesbar
SELECT * FROM pers ORDER BY nr ASC;     # Gut lesbar
```

In diesem Skript werden daher alle SQL-SchlÃ¼sselworte GROSS, hingegen alle Datenbank-, Tabellen-, Spalten- und sonstige Bezeichner klein geschrieben.

2h) Kommentare

Es gibt folgende MÃ¶glichkeiten, Kommentare in den SQL-Quelltext einzubauen:

Syntax	Typ	Beschreibung
-- ...	ANSI-SQL	Bis Zeilenende (Leerz. nach "--" nÃ¶tig!, MY!)
#...	Shell	Bis Zeilenende (MY!)
/*...*/	C	Beliebig viele Zeilen
/*! ...*/	C	Inhalt NUR von MySQL ausgefÃ¼hrt (MY!)
/*!NXXYY ...*/	C	Inhalt ab MySQL-Version N.XX.YY ausgefÃ¼hrt (MY!)
--...	ANSI-SQL	NICHT mÃ¶glich (Leerz. nach "--" nÃ¶tig!, MY!)

```
  | //...          | C++      | NICHT mÃ¶glich!                                    |
  +----------------+----------+---------------------------------------------------+

Die ersten 3 Kommentartypen werden vom "mysql"-Client VOR dem Transfer einer
SQL-Anweisung zum MySQL-Server ENTFERNT. Sollen sie doch mit Ã¼bertragen und
somit in den Logdateien abgelegt werden, dann den Option "--comments" beim
Aufruf des "mysql"-Client angeben (STD: --skip-comments). Kommentare der Form
/*!...*/ werden grundsÃ¤tzlich an den MySQL-Server Ã¼bertragen und stehen somit
immer in den Logdateien.

Beispiele:

  SHOW TABLES; # Kommentar bis zum Zeilenende      # OK
  SHOW TABLES; #Kommentar bis zum Zeilenende       # OK
  SHOW TABLES; -- Kommentar bis zum Zeilenende     # OK
  SHOW TABLES; --Kommentar bis zum Zeilenende      # FEHLER (Leerz. fehlt!)
  SHOW TABLES; // Dies ist KEIN Kommentar!         # FEHLER (nicht erlaubt)
  SHOW /* mehrzeiliger                            # OK (mehrzeiliger Kommentar)
          Kommentar                               # OK (mehrzeiliger Kommentar)
       */ TABLES;                                 # OK (mehrzeiliger Kommentar)
  SELECT * FROM user /*! STRAIGHT_JOIN */ host     # Von MySQL ausgefÃ¼hrt,
    ON user.Host = host.Host;                      # von anderen DB nicht
  CREATE /*!32302 TEMPORARY */ TABLE t1 (a INT);   # Ab MY!3.23.02 ausgefÃ¼hrt
  CREATE /*!99999 Kommentar */ TABLE t2 (a INT);   # NIE ausgef. aber erhalten


3) Typische MySQL-Datenbankbefehle (Tutorial)
---------------------------------------------

Dieser (lange) Abschnitt bietet eine EinfÃ¼hrung in die wichtigsten SQL-
Kommandos von MySQL anhand von Beispielen. Verwendet werden darin die beiden
Datenbanken "first" und "test" sowie mehrere Tabellen (z.B. "pers" und "age").
Eine ausfÃ¼hrliche Beschreibung dieser und anderer SQL-Anweisungen ist in den
folgenden Kapiteln und in --> mysql-admin-HOWTO.txt zu finden.

HINWEIS: SQL-Anweisungen dÃ¼rfen auf mehrere Zeilen umbrochen werden und sind
immer mit ";" (oder "\g" bzw. "\G") abzuschlieÃM-^_en.

TIP: Zu jeder neuen Datenbank einen EIGENEN MySQL-Benutzer (hier: "tom")
anlegen, ihm ein Passwort (hier: "geheim") sowie geeignete Zugriffsrechte
(hier: GRANT ALL PRIVILEGES = sÃ¤mtliche Rechte) geben. Er darf dann nur mit
dieser Datenbank arbeiten (sie muss dazu noch gar nicht existieren!):

  CREATE USER "tom"@"localhost";   # --> Leeres Passwort, keine Zugriffsrechte
                                   #
  GRANT ALL PRIVILEGES             # Alle Zugriffsrechte...
    ON first.*                     # ...auf alle Objekte der Datenbank "first"
      TO "tom"@"localhost"         # ...fÃ¼r Benutzer "tom@localhost"
        IDENTIFIED BY "geheim"     # ...mit Passwort "geheim"
          WITH GRANT OPTION;       # ...mit Rechteweitergabe (optional)

Passwort nachtrÃ¤glich Ã¤ndern:

  SET PASSWORD FOR "tom"@"localhost" = PASSWORD("geheim");

HINWEIS: Seit MY!5.6 ist wg. dem Plugin "validate_password" standardmÃ¤ÃM-^_ig ein
Passwort mit folgenden Eigenschaften nÃ¶tig:

  * MindestlÃ¤nge 8 Zeichen
  * Mindestens ein Grossbuchstabe
  * Mindestens ein Kleinbuchstabe
  * Mindestens eine Ziffer
  * Mindestens ein Sonderzeichen (auÃM-^_er Buchstabe und Ziffer)

TIP: Vorher "CREATE USER" ist nicht unbedingt notwendig, schadet aber auch
nicht, da das GRANT-Statement den Benutzer automatisch anlegt, sofern er noch
nicht existiert (solange SQL-Modus "NO_AUTO_CREATE_USER" nicht gesetzt ist).

TIP: Username == Datenbankname != Tabellenname != Spaltenname wÃ¤hlen, sonst
besteht Verwechslungsgefahr (oder mit PrÃ¤fix "u_" = User, "d_" = Database, "t_"
= Table, "c_" = Column arbeiten).

HINWEIS: Manche DB-Systeme legen zu jeder Datenbank automatisch einen Benutzer
gleichen Namens an, der alle Rechte bzgl. dieser Datenbank zugewiesen bekommt
(PostgreSQL, Oracle). MySQL kennt dieses Standard-Verhalten nicht.

Datenbank (Schema) "first" anlegen (meist als MySQL-Administrator "root"):

  CREATE DATABASE first;                # Fehler falls schon existent
  CREATE SCHEMA   first;                #   (analog)
  CREATE DATABASE IF NOT EXISTS first;  # Nur falls noch nicht existent (MY!)
  CREATE SCHEMA   IF NOT EXISTS first;  #   (analog MY!)
```

```
Datenbank (Schema) umbenennen ("root" oder Besitzer, nur von MY!5.1.7 bis
MY!5.1.23 verfÃ¼gbar da zu gefÃ¤hrlich!):

  RENAME DATABASE first TO new;   # "first" --> "new" umbenennen
  RENAME SCHEMA   new TO first;   #   (analog umgekehrt)

Benutzer "tom" nimmt per "mysql"-Client Verbindung mit dem MySQL-Server (lokal)
auf und wÃ¤hlt beim Verbindungsaufbau "first" als Default/Standard-Datenbank aus:

  mysql -utom -p                   # Password interaktiv abgefragt (sicher!)
  mysql -utom -pgeheim             # Keine Default-Datenbank --> "USE first"
  mysql -utom -pgeheim -Dfirst     # Default-Datenbank --> "first"
  mysql -utom -pgeheim first       # Default-Datenbank --> "first"
  mysql -utom -pgeheim -hglasgow   # MySQL-Server auf Rechner "glasgow"

Abfrage der Verbindungsdaten (Klammern "()" notwendig, da Funktionsaufrufe):

  SELECT DATABASE();                       # Standard-Datenbank (oder "NONE")
  SELECT SCHEMA();                         #   (analog)
  SELECT USER();                           # Benutzer-Anmeldung <User>@<Host> !=
  SELECT CURRENT_USER();                   #   Angemeld. Benutzer (evtl. anonym)
  SELECT SESSION_USER();                   #   (analog)
  SELECT SYSTEM_USER();                    #   (analog)
  SELECT CONNECTION_ID();                  # Sitzungs-ID (Thread, Prozess)
  SELECT VERSION();                        # MySQL-Server Version

Abfrage des Server-Status (Database, User, Server Version, Connection-ID,
Character set, Protocol Version, Connection type, UNIX Socket, Uptime, ...):

  STATUS   # Nur im "mysql"-Kommandozeilenclient vorhanden
  \s       # (analog)

Vorhandene Datenbanken auflisten:

  SHOW DATABASES;              # Alle Datenbanknamen auflisten
  SHOW SCHEMAS;                #   (analog)
  SHOW DATABASES LIKE "my%";   # Nur DB-Namen mit "my" am Anfang
  SHOW SCHEMAS   LIKE "my%";   #   (analog)

Als Default/Standard-Datenbank (Schema) "first" auswÃ¤hlen. Da es sich um einen
internen "mysql"-Clientbefehl handelt, darf der abschlieÃM-^_ende ";" weggelassen
werden (besser gar nicht erst daran gewÃ¶hnen!). Alle Bezeichner (Objektnamen)
ohne weitere Qualifizierung beziehen sich dann auf diese Datenbank (dieses
Schema) --> 2e) Datenbank-Objekte

  USE first    # Variante A (ohne ";")
  USE first;   # Variante B (mit ";")

Tabellen einer Datenbank auflisten:

  SHOW TABLES;                           # Alle aus Standard-DB "first"
  SHOW TABLES FROM mysql;                # Alle aus Verwaltungs-DB "mysql"
  SHOW TABLES LIKE "a%";                 # Mit Name "a..." aus Standard-DB "first"
  SHOW TABLES FROM mysql LIKE "%a%";     # Mit Name "...a..." aus DB "mysql"

Tabelle "pers" (bedingt wenn sie noch nicht existiert) in Default-DB anlegen
(3 Spalten, im 2. Fall mit einer anderen Engine statt Standard-Engine "InnoDB",
Spalten ohne/mit Defaultwert und mit NULL erlaubt/nicht erlaubt):

  CREATE TABLE pers (                 # STD-Engine = InnoDB
    nr      INT          NOT NULL,    # Komma       (     NULL verboten)
    vorname VARCHAR(30),              # Komma       (STD: NULL erlaubt)
    name    VARCHAR(30)               # KEIN Komma! (STD: NULL erlaubt)
  ) COMMENT = "Personen";             # Kommentar zur Tabelle

  CREATE TABLE IF NOT EXISTS pers (   # Kein Fehler falls schon existent
    nr      INT         NOT NULL,     # NULL verboten
    vorname VARCHAR(30) DEFAULT "",   # Defaultwert leere Zeichenkette
    name    VARCHAR(30) NULL          # NULL explizit erlaubt (sowieso STD)
  ) ENGINE = "MyISAM";                # Engine MyISAM (alt: TYPE statt ENGINE)

HINWEIS: Jedes DB-Objekt hat automatisch einen "Besitzer", nÃ¤mlich den User,
der es anlegt. Dieser User hat automatisch alle Rechte daran und kann es z.B.
umbenennen und auch wieder lÃ¶schen.

HINWEIS: Die Spalten-REIHENFOLGE einer Tabelle ist nur relevant bei:
  * Abfragen mit "*"
  * Speichern mit INSERT INTO <Tbl> VALUES (...) ohne Liste von Spaltennamen
    (irrelevant bei INSERT INTO <Tbl> (<Col1>, ...) VALUES (...))
  * Mehr als einer TIMESTAMP-Spalten in einer Tabelle
    (ERSTE TIMESTAMP-Spalte wird bei jeder Datensatz-ÃM-^Dnderung aktualisiert).
```

```
TemporÃ¤re Tabelle anlegen:

  CREATE TEMPORARY TABLE pers (      #
    nr      INT          DEFAULT 0   #
    vorname VARCHAR(30),             #
    name    VARCHAR(30)              #
  ) CHARACTER SET latin1;            # Standard-Zeichensatz fÃ¼r Textspalten

HINWEIS: TemporÃ¤re Tabellen
  * DÃ¼rfen genauso heiÃM-^_en wie echte Tabellen.
  * ÃM-^\BERDECKEN evtl. vorhandene gleichnamige echte Tabelle,
    bis sie wieder gelÃ¶scht werden.
  * Sind SITZUNGSBEZOGEN (d.h. pro Sitzung gleicher Name verwendbar,
    ohne dass es zu Kollisionen kommt).
  * Werden am Ende einer Sitzung automatisch entfernt,
    sofern nicht vorher manuell gelÃ¶scht (per DROP TEMPORARY <Tab>).
  --> mysql-admin-HOWTO.txt --> 8a) TemporÃ¤re Tabellen

Tabellenstruktur anzeigen (Spalten + Datentypen):

  DESCRIBE pers;              # Variante A
  DESCRIBE pers name;         # Variante B (nur Spalte "name")
  DESC pers;                  # Variante C
  DESC pers name;             # Variante D (nur Spalte "name")
  EXPLAIN pers;               # Variante E
  SHOW COLUMNS FROM pers;     # Variante F
  SHOW TABLE pers;            # FEHLER: Gibt es nicht!

VollstÃ¤ndige Tabellen-Definition anzeigen (mit Engine und allen Einstellungen):

  SHOW CREATE TABLE pers;     # Ausgabe horizontal
  SHOW CREATE TABLE pers\g    # Ausgabe horizontal (go)
  SHOW CREATE TABLE pers\G    # Ausgabe vertikal (ego, Option -E)
  SHOW TABLE pers;            # FEHLER: Gibt es nicht!

Alle DatensÃ¤tze einer Tabelle anzeigen (aktuell noch leer):

  SELECT * FROM pers;                    # Alle Spalten in Def.reihenfolge
  SELECT nr, vorname, name FROM pers;    #   (analog)
  SELECT vorname, nr, name FROM pers;    # Alle Spalten in anderer Reihenfolge
  SELECT vorname, name FROM pers;        # Nur ausgewÃ¤hlte Spalten
  SELECT nr FROM pers;                   # Nur ausgewÃ¤hlte Spalte

DatensÃ¤tze in Tabelle "pers" einfÃ¼gen (nicht angegebene Spalten werden mit
ihrem Defaultwert oder NULL gemÃ¤ÃM-^_ Angabe in Tabellen-Definition belegt):

  INSERT INTO pers (nr, vorname, name)
    VALUES (1, "Thomas", "Birnthaler");
  INSERT INTO pers (nr, vorname, name)
    VALUES (2, "Markus", "Mueller");
  INSERT INTO pers (nr, vorname, name)
    VALUES (8, "Andrea", "Bayer");

Bei vollstÃ¤ndigen DatensÃ¤tzen kann auf die Angabe der Spaltennamen verzichtet
werden. Dann mÃ¼ssen aber REIHENFOLGE und TYP der einzufÃ¼genden Daten exakt mit
der Spaltenreihenfolge in der Tabellen-Definition Ã¼bereinstimmen (Vorsicht!):

  INSERT INTO pers VALUES (7, "Heinz", "Bayer");
  INSERT INTO pers VALUES (NULL, "Hans", "Dampf");
  INSERT INTO pers VALUES (NULL, NULL, "Unbekannt");

Andere EinfÃ¼ge-Syntax:

  INSERT INTO pers
    SET nr      = 7,
        vorname = "Heinz",
        name    = "Bayer";

EinfÃ¼gen von mehr als einem Datensatz gleichzeitig (sehr effizient, MY!):
ACHTUNG: Server-Variable "max_allowed_packet" muss gross genug sein, um
Statement vollstÃ¤ndig zum Server Ã¼bertragen zu kÃ¶nnen):

  INSERT INTO pers (nr, vorname, name)        # Nur 1x Spaltennamen!
    VALUES (1,    "Thomas",  "Birnthaler"),   # Nur 1x VALUES!
           (2,    "Markus",  "Mueller"),      # "," zw. DatensÃ¤tzen!
           (8,    "Andrea",  "Bayer"),        #
           (9,    "Richard", "Seiler"),       #
           (7,    "Heinz",   "Bayer"),        #
           (5,    "Hans",    "Dampf"),        #
           (3,    NULL,      "Unbekannt");    # Anweisungsende
```

```
Laden der Daten aus einer externen CSV-Datei (Comma Separated Values, enthÃ¤lt
pro Datensatz eine Zeile abgeschlossen durch <NL> (Unix), <CR> <NL> (Windows)
oder <CR> (MacOS). Die Datensatz-Elemente mÃ¼ssen in SPALTENREIHENFOLGE
im Datensatz stehen und durch ein Tabulatorzeichen <TAB> getrennt sein.
Bei CSV-Dateien notwendige 1. Zeile mit Spaltennamen kann ignoriert werden.
Die Daten kÃ¶nnen auf Client ("LOCAL") oder Server (kein "LOCAL") liegen.

  LOAD DATA LOCAL INFILE "/path/to/pdata.txt" INTO TABLE pers;    # Linux A
  LOAD DATA LOCAL INFILE "/path/to/pdata.txt" INTO TABLE pers     # Linux B
    LINES TERMINATED BY "\n"                                      #
    IGNORE 1 LINES;                                               # 1.Zl ign.
  LOAD DATA LOCAL INFILE "D:/path/to/pdata.txt" INTO TABLE pers   # Windows
    LINES TERMINATED BY "\r\n"                                    #
    IGNORE 1 LINES;                                               # 1.Zl ign.
  LOAD DATA INFILE "D:\\path\\to\\pdata.txt" INTO TABLE pers      # Windows
    LINES TERMINATED BY "\r\n"                                    #
    IGNORE 1 LINES;                                               # 1.Zl ign.
  LOAD DATA INFILE "/path/to/pdata.txt" INTO TABLE pers           # MacOS
    FIELDS TERMINATED BY "\t"                                     # Tabs
    LINES TERMINATED BY "\r"                                      #
    IGNORE 1 LINES                                                # 1.Zl ign.
    (nr, vorname)        # Spalten in Reihenfolge laden (sonst alle)
    SET name = '';       # Spalten fix fÃ¼llen

ACHTUNG: Voraussetzungen:

* Damit LOAD LOCAL INFILE funktioniert, muss die Server-Variable "local_infile"
  bei Server UND Client auf "ON" gesetzt sein.
* Damit LOAD DATA INFILE ... (sowie LOAD_FILE() und SELECT ... INTO OUTFILE ...)
  funktionieren, muss das Recht "FILE" fÃ¼r den Benutzer gesetzt sein.
* Die Server-Variable "secure_file_priv" legt evtl. Verz. fest, in dem export.
  und importierten Dateien liegen MÃM-^\SSEN (z.B. "/var/lib/mysql-files").

Inhalt der Datei "pdata.txt" (NULL-Wert durch "\N" codiert!):

  +-------------------------------+   #
  |Nr<TAB>Vorname<TAB>Name<NL>    |   # 1. Zeile mit Spaltennamen --> Ignorieren
  +-------------------------------+   #
  |1<TAB>Thomas<TAB>Birnthaler<NL>|   # 1. Datensatz
  |2<TAB>Markus<TAB>Mueller<NL>   |   # 2. Datensatz
  |8<TAB>Andrea<TAB>Bayer<NL>     |   # 3. Datensatz
  |9<TAB>Richard<TAB>Seiler<NL>   |   # 4. Datensatz
  |7<TAB>Heinz<TAB>Bayer<NL>      |   # 5. Datensatz
  |\N<TAB>Hans<TAB>Dampf<NL>      |   # 6. Datensatz (\N = NULL-Wert)
  |\N<TAB>\N<TAB>Unbekannt<NL>    |   # 7. Datensatz (\N = NULL-Wert)
  +-------------------------------+   #

Um eine CSV-Datei zu erzeugen, folgende SELECT-Syntax verwenden
(kein LOCAL, d.h. Dateiablage nur auf Server mÃ¶glich, NICHT auf Client!):

  SELECT ...
    INTO OUTFILE <FilePath> [<SaveOpt>]
      FROM ...;

Als <SaveOpt> sind mÃ¶glich (in dieser Reihenfolge anzugeben!, "FIELDS" und
LINES nur 1x!, alle Elemente kÃ¶nnen fehlen, ebenso die Spalten)

  FIELDS TERMINATED BY <Char>              # STD: "\t" = Tabulator
         [OPTIONALLY] ENCLOSED BY <Char>   # STD: ""   = keines
         ESCAPED BY <Char>                 # STD: "\\" = \
  LINES STARTING BY <String>               # STD: ""   = leer
        TERMINATED BY <String>             # STD: "\n" = \n

Kopieren einer Tabelle: Struktur und/oder Indices und/oder DatensÃ¤tze.
ACHTUNG: Primary Key und Indices werden NICHT mitkopiert (ausser bei LIKE!):

                                                  # Struktur  Indices  Daten
  CREATE TABLE copy    SELECT * FROM pers;        #    OK       Nein    OK
  CREATE TABLE copy AS SELECT * FROM pers;        #    OK       Nein    OK
  CREATE TABLE copy    SELECT * FROM pers WHERE 0;#    OK       Nein   Nein (TRICK!)
  CREATE TABLE copy AS SELECT * FROM pers WHERE 0;#    OK       Nein   Nein
  CREATE TABLE copy    LIKE pers;                 #    OK        OK    Nein (MY!5.0)
  INSERT INTO  copy    SELECT * FROM pers;        #   Nein      Nein    OK

Um eine Tabelle VOLLSTÃM-^DNDIG zu kopieren (Struktur + Indices + DatensÃ¤tze),
die beiden folgenden Anweisungen kombinieren:
                                                  # Struktur  Indices  Daten
  CREATE TABLE copy LIKE pers;                    #    OK        OK    Nein (MY!5.0)
  INSERT INTO  copy SELECT * FROM pers;           #   Nein      Nein    OK

Kopieren bestimmter Spalten der DatensÃ¤tze aus einer Tabelle in eine andere
Tabelle (Anzahl Spalten von Quelle und Ziel MUSS Ã¼bereinstimmen!):
```

```
  INSERT INTO copy (nr, vorname, name)   # "vorname" und "name" vertauschen!
    SELECT nr, name, vorname             #
      FROM pers;                         #

Tabelleninhalt abfragen (ALLE DatensÃ¤tze mit ALLEN Spalten in ihrer
Definitionsreihenfolge bzw. falls Primary Key vorhanden sortiert nach diesem):

  SELECT * FROM copy;       #
  SELECT ALL * FROM copy;   # (analog, ALL ist STD)

Tabelleninhalt abfragen (ALLE DatensÃ¤tze mit bestimmten Spalten in der
angegebenen Reihenfolge):

  SELECT name, vorname FROM pers;

Tabelleninhalt abfragen (IDENTISCHE DatensÃ¤tze nur 1x anzeigen, analog GROUP BY
Ã¼ber gewÃ¤hlte Spalten, DISTINCT und DISTINCTROW verhalten sich identisch):

  SELECT DISTINCT    *             FROM copy;   # Alle Spalten
  SELECT DISTINCTROW *             FROM copy;   #   (analog)
  SELECT DISTINCT    name          FROM copy;   # Eine Spalte
  SELECT DISTINCT    name, vorname FROM copy;   # Zwei Spalten

Spezifische Spaltentexte (Aliase) fÃ¼r Ausgabe als Ã¼berschrift festlegen,
Standard ist der Spaltenname laut Tabellen-Definition (bei Spalten-Aliasen ist
GROSS/kleinschreibung nicht relevant, "AS" ist auch weglassbar):

  SELECT vorname "Vorname", name "Familienname" FROM pers;   # Ohne AS
  SELECT vorname AS "Vorname",                               # Mit AS
         name    AS "Familienname"                           # (besser lesbar)
    FROM pers;

Konstanten und Ergebnis einer Rechnung mit spezifischer Ã¼berschrift ausgeben
(STD: Konstante, Ergebnis oder Formel auch als Ã¼berschrift verwendet):

  SELECT 2010          AS "Jahr",
         "Ergebnis = " AS "Text",
         5 * 4 - 3 / 2 AS "Formel";

Beispiel:

  CREATE TABLE IF NOT EXISTS artikel (
    nr    INT,
    name  VARCHAR(50),
    preis NUMERIC(10,2),
    anz   INT
  );

  INSERT INTO artikel VALUES (1, "Locher", 4.95, 18),
                             (2, "Hefter", 9.90, 12),
                             (3, "Papier", 5.49, 123);

  SELECT name                    AS `Artikel`,      # Warum OK?
         preis                   AS "Nettopreis",   #
         round(preis * 0.19, 2) AS 'MwSt',         # Rundung auf 2 NkSt
         round(preis * 1.19, 2) AS "Bruttopreis"   # Rundung auf 2 NkSt
    FROM artikel;

Tabelleninhalt nach Spalte "name" AUFsteigend sortiert abfragen (ohne
GROSS/kleinschreibung zu berÃ¼cksichtigen; STD: ASC = ascending = aufsteigend,
der Deutlichkeit halber TROTZDEM hinschreiben!). Neben SpaltenNAMEN sind auch
SpaltenNUMMERN in ORDER BY erlaubt. Diese Nummern beziehen sich auf die
Reihenfolge der selektierten Spalten (bei "*" ist das die Reihenfolge der
Spalten in der Tabellen-Definition):

  SELECT * FROM pers ORDER BY name;              # GROSS/kleinschr. ignorieren
  SELECT * FROM pers ORDER BY name ASC;          #   (analog, besser!)
  SELECT * FROM pers ORDER BY BINARY name ASC;   # GROSS/kleinschr. beachten
  SELECT * FROM pers ORDER BY 3, 2;              # Nach Spalte 3+2 sortieren
  SELECT * FROM pers ORDER BY name, vorname;     #   (analog, Sp.name statt nr)

Tabelleninhalt nach Spalte "nr" ABsteigend sortiert abfragen
(DESC = descending, BINARY = GROSS/kleinschreibung beachten):

  SELECT * FROM pers ORDER BY nr DESC;           # GROSS/kleinschr. ignorieren
  SELECT * FROM pers ORDER BY BINARY nr DESC;    # GROSS/kleinschr. beachten

Sortierung des Tabelleninhalts nach Spalte "name" AUFsteigend, bei gleichem
Nachnamen nach Spalte "vorname" AUFsteigend und bei gleichem Vor+Nachnamen
nach Spalte "nr" ABsteigend (ohne GROSS/kleinschreibung zu berÃ¼cksichtigen):
```

```
  SELECT * FROM pers
    ORDER BY name     ASC,    # AUFsteigend (ASC besser immer hinschreiben!)
             vorname  ASC,    # AUFsteigend
             nr       DESC;   # ABsteigend

Teil der DatensÃ¤tze einer Tabelle Ã¼ber eine Bedingung abfragen
  * Sogenannte "WHERE-Klausel/clause"
  * Textvergleiche ignorieren GROSS/kleinschreibung (auÃM-^_er BINARY vor Operator)
  * LIKE        = Unscharfe Suche mit Wildcards % (beliebig viele belieb. Z.)
                                                 _ (1 beliebiges Zeichen)
  * REGEX/RLIKE = Unscharfe Suche mit RegulÃ¤ren AusdrÃ¼cken (s.u.)
  * i/-         = Index nutzbar/NICHT nutzbar (beschleunigt evtl. Suche)

  SELECT nr                                           # i = Index nutzbar
    FROM pers                                         # - = Index NICHT nutzbar
      WHERE name > "Andrea" AND name < "Thomas";      # i RÃ¤nder weglassen
      WHERE name >= "Andrea" AND name <= "Thomas";    # i RÃ¤nder einschlieÃM-^_en (A)
  ... WHERE name BETWEEN "Andrea" AND "Thomas";       # i RÃ¤nder einschlieÃM-^_en (B)
  ... WHERE name NOT BETWEEN "Andrea" AND "Thomas";   # i RÃ¤nder weglassen
  ... WHERE name IS NULL;                             # i Spalte LEER
  ... WHERE name IS NOT NULL;                         # i Spalte NICHT LEER
  ... WHERE name = NULL;                              # - FALSCH! immer NULL --> FALSE!
  ... WHERE name <> NULL;                             # - FALSCH! immer NULL --> FALSE!
  ... WHERE name = "Andrea" OR name = "Thomas";       # i Werteliste (A)
  ... WHERE name IN ("Andrea", "Thomas");             # i Werteliste (B)
  ... WHERE name NOT IN ("Andrea", "Thomas");         # i Ausschlussliste
  ... WHERE name LIKE "a%";                           # i "a" am Anfang
  ... WHERE BINARY name LIKE "%a%";                   # - "a" irgendwo
  ... WHERE name NOT LIKE "%a";                       # - KEIN "a" am Ende
  ... WHERE name NOT LIKE "__a%";                     # - KEIN "a" an 3. Z.pos.
  ... WHERE name REGEXP "abc";                        # - "abc" enthalten
  ... WHERE name RLIKE "abc";                         # - "abc" enthalten
  ... WHERE name NOT REGEXP "abc";                    # - KEIN "abc" enthalten
  ... WHERE name NOT RLIKE "abc";                     # - KEIN "abc" enthalten

Weitere Tabelle anlegen und fÃ¼llen ("age" statt "alter", da "ALTER" eine
SQL-Anweisung ist, durch EinschlieÃM-^_en in Backquotes `alter` wÃ¤re "alter" doch
als Tabellenname mÃ¶glich, "..." oder '...' funktioniert nicht!):

  CREATE TABLE age (
    nr            INT NOT NULL,
    geburtsdatum  DATE,
    jahre         INT,
    geschlecht    CHAR(1)
  );                                                  # Mehrere SÃ¤tze einfÃ¼gen
  INSERT INTO age VALUES (1, "1971-1-8",  39, "m"),   # Nur mÃ¶glich, da Werte
                         (2, "2001-5-13",  9, "m"),   # ALLER Spalten in der
                         (8, "1969-12-1", 41, "w"),   # korrekten Reihenfolge
                         (9, "1975-7-3",  35, "m");   # angegeben werden!
  INSERT INTO age                                     # Andere Schreibweise
    SET nr           = 7,
        geburtsdatum = "1966-1-1",
        jahre        = 44,
        geschlecht   = "m";

Zwei (oder mehr) Tabellen gemeinsam abfragen und alle Kombinationen aller
DatensÃ¤tze ausgeben (Kreuzprodukt = "CROSS JOIN" von "pers" und "age"). Die
Liste der Tabellen ist durch "," getrennt hintereinander anzugeben:

  SELECT *                          # CROSS JOIN (alle Kombinationen)
    FROM pers, age;                 #

Kreuzprodukt von zwei (oder mehr) Tabellen durchfÃ¼hren und Ergebnismenge Ã¼ber
gleiche Werte in der Spalte "nr" einschrÃ¤nken ("INNER JOIN" von "pers" und
"age"). Die mehrdeutige Spalte "nr" wird durch vorangestellte Tabellennamen
"pers" und "age" qualifiziert (genau spezifiziert):

  SELECT *                          # INNER JOIN mit WHERE (implizit)
    FROM pers, age                  #
      WHERE pers.nr = age.nr;       #

Alternativ "Aliase" (eindeutige Kurznamen, GROSS/kleinschreibung relevant) "p"
und "a" fÃ¼r die Tabellen "pers" und "age" einfÃ¼hren (besser lesbar und kÃ¼rzer):

  SELECT *                          # INNER JOIN mit WHERE (implizit)
    FROM pers AS p, age AS a        # Auch ... FROM pers p, age a ...
      WHERE p.nr = a.nr;            #

Gleiche Abfrage in expliziter JOIN-Syntax (Variante B+C ist nur mÃ¶glich,
falls die VerknÃ¼pfungsspalte in beiden Tabellen gleich heiÃM-^_t, wie hier "nr"):

  SELECT *                          # A) INNER JOIN mit ON
```

```
    FROM pers JOIN age               # (Spalte "nr" doppelt)
      ON pers.nr = age.nr;           # Allgemeine Syntax

  SELECT *                           # B) INNER JOIN mit USING
    FROM pers JOIN age               # (Spalte "nr" in beiden Tab. vorhanden)
      USING (nr);

  SELECT *                           # C) NATURAL JOIN (nur mÃ¶glich, falls
    FROM pers NATURAL JOIN age;      #  "nr" einzige gleichnamige Spalte ist)

TIP: Index oder Primary Key auf diversen Spalten anlegen (damit laufen obige
SELECT-Abfragen mit VerknÃ¼pfung beider Tabellen SEHR VIEL SCHNELLER!).

Nur Teil der DatensÃ¤tze ab bestimmter Position holen (<Rows> DatensÃ¤tze ab
Datensatz <Offset>, STD: alle ab 0), die DatensÃ¤tze sind beginnend mit "0"
nummeriert --- MY!).

  SELECT * FROM pers LIMIT 10;           # 10 SÃ¤tze ab Datensatz 0
  SELECT * FROM pers LIMIT 3, 5;         #  5 SÃ¤tze ab Datensatz 3
  SELECT * FROM pers LIMIT 5 OFFSET 3;   #  5 SÃ¤tze ab Datensatz 3

TIP: Abfrage SORTIEREN, sonst ist die Reihenfolge der DatensÃ¤tze zufÃ¤llig
gemÃ¤Ã-^_ ihrer "physikalischen" Engine-Reihenfolge.

  SELECT * FROM pers ORDER BY name, vorname LIMIT 10;

TIP: Immer einen PrimÃ¤rschlÃ¼ssel anlegen, der jeden Datensatz eindeutig
identifiziert. Falls dies aus den Daten heraus nicht mÃ¶glich ist, dafÃ¼r eine
Spalte mit einem "kÃ¼nstlichen SchlÃ¼ssel" (eindeutige Nummer) anlegen.

Index gleich beim Erstellen einer Tabelle mit anlegen (Variante A):

  CREATE TABLE pers (
    nr  INT NOT NULL,                      # NOT NULL bei PRIMARY KEY!
    ...                                    #
    PRIMARY KEY         ON (nr),           # PrimÃ¤rschlÃ¼ssel
    UNIQUE   INDEX idx2 ON (name),         # Eindeutige Spalte
             INDEX idx3 ON (name, vorname),# 2 Spalten verkettet
    FULLTEXT INDEX idx4 ON (vorname),      # Volltextsuche
    SPATIAL  INDEX idx4 ON (nr)            # Geometriedaten
  );

Index nach Erstellen einer Tabelle hinzufÃ¼gen (Variante B, jederzeit mÃ¶glich):

  CREATE PRIMARY KEY         ON pers (nr);             # FALSCH! --> ALTER TABLE
  CREATE UNIQUE INDEX   idx2 ON age  (nr);             # Eindeutige Spalte(n)
  CREATE INDEX          idx3 ON pers (name, vorname);  # 2 Spalten verkettet
  CREATE FULLTEXT INDEX idx4 ON age  (name);           # Volltextsuche
  CREATE SPATIAL INDEX  idx5 ON age  (nr);             # Geometriedaten

Index / Primary Key zu Tabelle hinzufÃ¼gen (Variante C, jederzeit mÃ¶glich):

  ALTER TABLE pers ADD PRIMARY KEY         (nr);     # Name unnÃ¶tig ("PRIMARY")
  ALTER TABLE age  ADD UNIQUE INDEX   idx2 (nr);     # Eindeutige Spalte(n)
  ALTER TABLE pers ADD INDEX          idx3 (name, vorname); # 2 Spalten verk.
  ALTER TABLE age  ADD FULLTEXT INDEX idx4 (name);   # Volltextsuche
  ALTER TABLE age  ADD SPATIAL INDEX  idx5 (nr);     # Geometriedaten

Indices zu einer Tabelle anzeigen:

  SHOW INDEX FROM pers;
  SHOW INDEX FROM age;

Index oder Primary Key einer Tabelle entfernen (jederzeit mÃ¶glich):

  ALTER TABLE pers DROP PRIMARY KEY;   # Kein Name nÃ¶tig
  ALTER TABLE pers DROP INDEX idx2;    # Name "idx2" nÃ¶tig, UNIQUE weglassen!
  DROP INDEX idx3 ON pers;             # Name "idx3" nÃ¶tig

TIP: Index auf PrÃ¤fix von Spalten bzw. AUF/ABsteigend erzeugen (STD: ASC),
fÃ¼r Tuningzwecke zur Platzersparnis interessant (MY!).

  CREATE INDEX idx8 ON pers (name(5), vorname(5));     # Platz sparen!
  CREATE INDEX idx9 ON pers (name ASC, vorname DESC);  # Auf+Absteigend

Alle DatensÃ¤tze Ã¤ndern (VORSICHT: keine WHERE-Bedingung --> ALLE DatensÃ¤tze!):

  UPDATE pers
    SET nr = nr + 100;

Bestimmte DatensÃ¤tze Ã¤ndern:
```

```
  UPDATE pers                       # UPDATE FROM <Tbl> geht in MySQL nicht!
    SET nr = 111,                   #
        name = DEFAULT              # DEFAULT-Wert aus Tab.definition einsetzen
      WHERE vorname = "Thomas";     #

DatensÃ¤tze ersetzen oder einfÃ¼gen: Analog INSERT falls Datensatz mit diesem
PrimÃ¤rschlÃ¼ssel/UNIQUE INDEX noch nicht vorhanden ist, sonst DELETE des alten
+ INSERT des neuen Datensatzes durchfÃ¼hren (PrimÃ¤rschlÃ¼ssel notwendig,
DELETE-Recht notwendig, AUTO_INCREMENT-Feld erhÃ¶ht sich --- MY!):

  REPLACE INTO copy (nr, vorname, name)   # (INTO optional)
    VALUES (1, "Thomas", "Birnthaler"),   # Mehrere DatensÃ¤tze erlaubt
           (2, "Hans",   "Dampf");        #
                                          #
  REPLACE INTO copy (nr, vorname, name)   # (INTO optional)
    SELECT nr, vorname, name              #
      FROM pers,                          #
        WHERE nr < 100;                   #
                                          #
  REPLACE pers                            # (INTO optional)
    SET nr      = 7,                      #
        vorname = "Heinz",                #
        name    = "Beier";                #

DatensÃ¤tze ersetzen oder einfÃ¼gen: Analog INSERT falls Datensatz mit diesem
PrimÃ¤rschlÃ¼ssel/UNIQUE INDEX noch nicht vorhanden, sonst UPDATE des alten
Datensatzes durchfÃ¼hren (bessere Variante, PrimÃ¤rschlÃ¼ssel notwendig,
AUTO_INCREMENT-Feld bleibt gleich --- MY!):

  INSERT INTO pers (nr, vorname, name)
    VALUES (1, "Thomas", "Birnthaler")
      ON DUPLICATE KEY UPDATE vorname = "Thomas",
                              name    = "Birnthaler";

HINWEIS: REPLACE Ã¤ndert AUTO_INCREMENT-Wert bereits vorhandener DatensÃ¤tze,
INSERT INTO ON DUPLICATE KEYS UPDATE macht das nicht (schwierigere Syntax)!

Bestimmte DatensÃ¤tze lÃ¶schen:

  DELETE FROM pers
    WHERE vorname = "Markus" OR nr >= 9;

Alle DatensÃ¤tze lÃ¶schen, Tabelle belassen (VORSICHT: keine WHERE-Bedingung!):

  DELETE FROM pers;      # Langsam (DatensÃ¤tze einzeln lÃ¶schen = Transaktion!)
  TRUNCATE TABLE pers;   # Schnell (Tabelle lÃ¶schen + neu anlegen, nicht in TA!)
  TRUNCATE pers;         #   (analog)

HINWEIS: Auch LÃ¶sch- und Update-Operation sind per LIMIT auf eine bestimmte
Anzahl DatensÃ¤tze beschrÃ¤nkbar (ob das sinnvoll ist, sei dahingestellt!). Dabei
sollten die DatensÃ¤tze mit ORDER BY sortiert werden, um sie nicht zufÃ¤llig
gemÃ¤Ã-^_ ihrer "physikalischen" Reihenfolge zu lÃ¶schen oder zu Ã¤ndern:

  DELETE FROM pers             # Max. 2 DatensÃ¤tze lÃ¶schen
    WHERE vorname = "Markus"   #
      ORDER BY name            # Wichtig!
        LIMIT 2;               #

  UPDATE pers                  # Max. 3 DatensÃ¤tze Ã¤ndern
    SET nr = nr + 1000         #
      WHERE nr < 1000          #
        ORDER BY name          # Wichtig!
          LIMIT 3;             #

TIP: Mit der "mysql"-Client-Option --i-am-a-dummy / --safe-updates / -U sind
UPDATE- und DELETE-Anweisungen gegen Weglassen einer WHERE- oder LIMIT-Klausel
geschÃ¼tzt (d.h. versehentliches Ã-^Dndern/LÃ¶schen ALLER DatensÃ¤tze --- auÃ-^_er per
TRUNCATE --- wird verhindert).

Anzahl DatensÃ¤tze, Spaltenwerte oder HÃ¤ufigkeit verschiedener Spaltenwerte in
einer Tabelle ermitteln (Gruppieren bzw. Aggregieren):

  SELECT COUNT(*) FROM pers;             # Anz. DatensÃ¤tze in Tab. (inkl. NULL!)
  SELECT COUNT(nr) FROM pers;            # Anz. Werte von Sp. "nr" (ohne NULL!)
  SELECT COUNT(DISTINCT nr) FROM pers;   # Anz. versch. Werte von Sp. "nr" (ohne NULL!)
  SELECT name, COUNT(name) FROM pers     # HÃ¤ufigkeit der Werte in Sp. "name"
    GROUP BY name;                       #
  SELECT name, COUNT(name) FROM pers     #   (analog, nur Werte mit HÃ¤uf. > 3)
    GROUP BY name                        #
      HAVING COUNT(name) > 3;            #
  SELECT name, COUNT(name) FROM pers     #   (analog + Gesamtsumme (NULL als Wert!)
    GROUP BY name                        #
```

```
      WITH ROLLUP;                        # (MY!) ACHTUNG: HAVING wirkt darauf!

Weitere typische Aggregatfunktionen analog COUNT (arbeiten auf der Menge aller
von einer Abfrage gefundenen DatensÃ¤tze):

  SELECT SUM(nr),            # Summe aller Werte von Spalte "nr"
         SUM(DISTINCT nr),   # Summe aller verschiedenen Werte von Spalte "nr"
         AVG(nr),            # Durchschnitt aller Werte von Spalte "nr"
         MIN(nr),            # Minimum aller Werte von Spalte "nr"
         MAX(nr)             # Maximum aller Werte von Spalte "nr"
    FROM pers;

Datensatz mit hÃ¶chster Nummer:

  SELECT DISTINCT * FROM pers WHERE nr = MAX(nr);  # FALSCH!
  SELECT DISTINCT * FROM pers                      # A) Subselect fÃ¼r Maximum
    WHERE nr = (SELECT MAX(nr) FROM pers);         #
  SET @max := (SELECT MAX(nr) FROM pers);          # B) Variable @max fÃ¼llen
  SET @max = (SELECT MAX(nr) FROM pers);           #    Variable @max fÃ¼llen
  SELECT * FROM pers WHERE nr = @max;              #    (:= / = ist Zuweisung)
  SELECT * FROM pers ORDER BY nr DESC LIMIT 1;     # C) Abbruch per LIMIT
  SELECT p1.* FROM pers p1 LEFT JOIN pers p2       # D) FALSCH!
    ON p1.nr <= p2.nr WHERE p2.nr IS NULL;         #

Datensatz mit niedrigster oder hÃ¶chster Nummer:

  SELECT * FROM pers WHERE nr = MIN(nr) OR nr = MAX(nr);   # A) FALSCH!
  SELECT * FROM pers WHERE nr IN (MIN(nr), MAX(nr));       # B) FALSCH!
  SELECT * FROM pers WHERE nr IN (                         # C) Subselect
    (SELECT MIN(nr) FROM pers),                            #
    (SELECT MAX(nr) FROM pers)                             #
  );                                                       #
  SELECT @min := MIN(nr), @max := MAX(nr) FROM pers;       # D) Variablen + ...
  SELECT MIN(nr), MAX(nr) INTO @min, @max;                 #      (analog MY!)
  SELECT * FROM pers WHERE nr = @min OR nr = @max;         # D1) ... log. Bed
  SELECT * FROM pers WHERE nr IN (@min, @max);             # D2) ... Menge
  SELECT * FROM pers WHERE nr = @min                       # D3) ... Union
    UNION ALL
  SELECT * FROM pers WHERE nr = @max;

Allgemeines SELECT-Statement (alle MÃ¶glichkeiten):

  SELECT <Col>, ...              # Spaltenauswahl
    FROM <Tbl>, ...              # Tabellenauswahl
      [WHERE <Cond>]             # Bedingung auf Spalten (Condition)
        [GROUP BY <Col>, ...     # Aggregation auf Spalten (DatensÃ¤tze zusammenfassen)
          [HAVING <Cond>]]       # Bedingung auf Aggregation (nur bei GROUP BY!)
            [ORDER BY ...]       # Sortierung nach Spalten (Name oder Nummer!)
              [LIMIT ...]        # Anz. DatensÃ¤tze + Startpos. begrenzen (MY!)

Beispiel:

  SELECT nr,
         COUNT(name) AS "Anz",
         name AS "Name"
    FROM pers
      WHERE nr > 2
        GROUP BY name
          ORDER BY nr DESC
            LIMIT 2;

HINWEIS: MySQL erlaubt SELECT-Anweisungen ohne Tabellen-Bezug, um z.B. das
Ergebnis von AusdrÃ¼cken zu berechnen (in Oracle ist dazu das AnhÃ¤ngen von
"FROM dual" notwendig!). Die bei MySQL (und Oracle) grundsÃ¤tzlich vorhandene
"Dummy"-Tabelle "dual" (enthÃ¤lt EINEN Datensatz ohne Spalten) gibt es daher
nur aus KompatibilitÃ¤tsgrÃ¼nden zu Oracle:

  SELECT SQRT(2);              # In MySQL erlaubt (in Oracle nicht!)
  SELECT SQRT(2) FROM dual;    # In MySQL und Oracle erlaubt (Ergebnis gleich)

Tabellenstruktur und -name Ã¤ndern:
  * Mehrere Ã„M-^Dnderungen gleichzeitig durch Komma getrennt angebbar
  * Dabei wird KOPIE mit Ã„M-^Dnderungen angelegt (dauert bei vielen Daten lange!)
    und anschlieÃM-^_end das ORIGINAL durch die KOPIE ersetzt
  * Tabelle wird GESPERRT, d.h. weitere Anfragen an sie mÃ¼ssen solange warten
  * Reines Umbenennen einer Tabelle wird ohne Kopie erledigt
  * Verschieben einer Tabelle in andere DB nur im gleichen Dateisystem mÃ¶glich
  * Beim Spaltentyp Ã¤ndern werden Spaltendaten so weit mÃ¶glich erhalten
    (evtl. abgeschnitten oder mit Leerzeichen aufgefÃ¼llt)
  * Tabelle/Engine Ã¤ndern NICHT auf Systemtabellen in DB "mysql" anwenden!

Ã„M-^Dnderungs-Operationen auf Tabellen (mehrere gleichzeitig erlaubt, COLUMN, TO
```

und CONSTRAINT dÃ¼rfen weggelassen werden):

Operation	Bedeutung
ADD [COLUMN]	Spalte einfÃ¼gen (FIRST, AFTER <Col>, STD: hinten)
DROP [COLUMN]	Spalte entfernen (mit Daten + Indices darauf)
MODIFY [COLUMN]	Sp.typ Ã¤ndern (Sp.name bleibt) oder Sp. verschieben
CHANGE [COLUMN]	Spalte neu definieren (Name + Typ, d.h. auch umben.)
ALTER [COLUMN]...	Defaultwert...
...SET DEFAULT	...Ã¤ndern
...DROP DEFAULT	...entfernen
RENAME [TO]	Tabellenname Ã¤ndern, Tabelle in andere DB verschieben
ORDER BY	DatensÃ¤tze fÃ¼r schnelle Abfrage sortieren (MY!)
ENGINE	Andere Datenbank-Engine verwenden (MY!)
TYPE	Andere Datenbank-Engine verwenden (veraltet, MY!)
IMPORT TABLESPACE	(MY!)
DISCARD TABLESPACE	(MY!)
ADD PRIMARY KEY	PrimÃ¤rschlÃ¼ssel hinzu (muss NOT NULL UNIQUE sein)
ADD UNIQUE INDEX	SekundÃ¤rschlÃ¼ssel hinzu (muss NOT NULL UNIQUE sein)
ADD {INDEX | KEY}	Index hinzu (muss nicht NOT NULL oder UNIQUE sein)
DROP PRIMARY KEY	PrimÃ¤rschlÃ¼ssel entfernen
DROP {INDEX | KEY}	Index entfernen
DISABLE KEYS	Alle Indices der Tabelle abschalten (nur Non-UNIQUE!)
ENABLE KEYS	Alle Indices der Tabelle aktivieren (nur Non-UNIQUE!)
ADD FOREIGN KEY	FremdschlÃ¼sselbezug hinzufÃ¼gen
DROP FOREIGN KEY	FremdschlÃ¼sselbezug entfernen
ADD CONSTRAINT	Spalten-BeschrÃ¤nkung hinzufÃ¼gen (ignoriert MY!)
DROP CONSTRAINT	Spalten-BeschrÃ¤nkung entfernen (nicht verfÃ¼gbar MY!)
CONVERT TO CHARACTER SET ... COLLATE ...	Zeichensatz konvertieren
[DEFAULT] CHARACTER SET ... COLLATE ...	Standard-Zeichensatz festlegen

Beispiele:

```
ALTER TABLE pers ...
... ADD COLUMN preis DECIMAL(10,2);       # STD: Spalte hinten anfÃ¼gen
... ADD COLUMN (strasse CHAR(30), plz LONG, ort CHAR(30));
... ... FIRST;                            # Als 1. Spalte einfÃ¼gen
... ... AFTER nr;                         # Nach Spalte "nr" einfÃ¼gen
... DROP   COLUMN name;                   # Spalte entfernen (inkl. Daten)
... MODIFY COLUMN vorname VARCHAR(50);    # Typ Ã¤ndern (Daten mÃ¶gl. erhalten)
... CHANGE COLUMN name name VARCHAR(20);  # (analog)
... ... FIRST;                            # Als 1. Spalte verschieben
... ... AFTER nr;                         # Nach Spalte "nr" verschieben
... CHANGE COLUMN vorname name CHAR(20);  # Name+Typ Ã¤nd. (Daten mÃ¶gl. erh.)
... CHANGE COLUMN name vorname CHAR(20);  # Name Ã¤nd. (Typ identisch wÃ¤hlen!)
... ALTER  COLUMN vorname SET DEFAULT ""; # Default Ã¤nd.: Leere Zeichenkette
... ALTER  COLUMN vorname DROP DEFAULT;   # Default Ã¤nd.: NULL
                                          #
... RENAME TO pers_old;                   # Tabelle umbenennen
... RENAME TO test2.pers;                 # Tabelle in andere DB verschieben
... ORDER BY vorname DESC, name ASC;      # DatensÃ¤tze sortieren
... ENGINE = MyISAM;                      # Engine Ã¤ndern (neu!)
... TYPE = MyISAM;                        # Engine Ã¤ndern (veraltet!)
                                          #
... ADD PRIMARY KEY (nr);                 # PrimÃ¤rschlÃ¼ssel hinzufÃ¼gen
... DROP PRIMARY KEY;                     # PrimÃ¤rschlÃ¼ssel entfernen
... INDEX idx1 (name, vorname);           # Index hinzufÃ¼gen
... DROP INDEX idx1;                      # Index entfernen
... UNIQUE INDEX idx2 (name, vorname);    # SekundÃ¤rschlÃ¼ssel hinzufÃ¼gen
... DROP UNIQUE INDEX idx2;               # SekundÃ¤rschlÃ¼ssel entfernen
... DISABLE KEYS;                         # Alle Indices abschalten (nur Non-UNIQUE)
... ENABLE KEYS;                          # Alle Indices aktivieren (nur Non-UNIQUE)
                                          #
... ADD FOREIGN KEY age(nr)               # FremdschlÃ¼ssel hinzufÃ¼gen
                REFERENCES pers(nr);      #
... DROP FOREIGN KEY age;                 # FremdschlÃ¼ssel entfernen
... ADD CONSTRAINT pruef CHECK (nr >= 1); # Spalten-BeschrÃ¤nkung hinzufÃ¼gen
... DROP CONSTRAINT pruef;                # Nicht verfÃ¼gbar!
                                          #
... CONVERT TO CHARACTER SET "latin1"     # Zeichensatz aller Sp. konvertieren
            COLLATE "latin1_german_ic";   #
... [DEFAULT] CARACTER SET = "latin1"     # Zeichensatz aller Sp. konvertieren
            COLLATE = "latin1_german_ic"; #
```

```
Tabelle(n) umbenennen (auch in andere Datenbank verschieben, wenn die beiden
Datenbanken auf dem gleichen Dateisystem liegen). Stellt eine "atomare"
Operation dar, auch bei gleichzeitiger Umbenennung mehrerer Tabellen. LÃ¤uft
sehr schnell ab, da keine Kopie erstellt wird:

  RENAME TABLE pers TO new [, ...];   # Nicht fÃ¼r temp. Tabelle
  ALTER TABLE new RENAME TO pers;     # Auch fÃ¼r temp. Tabelle
  RENAME TABLE pers TO copy,          # Zwei Tabellenamen vertauschen (FALSCH!)
               copy TO pers;          # (-> ERROR: Table 'copy' already exists)
  RENAME TABLE pers TO tmp,           # Zwei Tabellenamen vertauschen (OK!)
               copy TO pers,          #
               tmp  TO copy;          #

Tabelle (bedingt) lÃ¶schen (inkl. aller Daten und Indices!):

  DROP TABLE pers;              # Fehler falls nicht existent
  DROP TABLE IF EXISTS pers;    # Kein Fehler falls nicht existent
  DROP TEMPORARY TABLE pers;    # TemporÃ¤re Tabelle lÃ¶schen (nicht echte!)

Datenbank (bedingt) lÃ¶schen (alle Tabellen mit allen Daten!):

  DROP DATABASE first;             # Fehler falls nicht existent
  DROP SCHEMA   first;             #   (analog)
  DROP DATABASE IF EXISTS first;   # Kein Fehler falls nicht existent
  DROP SCHEMA   IF EXISTS first;   #   (analog)

Abfrage von Informationen zur vorhergehenden SQL-Anweisung (pro Sitzung):

  SELECT ROW_COUNT();               # Anz. verarb. DS (INSERT, UPDATE, DELETE, REPLACE)
  SELECT SQL_CALC_FOUND_ROWS ...;   # Vor Einsatz von FOUND_ROWS() notwendig!
  SELECT FOUND_ROWS();              # Gesamtanz. DatensÃ¤tze bei SELECT mit LIMIT
  SELECT LAST_INSERT_ID();          # Letzter AUTO_INCREMENT-Wert bei INSERT

Fehlermeldungen, Warnungen und Bemerkungen (Notes) zur vorhergehenden
SQL-Anweisung ausgeben (pro Sitzung):

  SHOW ERRORS;                # Nur Fehler ausgeben
  SHOW WARNINGS;              # Fehler, Warnungen, Bemerkungen (Notes) ausgeben
  SET max_error_count = 0;    # Aufzeichnung der Warnungen/Fehler ausschalten
  SET max_error_count = 64;   # Max. Anz. aufgezeichneter Warnungen/Fehler (STD)
  SET sql_notes = 1;          # Bemerkungen (Notes) 1=aufzeichnen/0=unterdrÃ¼cken
  SET sql_warnings = 1;       # INSERT-Warnung 1=aufzeichnen/0=unterdrÃ¼cken
  SELECT @@error_count;       # Anzahl gemerkter Fehler
  SELECT @@warning_count;     # Anzahl gemerkter Warnungen


4) MySQL-Datentypen
-------------------

Im Prinzip kÃ¶nnten man die Spaltenwerte einer Tabelle immer in Form von
Zeichenketten abspeichern. MySQL wÃ¼rde diese automatisch beim Zugriff in andere
Datentypen wie Ganz/FlieÃM-^_kommazahlen, Datums/Zeitwerte sowie Boolesche Werte
umwandeln. Der Konvertierungsaufwand wÃ¤re aber fÃ¼r jeden Datensatz bei jedem
Zugriff durchzufÃ¼hren und daher zu aufwendig.

Daher sollte abgestimmt auf die zu speichernden Werte fÃ¼r jede Tabellenspalte
einer der folgenden MySQL-Datentypen ausgewÃ¤hlt werden. Diese Auswahl ist nicht
immer eindeutig (z.B. kÃ¶nnte ein Jahr als INT, FLOAT, DECIMAL oder YEAR
gespeichert werden). Bei sinnvoller Auswahl des Datentyps gilt:

  * Erlaubter Wertebereich sinnvoll eingeschrÃ¤nkt (ungÃ¼ltige Daten abgelehnt)
    (neben dem Spaltennamen ein wichtiger Teil der Dokumentation)
  * Platzbedarf minimiert
  * Zugriffsgeschwindigkeit maximiert
  * Indices effizient nutzbar fÃ¼r Zugriff und Sortierung
  * Typspezifische MySQL-Funktionen und -Operatoren einsetzbar

TIP: Auch an evtl. zukÃ¼nftige Erweiterungen denken und z.B. nicht fÃ¼r das
Speichern einer Jahreszahl nur 2 Stellen ("Y2K-Problem") oder einer IP-Adresse
oder Netzwerk-Maske nur 4 Byte (IPv4) vorsehen (IPv6 benÃ¶tigt 16 Byte).

  +--------------------------+--------------+-------+--------------------------+
  | Datentyp                 | Typname      | Byte  | Wertebereich             |
  +--------------------------+--------------+-------+--------------------------+
  | Ganzzahl (mit Vorzeichen) |   TINYINT    | 1     | -127..128                |
  |                          |  SMALLINT    | 2     | -32768..32767            |
  |                          | MEDIUMINT    | 3     | -8388608..8388607        |
  |                          |       INT    | 4     | -2147483648..2147483647  |
  |                          |    BIGINT    | 8     | -9223372036854775808..   |
  |                          |              |       |  ..9223372036854775807   |
  +----------------------+-----+--------------+-------+--------------------------+
  | Ganzzahl (ohne Vz.) |   TINYINT UNSIGNED | 1     | 0..255                   |
```

	SMALLINT UNSIGNED	2	0..65535
	MEDIUMINT UNSIGNED	3	0..16777215 (16 Mio)
	INT UNSIGNED	4	0..4294967295 (4 Mrd)
	BIGINT UNSIGNED	8	0..18446744073709551615
FlieÃM-^_kommazahl (technisch, nicht kaufm.)	FLOAT	4	10^38 (7 Dez.stellen)
	DOUBLE	8	10^308 (15 Dez.stellen)
FlieÃM-^_kommazahl (ohne Vorz.)	FLOAT UNSIGNED	4	10^38 (7 Dez.stellen)
	DOUBLE UNSIGNED	8	10^308 (15 Dez.stellen)
Festkommazahl (L max. 65) (vor MY!5.0 als String)	DECIMAL(L,N)	L+2 L/9*4	LÃ¤nge L mit N Nkst. (ab MY!5.0!)
Festkommazahl (ohne Vorz.)	DECIMAL(L,N) UNSIGNED	L+2 L/9*4	LÃ¤nge L mit N Nkst. (ab MY!5.0!)
Boolean	BOOL	1	0/1 FALSE=0/TRUE=1
Bitfeld	BIT(N)	(N+7)/8	N Bits (1..64)
Zeichenkette (LÃ¤nge fest)	CHAR(L)	L	L=0..255
(LÃ¤nge fest)	BINARY(L)	L	L=0..255
(LÃ¤nge variabel)	VARCHAR(L)	L+1/2	L=0..255/65535 (MY!5.1)
(LÃ¤nge variabel)	VARBINARY(L)	L+1/2	L=0..255/65535 (MY!5.1)
Datum + Zeit	YEAR	1	JJJJ/JJ (1900-2155)
	DATE	3	JJJJ-MM-TT (1000-9999)
	TIME	3	hh:mm:ss
	DATETIME	8	JJJJ-MM-TT hh:mm:ss
(Sek. seit 1.1.1970 00:01)	TIMESTAMP	4	JJJJMMTThhmmss
Mikrosekunden <Fsp> 0..6 (Fractional seconds part) seit MY!5.6.4 (0..3 Byte)	TIME(Fsp)	3+0-3	hh:mm:ss.ffffff fff
	DATETIME(Fsp)	5+0-3	JJJJ-MM-TT hh:mm:ss.fff
	TIMESTAMP(Fsp)	4+0-3	JJJJMMTThhmmss.ffffff
Text (Sonderzeichen interpretiert)	TINYTEXT	L+1	0-255
	TEXT	L+2	0-65535
	MEDIUMTEXT	L+3	0-16777216 (16 Mio)
	LONGTEXT	L+4	0-4294967296 (4 Mrd)
Binary large object BLOB (binÃ¤re Daten, keine Interpretation von Sonderzeichen)	TINYBLOB	L+1	0-255
	BLOB	L+2	0-65535
	MEDIUMBLOB	L+3	0-16777216 (16 Mio)
	LONGBLOB	L+4	0-4294967296 (4 Mrd)
AufzÃ¤hlung (Single-Value)	ENUM(V1, ...)	1/2	1..255, 2=256-65535
Menge (Multi-Value)	SET(V1, ...)	1-4/8	1..8/16/24/32/64 Elem.
Unicodestring	NCHAR(L)	L	Synonym fÃ¼r CHAR
	NATIONAL VARCHAR(L)	L+1	Synonym fÃ¼r VARCHAR

MÃ¶gliche Attribute zu allen Datentypen (danach anzugeben):

Attribut	Bedeutung
DEFAULT <Val>	Defaultwert <Val> setzen, falls KEIN Wert angegeben
NOT NULL	Wert MUSS angegeben sein (evtl. per Defaultwert)
NULL	Wert darf weggelassen werden ("undefiniert", STD)

HINWEIS: Der Defaultwert <Val> muss konstant sein, Funktionen sind unzulÃ¤ssig.

MÃ¶gliche Attribute zu numerischen Datentypen (Ganzzahl, FlieÃM-^_komma, Festkomma):

Attribut	Bedeutung	
(<N>)	Ausgabebreite (INT)	
(<N>, <M>)	Ausgabebreite N und Nkst. M (FLOAT, DOUBLE, DECIMAL)	
UNSIGNED	Wert immer positiv	
ZEROFILL	Ausgabe mit fÃ¼hrenden Nullen gemÃ¤ÃM-^_ Breite (nicht DEC!)	
AUTO_INCREMENT	Pro NEU eingefÃ¼gtem Datensatz automatisch hochzÃ¤hlen (kÃ¼nstlicher Key, nur 1x pro Tabelle, NICHT DECIMAL!)	!!!
PRIMARY KEY	Spalte ist PrimÃ¤rschlÃ¼ssel (--> NOT NULL)	
CHARACTER SET <CharSet> COLLATE <Sort>	Zeichensatz und Sortierordnung	

```
MÃ¶gliche Attribute zu TIMESTAMP:

  +-----------------------------+---------------------------------------------+
  | Attribut                    | Bedeutung                                   |
  +-----------------------------+---------------------------------------------+
  | DEFAULT   CURRENT_TIMESTAMP | Neuer Datensatz --> aktuelle Zeit speichern |
  | ON UPDATE CURRENT_TIMESTAMP | Datensatz geÃ¤ndert --> akt. Zeit speichern  |
  +-----------------------------+---------------------------------------------+

Beispiel:

  DROP TABLE IF EXISTS data;
  CREATE TABLE data (
    nr         INT NOT NULL AUTO_INCREMENT,      #0  Automatisch fÃ¼llen
    anz        TINYINT(3) UNSIGNED,              #1  Ausgabebreite 3
    grad       FLOAT(7,1),                       #2  Ausgabeformat 12345.6
    betrag     DECIMAL(9,2),                     #3  Ausgabeformat 123456.78
    vorname    CHAR(30),                         #4  Zeichen (mit Leerz. aufgef.)
    name       VARCHAR(30) NOT NULL,             #5  Zeichen
    binaer1    BINARY(10),                       #6  Byte (mit 0-Byte aufgefÃ¼llt)
    binaer2    VARBINARY(8),                     #7  Byte
    ts1        TIMESTAMP,                        #8  Automatisch gefÃ¼llt (1.Sp.)
    ts2        TIMESTAMP(3),                     #9  NICHT " (2.Sp., Milli-Sek.)
    ts3        TIMESTAMP(6),                     #10 NICHT " (3.Sp., Mikro-Sek.)
    datum      DATE DEFAULT '2000-01-01',        #11
    uhrzeit    TIME DEFAULT '12:00:00',          #12
    datumzeit  DATETIME,                         #13 JJJJ-MM-TT hh:mm:ss
    jahr       YEAR(4),                          #14
    wahr       BOOL,                             #15 TRUE/FALSE
    flags      BIT(8),                           #16 Bitmuster 8 Bit = 1 Byte
    dokument   TEXT,                             #17
    antwort    ENUM('Ja', 'Nein', 'Vielleicht'), #18 Wert aus Liste
    menge      SET('gross','reich','maechtig'),  #19 Menge aus Listenwerten
    datei      BLOB,                             #20
    PRIMARY KEY (nr)   # Kein Komma, notwendig wg. AUTO_INCREMENT von Spalte nr
  );

  INSERT INTO data VALUES (
    NULL,                      #0  INT:           NÃ¤chste freie Nr. wg. AUTO_INCREMENT
    123,                       #1  TINYINT(3):    fÃ¼llen
    12345.6,                   #2  FLOAT(7,1):    fÃ¼llen
    876543.21,                 #3  DECIMAL(9,2):  fÃ¼llen
    "Thomas",                  #4  CHAR(30):      fÃ¼llen (mit Leerz. hinten auffÃ¼llen)
    "Birnthaler",              #5  VARCHAR(30):   fÃ¼llen
    0x41424344454647484 94A,   #6  BINARY(10):    fÃ¼llen (mit 0-Byte hinten auffÃ¼llen)
    x'6162636465666768',       #7  VARBINARY(8): fÃ¼llen
    NULL,                      #8  TIMESTAMP:     Aktuellen Zeitp. autom. eintragen
    "20120907112233.987654",   #9  TIMESTAMP:     fÃ¼llen
    "2012-09-07 11:22:33.123456", #10 TIMESTAMP:fÃ¼llen ("7.9.2012 11:22:33" NICHT OK!)
    "2012-08-03",              #11 DATE:          fÃ¼llen
    "23:59:59",                #12 TIME:          fÃ¼llen
    "2012-08-03 23:59:59",     #13 DATETIME:      fÃ¼llen ("3.8.2012 23:59:59" NICHT OK!)
    2012,                      #14 YEAR(4):       fÃ¼llen
    TRUE,                      #15 BOOL:          fÃ¼llen
    b'01000001',               #16 BIT(8):        fÃ¼llen ASCII-Code 65 = Zeichen 'A'
  # "Dokument(text)",          #17 TEXT:          fÃ¼llen (direkt) oder
  # LOAD_FILE("/etc/my.cnf"), #17 TEXT:          laden aus externer Datei
    LOAD_FILE("C:/ProgramData/MySQL/MySQL Server 5.6/my.ini"), #17 Windows
    "Vielleicht",              #18 ENUM:          Aufz.wert als String (EINER!)
    "gross,reich,maechtig",    #19 SET:           Elem. per "," getr. als String
  # "Blobdaten(binaer)"        #20 BLOB:          fÃ¼llen (direkt) oder
  # LOAD_FILE("/etc/my.cnf")  #20 BLOB:          laden aus externer Datei
    LOAD_FILE("C:/ProgramData/MySQL/MySQL Server 5.6/my.ini") #20 Windows
  # 0x41424344454647484 94A,   #20 BLOB:          fÃ¼llen (mit 0-Byte hinten)
  # x'6162636465666768',       #20 BLOB:          fÃ¼llen
  );

  SELECT * FROM data\G         # Ergibt (Spalten vertikal anzeigen)

    *************************** 1. row ***************************
           nr: 1                       #0  INT
          anz: 123                     #1  TINYINT(3)
         grad: 12345.6                 #2  FLOAT(7,1)
       betrag: 876543.21               #3  DECIMAL(9,2)
      vorname: Thomas                  #4  CHAR(30)
         name: Birnthaler              #5  VARCHAR(30)
      binaer1: ABCDEFGHIJ              #6  BINARY(10):   Zeichen zu Code 41 ... 4A
      binaer2: abcdefgh                #7  VARBINARY(8): Zeichen zu Code 61 ... 68
          ts1: 2012-09-11 18:07:08     #8  TIMESTAMP
          ts2: 2012-09-07 11:22:33     #9  TIMESTAMP
          ts3: 2012-09-07 11:22:33     #10 TIMESTAMP
```

```
       datum: 2012-08-03            #11 DATE
     uhrzeit: 23:59:59              #12 TIME
   datumzeit: 2012-08-03 23:59:59   #13 DATETIME
        jahr: 2012                  #14 YEAR(4)
        wahr: 1                     #15 BOOL
       flags: A                     #16 BIT(8)
    dokument: Dokument(text)        #17 TEXT
     antwort: Vielleicht            #18 ENUM
       menge: gross,reich,maechtig  #19 SET
       datei: Blobdaten(binaer)     #20 BLOB
   1 row in set (0.00 sec)
```

Alternative Namen fÃ¼r einige Datentypen:

Name	Alternativer Name
BOOL	BOOLEAN, TINYINT(1)
CHAR	CHARACTER
DECIMAL	DEC, NUMERIC
DOUBLE	REAL, DOUBLE PRECISION
INT	INTEGER
NCHAR	NATIONAL CHAR, NATIONAL CHARACTER
VARCHAR	CHARACTER VARYING

Beispiele fÃ¼r die Angabe von Konstanten (Literalen) fÃ¼r die Datentypen
(Q = Quotieren des Wertes mit "..." oder '...' notwendig):

Datentyp	Q	Beispiele fÃ¼r Konstanten/Literale
INT	-	0 123 -456 +7899123 0x10EF x'10EF' b'11010'
FLOAT	-	0.0 1.23 -0.456 1.23e14 -12E-34
DECIMAL	-	0.0 -123.000 +456789.00
BOOL	-	TRUE FALSE 0 1 "aaa" ""
BIT	x	b'10101100' 0x10EF x'10EF'
CHAR	x	"Hallo" 'Hallo' "" ''
VARCHAR	x	"Hallo" 'Hallo' "" ''
BINARY	x	0x10EF x'10EF' _utf8'uc' "" ''
VARBINARY	x	0x10EF x'10EF' _utf8'uc' "" ''
YEAR	-	69 99 13 1972 2010 2100
DATE	x	"2009-10-18" "0000-00-00" '2009-10-00'
TIME	x	"23:59:59" '09:45' "00:00:00"
DATETIME	x	"2009-10-18 22:30:00" YYYYMMDDhhmmss
	x	"2009-10-18 22:30:00.123456" YYYYMMDDhhmmss.nnnnnn
TIMESTAMP	x/-	"2009-10-18 22:30:00" 20091018223000 (Zahl, sic!)
		"2009-10-18 22:30:00.123456"
		20091018223000.12345 (Zahl, sic!)
TEXT	x	"Hallo" 'Hallo' "" '' _utf8'uc'
BLOB	x	0x10EF x'10EF' b'11010' "Daten..." 'Daten...'
ENUM	x	"1.Wert" '5.Wert'
SET	x	"1.Wert,5.Wert,8.Wert" "5.Wert" "" (leer)

Defaultwert und Defaultattribut NULL/NOT NULL fÃ¼r Datentypen:

Datentyp	Defaultwert	Defaultfall
INT	0	NULL
FLOAT	0.0	NULL
DECIMAL	0.0	NULL
BIT	b'' (kein Bit gesetzt)	NULL
BOOL	0 (FALSE)	NULL
CHAR	"" (leerer String)	NULL
VARCHAR	"" (leerer String)	NULL
BINARY	"" (leere Bytefolge)	NULL
VARBINARY	"" (leere Bytefolge)	NULL
YEAR	00/0000	NULL
DATE	"0000-00-00"	NULL
TIME	"00:00:00"	NULL
DATETIME	"0000-00-00 00:00:00"	NULL
TIMESTAMP	CURRENT_TIMESTAMP() (1.Spalte, akt. Datum+Zeit)	NOT NULL(!)

	"0000-00-00 00:00:00" (restliche Spalten)	NULL
TEXT	"" (leerer String)	NULL
BLOB	"" (leerer String)	NULL
ENUM	0 (falls NULL), "1. Wert" (falls NOT NULL)	NULL
SET	"" (leere Menge)	NULL

HINWEISE:

* Datentyp-Konvertierung erfolgt entweder automatisch oder explizit per
  (BINARY entspricht CAST(<String> AS BINARY)):

```
    BINARY <String>                  # GROSS/kleinschreibung berÃ¼cksichtigen
    CAST(<Val> AS <Typ>)             # Datentyp von <Val> in <Typ> umwandeln
    CONVERT(<Val> AS <Typ>)          # Datentyp von <Val> in <Typ> umwandeln
    CONVERT(<Val> USING <Transcode>) # Zeichensatz von <Val> konvertieren
```

* Zieldatentyp <Typ> einer Datentyp-Konvertierung kann sein:

BINARY[(n)]	Max. n Byte nutzen (mit 0x00 auffÃ¼llen)
CHAR[(n)]	Max. n Byte nutzen (mit Space auffÃ¼llen)
DATE	
DATETIME	
DECIMAL[(m[,d])]	
SIGNED [INTEGER]	
TIME	
UNSIGNED [INTEGER]	

* Folgende Datentyp-Konvertierungen sind mÃ¶glich:

Von	Nach
DECIMAL	CHAR VARCHAR DATE TIME DATETIME
CHAR VARCHAR	DECIMAL DATE TIME DATETIME
DATE TIME DATETIME	CHAR VARCHAR
BLOB TEXT	(keine!)

* INT/FLOAT/DOUBLE: Bei Ganzzahlen ist in Klammern eine "DarstellungsgrÃ¶Ã-^_e"
  (1-255) angebbar, bei FlieÃ-^_kommazahlen die "DarstellungsgrÃ¶Ã-^_e" und
  die "Anzahl der Nachkommastellen" (DSG >= NKST + 2). Sie bestimmen nicht
  die LÃ¤nge oder Genauigkeit der gespeicherten Zahl, sondern legen fest, mit
  welcher Breite und welcher Nachkommstellenanzahl die AUSGABE erfolgen soll.

* FLOAT/DOUBLE: Rundungsfehler beim Rechnen mÃ¶glich, besser DECIMAL nehmen,
  wenn es um finanzmathematische Rechnungen geht (z.B. Buchhaltung).

* DECIMAL: Zahlen dieses Typs sind vor MY!5.0 in Wirklichkeit Strings mit je
  einem Zeichen pro Ziffer, fÃ¼r das Komma und fÃ¼r das Vorzeichen (STD: 10.0)
  und werden bei Rechnungen nach DOUBLE konvertiert (max. 15 Stellen genau).

  Ab MY!5.0 wird dafÃ¼r "Precision Math" verwendet, d.h. eine kompaktere
  (9 Ziffern pro 4 Byte + Restziffern) und genauere Darstellung (max. 64 St.)
  benutzt und bei Rechnungen nicht mehr nach DOUBLE umgewandelt. Hat Ã¤hnliches
  Verhalten wie BCD-Format (Binary Coded Digits) fÃ¼r kaufmÃ¤nnisches Rechnen.

  KaufmÃ¤nnische oder finanzmathematische Daten wie Preise, Umsatz, ... mit
  DECIMAL darstellen (vermeidet Rundungsfehler von FLOAT/DOUBLE) oder statt
  in Euro mit 2 Nkst besser in Cent ohne Nkst in INT speichern.

  ACHTUNG: Dezimalkomma in Zahl (z.B. 1,50) ist (meist) ein Syntaxfehler,
  Dezimalkomma in String (z.B. "1,50") schneidet Rest nach Komma ab,
  falls als Zahl interpretiert (ignoriert).

* Zahlen kÃ¶nnen keine fÃ¼hrenden 0-en speichern
  + Falls das notwendig ist, CHAR/VARCHAR verwenden
    - Damit kann man trotzdem rechnen
    - Preis: Konvertierung String --> Zahl zur AusfÃ¼hrungszeit (langsam)!
  + Alternativ ZEROFILL (feste GesamtlÃ¤nge abhÃ¤ngig von Darstellungsbreite)

* BOOL: Kennt die Konstanten TRUE (Wahr, Wert 1) und FALSE (Falsch, Wert 0).
  Jeder Wert auÃ-^_er 0, FALSE (und NULL) ist TRUE (Wahr)!

* YEAR: Deckt folgende Bereiche ab (d.h. nur 255 Jahre darstellbar)
    YEAR(2): 1970-2069 (00-69 --> 20XX, 70- 99 --> 19XX)
    YEAR(4): 1900-2155 (00-99 --> 19XX, 100-255 --> 2XXX)
  ACHTUNG: Die Abbildungsregeln YY --> YYYY unterscheiden sich!

```
  YEAR(2) wird seit MY!5.X.X automatisch auf YEAR(4) abgebildet (deprecated)

* DATE: Werte decken die Jahre 01.01.1000-31.12.9999 ab. Jeder Teil (Tag, Monat,
  Jahr) ist auf "00" setzbar, um auszudrÃ¼cken, dass ein Datumteil unbekannt ist.
  Derartige Teile werden bei Sortierung als 1. Wert einsortiert. Ein insgesamt
  ungÃ¼ltiger Datumswert wird als "0000-00-00" gespeichert.

* Per SQL-Modus ist steuerbar, welche ungÃ¼ltigen Datumswerte akzeptiert werden:

  +---------------------+-----------------------------------------------------+
  | sql_mode = "..."    | Bedeutung                                           |
  +---------------------+-----------------------------------------------------+
  | ALLOW_INVALID_DATES | Beliebige Komb. von Mon=1..12 + Tag=1..31 erlaubt   |
  | NO_ZERO_DATE        | 0000-00-00 verboten (aber YYYY-01-00 / YYYY-00-01)  |
  | NO_ZERO_IN_DATE     | Jahr=00 oder Monat=00 oder Tag=00 nicht erlaubt     |
  +---------------------+-----------------------------------------------------+

  Beispiel ("_" und "-" in Option und Konfig-Datei mÃ¶glich):

    --sql_mode="ALLOW_INVALID_DATES,NO_ZERO_DATE"            # "mysqld"-Start
    sql_mode = ""                                            # In "my.cnf"
    SET GLOBAL sql_mode  = "ALLOW_INVALID_DATES";            # Server-Global
    SET SESSION sql_mode = "NO_ZERO_DATE,NO_ZERO_IN_DATE";   # Sitzungsbezogen
    SELECT @@GLOBAL.sql_mode;                                # Globaler Wert
    SELECT @@SESSION.sql_mode;                               # Sitzungsbez. Wert

* TIME: Werte decken Zeitbereich -838:59:59.000000 bis 838:59:59.999999 ab
  + Ab MY!5.6.4 auch auf Mikrosekunden .nnnnnn genau: TIME(<Fsp>)
    (Fsp = Fractional seconds part = 0-6 Stellen als zusÃ¤tzliche Angabe)

* DATETIME: Zusammenfassung von DATE und TIME. AbhÃ¤ngig vom SQL-Modus auch
  ungÃ¼ltige Datumswerte (z.B. "00.00.0000" oder Jahr/Tag/Monat="0") speicherbar.
  + Ab MY!5.6.4 auch auf Mikrosekunden .nnnnnn genau: DATETIME(<Fsp>)
    (Fsp = Fractional seconds part = 0-6 Stellen als zusÃ¤tzliche Angabe)

* TIMESTAMP: Wert als Anzahl Sekunden seit 1.1.1970 00:00:00 gespeichert,
  + Zeitraum: 01.01.1970 00:00:01 ... 19.01.2038 03:14:07 (UTC)
  + UNIX-Zeitrechnung mit 32-Bit Zahl mit Vorzeichen --> Y2K-Problem!
  + UnabhÃ¤ngig vom SQL-Modus nur gÃ¼ltige Datumswerte mÃ¶glich
    (kein "00.00.0000" oder Tag/Monat="0").
  + Ab MY!5.6.4 auch auf Mikrosekunden .nnnnnn genau: TIMESTAMP(<Fsp>)
    (Fsp = Fractional seconds part = 0-6 Stellen als zusÃ¤tzliche Angabe)

  Eine TIMESTAMP-Angabe hat die Form YYYYMMDDhhmmss.nnnnnn (Zahl oder "String")
  oder "YYYY-MM-DD hh:mm:ss.nnnnnn" ("String"), eine TIMESTAMP-Ausgabe hat je
  nach MySQL-Version unterschiedliche Form:

    YYYYMMDDhhmmss.nnnnnn           # Alt (vor MY!5.4)
    "YYYY-MM-DD hh:mm:ss.nnnnnn"    # Neu (seit MY!5.4)

  ACHTUNG: Der Wert der 1. TIMESTAMP-Spalte in einer Tabelle wird bei JEDER
  Ã„nderung eines Datensatzes auf aktuelle(s) Datum + Uhrzeit CURRENT_TIMESTAMP()
  gesetzt (weitere TIMESTAMP-Spalten bleiben unverÃ¤ndert). Soll diese Spalte
  bei Ã„nderungen des Datensatzes unverÃ¤ndert bleiben, ist ihr explizit der
  aktuelle Wert zuzuweisen:

    UPDATE ... SET ..., ts = ts;

  ACHTUNG: Ohne Angabe des Attributs NULL/NOT NULL erlauben alle Datentypen den
  Wert NULL auÃ„er dem Datentyp TIMESTAMP. Hier ist explizit das Attribut NULL
  anzugeben, wenn der NULL-Wert erlaubt sein soll.

  LÃ¤sst man Teile am Ende der Zeitangabe YYYYMMDDhhmmss / "YYYY-MM-DD hh:mm:ss"
  weg, dann wird der KLEINSTE mÃ¶gliche Wert dafÃ¼r eingesetzt. Die Bedingungen
  (a) bis (f) lassen also am rechten Rand des Zeitraums einiges weg, was man
  zunÃ¤chst so nicht erwartet. Erst Bedingung (7) umfasst den beabsichtigten
  Zeitraum vollstÃ¤ndig:
                                              # 1.1.12 00:00:00 ..
    ts BETWEEN 2012           AND 2013        # ..  1.1.13 00:00:00 (a)
    ts BETWEEN 201201         AND 201301      # ..  1.1.13 00:00:00 (b)
    ts BETWEEN 20120101       AND 20130131    # .. 31.1.13 00:00:00 (c)
    ts BETWEEN 20120101000000 AND 20130131000000  # .. 31.1.13 00:00:00 (d)
    ts BETWEEN 2012010100     AND 2013013123  # .. 31.1.13 23:00:00 (e)
    ts BETWEEN 201201010000   AND 201301312359    # .. 31.1.13 23:59:00 (f)
    ts BETWEEN 20120101000000 AND 20130131235959  # .. 31.1.13 23:59:59 (g)

  HINWEIS: Alle Elemente in Zeitangaben der Form YYYYMMDDhhmmss / "YYYY-MM-DD
  hh:mm:ss" sind mit 4 (Jahr) bzw. 2 Ziffern (sonst) anzugeben. Es ist
  unzulÃ¤ssig, die fÃ¼hrende Null bei einstelligem Tag, Monat, Stunde, Minute
  und Sekunde wegzulassen.

* CHAR(n) speichert n Zeichen mit Leerzeichen aufgefÃ¼llt (SPACE-padded),
```

```
          beim Lesen werden die Leerzeichen ENTFERNT (stripped).
  BINARY(n) speichert n Byte mit NUL-Bytes aufgefÃ¼llt (NUL-padded),
          beim Lesen werden die NUL-Bytes NICHT ENTFERNT (not stripped).

* VARCHAR(n) speichert 0-n Zeichen PLUS eine LÃ¤nge (1/2 Byte).
  VARBINARY(n) speichert 0-n Byte PLUS eine LÃ¤nge (1/2 Byte).

* ACHTUNG: CHAR, VARCHAR und TEXT ignorieren GROSS/kleinschreibung beim
  Vergleichen und Sortieren (case-INsensitive).
  BINARY, VARBINARY und BLOB berÃ¼cksichtigen GROSS/kleinschreibung beim
  Vergleichen und Sortieren (case-sensitive).

* TEXT/BLOB: Ein BLOB ist ein "Binary Large OBject" der LÃ¤nge 0-4 Mrd Byte.
  Nicht im Datensatz gespeichert, dort steht nur "Zeiger" der GrÃ¶Ã-^_e L Byte.
  Die Daten selber stehen pro Objekt an einer anderen Stelle.
  + TEXT: Textdaten, Vergleich und Sortierung erfolgt anhand Zeichensatz und
          Collation (ohne BerÃ¼cksichtigung der GROSS/kleinschreibung).
  + BLOB: BinÃ¤rdaten, Vergleich und Sortierung erfolgt anhand der Byte-Codes.

  Entsprechen VARCHAR/VARBINARY mit folgenden Unterschieden:
  + TEXT/BLOB-Spalten in Indices MÃM-^\SSEN eine PrÃ¤fix-LÃ¤ngenangabe haben.
  + TEXT/BLOB-Spalten kÃ¶nnen KEINE Defaultwerte haben.
  + Speicherung erfolgt nicht im Datensatz sondern auÃM-^_erhalb,
    im Datensatz steht nur ein Zeiger.

* ENUM: Wie Datentyp (VAR)CHAR verwendbar, der String wird aber als ZAHL
  gespeichert. Als Werte sind nur Elemente einer vordefinierten Werteliste
  erlaubt, anstelle des Elements wird platzsparend seine Nummer gespeichert.
  Ein String, der nicht in der Werteliste enthalten ist, entspricht NULL.
  Auch Index 1..N gemÃ¤Ã-^_ Werteliste speicherbar (1="Elem1", 2="Elem2", ...).

* SET: Wie Datentyp (VAR)CHAR verwendbar, eine Kombination von Strings wird
  als BITMUSTER gespeichert. Als Wert ist eine beliebige Kombination ("Menge")
  der Werte aus einer vordefinierten Werteliste (max. 64 Werte) erlaubt (auch
  die leere Menge ""). Anstelle der Element-Kombination wird platzsparend ein
  Bitmuster gespeichert, jede Bitposition entspricht einem Element der Menge.
  + Bit 1 an Bitposition N heiÃM-^_t, das N-te Elem. befindet sich in der Menge,
  + Bit 0 an Bitposition N heiÃM-^_t, das N-te Elem. befindet sich NICHT in der M.

* BIT: Kann eine Menge von 1..64 Bit (1-8 Byte) aufnehmen. Jedes Bit ist
  entweder gesetzt (Wert 1) oder nicht gesetzt (Wert 0).


4a) Datentyp-Optimierung (Performance/Speicherplatz)
----------------------------------------------------

* KleinstmÃ¶glichen Datentyp nutzen (z.B. MEDIUMINT statt INT)
  --> Weniger Festplattenplatz belegt --> Weniger Platten-I/O --> Schneller!
  ABER: Zukunftssicher auslegen (Y2K-Problem, IPv4 --> IPv6,
        UNIX-Timestamp endet am 19.01.2038)!

* Spalten mÃ¶glichst "NOT NULL" deklarieren!
  --> Weniger Platzverbrauch (NULL benÃ¶tigt zusÃ¤tzl. Byte pro Spalte)
  --> Index schneller!

* Mind. eine VARCHAR/VARBINARY/TEXT/BLOB-Spalte --> VARIABLE Record-LÃ¤nge
  Nur CHAR/BINARY-Spalten                    --> FIXE     Record-LÃ¤nge
  --> 4b) Fixe/Variable Record-LÃ¤nge (Row-Format)

* In SELECT keine Spalten-Konvertierung in der WHERE-Bedingung durchfÃ¼hren
  --> Index sonst nicht nutzbar
  --> FÃ¼r jeden Satz notwendig --> langsam!
  ACHTUNG: Findet implizit statt, falls mit Textspalte GERECHNET wird!

* In JOIN verwendete Spalten sollten exakt gleichen Datentyp + LÃ¤nge haben,
  --> Index nutzbar
  --> Bei (VAR)CHAR/(VAR)BINARY/TEXT/BLOB unterschiedliche LÃ¤nge toleriert

* MySQL-Optimierer macht selbstÃ¤ndig folgende Ã-^Dnderungen bei CREATE/ALTER TABLE
  (Speicherplatz minimal, Tabellenstruktur optimiert, autom. Konvertierung)

  +------------------+-------------------------------------------------------+
  | Datentyp         | Automatische Ã-^Dnderung                               |
  +------------------+-------------------------------------------------------+
  | VARCHAR(N<=4)    | CHAR(N)                                               |
  | VARBINARY(N<=4)  | BINARY(N)                                             |
  | CHAR(N>4)        | VARCHAR(N) (falls mind. eine VARCHAR-Spalte vorhanden)|
  | TIMESTAMP(N)     | TIMESTAMP (2 <= N geradzahlig <= 14)                  |
  | PRIMARY KEY      | Spalten erhalten Attribut NOT NULL                    |
  | YEAR(2)          | YEAR(4)                                               |
  +------------------+-------------------------------------------------------+
```

```
4b) Fixe/Variable Record-LÃ¤nge (Row-Format)
-------------------------------------------

* DatensÃ¤tze (Rows) kÃ¶nnen max. 65535 Byte lang sein
  (abhÃ¤ngig von Engine auch weniger) --> 26) Begrenzungen von MySQL (Limits)

* FIXES Rowformat (feste LÃ¤nge): ALLE Spalten NICHT von folgenden 4 Datentypen.
  VARIABLES Rowformat (unterschiedl. LÃ¤nge): Mind. EINE Spalte dieses Datentyps:

    VARCHAR   > 4 Zeichen
    VARBINARY > 4 Zeichen
    ...BLOB
    ...TEXT

* FIXES Rowformat hat Vorteile bei der Adressierung der DatensÃ¤tze,
  nutzt aber evtl. Plattenplatz schlechter (wg. FÃ¼llbyte)

* VARIABLES Rowformat nutzt Plattenplatz besser aus,
  benÃ¶tigt aber weitere Bytes (<=255 --> 1, >255 --> 2) zur LÃ¤ngeninformation

* Bei vielen LÃ¶sch/EinfÃ¼ge-Operationen in Tabelle fÃ¼hrt variables
  Rowformat leichter zur "Fragmentierung" (Zersplitterung) der Tabellendaten
  --> OPTIMIZE TABLE durchfÃ¼hren

* FÃ¼r Table Scan und Zugriff per Index ist fixes/variables Rowformat egal!

* Besondere Einstellungen bei Tabellen-Definition
  + MyISAM: ROW_FORMAT=FIXED/DYNAMIC (myisampack + myisamchk -rq --> COMPRESSED)
  + InnoDB: ROW_FORMAT=REDUNDANT/COMPACT (20% weniger Platz, CPU erhÃ¶ht)

* TIP: Breite Tabelle in mehrere Teile aufsplitten (fixer + variabler Anteil)


4c) AUTO_INCREMENT
------------------

* Jedem Datensatz in Tabelle automatisch eindeutigen Key zuordnen
  + "KÃ¼nstlicher" SchlÃ¼ssel (von Datenbank generiert)
  + Eindeutige positive Zahl (negativ --> GroÃM-^_e positive Zahl)
    - Startwert beim Anlegen der Tabelle wÃ¤hlbar (STD: 1):
        CREATE TABLE ... (...) ... AUTO_INCREMENT = <N> ...
        ALTER TABLE <Tbl> ... AUTO_INCREMENT = <N>;
    - Alternativ Zeile mit Wert N-1 einfÃ¼gen und wieder lÃ¶schen

* Nur in EINER Spalte pro Tabelle erlaubt
  + Spalten-Datentyp muss INT, FLOAT oder DOUBLE sein
    (DECIMAL nicht mÃ¶glich, UNSIGNED schon)
  + Index MUSS darauf liegen (nicht unbedingt UNIQUE, aber sinnvoll;
                              PRIMARY KEY am sinnvollsten)
  + Keine DEFAULT-Angabe mÃ¶glich

* EinfÃ¼gen von NULL/0 in diese Spalte setzt nÃ¤chsten noch nicht benutzten Wert
  + SQL-Modus "NO_AUTO_VALUE_ON_ZERO": 0 als Wert mÃ¶glich (nicht empfohlen!)
  + Spaltenwert angegeben
    - Schon vorhanden      --> EinfÃ¼gen wird verweigert
    - Noch nicht vorhanden --> EingefÃ¼gt + nÃ¤chster automatisch ermittelter
                               um 1 hÃ¶her (LÃ¼cken entstehen!)

* Letzten verwendeten AUTO_INCREMENT-Wert direkt nach INSERT abfragen
  (nur in derselben Sitzung/Transaktion):

    SELECT LAST_INSERT_ID();

* HINWEIS: Damit Master-Master-Replikation mÃ¶glich ist (2 gegenseitige Master
  oder Ring aus mehreren Mastern), mÃ¼ssen AUTO_INCREMENT-Spalten auf allen
  Mastern garantiert unterschiedliche Werte erzeugen. Dazu gibt es im
  MySQL-Server folgende Optionen:

    auto_increment_offset    = <N>   # Startwert von AUTO_INCREMENT-Spalten
    auto_increment_increment = <N>   # Schrittweite von AUTO_INCREMENT-Spalten

  Ist die Anzahl der sich gegenseitig replizierenden Master z.B. 5, dann
  sollten die einzelnen Master verschiedene Offsets 1, 2, 3, 4, 5 erhalten
  und bei allen Mastern die Schrittweite auf 5 gesetzt werden.

* Pro Tabelle ist ein eigener Startwert fÃ¼r AUTO_INCREMENT mÃ¶glich,
  aber keine Schrittweite (diese ist nur pro Server einstellbar).

* HINWEISE:
  + In PostgreSQL dafÃ¼r Datentyp "SERIAL" verwenden
  + In Oracle durch "Sequence"-Objekt nachbilden (deutlich komplexer)
```

```
Beispiel:

  CREATE TABLE pers (
    nr      INT NOT NULL AUTO_INCREMENT,      # Spalte "nr" automatisch fÃ¼llen
    vorname VARCHAR(30),                      #
    name    VARCHAR(30),                      #
    PRIMARY KEY (nr)                          # Wg. AUTO_INCREMENT nÃ¶tig
  ) AUTO_INCREMENT = 10;                      # Werte 1..9 Ã¼berspringen
                                              #
  INSERT INTO pers (nr, vorname, name)        #
    VALUES (NULL, "Thomas",  "Birnthaler"),   # --> nr=10 (LÃ¼cke von 1..9)
           (20,   "Markus",  "Mueller");      # --> nr=20 (LÃ¼cke von 11..19)
                                              #
  INSERT INTO pers (vorname, name)            #
    VALUES ("Andrea",  "Bayer"),              # --> nr=21
           ("Richard", "Seiler"),             # --> nr=22
           ("Heinz",   "Bayer");              # --> nr=23
                                              #
  INSERT INTO pers (nr, vorname, name)        #
    VALUES (20, "Hans",  "Birnthaler");       # --> verweigert


5) MySQL-Operatoren
-------------------

Folgende Operatoren sind als VerknÃ¼pfung in AusdrÃ¼cken (Expressions) verwendbar:

  +------------+-----------------------------------+-------------------------+
  | Typ        | Beispiele                         | Bemerkung               |
  +------------+-----------------------------------+-------------------------+
  | Arithmetik | +  -  *  /  DIV  %  MOD           | MOD/% = Modulo (Div.rest)|
  +------------+-----------------------------------+-------------------------+
  | Vergleich  | <  <=  >  >=                      | Nicht => und <=!        |
  |            | =  !=  <>  <=>                    | <=> ist NULL-sicheres = |
  |            | <Val> BETWEEN <Left> AND <Right>  | RÃ¤nder einschlieÃM-^_en     |
  |            | <Val> NOT BETWEEN <Left> AND <R.> | RÃ¤nder nicht einschl.   |
  |            | <Val> IN (<Val1>, <Val2>, ...)    | In Liste enthalten      |
  |            | <Val> NOT IN (<Val1>, <Val2>, ...)| Nicht in Liste enthalten|
  |            | <Val> LIKE <Muster> [ESCAPE <Char>]| Wildcards "_" "%" (s.u.)|
  |            | <Val> NOT LIKE <Muster>           | Wildcards "_" "%" (s.u.)|
  |            | <Val> REGEXP/RLIKE <RegEx>        | RegulÃ¤rer Ausdruck (s.u.)|
  |            | <Val> NOT REGEXP/RLIKE <RegEx>    | RegulÃ¤rer Ausdruck (s.u.)|
  |            | <Val1> SOUNDS LIKE <Val2>         | SOUNDEX(v1) = SOUNDEX(v2)|
  |            | <Val> IS NULL/UNKNOWN             | Gleich NULL/Unbekannt   |
  |            | <Val> IS NOT NULL/UNKNOWN         | Ungleich NULL/Unbekannt |
  |            | <Val> IS TRUE/FALSE               | Gleich TRUE/FALSE       |
  |            | <Val> IS NOT TRUE/FALSE           | Ungleich TRUE/FALSE     |
  +------------+-----------------------------------+-------------------------+
  | Logisch    | NOT  !                            | Erg: Nur TRUE/FALSE     |
  |            | OR  ||                            | Erg: Nur TRUE/FALSE     |
  |            | AND  &&                           | Erg: Nur TRUE/FALSE     |
  |            | XOR                               | Erg: Nur TRUE/FALSE     |
  +------------+-----------------------------------+-------------------------+
  | Bit        | |  &  ^  ~  <<  >>                | OR AND XOR INV L/R-SHIFT|
  +------------+-----------------------------------+-------------------------+
  | Zeichen    | BINARY ...                        | String binÃ¤r interpret. |
  |            | _CHARSET                          | String-Zeichensatz      |
  |            | COLLATE                           | String-Sortierordnung   |
  +------------+-----------------------------------+-------------------------+

Beispiele (Rechnen):

  SELECT 1 + 2 * 3 / 4 - 5;                                 # --> -2.5
  SELECT 17 / 3, 17 DIV 3, 17 % 3, 17 MOD 3;                # --> 5.6667 5 2 2

Beispiele (Vergleich):

  SELECT 10 < 4, 2 <= 5, 1 = 3, 1 <> 3;                     # --> 0 1 0 1
  SELECT 10 > 4, 2 >= 5, 1 <=> 3, 1 != 3;                   # --> 1 0 0 1
  SELECT 0 <=> 0, 0 <=> NULL, NULL <=> 0, NULL <=> NULL;    # --> 1 0 0 1
  SELECT 10 BETWEEN 0 AND 12, 10 NOT BETWEEN 0 AND 12;      # --> 1 0
  SELECT 10 IN (1, 2, 10), 10 NOT IN (1, 2, 10);            # --> 1 0
  SELECT "abc" LIKE "%b%", "abc" NOT LIKE "%b%";            # --> 1 0
  SELECT "abc" RLIKE "b", "abc" NOT RLIKE "b";              # --> 1 0
  SELECT "abc" SOUNDS LIKE "ebz";                           # --> 0
  SELECT NULL IS NULL, NULL IS NOT NULL;                    # --> 1 0
  SELECT TRUE IS TRUE, TRUE IS FALSE;                       # --> 1 0
  SELECT TRUE IS NOT TRUE, TRUE IS NOT FALSE;               # --> 0 1

Beispiele (Logische VerknÃ¼pfung):
```

```
  SELECT TRUE AND FALSE OR FALSE AND NOT TRUE;              # --> 0
  SELECT TRUE && FALSE  || FALSE &&   !  TRUE;              # --> 0
  SELECT TRUE XOR FALSE, TRUE XOR TRUE, FALSE XOR TRUE;     # --> 1 0 1

Beispiele (Bitweise VerknÃ¼pfung):

  SELECT BIN(b'11110001' | b'00111100');                    # --> 11111101 (253)
  SELECT BIN(b'11110001' & b'00111100');                    # --> 00110000  (48)
  SELECT BIN(b'11110001' ^ b'00111100');                    # --> 11001101 (205)
  SELECT BIN(~ b'00110101');                                # --> 11..1001010
  SELECT BIN(~ b'00110101' & b'11111111');                  # --> 11001010 (202)
  SELECT BIN(b'00110101' >> 3);                             # --> 00000110   (6)
  SELECT BIN(b'00110101' << 2);                             # --> 11010100 (212)
  SELECT BIN(~(b'00110101' & b'00001111'));                 # --> 11..111010
  SELECT BIN(~(b'00110101' | b'00001111') ^ b'00110011');   # --> 11..110011

Hinweise:
  * Division "/" ist immer FlieÃM-^_kommadivision,
    Division "DIV" schneidet den Divisionsrest ab.
  * StandardmÃ¤Ã-^_ig gibt es keinen Stringverkettungs-Operator (z.B. &  +  .  ||)
    --> CONCAT verwenden, damit sind beliebig viele Strings verkettbar
        (SET GLOBAL sql_mode = 'PIPES_AS_CONCAT' oder
         SET GLOBAL sql_mode = 'ANSI' --> Operator "||" verkettet Strings)
      CONCAT("abc", <Col>, "xyz")   # Immer mÃ¶glich
      "abc" || <Col> || "xyz"       # Falls sql_mode='PIPES_AS_CONCAT/ANSI'
  * Falls in Stringverkettung NULL vorkommen kann --> Auf "" abbilden:
      CONCAT("abc", IF(ISNULL(<Col>), "", <Col>), "xyz")
      CONCAT("abc", IFNULL(<Col>, ""), "xyz")
  * BINARY vor String in Vergleichen = != <> <=> < <= > >= BETWEEN...AND IN LIKE
    REGEX und in ORDER BY vor Spalte erzwingt BerÃ¼ck. der GROSS/kleinschreibung:
      SELECT * FROM pers WHERE BINARY name = "abc";
      SELECT * FROM pers WHERE BINARY name != "abc";
      SELECT * FROM pers WHERE BINARY name > "abc";
      SELECT * FROM pers WHERE BINARY name <= "abc";
      SELECT * FROM pers WHERE BINARY name BETWEEN "abc" AND "def";
      SELECT * FROM pers WHERE BINARY name LIKE "abc%";
      SELECT * FROM pers WHERE BINARY name REGEX "^abc$";
      SELECT * FROM pers ORDER BY BINARY name ASC;
  * LIKE-Wildcards fÃ¼r beliebige/beliebig viele Zeichen (<Muster>):
      "_" = GENAU EIN beliebiges Zeichen
      "%" = Beliebig viele beliebige Zeichen (auch 0!)
  * ESCAPE <Char> am Ende von LIKE setzt Entwertungs-Zeichen von "%" und "_"
    auf das Zeichen <Char> (STD: "\"). D.h. um mit LIKE nach den Zeichen "%"
    und "_" selbst zu suchen, muss man dieses Escape-Zeichen davor setzen:
      SELECT * FROM test WHERE name LIKE "\_\%";              # Standard "\"
      SELECT * FROM test WHERE name LIKE "@_@%" ESCAPE "@";   # Escape = "@"

Hinweise zu NULL:
  * NULL kommt in Ausdruck vor  --> Ergebnis NULL ("schwarzes Loch")
  * NULL kommt in Vergleich vor --> Ergebnis NULL (auÃM-^_er bei <=>)
  * Logische AusdrÃ¼cke verkÃ¼rzt ausgewertet (Short-Cut/Circuit Evaluation):
    + FALSE AND ... --> sofort FALSE (ohne Auswertung von ...)
    + TRUE OR ...   --> sofort TRUE  (ohne Auswertung von ...)
    D.h. hat ... Wert NULL, ist das Gesamtergebnis trotzdem FALSE bzw. TRUE!
  * Ist NULL das Ergebnis einer WHERE-Bedingung, entspricht dies dem Wert FALSE.
  * IS UNKNOWN analog IS NULL (andere Schreibweise, bei Bool. Werten Ã¼blich):
      SELECT 0 IS UNKNOWN, 1 IS UNKNOWN, NULL IS UNKNOWN;   # --> 0 0 1
      SELECT 0 IS NULL,    1 IS NULL,    NULL IS NULL;      # --> 0 0 1
  * "<=>" ist NULL-sicherer Vergleich auf gleichen Wert (Ersatz fÃ¼r "="):
      SELECT NULL <=> NULL;         # --> 1  (bei "=" --> NULL)
      SELECT NULL <=> TRUE;         # --> 0  (bei "=" --> NULL)
      SELECT NULL <=> FALSE;        # --> 0  (bei "=" --> NULL)
      SELECT NULL <=> "abc";        # --> 0  (bei "=" --> NULL)
      SELECT NULL <=> "";           # --> 0  (bei "=" --> NULL)
      SELECT NULL <=> "NULL";       # --> 0  (bei "=" --> NULL)
    Verhalten identisch zu "=" falls NULL nicht als VerknÃ¼pfungswert vorkommt.

Rangfolge/PrioritÃ¤t der Operatoren (durch Klammerung "(...)" Ã¤ndern):

  +----+--------------------------------+--------------------------------------+
  | Nr | Typ                            | Bemerkung                            |
  +----+--------------------------------+--------------------------------------+
  |  1 | BINARY                         | GROSS/kleinschreibung beachten       |
  |    | _CHARSET  COLLATE              | Zeichensatz + Sortierung anpassen    |
  |  2 | !  (NOT)                       | Negation (falls HIGH_NOT_PRECEDENCE) |
  |  3 | -  ~                           | UnÃ¤res Minus, Bitweise Invertierung  |
  |  4 | ||                             | Stringverkettung (falls              |
  |    |                                |   sql_mode='PIPES_AS_CONCAT/ANSI')   |
  |  5 | ^                              | Bitweise XOR                         |
  |  6 | *  /  DIV  %  MOD              | MOD/% = Modulo, DIV = ganzz. Div.    |
  |  7 | +  -                           | Addition/Subtraktion                 |
  |  9 | <<  >>                         | Bitweise Shift links/rechts          |
```

9	&	Bitweise AND
10	|	Bitweise OR
11	= != <> <=> < <= > >=	Vergleich
	IN IS LIKE REGEXP	Vergleich
12	BETWEEN ... AND ...	Bereich
	CASE ... WHEN ... THEN ... ELSE	Fallunterscheidung
13	NOT	Logische Negation
		(ab MY!5.0.2 vom Vorrang her hier!)
14	AND &&	Logisch AND
15	XOR	Logisch XOR
16	OR ||	Logisch OR (falls sql_mode="...")
17	:=	Zuweisung

HINWEIS: Der Operator NOT hatte vor MY!5.0.2 die gleiche (hohe) PrioritÃ¤t wie der Operator !, seither hat er eine PrioritÃ¤t zwischen BETWEEN und AND. Die alte (hohe) PrioritÃ¤t erhÃ¤lt man mit folgendem SQL-Modus:

```
  SET GLOBAL sql_mode = 'HIGH_NOT_PRECENDENCE';
```

HINWEIS: Falls der OR-Operator "||" Strings verketten soll (ANSI-SQL Standard):

```
  SET GLOBAL sql_mode = 'PIPES_AS_CONCAT';   # Alternative 1
  SET GLOBAL sql_mode = 'ANSI';              # Alternative 2
```

6) Boolesche Logik

In Boolescher Logik gibt es nur die beiden Konstanten TRUE und FALSE mit dem numerischen Wert 1 und 0. Umgekehrt werden alle numerischen Werte ungleich 0 auf TRUE abgebildet und der numerische Wert 0 auf FALSE. Strings im Booleschen Kontext werden in Zahlen umgewandelt, d.h. alle Strings mit Ergebnis "0" sind FALSE, die anderen sind TRUE:

Logische AusdrÃ¼cke ergeben 1 fÃ¼r TRUE und 0 fÃ¼r FALSE

```
  SELECT TRUE, FALSE, NOT TRUE, NOT FALSE;              # --> 1 0 0 1
  SELECT NOT 0, NOT 1, NOT 123, NOT -456, NOT 7.89;     # --> 1 0 0 0 0
  SELECT NOT "", NOT "abc", NOT "123def", NOT "0def";   # --> 1 1 0 1
```

Rechnen mit TRUE/FALSE entspricht Rechnen mit 0/1:

```
  SELECT TRUE / FALSE;   # --> NULL (Division durch 0!)
  SELECT FALSE / TRUE;   # --> 0.0000
  SELECT TRUE * FALSE;   # --> 0
  SELECT TRUE + FALSE;   # --> 1
  SELECT TRUE - FALSE;   # --> 1
  SELECT FALSE - TRUE;   # --> -1
```

Das Ergebnis eines Booleschen Ausdrucks kann bereits bei Betrachtung eines TEILS der damit verknÃ¼pften AusdrÃ¼cke feststehen, egal welchen Wert die restlichen verknÃ¼pften AusdrÃ¼cke haben (Short-Cut/Short-Circuit Evaluation = VerkÃ¼rzte Auswertung):

```
  <Expr1> AND <Expr2>   # <Expr2> nur ausgewertet, falls <Expr1> TRUE ergibt
  <Expr1> OR  <Expr2>   # <Expr2> nur ausgewertet, falls <Expr1> FALSE ergibt
```

Beispiel (SQRT(-1) ist nicht definiert):

```
  SELECT 0 AND SQRT(-1);      # --> 0    (obwohl SQRT(-1) undefiniert)
  SELECT 1 AND SQRT(-1);      # --> NULL (da     SQRT(-1) undefiniert)
  SELECT 1 OR SQRT(-1);       # --> 1    (obwohl SQRT(-1) undefiniert)
  SELECT 0 OR SQRT(-1);       # --> NULL (da     SQRT(-1) undefiniert)
```

7) Der Wert NULL

7a) Eigenschaften von NULL

* NULL steht fÃ¼r Wert "unbekannt" oder "undefiniert"
* NULL ist weder die Zahl 0 noch der leere String "",
  noch TRUE oder FALSE, sondern etwas vÃ¶llig anderes.
* Fast alle Operationen mit NULL ergeben NULL ("Schwarzes Loch")
  + Ausnahmen: <=>, AND, OR, IS NULL, IS NOT NULL,
               ISNULL(), IFNULL(), NULLIF(), COALESCE()
               SUM(), AVG(), COUNT(), ...
* BenÃ¶tigt extra Speicherplatz in Tabelle (pro NULL-Spalte 1 Byte oder 1 Bit)
  --> Spalten mÃ¶glichst mit NOT NULL DEFAULT <Wert> kennzeichnen (STD: NULL)
* NULL-sicheren Vergleich mit <=> statt = durchfuehren (MY!)

```
  SELECT 1 <=> NULL, NULL <=> NULL, 0 <=> NULL, "" <=> NULL;    # --> 0 1 0 0
  SELECT 1 = NULL, NULL = NULL, 0 = NULL, "" = NULL;            # --> 4x NULL
  SELECT NULL IS NULL, 0 IS NULL, 1 IS NULL, "" IS NULL;        # --> 1 0 0 0
  SELECT NULL IS NOT NULL, 0 IS NOT NULL, 1 IS NOT NULL,        #
         "" IS NOT NULL;                                        # --> 0 1 1 1
  SELECT ISNULL(NULL), ISNULL(0), ISNULL(1), ISNULL("");        # --> 1 0 0 0

Kommt Wert NULL in einem Ausdruck vor, so hat in der Regel der GESAMTE Ausdruck
als Ergebnis den Wert NULL:

  SELECT 3 * 4 - 1 + NULL AS "Wert";                            # --> NULL
  SELECT 1 = NULL, 1 != NULL, 1 < NULL, 1 > NULL;               # --> 4x NULL
  SELECT NULL = 1, NULL = NULL, 0 = NULL;                       # --> 3x NULL
  SELECT NULL != 1, NULL != NULL, 0 != NULL;                    # --> 3x NULL
  SELECT CONCAT(NULL, "abc", "def", NULL, "123");               # --> NULL
  SELECT CONCAT("abc", "def", NULL, "123");                     # --> NULL


7b) PrÃ¼fung auf Wert NULL
-------------------------

Zur expliziten PrÃ¼fung auf den Wert NULL gibt es folgende MÃ¶glichkeiten:

  +---------------------------------+-------------------------------------------+
  | Ausdruck                        | Bedeutung                                 |
  +---------------------------------+-------------------------------------------+
  | <Col> IS NULL                   | In WHERE-Bedingung                        |
  | <Col> IS NOT NULL               | In WHERE-Bedingung                        |
  | <Expr> IS NULL                  | In Ausdruck                               |
  | <Expr> IS NOT NULL              | In Ausdruck                               |
  | ISNULL(<Expr>)                  | 1 wenn <Expr> = NULL, sonst 0             |
  | IFNULL(<Expr1>, <Expr2>)        | <Expr1> wenn nicht NULL, sonst <Expr2>    |
  | IF(ISNULL(<E1>), <E2>, <E1>))   |   (analog)                                |
  | NULLIF(<Expr1>, <Expr2>)        | NULL wenn <Expr1>=<Expr2>, sonst <Expr1>  |
  | COALESCE(<Expr1>, <Expr2>, ...) | Erster Ausdruck <Expr_i> != NULL          |
  +---------------------------------+-------------------------------------------+

Beispiel:

  SELECT NULL IS NULL, NULL IS NOT NULL;                    # --> 1 0
  SELECT 0 IS NULL, 0 IS NOT NULL;                          # --> 0 1
  SELECT 1 IS NULL, 1 IS NOT NULL;                          # --> 0 1
  SELECT "" IS NULL, "" IS NOT NULL;                        # --> 0 1
  SELECT ISNULL(NULL), ISNULL(0), ISNULL(1), ISNULL("");    # --> 1 0 0 0
  SELECT IFNULL("eins", NULL), IFNULL(NULL, "zwei");        # --> eins zwei
  SELECT NULLIF("eins", "eins"), NULLIF("eins", "zwei");    # --> NULL eins
  SELECT COALESCE(NULL, NULL, "abc", NULL, 123);            # --> abc


7c) Vergleiche mit NULL
-----------------------

Vergleiche ergeben einen Booleschen Wert auÃM-^_er mit NULL wird verglichen,
dann resultiert der Wert NULL als Ergebnis (auÃM-^_er bei "<=>").

Der normale Vergleich "=" liefert daher:

  +-------+-------+-------+-------+
  |   =   | TRUE  | FALSE | NULL  |   # Ein NULL bei "=" ergibt insgesamt NULL
  +-------+-------+-------+-------+
  | TRUE  | TRUE  | FALSE | NULL  |
  | FALSE | FALSE | TRUE  | NULL  |
  | NULL  | NULL  | NULL  | NULL  |
  +-------+-------+-------+-------+

Ebenso gilt fÃ¼r den Vergleich "!=" (oder "<>"):

  +-------+-------+-------+-------+
  |  !=   | TRUE  | FALSE | NULL  |   # Ein NULL bei "!=" ergibt insgesamt NULL
  +-------+-------+-------+-------+
  | TRUE  | FALSE | TRUE  | NULL  |
  | FALSE | TRUE  | FALSE | NULL  |
  | NULL  | NULL  | NULL  | NULL  |
  +-------+-------+-------+-------+

Die anderen Vergleiche <Cmp> ("<", "<=", ">" und ">=") ergeben:

  +-------+------------+-------+
  | <Cmp> | !NULL      | NULL  |   # Ein NULL bei <Cmp> ergibt insgesamt NULL
  +-------+------------+-------+
  | !NULL | TRUE/FALSE | NULL  |
```

```
  | NULL  | NULL       | NULL  |
  +-------+------------+-------+

Der NULL-sichere Vergleich "<=>" liefert hingegen (MY!):

  +-------+-------+-------+-------+
  |  <=>  | TRUE  | FALSE | NULL  |
  +-------+-------+-------+-------+
  | TRUE  | TRUE  | FALSE | FALSE |   # NULL <=> TRUE  ergibt FALSE
  | FALSE | FALSE | TRUE  | FALSE |   # NULL <=> FALSE ergibt FALSE
  | NULL  | FALSE | FALSE | TRUE  |   # NULL <=> NULL  ergibt TRUE
  +-------+-------+-------+-------+


7d) Boolesche Logik mit NULL (ternÃ¤r/dreiwertig)
------------------------------------------------

Die Booleschen Operatoren AND, &&, OR, ||, XOR, NOT und ! werden auf die
VerknÃ¼pfung mit NULL-Werten erweitert (ternÃ¤re oder dreiwertige Logik).
Aufgrund der "verkÃ¼rzten Auswertung" von AND und OR gilt:

  * FALSE links von AND ergibt sofort FALSE (egal was rechts davon steht)
  * TRUE  links von OR  ergibt sofort TRUE  (egal was rechts davon steht)
  * FALSE in AND-VerknÃ¼pfung ergibt FALSE   (egal ob NULL-Werte verknÃ¼pft)
  * TRUE  in OR-VerknÃ¼pfung  ergibt TRUE    (egal ob NULL-Werte verknÃ¼pft)
  * NULL  in XOR-VerknÃ¼pfung ergibt NULL    (egal was sonst verknÃ¼pft)

Hier die dreiwertigen Auswertungstabellen der Booleschen Operatoren:

  +-------+-------+-------+-------+
  | AND &&| TRUE  | FALSE | NULL  |  # Ein FALSE bei AND ergibt insgesamt FALSE!
  +-------+-------+-------+-------+
  | TRUE  | TRUE  | FALSE | NULL  |
  | FALSE | FALSE | FALSE | FALSE |
  | NULL  | NULL  | FALSE | NULL  |
  +-------+-------+-------+-------+

  +-------+-------+-------+-------+
  | OR || | TRUE  | FALSE | NULL  |  # Ein TRUE bei OR ergibt insgesamt TRUE!
  +-------+-------+-------+-------+
  | TRUE  | TRUE  | TRUE  | TRUE  |
  | FALSE | TRUE  | FALSE | NULL  |
  | NULL  | TRUE  | NULL  | NULL  |
  +-------+-------+-------+-------+

  +-------+-------+-------+-------+
  |  XOR  | TRUE  | FALSE | NULL  |  # Ein NULL bei XOR ergibt insgesamt NULL!
  +-------+-------+-------+-------+
  | TRUE  | FALSE | TRUE  | NULL  |
  | FALSE | TRUE  | FALSE | NULL  |
  | NULL  | NULL  | NULL  | NULL  |
  +-------+-------+-------+-------+

  +-------+-------+
  | NOT ! |       |  # NULL negiert ergibt NULL!
  +-------+-------+
  | TRUE  | FALSE |
  | FALSE | FALSE |
  | NULL  | NULL  |
  +-------+-------+

Beispiele:

  SELECT TRUE  AND NULL, NULL AND TRUE;    # --> NULL NULL
  SELECT FALSE AND NULL, NULL AND FALSE;   # --> 0 0
  SELECT TRUE  &&  NULL, NULL &&  TRUE;    # --> NULL NULL
  SELECT FALSE &&  NULL, NULL &&  FALSE;   # --> 0 0
  SELECT TRUE  OR  NULL, NULL OR  TRUE;    # --> 1 1
  SELECT FALSE OR  NULL, NULL OR  FALSE;   # --> NULL NULL
  SELECT TRUE  ||  NULL, NULL ||  TRUE;    # --> 1 1
  SELECT FALSE ||  NULL, NULL ||  FALSE;   # --> NULL NULL
  SELECT TRUE  XOR NULL, NULL XOR TRUE;    # --> NULL NULL
  SELECT FALSE XOR NULL, NULL XOR FALSE;   # --> NULL NULL
  SELECT NOT NULL, ! NULL;                 # --> NULL NULL


8) RegulÃ¤re AusdrÃ¼cke in MySQL (RegEx)
--------------------------------------

Eigenschaften:
  * RegulÃ¤re AusdrÃ¼cke nennt man auch "RegEx" oder "Pattern" (Muster)
    (von englisch "Regular Expressions")
```

```
  * Vergleichen wie LIKE/NOT LIKE Texte mit einem Muster --> TRUE/FALSE
    (stark erweiterte FÃ¤higkeiten gegenÃ¼ber LIKE/NOT LIKE)
  * Kennen wesentlich mehr "Metazeichen" als LIKE (kennt nur "%" und "_"),
    um ein Muster auszudrÃ¼cken, das mit dem Text verglichen wird
    + Wildcard "_" <=> RegEx "."
    + Wildcard "%" <=> RegEx ".*"
  * Operatoren: <Expr> REGEX <RegEx>
                <Expr> RLIKE <RegEx>
                <Expr> NOT REGEX <RegEx>
                <Expr> NOT RLIKE <RegEx>
  * Vergleiche mit RegulÃ¤ren AusdrÃ¼cken kÃ¶nnen KEINEN Index nutzen!
  * RegEx Muss IRGENDWO in Zeichenkette passen, damit TRUE
    (auÃM-^_er bei Verwendung von ^ bzw. $, dann am Anfang bzw. Ende)

  +--------------+-------------------------------------------------------+
  | Metazeichen  | Bedeutung                                             |
  +--------------+-------------------------------------------------------+
  | X            | Zeichen X steht fÃ¼r sich selbst (sofern kein Metazeichen) |
  | .            | 1 beliebiges Zeichen                                  |
  | [...]        | 1 Zeichen aus Zeichenmenge ... (z.B. [A-Z])           |
  | [^...]       | 1 Zeichen NICHT aus Zeichenmenge ... (z.B. [^0-9])    |
  | [[:CLASS:]]  | Zeichenklasse (siehe unten, [[ und ]] sind doppelt!)  |
  +--------------+-------------------------------------------------------+
  | ...*         | 0-N mal Zeichen/geklammerter Ausdruck ... davor (= {0,}) |
  | ...+         | 1-N mal Zeichen/geklammerter Ausdruck ... davor (= {1,}) |
  | ...?         | 0,1 mal Zeichen/geklammerter Ausdruck ... davor (= {0,1}) |
  | ...{N}       | N   mal Zeichen/geklammerter Ausdruck ... davor       |
  | ...{N,M}     | N-M mal Zeichen/geklammerter Ausdruck ... davor       |
  +--------------+-------------------------------------------------------+
  | ^            | Stringanfang                                          |
  | $            | Stringende                                            |
  | [[:<:]]      | Wortanfang (Wort = Folge von alnum + "_")             |
  | [[:>:]]      | Wortende (Wort = Folge von alnum + "_")               |
  +--------------+-------------------------------------------------------+
  | (...)        | Ausdruck ... gruppieren (fÃ¼r | * + ? {...})           |
  | ...|...      | Alternation/Alternative (ODER)                        |
  | \X           | Metazeichen X quotieren fÃ¼r ^ $ . [ ] | * + ? { } ( ) \   |
  | \n \r \t ... | Escape-Sequenz (Steuerzeichen)                        |
  +--------------+-------------------------------------------------------+

Zeichenklassen fassen Zeichen mit einer bestimmten Eigenschaft zusammen
(genauer Inhalt hÃ¤ngt von der locale-Einstellung per CHARACTER SET ab):

  +--------+-----------------------------------------------------------+
  | Klasse | Bedeutung                                                 |
  +--------+-----------------------------------------------------------+
  | alnum  | Buchstaben und Ziffern (alpha + digit)                    |
  | alpha  | Buchstaben (upper + lower)                                |
  | cntrl  | Steuerzeichen (Code 0-31)                                 |
  | blank  | Leerzeichen (Space + Tabulator)                           |
  | digit  | Ziffern (0-9)                                             |
  | empty  | Leerzeichen (Space + Tabulator)                           |
  | graph  | Druckbare Zeichen (print ohne Leerzeichen)                |
  | lower  | Kleine Buchstaben (a-z)                                   |
  | print  | Druckbare Zeichen (mit Leerzeichen)                       |
  | punct  | Interpunktionszeichen (print -- space -- alnum)           |
  | space  | Leerraum (Space, FormFeed, Newline, Carriage Return, Tabulator) |
  | upper  | GroÃM-^_e Buchstaben (A-Z)                                |
  | xdigit | Hexadezimalziffern (0-9 + a-f/A-F)                        |
  +--------+-----------------------------------------------------------+

Beispiele (keine Indices verwendbar!):

  SELECT nr, name
    FROM pers
      WHERE name REGEXP "^abc$";                  # Genau "abc"
  ... WHERE name = "abc";                         #   (analog)
  ... WHERE name REGEXP "abc$";                   # "abc" am Ende
  ... WHERE name LIKE "%abc";                     #   (analog)
  ... WHERE name REGEXP "abc";                    # "abc" enthalten
  ... WHERE name LIKE "%abc%";                    #   (analog)
  ... WHERE name NOT REGEXP "abc";                # KEIN "abc" enthalten
  ... WHERE name NOT LIKE "%abc%";                #   (analog)
  ... WHERE name RLIKE "^a*$";                    # "", "a", "aa", "aaa", ...
  ... WHERE name = "" OR name = "a" OR ...        #   (analog)
  ... WHERE name RLIKE "^a+$";                    # "a", "aa", "aaa", ...
  ... WHERE name = "a" OR name = "aa" OR ...      #   (analog)
  ... WHERE name RLIKE "thomas|hans";             # Enth. "thomas" oder "hans"
  ... WHERE name = "thomas" OR name = "hans";     #   (analog)
  ... WHERE name RLIKE "ab{1,8}a";                # "aba","abba", ...max. 8 "b"
  ... WHERE name = "aba" OR name = "abba" OR ...; #   (analog)
```

Mit LIKE, AND, OR nicht mehr formulierbar:

```
  ... WHERE name RLIKE "^[0-9]+$";         # Nur Zahl akzept. (mind. 1 Ziffer)
  ... WHERE name RLIKE "^[[:digit:]]+$";   # Nur Zahl akzept. (mind. 1 Ziffer)
```

9) MySQL-Funktionen

MySQL kennt eine groÃM-^_e Menge eingebauter Funktionen, die beliebig in SELECT-Anweisungen zum Konvertieren/VerknÃ¼pfen von Spalten sowie in WHERE-Bedingungen einsetzbar sind (auf diesen Spalten liegende Indices sind in so einem Fall nicht nutzbar).

Beispiele fÃ¼r den Aufruf von in MySQL eingebauten Funktionen:

```
  SELECT CONCAT(vorname, " ", name) AS "Name" FROM pers;
  SELECT (YEAR(CURDATE()) - YEAR(geburtsdatum)) AS "Alter" FROM age;            ###
  SELECT NOW(), ADDDATE(NOW(), INTERVAL 14 DAY);
```

Folgende Funktionen sind in MySQL vordefiniert. Diese Liste ist erweiterbar durch Laden von externen User Defined Functions (UDF) und durch in SQL programmierte benutzerdefinierte Funktionen (--> 19) Stored Routines):

Bereich	Eingebaute Funktionen + ihre Parameter

Datenbank	USER() SESSION_USER() SYSTEM_USER() # Anmeldungsuser
	CURRENT_USER() # Interner User
	DATABASE() SCHEMA()
	CONNECTION_ID() # Sitzungs-ID
	VERSION() # MySQL-Server

Queryergebnis	FOUND_ROWS() # Bei LIMIT mit SQL_CALC_FOUND_ROWS LAST_INSERT_ID() # Bei AUTO_INCREMENT ROW_COUNT() # Bei INSERT, UPDATE, DELETE, REPLACE
Arithmetik	ABS(zahl) # Betrag GREATEST(zahl1, zahl2, ...) # GrÃ¶Ã-^_te Zahl LEAST(zahl1, zahl2, ...) # Kleinste Zahl MOD(x, y) # Divisionsrest x / y RAND([seed]) # Zufallszahl 0 <= z < 1 SIGN(zahl) # Vorzeichen
Potenzierung	EXP(zahl) POW(num, exp) POWER(num, exp) SQRT(zahl)
Logarithmus	LN(zahl) LOG(zahl [, basis]) LOG2(zahl) LOG10(zahl)
Trigonometrie	ACOS(zahl) ASIN(zahl) ATAN(zahl) ATAN2(x, y) COS(zahl) COT(zahl) DEGREES(rad) PI() RADIANS(dec) SIN(zahl) TAN(zahl)
Rundung	CEILING(zahl) CEIL(zahl) FLOOR(zahl) FORMAT(zahl, dezstellen [, locale]) ROUND(zahl [, dezstellen]) TRUNCATE(zahl, dezstellen)
Bit	BIT_COUNT(zahl) BIT_LENGTH(str) EXPORT_SET(zahl, ein, aus [,trennzeichen, [numbits]]) MAKE_SET(bits, str1, str2, ...)
String	COERCIBILITY(str) COLLATION(str) INSERT(str, pos, len, neu) WEIGHT_STRING(str [AS {CHAR|BINARY} (n)] [LEVEL n] [,opt])
StringlÃ¤nge	CHAR_LENGTH(str) CHARACTER_LENGTH(str)

	LENGTH(str) CHAR/CHARACTER/OCTET_LENGTH(str)
String-verkettung	CONCAT(str1, str2, ...) CONCAT_WS(trennzeichen, str1, str2, ...)
String-wiederholung	REPEAT(str, anz) SPACE(anz)
Stringteile extrahieren	LEFT(str, len) MID(str, pos, len) --> SUBSTR(...) RIGHT(str, len) SUBSTR(str, pos) SUBSTRING(...) SUBSTR(str, pos, len) SUBSTRING(...) SUBSTR(str FROM pos FOR len) SUBSTRING(...) SUBSTR(str FROM len) SUBSTRING(...) SUBSTR_INDEX(str, zeichen, anz) SUBSTRING_INDEX)(...)
String verkÃ¼rzen + auffÃ¼llen	LPAD(str, len, fuellstr) LTRIM(str) RPAD(str, len, fuellstr) RTRIM(str) TRIM([BOTH | LEADING | TRAILING] [zeichen] [FROM] str)
String-umwandlung	LCASE(str) LOWER(str) UCASE(str) UPPER(str) REVERSE(str)
Stringsuche + ersatz	INSTR(str, teil) REPLACE(str, alt, neu) SOUNDEX(str)
Stringvergl.	FIELD(str, str1, str2, ...) # --> ELT(...) FIND_IN_SET(str, menge) LOCATE(teil, str [, zahl]) MATCH (spalte1, ...) AGAINST (str [modifier]) POSITION(teil, str) --> LOCATE STRCMP(str1, str2)
Datum	CURDATE() CURRENT_DATE() # --> Datum YYYY-MM-DD DATE(datum/zeit) DATE_FORMAT(datum, format) EXTRACT(zeitabschnitt FROM datetime) FROM_DAYS(tage) LAST_DAY(datum) MAKEDATE(jahr, tag) STR_TO_DATE(str, formatmuster) UTC_DATE()
Datumteile	DAYNAME(datum) DAYOFMONTH(datum) DAY(datum) DAYOFWEEK(datum) DAYOFYEAR(datum) MONTH(datum) MONTHNAME(datum) QUARTER(datum) WEEK(datum) WEEKDAY(datum) --> Mo=0, Di=1, ..., Sa=5, So=6 WEEKOFYEAR(datum) YEAR(datum) YEARWEEK(datum [, modus])
Datumrechnung	ADDDATE(datum, INTERVAL anz typ) --> DATE_ADD(...) DATE_ADD(datum, INTERVAL zeitspanne typ) DATE_SUB(datum, INVERVAL zeitspanne typ) DATEDIFF(neu_datum, alt_datum) # --> TAGE PERIOD_ADD(datum, monate) PERIOD_DIFF(datum1, datum2) SUBDATE(datum, INVERVAL anz typ) --> DATE_SUB(...) TO_DAYS(datum) --> Anz. Tage seit 01.01.0000 TO_SECONDS(datum) --> Anz. Sekunden seit 01.01.0000
Zeit	CONVERT_TZ(datum/zeit, zeitzone, zeitzone) CURTIME() CURRENT_TIME() LOCALTIME() MAKETIME(stunde, minute, sekunde) SEC_TO_TIME(sek) TIME_FORMAT(zeit, format) TIME_TO_SEC(zeit) UTC_TIME()
Zeitteile	HOUR(zeit) MINUTE(zeit)

	MICROSECOND(zeit) SECOND(zeit)
Zeitrechnung	ADDTIME(zeit, zeit) SUBTIME(datum/zeit, datum/zeit) TIMEDIFF(zeit, zeit)
Timestamp	FROM_UNIXTIME(sek [,format]) # --> Datum LOCALTIMESTAMP() NOW([stellen]) CURRENT_TIMESTAMP([stellen]) TIMESTAMP(datum, zeit) TIMESTAMPADD(intervall, wert, datum/zeit) TIMESTAMPDIFF(intervall, wert, datum/zeit) UNIX_TIMESTAMP([datum]) # --> Sekunden seit 1.1.1970 UTC_TIMESTAMP()
Bedingung	CASE wert WHEN auswahl THEN wert ... ELSE wert END ELT(zahl, str1, str2, ...) # --> FIELD(...) IF(test, wert1, wert2) INTERVAL(x, grenze1, grenze2, ...)
NULL-Vergleich	COALESCE(wert1, wert2, ...) IFNULL(wert1, wert2) ISNULL(ausdruck) NULLIF(wert1, wert2)
Konvertierung	ASCII(zeichen) BIN(dezimalzahl) CAST(ausdruck AS typ) CONVERT(ausdruck, typ) CHAR(num1 [,num2, ...]) [USING zeichensatz] CONV(zahl, basis1, basis2) CONVERT(ausdruck USING zeichensatz) GET_FORMAT(datentyp, formattyp) HEX(dezimalzahl) OCT(dezimalzahl) ORD(str) QUOTE(str) UNHEX(str) FROM_BASE64(str) TO_BASE64(str)
Zeichen-codierung + sortierung	CHARSET() COERCIBILITY() COLLATION()
Advisory Locking (User Level)	GET_LOCK(name, timeout) IS_FREE_LOCK(name) IS_USED_LOCK(name) RELEASE_LOCK(name)
Komprimierung	COMPRESS(str) UNCOMPRESS(str) UNCOMPRESSED_LENGTH(str)
VerschlÃ¼sseln EntschlÃ¼sseln	AES_DECRYPT(str, password) AES_ENCRYPT(str, password) DES_DECRYPT(str [, password]) DES_ENCRYPT(str [, password]) DECODE(blob, passwort) ENCODE(blob, passwort) ENCRYPT(str, salt) OLD_PASSWORD(str) PASSWORD(str)
Hashing	CRC32(str) MD5(str) SHA(str) SHA1(str)
Eindeutige ID	UUID() # --> 128-Bit Zahl UUID_SHORT() # --> 64-Bit Zahl
Netzwerk	INET_ATON(adresse) INET_NTOA(zahl)

Sonstige	BENCHMARK(anz, funktion)
	LOAD_FILE(dateiname) # Datei als String einlesen
	MASTER_POS_WAIT(dateiname, pos [, timeout])
	SLEEP(sek) # Optional .microsec

(C) 2024 OSTC GmbH (http://www.ostc.de)

9a) Aggregatfunktionen und Gruppierung (Aggregation)

Aggregatfunktionen fassen in einer SELECT-Anweisung die Werte einer Spalte <Col> Ã¼ber ALLE DatensÃ¤tze zusammen. Kommt eine GROUP-BY-Klausel bezÃ¼glich einer (anderen) Spalte <Col2> vor, erfolgt die Zusammenfassung PRO unterschiedlichem Wert dieser Spalte <Col2> (mehrere GROUP-BY-Spalten sind mÃ¶glich). MySQL kennt folgende Aggregatfunktionen:

Funktion	Beschreibung
COUNT(*)	Anzahl ALLER DatensÃ¤tze (auch NULL!)
COUNT(<Col>)	Anzahl aller Werte ungleich NULL
COUNT(DISTINCT <Col>)	Anzahl aller verschiedenen Werte ungleich NULL
SUM(<Col>)	Summe aller Werte
SUM(DISTINCT <Col>)	Summe aller verschiedenen Werte (MY!5.1)
AVG(<Col>)	Durchschnitt aller Werte
AVG(DISTINCT <Col>)	Durchschnitt aller verschiedenen Werte (MY!5.1)
MIN(<Col>)	Minimum aller Werte
MAX(<Col>)	Maximum aller Werte
STD(<Col>)	Standardabweichung Werte ungleich NULL (MY!)
STDDEV(<Col>)	Standardabweichung Werte ungleich NULL (MY!)
STDDEV_POP(<Col>)	Standardabweichung Werte ungleich NULL
STDDEV_SAMP(<Col>)	Standardabweichung aller Werte (auch NULL!)
VARIANCE(<Col>)	Varianz aller Werte ungleich NULL (MY!)
VAR_POP(<Col>)	Varianz aller Werte ungleich NULL
VAR_SAMP(<Col>)	Varianz aller Werte (auch NULL!)
BIT_AND(<Col>)	Bitweise UND-VerknÃ¼pfung aller Werte (MY!)
BIT_OR(<Col>)	Bitweise ODER-VerknÃ¼pfung aller Werte (MY!)
BIT_XOR(<Col>)	Bitweise XOR-VerknÃ¼pfung aller Werte (MY!)
GROUP_CONCAT()	Verkettung aller ermittelten Gruppenwerte (MY!)

HINWEIS: Je nachdem ob eine Spalte <Col> NULL-Werte enthÃ¤lt oder nicht gilt:

```
COUNT(*) = COUNT(<Col>)    # Kein NULL-Wert in <Col>
COUNT(*) > COUNT(<Col>)    # Mind. ein NULL-Wert in <Col>
```

Beispiele:

```
CREATE TABLE bestellung (
  nr     INT,
  anz    INT,
  preis  FLOAT,
  summe  FLOAT,
  rabatt FLOAT
);

INSERT INTO bestellung VALUES
  (1, 2, 123.45, 2 * 123.45, 10),
  (2, 5,  10.45, 5 *  10.45,  0),
  (3, 1,  67.00, 1 *  67.00,  5);

SELECT COUNT(name)      AS "Artikelanzahl"      FROM bestellung;
SELECT AVG(preis)       AS "Durchschnittspreis" FROM bestellung;
SELECT SUM(preis * anz) AS "Gesamtpreis"        FROM bestellung;
SELECT MIN(preis)       AS "Billig",
       MAX(preis)       AS "Teuer"              FROM bestellung;
```

Mittels der Klausel GROUP BY lassen sich DatensÃ¤tze basierend auf GLEICHEN Werten einer oder mehrerer Spalten GRUPPIEREN (zusammenfassen). Ã¨M-^\ber die obigen Aggregatfunktionen entstehen dabei aus den Einzelwerten der RESTLICHEN Spalten der jeweils zusammengefassten DatensÃ¤tze Gesamtwerte.

Eigenschaften:
* Pro VERSCHIEDENEM Wert einer GROUP BY Spalte entsteht EINE Ergebniszeile
* DatensÃ¤tze mit NULL-Wert in gruppierter Spalte werden (meist) ignoriert
* Die gruppierten Spaltenwerte sind sortierbar:
  + aufsteigend (ASC, STD)
  + absteigend (DESC)
* Gesamtzeile bzgl. gruppierter Spalte mit Zusatz "WITH ROLLUP" mÃ¶glich
  + Spaltenwert dieses Datensatzes ist "NULL"
  + Werttext fÃ¼r Gesamtzeile per IFNULL festlegen
    IFNULL(<GroupCol>, "<NullText>") AS <ColName>
  + Nicht zusammen mit Sortierung verwendbar
  + NULL nicht in HAVING als Bedingung verwendbar

```
Syntax:

  SELECT ...
    FROM ...
      GROUP BY <Col> [ASC|DESC] [WITH ROLLUP] {, ...}
        [FOR UPDATE |
         LOCK IN SHARE MODE |
         PROCEDURE <ProcExternal>(...) |
         PROCEDURE ANALYSE([<MaxElem>[, <MaxMem>]]) ]

Beispiele:

  SELECT SUM(summe) AS "Gesamt"
    FROM bestellung
      GROUP BY nr                                    # STD: ASC (aufsteigend)
        HAVING Gesamt > 100;

  SELECT nr, AVG(preis) AS "Durchschnittspreis"
    FROM artikel
      GROUP BY nr DESC                               # STD: ASC (aufsteigend)
        HAVING COUNT(nr) > 1;

  CREATE TABLE verkaeufe (
    jahr    INT NOT NULL,
    land    VARCHAR(20) NOT NULL,
    produkt VARCHAR(32) NOT NULL,
    gewinn  INT
  );

  INSERT INTO verkaeufe (jahr, land, produkt, gewinn) VALUES
    (2009, "Deutschland", "Mixer",   11.3),
    (2010, "Deutschland", "Mixer",   10.7),
    (2009, "England",     "Mixer",   11.0),
    (2010, "England",     "Mixer",    9.5),
    (2009, "Frankreich",  "Mixer",   12.1),
    (2010, "Frankreich",  "Mixer",   10.2),
    (2009, "Deutschland", "Toaster", 21.3),
    (2010, "Deutschland", "Toaster", 20.7),
    (2009, "England",     "Toaster", 21.0),
    (2010, "England",     "Toaster", 19.5),
    (2009, "Frankreich",  "Toaster", 22.1),
    (2010, "Frankreich",  "Toaster", 20.2),
    (2009, "Deutschland", "Kocher",   1.3),
    (2010, "Deutschland", "Kocher",   0.7),
    (2009, "England",     "Kocher",   1.0),
    (2010, "England",     "Kocher",   0.5),
    (2009, "Frankreich",  "Kocher",   2.1),
    (2010, "Frankreich",  "Kocher",   0.2);

  SELECT jahr, SUM(gewinn)                    # --> jahr SUM(gewinn)
    FROM verkaeufe                            #     2009         102
      GROUP BY jahr;                          #     2010          94

  SELECT jahr, SUM(gewinn)                    # --> jahr SUM(gewinn)
    FROM verkaeufe                            #     2009         102
      GROUP BY jahr                           #     2010          94
        WITH ROLLUP;                          #     NULL         196

  SELECT IFNULL(jahr, "GESAMT"), SUM(gewinn) # --> jahr SUM(gewinn)
    FROM verkaeufe                            #     2009         102
      GROUP BY jahr                           #     2010          94
        WITH ROLLUP;                          #     GESAMT       196

  SELECT jahr, land, SUM(gewinn)              # Mehrere Gruppen
    FROM verkaeufe                            #
      GROUP BY jahr ASC, land DESC;           #

  SELECT jahr, land, SUM(gewinn)              #
    FROM verkaeufe                            #
      GROUP BY jahr, land                     # Mehrere Gruppen
        WITH ROLLUP;                          # + Gesamt

  SELECT IFNULL(jahr, "JAHRE")    AS jahr,    # Gesamtwert umbenennen
         IFNULL(land, "LAENDER")  AS land,    # Gesamtwert umbenennen
         SUM(gewinn)                          #
    FROM verkaeufe                            #
      GROUP BY jahr, land                     # Mehrere Gruppen
        WITH ROLLUP;                          # + Gesamt


10) SchlÃ¼ssel (Key) und Index
```

```
-----------------------------

10a) Eigenschaften
------------------

"SchlÃ¼ssel" (Key) und "Index" haben zunÃ¤chst nichts miteinander zu tun:
  * Ein "SchlÃ¼ssel" adressiert jeden Datensatz einer Datenbanktabelle EINDEUTIG
    + Aus einer Spalte oder Kombination mehrerer Spalten gebildet
    + Nutzen: Beziehung zwischen Tabellen herstellen
    + NUR EIN (primÃ¤rer) SchlÃ¼ssel pro Tabelle mÃ¶glich (PRIMAY KEY)
  * Ein "Index" wird fÃ¼r eine Spalte oder Kombination von Spalten angelegt,
    nach denen hÃ¤ufig gesucht oder sortiert wird
    + Ã„hnlich einem Inhaltsverz. zu einem Buch:
      - Suche nach DatensÃ¤tzen beschleunigen (WHERE)
      - Sortierung von DatensÃ¤tzen beschleunigen (ORDER BY)
      - Eindeutigkeit prÃ¼fen (UNIQUE KEY)
      - Beziehung herstellen (FOREIGN KEY)
    + MEHRERE Indices pro Tabelle mÃ¶glich
    + Muss nicht unbedingt eindeutig (UNIQUE) sein
  * Zusammenhang:
    + Zu jeden SchlÃ¼ssel einer Datenbank meist ein Index erstellt (kein MUSS)

Unterscheidung:
  * PrimÃ¤rschlÃ¼ssel (PRIMARY KEY)
    + Kennzeichnet JEDEN Datensatz EINDEUTIG (z.B. Abteilungsnummer)
    + Meist (abstrakte) fortlaufende Nummer
    + HÃ¤ufig automatisch beim Anlegen eines Datensatzes erzeugt
    + NUR EIN PrimÃ¤rschlÃ¼ssel pro Tabelle mÃ¶glich und notwendig
    + Eine PrimÃ¤rschlÃ¼sselspalte darf NIEMALS LEER sein (NOT NULL)
    + Komplexe/lange PrimÃ¤rschlÃ¼ssel verlangsamen Datenbank-Operationen
      (am besten nur Spalte vom Typ "kleine ganze Zahl" verwenden)
  * SekundÃ¤rschlÃ¼ssel
    + Kennzeichnet ebenfalls Datensatz eindeutig (z.B. Abteilungsname)
    + MEHRERE SekundÃ¤rschlÃ¼ssel pro Tabelle mÃ¶glich
    + SekundÃ¤rschlÃ¼ssel darf LEER sein (NULL)
  * FremdschlÃ¼ssel (FOREIGN KEY)
    + Bezeichnet die Ã„nderbereinstimmung einer/mehrerer Spalten in einer Tabelle
      mit der/den PrimÃ¤rschlÃ¼sselspalte(n) einer anderen Tabelle ("Verweise")
    + MEHRERE FremdschlÃ¼ssel pro Tabelle mÃ¶glich
    + KEINE ZIRKULÃ„REN Referenzen auf FremdschlÃ¼ssel erstellen!
    + HÃ¤ufig in "Lookup-Tabelle" verwendet, die 2 oder mehr Tabellen verknÃ¼pft
    + 1:1-Beziehung, 1:N-Beziehung, N:M-Beziehung
  * Index
    + Erleichtern Suche nach DatensÃ¤tzen vor allem in groÃ§en Tabellen
    + EinfÃ¼gen, Ã„ndern, LÃ¶schen von DatensÃ¤tzen erfordert Index-Aktualisierung
      --> Kann viel Zeit kosten
    + TIP: Indices groÃ§er Tabellen erst NACH dem FÃ¼llen anlegen

Hinweise:
  * Nur 1 PrimÃ¤rschlÃ¼ssel pro Tabelle erlaubt
    + Automatisch UNIQUE
    + Automatisch NOT NULL
    + Automatisch auch ein Index
    + Darf mehr als 1 Spalte enthalten!
  * Mehrere UNIQUE Indices anlegbar
    + Automatisch SekundÃ¤rschlÃ¼ssel
    + NULL-Werte darin erlaubt
  * Index auf SpaltenprÃ¤fix beschrÃ¤nkbar (z.B. name(5), vorname(5))
    (nicht auf Suffix --> TRICK: Daten per REVERSE() umgedreht speichern)
    - Sollte selektiv genug sein
    - Spart Platz
    - Bei Spaltentyp TEXT/BLOB notwendig
  * Tabelleninhalt wird durch Index nicht verÃ¤ndert
  * Index = "Zeiger" auf DatensÃ¤tze in sortierter Reihenfolge
  * Index belegt zusÃ¤tzlichen Plattenplatz
  * Index enthÃ¤lt KEINE zusÃ¤tzlichen Daten,
    sondern erlauben nur schnelleren Lese-Zugriff
    + Index daher jederzeit lÃ¶schbar
    + Index daher jederzeit aus Tabellendaten wieder herstellbar
      (in Backup nicht aufnehmen)
    + Beschleunigt nur Zugriff Ã¼ber indizierte Spalten
  * Beim Erzeugen + Ã„ndern von DatensÃ¤tzen kosten Indices zusÃ¤tzlichen Aufwand,
    da jeder Schreibvorgang auf DatensÃ¤tzen die Indices mitpflegen muss
    - NachtrÃ¤gliche Index-Erzeugung --> Tabelle fÃ¼r Schreiben gesperrt!

HINWEIS: Manche DB-Systeme fordern zu jeder Tabelle einen PrimÃ¤rschlÃ¼ssel,
MySQL kann darauf auch verzichten (nur in AusnahmefÃ¤llen wirklich sinnvoll).

10b) Erstellen/entfernen
------------------------

PrimÃ¤rschlÃ¼ssel erstellen/entfernen (automatisch/muss NOT NULL + UNIQUE sein):
```

```
  * Beim Erzeugen einer Tabelle (1.Form mehrspaltig, 1+2.Form = einspaltig)
      CREATE TABLE <Tbl> (                CREATE TABLE <Tbl> (
        nr INT NOT NULL AUTO_INCREMENT,     nr INT AUTO_INCREMENT PRIMARY KEY,
        ...,                                ...,
        PRIMARY KEY (nr)                    ...
      );                                  );
  * NachtrÃ¤glich erzeugen
    - Nur mÃ¶glich, wenn Spalte "nr" NOT NULL + UNIQUE (KEINE doppelten Werte)!
        ALTER TABLE <Tbl> ADD PRIMARY KEY (nr);   # OK
        CREATE PRIMARY KEY ON <Tbl> (nr);         # FALSCH! --> ALTER TABLE!
  * Entfernen
      ALTER TABLE <Tbl> DROP PRIMARY KEY;

SekundÃ¤rschlÃ¼ssel erstellen/entfernen:
  * Beim Erzeugen einer Tabelle
      CREATE TABLE pers (
        nr      INT      NOT NULL AUTO_INCREMENT,
        name    CHAR(30) NOT NULL,
        vorname CHAR(30) NOT NULL,
        PRIMARY KEY (nr),
        UNIQUE INDEX idx1 (name, vorname)   # Oder nur UNIQUE ohne INDEX
      );
  * NachtrÃ¤glich erzeugen
    + Nur mÃ¶glich, wenn Spalte "nr" UNIQUE (KEINE doppelten Werte enthÃ¤lt!),
    + NULL-Werte sind nicht erlaubt
        ALTER TABLE pers ADD UNIQUE INDEX idx1 (name, vorname);   # INDEX weglassbar
        CREATE UNIQUE INDEX idx1 ON pers (name, vorname);
  * Entfernen
      ALTER TABLE pers DROP INDEX idx1;   # Nur INDEX (ohne UNIQUE, ...)

Index erstellen, entfernen, (de)aktivieren und anzeigen
  * Allgemeine Syntax:
      CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX <Idx>
        [USING {BTREE | HASH | RTREE}]
          ON <Tbl> (<Col> [(<Len>)] [ASC | DESC], ...);
    + Indexname <Idx> nÃ¶tig, um Index gezielt entfernen zu kÃ¶nnen
      --> MUSS pro Tabelle eindeutig sein
    + Index-Art wÃ¤hlbar (UNIQUE, FULLTEXT, SPATIAL)
        CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX <Idx> ...;
    + Indexsortierung wÃ¤hlbar (in MySQL derzeit ignoriert, STD: ASC)
        ... ON <Tbl> (<Col1> ASC, <Col2> DESC, ...);
    + FÃ¼r Index benutzte SpaltenprÃ¤fix-LÃ¤nge wÃ¤hlbar
        ... ON <Tbl> (<Col1>(<Len1>), <Col2>(<Len2>), ...);
    + Index-Struktur wÃ¤hlbar (BTREE, HASH, RTREE)
      - MyISAM: BTREE, RTREE
      - InnoDB: BTREE
      - MEMORY: BTREE, HASH
      - NDB:    BTREE, HASH
        CREATE INDEX <Idx> [USING {BTREE | HASH | RTREE}] ...;
  * Beim Erzeugen einer Tabelle
      CREATE TABLE <Tbl> (
        nr       INT      NOT NULL,
        vorname  CHAR(30) NOT NULL,
        name     CHAR(30) NOT NULL,
        ...,
        INDEX idx1 (nr),
        INDEX idx2 (name, vorname)
      );
  * NachtrÃ¤glich erzeugen
      CREATE INDEX <Idx> ON <Tbl> (<Col>, ...);
      ALTER TABLE <Tbl> CREATE INDEX <Idx> (<Col>, ...);
  * Entfernen
      DROP INDEX <Idx> ON <Tbl>;
      ALTER TABLE <Tbl> DROP INDEX <Idx>;
  * Aktivieren/deaktivieren
      ALTER TABLE <Tbl> DISABLE KEYS;   # Alle Non-UNIQUE Indices abschalten
      ALTER TABLE <Tbl> ENABLE KEYS;    # Alle Non-UNIQUE Indices wieder aufbauen
      ALTER INDEX <Idx> ACTIVE;         # In MySQL NICHT mÃ¶glich (MY!)
      ALTER INDEX <Idx> INACTIVE;       # In MySQL NICHT mÃ¶glich (MY!)
  * Indices einer Tabelle anzeigen lassen (mit KardinalitÃ¤tswert N/S)
      SHOW INDEX FROM <Tbl>;
  * Indices einer Datenbank anzeigen lassen (NICHT unter MySQL, MY!)
      SHOW INDICES;

10c) FULLTEXT-Index
--------------------

Eigenschaften:
  * Nur bei Engine "MyISAM" mÃ¶glich!
  * Nur auf Spaltentypen CHAR, VARCHAR, TEXT mÃ¶glich
  * Suche nach einzelnen Worten und beliebig langen Texten mÃ¶glich
  * Nach einigen englischen "Stopwords" wir NICHT gesucht (z.B. "and", "or")
```

```
  * Suche NICHT mit LIKE/REGEX mÃ¶glich
  * Spezielle Suchsyntax:
      MATCH (<Col1>, ...) AGAINST (<String> [<Modifier>])
    mit <Modifier>
      IN BOOLEAN MODE                                 # Syntax +xxx -yyy ...
      IN NATURAL LANGUAGE MODE                        # Ab MY!5.1.7
      IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION   # Ab MY!5.1.7
      WITH QUERY EXPANSION                            #

10d) SPATIAL-Index
------------------

Eigenschaften:
  * Nur bei Engine "MyISAM" mÃ¶glich!
  * Nur auf Geometrie-Spaltentypen mÃ¶glich
  * Basiert auf dem OpenGIS Geometrie-Modell

10e) Index-Nutzung
------------------

  * Indices verwendet bei SELECT (UPDATE, DELETE, REPLACE), ORDER BY und JOIN.
  * Datenbank sucht zu jeder Abfrage SELBSTÃM-^DNDIG sinnvoll einzusetzende Indices
    (mit Hilfe des "Query Optimizer")
  * Tabellen/Indices enthalten Informationen Ã¼ber SelektivitÃ¤t und statistische
    Verteilung der indizierten Daten (Histogramm)
    + Bsp: Index auf Spalte "Geschlecht" ist wenig nÃ¼tzlich,
           da immer etwa auf die HÃ¤lfte der Daten zugegriffen wird
    + Bsp: Index auf SchlÃ¼sselspalte "nr" ist sehr nÃ¼tzlich (da eindeutig)
  * Index kann eine/mehrere Spalten einer Tabelle umfassen ("Composite Index")
    + Umfasst er mehrere Spalten, dann sind alle Spalten-Kombinationen
      von links nach rechts ebenfalls enthalten (PrÃ¤fixe)!
    + Ein mehrspaltiger Index ist immer dann nutzbar,
      wenn ALLE Spalten von links nach rechts in WHERE-Klausel vorkommen
    + Beispiel:
        CREATE INDEX multi ON adresse (name, vorname, strasse);
      Folgende Abfragen nutzen diesen Index:
        SELECT * FROM adresse WHERE name LIKE "A%";
        SELECT * FROM adresse WHERE name LIKE "A%" AND
                                    vorname LIKE "B%"
        SELECT * FROM adresse WHERE name LIKE "A%" AND
                                    vorname LIKE "B%" AND
                                    strasse LIKE "C%";
      Folgende Abfragen kÃ¶nnen diesen Index NICHT nutzen:
        SELECT * FROM adresse WHERE vorname LIKE "B%";
        SELECT * FROM adresse WHERE strasse LIKE "B%";
        SELECT * FROM adresse WHERE vorname LIKE "B%" AND
                                    strasse LIKE "C%";
  * Werden nur Datenspalten gelesen, die in EINEM Index enthalten sind:
    + Index enthÃ¤lt bereits alle benÃ¶tigten Daten
    + Zugriff auf Datentabelle nicht nÃ¶tig
    + "Covering Index"
    + Beschleunigung
    + Beispiel:
        SELECT name FROM adresse WHERE name LIKE "A%" AND
                                    vorname LIKE "B%"
  * Index bei folgenden Vergleichen verwendbar:
      <=>                           # NULL-sicheres "="
      =                             # Gleich
      <                             # Kleiner
      <=                            # Kleiner oder gleich
      >                             # GrÃ¶Ã-^_er
      >=                            # GrÃ¶Ã-^_er oder gleich
      LIKE "abc%"                   # Platzhalter "%" hinten
      LIKE "abc__"                  # Platzhalter "_" hinten
      BETWEEN <Min> AND <Max>       # Bereich einschlieÃ-^_lich Grenzen
  * KEIN Index bei folgenden Vergleichen verwendbar:
      !=                            # Ungleich
      <>                            # Ungleich
      LIKE "%abc"                   # Platzhalter "%" vorne
      LIKE "__abc"                  # Platzhalter "_" vorne
      RLIKE "abc"                   # RegulÃ¤rer Ausdruck
      REGEX "abc"                   # RegulÃ¤rer Ausdruck
      NOT RLIKE "abc"               # RegulÃ¤rer Ausdruck
      NOT REGEX "abc"               # RegulÃ¤rer Ausdruck
      NOT BETWEEN <Min> AND <Max>   # Nicht in Bereich einschlieÃ-^_lich Grenzen
  * Werden mehrere Bedingungen per AND/OR verknÃ¼pft, dann wird meist der Index
    zu den Spalten einer Bedingung verwendet, der am SELEKTIVSTEN ist, d.h.
    die wenigsten DatensÃ¤tze selektiert. Diese DatensÃ¤tze werden alle gelesen
    und die restlichen Bedingungen dagegen geprÃ¼ft
  * Index auch bei Sortierung ORDER BY und Gruppierung GROUP BY verwendbar
  * SchlÃ¼sselworte "KEY" und "INDEX" sind synonym
```

```
Indices werden verwendet bei:
  * Auswahl DatensÃ¤tze gemÃ¤Ã-^_ WHERE-Klausel
  * DatensÃ¤tze weglassen
  * Doppelte EintrÃ¤ge rauswerfen (DISTINCT)
  * JOIN: Typ und LÃ¤nge mÃ¼ssen gleich sein (Konvertierung verhindert Index!)
  * MIN/MAX fÃ¼r bestimmte Spalte
  * ORDER BY/GROUP BY falls ausgefÃ¼hrt auf PrÃ¤fix eines Index
  * Daten + Bedingung durch Index befriedigt und numerisch
  * = > >= < <= BETWEEN LIKE (mit fixem PrÃ¤fix)
  * <Col> IS NULL wenn Index auf <Col>
  * Index muss in jeder AND-Gruppe der WHERE-Klausel verwendbar sein
    (kann auch 1-elementig ohne AND sein!)
  * LIMIT

Indices werden NICHT verwendet bei:
  * Lese-Operationen, die groÃM-^_en Prozentsatz einer Tabelle lesen
  * OR auf Spalten in verschiedenen Indices --> IN, UNION [ALL]!
  * RegulÃ¤ren AusdrÃ¼cken


10f) Index-Optimierung
----------------------

Richtlinien fÃ¼r das Erstellen von Indices:
  * PrimÃ¤r-, SekundÃ¤r- und FremdschlÃ¼ssel erhalten automatisch einen Index
    + PRIMARY KEY so kurz wie mÃ¶glich und eindeutig
    + Nur im absoluten Ausnahmefall keinen PrimÃ¤rschlÃ¼ssel definieren
      (DatensÃ¤tze dann nicht eindeutig identifizierbar)
  * Spalten mit wenigen unterschiedlichen Werten (z.B. Anrede, Geschlecht)
    bringen mit Index KEINEN Geschwindigkeitsgewinn (KardinalitÃ¤t zu gering)
  * Indices auf Tabellen einsetzen, die oft gefiltert/gruppiert/sortiert werden
    + Indices auf Tab. mit wenig DatensÃ¤tzen (<=12) bringt KEINEN Geschw.gewinn
    + Mehr als 10% DatensÃ¤tze einer Tab. stÃ¤ndig gelesen --> Index nicht sinnvoll
  * Zwei Indices notwendig fÃ¼r auf- + absteigendes Sortieren (ASC, DESC)
  * Sortierkriterium an letzter Stelle im Index --> (teuren) Filesort sparen
  * Aggregatfunktionen nutzen keinen Index
    + COUNT, SUM, AVG, STDDEV, ...
    + Da alle DatensÃ¤tze durchlaufen werden mÃ¼ssen
    + MIN/MAX fÃ¼r bestimmte Spalte schon
  * Ohne Indices wÃ¤re immer ein "Full Table Scan" notwendig
    + Beim (fast) vollstÃ¤ndigen Durchlesen einer Tabelle KEINEN Index benutzen!
      --> Evtl. Cursor, Handler besser
  * MySQL: DatensÃ¤tze und Indices IMMER in getrennten Dateien (InnoDB?)
    + Kostet zusÃ¤tzlichen READ bei Zugriff Index --> Daten
    + Evtl. "Full Table Scan" schneller --> Index lesen unnÃ¶tig
    + Evtl. "Covering Index" ausreichend --> Daten lesen unnÃ¶tig
    + Nur 1x Speicherplatz fÃ¼r Indexwert bei DatensÃ¤tzen mit gleichem Indexwert
    + Delete-Operationen "degenerieren" Tabelle nicht
    + Caching getrennt mÃ¶glich
  * Nur unbedingt notwendige Indices nutzen (Schreibperformance sonst schlecht)
  * Am hÃ¤ufigsten benutzte Spalten(kombination) indizieren
    + Mehrere Spalten nach abnehmender HÃ¤ufigkeit sortieren
    + Zusammengesetzter Index nutzbar in mehreren FÃ¤lle: 1.Sp, 1+2.Sp, ...
    + Spalte mit mehr doppelten Werten vorne (KardinalitÃ¤t)
  * Nur ausreichend lange PrÃ¤fixe indizieren (Platz, Hits)
    + FÃ¼r BLOB/TEXT ein MUSS!
    + KardinalitÃ¤t reduziert
    + TRICK: REVERSE oder SUBSTR verwenden, falls Anfang statisch

Eigenschaften von Indices:
  * Automatisch PrÃ¤fix- und LÃ¤ngen-komprimiert (MY!)
  * MySQL verwendet pro Tabelle in einer Query maximal EINEN Index
    + Normalerweise Index mit kleinster Treffermenge (Anz. passender DatensÃ¤tze)
    + Evtl. mehrspaltiger Index sinnvoll!

Verschiedene Indexverfahren mÃ¶glich (MY!):
  * BTREE: Fast immer mÃ¶glich
  * RTREE: FÃ¼r geometrische Daten
  * HASH:  Bei MEMORY-Tabellen
    + EinschrÃ¤nkung des HASH-Index
      - NUR fÃ¼r = und <=>
      - Nicht fÃ¼r != <> < <= > >= verwendbar
      - Nicht fÃ¼r ORDER BY und GROUP BY verwendbar
      - Nur vollstÃ¤ndiger Index nutzbar, keine PrÃ¤fixe

Index-Hint: Manuell Hinweis zur Indexnutzung in SELECT pro Tab. angebbar (MY!):
  * Nach FROM <Tbl>:
      USE    INDEX = NUR diese Indices benutzen ("Full Table Scan" mÃ¶glich)
      FORCE  INDEX = Analog USE INDEX ("Full Table Scan" NUR wenn unmÃ¶glich)
      IGNORE INDEX = Diese Indices NICHT benutzen
  * Name des PrimÃ¤ren Index ist "PRIMARY"
  * USE INDEX Liste darf auch leer sein --> Keine Indices benutzen!
```

```
  * Auswahl fÃ¼r welche Operation (fehlt --> FÃ¼r alle 3 FÃ¤lle gÃ¼ltig):
      FOR JOIN     = FÃ¼r Tabellen-Join
      FOR ORDER BY = FÃ¼r Sortierung
      FOR GROUP BY = FÃ¼r Gruppierung

  +-----------------------------------------------------------------------------+
  | FROM <Tbl> USE    INDEX [FOR {JOIN | ORDER BY | GROUP BY}] (<Idx1>, ...)   |
  |            FORCE  INDEX [FOR {JOIN | ORDER BY | GROUP BY}] (<Idx1>, ...)   |
  |            IGNORE INDEX [FOR {JOIN | ORDER BY | GROUP BY}] (<Idx1>, ...)   |
  +-----------------------------------------------------------------------------+

Beispiel:

  SELECT * FROM table1 USE INDEX (idx1, idx2)
    WHERE col1 = 1 AND col2 = 2 AND col3 = 3;

  SELECT * FROM table1 IGNORE INDEX (idx3)
    WHERE col1 = 1 AND col2 = 2 AND col3 = 3;

  SELECT * FROM t1 USE INDEX (i1)
                  IGNORE INDEX FOR ORDER BY (i2),
               t2 FORCE INDEX (i3),
    WHERE t1.id = t2.id
      ORDER BY a;

Indices reorganisieren:
  * Sinnvoll nach dem Laden von Daten
  * Cardinality = Durchschnittliche Datensatzanz. mit gleichem Wert berechnen
      ANALYZE TABLE <Tbl>;
      OPTIMIZE TABLE <Tbl>;
      myisamchk --analyze / -a <Tbl>
      SHOW INDEX FROM <Tbl>;
      myisamchk --description --verbose <Tbl>
  * Evtl. Index UND Daten gemÃ¤Ã-^_ einem Index sortieren
    (fÃ¼r sortierten Zugriff auf alle Daten gemÃ¤Ã-^_ UNIQUE Index)
    (kann beim 1. Mal sehr lange dauern)
      myisamchk --sort-index / -S <Tbl>        # Indices sortieren
      myisamchk --sort-records=N / -R N <Tbl>  # GemÃ¤Ã-^_ Index N sortieren


10g) FremdschlÃ¼ssel (Foreign Keys) und Referenzielle IntegritÃ¤t
----------------------------------------------------------------

Grund fÃ¼r FremdschlÃ¼ssel:
  * Dokumentieren Beziehungen zwischen Tabellen (Master/Detail, Vater/Kind)
  * Verhindern EinfÃ¼gen von Inkonsistenzen in DB durch Programmierer
    + "Referenzielle IntegritÃ¤t" (Referential Integrity) erhalten
    + Reihenfolge von EinfÃ¼ge/Ã-^Dnderungs/LÃ¶sch-Operationen nicht zu beachten
      (verhindert "verwaiste Kinder" = orphaned childs)
    + Fehlerbehandlung bei Unterbrechung
  * Zentrale Constraint-PrÃ¼fung (in Anwendung verzichtbar + einheitlich)
  * Kaskadierende UPDATEs/DELETEs vereinfachen Anwendungscode

Zusammenhang zwischen Tabellen = Beziehung zw. PrimÃ¤r- und FremdschlÃ¼sseln:
  * Beim UPDATE/DELETE eines Datensatzes aus einer (Eltern)Tabelle werden alle
    damit Ã¼ber den gleichen FremdschlÃ¼ssel verknÃ¼pften DatensÃ¤tze in zugehÃ¶rigen
    (Kind)Tabellen ebenfalls AUTOMATISCH von der Datenbank gelÃ¶scht/geÃ¤ndert
  * Beim INSERT eines Datensatzes in eine (Kind)Tabelle mit FremdschlÃ¼ssel
    wird geprÃ¼ft, ob eine Entsprechung in referenzierter (Eltern)Tabelle
    vorhanden ist. Wenn nein: Fehlermeldung + Datensatz nicht eingefÃ¼gt

Eigenschaften:
  * GrundsÃ¤tzlich bei Tabellen-Definition angebbar
    + Bei allen Engines auÃ-^_er "InnoDB" syntaktischer Zucker (reine Doku)
    + Nur bei "InnoDB" gespeichert (und "PBXT")
    + Speicherung und Implementierung fÃ¼r "MyISAM" geplant
  * Workaround falls nur ON DELETE nÃ¶tig:
    + Multi-Table DELETE
  * Erzeugen zusÃ¤tzlichen Overhead
    + Besser selber machen oder an DB delegieren?
      HÃ¤ngt von Anwendung ab (1x implementieren statt mehrmals)
  * Restore individueller Tabellen von Backup erschwert
    --> Foreign Keys + Constraints + Trigger temporÃ¤r abschalten!
  * Sinnvoll fÃ¼r Cascading Updates/Deletes
  * Nicht sinnvoll fÃ¼r Constraints --> Trigger besser --> Noch besser ENUM-Typ

Bedingungen:
  * Bezugstabelle muss vorhanden sein
  * Datentyp und DatengrÃ¶Ã-^_e korrespondierender Spalten MUSS identisch sein!
    (bei CHAR/VARCHAR darf LÃ¤nge verschieden sein)
  * Auf korrespondierenden Spalten muss ein Index liegen
  * VerknÃ¼pfte Spalten mÃ¼ssen NOT NULL sein und UNIQUE Index haben
```

```
    (wird aber nicht erzwungen)
  * Bei temporÃ¤ren Tabellen NICHT erlaubt

Vorteile/Nachteile:
  * Nicht umsonst (zusÃ¤tzlicher Lookup bei jeder Ã-^Dnderung)
  * Arbeitet "Zeile fÃ¼r Zeile" (auch bei multiplem INSERT/UPDATE/DELETE)
  * Index notwendig (evtl. groÃ-^_ und zu sonst nichts nutze)

Erzeugen einer (Kind)Tabelle mit FremdschlÃ¼sseln einer/mehrere (Eltern)Tabellen
("CONSTRAINT <Name>" optional):

  CREATE TABLE <Tbl> (    # Kindtabelle
    ma_id INT NOT NULL,   # FremdschlÃ¼ssel aus Elterntabelle
    pr_id INT NOT NULL,   # FremdschlÃ¼ssel aus Elterntabelle
    ...,
    FOREIGN KEY (ma_id) REFERENCES mitarbeiter (id)   # Bezug zur Elterntabelle
      ON UPDATE CASCADE                               # E-Aktion --> K-Aktion
      ON DELETE CASCADE,                              # E-Aktion --> K-Aktion
    CONSTRAINT cpj FOREIGN KEY (pr_id) REFERENCES projekt (id)  # Bezug z.E.
      ON UPDATE CASCADE                               # E-Aktion --> K-Aktion
      ON DELETE CASCADE                               # E-Aktion --> K-Aktion
  ) ENGINE = "InnoDB";

FremdschlÃ¼ssel nachtrÃ¤glich erzeugen (nur mÃ¶glich, wenn verknÃ¼pfte Tabellen ex.
sowie verknÃ¼pfte Spalten NOT NULL sind und KEINE doppelten Werte enthalten):

  ALTER TABLE <Tbl> ADD [CONSTRAINT <Name>]     # Kindtabelle
    FOREIGN KEY (<Col>, ...)                    # Spalte in Kindtabelle
      REFERENCES <Tbl> (<Col>, ...)             # Spalten in Elterntabelle
        [MATCH <MatchOpt>]                      # Match-Klausel: Weglassen!
          [ON UPDATE <RefOpt>]                  # STD: NO ACTION
          [ON DELETE <RefOpt>]                  # STD: NO ACTION
    ...;

Verhalten eines FremdschlÃ¼ssels steuern:
  * MATCH-Klausel <MatchOpt> definiert Behandlung von NULL-Werten in
    mehrspaltigen FremdschlÃ¼sseln beim Vergleich mit dem PrimÃ¤rschlÃ¼ssel
    (derzeit ignoriert, d.h. immer SIMPLE)
      MATCH SIMPLE          # Mehrsp. Fremdschl. darf teilw./ganz NULL sein (STD)
      MATCH PARTIAL         # Mehrsp. Fremdschl. darf nicht vollstÃ¤ndig NULL sein
      MATCH FULL            # Mehrsp. Fremdschl. muss vollstÃ¤ndig ungl. NULL sein
  * Verhalten beim EinfÃ¼gen/LÃ¶schen steuern:
      ON DELETE <RefOpt>    # Beim LÃ¶schen von Zeilen in Elterntabelle
      ON UPDATE <RefOpt>    # Beim Ã-^Dndern von Zeilen in Elterntabelle
  * Referenz-Optionen <RefOpt> (STD: Anweisung wird abgebrochen)
      NO ACTION             # Abbruch der Ã-^Dnderung in Elterntabelle (STD)
      RESTRICT              #   (analog NO ACTION)
      CASCADE               # LÃ¶schen/Ã-^Dndern transitiv auf Kindtabelle ausdehnen
      SET NULL              # Ref. Sp. in Kindtab. auf NULL setzen
      SET DEFAULT           # Ref. Sp. in Kindtab. auf Default setzen (nicht MY!)

FremdschlÃ¼ssel aus Kindtabelle entfernen:

  ALTER TABLE <Tbl> DROP FOREIGN KEY <Idx>;

FremdschlÃ¼ssel in Kindtabelle anzeigen:

  SHOW CREATE TABLE <Tbl>;
  SHOW TABLE STATUS FROM <Db> LIKE <Tbl>;

FremdschlÃ¼ssel Ã¼berprÃ¼fen vor Daten laden aus- und danach wieder einschalten:

  SET foreign_key_checks = 0;     # PrÃ¼fung ausschalten
  SOURCE <SqlFile>;               # Kein "..." um <SqlFile>!
  SET foreign_key_checks = 1;     # PrÃ¼fung einschalten


11) Joins
---------

Ein Join ist eine VerknÃ¼pfung von zwei Tabellen zu einer Gesamttabelle Ã¼ber
eine AbhÃ¤ngigkeit zwischen den Tabellen (PrimÃ¤rschlÃ¼ssel + FremdschlÃ¼ssel).
Das Ergebnis ist wieder eine (temporÃ¤re) Tabelle (und wieder ein JOIN mit einer
anderen Tabelle durchgefÃ¼hrt werden). MySQL unterstÃ¼tzt folgende JOIN-Arten:

  +---------------+-------------------------------------------------------+
  | Typ           | Bedeutung                                             |
  +---------------+-------------------------------------------------------+
  | [CROSS] JOIN  | Kombination ALLER DatensÃ¤tze der 1. Tabelle mit ALLEN |
  |               | der 2. Tabelle ("Kreuzprodukt", "Karthesisches Produkt") |
  |               | (KEINE Beziehung, andere Schreibweise: <Tbl1>, <Tbl2>). |
  +---------------+-------------------------------------------------------+
```

[INNER] JOIN	Kombin. Datens. aus 1. Tab. mit PASSENDEN aus 2. Tab. bzgl. einer/mehrerer Spalten. DatensÃ¤tze beider Tab. weglassen, die in Kombinationsspalte(n) keinen passend. Wert enthalten
{LEFT | RIGHT} [OUTER] JOIN	Analog [INNER] JOIN, aber JEDER Datensatz aus LEFT=linker bzw. RIGHT=rechter Tabelle ist im Ergebnis vorhanden (Spaltenwerte der anderen Tabelle evtl. alle NULL)
FULL [OUTER] JOIN	Kombination aus LEFT + RIGHT JOIN, d.h. JEDER Datensatz der linken + rechten Tabelle ist mind. 1x im Ergebnis vorhanden (erst ab MY!5.1 verfÃ¼gbar!)
STRAIGHT_JOIN	MySQL-Optimierung ignorieren und Daten benutzergesteuert in Reihenfolge der JOINs in FROM-Clause zusammenfÃ¼gen (MY!) (linke Tabelle immer zuerst VOR rechter Tab. gelesen), entspricht JOIN (d.h "Kreuzprodukt").
NATURAL [{LEFT | RIGHT} [OUTER]] JOIN	Automatisch verknÃ¼pfen Ã¼ber ALLE gemeinsamen Spaltennamen (USING ... unnÃ¶tig)

Hinweise:
* MySQL-Optimizer macht immer LEFT JOIN von links nach rechts
  (d.h. sortiert Tabellen in Zugriffsreihenfolge von links nach rechts)
* SchlÃ¼sselworte CROSS, INNER und OUTER dÃ¼rfen weggelassen werden
  (aus GrÃ¼nden der klareren Ausdrucks besser verwenden!)
* RIGHT-Join wird in LEFT-Join umgewandelt durch Vertauschen der Seiten (MY!)
* RIGHT-Join aus PortabilitÃ¤tsgrÃ¼nden besser als LEFT-Join formulieren (MY!)

Weitere spezielle Arten von Joins:

Begriff	Bedeutung
Self-Join	VerknÃ¼pfung einer Tabelle mit sich selbst (rekursiv)
Equi-Join	VerknÃ¼pfung Ã¼ber "="-Relation
Theta-Join	VerknÃ¼pfung Ã¼ber andere Rel. als "=" (!= <> < <= > >=)
Semi-Join	Analog Natural Join, dann Reduktion auf Spalten der 1. Tab.

Syntax:
```
  <TblReferences> =
      <TblRef> [, <TblRef>] ...

  <TblRef> = <TblFactor> | <JoinTable>

  <TblFactor> =
      <Tbl> [[AS] <Alias>] [<IdxHint>]]
    | <TblSubquery> [AS] <Alias>
    | (<TblRef>)
    | {OJ <TblRef> LEFT OUTER JOIN <TblRef>    # ODBC-Syntax, {...} hinschreiben
          ON <Cond>}

  <JoinTable> = <TblRef> [INNER | CROSS] JOIN                  <TblFactor> [<JoinCond>]
              | <TblRef> STRAIGHT_JOIN                         <TblFactor> [ON <Cond>]
              | <TblRef> {LEFT | RIGHT} [OUTER] JOIN           <TblRef>    <JoinCond>
              | <TblRef> NATURAL [{LEFT | RIGHT} [OUTER]] JOIN <TblFactor>

  <JoinCond> = ON <Cond>
             | USING (<Col>, ...)

  <IdxHint> = USE    {INDEX | KEY} [FOR JOIN] (<Idx1>, ...)
            | FORCE  {INDEX | KEY} [FOR JOIN] (<Idx1>, ...)
            | IGNORE {INDEX | KEY} [FOR JOIN] (<Idx1>, ...)
```

Beispiele:

```
  SELECT a.name, a.preis, h.name     # INNER JOIN
    FROM artikel a INNER JOIN hersteller h
      ON a.name = h.name             # MÃ¶glich: Spalten verschiedennamig
        WHERE a.preis > 200;

  SELECT a.name, h.name              # LEFT OUTER JOIN
    FROM artikel a LEFT OUTER JOIN hersteller h
      USING (name);                  # Notwendig: Spalten gleichnamig

  SELECT *                           # NATURAL JOIN
    FROM pers NATURAL JOIN age;      # Notwendig: Spalten gleichnamig
```

12) Mengen-Operationen

```
Mengen-Operationen erlauben die VerknÃ¼pfung von zwei oder mehr Selektionen
(Tabellen) zu einer Gesamtselektion:

  +------------------+-----------------------------------------------------+
  | Mengenoperation  | Bedeutung                                           |
  +------------------+-----------------------------------------------------+
  | UNION [DISTINCT] | Vereinigung (STD: doppelte DatensÃ¤tze weglassen)    |
  | UNION ALL        | Vereinigung (inkl. doppelte DatensÃ¤tze)             |
  | INTERSECT        | Durchschnitt = gemeinsame DatensÃ¤tze (NICHT in  MY!) |
  | EXCEPT/MINUS     | Differenz = 1. minus 2. Selektion (NICHT in MY!)    |
  +------------------+-----------------------------------------------------+

Eigenschaften:
  * Spaltenanzahl der verknÃ¼pften Selektionen muss gleich sein
  * Datentypen der Spalten mÃ¼ssen positionsweise kompatibel sein
    + Zeichenkette
    + Zahl
    + Datum
    + Zeit
    + ...
  * Spaltennamen des Ergebnisses = Spaltennamen der ersten Selektion
  * Doppelte DatensÃ¤tze werden standardmÃ¤Ã-^_ig weggelassen (Menge!),
    auÃ-^_er UNION ALL wird verwendet (STD: DISTINCT)
  * Sortierung der Daten wird zerstÃ¶rt (mÃ¶glich: Gesamtergebnis sortieren)

Beispiel:

  SELECT name, vorname FROM pers        # Selektion 1
    UNION                               # STD: DISTINCT = doppelte weglassen
  SELECT name, vorname FROM copy;       # Selektion 2
                                        #
  SELECT name, vorname FROM pers        # Selektion 1
    UNION ALL                           # Doppelte beibehalten
  SELECT name, vorname FROM copy;       # Selektion 2


13) Unterabfragen (Subqueries/Subselect)
----------------------------------------

Eigenschaften:
  * SELECT-Statement eingebettet ("nested") in anderem Statement
    + "Outer query", "Outer Statement"
    + "Inner query", "Subquery"
  * Seit MY!4.01 alles gemÃ¤Ã-^_ SQL-Standard erlaubt + einige Erweiterungen
  * IMMER in Klammern (...) zu setzen!
  * Verschachtelungstiefe beliebig
  * Ganz auÃ-^_en muss SELECT, INSERT, UPDATE, DELETE stehen
  * Tabelle kann nicht gleichzeitig modifiziert und gelesen werden
  * Als Ersatz fÃ¼r JOIN und UNION einsetzbar
  * Vorsicht bei NULL-Werten und leeren Tabellen!

Syntax:
  #  ---- Outer Query ----             --- Inner Query ---
  SELECT * FROM t1   WHERE col1    = (SELECT col1    FROM t2);
  SELECT * FROM t1 a WHERE a.col1  = (SELECT b.col1  FROM t2 b);
  SELECT * FROM t1   WHERE t1.col1 = (SELECT t2.col1 FROM t2);
  #  -- Outer Statement --            ----- Subquery ----

Ergebnis einer Subquery kann sein:
  * EIN Skalar (einzelner Wert):
      =  (<SubQuery>)
      <> (<SubQuery>)
      >  (<SubQuery>)
      <  (<SubQuery>)
      >= (<SubQuery>)
      <= (<SubQuery>)
  * EIN Datensatz (ROW-Subquery mit "Row Constructor"):
      ("wert1", "wert2", ...)    = (<SubQuery>)
      ROW("wert1", "wert2", ...) = (<SubQuery>)
      EXISTS (<SubQuery>)             # TRUE, wenn mind. 1 Datensatz zurÃ¼ck
      NOT EXISTS (<SubQuery>)         # TRUE, wenn kein Datensatz zurÃ¼ck
  * EINE Datenspalte (Synonym fÃ¼r ALL ist SOME):
      ... =  ANY (<SubQuery>)         # Mind. EIN Wert aus <SubQuery> gleich
      ... <> ANY (<SubQuery>)         # Mind. EIN Wert aus <SubQuery> ungleich
      ... >  ANY (<SubQuery>)         # Mind. EIN Wert aus <SubQuery> kleiner
      ... <  ANY (<SubQuery>)         # Mind. EIN Wert aus <SubQuery> grÃ¶Ã-^_er
      ... <= ANY (<SubQuery>)         # Mind. EIN Wert aus <SubQuery> kleiner gleich
      ... >= ANY (<SubQuery>)         # Mind. EIN Wert aus <SubQuery> grÃ¶Ã-^_er gleich
      ... IN     (<SubQuery>)         # Entspricht "= ANY"
      ... =  ALL (<SubQuery>)         # ALLE Werte aus <SubQuery> gleich
      ... <> ALL (<SubQuery>)         # ALLE Werte aus <SubQuery> ungleich
```

```
      ... >  ALL (<SubQuery>)          # ALLE Werte aus <SubQuery> grÃ¶Ã-^_er
      ... <  ALL (<SubQuery>)          # ALLE Werte aus <SubQuery> kleiner
      ... <= ALL (<SubQuery>)          # ALLE Werte aus <SubQuery> kleiner gleich
      ... >= ALL (<SubQuery>)          # ALLE Werte aus <SubQuery> grÃ¶Ã-^_er gleich
      ... NOT IN (<SubQuery>)          # Entspricht "<> ALL"
  * EINE Tabelle (mehrere DatensÃ¤tze mit mehreren Datenspalten)
      ... JOIN... (<SubQuery>) AS ...  # Verbundanweisung (beliebigen Typs)

Row Constructor:

  ("tom", 2)    = (SELECT sp1, sp2 FROM t3 WHERE ...)    # Syntax 1
  ROW("tom", 2) = (SELECT sp1, sp2 FROM t3 WHERE ...)    # Syntax 2

Fast identisch (bzw. ganz identisch bei UNIQUE Index Spalte in WHERE ...) mit:

    "tom" = (SELECT sp1 FROM t3 WHERE ...) AND
        2 = (SELECT sp2 FROM t3 WHERE ...)

Beispiel:

  SELECT name, geburtsdatum
    FROM pers, age
      WHERE geburtsdatum = (SELECT MAX(geburtsdatum) FROM age)
        AND pers.nr = age.nr;

  SELECT name, geburtsdatum
    FROM pers, age
      WHERE geburtsdatum >= ALL (SELECT geburtsdatum FROM age)
        AND pers.nr = age.nr;

  SELECT name, preis
    FROM artikel
      WHERE preis = (SELECT MIN(preis) FROM artikel)
         OR preis = (SELECT MAX(preis) FROM artikel);

  SELECT name, preis
    FROM artikel
      WHERE preis <= ANY (SELECT preis FROM artikel);

  DELETE FROM t1
  WHERE s11 > ANY
    (SELECT COUNT(*) FROM t2                  # Subquery 1
     WHERE NOT EXISTS
       (SELECT * FROM t3                      # Subquery 2
        WHERE ROW(5 * t2.s1, 77) =
          (SELECT 50, 11 * s1 FROM t4         # Subquery 3
           UNION
           SELECT 50, 77 FROM                 # Subquery 4
             (SELECT * FROM t5) AS t5)));     # Subquery 5


14) Transaktionen
-----------------

Ein DBMS soll die Datenkonsistenz sichern bei:
  * Gleichzeitigem (Schreib)Zugriff mehrerer Benutzer ("Concurrency")
  * Server-Absturz (Stromausfall)
  * Hardwaredefekten
  * Programmierfehlern
  * Netzwerkfehlern (Verbindungsabbruch)
  * Zugriffsrechte-Problemen

Transaktion (TA) = Gruppe von zusammengehÃ¶renden SQL-Anweisungen
  A) Entweder gemeinsam VOLLSTÃM-^DNDIG ausgefÃ¼hrt
  B) Oder GAR NICHT ausgefÃ¼hrt (in Ausgangszustand zurÃ¼ckversetzt)
     + Bei Widerspruch
     + Bei Fehler
     + Bei Zugriffsverletzung
    Nachteil: LÃ¤ngere AusfÃ¼hrungszeit
              Gegenseitige Behinderung
              Deadlocks (ÃM-^\ber-Kreuz-Exklusiv-Zugriffe) mÃ¶glich

ACID-Eigenschaften einer Transaktion TA:

  +--------------+-----------------------------------------------------+
  | Eigenschaft  | Beschreibung                                        |
  +--------------+-----------------------------------------------------+
  | A)tomicity   | TA entweder vollstÃ¤ndig oder gar nicht durchgefÃ¼hrt |
  | C)onsistency | DB vor+nach TA konsistent (nicht unbedingt wÃ¤hrend!)|
  | I)solation   | Gleichzeitig ablaufende TAs beeinflussen sich nicht |
  | D)urability  | Ergebnis einer erfolgreichen TA steht dauerhaft in DB |
  +--------------+-----------------------------------------------------+
```

```
Transaktionen Ã¼ber mehrere SQL-Anweisungen:
  * NUR mit Engines "InnoDB" + "BDB" (MY!)
  * Bei allen anderen Engines ist jede einzelne SQL-Anweisung eine TA
    (sofern der MySQL-Server nicht abstÃ¼rzt)
  * Mit "MyISAM" nur Locken kompletter Tabellen mÃ¶glich (LOCK/UNLOCK)
  * Engine pro Tabelle wÃ¤hlbar --> Optimale Kombination auswÃ¤hlen
  * TA-Locking-Verhalten
    + InnoDB: Row-Level
    + BDB:    Page-Level (evtl. mehrere "benachbarte" Rows auch gesperrt)

StandardmÃ¤Ã-^_ig ist bei MySQL "Autocommit"-Modus eingeschaltet (MY!):
  * D.h. JEDE Anweisung fÃ¼r sich stellt Transaktion dar
  * Startet eine Transaktion mit "BEGIN", wird Autocommit-Modus abgeschalten
    (d.h. COMMIT/ROLLBACK zum Abschluss/Abbruch der Transaktion notwendig)
  * Deaktivieren sorgt dafÃ¼r, dass "BEGIN" nicht mehr notwendig ist und alles
    bis zum nÃ¤chsten "COMMIT/ROLLBACK" automatisch eine Transaktion darstellt
  * Ã-^Ndern des Autocommit-Modus durch:
      SET AUTOCOMMIT = 1;   # Jede Aktion ist TA, BEGIN startet TA bis COMMIT
      SET AUTOCOMMIT = 0;   # StÃ¤ndig TA, COMMIT/ROLLBACK beendet+startet neue

Transaktionen manuell einleiten und mit COMMIT abschlieÃ-^_en bzw. mit ROLLBACK
abbrechen (AUTOCOMMIT von BEGIN automatisch deaktiviert):

  START TRANSACTION;                      # TA beginnen (auch BEGIN [WORK])
  ...                                     #
  SAVEPOINT <Name>;                       # Marke <Name> setzen (Label)
  ...                                     #
  RELEASE SAVEPOINT <Name>;               # Savepoint <Name> freigeben
  ...                                     #
  ROLLBACK [WORK] TO SAVEPOINT <Name>;    # Bis Marke <Name> zurÃ¼cknehmen
  ...                                     #
  COMMIT [WORK];                          # TA abschlieÃ-^_en (ALLES!)
  ROLLBACK [WORK];                        # TA abbrechen (ALLES!)

Folgende Anweisungen (meist DDL = Data Definition Language) beenden Transaktion
ebenfalls (neben COMMIT/ROLLBACK), d.h. fÃ¼hren automatisch "COMMIT" durch:

  +---------------------+-----------------------------------------------------+
  | CREATE DATABASE ... | Datenbank anlegen                                   |
  | DROP DATABASE ...   | Datenbank lÃ¶schen (inkl. Tabellen)                  |
  | CREATE TABLE ...    | Tabelle anlegen                                     |
  | DROP TABLE ...      | Tabelle lÃ¶schen (inkl. Struktur)                    |
  | TRUNCATE TABLE ...  | Tabellendaten lÃ¶schen (da DROP + CREATE TABLE)      |
  |                     | (DELETE FROM <Tbl>; ohne WHERE beendet TA nicht!)   |
  | ALTER TABLE ...     | Tabellen-Struktur Ã¤ndern                            |
  | RENAME TABLE ...    | Tabelle umbenennen                                  |
  | CREATE INDEX ...    | Index anlegen                                       |
  | DROP INDEX ...      | Index lÃ¶schen                                       |
  | LOCK TABLES ...     | Tabellen sperren (bis UNLOCK TABLES)                |
  | BEGIN ...           | TA sind nicht verschachtelbar!                      |
  | FLUSH LOGS ...      | Alle/einige Caches auf Platte schreiben             |
  +---------------------+-----------------------------------------------------+

Hinweise:
  * Transaktionen sind nicht verschachtelbar (MY!)
  * Statt "BEGIN [WORK]" auch "START TRANSACTION" mÃ¶glich
    (in Prozeduren verwenden, da "BEGIN" dort einen Block einleitet!)
  * Tritt KEIN EINZIGER Fehler auf, werden ALLE Ã-^Dnderungen in TA durchgefÃ¼hrt
  * Tritt EIN Fehler auf, werden ALLE bisherigen Ã-^Dnderungen in TA widerrufen
    (und die noch folgenden Anweisungen nicht mehr durchgefÃ¼hrt)
  * Bei Anwendung einer Transaktion auf mind. 1 transaktions-UN-sichere Tabelle
    wird jede Ã-^Dnderung sofort durchgefÃ¼hrt (MyISAM, entspricht AUTOCOMMIT=1)
  * "ROLLBACK" auf nicht transaktionssicherer Tabelle fÃ¼hrt zu Fehlermeldung
  * Protokolldateien
    + ASCII: Zeichnet "ROLLBACK" auf
    + BinÃ¤r: Zeichnet "ROLLBACK" NICHT auf

Transaktions-Level (Isolations-Ebene) einstellen (STD: 2 = REPEATABLE READ):
  * SESSION = FÃ¼r AKTUELLE Sitzung
    GLOBAL  = FÃ¼r ALLE danach neu gestartete Sitzungen (nicht fÃ¼r bestehende!)
    Sonst   = FÃ¼r NÃ-^DCHSTE Transaktion (1x!)
  * Beim Start des MySQL-Servers festlegen durch:
      --transaction-isolation = <Level>       # STD: 2

  SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL <Level>;

  +------------------+---+------------+-----------------------------------+
  |                  |   |            |  F e h l e r m Ã¶ g l i c h k e i t |
  |      Level       | N | Datensatz- +-------+-------+-----------+-------+
  |                  | r |   sperre   | Lost  | Dirty | Nonrepeat | Phantom |
  |                  |   |            | Update| Read  | able Read | Read    |
```

--	-	--	ja	ja	ja	ja	
READ UNCOMMITTED	0	--	--	ja	ja	ja	
READ COMMITTED	1	write	--	--	ja	ja	Oracle-STD
REPEATABLE READ	2	read/write	--	--	--	ja	Mysql-STD
SERIALIZABLE	3	read/write	--	--	--	--	

Hinweise:
* Standard von MySQL ist Level 2 (REPEATABLE READ)
* Oracle
  + Standard ist Level 1 (READ COMMITTED)
  + Kennt nicht Level 0 (READ UNCOMMITTED) und Level 2 (REPEATABLE READ)
  + Alternative Level-Namen bei Oracle
    READ UNCOMMITTED = SNAPSHOT
    REPEATABLE READ = SNAPSHOT TABLE STABILITY
  + Weitere Angaben nach TRANSACTION mÃ¶glich:
    READ WRITE | READ ONLY # Daten Ã¤ndern (STD) / nur lesen
    WAIT | NO WAIT # Auf Abschluss anderer TA mit Zugriff auf gl.
    # Tab. warten (STD), sonst TA sofort abbrechen

Beispiel:

```
@ueberweisung = 1000;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN;
  UPDATE konto SET betrag = betrag - @ueberweisung WHERE nr = 123456789;
  UPDATE konto SET betrag = betrag + @ueberweisung WHERE nr = 987654321;
COMMIT;
```

15) Locking

Tabellen/View-Lock:
* Tabellen/Views fÃ¼r exklusiven Zugriff sperren + wieder freigeben
* Differenzierung: Schreib-Lock oder Lese/Schreib-Lock
* Alle angegebenen Tabellen werden vollstÃ¤ndig gesperrt
* Funktioniert bei jeder Engine!
* WRITE hat hÃ¶here PrioritÃ¤t als READ

```
LOCK TABLES                          # Sperren mehrerer Tabellen gleichzeitig:
  table1              WRITE,         # Lese+Schreibschutz fÃ¼r alle anderen
  table1 AS alias1    READ,          # Schreibschutz (Lesen fÃ¼r alle erlaubt)
  table2 AS alias2    READ LOCAL,    # INSERT zulassen solange kein Konflikt
  table3 LOW_PRIORITY WRITE;         # Erst sperren wenn kein READ-Lock (warten)
...                                  # Operationen auf gelockten Tabellen ...
UNLOCK TABLES;                       # Freigeben aller Locks
```

Lock-Typ	Bedeutung
WRITE	Lese- und Schreibschutz fÃ¼r alle auÃM-^_er Anforderer
READ	Schreibschutz fÃ¼r alle auÃM-^_er Anf. (Lesen fÃ¼r alle OK)
READ LOCAL	Schreibschutz, aber INSERT OK solange kein Konflikt
LOW_PRIORITY WRITE	Erst sperren wenn kein READ-Lock (warten)

Tabellen gegen Zugriffe durch andere Threads sperren
(alle benÃ¶tigten Tabellen gleichzeitig!):
* Wartet, bis alle anderen Zugriffe auf die Tabellen beendet sind
  LOCK TABLES <Tbl> <Typ>, ...;
* <Typ>
  IN SHARE MODE #
  IN SHARE MODE NOWAIT #
  IN EXCLUSIVE MODE #
  IN EXCLUSIVE MODE NOWAIT #

Beispiel:

```
LOCK TABLES pers WRITE;
ALTER TABLE pers DISABLE KEYS;
INSERT INTO pers (nr, vorname, name) VALUES
  (1, "Thomas",  "Birnthaler"),
  (2, "Markus",  "Mueller"),
  (8, "Andrea",  "Bayer");
ALTER TABLE pers ENABLE KEYS;
UNLOCK TABLES;
```

Hinweise:
* LOCK/UNLOCK beziehen sich auf aktuelle Sitzung
* LOCK fÃ¼hrt implizites COMMIT durch

```
  * LOCK (und Transaktions-Beginn) fÃ¼hrt implizites UNLOCK durch
  * Wartet, bis ALLE Tabellen gemeinsam gelockt werden kÃ¶nnen
  * Deadlock-frei (Reihenfolge der Locks durch Datenbank gewÃ¤hlt)
  * Lock auf temp. Tabelle erlaubt aber ignoriert (sowieso sitzungsbezogen)
  * LOCK auf View lockt alle ihre Basis-Tabellen!
  * Nach LOCK nur auf gelockte Tabellen zugreifbar (auf andere nicht)!
  * Mehrfachzugriff per Tabellen-Alias braucht Mehrfachlock mit diesen Aliasen!
  * Transaktion-Simulation fÃ¼r MyISAM (und andere nicht TA-fÃ¤hige Engines)
  * Tabelle lÃ¶schen nach LOCK mÃ¶glich, aber nicht Tabelle anlegen oder TRUNCATE

Advisory Lock (anwendungsbezogen):
  * Frei definierbare Locks (per Name)
  * Anwendungen mÃ¼ssen Lock selbst anfordern
  * Anwendungen, die sich nicht daran beteiligen, werden nicht mit einbezogen
    (kÃ¶nnen machen was sie wollen)
  * Deadlock mÃ¶glich (DB kÃ¼mmert sich nicht um ihre AuflÃ¶sung)

  +--------------------------+-------------------------------------------------+
  | Funktion                 | Bedeutung                                       |
  +--------------------------+-------------------------------------------------+
  | GET_LOCK(<Name>, <sec>)  | Lock <Name> beantragen (Timeout nach <Sec> Sek. |
  |                          | (Erg: 1=erhalten, 0=nicht erh., NULL=Fehler)    |
  | IS_FREE_LOCK(<Name>)     | Lock <Name> frei? (Erg: 1=ja, 0=nein)           |
  | IS_USED_LOCK(<Name>)     | Lock <Name> belegt? (Erg: 1=ja, 0=nein)         |
  | RELEASE_LOCK(<Name>)     | Lock <Name> freigeben                           |
  +--------------------------+-------------------------------------------------+


16) Views (Sichten)
-------------------

Views (Sichten) sind vordefinierte gespeicherte benannte Abfragen (SELECT-
Anweisungen), sie werden auch "virtuelle" Tabellen (derived table) genannt.

Zweck:
  * Tabellen und Spalten umbenennen (um Ã„nderungen nach AuÃ„nderungen zu verbergen)
  * ZusÃ¤tzliche berechnete Spalten einfÃ¼hren
  * ZugriffsbeschrÃ¤nkung (per Sichtbarkeit von Spalten und Datenzeilen)
  * Zugriff vereinfachen (VerknÃ¼pfung mehrerer Tab. sieht wie eine Tab. aus)
  * Verbergen konkrete Tabellenstruktur (Ã¤nderungsfreundlich)
  * Schrittweises "Refactoring" eines Schemas (alte Tab. wg. Alt-SW beibehalten)
    + Migration/Umprogrammierung von Applikationen ermÃ¶glichen

Eigenschaften:
  * Ableitung aus einer oder mehreren Basis-Tabellen oder anderen (Unter-)Views
    auf der Basis von Joins, Unions und Unterabfragen
    + Spalten oder AusdrÃ¼cke mit Funktionen, Konstanten, Spalten, Operatoren...
    + ORDER BY ist mÃ¶glich (aber evtl. bei weiterem ORDER BY ignoriert)
  * Verwendbar wie eine normale Tabelle (Pseudotabelle)
    + Tabellen und Views teilen sich selben Namensraum in einer Datenbank
  * Spaltennamen <Col>
    + Weglassen --> Spaltennamen automatisch durch SELECT gebildet
    + Angeben   --> Anzahl Spalten von View und SELECT MUSS gleich sein!
  * Ã„nderungen Ã¼ber Views mÃ¶glich:
    + Daten abfragen: Immer
    + Daten Ã¤ndern:   Unter bestimmten Bedingungen
    + Daten einfÃ¼gen: Unter bestimmten Bedingungen
    + Daten lÃ¶schen:  Unter bestimmten Bedingungen
  * View wird zum Definitionszeitpunkt "eingefrohren"
    + "Updatability Flag" wird erstellt (UPDATE/INSERT prinzipiell mÃ¶glich)
    + Ã„nderungen an Unter-Tabellen oder -Views wirken sich nicht aus
    + LÃ¶schen von Unter-Tab/Views --> Fehlermeldung erst bei View-Verwendung
  * BeschrÃ¤nkung der EinfÃ¼ge-Daten auf ErfÃ¼llung der View-WHERE-Klausel mÃ¶glich
    (WITH CHECK OPTION --> PrÃ¼ft, ob Daten mit WHERE-Bed. der View ausgefiltert)

BeschrÃ¤nkungen:
  * Nur dann aktualisierbar ("Updatable View", d.h. INSERT/UPDATE/DELETE), wenn
    1:1-Beziehung zwischen DatensÃ¤tzen in View und Basis-Tabellen vorhanden
    + Nicht mÃ¶glich bei Verwendung von Aggregat-Funktionen SUM, MIN, MAX, COUNT,
      sowie DISTINCT, GROUP BY, HAVING, UNION, UNION ALL, Subqueries, ...
    + Per Ausdruck abgeleitete View-Spalten sind nicht updatebar
    + Alle zu Ã¤ndernden Spalten mÃ¼ssen in EINER Basis-Tabelle liegen
    + Nicht mÃ¶glich bei Realisierung der View per temp. Tabelle (TEMPTABLE)
  * Neue SÃ¤tze einfÃ¼gbar ("Insertable", d.h. INSERT) wenn (zusÃ¤tzlich zu oben)
    + Kein Basis-Spaltenname doppelt verwendet
    + Alle Spalten der Basis-Tabelle ohne Defaultwert in View enthalten
    + Keine per Ausdruck abgeleiteten View-Spalten
    + Nur EINE Basis-Tabelle ist betroffen
  * Kein Index auf Views mÃ¶glich
    + Ausnutzung von Indices bei MERGE mÃ¶glich, bei TEMPTABLE nicht
  * Keine temporÃ¤re Tabelle als Basis mÃ¶glich
  * Kein Trigger mit View assozierbar (manche DB erlauben "INSTEAD OF" Trigger)
```

```
  * Ableitungs-Algorithmus MERGE/TEMPTABLE ist Eigenschaft der View-Definition
    und unabhÃ¤ngig von der darauf auszufÃ¼hrenden SQL-Anweisung (MY!)
  + FÃ¼hren evtl. Entwickler in die Irre (sieht einfach aus obwohl sehr komplex)
  + Eigene Optimierung (getrennt von Tabellen) --> Nicht so gut ausgetestet
  + Schlecht editierbar da Originaltext verloren geht (MY!)
    --> S281 (Roland Bouman)
  * In Liste der Tabellen mit enthalten --> Mit PrÃ¤fix "v_" und "t_" unterscheidbar machen

"Materialized" View (nicht in MY!):
  + Unsichtbare Tabelle die periodisch upgedated wird
  * Kostet Speicherplatz
  * Wird nicht bei jeder VerÃ¤nderung der Basis-Tabellen aktualisiert
  * VerhÃ¤lt sich wie statische Tabelle --> Schneller als normale View
  * FÃ¼r hÃ¤ufige View-Abfragen, deren Basis-Tabellen sich selten Ã¤ndern
  * Simulierbar per EVENT/TRIGGER, der View asynchron/synchron als echte Tabelle erstellt (MY!)
  * "Permanente temporÃ¤re" Tabelle

Definition von Views:

  CREATE [OR REPLACE]                                 # REPLACE=View ersetzen
    [ALGORITHM = (MERGE | TEMPTABLE | UNDEFINED)]     # (STD: UNDEFINED, MY!)
    [DEFINER = {<User> | CURRENT_USER}]               #
    [SQL SECURITY {DEFINER | INVOKER}]                #
      VIEW <View> [(<Col>, ...)]                      # Name + opt. Spaltennamen
        AS <Select>                                   # Beliebige SELECT-Anweisung
          [WITH [CASCADED | LOCAL] CHECK OPTION];     # ErfÃ¼llt INSERT WHERE-Bed.?

Konstruktions-Algorithmus (ALGORITHM = ...):
  * MERGE:     View-Definition textuell in View-verwendende Anweisung einsetzen
               ("Query Rewrite", sogar fÃ¼r verschachtelte Views mÃ¶glich)
               + Effizienter
               + Updatebar
               + Locks lÃ¤nger gehalten
               + Nicht verwendbar bei Aggregat-Funktionen SUM, MIN, MAX, COUNT
                 DISTINCT, GROUP BY, HAVING, LIMIT, UNION, UNION ALL, Subqueries
                 (allgemein falls 1:1-Beziehung zwischen Zeilen in
                  Basis-Tabellen und Ergebnis-Tabelle verloren geht)
  * TEMPTABLE: TemporÃ¤re Tabelle fÃ¼r View-Ergebnis erzeugen (jedesmal!)
               + Weniger Effizient
               + Nicht updatebar
               + Locks kÃ¼rzer
  * UNDEFINED: MERGE (vorzugsweise) oder TEMPTABLE autom. aussuchen (MY!)

PrÃ¼fung ob INSERT die View-WHERE-Bedingung erfÃ¼llt (WITH ... CHECK OPTION):
  * CASCADED: Definierte View und alle Unter-Views prÃ¼fen (STD)
  * LOCAL:    Nur definierte View prÃ¼fen, nicht Unter-Views

Alle Views auflisten (zusammen mit Tabellen, gleicher Namespace):

  SHOW TABLES;

View-Definition anzeigen:

  SHOW CREATE VIEW <View>;

View Ã¤ndern (Charakteristika und Inhalt, nicht Name):

  ALTER VIEW <View> [(<Col>, ...)]
    [ALGORITHM = (MERGE | TEMPTABLE | UNDEFINED)]
      AS <Select>
        [WITH [CASCADED | LOCAL] CHECK OPTION];

View(s) umbenennen (aber nicht in andere Datenbank verschieben!,
stellt eine "atomare" Operation dar, auch bei Umbenennung mehrerer Views):

  RENAME TABLE <View> TO <NewView> [, ...];
  ALTER TABLE <View> RENAME TO <NewView>;

View lÃ¶schen:

  DROP VIEW <View>;
  DROP VIEW IF EXISTS <View>;   # Kein Fehler falls nicht existent

Algorithmus ermitteln den View benutzt:

  EXPLAIN SELECT * FROM <View>;            # "TEMPTABLE" falls "DERIVED" in Ausgabe
  EXPLAIN EXTENDED SELECT * FROM <View>;   # "TEMPTABLE" falls "DERIVED" in Ausgabe

Beispiele fÃ¼r Views:

  #---------------------------------------------------------------------------
  # View 1
```

```
#------------------------------------------------------------------------
DROP TABLE IF EXISTS t1;
CREATE TABLE t1 (
  anz   INT,
  preis INT
);

INSERT INTO t1 VALUES
  (3, 50),
  (7, 80),
  (9, 99),
  (1, 12);

DROP VIEW IF EXISTS v1;

CREATE VIEW v1 AS
  SELECT anz, preis, anz * preis AS "wert"
    FROM t1;

SELECT * FROM v1;

DROP VIEW v1;
DROP TABLE t1;

#------------------------------------------------------------------------
# View 2
#------------------------------------------------------------------------
DROP TABLE IF EXISTS t2;
CREATE TABLE t2 (
  nr   INT,
  name CHAR(30)
);

INSERT INTO t2 VALUES
  (1, "a"),
  (2, "abcde"),
  (5, "test"),
  (4, "text");

DROP VIEW IF EXISTS v2;

CREATE VIEW v2 AS
  SELECT *, "konstant"
    FROM t2 WHERE nr = LENGTH(name);

SELECT * FROM v2;

DROP VIEW v2;
DROP TABLE t2;

#------------------------------------------------------------------------
# View 3
#------------------------------------------------------------------------
DROP VIEW IF EXISTS v3;

CREATE VIEW v3 (Artikel, Hersteller) AS
  SELECT a.name, h.name
    FROM artikel AS a, hersteller AS h
      WHERE a.name = h.name;

#------------------------------------------------------------------------
# View 4
#------------------------------------------------------------------------
DROP VIEW IF EXISTS v4;

CREATE VIEW v4 (Name, Artikel) AS
  SELECT p.name, a.name
    FROM artikel a, bestellung b, pers p
      WHERE a.nr     = b.nr
       AND p.nr      = b.nr
       AND p.name    = "MÃ¼ller"
       AND p.vorname = "Oskar";

#------------------------------------------------------------------------
# View 5
#------------------------------------------------------------------------
DROP VIEW IF EXISTS v5;

CREATE VIEW v5 AS
  SELECT * FROM artikel       # ALLE Spaltennamen Ã¼bernommen
    WHERE preis > 200         # Bedingung an DatensÃ¤tze
      WITH CHECK OPTION;      # PrÃ¼fen ob Bed. bei Einf./Ã¤ndern erfÃ¼llt
```

17) Variablen

MySQL kennt folgende Variablen-Arten:

* @@SERVER: Einstellungen/Attribute MySQL-Server (fix) --> SHOW VARIABLES;
  --> SHOW GLOBAL VARIABLES;
  --> SHOW LOCAL VARIABLES;
* @SESSION: Sitzungsbezogen (frei wÃ¤hlbar, read-write)
* LOKAL: Stored Procedure/Event/Trigger (frei wÃ¤hlbar, read-write)
* STATUS: Messwerte MySQL-Server (fix, read-only) --> SHOW/FLUSH STATUS;

<Var> ist im Folgenden ein Variablen-Bezeichner (Identifier), d.h. er kann aus den Zeichen A-Z, a-z, 0-9, _ und $ bestehen, fÃ¼hrende Ziffern sind nicht erlaubt --> 2f) Identifier (Bezeichner).

Typ	Bedeutung
@@<var>	SERVER-Variable (klein geschrieben mit "_") + Vom Server festgelegt - Feste Anzahl (etwa 450 je nach MySQL-Version) - FIXE Namen <var> - Enthalten Einstellungen/Attribute des MySQL-Servers - Neue Version --> kennt neue zusÃ¤tzlich / veraltete nicht mehr - Server-Neustart --> Neu initialisiert - Lesbar + sehr viele schreibbar (evtl. mit SUPER-Berechtigung) + Startwert - Fest eingebaut in "mysqld"-Server ODER - Per Options-Parameter beim Server-Start Ã¼bergeben ODER - Aus Server-Konfig "my.cnf/ini" beim Start + Viele doppelt vorhanden - GLOBAL: Server-weiter (Default)wert - SESSION: FÃ¼r akt. Session (GLOBAL-Kopie bei Verb.-Aufbau) + GÃ¼ltigkeitsbereich: - Einige GLOBAL STATISCH (nur beim Server-Start einstellbar) - Einige GLOBAL DYNAMISCH (jederzeit mit SUPER-Recht Ã¤nderbar) - Viele SESSION-bezogen (jederzeit fÃ¼r akt. Sitzung Ã¤nderbar) + Session-Wert wird mit globalem Wert initialisiert - Session-Wert geÃ¤ndert --> "Ã¼berdeckt" Globalen Wert - Verschwindet am Sitzungsende + Auflisten per SHOW [GLOBAL|SESSION] VARIABLES [LIKE "..."]
@<var>	SESSION-Variable (Sitzungsbezogen): + Benutzerdefiniert - Name <var> frei wÃ¤hlbar - Beliebig viele - NICHT ZU DEKLARIEREN (Datentyp beliebig, 1 Wert) - Lesbar + schreibbar + GÃ¼ltigkeitsbereich: - Auf Sitzung beschrÃ¤nkt --> Gleichnamige versch. Sitzungen Ã¼berschneiden sich NICHT - Verschwinden am Ende der Sitzung (Verbindungsende) + Nicht per SQL-Anweisung auflistbar
<var>	LOKALE Variable + Aufruf-Parameter in Stored Proc./Event/Trigger + Benutzerdefiniert - Name <var> frei wÃ¤hlbar - Beliebig viele - Vor Verwendung zu DEKLARIEREN (mit SQL-Datentyp) (z.B. DECLARE betrag FLOAT DEFAULT 0.0;) - Lesbar + schreibbar + GÃ¼ltigkeitsbereich: - Auf Stored Procedure/Event/Trigger/... beschrÃ¤nkt - Verschwinden am Ende von Stored Procedure/Event/Trigger/... + Nicht per SQL-Anweisung auflistbar
<Var>	STATUS-Variable (groÃM-^_er Anfangsbuchstabe, Rest klein mit "_") + Vom Server festgelegt - Feste Anzahl (etwa 350 je nach MySQL-Version) - FIXE Namen <Var> - Enthalten Messwerte des MySQL-Servers - Neue Version --> kennt neue zusÃ¤tzlich / veraltete nicht mehr - Server-Neustart --> Neu initialisiert - Nur lesbar + GÃ¼ltigkeitsbereich: - Serverweit gÃ¼ltig - Mit FLUSH STATUS auf 0-Wert zurÃ¼cksetzen (SUPER-Berechtigung) + Auflisten per SHOW STATUS [LIKE "..."]

```
Eigenschaften:
  * Variablenname darf kein Objektname und SQL-SchlÃ¼sselwort sein
    (auÃM-^_er er wird durch `...` eingerahmt --> 2f) Identifier (Bezeichner))
  * Bis MY!4.1 wird GROSS/kleinschreibung unterschieden, ab MY!5.0 nicht mehr
  * ÃM-^Dnderung von Globalen Server-Variablen --> SUPER-Recht notwendig

Session-Variable @v einen Wert zuweisen (Variable dabei evtl. neu angelegt):

  SET @v = 3;                          # Schlecht, besser ":=" verwenden!
  SET @v := 3;                         # Besser als Verwendung von "="!
  @v = 3;                              # Fehler (SET davor notwendig)
  @v := 3;                             # Fehler (SET davor notwendig)
  SELECT @v := 3;                      # Zuweisung eines SELECT-Ergebnisses
  SELECT @v := COUNT(*) FROM pers;     #   (analog)
  SELECT COUNT(*) FROM pers INTO @v;   #   (analog)
  SELECT name, vorname FROM pers       # Mehrere Variablen gleichzeitig
    WHERE nr = 1 INTO @v1, @v2;        #   MY!5.0

ACHTUNG: Folgende Anweisung ist KEINE Zuweisung sondern ein VERGLEICH der
Session-Variablen @v mit dem Ergebnis von "SELECT COUNT(*) FROM pers". Gibt je
nach Wert von @v und dem SELECT-Ergebnis TRUE oder FALSE zurÃ¼ck (Wert von @v
bleibt unverÃ¤ndert):

  SELECT @v = COUNT(*) FROM pers;      # Fehler: Vergleich (statt Zuweisung)
  SELECT @v := COUNT(*) FROM pers;     # OK: Zuweisung

Server-Variable (Session oder Global) auflisten:

  SHOW SESSION VARIABLES;              # ALLE sessionbezogenen Server-Var.
  SHOW LOCAL   VARIABLES;              # ALLE sessionbezogenen Server-Var.
  SHOW GLOBAL  VARIABLES;              # ALLE globalen Server-Var.
  SHOW         VARIABLES;              # ALLE Server-Var. (Session, sonst Global)
  SHOW SESSION VARIABLES LIKE "b%";    # PASSENDE Server-Var. (Session)
  SHOW LOCAL   VARIABLES LIKE "b%";    # PASSENDE Server-Var. (Session)
  SHOW GLOBAL  VARIABLES LIKE "b%";    # PASSENDE Server-Var. (Global)
  SHOW         VARIABLES LIKE "b%";    # PASSENDE Server-Var. (Sess., sonst Global)

Status-Variablen auflisten:

  SHOW STATUS;                         # Alle Status-Var. (Server)
  SHOW STATUS LIKE "b%";               # Einige Status-Var. (Server)

Variablen-Wert anzeigen:

  SELECT wait_timeout;                 # Lokale Var. in Stored Procedure/Event
  SELECT @wait_timeout;                # Session
  SELECT @@SESSION.wait_timeout;       # Server-Session
  SELECT @@GLOBAL.wait_timeout;        # Server-Global
  SELECT @@wait_timeout;               # Server-Session ODER -global
                                       # (Session "Ã¼berdeckt" Global)

Einige Server-Variable sind nur Global verfÃ¼gbar:

  SELECT @@SESSION.binlog_cache_size; # Fehler da nur Global!
  SELECT @@GLOBAL.binlog_cache_size;  # Global
  SELECT @@binlog_cache_size;          # Automatisch Global

Belegung von Variablen mit einem Wert:

  SET wait_timeout           = 900;   # 0) Lokal (nur in Stored Procedure)
  SET @wait_timeout          = 901;   # 1) Session
  SET @@SESSION.wait_timeout = 902;   # 2a) Server-Session
  SET SESSION wait_timeout   = 903;   # 2b) Server-Session, Ã¼berschreibt 2a)
  SET @@GLOBAL.wait_timeout  = 904;   # 3a) Server-Global
  SET GLOBAL wait_timeout    = 905;   # 3b) Server-Global, Ã¼berschreibt 3a)

Erneut Variablen-Wert anzeigen:

  SELECT wait_timeout;                 # 0) Stored Procedure  --> 900
  SELECT @wait_timeout;                # 1) Session           --> 901
  SELECT @@SESSION.wait_timeout;       # 2) Server-Session    --> 903
  SELECT @@GLOBAL.wait_timeout;        # 3) Server-Global     --> 905
  SELECT @@wait_timeout;               # 4) Server-Session ODER -global --> 903
                                       #    (Session "Ã¼berdeckt" Global)


18) Prepared Statements (Vorbereitete Anweisungen)
--------------------------------------------------

Prepared Statements sind vorbereitete SQL-Anweisungen ("Template/Skeleton") mit
PLATZHALTERN der Form "?" (oder ":name", die erst beim Aufruf mit WERTEN (aus
```

```
Variablen) gefÃ¼llt werden. Ein Prepared Statement ist entweder
  * ein String-Literal "..." oder
  * eine Text-Variable mit einer SQL-Anweisung als Inhalt

Eigenschaften:
  * Nur lokal pro Sitzung existent (liegen nicht auf Platte!)
    --> Gleichnamige verschiedener Sitzungen Ã¼berschneiden sich nicht!
  * Prepared Statement --> Statement-Handle <PrepStmt>
  * Hauptzweck
    + In Stored Procedures einsetzbar ("dynamic SQL")
    + EinschrÃ¤nkungen:
      - Datenbank-, Tabellen- und Spalten-Namen sind Identifier
        --> Nicht per PLATZHALTER parametrisierbar
      - LIMIT und OFFSET parametrisierbar seit MY!5.6.x
  * Geschwindigkeitsgewinn
    + SQL-Anweisung nur 1x zu kompilieren
    + Optimierung nur 1x durchzufÃ¼hren
    + Daten binÃ¤r statt ASCII-Text (DATE 10 --> 3, BLOB, TEXT, ..)
    + Bei jeder AusfÃ¼hrung nur Parameter fÃ¼r Platzhalter senden
  * Sicherheitsgewinn
    + Escape/Quote unnÃ¶tig
    + "SQL-Injection" unmÃ¶glich (Benutzer-Eingabe kann SQL-Anweisung nicht manip)
    + REGEL: "Never trust incoming Data!"
    + RegulÃ¤re AusdrÃ¼cke zum PrÃ¼fen der Benutzer-Daten bieten sich an!
  * Erneut Prepared Statement mit gleichem Namen --> Altes wird Ã¼berschrieben
  * BinÃ¤res Protokoll nur von API genutzt, nicht vom SQL-Interface

Drei prinzipielle Typen:
  * Client-side: Client schickt Endergebnis der Ersetzung (Emulation)
  * Server-side: Statement Identifier, BinÃ¤res Protokoll, Parameter (API)
  * SQL-Interface: Trennung PREPARE + EXECUTE, Textprotokoll, SQL-Variablen

Optimierung mehrstufig:
  * Bei Definition:
    + SQL-Text parsen
    + Negationen entfernen
    + Subqueries umschreiben
  * Erste AusfÃ¼hrung:
    + Nested Joins vereinfachen
    + OUTER JOIN --> INNER JOIN falls mÃ¶glich
  * Jede AusfÃ¼hrung:
    + Partitionen weglassen
    + COUNT(), MIN(), MAX() weglassen wo mÃ¶glich
    + Konstante Subexpression weglassen
    + Konstante Tabellen erkennen
    + IdentitÃ¤ten verteilen
    + Zugriffsmethoden "ref", "range" und "index_merge" analysieren + optimieren
    + Join Reihenfolge optimieren

BeschrÃ¤nkungen:
  * Nur lokal pro Sitzung existent (liegen nicht auf Platte!)
    + Statement-Handle <PrepStmt> nur darin verwendbar
  * In Prozeduren verwendbar, aber NICHT in Funktionen, Triggern und Events
  * Nicht im Query-Cache (vor MY!5.1, spÃ¤ter schon)
  * Platzhalter nur fÃ¼r Datenwerte, NICHT aber fÃ¼r SQL-SchlÃ¼sselworte und
    fÃ¼r Datenbank-, Tabellen- und Spalten-Namen einsetzbar
    + LIMIT/OFFSET-Werte mÃ¶glich seit MY!5.6.X
  * Nur Variable in Platzhalter einsetzbar, keine AusdrÃ¼cke oder Literale
    + Anzahl Variable muss gleich Anzahl Platzhalter sein
  * Nur eine SQL-Anweisung pro Prepared Statement erlaubt (kein ";")
  * Nicht verschachtelbar
  * Einmaliger Einsatz langsamer als Ã¤quiv. normale SQL-Anweisung auszufÃ¼hren
  * "Memory Leak" mÃ¶glich, falls Freigabe vergessen wird
  * Maximal "max_prepared_stmt_count" pro Server mÃ¶glich (Wert 0 --> keine)

Prepared Statement mit Platzhaltern (ergibt Statement-Handle <PrepStmt>):

  PREPARE <PrepStmt>      # Beim AusfÃ¼hren ausgefÃ¼llt, AnfÃ¼hrungszeichen nÃ¶tig,
    FROM <String>;        # "?" als Wert-Platzhalter, Textvariable erlaubt

Prepared Statement mit Variablenwerten fÃ¼llen und ausfÃ¼hren:

  EXCECUTE <PrepStmt>;    # Ohne Platzhalter
  EXCECUTE <PrepStmt>     # Falls Platzhalter "?" verwendet
    USING <Var1>, ...;    # MÃM-^\SSEN Variable sein, keiner Ausdr./Literale!

Prepared-Statement anzeigen gibt es nicht!

Prepared Statement lÃ¶schen (Speicher freigeben, "Memory Leak" falls vergessen):

  DEALLOCATE PREPARE <PrepStmt>;   # OK
  DROP PREPARE <PrepStmt>;         # OK
```

```
Beispiel (Platzhalter "?" im Statement NICHT in "..." oder '...' setzen,
auch wenn der reprÃ¤sentierte Wert dies eigentlich erfordern wÃ¼rde):

  # Beispiel 1
  DROP TABLE IF EXISTS buecher;
  CREATE TABLE buecher (
    titel CHAR(50),
    autor CHAR(30),
    isbn  INT
  );

  PREPARE neues_buch
    FROM "INSERT INTO buecher(titel, autor, isbn)
          VALUES (?, ?, ?)";

  SET @titel = "MySQL fÃ¼r AnfÃ¤nger";   # Variable setzen
  SET @autor = "Thomas Birnthaler";    # Variable setzen
  SET @isbn  = "3-987654-321-0";       # Variable setzen

  EXECUTE neues_buch USING @titel, @autor, @isbn;                # OK
  EXECUTE neues_buch USING "Carrie", "King", "9-876543-210-X";  # Falsch, nur
                                                                # Variable OK!
  SELECT * FROM buecher;

  DEALLOCATE PREPARE neues_buch;         # OK
  DROP PREPARE neues_buch;               # Ebenfalls OK

  # Beispiel 2
  PREPARE stmt1 FROM "SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse";
  SET @a = 3;
  SET @b = 4;
  EXECUTE stmt1 USING @a, @b;
  DEALLOCATE PREPARE stmt1;

  # Beispiel 3
  SET @s = "SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse";
  PREPARE stmt2 FROM @s;
  SET @a = 6;
  SET @b = 8;
  EXECUTE stmt2 USING @a, @b;
  DEALLOCATE PREPARE stmt2;

  # Beispiel 4
  DROP TABLE IF EXISTS t1;
  CREATE TABLE t1 (a INT NOT NULL);
  INSERT INTO t1 VALUES (4), (8), (11), (32), (80), (90), (100);
  SELECT * FROM t1;

  SET @table  = "t1";
  SET @limit  = 2;
  SET @offset = 3;
  SET @s = CONCAT("SELECT * FROM ", @table, " LIMIT ", @limit,
                                            " OFFSET ", @offset);
  SELECT @s as "Statement1";
  PREPARE stmt3 FROM @s;
  EXECUTE stmt3;
  DEALLOCATE PREPARE stmt3;

  /*!50700
  SET @s = CONCAT("SELECT * FROM ", @table, " LIMIT ? OFFSET ?");
  SELECT @s as "Statement2";
  PREPARE stmt4 FROM @s;
  EXECUTE stmt4 USING 1, 3;
  EXECUTE stmt4 USING "1", "3";
  DEALLOCATE PREPARE stmt4; */

  # Beispiel 5
  # 1x an den Server schicken
  PREPARE select_boot
    FROM "SELECT b.boot_id, bd.laenge
          FROM boot AS b JOIN boot_detail AS bd
          WHERE b.name = ? AND bd.typ = CONCAT("fest_", ?);

  # Nx ausfuehren mit unterschiedlichen Werten
  EXECUTE select_boot USING @name, @typ;


19) Stored Routines (Stored Procedures/Functions)
-------------------------------------------------

Eigenschaften:
```

```
  * Dienen der Kapselung von GeschÃ¤ftslogik in der Datenbank (nur 1x),
    viele verschiedene Anwendungen rufen sie nur auf (statt Nx programmiert)
  * Routine = Prozedur oder Funktion
    + Funktion hat RÃ¼ckgabewert
      --> In AusdrÃ¼cken oder per SELECT aufzurufen (gerne vergessen!)
    + Prozedur hat KEINEN RÃ¼ckgabewert (auÃM-^_er per INOUT/OUT-Parameter),
      --> Mit "CALL" aufzurufen (gerne vergessen!)
  * An eine Datenbank <Db> GEBUNDEN (werden mit dieser Datenbank gelÃ¶scht!),
    eigentliche Definition liegt in zentraler Datenbank "mysql"
  * Anweisungen in Funk./Proz. beziehen sich auf eine einzige Datenbank
    (auf beim Aufruf aktuell mit USE ausgewÃ¤hlte)
  * KÃ¶rper (Body) besteht aus einer/mehreren SQL-Anweisungen
    (mehrere in BEGIN...END einschlieÃM-^_en)
  * Erlaubt sind die meisten DDL-, DML- und DCL-Anweisungen
    (aber keine Transaktionssteuerung!), verboten sind:
    + LOCK/UNLOCK TABLES
    + ALTER VIEW
    + LOAD DATA/TABLE
    + PREPARE, EXECUTE, DEALLOCATE PREPARE   # nicht in Functions/Trigger
    + Alle in Prepared Statements nicht erlaubten Anweisungen
  * Rechte zum Erstellen/ÃM-^Dndern (CREATE ROUTINE, ALTER ROUTINE)
    und zum AusfÃ¼hren (EXECUTE) notwendig
  * Lokale Variablen mÃ¶glich
    + Definition:                  DECLARE <Var> <Typ> DEFAULT <Wert>;
    + Initialisierung:             SET <Var> := <Wert>;
    + Wert in Anweisung verwenden: ... <Var> ...
    + Parameter sind ebenfalls automatisch lokale Variablen
  * Aufruf:
      CALL <Proc>;          # In Default/Standard-Datenbank
      CALL <Db>.<Proc>      # In anderer Datenbank <Db>
      SELECT <Func>;        # Oder im Rahmen eines Ausdrucks
      SELECT <Db>.<Func>;   # Analog in anderer Datenbank <Db>
  * Eigene Funktionen wie eingebaute MySQL-Funktionen verwendbar
  * Rekursiver Aufruf in Funktionen/Prozeduren erlaubt (FakultÃ¤t)

BeschrÃ¤nkungen:
  * Anzahl und Typ der Parameter muss beim Aufruf eingehalten werden
  * Nur EIN Wert pro Variable Ã¼bergebbar (keine Arrays/Objekte/Strukturen)
  * Variablen NICHT fÃ¼r Datenbank/Tabellen/Spaltennamen nutzbar
    (aber fÃ¼r LIMIT/OFFSET-Werte!)
  * Macht "Prelocking" aller darin benÃ¶tigten Tabellen!
  * Replikation des Aufrufs oder der ÃM-^Dnderungen durch den Aufruf?
    + MY!5.0: Binary Logging, alle DETERMINISTIC oder log_bin_trust_function_creators
  * Bei jedem Aufruf geparst und optimiert (auch wenn DETERMINISTIC)

Hinweise:
  * SET vor Variablenzuweisung gerne vergessen
  * CALL vor Prozeduraufruf gerne vergessen
  * SELECT vor Funktionsaufruf notwendig
  * Das Abschlusszeichen von SQL-Anweisungen ";" (SQL-Delimiter) ist VOR der
    Definition einer Prozedur/Funktion temporÃ¤r auf "//" oder Ã¤hnliches setzen
    und DANACH wieder auf ";" zurÃ¼cksetzen, damit ";" in der Definition zur
    Trennung der SQL-Statements verwendbar ist (MY!).

      DELIMITER //
      ... # Prozedur/Funktions-Definition, die SQL-Anweisungen mit ";" enthÃ¤lt,
      ... # ohne die soforige AusfÃ¼hrung dieser SQL-Anweisungen auszulÃ¶sen
      ...
      DELIMITER ;

Vorteile/Nachteile:   !!!
  * S283

HINWEIS: Beispiele fÃ¼r alle Kontrollstrukturen folgen am Ende des Abschnitts


19a) Lokale Variablen
---------------------

Eigenschaften:
  * Nur innerhalb Prozeduren/Funktionen/Event/Trigger erlaubt
  * Nur am Anfang deklarierbar (direkt nach BEGIN)
  * Jeder SQL-Datentyp in Deklaration verwendbar (INT, FLOAT, CHAR(20), ...)
  * Initialisierbar per DEFAULT-Anweisung (sonst NULL als Startwert)
  * Kein "@" oder "@@" vor Variablenname (--> Session- oder Server-Variable)
  * Verschwinden beim Verlassen von Stored Procedure/Event

  DECLARE anz  INT;                       # Mit NULL initialisiert
  DECLARE max  FLOAT    DEFAULT 0.0;      # Mit 0.0 initialisiert
  DECLARE sum  FLOAT    DEFAULT 0.0;      # Mit 0.0 initialisiert
  DECLARE str  CHAR(30) DEFAULT "";       # Mit "" initialisiert
  DECLARE str2 CHAR(30) DEFAULT "Summe";  # Mit "Summe" initialisiert
```

```
19b) Wertzuweisung an Variable
------------------------------

Direkte Wertangabe oder Ergebnis einer Berechnung zuweisen
(Berechnung geht in DECLARE wahrscheinlich nicht):

  SET max = 9999;                     # Mit Vergleich "=" verwechselbar
  SET str = "Hallo";                  # Mit Vergleich "=" verwechselbar
  SET str = CONCAT(str, " welt!");    # Mit Vergleich "=" verwechselbar
  SET anz = anz + 1;                  # Mit Vergleich "=" verwechselbar
  SET max := 9999;                    # NICHT mit Vergleich "=" verwechselbar
  SET str := "Hallo";                 # NICHT mit Vergleich "=" verwechselbar
  SET str := CONCAT(str, " welt!");   # NICHT mit Vergleich "=" verwechselbar
  SET anz := anz + 1;                 # NICHT mit Vergleich "=" verwechselbar

HINWEISE:
  + Zugewiesener Wert muss zu Datentyp passen
  + In Ausdruck Aufruf von MySQL-Funktionen und eigener Funktionen erlaubt

SELECT-Ergebnis zuweisen:

  SELECT COUNT(*)   FROM pers    INTO anz;        # @anz und @@anz auch mÃ¶glich
  SELECT MAX(preis) FROM artikel INTO max;        #
  SELECT SUM(anz)   FROM artikel INTO sum;        #
  SELECT SUM(anz)   INTO sum     FROM artikel;    # Auch mÃ¶glich
  SELECT MIN(preis), MAX(preis), SUM(anz)         # Mehrere Variablen auf einmal
    FROM artikel INTO min, max, sum;              #
  SELECT SUM(anz)   INTO sum     FROM artikel;    # Auch mÃ¶glich
  SELECT func(1, 3.14, "text")   INTO str;        # Funktionsergebnis

HINWEISE:
  * GENAU EIN Datensatz muss selektiert werden (Trick: LIMIT 1 anhÃ¤ngen),
    (mehrere SÃ¤tze mÃ¼ssen per CURSOR-Zugriff in einer Schleife behandelt werden)
  * Anzahl selektierter Spalten muss gleich Anzahl Variablen sein.


19c) Ausgabe von Variablen oder Daten
-------------------------------------

  SELECT @str AS "Titel";                                 # Einzelwert
  SELECT "Fester Text: ", @wert, " Euro";                 # Werteliste
  SELECT @sum / @anz * 100;                               # Ausdruck
  SELECT CONCAT("Fester Text: ", 17.00, @str) AS Text;    # Verketteter Text
  SELECT func(1, 3.14, @str2) AS "Funktionswert"          # Funktionsergebnis


19d) Kontrollstrukturen (Block/Compound Statement)
--------------------------------------------------

Eigenschaften:
  * Ein BEGIN...END-Block fasst mehrere SQL-Anweisungen zusammen
    (statt einzelner SQL-Anweisung verwendbar)
  * Benennung per Marke <Label> mÃ¶glich
    (fÃ¼r ITERATE, LEAVE, END, END LOOP, END WHILE, END REPEAT)
  * Zu Beginn lokale Variablen per DECLARE-Anweisungen einrichtbar
    + Definiert Geltungsbereich
    + Ausserhalb nicht mehr vorhanden
  * Optionale Marke <Label> vor BEGIN kann max. 16 Zeichen lang sein
    + Falls verwendet, am Kontrollstruktur-Ende zu wiederholen
    + Als Zielname von benannten ITERATE- und LEAVE-Anweisungen

  [<Label>:] BEGIN   # Evtl. Marke setzen
    ...
  END [<Label>];     # Marke wiederholen falls verwendet!


19e) Kontrollstrukturen (Verzweigungen)
---------------------------------------

IF-Verzweigung: Anweisungen nach THEN der von oben nach unten ersten wahren
Bedingung <CondN> werden ausgefÃ¼hrt; oder falls keine <CondN> TRUE wird, die
Anweisungen nach ELSE; oder falls ELSE nicht vorhanden: gar keine Anweisung):

  IF <Cond1> THEN       #
    <Stmt>;             # Anweisung
    ...                 # ...
  ELSEIF <Cond2> THEN   # Beliebig oft (auch weglassbar)
    <Stmt>;             # Anweisung
    ...                 # ...
  ELSE                  # Weglassbar
```

```
    <Stmt>;             # Anweisung
    ...                 # ...
  END IF;               # ";" nicht vergessen!

CASE-Verzweigung: Anweisungen nach THEN der von oben nach unten ersten wahren
Bedingung <CondN> werden ausgefÃ¼hrt; oder falls keine <CondN> TRUE wird, die
Anweisungen nach ELSE; oder falls ELSE nicht vorhanden: gar keine Anweisung):

  CASE                              # Variante A: Bedingungen prÃ¼fen
    WHEN <Cond1> THEN <Stmt>; ...   # Bedingung <Cond1> prÃ¼fen
    WHEN <Cond2> THEN <Stmt>; ...   # Bedingung <Cond2> prÃ¼fen
    ...                             #
    ELSE <Stmt>; ...                # Besser immer angeben!
  END CASE;                         # ";" nicht vergessen!

CASE-Fallunterscheidung: <Expr> wird ausgewertet und das Ergebnis von oben nach
unten mit den Werten <ValN> verglichen und Anweisungen nach zugehÃ¶rigem THEN
ausgefÃ¼hrt; oder falls kein WHEN-Fall zutrifft, die Anweisungen nach ELSE; oder
falls ELSE nicht vorhanden: gar keine Anweisung):

  CASE <Expr>                       # Variante B: Ergebnis mit Wert vergleichen
    WHEN <Val1> THEN <Stmt>; ...    # Ergebnis mit Wert <Val1> vergleichen
    WHEN <Val2> THEN <Stmt>; ...    # ...
    ...                             # ...
    ELSE <Stmt>; ...                # Besser immer angeben!
  END CASE;                         # ";" nicht vergessen!

ACHTUNG: Tritt bei CASE ohne ELSE-Zweig ein Fall auf, der nicht von einem WHEN
abgefragt wird, wird eine Fehlermeldung ausgegeben (seltsames Verhalten!)
--> Immer ELSE-Zweig angeben!


19f) Kontrollstrukturen (Schleifen)
-----------------------------------

Hinweise (Beispiele folgen weiter unten):
  * Marke <Label> kann fehlen
    (falls verwendet, ist sie am Ende der Konstrollstruktur zu wiederholen!)
  * LEAVE bricht Schleife vorzeitig ab (es geht danach weiter)
  * ITERATE springt zum Schleifenanfang (nÃ¤chster Durchlauf)
  * Bei verschachtelten Schleifen ist Marke <Label> notwendig,
    um eine der Ã¤uÃM-^_eren Schleifen neu zu starten oder abzubrechen
  * Es gibt kein GOTO <Label> (wurde kurzzeitig getestet, aber wieder entfernt)

Endlosschleife:

  [<Label>:] LOOP         # Evtl. Marke setzen
    <Stmt>;               # Anweisung in Schleife
    ...                   # ...
    ITERATE [<Label>];    # Schleife neu starten
    LEAVE [<Label>];      # Schleife verlassen
    ...                   # ...
  END LOOP [<Label>];     # Marke wiederholen falls verwendet

Nichtabweisende Schleife (Inneres mind. 1x ausgefÃ¼hrt!):

  [<Label>:] REPEAT                     # Evtl. Marke setzen
    <Stmt>;                             # Anweisung in Schleife
    ...                                 # ...
    ITERATE [<Label>];                  # Schleife neu starten
    LEAVE [<Label>];                    # Schleife verlassen
    ...                                 # ...
  UNTIL <Cond> END REPEAT [<Label>];    # Marke wiederholen falls verwendet

Abweisende Schleife (Inneres evtl. gar nicht ausgefÃ¼hrt!):

  [<Label>:] WHILE <Cond> DO    # Evtl. Marke setzen
    <Stmt>;                     # Anweisung
    ...                         # ...
    ITERATE [<Label>];          # Schleife neu starten
    LEAVE [<Label>];            # Schleife verlassen
    ...                         # ...
  END WHILE [<Label>];          # Marke wiederholen falls verwendet


19g) Prozeduren
---------------

Hinweise:
  * Name, Parameter (leere Liste erlaubt) und ProzedurkÃ¶rper sind anzugeben
    + Parameter sind in der Form [<InOut>] <Name> <Typ> anzugeben,
      (z.B. INOUT wert TINYINT, IN ist Standard)
```

```
    + <Typ> ist ein beliebiger SQL-Datentyp (OHNE LÃ¤ngenangabe!)
  * Eingabeparameter verhalten sich wie initialisierte lokale Variablen:
      DECLARE <Var> <Typ> DEFAULT <Wert_von_aussen>;
  * Kann per SELECT-Anweisung oder per OUT/INOUT-Parameter Ergebnis zurÃ¼ckgeben
  * Prozedur ohne Parameter darf ohne "()" aufgerufen werden

Parameter-Eigenschaft <InOut>:

  +-------+--------------+-----------------------------------------------------+
  | Form  | Typ          | Bemerkung                                           |
  +-------+--------------+-----------------------------------------------------+
  | IN    | Eingabe (STD)| Variable/Ausdruck/Literal ("abc", 123.45, abs(-1))  |
  | OUT   | Ausgabe      | Muss Variable sein (z.B. @var)                      |
  | INOUT | Ein/Ausgabe  | Muss Variable sein (z.B. @var)                      |
  +-------+--------------+-----------------------------------------------------+

Definition einer Prozedur:

  DELIMITER //
  CREATE PROCEDURE <Proc> ([<InOut>] <Param> <Typ>, ...)
    [[NOT] DETERMINISTIC]
    {CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA}
    [SQL SECURITY {DEFINER | INVOKER}]
    [COMMENT <String>]
  BEGIN
    <Stmt>;
    ...
  END //
  DELIMITER ;

Aufruf einer Prozedur:
  * Anzahl Ã¼bergebener Argumente muss zu Definition passen
  * FÃ¼r OUT- und INOUT-Parameter muss eine Variable Ã¼bergeben werden
  * FÃ¼r IN-Parameter darf auch Wert/Konstante Ã¼bergeben werden

Beispiel:

  # Definition
  DROP PROCEDURE IF EXISTS eins;
  DELIMITER //
  CREATE PROCEDURE eins(IN anz INT, IN preis FLOAT, INOUT text CHAR(20))
  BEGIN
    SELECT anz AS "1.", preis AS "2.", text AS "3.";
    SET text = CONCAT(ROUND(anz * preis, 2), " Euro");
  END //
  DELIMITER ;

  # Aufruf
  SET @str = "Text";              # Variable belegen
  CALL eins(1, 3.14, @str);       # Variable Ã¼bergeben
  SELECT @str;                    # Variable ausgeben

  # Definition
  DROP PROCEDURE IF EXISTS zwei;
  DELIMITER //
  CREATE PROCEDURE zwei(IN name CHAR(20))
  BEGIN
    SELECT user, host, password FROM mysql.user WHERE user = name LIMIT 1;
  END //
  DELIMITER ;

  # Aufruf
  CALL zwei("root");                                          # Variable Ã¼bergeben
  CALL zwei("tom");                                           # Variable Ã¼bergeben
  #SET @user, @host, @password = CALL zwei("tom");            # Geht leider nicht!
  #SELECT CALL zwei("tom") INTO @user, @host, @password;      # Geht leider nicht!

Informationen Ã¼ber alle Prozeduren auflisten
(Datenbank, Name, Typ, Ersteller, Erstellungs- und Ã„nderungsdatum):

  SHOW PROCEDURE STATUS;
  SHOW PROCEDURE STATUS LIKE "<Muster>";

Prozedur-Definition anzeigen:

  SHOW CREATE PROCEDURE <Proc>;         # OK
  SHOW PROCEDURE CODE <Proc>;           # Fehler!

Charakteristik einer gespeicherten Prozedur Ã¤ndern (nicht Name oder Inhalt!):

  ALTER PROCEDURE <Proc>
    {CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA} |
```

```
    SQL SECURITY {DEFINER | INVOKER} |
    COMMENT <String>

LÃ¶schen einer Prozedur:

  DROP PROCEDURE <Proc>;
  DROP PROCEDURE IF EXISTS <Proc>;   # Kein Fehler falls nicht existent

Beispiele fÃ¼r Prozeduren:

  DELIMITER //   # Nicht vergessen!

  #--------------------------------------------------------------------------
  # Prozedur 1
  #--------------------------------------------------------------------------
  DROP PROCEDURE IF EXISTS bench //     # ACHTUNG: ; fÃ¼hrt zu Fehler

  CREATE PROCEDURE bench(IN anz INT)
  BEGIN
    SELECT NOW();
    WHILE anz > 0 DO
      SET anz = anz - 1;
    END WHILE;
    SELECT NOW();
  END //

  CALL bench(10) //
  CALL bench(1000) //
  CALL bench(100000) //
  CALL bench(10000000) //

  #--------------------------------------------------------------------------
  # Prozedur 2
  #--------------------------------------------------------------------------
  DROP PROCEDURE IF EXISTS conv_xml_special //

  CREATE PROCEDURE conv_xml_special(INOUT str CHAR(255))
  BEGIN
    SET str = REPLACE(str, "&", "&amp;");
    SET str = REPLACE(str, "<", "&lt;");
    SET str = REPLACE(str, ">", "&gt;");
  END //

  SET @str = "a && (b <> c)" //
  SELECT @str //
  CALL conv_xml_special(@str) //
  SELECT @str //

  #--------------------------------------------------------------------------
  # Prozedur 3
  #--------------------------------------------------------------------------
  DROP TABLE IF EXISTS pers //

  CREATE TABLE pers (                  # Variante A (Engine = InnoDB = STD)
    nr      INT,
    vorname VARCHAR(30),
    name    VARCHAR(30)
  ) //

  INSERT INTO pers (nr, vorname, name)
    VALUES (1, "Thomas",  "Birnthaler"),
           (2, "Markus",  "Mueller"),
           (8, "Andrea",  "Bayer"),
           (9, "Richard", "Seiler"),
           (7, "Heinz",   "Bayer") //

  DROP PROCEDURE IF EXIST select_limited //

  CREATE PROCEDURE select_limited(IN grenze INT)
  BEGIN
    DECLARE anz INT DEFAULT 0;
                                       # Leerzeile zur Ã¼-^\bersicht
    SELECT COUNT(*) INTO anz FROM pers;
    IF anz <= grenze THEN
      SELECT * FROM pers;
    ELSE
      SELECT * FROM pers LIMIT grenze;   # Erlaubt als Parameter!
    END IF;
  END //

  CALL select_limited(100) //
  CALL select_limited(6) //
```

```
CALL select_limited(5) //
CALL select_limited(4) //

#-----------------------------------------------------------------------------
# Prozedur 4
#-----------------------------------------------------------------------------
DROP TABLE IF EXISTS artikel //

CREATE TABLE artikel (
  nr     INT,
  bez    CHAR(30),
  anz    INT,
  preis  FLOAT(6,2)
) //

INSERT INTO artikel (nr, bez, anz, preis)
  VALUES (1, "Artikel A", 100,   7.99),
         (2, "Artikel B",  10,  19.50),
         (3, "Artikel C",   5,  79.80),
         (4, "Artikel D",   1, 123.00),
         (5, "Artikel E",   0,   2.49) //

DROP PROCEDURE IF EXISTS max_preis //

CREATE PROCEDURE max_preis(IN grenze FLOAT)
BEGIN
  DECLARE max  FLOAT    DEFAULT 0.0;
                                        # Leerzeile zur ÃM-^\bersicht
  SELECT MAX(preis) INTO max FROM artikel;
  IF max > grenze THEN
    SELECT "Maximaler Preis: ", max;
  END IF;
END //

CALL max_preis( 20) //
CALL max_preis(100) //
CALL max_preis(200) //

#-----------------------------------------------------------------------------
# Prozedur 5
#-----------------------------------------------------------------------------
DROP TABLE IF EXISTS artikel //

CREATE TABLE artikel (
  nr     INT,
  bez    CHAR(30),
  anz    INT,
  preis  FLOAT(6,2)
) //

INSERT INTO artikel (nr, bez, anz, preis)
  VALUES (1, "Artikel A", 100,   7.99),
         (2, "Artikel B",  10,  19.50),
         (3, "Artikel C",   5,  79.80),
         (4, "Artikel D",   1, 123.00),
         (5, "Artikel E",   0,   2.49) //

DROP PROCEDURE IF EXISTS summe_artikel //

CREATE PROCEDURE summe_artikel(IN artnr INT)
BEGIN
  DECLARE artanz INT DEFAULT 0;
                                          # Leerzeile zur ÃM-^\bersicht
  SELECT SUM(anz) INTO artanz FROM artikel
    WHERE nr = artnr
      GROUP BY nr;

  # SELECT "Artanz: ", artanz;    # DEBUG-Ausgabe

  IF artanz > 1 THEN
    SELECT DISTINCT nr, anz
      FROM artikel
        WHERE nr = artnr;
  END IF;
END //

CALL summe_artikel(1) //
CALL summe_artikel(2) //
CALL summe_artikel(3) //
CALL summe_artikel(4) //
CALL summe_artikel(5) //
CALL summe_artikel(6) //
```

```
  #------------------------------------------------------------------------
  # Prozedur 6
  #------------------------------------------------------------------------
  DROP TABLE IF EXISTS artikel //

  CREATE TABLE artikel (
    nr     INT,
    bez    CHAR(30),
    anz    INT,
    preis  FLOAT(6,2)
  ) //

  INSERT INTO artikel (nr, bez, anz, preis)
    VALUES (1, "Artikel A", 100,   7.99),
           (2, "Artikel B",  10,  19.50),
           (3, "Artikel C",   5,  79.80),
           (4, "Artikel D",   1, 123.00),
           (5, "Artikel E",   0,   2.49) //

  DROP PROCEDURE avg_sum_max_min //

  CREATE PROCEDURE avg_sum_max_min(IN typ CHAR)
  BEGIN
    # CASE-Variante A
    CASE
      WHEN typ = "G" THEN SELECT AVG(preis) AS "Durchschnitt" FROM artikel;
      WHEN typ = "S" THEN SELECT SUM(preis) AS "Summe"        FROM artikel;
      WHEN typ = "A" THEN SELECT MAX(preis) AS "Maximum"      FROM artikel;
      WHEN typ = "I" THEN SELECT MIN(preis) AS "Minimum"      FROM artikel;
      ELSE SELECT "Geben Sie G, S, A oder I ein";   # Muss da sein!
    END CASE;

    # CASE-Variante B
    CASE typ
      WHEN "G" THEN SELECT AVG(preis) AS "Durchschnitt" FROM artikel;
      WHEN "S" THEN SELECT SUM(preis) AS "Summe"        FROM artikel;
      WHEN "A" THEN SELECT MAX(preis) AS "Maximum"      FROM artikel;
      WHEN "I" THEN SELECT MIN(preis) AS "Minimum"      FROM artikel;
      ELSE SELECT "Geben Sie G, S, A oder I ein";   # Muss da sein!
    END CASE;
  END //

  CALL avg_sum_max_min("G") //
  CALL avg_sum_max_min("S") //
  CALL avg_sum_max_min("A") //
  CALL avg_sum_max_min("I") //
  CALL avg_sum_max_min("X") //

  DELIMITER ;     # Nicht vergessen!


19h) Funktionen
---------------

Hinweise:
  * Name, Parameter (leere Liste erlaubt), RÃ¼ckgabetyp, FunktionskÃ¶rper nÃ¶tig
    + Parameter sind in der Form <Name> <Typ> anzugeben (z.B. Wert FLOAT)
    + <Typ> ist ein beliebiger SQL-Datentyp (OHNE LÃ¤ngenangabe!)
    + MUSS mind. 1x per RETURN <Expr> Wert vom pass. Typ zurÃ¼ckgeben (oder NULL)
  * In einer Funktion sind KEINE Tabellenzugriffe erlaubt
  * Ebenso dÃ¼rfen darin KEINE Prozeduren aufgerufen werden
  * Gibt per RÃ¼ckgabewert RETURN <Wert> Ergebnis zurÃ¼ckgeben
  * Funktion ohne Parameter darf ohne "()" aufgerufen werden

Definition einer Funktion:

  DELIMITER //
  CREATE FUNCTION <Func> (<Param> <Typ>, ...)
    RETURNS <Typ>
    [[NOT] DETERMINISTIC]
    {CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA}
    [SQL SECURITY {DEFINER | INVOKER}]
    [COMMENT <String>]
  BEGIN
    <Stmt>;
    ...
    RETURN <Expr>;   # WerterÃ¼ckgabe beliebig oft (mind. 1x!)
    ...
  END //
  DELIMITER ;
```

```
Aufruf einer Funktion:

  SELECT func(1, 3.14, "text");                # A) Ergebnis als Datensatz
  SET @var = SELECT func(1, 3.14, "text");     # B) Ergebnis in Variable speich.
  SELECT func(1, 3.14, "text") INTO @var;      # C) Ergebnis in Variable speich.
  SELECT 3.14 * func(1);                       # D) Als Teil eines Ausdrucks

Informationen Ã¼ber alle Funktionen auflisten
(Datenbank, Name, Typ, Ersteller, Erstellungs- und Ã„-^Dnderungsdatum):

  SHOW FUNCTION STATUS;
  SHOW FUNCTION STATUS LIKE "<Muster>";

Funktions-Definition anzeigen:

  SHOW CREATE FUNCTION <Func>;          # OK
  SHOW FUNCTION CODE <Func>;            # Fehler!

Charakteristik einer gespeicherten Funktion Ã¤ndern
(nicht Name, Parameter oder Inhalt!):

  ALTER FUNCTION <Func>
    [[NOT] DETERMINISTIC]
    {CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA} |
    SQL SECURITY {DEFINER | INVOKER} |
    COMMENT <String>;

LÃ¶schen einer Funktion:

  DROP FUNCTION <Func>;
  DROP FUNCTION IF EXISTS <Func>;   # Kein Fehler falls nicht existent

Beispiele fÃ¼r Funktionen:

  DELIMITER //

  #-------------------------------------------------------------------------
  # Funktion 1
  #-------------------------------------------------------------------------
  DROP FUNCTION IF EXISTS brutto_netto //

  CREATE FUNCTION brutto_netto(typ CHAR, wert1 FLOAT, wert2 FLOAT)
    RETURNS FLOAT
    NO SQL
  BEGIN
    CASE
      WHEN typ = "G" THEN RETURN wert1 * 100 / wert2;     # G=P*100/S
      WHEN typ = "S" THEN RETURN wert1 * 100 / wert2;     # S=P*100/G
      WHEN typ = "P" THEN RETURN wert1 * wert2 / 100;     # P=G*S/100
      ELSE RETURN 0.0;
    END CASE;
  END //

  SELECT brutto_netto("G", 1234, 100) //    # G)rundwert
  SELECT brutto_netto("S", 1234, 100) //    # S)atz
  SELECT brutto_netto("P", 1234, 1000) //   # P)rozentwert
  SELECT brutto_netto("X", 1234, 100) //

  DELIMITER ;

  #-------------------------------------------------------------------------
  # Funktion 2
  #-------------------------------------------------------------------------
  DROP FUNCTION IF EXISTS sumup_to_n //

  CREATE FUNCTION sumup_to_n(n INT)
    RETURNS INT
    NO SQL
  BEGIN
    DECLARE i   INT DEFAULT 1;
    DECLARE sum INT DEFAULT 0;
                                          # Leerzeile zur Ã„-^\bersicht
    IF n = 0 THEN
      RETURN 0;
    ELSE
      WHILE i <= n DO
        SET sum = sum + i;
        SET i = i + 1;
      END WHILE;
    END IF;
    RETURN sum;
  END //
```

```
SELECT sumup_to_n(0) //
SELECT sumup_to_n(100) //
SELECT sumup_to_n(1000) //
SELECT sumup_to_n(10000) //

#-------------------------------------------------------------------------
# Funktion 3
#-------------------------------------------------------------------------
DROP FUNCTION IF EXISTS area_or_volume //

CREATE FUNCTION area_or_volume(typ CHAR, radius FLOAT)
  RETURNS FLOAT
  NO SQL
BEGIN
  CASE
    WHEN typ = "A" THEN RETURN PI() * POW(radius, 2);           # A=pi*r^2
    WHEN typ = "V" THEN RETURN 4 * PI() / 3 * POW(radius, 3);   # V=4/3*pi*r^3
    ELSE RETURN 0.0;
  END CASE;
END //

SELECT area_or_volume("A", 100) //
SELECT area_or_volume("V", 100) //
SELECT area_or_volume("X", 100) //

#-------------------------------------------------------------------------
# Funktion 4
#-------------------------------------------------------------------------
DROP FUNCTION IF EXISTS roman //

CREATE FUNCTION roman(n INT)
  RETURNS CHAR(30)
  NO SQL
BEGIN
  DECLARE erg CHAR(30) DEFAULT "";
                                        # Leerzeile zur ÃM-^\bersicht
  WHILE n >= 1000 DO
    SET erg = CONCAT(erg, "M");
    SET n = n - 1000;
  END WHILE;
  # SELECT "ONE: ", erg, n;   # Nicht erlaubt!
  CASE
    WHEN n >= 900 THEN SET erg = CONCAT(erg, "CM"); SET n = n - 900;
    WHEN n >= 500 THEN SET erg = CONCAT(erg, "D");  SET n = n - 500;
    WHEN n >= 400 THEN SET erg = CONCAT(erg, "CD"); SET n = n - 400;
    ELSE SET erg = erg;   # Sonst Fehlermeldung fÃ¼r 300-000
  END CASE;

  WHILE n >= 100 DO
    SET erg = CONCAT(erg, "C");
    SET n = n - 100;
  END WHILE;
  CASE
    WHEN n >= 90 THEN SET erg = CONCAT(erg, "XC"); SET n = n - 90;
    WHEN n >= 50 THEN SET erg = CONCAT(erg, "L");  SET n = n - 50;
    WHEN n >= 40 THEN SET erg = CONCAT(erg, "XL"); SET n = n - 40;
    ELSE SET erg = erg;   # Sonst Fehlermeldung fÃ¼r 30-00
  END CASE;

  WHILE n >= 10 DO
    SET erg = CONCAT(erg, "X");
    SET n = n - 10;
  END WHILE;
  CASE
    WHEN n >= 9 THEN SET erg = CONCAT(erg, "IX"); SET n = n - 9;
    WHEN n >= 5 THEN SET erg = CONCAT(erg, "V");  SET n = n - 5;
    WHEN n >= 4 THEN SET erg = CONCAT(erg, "IV"); SET n = n - 4;
    ELSE SET erg = erg;   # Sonst Fehlermeldung fÃ¼r 3-0
  END CASE;

  WHILE n >= 1 DO
    SET erg = CONCAT(erg, "I");
    SET n = n - 1;
  END WHILE;

  RETURN erg;
END //

SELECT roman(0) //
SELECT roman(1) //
SELECT roman(4) //
```

```
SELECT roman(5) //
SELECT roman(6) //
SELECT roman(9) //
SELECT roman(10) //
SELECT roman(11) //
SELECT roman(49) //
SELECT roman(50) //
SELECT roman(51) //
SELECT roman(99) //
SELECT roman(100) //
SELECT roman(101) //
SELECT roman(222) //
SELECT roman(333) //
SELECT roman(444) //
SELECT roman(499) //
SELECT roman(500) //
SELECT roman(501) //
SELECT roman(666) //
SELECT roman(999) //
SELECT roman(1000) //
SELECT roman(1001) //
SELECT roman(1962) //
SELECT roman(1999) //
SELECT roman(2000) //
SELECT roman(2007) //
SELECT roman(4567) //
SELECT roman(9999) //

#-----------------------------------------------------------------------------
# Funktion 5 (iterative Variante)
#-----------------------------------------------------------------------------
DROP FUNCTION IF EXISTS ifakultaet //

CREATE FUNCTION ifakultaet(n INT)
  RETURNS DECIMAL(65)
  NO SQL
BEGIN
  DECLARE erg DECIMAL(65) DEFAULT 1;
                                          # Leerzeile zur ÃM-^\bersicht
  IF n < 0 OR n > 50 THEN
    SET erg = NULL;
  ELSE
    WHILE n > 0 DO
      SET erg = erg * n;
      SET n = n - 1;
    END WHILE;
  END IF;
  RETURN erg;
END //

SELECT ifakultaet(-10) AS "FakultÃ¤t von -10" // //
SELECT ifakultaet(0)   AS "FakultÃ¤t von 0" //
SELECT ifakultaet(1)   AS "FakultÃ¤t von 1" //
SELECT ifakultaet(5)   AS "FakultÃ¤t von 5" //
SELECT ifakultaet(10)  AS "FakultÃ¤t von 10" //
SELECT ifakultaet(50)  AS "FakultÃ¤t von 50" //
SELECT ifakultaet(100) AS "FakultÃ¤t von 100" //

#-----------------------------------------------------------------------------
# Funktion 6 (rekursive Variante)
# ERROR 1424 (HY000): Recursive stored functions and triggers are not allowed
#-----------------------------------------------------------------------------
DROP FUNCTION IF EXISTS rfakultaet //

CREATE FUNCTION rfakultaet(n INT)
  RETURNS DECIMAL(65)
  NO SQL
BEGIN
  DECLARE erg DECIMAL(65) DEFAULT 1;
                                          # Leerzeile zur ÃM-^\bersicht
  IF n < 0 OR n > 50 THEN
    RETURN NULL;
  ELSEIF n <= 1 THEN
    RETURN n;
  ELSE
    RETURN (n * rfakultaet(n - 1));
  END IF;
END //

SELECT rfakultaet(-10) AS "FakultÃ¤t von -10" // //
SELECT rfakultaet(0)   AS "FakultÃ¤t von 0" //
SELECT rfakultaet(1)   AS "FakultÃ¤t von 1" //
```

```
  SELECT rfakultaet(5)   AS "FakultÃ¤t von 5" //
  SELECT rfakultaet(10)  AS "FakultÃ¤t von 10" //
  SELECT rfakultaet(50)  AS "FakultÃ¤t von 50" //
  SELECT rfakultaet(100) AS "FakultÃ¤t von 100" //

  #-------------------------------------------------------------------------
  # Funktion 7 (Ersatz fÃ¼r DATEDIFF(CURDATE(), geburts_datum))
  # Alter zu einem Geburtsdatum bezogen auf heute = NOW() ausrechnen
  # Einfache Version: Nur die Jahre voneinander abziehen (Hilfsfkt. YEAR(d))
  # Komplexe Version: Auch Monate und Tage voneinander abziehen
  #                   (Hilfsfkt. MONTH(d), DAY()) und Kommazahl zurÃ¼ckliefern
  # ACHTUNG: SQL-SchlÃ¼sselwort (z.B. "alter" nicht als Fktname verwendbar!)
  #-------------------------------------------------------------------------
  DROP FUNCTION IF EXISTS age //

  CREATE FUNCTION age(geburts_datum DATE)
    RETURNS FLOAT
    NO SQL
  BEGIN
    DECLARE year_diff  INT DEFAULT 0;
    DECLARE month_diff INT DEFAULT 0;
    DECLARE day_diff   INT DEFAULT 0;
                                          # Leerzeile zur Ã¼bersicht
    # Differenzen der 3 Komponenten ermitteln
    SET year_diff  = YEAR(CURDATE())  - YEAR(geburts_datum);
    SET month_diff = MONTH(CURDATE()) - MONTH(geburts_datum);
    SET day_diff   = DAY(CURDATE())   - DAY(geburts_datum);

    # Negative Differenz erfordert "Ã¼bertrag" von nÃ¤chstgrÃ¶Ã¼_erer Komponente
    IF day_diff < 0 THEN
      SET day_diff   = day_diff + 30;
      SET month_diff = month_diff - 1;
    END IF;

    IF month_diff < 0 THEN
      SET month_diff = month_diff + 12;
      SET year_diff  = year_diff - 1;
    END IF;

    # Debugausgabe in Funktionen nicht erlaubt!
    # SET diff = "Differenz: ", DATEDIFF(CURDATE(), geburts_datum);

    # Ganze Jahre + "Bruchteil" eines Jahres summieren
    RETURN year_diff + (month_diff / 12) + (day_diff / 360);
  END //

  SELECT age("1962.07.01") //
  SELECT age("2007.05.09") //
  SELECT age("2006.05.09") //
  SELECT age("2005.05.09") //
  SELECT age("2004.04.01") //
  SELECT age("2004.05.01") //
  SELECT age("2004.05.09") //
  SELECT age("2004.05.17") //
  SELECT age("2004.06.17") //

  DELIMITER ;


20) Trigger
-----------

LÃ¶st automatische Reaktion beim EinfÃ¼gen, Ã¤ndern, LÃ¶schen von DatensÃ¤tzen in
Tabellen aus. Kann VOR und/oder NACH der auslÃ¶senden SQL-Anweisung erfolgen.

Zweck:
  * PrÃ¼fung/Korrektur/Berechnung von Spaltenwerten mit BEFORE-Trigger
    (z.B. "Matchcode" aus PLZ, Vorname, Nachname zur Dublettenerkennung erstellen)
  * Andere Tabellen updaten (History, Denormalisierung) mit AFTER-Trigger
  * Constraints oder Business Rules realisieren

Eigenschaften:
  * EINER Tabelle zugeordnet (NICHT View, NICHT temp. Tabelle!)
    (Tabelle mit passenden Spalten muss existieren)
  * VOR/NACH SQL-Anweisung auf einer Tabelle autom. ausgefÃ¼hrt
    (BEFORE/AFTER: INSERT, UPDATE, DELETE, REPLACE = INSERT/UPDATE)
    + Pro Datensatz:     FOR EACH ROW       (Standard in MY!)
    + Pro SQL-Anweisung: FOR EACH STATEMENT (nicht vorhanden in MY!)
  * Gespeichert pro Tabelle <Tbl> in Datei "<Tbl>.TRG" (+ "<Tbl>.TRN")
  * Name eines Triggers muss pro Datenbank (Schema) eindeutig sein
  * TIMESTAMP- und AUTO_INCREMENT-Spalten haben eine Art "Default-Trigger"
```

```
BeschrÃ¤nkungen:
  * Ein Trigger LOCKT wÃ¤hrend AusfÃ¼hrung ALLE beteiligten Tabellen vollstÃ¤ndig!
    --> So schnell wie mÃ¶glich durchlaufen!
  * NUR EIN Trigger pro Kombination <Time> + <Event> mÃ¶glich
    + Mehrere Trigger zu unterschiedlichem <Time> + <Event> pro Tabelle mÃ¶glich
    + In MY!5.7.x auch mehrere Trigger pro Kombination <Time> + <Event> erlaubt
  * Trigger haben gleiche EinschrÃ¤nkungen wie Stored Procedures
    (als <Stmt> zw. BEGIN...END alle in Stored Procedures erlaubten mÃ¶glich)
    + Keine Prepared Statements darin erlaubt
    + Keine Transaktionen oder Locks kÃ¶nnen darin beginnen/enden
  * Im Trigger Zugriff auf andere Tabellen erlaubt
    (schreiben auf Tabelle fÃ¼r die Trigger ausgelÃ¶st wurde nicht erlaubt)
  * Im Trigger Stored Procedures per CALL und Funktionen aufrufbar
    + DÃ¼rfen keine Daten per SELECT produzieren
    + Datenaustausch nur per OUT/INOUT- und RÃ¼ckgabe-Parameter
  * Derzeit nicht durch kaskadierte Foreign Key Aktionen ausgelÃ¶st (kommt noch!)
  * Per LEAVE vorzeitiges Trigger-Ende mÃ¶glich (nicht mit RETURN)
  * Nur in BEFORE ist NEW.<Col> schreibbar
  * In BEFORE hat NEW.<Col> einer AUTO_INCREMENT-Spalte den Wert "0"
  * FÃ¼r Compound-Statements als Trigger-Code ist Delimiter ";" zu Ã¤ndern
    (analog Stored Procedures)
  * Row-based Replication:       Trigger auf Slave werden NICHT ausgelÃ¶st
    Statement-based Replication: Trigger auf Slave werden ausgelÃ¶st
  * TRICK: ROW_COUNT() ist 1 auÃM-^_er fÃ¼r den ersten Datensatz
    --> Trigger nur 1x pro Statement ausfÃ¼hren --> FOR EACH STATEMENT Emulation

Fehlerbehandlung:
  * Scheitert BEFORE-Trigger, wird eigentliche SQL-Anweisung NICHT durchgefÃ¼hrt
  * BEFORE-Trigger durchgefÃ¼hrt, auch wenn eigentliche SQL-Anweisung scheitert
  * AFTER-Trigger nur durchgefÃ¼hrt, wenn BEFORE-Trigger (falls vorhanden)
                                    UND SQL-Anweisung klappen
  * Fehler im BEFORE/AFTER-Trigger fÃ¼hrt zu Fehler in auslÃ¶sender SQL-Anweisung
    + Transaktion:
      - Fehler der SQL-Anweisung nehmen ausgelÃ¶ste Trigger wieder zurÃ¼ck
      - Trigger-Fehler lÃ¶sen RÃ¼cknahme der ganzen SQL-Anweisung aus
    + Nicht-transakt. Tabellen:
      - AusgelÃ¶ste Trigger bei Fehler in SQL-Anweisung nicht zurÃ¼ckgenommen
      - Ebenso umgekehrt

Zeitlicher Ablauf (ro=read-only, rw=read-write) beim Abarbeiten einer
SQL-Anweisung <Stmt>:

  +-----------------------------------------------+--------+
  |  Zeitliche Reihenfolge der ausgelÃ¶ste Trigger  | Daten  |
  +-----------------+---------------+-------------+ Zugriff|
  | SQL-Anweisung   |   Key fehlt   | Key vorhanden | Daten  |
  +-----------------+---------------+---------------+--------+
  | INSERT ...      | BEFORE INSERT | BEFORE INSERT | OLD ro |
  |                 |    <Stmt>     |      ---      |        |
  |                 |  AFTER INSERT |      ---      |        |
  +-----------------+---------------+---------------+--------+
  | UPDATE ...      |      ---      | BEFORE UPDATE | OLD ro |
  |                 |      ---      |     <Stmt>    |        |
  |                 |      ---      |  AFTER UPDATE | NEW rw |
  +-----------------+---------------+---------------+--------+
  | DELETE ...      |      ---      | BEFORE DELETE | OLD ro |
  |                 |      ---      |     <Stmt>    |        |
  |                 |      ---      |  AFTER DELETE |        |
  +-----------------+---------------+---------------+--------+
  | REPLACE ...     | BEFORE INSERT | BEFORE INSERT | NEW rw |
  |                 |      ---      | BEFORE DELETE |        |
  |                 |    <Stmt>     |     <Stmt>    |        |
  |                 |      ---      |  AFTER DELETE |        |
  |                 |  AFTER INSERT |  AFTER INSERT |        |
  +-----------------+---------------+---------------+--------+
  | INSERT ...      | BEFORE INSERT | BEFORE INSERT | NEW rw |
  | ON DUPLICATE    |      ---      | BEFORE UPDATE |        |
  | KEY UPDATE ...  |    <Stmt>     |     <Stmt>    |        |
  |                 |      ---      |  AFTER UPDATE |        |
  |                 |  AFTER INSERT |      ---      |        |
  +-----------------+---------------+---------------+--------+

Definition eines Triggers:

  DELIMITER //
  CREATE
    [DEFINER {<User> | CURRENT USER}]
    TRIGGER <Trig> <Time> <Event>
  ON <Tbl>                                    # Genau EINE Tabelle
  FOR EACH ROW                                # MySQL-EinschrÃ¤nkung
  BEGIN                                       # Oder nur <Stmt>;
    <Stmt>;
```

```
    ...
  END //
  DELIMITER ;

MÃ¶gliche Trigger-Zeitpunkte <Time>:

  +---------+-----------------------+
  | Time    | Bedeutung             |
  +---------+-----------------------+
  | BEFORE  | Vor TabellenÃ¤nderung  |
  | AFTER   | Nach TabellenÃ¤nderung |
  +---------+-----------------------+

MÃ¶gliche Trigger-Ereignisse <Event>:

  +--------+-------------------------------------------------------------------+
  | Event  | Bedeutung                                                         |
  +--------+-------------------------------------------------------------------+
  | INSERT | EinfÃ¼gen einer neuen Zeile (INSERT, LOAD DATA, REPLACE)           |
  | UPDATE | Ã¤ndern einer Zeile (UPDATE)                                       |
  | DELETE | LÃ¶schen einer Zeile (DELETE, REPLACE, nicht DROP TABLE, TRUNCATE) |
  +--------+-------------------------------------------------------------------+

Zugriff auf alte/neue Werte der Tabellenspalten wÃ¤hrend Trigger-DurchfÃ¼hrung:

  +-----------+-------------------------------------------------------------+
  | Name      | Bedeutung                                                   |
  +-----------+-------------------------------------------------------------+
  | OLD.<Col> | Alter Wert vor  DELETE und UPDATE (read only)               |
  | NEW.<Col> | Neuer Wert nach INSERT und UPDATE (read write in BEFORE)    |
  +-----------+-------------------------------------------------------------+

Trigger anzeigen:

  SHOW TRIGGERS;                          # Default <Db>
  SHOW TRIGGERS LIKE "a%";                # Default <Db>, Tabellenname
  SHOW TRIGGERS FROM <Db> LIKE "a%";      # Spezielle <Db>, Tabellenname
  SELECT TRIGGER_NAME, EVENT_MANIPULATION, EVENT_OBJECT_TABLE, ACTION_STATEMENT
    FROM INFORMATION_SCHEMA.TRIGGERS
      WHERE TRIGGER_SCHEMA = <Db>;

Trigger Ã¤ndern = mit gleichem Namen einen neuen Trigger definieren

Trigger lÃ¶schen (kein Bezug auf Tabellenname!):

  DROP TRIGGER <Trig>;
  DROP TRIGGER IF EXISTS <Trig>;    # Kein Fehler falls nicht existent

Alle Trigger einer Tabelle werden mit der Tabelle gelÃ¶scht:

  DROP TABLE <Tbl>;

Beispiele fÃ¼r Trigger:

  #-------------------------------------------------------------------------
  # Trigger 0 (Summe aller eingefÃ¼gten Spaltenwerte ermitteln)
  #-------------------------------------------------------------------------
  DROP TABLE IF EXISTS account;             # LÃ¶scht auch Trigger

  CREATE TABLE account (
    acct_num INT,
    amount   DECIMAL(10,2)
  );

  SET @sum = 0;
  SET @old = 0;

  DROP TRIGGER IF EXISTS t0;

  DELIMITER //

  CREATE TRIGGER t0 BEFORE UPDATE ON account
  FOR EACH ROW
  BEGIN
    SET @sum = @sum + NEW.amount;
    SET @old = @old + OLD.amount;
  END //

  DELIMITER ;

  SHOW TRIGGERS LIKE "account%";   # Tabellename
  INSERT INTO account VALUES
```

```
  (1,   14.98),
  (2, 1937.50),
  (3, -100.00);

UPDATE account SET amount = amount * 1.19;
SELECT @old, @sum AS "Gesamt";
SET @sum = 0;
SET @old = 0;
UPDATE account SET amount = amount * 2.0 WHERE acct_num = 2;
SELECT @old, @sum AS "Gesamt";

#-----------------------------------------------------------------------
# Trigger 1 (2 Spalten immer in 1. Zeichen GROSS, Rest klein umwandeln)
# CONSTRAINT-Ersatz!
#-----------------------------------------------------------------------
DROP TRIGGER IF EXISTS t1;

CREATE TRIGGER t1 BEFORE INSERT ON pers   # FÃ¼r UPDATE gl. Trigger definieren
FOR EACH ROW
BEGIN
  SET NEW.vorname = CONCAT(UCASE(LEFT(NEW.vorname, 1)),
                           LCASE(SUBSTR(NEW.vorname, 2)))
  SET NEW.name    = CONCAT(UCASE(LEFT(NEW.name, 1)),
                           LCASE(SUBSTR(NEW.name, 2)))
END //

#-----------------------------------------------------------------------
# Trigger 2 (Preis hat minimal Wert 0 und maximal Wert 100)
# CONSTRAINT-Ersatz!
#-----------------------------------------------------------------------
DROP TRIGGER IF EXISTS t2;

CREATE TRIGGER t2 BEFORE INSERT ON artikel   # FÃ¼r UPDATE gl. Trigger definieren
FOR EACH ROW
BEGIN
  IF NEW.preis < 0 THEN
    SET NEW.preis = 0;
  ELSEIF NEW.preis > 100 THEN
    SET NEW.preis = 100;
  END IF;
END //

#-----------------------------------------------------------------------
# Trigger 3 (Alter in Jahren aus Akt. Datum - Geburtsdatum ausrechnen)
#-----------------------------------------------------------------------
DROP TRIGGER IF EXISTS t3;

CREATE TRIGGER t3 BEFORE INSERT ON age
FOR EACH ROW
BEGIN
  SET NEW.jahre = YEAR(CURDATE()) - YEAR(NEW.geburtsdatum);
END //

#-----------------------------------------------------------------------
# Trigger 4 (Bestellsumme aus Preis * Anzahl - Rabatt ausrechnen)
#-----------------------------------------------------------------------
DROP TRIGGER IF EXISTS t4;

CREATE TRIGGER t4 BEFORE UPDATE ON bestellung
FOR EACH ROW
BEGIN
  SET NEW.summe = NEW.anz * NEW.preis * (100.0 - NEW.rabatt) / 100.0;
END //

#-----------------------------------------------------------------------
# Trigger 5 (alte Spaltenwerte vor LÃ¶schen der DatensÃ¤tze in Var. merken)
#-----------------------------------------------------------------------
DROP TRIGGER IF EXISTS t5;

CREATE TRIGGER t5 AFTER DELETE ON pers
FOR EACH ROW
BEGIN
  SET @vorname = OLD.vorname;    # Bei jeder LÃ¶schoperation Ã¼berschrieben
  SET @name    = OLD.name;       # Nach Tabellen lÃ¶schen noch verfÃ¼gbar
  SET @plz     = OLD.plz;
  SET @ort     = OLD.ort;
END //

#-----------------------------------------------------------------------
# Trigger 6 (Summe aller in eine Tabelle eingefÃ¼gten Werte bilden)
#-----------------------------------------------------------------------
DROP TABLE IF EXISTS bestellung;
```

```
  CREATE TABLE bestellung (
    nr     INT,
    anz    INT,
    preis  FLOAT,
    summe  FLOAT,
    rabatt FLOAT
  );

  DROP TRIGGER IF EXISTS t6;

  CREATE TRIGGER t6 BEFORE INSERT ON bestellung
  FOR EACH ROW
  BEGIN
    # Bei jeder EinfÃ¼geoperation Ã¼berschrieben
    SET @total = @total + NEW.anz * NEW.preis * (100.0 - NEW.rabatt) / 100.0;
  END //

  SET @total = 0;
  INSERT INTO bestellung VALUES (1, 1, 65.00, 1 * 65.00, 0),
                                (2, 5, 10.99, 5 * 10.90, 2),
                                (3, 2,  5.49, 2 *  5.49, 2);
  SELECT @total AS "Gesamtsumme";

  #-----------------------------------------------------------------------------
  # Trigger 7 (Daten in andere Tabellen weitergeben)
  #-----------------------------------------------------------------------------
  DROP TRIGGER IF EXISTS t7;

  CREATE TRIGGER t7 BEFORE INSERT ON test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3                    WHERE a3 = NEW.a1;
    UPDATE      test4 SET sum = sum + 1 WHERE a4 = NEW.a4;
  END //

  # TODO

  DELIMITER ;


21) Condition Handler (Error/Warning)
-------------------------------------

Fehler- oder Warnungs-Situationen als Ergebnis-Status einer SQL-Anweisung
erfordern gelegentlich eine besondere Reaktion. Mit "Condition Handlern" wird
festgelegt, wie auf einen bestimmten SQL-Status reagiert werden soll.
Insbesondere in Triggern, Routinen (Stored Procedures) und bei der Verwendung
eines Cursors ist das hÃ¤ufig sinnvoll.

SQL-Fehler und -Warnungen werden durch einen 5-stelligen SQL-Status "XXXXX"
(SQLSTATE: in SQL standardisiert) dargestellt (mit externem Programm "perror"
in entsprechende Textmeldung umwandeln). Ihm entspricht ein MySQL-Fehlercode
(ERROR-Code: MySQL-spezifisch) aus dem er generiert wird. Die ersten beiden
Zeichen des SQL-Status "XXXXX" legen die FEHLERKLASSE fest:

  +-------+---------------------------+-----------------------+
  | Klasse| Bedeutung                 | Beendet Programmfluss |
  +-------+---------------------------+-----------------------+
  | 00... | Erfolg                    |          Nein         |
  | 01... | Warnung                   |          Nein         |
  | 02... | Nicht gefunden            |           Ja          |
  | XX... | Fehler (sonstige PrÃ¤fixe) |           Ja          |
  +-------+---------------------------+-----------------------+

Beispiele:
  * SQL-Status "00000" bzw. MySQL-Fehlercode 0 zeigen die ERFOLGREICHE
    AusfÃ¼hrung einer SQL-Anweisung an (nicht in Condition Handler verwenden!)
  * SQL-Status "02000" bzw. MySQL-Fehlercode 1329 bedeutet "No Data" und tritt
    auf, wenn ein Cursor das Datenende erreicht bzw. eine SELECT...INTO <Var>
    Anweisung keine Ergebnisdaten produziert.

Mit einer Condition-Definition wird einem SQL-Status ein frei wÃ¤hlbarer Name
<CondName> zugeordnet, der spÃ¤ter in einem Condition Handler benutzt werden
kann. Dies dient ausschlieÃM-^_lich der Bequemlichkeit und Dokumentation, es ist
auch mÃ¶glich, den SQL-Status direkt im Condition Handler zu verwenden.

  DECLARE <CondName> CONDITION FOR
    { SQLSTATE [VALUE] "XXXXX"        # SQL-Status "XXXXX"
    | <Zahl> };                       # MySQL-Fehlercode <Zahl>
```

```
Definition eines Condition Handlers fÃ¼r einen (oder mehrere) SQL-Status. Trifft
eine der Bedingungen zu (tritt ein SQL-Status auf), dann wird die zugehÃ¶rige
einfache oder Verbund-Anweisung ausgefÃ¼hrt. Ein Handler bezieht sich immer auf
den umgebenden BEGIN-END-Block, in dem er deklariert wurde. Tritt ein
SQL-Status auf, fÃ¼r den kein Handler deklariert wurde, so ist EXIT die
Standardaktion (verlassen des umgebenden BEGIN-END-Blocks).

  # Variante A (EIN Statement)        # Variante B (MEHRERE Statements)
  DECLARE <HandlerType> HANDLER       # DECLARE <HandlerType> HANDLER
    FOR <CondValue> {, <CondValue>}   #   FOR <CondValue> {, <CondValue>}
      <Stmt>;                         # BEGIN
                                      #   <Stmt>;
                                      #   ...
                                      # END;

                              # Umgebender Block (BEGIN ... END) ...
  <HandlerType> = CONTINUE    # ... lÃ¤uft nach Handler-Anweisung weiter
                | EXIT        # ... wird nach Handler-Anweisung verlassen (STD)
                | UNDO        # ... (nicht unterstÃ¼tzt)

  <CondValue> = SQLSTATE [VALUE] "XXXXX"     # SQL-Status "XXXXX"
              | <Zahl>                       # MySQL-Fehlercode <Zahl>
              | <CondName>                   # Name einer vorher def. Condition
              | SQLWARNING                   # SQL-Status "01..."
              | NOT FOUND                    # SQL-Status "02..."
              | SQLEXCEPTION                 # Rest auÃM-^_er OK, "01...", "02..."

Zum Ignorieren eines SQL-Status einen leeren BEGIN-END-Block verwenden:

  DECLARE CONTINUE HANDLER FOR SQLWARNING BEGIN END;   # Leerer BEGIN-END-Block

Typische MySQL-Fehlercodes / SQL-Stati sind durch ANSI SQL und ODBC
standardisiert; MySQL-Fehlercodes sind nur fÃ¼r MySQL relevant
(vollstÃ¤ndige Liste --> 25a) MySQL-Fehlercode / SQL-Status):

  +------+---------+-------------------------------------------------------+
  | Code | Status  | Message (Bedeutung)                                   |
  +------+---------+-------------------------------------------------------+
  |    0 | "00000" | Erfolg (kein Fehler)                                  |
  +------+---------+-------------------------------------------------------+
  | 1311 | "01000" | Referring to uninitialized variable %s                |
  +------+---------+-------------------------------------------------------+
  | 1329 | "02000" | No data -- zero rows fetched, selected, or processed  |
  +------+---------+-------------------------------------------------------+
  | 1022 | "23000" | Can't write; duplicate key in table ...               |
  | 1037 | "HY001" | Out of memory; restart server and try again           |
  | 1038 | "HY001" | Out of sort memory; increase server sort buffer size  |
  | 1040 | "08004" | Too many connections                                  |
  | 1041 | "HY000" | Out of memory                                         |
  | 1042 | "08S01" | Can't get hostname for your address                   |
  | 1044 | "42000" | Access denied for user '...' to database '...'        |
  | 1046 | "3D000" | No database selected                                  |
  | 1047 | "08S01" | Unknown command                                       |
  +------+---------+-------------------------------------------------------+
  |
  +------+---------+-------------------------------------------------------+

Beispiel 1:

  DELIMITER //
  CREATE PROCEDURE HandlerDemo1()
  BEGIN
    DECLARE uninit INT;
    DECLARE dummy  CHAR(100);

    DECLARE CONTINUE HANDLER FOR SQLSTATE "01000" BEGIN
      SELECT "Uninitialisierte Variable!";
      SET @cnt := @cnt + 1;
    END;
    DECLARE CONTINUE HANDLER FOR SQLSTATE "02000" BEGIN
      SELECT "Keine Daten selektiert!";
      SET @x := 9;
    END;

    SET @x = 1; SELECT user FROM mysql.user WHERE TRUE  LIMIT 1 INTO dummy;
    SELECT @x, @cnt, dummy, uninit + 1;
    SET @x = 2; SELECT user FROM mysql.user WHERE FALSE LIMIT 1 INTO dummy;
    SELECT @x, @cnt, dummy, uninit + 1;
    SET @x = 3; SELECT user FROM mysql.user WHERE FALSE LIMIT 1 INTO dummy;
    SELECT @x, @cnt, dummy, uninit + 1;
    SET @x = 4; SELECT user FROM mysql.user WHERE TRUE  LIMIT 1 INTO dummy;
    SELECT @x, @cnt, dummy, uninit + 1;
```

```
  END //
  DELIMITER ;

  CALL HandlerDemo1();
  DROP PROCEDURE HandlerDemo1;

Beispiel 2:

  CREATE TABLE t (
    id INT PRIMARY KEY
  );

  DELIMITER //
  CREATE PROCEDURE HandlerDemo2()
  BEGIN
    DECLARE DuplicateKeyError CONDITION FOR SQLSTATE "23000";

    DECLARE CONTINUE HANDLER FOR DuplicateKeyError SET @err = "dup1";   # Proz.

    SET @x = 1;
    INSERT INTO t VALUES (1);
    SELECT @x, @err;
    SET @x = 2;
    INSERT INTO t VALUES (2);
    SELECT @x, @err;

    BEGIN
      DECLARE EXIT HANDLER FOR DuplicateKeyError SET @err = "dup2";   # Block
      SET @x = 3;
      INSERT INTO t VALUES (3);
      SELECT @x, @err;
      SET @x = 4;
      INSERT INTO t VALUES (1);    # Ups! ---+ EXIT HANDLER
      SELECT @x, @err;             #         |
      SET @x = 5;                  #         |
      INSERT INTO t VALUES (5);    #         |
      SELECT @x, @err;             #         |
    END;                           #         |
                                   #         |
    SET @x = 6;                    # <-------+
    INSERT INTO t VALUES (6);
    SELECT @x, @err;
    SET @x = 7;
    INSERT INTO t VALUES (1);      # Ups! ---+ CONTINUE HANDLER
    SELECT @x, @err;               # <-------+
    SET @x = 8;
    INSERT INTO t VALUES (8);
    SELECT @x, @err;
  END //
  DELIMITER ;

  CALL HandlerDemo2();
  DROP PROCEDURE HandlerDemo2;
  DROP TABLE t;

  SELECT @x, @err;

Beispiel 3 (Eine Handler-Anweisung kann kein ITERATE oder LEAVE mit dem Label
eines die Handler-Anweisung umgebenden Blockes enthalten; Ã¼ber eine
Status-Variable, die der Handler verÃ¤ndert, kann der umgebende Block trotzdem
prÃ¼fen, ob der Handler ausgelÃ¶st wurde):

  # So geht es nicht!
  DELIMITER //
  CREATE PROCEDURE HandlerDemo3a()
  BEGIN
    DECLARE i INT DEFAULT 3;

    retry: REPEAT
      BEGIN
        DECLARE CONTINUE HANDLER FOR SQLWARNING BEGIN ITERATE retry; END;
      END;
      SET i := i - 1;
      SELECT i, warn;
    UNTIL i < 0 END REPEAT;
  END //
  DELIMITER ;

  CALL HandlerDemo3a();
  DROP PROCEDURE HandlerDemo3a;

  # So ist es OK!
```

```
  DELIMITER //
  CREATE PROCEDURE HandlerDemo3b()
  BEGIN
    DECLARE i    INT  DEFAULT 3;
    DECLARE warn BOOL DEFAULT FALSE;

    retry: REPEAT
      BEGIN
        DECLARE CONTINUE HANDLER FOR SQLWARNING SET warn = TRUE;
      END;
      SET i := i - 1;
      SELECT i, warn;
    UNTIL warn OR i < 0 END REPEAT;
  END //
  DELIMITER ;

  CALL HandlerDemo3b();
  DROP PROCEDURE HandlerDemo3b;


22) Cursor (Zeiger)
-------------------

Zum SEQUENTIELLEN Lesen (und evtl. Schreiben) von DatensÃ¤tzen, d.h. das
MENGENORIENTIERTE Verhalten von SQL wird umgangen. Sinnvoll z.B. in Stored
Procedures und in Anwendungen mit sequentiellem Lese/Schreibverhalten (COBOL).

EinschrÃ¤nkungen:
  * Nur in Stored Routine, Trigger und Event erlaubt
  * Asensitive (Kopie der Ergebnistabelle mÃ¶glich aber nicht unbedingt nÃ¶tig)
  * Read-only (keine Updates mÃ¶glich, MY!)
  * Forward-only
  * Nonscrollable (nur eine Leserichtung, keine DatensÃ¤tze Ã¼berspringbar, MY!)
  * Nonholdable (nach Commit geschlossen)
  * Namenslos (Handler = Cursor-ID)
  * Server-side (materialisierte temporÃ¤re Tabelle)   # Keine Client-side
  * Mehrere gleichzeitig Ã¶ffenbar
  * Verschachtelbar

Syntax:
  DECLARE <Cursor> CURSOR FOR <SelectStmt>;   # Cursor mit SQL-Anw. verknÃ¼pfen
  OPEN <Cursor>;                              # Abfrage durchfÃ¼hren (Puffer)
  FETCH <Cursor> {INTO | USING} <Var>, ...;   # EINEN Ergebnissatz in Var. holen
  CLOSE <Cursor>;                             # Pufferinhalt verwerfen

Hinweise:
  + FÃ¼hrt vollstÃ¤ndige Abfrage beim ÃM-^Vffnen des Cursors aus (evtl. LIMIT!)
    --> TemporÃ¤re Tabelle
  + Zielvariablen mÃ¼ssen zu Spalten passenden Datentyp haben
  + PrÃ¼fung per Handler Ã¼ber Statusvariable ob letzter Datensatz gelesen
  + VerknÃ¼pfung des Cursors bleibt bestehen (erneut Ã¶ffenbar)
  + Deklarations-Reihenfolge: Erst Variablen, dann Cursor, dann Handler
  + Cursor muss nicht durch alle DatensÃ¤tze laufen

HANDLER zur Ende-Erkennung notwendig (SQLSTATE 02000 bzw. NOT FOUND):

  DECLARE done INT DEFAULT 0;                                # A) Var. initial.
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;       # A) Var. setzen
  DECLARE CONTINUE HANDLER FOR SQLSTATE "02000" BEGIN END;   # B) Ignorieren

Beispiel:

  CREATE TABLE IF NOT EXISTS pers (
    nr      INT NOT NULL,
    vorname VARCHAR(30),
    name    VARCHAR(30)
  );

  # Tabelle Personen
  #CREATE TABLE copy SELECT * FROM pers;   # +Struktur, -Indices, +Daten
  CREATE TABLE copy LIKE pers;             # +Struktur, +Indices, -Daten
  INSERT INTO  copy SELECT * FROM pers;    # -Struktur, -Indices, +Daten
  CREATE TABLE res  LIKE pers;             # +Struktur, +Indices, -Daten

  DELIMITER //

  CREATE PROCEDURE cursor_demo()
  BEGIN
    DECLARE done     INT DEFAULT 0;
    DECLARE n1, n2   INT;
    DECLARE vn1, nn1 CHAR(20);
    DECLARE vn2, nn2 CHAR(20);
```

```
    DECLARE cur1 CURSOR FOR SELECT nr, vorname, name FROM pers;
    DECLARE cur2 CURSOR FOR SELECT nr, vorname, name FROM copy;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN cur1;
    OPEN cur2;

    REPEAT
      FETCH cur1 INTO n1, vn1, nn1;
      FETCH cur2 INTO n2, vn2, nn2;
      IF NOT done THEN
        IF n1 < n2 THEN
          INSERT INTO res VALUES (n1, CONCAT("A", vn2), n2);
        ELSE
          INSERT INTO res VALUES (n2, CONCAT("B", vn2), n2);
        END IF;
      END IF;
    UNTIL done END REPEAT;

    CLOSE cur1;
    CLOSE cur2;
  END //

  DELIMITER ;

  CALL cursor_demo;
  SELECT * FROM res;
  DROP PROCEDURE cursor_demo;
  TRUNCATE TABLE res;


23) Table Handler
-----------------

Zum DIREKTEN Lesen von den in einer Storage-Engine gespeicherten DatensÃ¤tzen
EINER Tabelle (direkter Zugriff auf Storage Engine Interface analog ISAM ohne
Umweg Ã¼ber SQL). VerhÃ¤lt sich wie SELECT mit folgenden Unterschieden:
  * Nur EINE Tabelle zugreifbar
  * Es wird immer auf ALLE Spalten der Tabelle zugegriffen
  * Sortierung der DatensÃ¤tze gemÃ¤ÃM-^_ physischer Speicherung oder EINEM Index
  * Positionierung in DatensÃ¤tzen mÃ¶glich

Eigenschaften:
  * Zugriff auf EINE Tabelle gleichzeitig
  * Direkter Zugriff auf Storage Engine Interface (analog ISAM)
  * VerfÃ¼gbar fÃ¼r MyISAM- und InnoDB-Engine
  * Pro Session getrennt
  * Holt DatensÃ¤tze in "natÃ¼rlicher" Reihenfolge (wie gespeichert)
  * Zum ersten/letzten Datensatz springen und rÃ¼ckwÃ¤rts durchlaufen mÃ¶glich
  * Tabelle NICHT gesperrt zwischen zwei Handleraufrufen
    --> Lesekonsistenz wird nicht erzwungen (z.B. dirty read mÃ¶glich)
  * Kein Schreibzugriff (read only)

GrÃ¼nde fÃ¼r die Nutzung:
  * Schneller "Full Table Scan" = Zugriff auf ALLE Spalten EINER Tabelle
  * Schneller und flexibler als SELECT (solange nur Zugriff auf eine Tabelle)
    + Weniger SQL-Anweisungs-Parsing notwendig
    + Kein Overhead fÃ¼r Optimierer und Query-PrÃ¼fung
  * Zur Portierung von Anwendungen, die low-level ISAM-Zugriffe einsetzen
    + Durchlaufen mehrerer/aller DatensÃ¤tze einfacher als mit SELECT
      ("Full Table Scan", z.B. fÃ¼r GUI-Anwendungen)

Tabelle Ã¶ffnen fÃ¼r Lesezugriffe (Alias statt Tabellenname verwendbar):

  HANDLER <Tbl> OPEN [[AS] <Alias>];

Hinweise:
  * EINZELNEN Datensatz aus EINER Tabelle (Zugriff Ã¼ber EINEN Index) einlesen
  * Datensatz erfÃ¼llt WHERE-Bedingung
  * Statt einem bis zu LIMIT DatensÃ¤tze einlesbar

A) ALLE DatensÃ¤tze in "natÃ¼rlicher" Reihenfolge (wie gespeichert) durchlaufen:

 HANDLER <Tbl> READ {FIRST | NEXT}
   [WHERE <Cond>]
   [LIMIT ...];

B) ALLE DatensÃ¤tze in Index-Reihenfolge (wie sortiert) durchlaufen (Name
  `PRIMARY` (Backquotes!) verwenden, um den PRIMARY KEY fÃ¼r den Zugriff zu
   benutzen):
```

```
 HANDLER <Tbl> READ <Idx> {FIRST | NEXT | PREV | LAST}
   [WHERE <Cond>]
   [LIMIT ...];

C) NUR DatensÃ¤tze ab Indextreffer in Index-Reihenfolge (wie sortiert)
   durchlaufen. Anzahl in (...) Ã¼bergebener Werte <Vali> muss kleiner gleich
   Anzahl Index-Komponenten sein (leftmost part). Die Werte mÃ¼ssen passend zur
   Reihenfolge der Index-Komponenten sein:

 HANDLER <Tbl> READ <Idx> {= | < | <= | > | >=} (<Val1>, ...)
   [WHERE <Cond>]
   [LIMIT ...];

Handler schlieÃM-^_en (erfolgt auch automatisch am Sessionende):

  HANDLER <Tbl/Alias> CLOSE;

Beispiel:

  HANDLER pers OPEN AS h_pers;

  HANDLER h_pers READ idx2 = ("Bayer");
  HANDLER h_pers READ idx2 > ("Bayer", "Heinz");
  HANDLER h_pers READ idx2 < ("Bayer");
  HANDLER h_pers READ idx2 <= ("Bayer");
  HANDLER h_pers READ idx2 >= ("Bayer") LIMIT 2;
  HANDLER h_pers READ idx2 >= ("Bayer") WHERE nr >= 9;

  HANDLER h_pers READ `PRIMARY` FIRST;
  HANDLER h_pers READ `PRIMARY` NEXT;
  HANDLER h_pers READ `PRIMARY` NEXT LIMIT 2;

  HANDLER h_pers READ `PRIMARY` LAST;
  HANDLER h_pers READ `PRIMARY` PREV;
  HANDLER h_pers READ `PRIMARY` PREV LIMIT 3;

  HANDLER h_pers READ FIRST;
  HANDLER h_pers READ NEXT;
  HANDLER h_pers READ NEXT LIMIT 3;

  HANDLER h_pers CLOSE;


24) Events (Ereignisse)
-----------------------

MySQL besitzt ab MY!5.1 einen SCHEDULER, der benutzergesteuert zu bestimmten
Zeitpunkten (Events) EINMALIG oder WIEDERHOLT Aufgaben (Tasks) in Form von
SQL-Anweisungen durchfÃ¼hrt (analog Linux "crontab"/"cronjob").

Zweck:
  * Periodische DB-VerwaltungstÃ¤tigkeiten
    + Tabellen-Statistik periodisch aktualisieren
    + Protokollierung des DB-Zustandes
    + Backup oder Synchronisation/Replikation auslÃ¶sen
    + Datenbank-Pflege auslÃ¶sen
  * Periodische Generierung von Tabellen mit Zwischenergebnissen (Top10)
    --> "Materialized" Views

Eigenschaften:
  * Ab MY!5.1 verfÃ¼gbar
  * Event-Name muss pro Datenbank <Db> eindeutig sein
  * Einmalig (AT) oder wiederholt (EVERY) mÃ¶glich (<Schedule>)
  * EVERY wiederholt im Zeitraum STARTS bis ENDS
    + Keine STARTS-Angabe --> Definitionszeitpunkt = CURRENT_TIMESTAMP()
    + Keine ENDS-Angabe   --> Gilt ewig
  * Event automatisch lÃ¶schbar, wenn erledigt (Zeitraum verstrichen)
    + ON COMPLETION NOT PRESERVE
  * Events haben die gleichen EinschrÃ¤nkungen wie Stored Procedures
    + Kann keinen Trigger / Stored Procedure / Event erzeugen
  * WÃ¤hlbar ob repliziert oder nicht (DISABLE ON SLAVE)
  * Erneutes AuslÃ¶sen wÃ¤hrend Durchlauf von MySQL nicht verhindert
    (manuell Mutex/Lock in Event-Aktion --> GET/IS_FREE/IS_USED/RELEASE_LOCK)
  * Mehrere kÃ¶nnen gleichzeitig laufen (CONNECTION_ID() ist eindeutig)
  * Kein SQL-Standard
  * Ã¼M-^\bersicht: INFORMATION_SCHEMA.EVENTS

Syntax:
  CREATE
    [DEFINER = {<User> | CURRENT_USER}]         # Ersteller
    EVENT [IF NOT EXISTS] [<Db>.]<Event>        # Event-Name (DB-eindeutig)
```

```
      ON SCHEDULE <Schedule>                    # AT/EVERY ...
        [ON COMPLETION [NOT] PRESERVE]          # Autom. lÃ¶schen bei Abschluss
        [ENABLE | DISABLE | DISABLE ON SLAVE]   # Aktivieren/Deaktivieren
        [COMMENT <String>]                      # Kommentar
      DO <Stmt>;                                # Oder DO BEGIN <Stmt>;... END;

  <Schedule> = AT <Timestamp> [+ INTERVAL <N> <Interval>]
             | EVERY <Interval> [STARTS <Timestamp>] [ENDS <Timestamp>]

  <Timestamp> = CURRENT_TIMESTAMP()      # Aktueller Zeitpunkt
              | "YYYY-MM-DD hh:mm:ss"    # Datum + Uhrzeit
              | YYYYMMDDhhmmss           # Datum + Uhrzeit

  <Interval> = YEAR | QUARTER | MONTH | WEEK | DAY
             | HOUR | MINUTE | SECOND
             | YEAR_MONTH
             | DAY_HOUR | DAY_MINUTE | DAY_SECOND
             | HOUR_MINUTE | HOUR_SECOND
             | MINUTE_SECOND

Beispiele fÃ¼r Intervalle (A=AuflÃ¶sung: M=Monat, S=Sekunde):

  +-----------------------------+---+-----------------------------------------+
  |          Intervall          | A | Bedeutung                               |
  +-----------------------------+---+-----------------------------------------+
  |             1 YEAR          | M | 1x pro Jahr                             |
  |             2 QUARTER       | M | Alle 2 Quartale                         |
  |             3 MONTH         | M | Alle 3 Monate                           |
  |             5 WEEK          | S | Alle 5 Wochen                           |
  |             3 DAY           | S | Alle 3 Tage                             |
  |             2 HOUR          | S | Alle 2 Stunden                          |
  |            15 MINUTE        | S | Alle 15 Minuten                         |
  |            45 SECOND        | S | Alle 45 Sekunden                        |
  +-----------------------------+---+-----------------------------------------+
  |         "2-2" YEAR_MONTH    | M | Alle 2 Jahre und 2 Monate               |
  |       "10 05" DAY_HOUR      | S | Alle 10 Tage und 5 STD                  |
  |    "10 02:10" DAY_MINUTE    | S | Alle 10 Tage, 2 STD und 10 Min          |
  | "10 02:10:30" DAY_SECOND    | S | Alle 10 Tage, 2 STD, 10 Min und 30 Sek  |
  |       "10:05" HOUR_MINUTE   | S | Alle 10 STD und 5 Min                   |
  |    "10:05:30" HOUR_SECOND   | S | Alle 10 STD, 5 Min und 30 Sek           |
  |       "10:30" MINUTE_SECOND | S | Alle 10 Min und 30 Sek                  |
  +-----------------------------+---+-----------------------------------------+

ÃM-^\bliche Praxis: Komplexe Aufgabe als Stored Procedure schreiben +
                per CALL in Event aufrufen
                --> Testen einfacher
                --> Manueller Aufruf jederzeit mÃ¶glich

Beispiel (Stored Procedure "optimize_tables()"):

  DROP EVENT IF EXISTS optimize_db;
  CREATE EVENT optimize_db
    ON SCHEDULE
      EVERY 1 WEEK
    DO
      CALL optimize_tables('db');

Erneutes AuslÃ¶sen wÃ¤hrend langem Durchlauf verhindern:

  DROP EVENT IF EXISTS optimize_db;
  CREATE EVENT optimize_db
    ON SCHEDULE
      EVERY 1 WEEK
  BEGIN
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION   # "dummy" continue handler
      BEGIN /*empty*/ END;
    IF GET_LOCK('optimize_db', 0) THEN
      DO CALL optimize_tables('db');
    END IF;
    DO RELEASE_LOCK('optimize_db');
  END

Beispiel (in einer Stunde 1x UPDATEn --- AT):

  DROP EVENT IF EXISTS e_next_hour;
  CREATE EVENT e_next_hour
    ON SCHEDULE
      AT CURRENT_TIMESTAMP() + INTERVAL 1 HOUR
    DO
      UPDATE myschema.mytable SET mycol = mycol + 1;

Beispiel (stÃ¼ndlich die Sitzungstabelle lÃ¶schen --- EVERY):
```

```
  DROP EVENT IF EXISTS e_next_hour;
  CREATE EVENT e_hourly_sess_delete
    ON SCHEDULE
      EVERY 1 HOUR
      COMMENT "StÃ¼ndlich die Sitzungstabelle lÃ¶schen"
    DO
      DELETE FROM site_activity.sessions;
      # TRUNCATE site_activity.sessions;   # Evtl. nicht erlaubt

Beispiel (am nÃ¤chsten Tag eine Stored Procedure aufrufen --- AT):

  DROP EVENT IF EXISTS e_call_myproc;
  CREATE EVENT e_call_myproc
    ON SCHEDULE
      AT CURRENT_TIMESTAMP() + INTERVAL 1 DAY
    DO
      CALL myproc(5, 27);

Beispiel (alle 5 Sekunden eine Anweisungsfolge durchfÃ¼hren --- EVERY):

  DELIMITER //
  DROP EVENT IF EXISTS e_every_5_sec //
  CREATE EVENT e_every_5_sec
    ON SCHEDULE
      EVERY 5 SECOND
    DO BEGIN
      DECLARE CONTINUE HANDLER FOR SQLEXCEPTION BEGIN END;
      INSERT INTO t1 VALUES (0);
      UPDATE t2 SET s1 = s1 + 1;
    END //
  DELIMITER ;

Beispiel (tÃ¤glich Anzahl Sitzungen summieren und Sitzungstabelle lÃ¶schen --- EVERY):

  DELIMITER //
  DROP EVENT IF EXISTS e_daily_aggregate_sess //
  CREATE EVENT e_daily_aggregate_sess
    ON SCHEDULE
      EVERY 1 DAY
      COMMENT "Summe Anzahl Sitzungen sichern und Sitzungstabelle lÃ¶schen"
    DO BEGIN
      INSERT INTO site_activity.totals (when, total)
        SELECT CURRENT_TIMESTAMP(), COUNT(*)
          FROM site_activity.sessions;
      DELETE FROM site_activity.sessions;
    END //
  DELIMITER ;

Beispiel (zwei Tabellen wÃ¶chentlich um 1 Uhr Mitternacht optimieren --- EVERY):

  DELIMITER //
  DROP EVENT IF EXISTS e_opt_tables //
  CREATE EVENT e_opt_tables
    ON SCHEDULE
      EVERY 1 WEEK
      STARTS "2015-01-01 00:00:00"
      ENDS   "2015-12-31 00:00:00"
      ON COMPLETION NOT PRESERVE
    DO BEGIN
      OPTIMIZE TABLE test.t1;
      OPTIMIZE TABLE test.t2;
    END //
  DELIMITER ;

Beispiel (jeden Sonntag etwas durchfÃ¼hren --- EVERY):

  DELIMITER //
  DROP EVENT IF EXISTS e_every_sunday_do_something //
  CREATE EVENT e_every_sunday_do_something
    ON SCHEDULE
      -- Taeglich den Event ausfuehren
      EVERY 1 DAY
      -- Immer um 23:00 soll der Event stattfinden
      STARTS "2014-01-01 23:00:00"
    DO BEGIN
      -- Heute Sonntag (Wert 6)? --> Nur dann etwas durchfuehren!
      IF WEEKDAY(CURRENT_DATE()) = 6 THEN
        -- Eigentliche Taetigkeit
        OPTIMIZE TABLE test.t1;
        OPTIMIZE TABLE test.t2;
      END IF;
```

```
    END //
  DELIMITER ;

Beispiel (jeden Arbeitstag Mo-Fr etwas durchfÃ¼hren --- EVERY):

  DELIMITER //
  DROP EVENT IF EXISTS e_every_workday_do_something //
  CREATE EVENT e_every_workday_do_something
    ON SCHEDULE
      -- Taeglich den Event ausfuehren
      EVERY 1 DAY
      -- Immer um 23:30 soll der Event stattfinden
      STARTS "2014-01-01 23:30:00"
    DO BEGIN
      -- Heute Wochentag (Wert 0-4)? --> Nur dann etwas durchfuehren!
      IF WEEKDAY(CURRENT_DATE()) >= 0 AND WEEKDAY(CURRENT_DATE()) <= 4 THEN
        -- Eigentliche Taetigkeit
        -- Schleife ueber alle Tab. einer Db. (als Parameter Ã¼bergeben)
        PREPARE ...
        EXEC
      END IF;
    END //
  DELIMITER ;

Zustand des Event-Schedulers anzeigen/Ã¤ndern (ein Thread):

  SET GLOBAL event_scheduler = 1;     # 1=ON, 0=OFF/DISABLED
  SELECT @@event_scheduler;
  SHOW VARIABLES LIKE "event_scheduler%";

Liste aller/einiger Events mit Eigenschaften anzeigen:

  SHOW EVENTS;
  SHOW EVENTS LIKE "optimize_%";

Definition eines Events anzeigen:

  SHOW CREATE EVENT <Event>;

Event Ã¤ndern (z.B. AusfÃ¼hrungszeitraum und -zeitpunkte Ã¤ndern):

  ALTER
    [DEFINER = {<User> | CURRENT_USER}]
    EVENT <Event>
      [ON SCHEDULE <Schedule>]
        [ON COMPLETION [NOT] PRESERVE]
        [RENAME TO <NewEvent>]
        [ENABLE | DISABLE | DISABLE ON SLAVE]
        [COMMENT <String>]
        [DO <Stmt>];                # Oder DO BEGIN <Stmt>;... END;

Event deaktivieren/aktivieren:

  ALTER EVENT <Event> DISABLE;
  ALTER EVENT <Event> ENABLE;

Event umbenennen:

  ALTER EVENT <Event> RENAME TO <NewEvent>;

Event lÃ¶schen:

  DROP EVENT <Event>;
  DROP EVENT IF EXISTS <Event>;   # Kein Fehler falls nicht existent

Recht vergeben, Events zu erstellen, Ã¤ndern und lÃ¶schen:

  GRANT EVENT ON <Db>.* TO <User>@<Host>;

Recht wegnehmen, Events zu erstellen, Ã¤ndern und lÃ¶schen:

  REVOKE EVENT ON <Db>.* TO <User>@<Host>;


25) Signale
-----------

Signale dienen zur RÃ¼ckgabe eines Fehlerwertes an einen Handler, einen Benutzer
einer Anwendung oder einen Client (MY!5.5).
  * Legen folgende Fehler-Eigenschaften fest:
    + Fehler-Nummer
    + Status-Wert
```

```
    + Fehler-Nachricht
  * Signale sind nicht nur in Compound Statements, sondern Ã¼berall verwendbar
    (z.B. in Event, Trigger, SQL-Anweisung: MY!-Erweiterung)
  * Signale kÃ¶nnen nur SQL-Status verwenden, keine MySQL-Fehlercodes
    + VollstÃ¤ndige Liste --> 25a) MySQL-Fehlercode / SQL-Status
  * Signal-Klassen:

  +-------+--------------------------+-----------------------+
  | Klasse| Bedeutung                | Beendet Programmfluss |
  +-------+--------------------------+-----------------------+
  | 00... | Erfolg                   |          Nein         |
  | 01... | Warnung                  |          Nein         |
  | 02... | Nicht gefunden           |           Ja          |
  | XX... | Fehler (sonstige PrÃ¤fixe) |           Ja          |
  +-------+--------------------------+-----------------------+

Syntax:
  SIGNAL <CondValue>                        # ZurÃ¼ckzugebener Fehlerwert
    [SET <SignalInfo> {, <SignalInfo>}]

  <CondValue> = SQLSTATE [VALUE] "XXXXX"    # SQL-Status "XXXXX"
              | <CondName>                  # Name einer vorher def. Condition

  <SignalInfo> = <CondInfoItem> = <SimpelValueSpec>

  <CondInfoItem> = { CLASS_ORIGIN          #
                   | SUBCLASS_ORIGIN       #
                   | CONSTRAINT_CATALOG    #
                   | CONSTRAINT_SCHEMA     #
                   | CONSTRAINT_NAME       #
                   | CATALOG_NAME          #
                   | SCHEMA_NAME           #
                   | TABLE_NAME            #
                   | COLUMN_NAME           #
                   | CURSOR_NAME           #
                   | MESSAGE_TEXT          #
                   | MYSQL_ERRNO }         #

  <SimpleValueSpec> =

Beispiel:

  CREATE PROCEDURE p(err INT)
  BEGIN
    DECLARE user CONDITION FOR SQLSTATE "45000";
    IF err = 0 THEN
      SIGNAL SQLSTATE "01000";                      # Warnung --> Weiter
    ELSEIF err = 1 THEN
      SIGNAL SQLSTATE "45000"                       # Fehler --> Abbruch
        SET MESSAGE_TEXT = "An error occurred";
    ELSEIF err = 2 THEN
      SIGNAL user                                   # Fehler --> Abbruch
        SET MESSAGE_TEXT = "An error occurred";
    ELSE
      SIGNAL SQLSTATE "01000"                       # Warnung --> Weiter
        SET MESSAGE_TEXT = "A warning occurred",
            MYSQL_ERRNO  = 1000;
      SIGNAL SQLSTATE "45000"                       # Fehler --> Abbruch
        SET MESSAGE_TEXT = "An error occurred",
            MYSQL_ERRNO  = 1001;
    END IF;
  END;

  SIGNAL SQLSTATE "77777";
  CREATE TRIGGER t_bi BEFORE INSERT ON t
    FOR EACH ROW SIGNAL SQLSTATE "77777";
  CREATE EVENT e ON SCHEDULE EVERY 1 SECOND
    DO SIGNAL SQLSTATE "77777";

  CREATE PROCEDURE p (divisor INT)
  BEGIN
    DECLARE divide_by_zero CONDITION FOR SQLSTATE "22012";
    IF divisor = 0 THEN
      SIGNAL divide_by_zero;
    END IF;
  END;


25a) MySQL-Fehlercode / SQL-Status
----------------------------------

SQL-Fehler und -Warnungen werden durch einen 5-stelligen SQL-Status "XXXXX"
```

dargestellt (per externem Programm "perror" in entsprechende Textmeldung
umwandelbar). Ihm entspricht ein MySQL-Fehlercode, aus dem er generiert wird.
Die ersten beiden Zeichen des SQL-Status "XXXXX" legen die FEHLERKLASSE fest:

Klasse	Bedeutung	Beendet Programmfluss
00...	Erfolg	Nein
01...	Warnung	Nein
02...	Nicht gefunden	Ja
NN...	Fehler (sonstige PrÃ¤fixe)	Ja

Beispiele:
  * SQL-Status "00000" bzw. MySQL-Fehlercode 0 zeigen die ERFOLGREICHE
    AusfÃ¼hrung einer SQL-Anweisung an (nicht in Condition Handler verwenden!)
  * SQL-Status "02000" bzw. MySQL-Fehlercode 1329 bedeutet "No Data" und tritt
    auf, wenn ein Cursor das Datenende erreicht bzw. eine SELECT...INTO <Var>
    Anweisung keine Ergebnisdaten produziert.

Mit einer "Condition-Definition" kann einem SQL-Status ein frei wÃ¤hlbarer Name
<CondName> zugeordnet werden, der spÃ¤ter in einem "Condition Handler" genutzt
werden kann. Dies dient ausschlieÃM-^_lich der Bequemlichkeit und Dokumentation, es
ist auch mÃ¶glich, den SQL-Status direkt im Condition Handler zu verwenden.

```
  DECLARE <CondName> CONDITION FOR
    { SQLSTATE [VALUE] "XXXXX"      # SQL-Status "XXXXX"
    | <Zahl> };                     # MySQL-Fehlercode <Zahl>
```

Mit "Condition Handlern" wird festgelegt, wie auf bestimmte SQL-Status reagiert
werden soll. Zum Ignorieren eines SQL-Status einen leeren BEGIN-END-Block
verwenden:

```
  DECLARE CONTINUE HANDLER FOR SQLWARNING BEGIN END;   # Leerer BEGIN-END-Block
```

Signale dienen zur RÃ¼ckgabe eines Fehlerwertes an einen Handler, einen Benutzer
einer Anwendung oder einen Client (MY!5.5). Sie kÃ¶nnen nur einen SQL-Status
verwenden, keine MySQL-Fehlercodes

Zu finden in folgenden Dateien:
  * SQL-Status (5-stellige Nummern + Namen), durch ANSI SQL und ODBC standardisiert
    --> "include/sql_state.h"
  * MySQL-Fehlercode (mapped auf SQL-Status)
    --> "include/mysqld_error.h"
  * Messages
    --> "mysql/share/german/errmsg.txt"
    --> "mysql/share/english/errmsg.txt"

26) Begrenzungen von MySQL (Limits)

MySQL kennt folgende Begrenzungen (teilweise Engine-abhÃ¤ngig):

Limit	Bedeutung
65535	Byte pro Datensatz (InnoDB: 8000)
4096	Spalten pro Tabelle (InnoDB: 1000)
4 Mrd	Datenbanken (InnoDB)
4 Mrd	Tabellen pro Datenbank (InnoDB)
64 TB	Tablespace-GrÃ¶ÃM-^_e (InnoDB)
64 TB	Tabellen-GrÃ¶ÃM-^_e (InnoDB)
4 Mrd	DatensÃ¤tze pro Tabelle (32 Bit Zeiger, alt)
2x10^19	DatensÃ¤tze pro Tabelle (64 Bit Zeiger, neu)
64	Indices pro Tabelle (bei Neukompilierung bis zu 128)
16	Spalten zu "Composite Index" kombinierbar (mind.)
1000	Byte Index-LÃ¤nge (InnoDB: ersten 767)
61	Tabellen pro View
61	Tabellen pro Join
8192	Partitionen pro Tabelle (1024 bis MY!5.6.6)
1 GB	StatementlÃ¤nge (Protokoll, "--max-allowed-packet")
4 GB	KommentarlÃ¤nge
16384	Verbindungen ("--max_connections", STD: 151)

Maximale LÃ¤nge von Bezeichnern (siehe 6.2.2 Privilege System Grant Tables):

Max	Bezeichner	
64	Datenbank	
64	Tabelle	
64	Spalte	
64	Routine	
64	Alias	# Vor MY!5.6 255 Zeichen erlaubt
60	Host	
16	Benutzer	
41	Passwort	

Begrenzungen der InnoDB-Engine:

Max.	Bedeutung
8000	Byte pro Datensatz (etwa halbe Speicherseite)
1000	Spalten pro Tabelle
767	Byte Index-LÃ¤nge

27) MySQL testen

Zum Testen eines MySQL-Server oder seiner Client-API stehen einige Programme
zur VerfÃ¼gung:

Programm	Beschreibung
mysqlslap	Server-Lasttest per Simulation vieler Clients
mysql-test-run.pl	Server-Test per "mysqltest" steuern (Skript)
mysqltest	Server-Test und Vergleich mit anderen Ergeb.
mysqltest_embedded	Server-Test und Vergleich mit anderen Ergeb.
mysql_client_test	Test der MySQL-Client-API (Skript)
mysql_client_test_embedded	Test der MySQL-Client-API (Skript)

Ein typischer Datenbank-Lasttest des MySQL-Servers per "mysqlslap"
(mÃ¶gliche Werte von "sql-load-type": read, write, key, update, mixed):

```
  mysqlslap -utom -pgeheim \
            --engine=myisam \
            --iterations=200 \
            --concurrency=20 \
            --number-char-cols=10 \
            --number-int-cols=4 \
            --detach=10000 \
            --auto-generate-sql \
            --auto-generate-sql-load-type=mixed \
            --debug-check \
            --debug-info \
            --verbose


==========================================================================
```