MySQL™ – "after select"

Vortrag vom 06.05.2004 auf der PHP-Usergroup Hannover

von Frank Staude <staude@trilos.de>

1 EINLEITUNG 7

2 VORAUSSETZUNGEN 7

3 DIE LIZENZ VON MYSQL 8

4 KOMANDOZEILENPOWER 9

4.1 Html export 9

4.2 XML export 10

4.3 Spaltenüberschriften unterdrücken 10

4.4 Ausgaben nummerieren 11

4.5 Lange Ausgabezeilen sind unleserlich 12

5 TABELLENTYPEN 14

5.1 HEAP 14

5.2 ISAM 15

5.3 InnoDB 15

5.4 MERGE 15

5.5 MyISAM 17

6 TRANSAKTIONEN 17

7 MYSQL PRAXIS 18

7.1 Mit NULL Werten umgehen 18

7.2 Einen Vergleich ausgeben 19

7.3 Stringverarbeitung 20

7.3.1 Einen Teilstring suchen 20

7.3.2 Mustervergleich mit SQL-Mustern 20

7.3.3 Mustervergleich mit regulären Ausdrücken 21

7.4 Umgang mit Datumswerten 21

7.5 Weitere Spannende Dinge 22

8 TOOLS 22

8.1 phpMyAdmin 22

8.2 DB Designer4 ..... **25**

8.3 MYSQL Admin **28**

9 ANHANG **30**

9.1 Datentypen **30**
9.1.1 Numerische Datentypen 30
9.1.1.1 TINYINT[(M)] [UNSIGNED] [ZEROFILL] 30
9.1.1.2 SMALLINT[(M)] [UNSIGNED] [ZEROFILL] 30
9.1.1.3 MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL] 30
9.1.1.4 INT[(M)] [UNSIGNED] [ZEROFILL] 30
9.1.1.5 INTEGER[(M)] [UNSIGNED] [ZEROFILL] 30
9.1.1.6 BIGINT[(M)] [UNSIGNED] [ZEROFILL] 30
9.1.1.7 FLOAT(genauigkeit) [ZEROFILL] 31
9.1.1.8 FLOAT[(M,D)] [ZEROFILL] 31
9.1.1.9 DOUBLE[(M,D)] [ZEROFILL] 31
9.1.1.10 DOUBLE PRECISION[(M,D)] [ZEROFILL] 31
9.1.1.11 REAL[(M,D)] [ZEROFILL] 31
9.1.1.12 DECIMAL[(M[,D])] [ZEROFILL] 31
9.1.1.13 NUMERIC(M,D) [ZEROFILL] 31
9.1.2 String Datentypen 32
9.1.2.1 [NATIONAL] CHAR(M) [BINARY] 32
9.1.2.2 [NATIONAL] VARCHAR(M) [BINARY] 32
9.1.2.3 TINYBLOB | TINYTEXT 32
9.1.2.4 BLOB | TEXT 32
9.1.2.5 MEDIUMBLOB | MEDIUMTEXT 32
9.1.2.6 LONGBLOB | LONGTEXT 32
9.1.3 Datumstypen 32
9.1.3.1 DATE 32
9.1.3.2 DATETIME 33
9.1.3.3 TIMESTAMP[(M)] 33
9.1.3.4 TIME 33
9.1.3.5 YEAR[(2|4)] 33
9.1.4 Komplexe Datentypen 33
9.1.4.1 ENUM('wert1','wert2',...) 33
9.1.4.2 SET('wert1','wert2',...) 33

9.2 Funktionen **33**
9.2.1 Nicht typenspezifische Operatoren und Funktionen 34
9.2.1.1 Klammer 34
9.2.1.2 Vergleichsoperatoren 34
9.2.1.3 = ... 34
9.2.1.4 <> , != 35
9.2.1.5 <= 35
9.2.1.6 < 35
9.2.1.7 >= 35
9.2.1.8 > 35
9.2.1.9 <=> 35
9.2.1.10 IS NULL , IS NOT NULL 35
9.2.1.11 ausdruck BETWEEN min AND max 35
9.2.1.12 ausdruck IN (wert,...) 36
9.2.1.13 ausdruck NOT IN (wert,...) 36
9.2.1.14 ISNULL(ausdruck) 36
9.2.1.15 COALESCE(liste) 36
9.2.1.16 INTERVAL(N,N1,N2,N3,...) 37
9.2.1.17 Logische Operatoren 37
9.2.1.18 NOT , ! 37
9.2.1.19 OR , || 37
9.2.1.20 AND , && 38
9.2.1.21 Ablaufsteuerungsfunktionen 38

9.2.1.22 IFNULL(ausdruck1,ausdruck2) ... 38
9.2.1.23 NULLIF(ausdruck1,ausdruck2) ... 38
9.2.1.24 IF(ausdruck1,ausdruck2,ausdruck3) ... 38
9.2.1.25 CASE ... 39
9.2.2 Zeichenketten-Funktionen ... 39
9.2.2.1 ASCII(zeichenkette) ... 39
9.2.2.2 ORD(zeichenkette) ... 39
9.2.2.3 CONV(N,von_basis,zu_basis) ... 40
9.2.2.4 BIN(N) ... 40
9.2.2.5 OCT(N) ... 40
9.2.2.6 HEX(N) ... 40
9.2.2.7 CHAR(N,...) ... 40
9.2.2.8 CONCAT(zeichenkette1,zeichenkette2,...) ... 41
9.2.2.9 CONCAT_WS(trennzeichen, zeichenkette1, zeichenkette2,...) ... 41
9.2.2.10 LENGTH, OCTET_LENGTH, CHAR_LENGTH, CHARACTER_LENGTH ... 41
9.2.2.11 LOCATE, POSITION ... 41
9.2.2.12 LOCATE(teilzeichenfolge,zeichenkette,position) ... 41
9.2.2.13 INSTR(zeichenkette,teilzeichenfolge) ... 42
9.2.2.14 LPAD(zeichenkette,laenge,fuellzeichenkette) ... 42
9.2.2.15 RPAD(zeichenkette,laenge,fuellzeichenkette) ... 42
9.2.2.16 LEFT(zeichenkette,laenge) ... 42
9.2.2.17 RIGHT(zeichenkette,laenge) ... 42
9.2.2.18 SUBSTRING, MID ... 42
9.2.2.19 LTRIM(zeichenkette) ... 43
9.2.2.20 RTRIM(zeichenkette) ... 43
9.2.2.21 TRIM ... 43
9.2.2.22 SOUNDEX(zeichenkette) ... 43
9.2.2.23 SPACE(N) ... 44
9.2.2.24 REPLACE ... 44
9.2.2.25 REPEAT(zeichenkette,zaehler) ... 44
9.2.2.26 REVERSE(zeichenkette) ... 44
9.2.2.27 insert ... 44
9.2.2.28 ELT ... 44
9.2.2.29 FIELD ... 45
9.2.2.30 FIND_IN_SET ... 45
9.2.2.31 MAKE_SET ... 45
9.2.2.32 EXPORT_SET ... 45
9.2.2.33 LCASE, LOWER ... 45
9.2.2.34 UCASE, UPPER ... 46
9.2.2.35 LOAD_FILE(datei) ... 46
9.2.3 Zeichenketten-Vergleichsfunktionen ... 46
9.2.3.1 ausdruck LIKE muster [ESCAPE 'fluchtzeichen'] ... 47
9.2.3.2 ausdruck NOT LIKE muster [ESCAPE 'fluchtzeichen'] ... 47
9.2.3.3 ausdruck REGEXP muster , ausdruck RLIKE muster ... 47
9.2.3.4 ausdruck NOT REGEXP muster ,ausdruck NOT RLIKE muster ... 48
9.2.3.5 STRCMP(ausdruck1,ausdruck2) ... 48
9.2.3.6 MATCH (spalte1,spalte2,...) AGAINST (ausdruck) ... 48
9.2.4 Groß-/Kleinschreibung ... 48
9.2.5 Numerische Funktionen ... 49
9.2.5.1 Arithmetische Operationen ... 49
9.2.5.2 + ... 49
9.2.5.3 - ... 49
9.2.5.4 * ... 49
9.2.5.5 / ... 49
9.2.5.6 Mathematische Funktionen ... 49
9.2.5.7 - ... 49
9.2.5.8 ABS(X) ... 50
9.2.5.9 SIGN(X) ... 50
9.2.5.10 MOD(N,M) , % ... 50
9.2.5.11 FLOOR(X) ... 50
9.2.5.12 CEILING(X) ... 50

9.2.5.13 ROUND(X) ... 50
9.2.5.14 ROUND(X,D) ... 51
9.2.5.15 EXP(X) ... 51
9.2.5.16 LOG(X) ... 51
9.2.5.17 LOG10(X) ... 51
9.2.5.18 POW(X,Y) , POWER(X,Y) ... 51
9.2.5.19 SQRT(X) ... 51
9.2.5.20 PI() ... 51
9.2.5.21 COS(X) ... 52
9.2.5.22 SIN(X) ... 52
9.2.5.23 TAN(X) ... 52
9.2.5.24 ACOS(X) ... 52
9.2.5.25 ASIN(X) ... 52
9.2.5.26 ATAN(X) ... 52
9.2.5.27 ATAN2(Y,X) ... 52
9.2.5.28 COT(X) ... 53
9.2.5.29 RAND() , RAND(N) ... 53
9.2.5.30 LEAST(X,Y,...) ... 53
9.2.5.31 GREATEST(X,Y,...) ... 54
9.2.5.32 DEGREES(X) ... 54
9.2.5.33 RADIANS(X) ... 54
9.2.5.34 TRUNCATE(X,D) ... 54
9.2.6 Datums- und Zeit-Funktionen ... 54
9.2.6.1 DAYOFWEEK(datum) ... 54
9.2.6.2 WEEKDAY(datum) ... 54
9.2.6.3 DAYOFMONTH(datum) ... 55
9.2.6.4 DAYOFYEAR(datum) ... 55
9.2.6.5 MONTH(datum) ... 55
9.2.6.6 DAYNAME(datum) ... 55
9.2.6.7 MONTHNAME(datum) ... 55
9.2.6.8 QUARTER(datum) ... 55
9.2.6.9 WEEK(datum) , WEEK(datum,erste) ... 55
9.2.6.10 YEAR(datum) ... 55
9.2.6.11 YEARWEEK(datum) , YEARWEEK(datum,erste) ... 55
9.2.6.12 HOUR(zeit) ... 56
9.2.6.13 MINUTE(zeit) ... 56
9.2.6.14 SECOND(zeit) ... 56
9.2.6.15 PERIOD_ADD(P,N) ... 56
9.2.6.16 PERIOD_DIFF(P1,P2) ... 56
9.2.6.17 DATE_ADD, DATE_SUB, ADDDATE, SUBDATE ... 56
9.2.6.18 EXTRACT(typ FROM datum) ... 58
9.2.6.19 TO_DAYS(datum) ... 58
9.2.6.20 FROM_DAYS(N) ... 58
9.2.6.21 DATE_FORMAT(datum,format) ... 58
9.2.6.22 TIME_FORMAT(zeit,format) ... 60
9.2.6.23 CURDATE() , CURRENT_DATE ... 60
9.2.6.24 CURTIME() , CURRENT_TIME ... 60
9.2.6.25 NOW() , SYSDATE() , CURRENT_TIMESTAMP ... 60
9.2.6.26 UNIX_TIMESTAMP() , UNIX_TIMESTAMP(datum) ... 60
9.2.6.27 FROM_UNIXTIME(unix_zeitstempel) ... 61
9.2.6.28 FROM_UNIXTIME(unix_zeitstempel,format) ... 61
9.2.6.29 SEC_TO_TIME(sekunden) ... 61
9.2.6.30 TIME_TO_SEC(zeit) ... 61
9.2.7 Weitere Funktionen ... 61
9.2.7.1 Bit-Funktionen ... 61
9.2.7.2 | ... 61
9.2.7.3 & ... 61
9.2.7.4 << ... 61
9.2.7.5 >> ... 62
9.2.7.6 ~ ... 62
9.2.7.7 BIT_COUNT(N) ... 62

9.2.7.8 Verschiedene Funktionen ... 62
9.2.7.9 DATABASE() ... 62
9.2.7.10 USER() , SYSTEM_USER() , SESSION_USER() ... 62
9.2.7.11 PASSWORD(zeichenkette) ... 62
9.2.7.12 ENCRYPT(zeichenkette[,salt]) ... 62
9.2.7.13 ENCODE(zeichenkette,passwort_zeichenkette) ... 63
9.2.7.14 DECODE(crypt_zeichenkette,passwort_zeichenkette) ... 63
9.2.7.15 MD5(zeichenkette) ... 63
9.2.7.16 LAST_INSERT_ID([ausdruck]) ... 63
9.2.7.17 FORMAT(X,D) ... 64
9.2.7.18 VERSION() ... 64
9.2.7.19 CONNECTION_ID() ... 64
9.2.7.20 GET_LOCK(zeichenkette,zeitueberschreitung) ... 64
9.2.7.21 RELEASE_LOCK(zeichenkette) ... 65
9.2.7.22 BENCHMARK(zaehler,ausdruck) ... 65
9.2.7.23 INET_NTOA(ausdruck) ... 65
9.2.7.24 INET_ATON(ausdruck) ... 65
9.2.7.25 MASTER_POS_WAIT(log_name, log_position) ... 65
9.2.8 Funktionen zur Benutzung bei GROUP BY-Klauseln ... 66
9.2.8.1 COUNT(ausdruck) ... 66
9.2.8.2 COUNT(DISTINCT ausdruck,[ausdruck...]) ... 66
9.2.8.3 AVG(ausdruck) ... 66
9.2.8.4 MIN(ausdruck) , MAX(ausdruck) ... 66
9.2.8.5 SUM(ausdruck) ... 66
9.2.8.6 STD(ausdruck) , STDDEV(ausdruck) ... 67
9.2.8.7 BIT_OR(ausdruck) ... 67
9.2.8.8 BIT_AND(ausdruck) ... 67

9.3 Links ... 67

1 Einleitung

Warum ein Vortrag über MySQL und warum “after select”?

Ich habe im laufe der Jahre in vielen Gesprächen die Erfahrung gemacht das MySQL zwar sehr, sehr häufig verwendet wird, aber die wenigsten wissen was für ein mächtiges Werkzeug sie da in den Händen halten – bei vielen „Hobbyprogrammierern“ geht es über *update*, *insert* und ein *select * from* nicht hinaus was oftmals zur folge hat das MySQL gar nicht zeigen kann was es kann bzw. das oftmals Aufgaben umständlich durch zusätzliche Programmierung erledigt werden, die man eigentlich durch MySQL hätte erledigen können. Oftmals hat dies auch zur Folge, dass die Performance sehr zu wünschen lässt – deshalb der Titel "after select".

Dieser Vortrag kann und will nicht ein MySQL – Handbuch oder ein allgemeines Datenbanken - Buch ersetzen – ganz im Gegenteil. Auch kann ich nicht all das schreiben, was ich für interessant und wichtig halte, dann hätten wir eine Wochenschulung. Aber ich habe die Hoffnung, den einen oder anderen hiermit zumindest Appetit auf mehr zu machen.

Ich würde mich aber freuen, wenn dieser Vortrag den einen oder anderen dazu animiert, sich mit entsprechender Literatur zu befassen.

Der Anhang ist etwas ausführlicher geworden, als zunächst geplant. Ich fand es aber ganz sinnvoll, wenn man diesen Vortrag neben dem Rechner liegen hat und ihn auch als Nachschlagewerk verwenden kann. Deshalb habe ich mich dazu entschieden, alle Funktionen und Datentypen die MySQL bietet im Anhang aufzulisten. Diese Auflistung ist dem wirklich guten MySQL-Handbuch entnommen.

Das Titelbild ist von der Internet Movie Database (www.imdb.com) und zeigt Buster Keaton in dem Film "The Navigator"

2 Voraussetzungen

Alle Beispiele in diesem Vortrag sind auf MySQL. 4.0.18 entstanden. PHP Code wurde unter PHP 4.3.3 unter Windows erstellt. Dennoch ist keines der Beispiele in irgendeiner Form Plattformabhängig. Windows ist es deshalb, weil es die Plattform ist, die mir überall da zur Verfügung stand, wo ich an diesem Text gearbeitet habe. Des weiteren setze ich voraus, dass PHP und MySQL installiert ist, dass Du grundsätzlich weißt wie Relationale Datenbanken funktionieren und auch schon mit PHP etwas aus einer MySQL Datenbank ausgelesen hast.

3 Die Lizenz von MySQL

Die Lizenz von MySQL gibt immer wieder Anlass zu Spekulationen, weil das Programm sowohl als GPL Version verfügbar ist, als auch in einer Kommerziellen Version.

Dabei ist es ganz einfach: Wer mit den Einschränkungen der GPL Version nicht leben kann bzw. will, braucht die kommerzielle. Ich zitiere dazu aus dem MySQL Handbuch von mysql.de

Im Wesentlichen ist unsere Lizenzpolitik und die Interpretation der GPL wie folgt:

Beachten Sie bitte, dass ältere Versionen von MySQL immer noch einer strengeren Lizenz unterliegen. Sehen Sie in der Dokumentation der betreffenden Version wegen entsprechender Informationen nach. Wenn Sie eine kommerzielle Lizenz benötigen, weil die GPL-Lizenz nicht zu den Anforderungen Ihrer Applikation passt, können Sie eine Lizenz unter https://order.mysql.com/ kaufen.

Für normalen internen Gebrauch kostet MySQL nichts. Sie brauchen uns nichts zu bezahlen, wenn Sie nicht wollen.

Eine Lizenz wird benötigt:

- *Wenn Sie ein Programm, das nicht freie Software ist, mit Code des MySQL-Servers oder der Client-Programme verbinden, die den GPL-Copyrights unterliegen. Das ist zum Beispiel der Fall, wenn Sie MySQL als eingebetteten Server (Embedded Server) in Ihren Applikationen benutzen, oder wenn Sie dem MySQL-Server Erweiterungen hinzufügen, die nicht freie Software sind. In diesen Fällen würden Ihre Applikation bzw. Ihr Code ebenfalls GPL werden, weil die GPL in solchen Fällen wie ein Virus wirkt. Sie können dieses Problem vermeiden, wenn Sie den MySQL-Server mit einer kommerziellen Lizenz von MySQL AB erwerben. Siehe http://www.gnu.org/copyleft/gpl-faq.html.*
- *Wenn Sie eine kommerzielle Applikation haben, die NUR mit MySQL funktioniert, und wenn Sie die Applikation zusammen mit dem MySQL-Server ausliefern. Wir betrachten so etwas als Einbindung, selbst wenn es über das Netzwerk geschieht.*
- *Wenn Sie eine Distribution von MySQL besitzen und nicht den Quelltext für Ihre Kopie des MySQL-Servers zur Verfügung stellen, so wie es in der GPL-Lizenz festgelegt ist.*

*Eine Lizenz wird **NICHT** benötigt:*

- *Sie benötigen keine Lizenz, wenn Sie den Client-Code in kommerzielle Programme einschließen. Der Client-Teil von MySQL unterliegt der LGPL GNU Library General Public License-Lizenz. Der mysql-Kommandozeilen-Client beinhaltet Code der readline-Bibliothek, die unter GPL steht.*
- *Wenn Sie für Ihre Art der Benutzung von MySQL keine Lizenz benötigen, aber MySQL mögen und die weitere Entwicklung fördern wollen, sind Sie herzlich eingeladen, in jedem Fall eine Lizenz zu erwerben.*
- *Wenn Sie MySQL in einem kommerziellen Zusammenhang benutzen und davon profitieren, bitten wir Sie, dass Sie die Weiterentwicklung von MySQL*

fördern, indem Sie einen bestimmten Grad von Support kaufen. Wir meinen, dass es vernünftig ist, wenn wir Sie bitten, MySQL zu unterstützen, wenn MySQL Ihren Geschäften hilft. (Ansonsten würde es bei Support-Anfragen bedeuten, dass Sie nicht nur etwas für umsonst benutzen, in das wir eine Menge Arbeit gesteckt haben, sondern dass Sie uns auch noch auffordern, kostenlosen Support zu liefern.)

In Situationen, wo eine MySQL-Lizenz benötigt wird, brauchen Sie eine Lizenz pro Maschine, auf der der MySQL-Server läuft. Eine Mehrprozessor-Maschine zählt jedoch als eine einzelne Maschine, und es gibt keine Beschränkung hinsichtlich der Anzahl von MySQL-Servern, die auf einer Maschine laufen, oder hinsichtlich der Anzahl von Clients, die zur gleichen Zeit mit einem Server verbunden sind, der auf dieser Maschine läuft!

Falls Sie nicht sicher sind, ob für Ihre spezielle Benutzung von MySQL eine Lizenz erforderlich ist, lesen Sie diesen Abschnitt bitte nochmals, bevor Sie uns kontaktieren.

4 Kommandozeilenpower

Den Anfang macht die Kommandozeile. Auch hier gibt es zwei weitgehenst unbeobachtete Parameter, deren Kenntnis einem aber durchaus Programmierarbeit ersparen kann, geht es nur um das erzeugen eines HTML oder XML Exports des Ergebnisses einer Abfrage.

4.1 Html export

Gegeben ist folgende Tabelle:

mysql> select * from os;

```
+----+---------+
| id | os      |
+----+---------+
|  1 | Linux   |
|  3 | Solaris |
|  4 | Irix    |
|  5 | QNX     |
|  6 | Windows |
+----+---------+
```

Um dieses Ergebnis „schnell“ nach HTML zu wandeln, verwenden wir den Parameter –H der HTML Output erzeugt. Nicht verwechseln mit –h der den Host spezifiziert. Da meine Datenbank „vortrag“ heißt und die das MySQL lokal auf meinem Rechner laufen habe, lautet der vollständige Aufruf:

```
mysql vortrag -H -u username -p -e "select * from os"
Enter password: ******

Und das Ergebnis sieht so aus:

<TABLE BORDER=1>
<TR>
  <TH>id</TH><TH>os</TH>
</TR>
```

```
<TR>
  <TD>1</TD><TD>Linux</TD>
</TR>
<TR>
  <TD>3</TD><TD>Solaris</TD>
</TR>
<TR>
  <TD>4</TD><TD>Irix</TD>
</TR>
<TR>
  <TD>5</TD><TD>QNX</TD>
</TR>
<TR>
  <TD>6</TD><TD>Windows</TD>
</TR>
</TABLE>
```

4.2 XML export

Nach dem –H für den HTML-Export steht, ist es nicht weiter verwunderlich, dass man mit –X an der Kommandozeile das Ergebnis als XML bekommt. Die vollständige Aufrufzeile lautet:

```
mysql vortrag -X -u username -p -e "select * from os"
Enter password: ******

Und liefert als Ergebnis folgendes:

<?xml version="1.0"?>
<resultset statement="select * from os">
  <row>
        <id>1</id>
        <os>Linux</os>
  </row>
  <row>
        <id>3</id>
        <os>Solaris</os>
  </row>
  <row>
        <id>4</id>
        <os>Irix</os>
  </row>
  <row>
        <id>5</id>
        <os>QNX</os>
  </row>
  <row>
        <id>6</id>
        <os>Windows</os>
  </row>
</resultset>
```

4.3 Spaltenüberschriften unterdrücken

Manchmal möchte man nicht, dass in dem Abfrageergebnis die Spaltenüberschriften mit ausgegeben werden. Natürlich könnte man sich mit sed an der Kommandozeile da was passendes zurecht bauen, aber das ist nicht nötig. MySQL stellt da sogar einen passenden Schalter beim Client zur Verfügung.

Normalerweise sieht das Ergebnis so aus:

```
mysql vortrag   -e "select * from os"  -u username -p
Enter password: ******

+----+---------+
| id | os      |
+----+---------+
|  1 | Linux   |
|  3 | Solaris |
|  4 | Irix    |
|  5 | QNX     |
|  6 | Windows |
+----+---------+
```

Mit dem Schalter -N oder auch –skip-column-names kann man MySQL veranlassen die Spaltenbeschriftung zu unterdrücken.

```
mysql vortrag -N   -e "select * from os"  -u username -p
Enter password: ******

+---+---------+
| 1 |   Linux |
| 3 | Solaris |
| 4 |    Irix |
| 5 |     QNX |
| 6 | Windows |
+---+---------+
```

4.4 Ausgaben nummerieren

Es gibt Situationen, da möchte man seine Ausgabezeilen nummeriert haben. Das kann man natürlich an der Kommandozeile machen, indem man die Ausgabe von MySQL in ein cat –n pipe't oder mit PHP bei einer Webanwendung selbst nummeriert – aber warum? MySQL kann auch hier die Arbeit gleich mit übernehmen.

Beispielausgabe:

```
mysql vortrag   -e "select os, id from os order by os"  -u username -p
Enter password: ******

+---------+----+
| os      | id |
+---------+----+
| Irix    |  4 |
| Linux   |  1 |
| QNX     |  5 |
| Solaris |  3 |
| Windows |  6 |
+---------+----+
```

Diese Ausgabe möchte ich durch eine Zeilennummer ergänzen:

```
mysql vortrag   -e "set @z=0; select @z := @z+1 as zeile, os, id from os
order by os"  -u username -p
Enter password: ******
```

```
+-------+---------+----+
| zeile | os      | id |
+-------+---------+----+
|     1 | Irix    |  4 |
|     2 | Linux   |  1 |
|     3 | QNX     |  5 |
|     4 | Solaris |  3 |
|     5 | Windows |  6 |
+-------+---------+----+
```

Was passiert hier? Es werden zwei SQL Kommandos abgesetzt.
`@z=0;`
Definiert eine SQL-Variable und weist ihr den Wert 0 zu.

```
select @z := @z+1 as zeile, os, id from os order by os
```

Der Variablen Z wird ein Wert zugewiesen der sich aus dem alten Wert der Variable Z addiert mit 1 ergibt. Diese Addition wird für jede Zeile des Abfrageergebnisses durchgeführt. Die Variable Z wird als Spalte ‚zeile' ausgegeben.

4.5 Lange Ausgabezeilen sind unleserlich

Wenn das Ergebnis einer Abfrage sehr viele Spalten liefert ist die Ausgabe sehr schwer zu lesen.

```
mysql mysql  -e "select * from user"  -u username -p
Enter password: ******
+-----------+-------+-----------------+------------+------------+-------
------+------------+------------+-----------+------------+-------------
-+--------------+-----------+-----------+-----------------+------------+--
---------+-------------+-----------+----------------------+------------
------+-------------+-----------------+------------------+----------+-----
-------+------------+-------------+--------------+------------+--------
---------+
| Host      | User  | password          | Select_priv | Insert_priv |
pdate_priv | Delete_priv | Create_priv | Drop_priv | Reload_priv |
Shutdown_priv | Process_priv | File_priv | Grant_priv | References_priv |
Index_priv | Alter_priv | Show_db_priv | Super_priv | Create_tmp_table_priv
| Lock_tables_priv | Execute_priv | Repl_slave_priv | Repl_client_priv |
ssl_type | ssl_cipher | x509_issuer | x509_subject | max_questions |
max_updates | max_connections |
+-----------+-------+-----------------+------------+------------+-------
------+------------+------------+-----------+------------+-------------
-+--------------+-----------+-----------+-----------------+------------+--
---------+-------------+-----------+----------------------+------------
------+-------------+-----------------+------------------+----------+-----
-------+------------+-------------+--------------+------------+--------
---------+
| localhost | root  | 02d316f8753be75a | Y           | Y           | Y | Y
| Y           | Y         | Y           | Y            | Y      | Y
Y          | Y               | Y          | Y          | Y           | Y
Y                     | Y               | Y | Y               | Y
|           |             |           |             0 |           0 |
|| %         | root  | 02d316f8753be75a | Y           | Y           | Y | Y
Y          | Y         | Y           | Y            | Y      | Y         |
| Y               | Y          | Y          | Y          | Y           | Y
Y                | Y | Y               | Y                |          |
|                       |             0 |           0 |               0 |
```

Das Ergebnis wird überschaubarer, wenn man MySQL veranlasst in das „vertikale“ Ausgabeformat umzuschalten. Dies geschieht an der Kommandozeile durch den Parameter –E bzw. –vertical oder im Interaktiven Arbeiten durch Abschließen der Eingabe mit \G statt ; oder \g

Dann wird das Ergebnis wesentlich menschenfreundlicher präsentiert.

```
mysql mysql  -e "select * from user\G"  -u username -p
Enter password: ******
*************************** 1. row ***************************
                 Host: localhost
                 User: root
             password: 124Mj6f8gw6ns75a
          Select_priv: Y
          Insert_priv: Y
          Update_priv: Y
          Delete_priv: Y
          Create_priv: Y
            Drop_priv: Y
          Reload_priv: Y
        Shutdown_priv: Y
         Process_priv: Y
            File_priv: Y
           Grant_priv: Y
      References_priv: Y
           Index_priv: Y
           Alter_priv: Y
         Show_db_priv: Y
           Super_priv: Y
Create_tmp_table_priv: Y
     Lock_tables_priv: Y
         Execute_priv: Y
      Repl_slave_priv: Y
     Repl_client_priv: Y
             ssl_type:
           ssl_cipher:
          x509_issuer:
         x509_subject:
        max_questions: 0
          max_updates: 0
      max_connections: 0
```

5 Tabellentypen

Standardmäßig sind neue Tabellen in MySQL vom Typ MyISAM. Aber welche Tabellentypen gibt es noch? Welche Tabelle zu welcher Gelegenheit? Eine erste Übersicht liefert folgende Tabelle.

Tabellentyp	Transaktionen
HEAP	Nein
ISAM	Nein
InnoDB	Ja
MERGE	Nein
MyISAM	Nein

Man kann zwei Arten von Tabellen unterscheiden: transaktionssichere Tabellen und nicht transaktionssichere Tabellen.
Die Vorteile transaktionssicherer Tabellen

- Sicherer. Auch bei Absturz kann die Datenbank über automatische Wiederherstellung oder Backup + Transaktions-Log wiederhergestellt werden.
- Man kann mehrere Statements kombinieren und diese in einem Rutsch mit COMMIT akzeptieren.
- Man kann mit ROLLBACK die in der Datenbank gemachten Änderungen verwerfen.

Die Vorteile nicht transaktionssicherer Tabellen:

- Viel schneller, weil kein transaktionsoverhead.
- Weniger Speicherbedarf, weil kein transaktionsoverhead.
- Weniger Speicherbedarf im Hauptspeicher bei Aktualisierungen.

Natürlich können bei Abfragen transaktionssichere und nicht transaktionssichere Tabellen kombiniert werden.

5.1 HEAP

HEAP-Tabellen werden im Arbeitsspeicher gespeichert. Das macht sie sehr schnell, aber wenn MySQL abstürzt, sind alle Daten weg. HEAP-Tabellen sind sehr nützlich für temporäre Tabellen.

- HEAP-Tabellen haben folgende Eigenschaften:
- HEAP unterstützt keine BLOB/TEXT-Spalten.
- HEAP unterstützt keine AUTO_INCREMENT-Spalten.
- HEAP unterstützt keinen Index auf eine NULL-Spalte.
- Es darf keine nicht eindeutigen Schlüssel auf eine HEAP-Tabelle geben
- Du kannst nicht nach dem nächsten Eintrag in der Reihenfolge suchen (also den Index benutzen, um ein ORDER BY zu machen).
- Du brauchst genug zusätzlichen Arbeitsspeicher für alle HEAP-Tabellen, die Du gleichzeitig benutzen willst. ☺

5.2 ISAM

ISAM ist das alte, ursprüngliche Datenformat von MySQL. Es ist inzwischen von MyISAM abgelöst und wird in absehbarer Zeit nicht mehr zur Verfügung stehen. Der Index wird in einer Datei mit der Endung .ISM gespeichert, und die Daten in einer Datei mit der Endung .ISD. Tabellen vom Typ ISAM können mit dem Dienstprogramm isamchk geprüft / repariert werden.

ISAM-Tabellen haben folgende Eigenschaften:

- Komprimierte und Festlängen-Schlüssel
- Feste und dynamische Datensatzlängen
- 16 Schlüssel mit 16 Schlüsselteilen pro Schlüssel
- Maximale Schlüssellänge 256 (Vorgabe)
- Daten werden im Maschinenformat gespeichert. Das ist schnell, aber Maschinen- / Betriebssystem-abhängig.
- ISAM-Tabellen sind nicht binärportabel zwischen verschiedenen Betriebssystemen / Plattformen.
- Handhabt keine Tabellen > 4 GB.

5.3 InnoDB

InnoDB-Tabellen sind transaktions-sicher. Desweiteren unterstützen sie FOREIGN KEY-Beschränkungen. InnoDB speichert seine Tabellen und Indexe in einem Tabellenplatz (Tablespace), der aus mehreren Dateien bestehen kann. Das unterscheidet sich beispielsweise von MyISAM-Tabellen, bei denen jede Tabelle als separate Datei gespeichert ist. InnoDB-Tabellen können jede beliebige Größe annehmen, sogar auf Betriebssystemen, deren Dateigröße auf 2 GB beschränkt ist.

Zur Handhabung von Transaktionen siehe Punkt 6

Eine Fremdschlüsseldefinition lautet wie folgt

```
FOREIGN KEY (index_spalten_name, ...) REFERENCES tabellen_name
(index_spalten_name, ...)
```

Es müssen beide Tabellen vom Typ InnoDB sein.

```
CREATE TABLE eltern(id INT NOT NULL, PRIMARY KEY (id)) TYPE=INNODB;
CREATE TABLE kind(id INT, eltern_id INT, INDEX par_ind (eltern_id),
           FOREIGN KEY (eltern_id) REFERENCES eltern(id)) TYPE=INNODB;
```

Wer Fremdschlüssel einsetzen will sollte unbedingt einen Blick in das MySQL Handbuch zu dem Thema werfen und sich das genaue Verhalten verinnerlichen. Auch um entscheiden zu können, ob das, was er damit erreichen möchte, auch wirklich realisierbar ist.

5.4 MERGE

Eine MERGE-Tabelle ist eine Sammlung von identischen MyISAM-Tabellen, die wie eine einzige Tabelle benutzt werden können. Auf diese Sammlung von Tabellen können nur SELECT, DELETE und UPDATE ausgeführt werden. Wenn Du eine

MERGE-Tabelle gelöscht (DROP) hast, dann ist nur die Zusammenfassung der Tabellen zu einer Tabelle gelöscht, nicht die tatsächlichen Tabellen. Identische Tabellen heißt, dass die Tabellen mit identischen Spalten- und Schlüsselinformationen erzeugt wurden.

Beispielsweise könnte man die Log-Informationen des Webservers jeweils monatsweise in eine Tabelle schreiben. Die vergangenen Monate könnten mit myisampack komprimiert sein und dennoch könnte man mit einer MERGE-Tabelle über alle Tabellen mit Log-Informationen alle durchsuchen.

Eine andere Anwendung wäre auf diese Art und Weise die vorhandenen Dateigrößenbeschränkungen der Betriebssysteme zu umgehen.

Allerdings haben MERGE-Tabellen auch Nachteile:

- Man kann nur identische MyISAM-Tabellen für eine MERGE-Tabelle benutzen.
- AUTO_INCREMENT-Spalten werden bei INSERT nicht automatisch aktualisiert.
- REPLACE funktioniert nicht.
- MERGE-Tabellen benutzen mehr Datei-Deskriptoren. Eine MERGE-Tabelle, die über 10 Tabellen mappt, und 10 Benutzer diese benutzen, benötigt 10 * 10 + 10 Datei-Deskriptoren (10 Daten-Dateien für 10 Benutzer und 10 gemeinsam genutzte Index-Dateien).
- Lesevorgänge von Schlüsseln sind langsamer.

Wie lege ich denn nun so was an?

```
CREATE TABLE t1 (a INT AUTO_INCREMENT PRIMARY KEY, nachricht CHAR(20));

CREATE TABLE t2 (a INT AUTO_INCREMENT PRIMARY KEY, nachricht CHAR(20));


INSERT INTO t1 (nachricht) VALUES ("test"),("tabelle"),("t1");

INSERT INTO t2 (nachricht) VALUES ("test"),("tabelle"),("t2");


CREATE TABLE gesamt (a INT NOT NULL, nachricht CHAR(20), KEY(a)) TYPE=MERGE
UNION=(t1,t2) INSERT_METHOD=LAST;
```

Das kann man dann so abfragen:

```
mysql> select * from gesamt;
+---+-----------+
| a | nachricht |
+---+-----------+
| 1 | test      |
| 2 | table     |
| 3 | t1        |
| 1 | test      |
| 2 | table     |
| 3 | t2        |
+---+-----------+
```

5.5 MyISAM

MyISAM ist der default Tabellentyp in MySQL.

Eigenschaften von MyISAM Tabellen

- Unterstützung für große Dateien (63-Bit) auf Dateisystemen / Betriebssystemen, die große Dateien unterstützen.
- Alle Daten werden mit dem niedrigen Byte zuerst gespeichert. Das macht die Daten Maschinen- und Betriebssystem-unabhängig. (Außer evt. Embedded Systemen)
- BLOB- und TEXT-Spalten können indiziert werden.
- NULL-Werte sind in indizierten Spalten erlaubt.
- Die maximale Schlüssellänge beträgt vorgabemäßig 500 Bytes

6 Transaktionen

Eine Transaktion fasst eine Reihe von Anweisungen zusammen und sichert uns die folgenden Eigenschaften zu:

- Kein anderer Client kann die in der Transaktion verwendeten Daten verändern, während die Transaktion noch läuft. Das System verhält sich so, als hätte man den Server ganz für sich alleine. Transaktionen bewirken also eine Serialisierung des Zugriffs auf gemeinsam genutzte Ressourcen bei Operationen, die aus mehreren Anweisungen bestehen.
- Anweisungen einer Transaktion werden als Einheit betrachtet und bestätigt (COMMIT), aber nur, wenn alle Operationen erfolgreich ausgeführt werden. Tritt ein Fehler auf, werden alle vor diesem Fehler vorgenommenen Operationen zurückgenommen (ROLLBACK). Die verwendeten Tabellen befinden sich nach dem Rollback wieder in dem Ursprungszustand, ganz so als wären die Anweisungen nie ausgeführt worden.

Um mit Transaktionen zu arbeiten bedarf es nicht sehr viel.

- Eine oder mehrere Tabellen von einem Typ, der Transaktionen unterstützt.
- Sicherstellen, dass der autocommit Modus ausgeschaltet ist.

Beispiel:

```
mysql> CREATE TABLE t (i INT) TYPE = InnoDB;

mysql> begin;

mysql> INSERT INTO t (i) VALUES(1);

mysql> INSERT INTO t (i) VALUES(2);

mysql> COMMIT;
```

```
mysql> SELECT * FROM t;
+------+
| i    |
+------+
|    1 |
|    2 |
+------+

Bei einem Fehler verwendet man statt dem COMMIT ein ROLLBACK um die
Transaktion wieder aufzuheben.

mysql> CREATE TABLE t (i INT) TYPE = InnoDB;

mysql> begin;

mysql> INSERT INTO t (i) VALUES(1);

mysql> INSERT INTO t (x) VALUES(2);

ERROR 1054 at line 5: Unknown column 'x' in 'field list'

mysql> ROLLBACK;


mysql> SELECT * FROM t;
+------+
| i    |
+------+
```

Statt immer einen Anweisungsblock mit BEGIN einzufassen kann auch mit SET AUTOCOMMIT = 1; der Autocommitmodus ausgeschaltet werden.
ABER: **NICHT ALLES** kann mit ROLLBACK rückgängig gemacht werden. Wer ein DROP TABLE durchführt sollte sich nicht wundern, dass die Tabelle auch nach einem ROLLBACK verschwunden bleibt.

7 MySQL Praxis

7.1 Mit NULL Werten umgehen

Ein Abfrageergebnis enthält NULL-Werte, wie gehe ich in PHP damit um?

PHP stellt in Ergebnismengen NULL-Werte als nicht gesetzte Werte dar. Das heißt, dass wir mit der Funktion isset() einen solchen nicht gesetzten Wert entdecken können. Das sieht so aus:

```
$result = mysql_query ("SELECT *  FROM TEST", $dbh);
while ($row = mysql_fetch_row($result))
{
      while (list ($key, $value) = each ($row))
      {
            If (!isset($row[$key]))
                  $row[$key] = "NULL";
      }
}
```

In PHP4 gibt es den Wert NULL. Hier findet der Vergleich dann mit dem Dreifach-gleich-Operator statt.

```
If ($row[$key] === NULL)
      $row[$key] = "NULL";
```

7.2 Einen Vergleich ausgeben

Manchmal hat man ja das Problem, dass eine Where-Klausel nicht so ganz das tut was man erwartet, oder man will wissen, warum sie genau das Ergebnis ausgibt, das sie ausgibt. Dazu wäre es gut, wenn man sich die Ergebnisse von Vergleichen anzeigen lassen könnte.
Man kann die Ergebnisse von Vergleichen ausgeben, wenn man sie "normal" in sein Select schreibt:

```
mysql> select 'a' > 'A';
+-----------+
| 'a' > 'A' |
+-----------+
|         0 |
+-----------+
```

```
mysql> select 'a' = 'A';
+-----------+
| 'a' = 'A' |
+-----------+
|         1 |
+-----------+
```

Wenn wir also folgende Abfrage an unsere Tabelle mit den Betriebssystemen stellen:

```
mysql> select id, os from os where id > 4;
+----+---------+
| id | os      |
+----+---------+
|  5 | QNX     |
|  6 | Windows |
+----+---------+
```

Und wir wissen wollen warum MySQL so entschieden hat, dann holen wir die Bedingung nach vorne in das select und lassen uns den Vergleich ausgeben.

```
mysql> select id, os, id > 4 from os;
+----+---------+--------+
| id | os      | id > 4 |
+----+---------+--------+
|  1 | Linux   |      0 |
|  3 | Solaris |      0 |
|  4 | Irix    |      0 |
|  5 | QNX     |      1 |
|  6 | Windows |      1 |
+----+---------+--------+
```

7.3 Stringverarbeitung

7.3.1 Einen Teilstring suchen

Um herauszufinden, ob ein bestimmter String in einem anderen String vorkommt, kann man die LOCATE Funktion verwenden. LOCATE hat 2 feste und ein optionales Argument.
Argument 1 ist der String, nachdem gesucht wird. Argument 2 ist der String, in dem gesucht wird und der dritte Parameter ist die Position, ab der gesucht wird. Der Rückgabewert ist die Position, an der der gesuchte String gefunden wurde, oder 0, wenn es keinen Treffer gegeben hat.

```
mysql> select os, locate('i',os) from os;
+---------+----------------+
| os      | locate('i',os) |
+---------+----------------+
| Linux   |              2 |
| Solaris |              6 |
| Irix    |              1 |
| QNX     |              0 |
| Windows |              2 |
+---------+----------------+
```

```
mysql> select os, locate('i',os,4) from os;
+---------+------------------+
| os      | locate('i',os,4) |
+---------+------------------+
| Linux   |                0 |
| Solaris |                6 |
| Irix    |                0 |
| QNX     |                0 |
| Windows |                0 |
+---------+------------------+
```

7.3.2 Mustervergleich mit SQL-Mustern

Anstelle eines einfachen Vergleiches kann man mit den Funktionen LIKE und NOT LIKE einen Mustervergleich durchführen.
Vergleiche mit SQL-Mustern benutzen statt = und != die Operatoren LIKE und NOT LIKE, um einen Vergleich mit einem String durchzuführen. Die Suchmuster können zwei besondere Metazeichen enthalten: _ für jedes einzelne Zeichen und % für eine beliebige Folge von Zeichen.

```
mysql> select os from os where os LIKE 'in';
Empty set (0.07 sec)
```

```
mysql> select os from os where os LIKE '%in%';
+---------+
| os      |
+---------+
| Linux   |
| Windows |
+---------+
```

```
mysql> select os from os where os LIKE '_in%';
+---------+
| os      |
+---------+
| Linux   |
| Windows |
+---------+
```

7.3.3 Mustervergleich mit regulären Ausdrücken

Wenn die Möglichkeiten von LIKE nicht mehr ausreichen, dann kann man auch mit Hilfe von Regulären Ausdrücken suchen. MySQL bietet dafür den REGEXP Operator. Reguläre Ausdrücke werde ich nicht weiter beschreiben. Zu einem, weil wir vor 3 Monaten darüber einen Vortrag hier in der PHPUG Hannover hatten, zum anderen, weil man alleine über das Thema ganze Bücher schreiben kann. Oreilly hat da übrigens ein ganz gutes zu im Angebot.

Suchen wir also mit einem RegExp in der Datenbank;

```
mysql> select os from os where os REGEXP 'in.*';

+---------+
| os      |
+---------+
| Linux   |
| Windows |
+---------+


mysql> select os from os where os REGEXP '^Win';

+---------+
| os      |
+---------+
| Windows |
+---------+
```

7.4 Umgang mit Datumswerten

MySQL gibt Datumswerte normalerweise in dem Format YYYY-MM-DD aus. Das entspricht ISO-8601 und ist damit das gültige Datumsformat, allerdings ist der Mensch in unseren Breitengraden mehr an DD.MM.YYYY gewöhnt. Also muss das in der Ausgabe entsprechend gewandelt werden.

Mit Hilfe der vielen Datums und Zeit-Funktionen von MySQL (Kapitel 9.2.6) können alle erdenklichen Konvertierungen direkt von der Datenbank erledigt werden, so dass man sich in seinen Clientprogramm da nicht drum kümmern muss.

Auch das Konvertieren von Datumswerten zu Tagen, Sekunden, Rechnen mit Zeitintervallen und Altersberechnungen sind dort bei den jeweiligen Funktionen beschrieben.

7.5 Weitere Spannende Dinge…

…die MySQL kann, aber hier und heute nicht weiter im Detail beleuchtet werden sollen.

- FULLTEXT zum Suchen in Texten
- Konfiguration des Servers
- Mitgelieferte Tools wie mysqladmin, mysqldump, mysqlimport, mysqlhotcopy usw.
- Lokalisierung
- Die Logdateien
- Optimierung
- Replikation
- Die Kommandos OPTIMIZE TABLE und ANALYZE TABLE

Ja, es macht Sinn das MySQL Handbuch zu lesen ☺

8 Tools

Im folgenden 3 Programme die einem das Leben und den Umgang mit MySQL wesentlich leichter machen.

8.1 phpMyAdmin

Den phpMyAdmin muss ich wahrscheinlich nicht vorstellen. Er ist das defakto Tool zum bearbeiten von MySQL Tabellen und wahrscheinlich bei (fast) jedem Hoster bereits mit installiert.

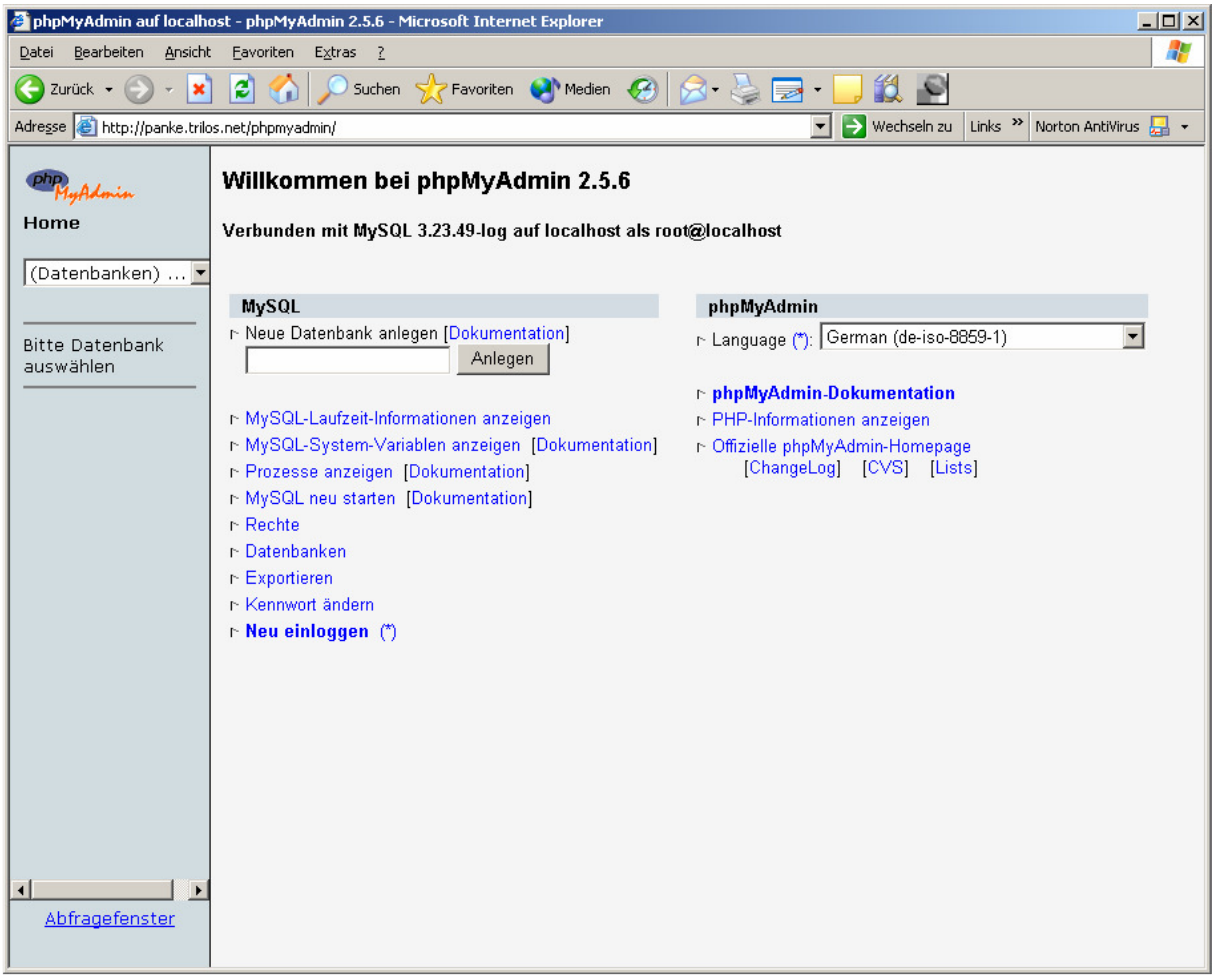


Abbildung 1 Startseite phpMyAdmin

Der phpMyAdmin ist ein Webbasiertes Datenbank-Frontend. Mit ihm kann man seine Tabellen per Browser pflegen. Neue Datenbanken / Tabellen anlegen, Inhalte bearbeiten. Datenbanken sichern und vieles mehr.

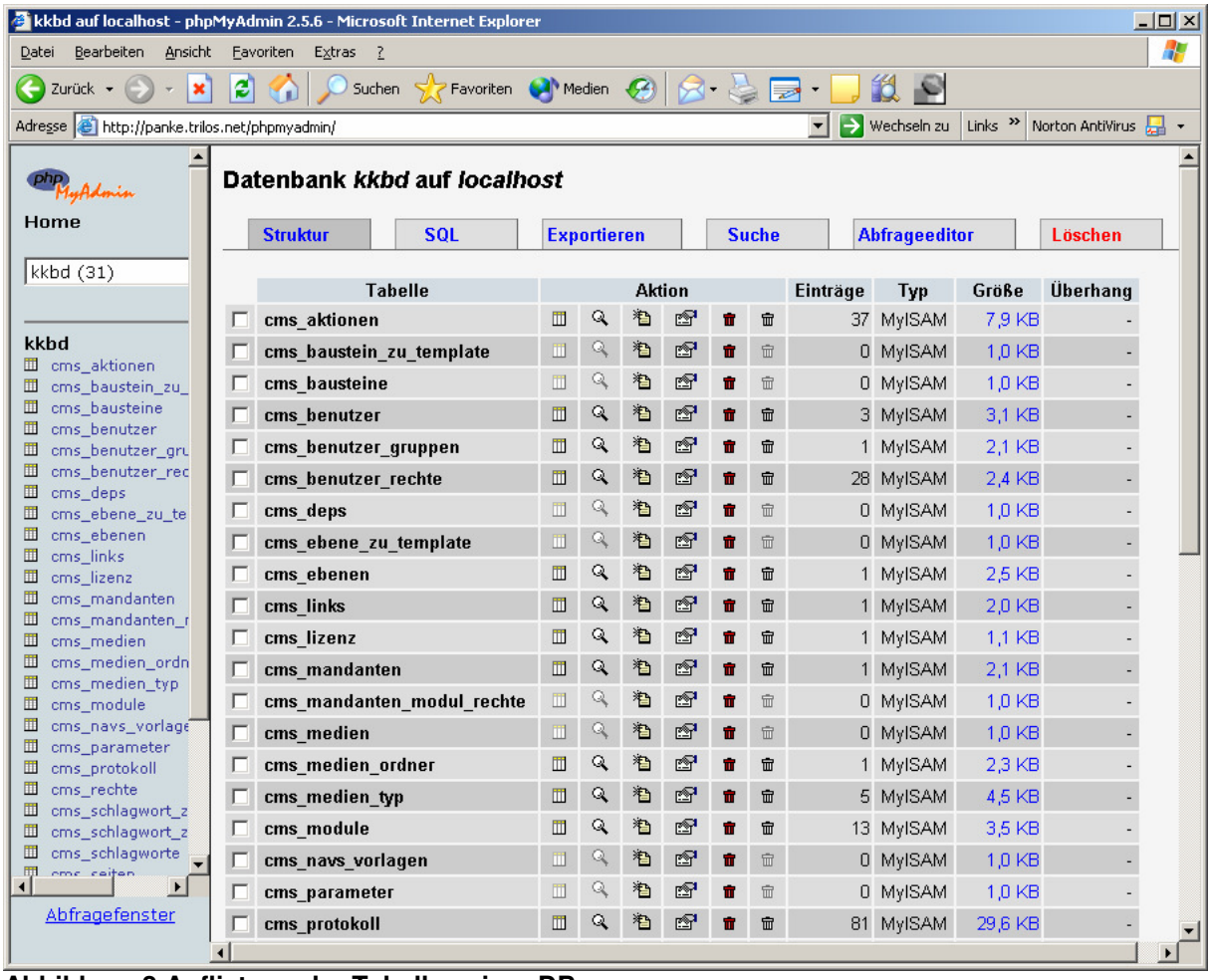


Abbildung 2 Auflistung der Tabellen einer DB

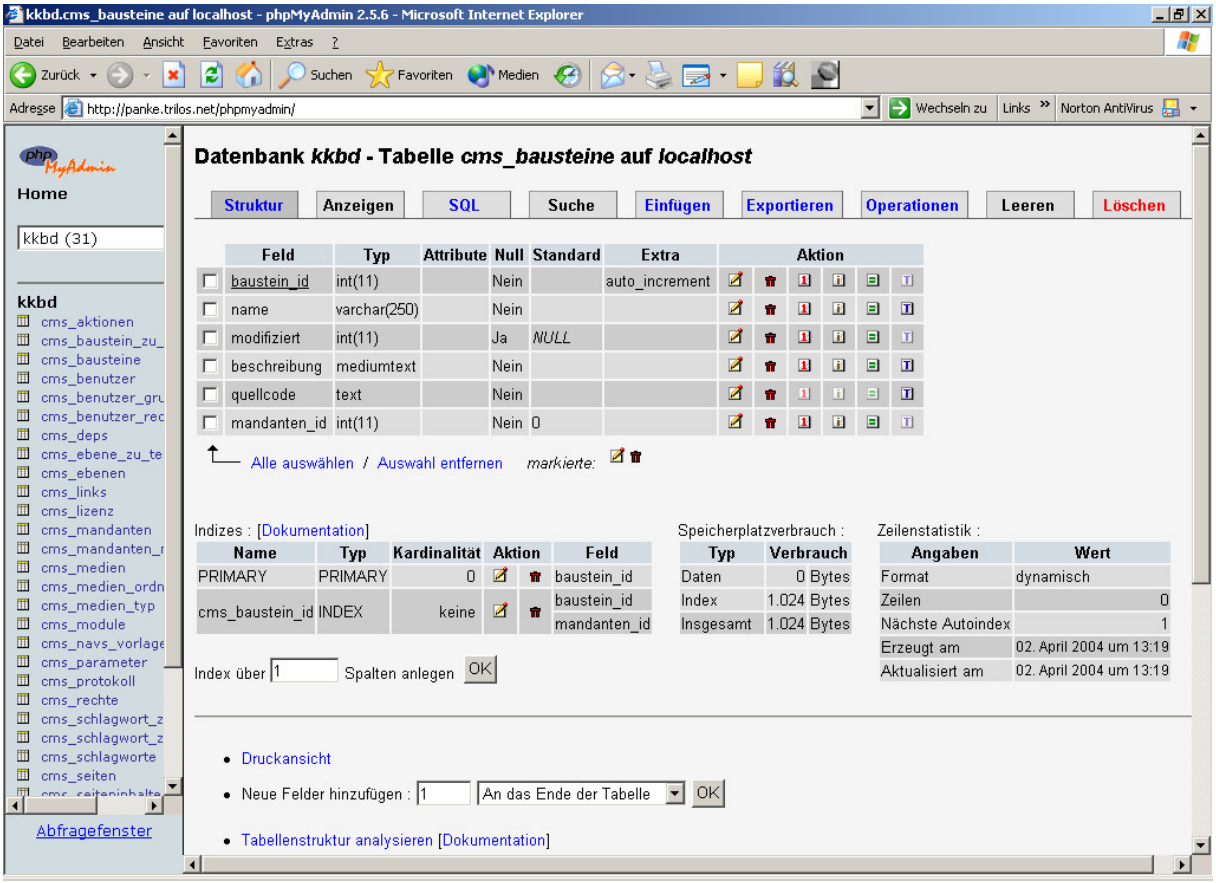


Abbildung 3 Detailansicht einer Tabelle

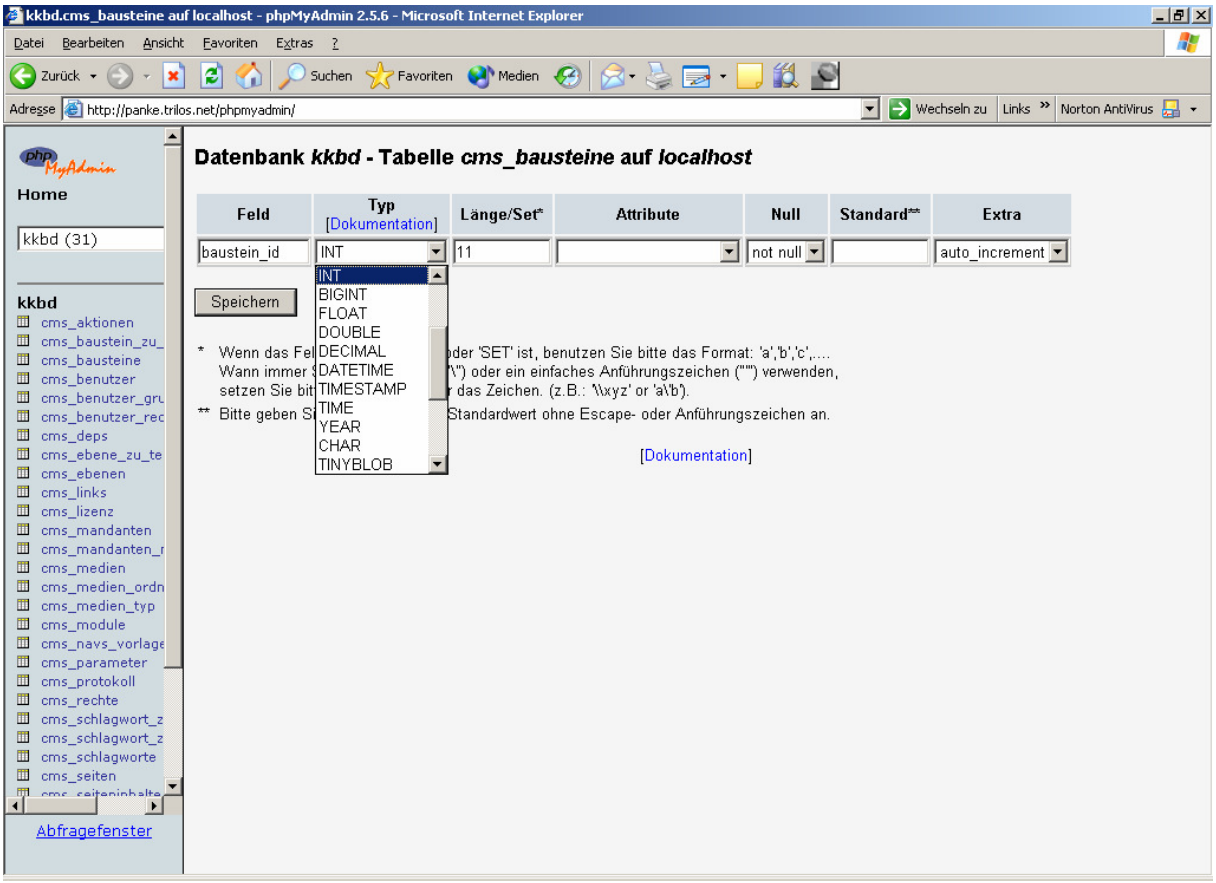


Abbildung 4 Bearbeiten einer Spalte

8.2 DB Designer4

Der DBDesigner von FabForce ist eine Anwendung für Windows und Linux. DBDesigner ist ein visuelles Datenbankmodellierungstool. Es kann außer mit MySQL auch mit ODBC, Oracle und MSSQL Datenbanken reden. Desweiteren bietet es die Möglichkeit zum Reverse Engeniering und Synchronisation mit Datenbanken.

DBDesigner ist meiner Meinung nach ein ganz heißer Tipp. Unbedingt ansehen! Ist seltsamerweise kaum bekannt.

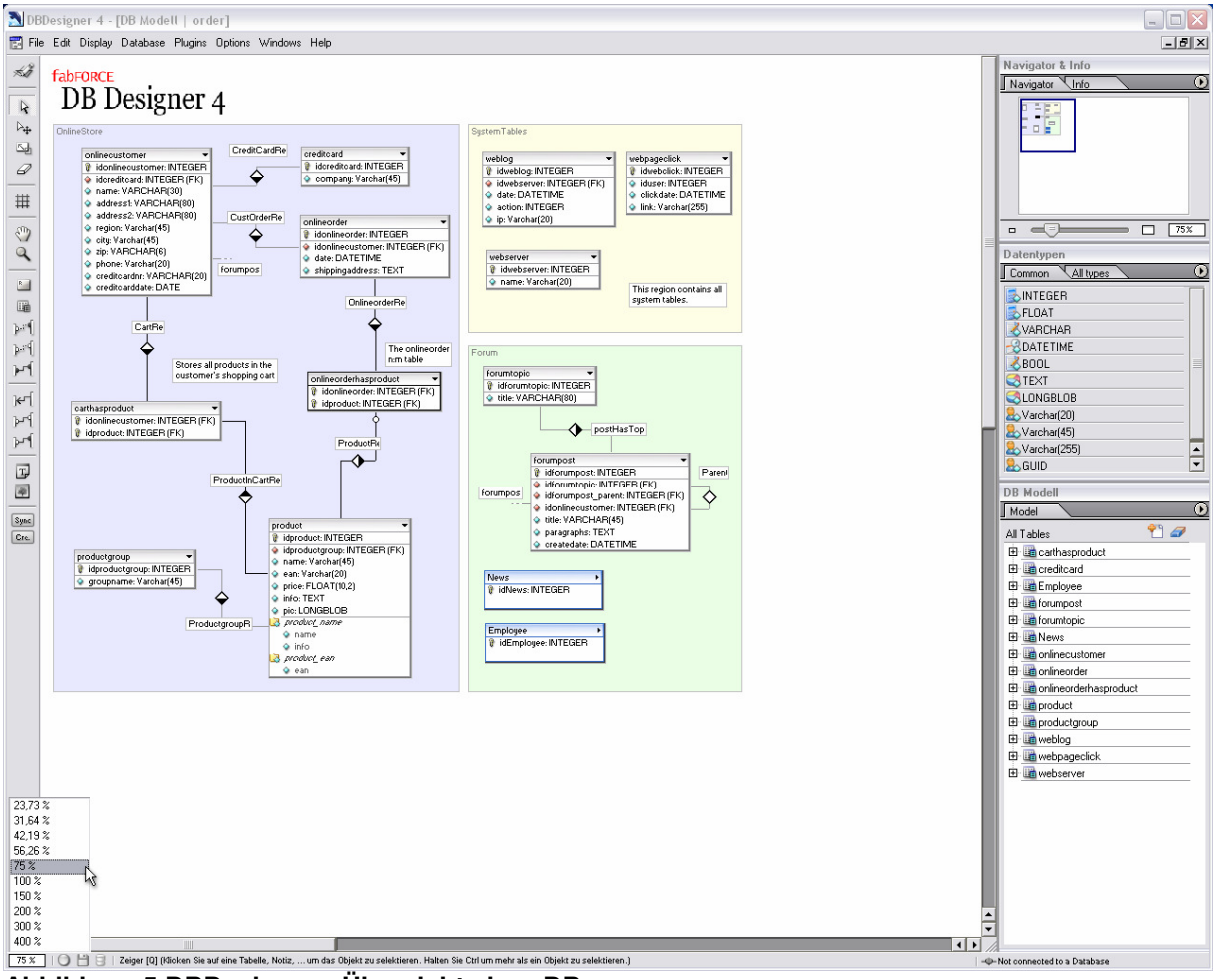


Abbildung 5 DBDesigner - Übersicht einer DB

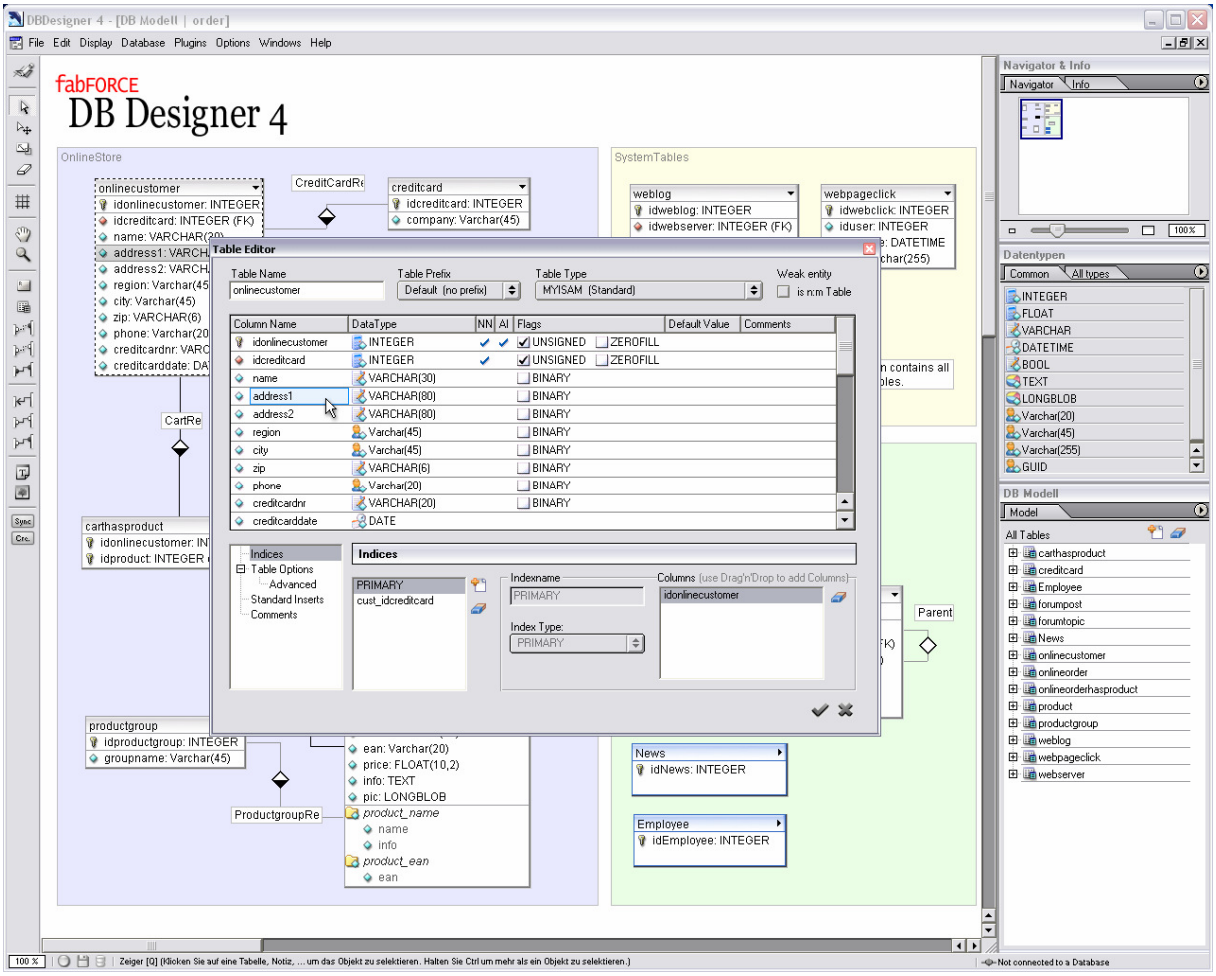
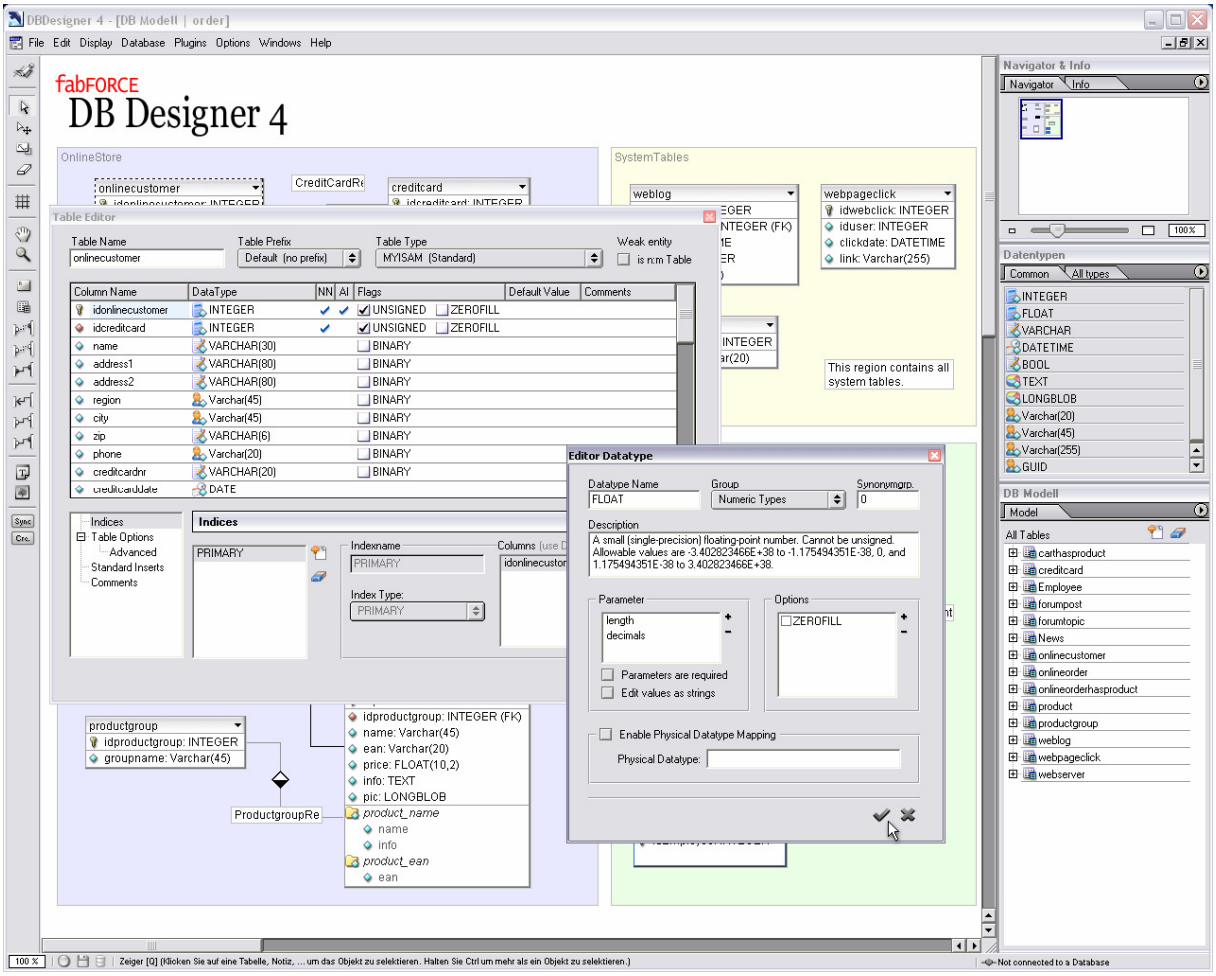
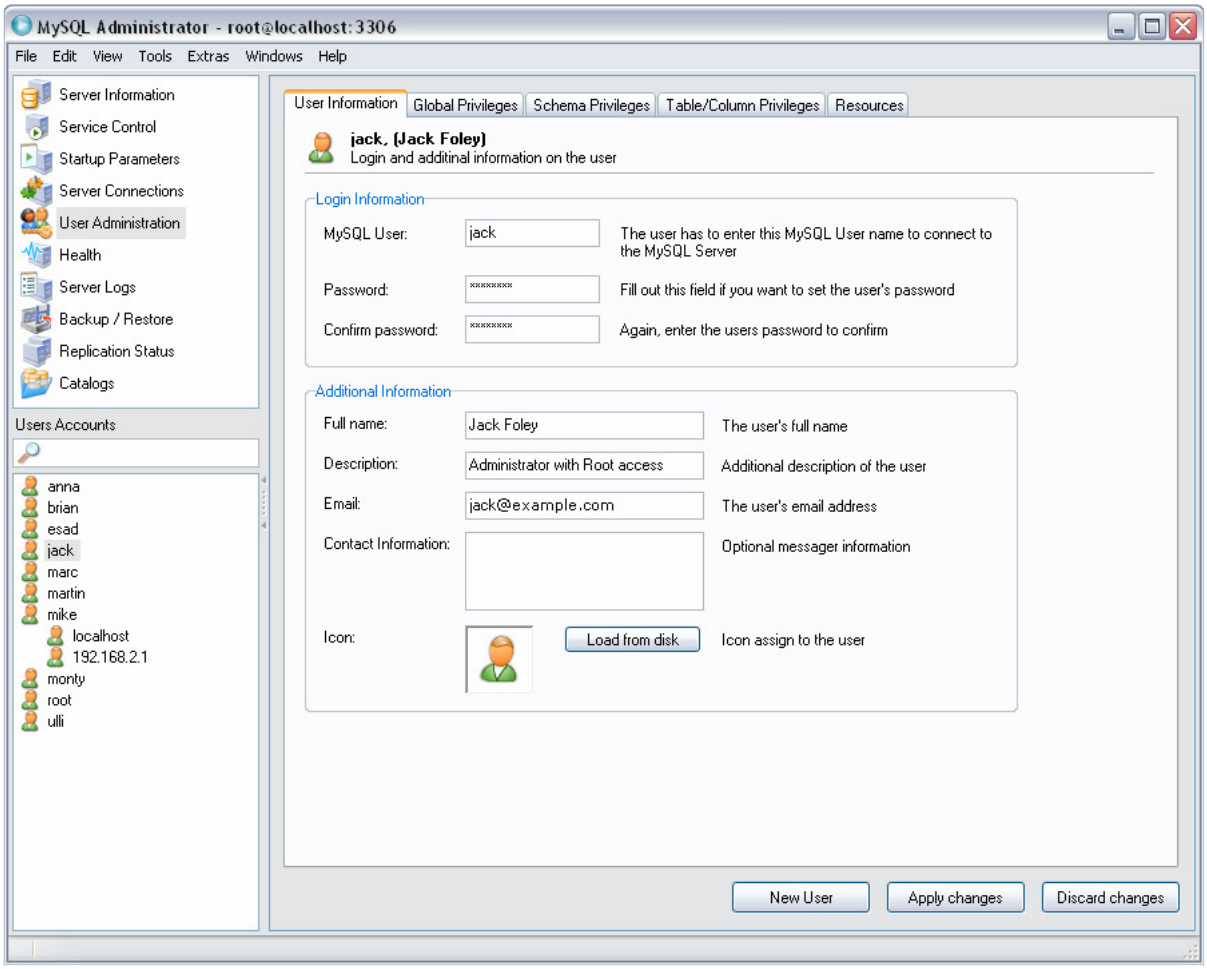


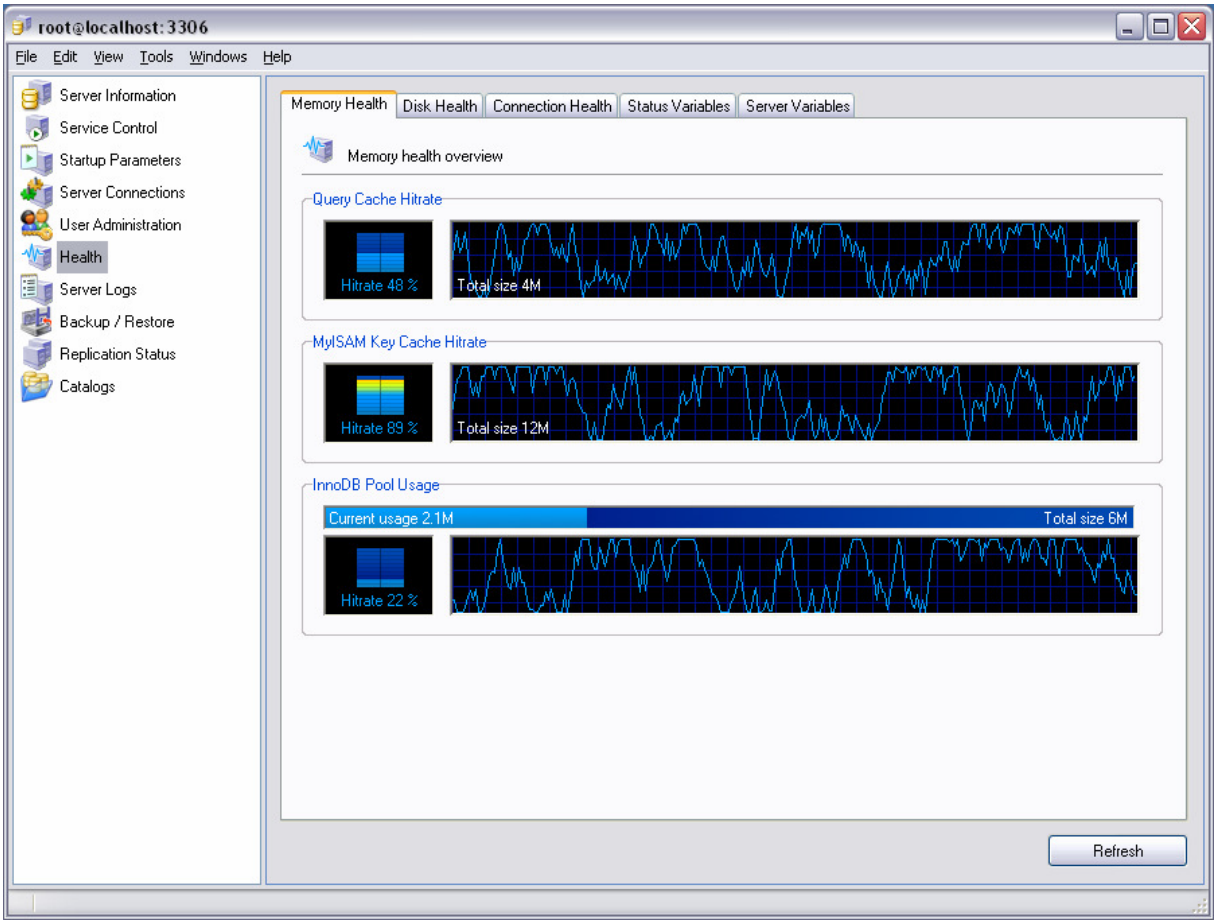
Abbildung 6 Bearbeiten einer Tabelle

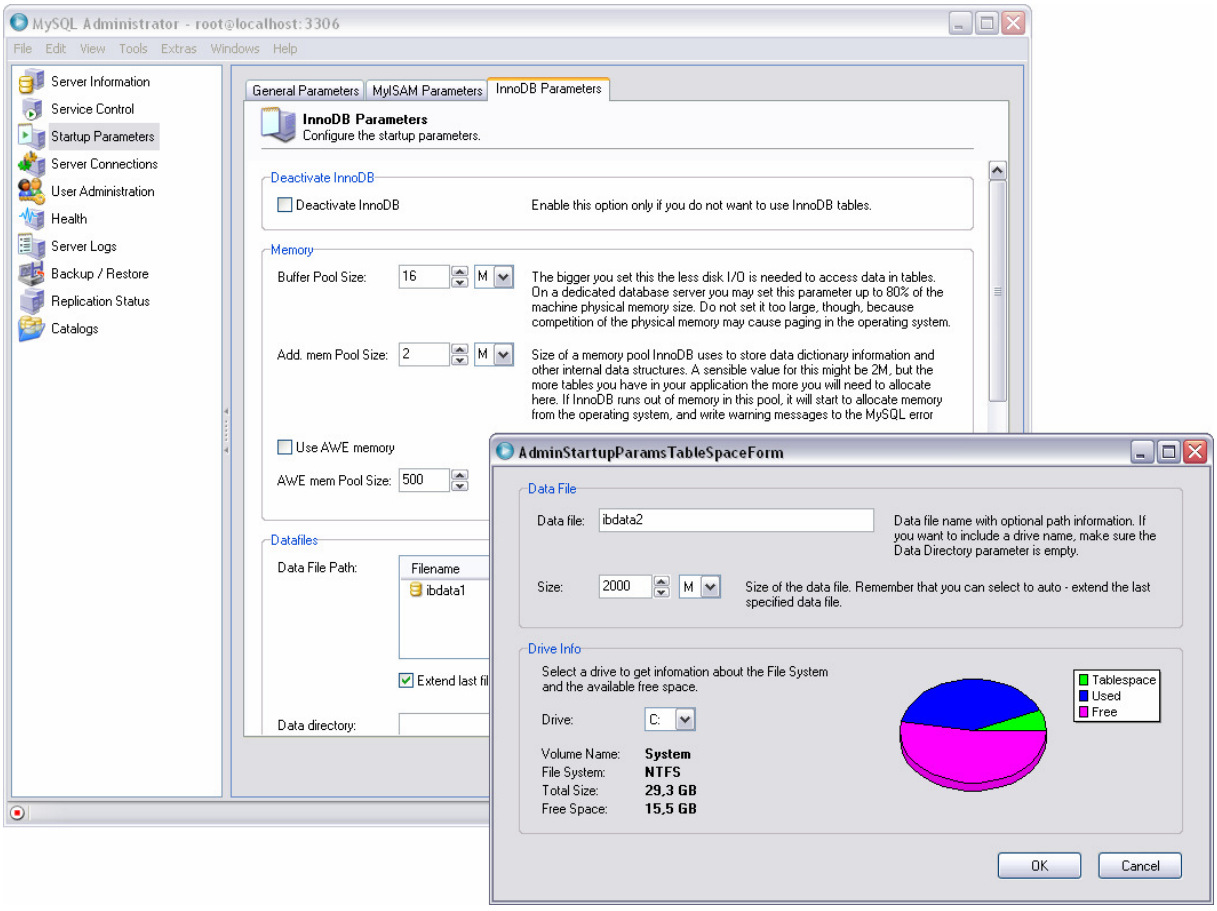

DBDesigner 4 - [DB Modell | order]
File Edit Display Database Plugins Options Windows Help
fabFORCE
DB Designer 4
OnlineStore
SystemTables
Table Editor
Table Name
Table Prefix
Table Type
Weak entity
Column Name
DataType
Flags
Default Value
Comments
Indices
Table Options
Advanced
Standard Inserts
Comments
Editor Datatype
Datatype Name
Group
Synonymgrp.
Description
Parameter
Options
ZEROFILL
Parameters are required
Edit values as strings
Enable Physical Datatype Mapping
Physical Datatype:
This region contains all system tables.
Navigator & Info
Datentypen
DB Modell
All Tables
Not connected to a Database

8.3 MYSQL Admin

Der MySQLAdmin ist ein grafisches Verwaltungstool für Windows und Linux direkt vom MySQL Hersteller. Alle Verwaltungsaufgaben lassen sich sehr einfach über ein GUI erledigen.




root@localhost:3306
File Edit View Tools Windows Help
Server Information
Service Control
Startup Parameters
Server Connections
User Administration
Health
Server Logs
Backup / Restore
Replication Status
Catalogs
Memory Health
Disk Health
Connection Health
Status Variables
Server Variables
Memory health overview
Query Cache Hitrate
Hitrate 48 %
Total size 4M
MyISAM Key Cache Hitrate
Hitrate 89 %
Total size 12M
InnoDB Pool Usage
Current usage 2.1M
Total size 6M
Hitrate 22 %
Refresh


MySQL Administrator - root@localhost:3306
File Edit View Tools Extras Windows Help
Server Information
Service Control
Startup Parameters
Server Connections
User Administration
Health
Server Logs
Backup / Restore
Replication Status
Catalogs
General Parameters
MyISAM Parameters
InnoDB Parameters
InnoDB Parameters
Configure the startup parameters.
Deactivate InnoDB
Deactivate InnoDB
Enable this option only if you do not want to use InnoDB tables.
Memory
Buffer Pool Size:
16
M
The bigger you set this the less disk I/O is needed to access data in tables. On a dedicated database server you may set this parameter up to 80% of the machine physical memory size. Do not set it too large, though, because competition of the physical memory may cause paging in the operating system.
Add. mem Pool Size:
2
M
Size of a memory pool InnoDB uses to store data dictionary information and other internal data structures. A sensible value for this might be 2M, but the more tables you have in your application the more you will need to allocate here. If InnoDB runs out of memory in this pool, it will start to allocate memory from the operating system, and write warning messages to the MySQL error
Use AWE memory
AWE mem Pool Size:
500
Datafiles
Data File Path:
Filename
ibdata1
Extend last fil
Data directory:
AdminStartupParamsTableSpaceForm
Data File
Data file:
ibdata2
Data file name with optional path information. If you want to include a drive name, make sure the Data Directory parameter is empty.
Size:
2000
M
Size of the data file. Remember that you can select to auto - extend the last specified data file.
Drive Info
Select a drive to get infomation about the File System and the available free space.
Drive:
C:
Volume Name: System
File System: NTFS
Total Size: 29,3 GB
Free Space: 15,5 GB
Tablespace
Used
Free
OK
Cancel

9 Anhang

9.1 Datentypen

Aus dem MySQL-Handbuch:

9.1.1 Numerische Datentypen

9.1.1.1 TINYINT[(M)] [UNSIGNED] [ZEROFILL]

Eine sehr kleine Ganzzahl. Der vorzeichenbehaftete Bereich ist -128 bis 127. Der vorzeichenlose Bereich ist 0 to 255.

9.1.1.2 SMALLINT[(M)] [UNSIGNED] [ZEROFILL]

Eine kleine Ganzzahl. Der vorzeichenbehaftete Bereich ist -32768 bis 32767. Der vorzeichenlose Bereich ist 0 bis 65535.

9.1.1.3 MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]

Eine Ganzzahl mittlerer Größe. Der vorzeichenbehaftete Bereich ist -8388608 bis 8388607. Der vorzeichenlose Bereich ist 0 bis 16777215.

9.1.1.4 INT[(M)] [UNSIGNED] [ZEROFILL]

Eine Ganzzahl normaler Größe. Der vorzeichenbehaftete Bereich ist -2147483648 bis 2147483647. Der vorzeichenlose Bereich ist 0 bis 4294967295.

9.1.1.5 INTEGER[(M)] [UNSIGNED] [ZEROFILL]

Ein Synonym für INT.

9.1.1.6 BIGINT[(M)] [UNSIGNED] [ZEROFILL]

Eine große Ganzzahl. Der vorzeichenbehaftete Bereich ist -9223372036854775808 bis 9223372036854775807. Der vorzeichenlose Bereich ist 0 bis 18446744073709551615. Einige Dinge solltest Du bei BIGINT-Spalten im Hinterkopf haben:

- Weil alle arithmetischen Berechnungen mit vorzeichenbehafteten BIGINT- oder DOUBLE-Werten durchgeführt werden, solltest Du keine vorzeichenlosen Ganzzahlen größer als 9223372036854775807 (63 Bits) benutzen, außer bei Bit-Funktionen! Wenn doch, können einige der letzten Ziffern im Ergebnis falsch sein, weil Rundungsfehler beim Umwandeln von BIGINT in DOUBLE auftreten. MySQL 4.0 kann BIGINT in folgenden Fällen handhaben:
  - Benutze Ganzzahlen, um große vorzeichenlose Wert in einer BIGINT-Spalte zu speichern.
  - Bei MIN(große_ganzzahl_spalte) und MAX(große_ganzzahl_spalte).
  - Bei der Benutzung der Operatoren (+, -, * usw.), wenn beide Operanden Ganzzahlen sind.
- Du kannst immer einen genauen Ganzzahlwert in einer BIGINT-Spalte speichern, wenn Du sie als Zeichenkette speichern, denn in diesem Fall wird dieser nicht zwischendurch als Double dargestellt.
- `-', `+' und `*' benutzen arithmetische BIGINT-Berechnungen, wenn beide Argumente INTEGER-Werte sind! Das heißt, wenn Du zwei Ganzzahlen multiplizierst (oder Ergebnisse von Funktionen, die Ganzzahlen

zurückgeben), erhältst Du vielleicht falsche Ergebnisse, wenn das Ergebnis größer als 9223372036854775807 ist.

9.1.1.7 FLOAT(genauigkeit) [ZEROFILL]

Eine Fließkommazahl. Kann nicht vorzeichenlos sein. genauigkeit ist <=24 bei einer Fließkommazahl einfacher Genauigkeit und zwischen 25 und 53 bei einer Fließkommazahl doppelter Genauigkeit. Diese Typen sind wie die unten beschriebenen FLOAT und DOUBLE-Typen. FLOAT(X) hat denselben Wertebereich wie die entsprechenden FLOAT- und DOUBLE-Typen, jedoch ist die Anzeigebreite und die Anzahl der Dezimalstellen undefiniert.

9.1.1.8 FLOAT[(M,D)] [ZEROFILL]

Eine kleine Fließkommazahl (einfacher Genauigkeit). Kann nicht vorzeichenlos sein. Der Wertebereich umfasst -3.402823466E+38 bis -1.175494351E-38, 0 und 1.175494351E-38 bis 3.402823466E+38. M ist die Anzeigebreite und D ist die Anzahl von Dezimalstellen. FLOAT ohne Argument oder mit einem Argument <= 24 steht für eine Fließkommazahl einfacher Genauigkeit.

9.1.1.9 DOUBLE[(M,D)] [ZEROFILL]

Eine normal große Fließkommazahl (doppelter Genauigkeit). Kann nicht vorzeichenlos sein. Der Wertebereich umfasst -1.7976931348623157E+308 bis -2.2250738585072014E-308, 0 und 2.2250738585072014E-308 bis 1.7976931348623157E+308. M ist die Anzeigebreite und D ist die Anzahl von Dezimalstellen. DOUBLE ohne Argument oder FLOAT(X) mit 25 <= X <= 53 steht für eine Fließkommazahl doppelter Genauigkeit.

9.1.1.10 DOUBLE PRECISION[(M,D)] [ZEROFILL]

Synonyme für DOUBLE.

9.1.1.11 REAL[(M,D)] [ZEROFILL]

Synonyme für DOUBLE.

9.1.1.12 DECIMAL[(M[,D])] [ZEROFILL]

Eine unkomprimierte Fließkommazahl. Kann nicht vorzeichenlos sein. Verhält sich wie eine CHAR-Spalte: ``Unkomprimiert'' bedeutet, dass die Zahl als Zeichenkette gespeichert wird, wobei ein Zeichen für jede Ziffer des Wertes steht. Der Dezimalpunkt und, bei negativen Zahlen, das `-'-Zeichen, werden in M nicht mitgezählt (aber hierfür wird Platz reserviert). Wenn D 0 ist, haben Werte keinen Dezimalpunkt oder Bruchteil. Der maximale Wertebereich von DECIMAL-Werte ist derselbe wie für DOUBLE, aber der tatsächliche Wertebereich einer gegebenen DECIMAL-Spalte kann durch die Auswahl von M und D eingeschränkt sein. Wenn D weggelassen wird, wird es auf 0 gesetzt. Wenn M ausgelassen wird, wird es auf 10 gesetzt.

9.1.1.13 NUMERIC(M,D) [ZEROFILL]

Synonym für DECIMAL.

9.1.2 String Datentypen

9.1.2.1 [NATIONAL] CHAR(M) [BINARY]

Eine Zeichenkette fester Länge, die beim Speichern rechts stets mit Leerzeichen bis zur angegebenen Länge aufgefüllt wird. Der Wertebereich von M ist 1 bis 255 Zeichen. Leerzeichen am Ende werden beim Abruf des Wertes entfernt. CHAR-Werte werden nach dem vorgabemäßigen Zeichensatz ohne Berücksichtigung der Groß-/Kleinschreibung sortiert und verglichen, es sei denn, dass Schlüsselwort BINARY wird angegeben. NATIONAL CHAR (Kurzform NCHAR) ist die Art, wie ANSI-SQL bei einer CHAR-Spalte festlegt, dass der vorgabemäßige Zeichensatz verwendet werden soll. Das ist der Vorgabewert in MySQL. CHAR ist eine Abkürzung für CHARACTER. MySQL erlaubt das Anlegen einer Spalte des Typs CHAR(0). Das ist hauptsächlich nützlich, wenn Sie mit alten Applikationen kompatibel sein müssen, die auf die Existenz einer Spalte vertrauen, den Wert aber nicht tatsächlich benutzen. Es ist ebenfalls nett, um eine Spalte anzulegen, die nur 2 Werte annehmen kann: Eine CHAR(0), die nicht als NOT NULL definiert ist, belegt nur 1 Bit und kann 2 Werte annehmen: NULL oder "".

9.1.2.2 [NATIONAL] VARCHAR(M) [BINARY]

Eine Zeichenkette variabler Länge. **HINWEIS:** Leerzeichen am Ende werden bei der Speicherung des Wertes entfernt (das unterscheidet den Typ von der ANSI-SQL-Spezifikation). Der Wertebereich von M ist 1 bis 255 Zeichen. VARCHAR-Werte werden nach dem vorgabemäßigen Zeichensatz ohne Berücksichtigung der Groß-/Kleinschreibung sortiert und verglichen, es sei denn, dass Schlüsselwort BINARY wird angegeben.

9.1.2.3 TINYBLOB | TINYTEXT

Eine BLOB- oder TEXT-Spalte mit einer maximalen Länge von 255 (2^8 - 1) Zeichen.

9.1.2.4 BLOB | TEXT

Eine BLOB- oder TEXT-Spalte mit einer maximalen Länge von 65535 (2^16 - 1) Zeichen.

9.1.2.5 MEDIUMBLOB | MEDIUMTEXT

Eine BLOB- oder TEXT-Spalte mit einer maximalen Länge von 16777215 (2^24 - 1) Zeichen.

9.1.2.6 LONGBLOB | LONGTEXT

Eine BLOB- oder TEXT-Spalte mit einer maximalen Länge von 4294967295 (2^32 - 1) Zeichen. Beachte, dass Du nicht den gesamten Wertebereich dieses Typs benutzen kannst, weil das Client-Server-Protokoll und MyISAM-Tabellen momentan eine Beschränkungen auf 16 MB pro Kommunikationspaket / Tabellenzeile haben.

9.1.3 Datumstypen

9.1.3.1 DATE

Ein Datum. Der unterstützte Wertebereich ist '1000-01-01' bis '9999-12-31'. MySQL zeigt DATE-Werte im 'YYYY-MM-DD'-Format an, gestattet jedoch, DATE-Spalten Werte entweder als Zeichenketten oder als Zahlen zuzuweisen.

9.1.3.2 DATETIME

Eine Datums-/Zeit-Kombination. Der unterstützte Wertebereich ist '1000-01-01 00:00:00' bis '9999-12-31 23:59:59'. MySQL zeigt DATETIME-Werte im 'YYYY-MM-DD HH:MM:SS'-Format an, gestattet jedoch, DATETIME-Spalten Werte entweder als Zeichenketten oder als Zahlen zuzuweisen.

9.1.3.3 TIMESTAMP[(M)]

Ein Zeitstempel. Der Wertebereich ist '1970-01-01 00:00:00' bis irgendwann im Jahr 2037. MySQL zeigt TIMESTAMP-Werte im YYYYMMDDHHMMSS-, YYMMDDHHMMSS-, YYYYMMDD- oder YYMMDD-Format an, abhängig davon, ob M 14 (oder fehlend), 12, 8 oder 6 ist, gestattet aber, dass Du TIMESTAMP-Spalten Werte entweder als Zeichenketten oder als Zahlen zuweist. Eine TIMESTAMP-Spalte ist nützlich, um Datum und Zeit einer INSERT- oder UPDATE-Operation zu speichern, weil sie automatisch auf das Datum und die Zeit der jüngsten Operation gesetzt wird, wenn Du nicht selbst einen Wert zuweist. Du kannst sie auch auf das aktuelle Datum und die aktuelle Zeit setzen, indem Du einen NULL-Wert zuweist. Ein TIMESTAMP wird immer mit 4 Bytes gespeichert. Das M-Argument betrifft nur die Anzeige der TIMESTAMP-Spalte. Beachte, dass TIMESTAMP(X)-Spalten, bei denen X 8 oder 14 ist, als Zahlen interpretiert werden, während andere TIMESTAMP(X)-Spalten als Zeichenketten interpretiert werden.

9.1.3.4 TIME

Ein Zeit-Typ. Der Wertebereich ist '-838:59:59' bis '838:59:59'. MySQL zeigt TIME-Werte im 'HH:MM:SS'-Format an, gestattet aber, TIME-Spalten Werte entweder als Zeichenketten oder als Zahlen zuweisen.

9.1.3.5 YEAR[(2|4)]

Ein Jahr in 2- oder 4-Ziffernformat (Vorgabe ist 4-Ziffern). Die zulässigen Werte reichen von 1901 bis 2155 sowie 0000 im 4-Ziffern-Jahresformat, und von 1970 bis 2069 beim 2-Ziffernformat (70 bis 69). MySQL zeigt YEAR-Werte im YYYY-Format an, gestattet aber, YEAR-Spalten Werte entweder als Zeichenketten oder als Zahlen zuweisen.

9.1.4 Komplexe Datentypen

9.1.4.1 ENUM('wert1','wert2',...)

Eine Aufzählung. Ein Zeichenkettenobjekt, das nur einen Wert haben kann, der aus den Auflistungswerten 'wert1', 'wert2', ..., NULL oder dem speziellen ""-Fehlerwert ausgewählt wird. Eine ENUM kann maximal 65535 unterschiedliche Werte haben.

9.1.4.2 SET('wert1','wert2',...)

Eine Reihe. Ein Zeichenkettenobjekt, das 0 oder mehr Werte haben kann, von denen jeder aus den Auflistungswerten 'wert1', 'wert2', ... ausgewählt werden muss. Eine SET kann maximal 64 Elemente haben.

9.2 Funktionen

Aus dem MySQL-Handbuch:

9.2.1 Nicht typenspezifische Operatoren und Funktionen

9.2.1.1 Klammer

```
( ... )
```

Benutze Klammern, um die Reihenfolge der Auswertung in einem Ausdruck zu erzwingen. Beispiel:

```
mysql> select 1+2*3;
        -> 7
mysql> select (1+2)*3;
        -> 9
```

9.2.1.2 Vergleichsoperatoren

Vergleichsoperationen ergeben einen Wert von 1 (TRUE), 0 (FALSE) oder NULL. Diese Funktionen funktionieren sowohl bei Zahlen als auch bei Zeichenketten. Zeichenketten werden bei Bedarf automatisch in Zahlen und Zahlen in Zeichenketten umgewandelt (wie in Perl oder PHP).

MySQL führt Vergleiche nach folgenden Regeln durch:

- Wenn ein oder beide Argumente NULL sind, ist das Ergebnis des Vergleichs NULL, außer beim <=> Operator.
- Wenn beide Argumente in einer Vergleichsoperation Zeichenketten sind, werden sie als Zeichenketten verglichen.
- Wenn beide Argumente Ganzzahlen sind, werden sie als Ganzzahlen verglichen.
- Hexadezimale Werte werden als binäre Zeichenketten behandelt, wenn sie nicht mit einer Zahl verglichen werden.
- Wenn eins der Argumente eine TIMESTAMP- oder DATETIME-Spalte ist und das andere Argument eine Konstante, wird die Konstante in einen Zeitstempel umgewandelt, bevor der Vergleich durchgeführt wird.
- In allen anderen Fällen werden die Argumente als Fließkommazahlen verglichen.

Vorgabemäßig werden Zeichenketten-Vergleiche unabhängig von der verwendeten Groß-/Kleinschreibung durchgeführt, indem der aktuelle Zeichensatz benutzt wird (vorgabemäßig ISO-8859-1 Latin1, der auch für englisch exzellent funktioniert).

Die unten stehenden Beispiele erläutern die Umwandlung von Zeichenketten in Zahlen für Vergleichsoperationen:

```
mysql> SELECT 1 > '6x';
        -> 0
mysql> SELECT 7 > '6x';
        -> 1
mysql> SELECT 0 > 'x6';
        -> 0
mysql> SELECT 0 = 'x6';
        -> 1
```

9.2.1.3 =

Gleich:

```
mysql> select 1 = 0;
        -> 0
mysql> select '0' = 0;
        -> 1
mysql> select '0.0' = 0;
        -> 1
mysql> select '0.01' = 0;
        -> 0
mysql> select '.01' = 0.01;
        -> 1
```

9.2.1.4 <> , !=

Ungleich:

```
mysql> select '.01' <> '0.01';
        -> 1
mysql> select .01 <> '0.01';
        -> 0
mysql> select 'zapp' <> 'zappp';
        -> 1
```

9.2.1.5 <=

Kleiner oder gleich:

```
mysql> select 0.1 <= 2;
        -> 1
```

9.2.1.6 <

Kleiner als:

```
mysql> select 2 < 2;
        -> 0
```

9.2.1.7 >=

Größer oder gleich:

```
mysql> select 2 >= 2;
        -> 1
```

9.2.1.8 >

Größer als:

```
mysql> select 2 > 2;
        -> 0
```

9.2.1.9 <=>

Null-sicheres gleich:

```
mysql> select 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
        -> 1 1 0
```

9.2.1.10 IS NULL , IS NOT NULL

Testet, ob eine Wert NULL ist oder nicht:

```
mysql> select 1 IS NULL, 0 IS NULL, NULL IS NULL;
        -> 0 0 1
mysql> select 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
        -> 1 1 0
```

9.2.1.11 ausdruck BETWEEN min AND max

Wenn ausdruck größer oder gleich min ist und ausdruck kleiner oder gleich max ist, gibt BETWEEN 1 zurück, andernfalls 0. Das ist äquivalent zum Ausdruck (min

<= ausdruck AND ausdruck <= max), wenn alle Argumente vom selben Typ sind. Das erste Argument (ausdruck) legt fest, wie der Vergleich durchgeführt wird:

- Wenn ausdruck eine TIMESTAMP-, DATE- oder DATETIME-Spalte ist, werden MIN() und MAX() im selben Format formatiert als wären sie Konstanten.
- Wenn ausdruck ein Zeichenketten-Ausdruck ohne Berücksichtigung der Groß-/Kleinschreibung ist, wird ein Zeichenkettenvergleich ohne Berücksichtigung der Groß-/Kleinschreibung durchgeführt.
- Wenn ausdruck ein Zeichenketten-Ausdruck mit Berücksichtigung der Groß-/Kleinschreibung ist, wird ein Zeichenkettenvergleich mit Berücksichtigung der Groß-/Kleinschreibung durchgeführt.
- Wenn ausdruck ist ein Ganzzahl-Ausdruck ist, wird ein Ganzzahlvergleich durchgeführt.
- Ansonsten wird ein Fließkommazahlenvergleich durchgeführt.

```
mysql> select 1 BETWEEN 2 AND 3;
        -> 0
mysql> select 'b' BETWEEN 'a' AND 'c';
        -> 1
mysql> select 2 BETWEEN 2 AND '3';
        -> 1
mysql> select 2 BETWEEN 2 AND 'x-3';
        -> 0
```

9.2.1.12 ausdruck IN (wert,...)

Gibt 1 zurück, wenn ausdruck einen Wert hat, der in der IN-Liste enthalten ist, ansonsten 0. Wenn alle Werte Konstanten sind, werden alle Werte gemäß dem Typ von ausdruck ausgewertet und sortiert. Danach wird ein Element mittels binärer Suche gesucht. Das heißt, dass IN sehr schnell ist, wenn die IN-Werteliste ausschließlich aus Konstanten besteht. Wenn ausdruck ein Zeichenketten-Ausdruck mit Berücksichtigung der Groß-/Kleinschreibung ist, wird der Zeichenkettenvergleich unter Berücksichtigung der Groß-/Kleinschreibung durchgeführt:

```
mysql> select 2 IN (0,3,5,'wefwf');
        -> 0
mysql> select 'wefwf' IN (0,3,5,'wefwf');
        -> 1
```

9.2.1.13 ausdruck NOT IN (wert,...)

Dasselbe wie NOT (ausdruck IN (wert,...)).

9.2.1.14 ISNULL(ausdruck)

Wenn ausdruck NULL ist, gibt ISNULL() 1 zurück, ansonsten 0:

```
mysql> select ISNULL(1+1);
        -> 0
mysql> select ISNULL(1/0);
        -> 1
```

Beachten Sie, dass ein Vergleich von NULL-Werten mit = immer UNWAHR ergibt!

9.2.1.15 COALESCE(liste)

Gibt das erste Nicht-NULL-Element in der Liste zurück:

```
mysql> select COALESCE(NULL,1);
```

```
        -> 1
mysql> select COALESCE(NULL,NULL,NULL);
        -> NULL
```

9.2.1.16 INTERVAL(N,N1,N2,N3,...)

Gibt 0 zurück, wenn N < N1, 1, wenn N < N2 usw. Alle Argumente werden als Ganzzahlen behandelt. Es ist erforderlich, dass N1 < N2 < N3 < ... < Nn ist, damit diese Funktion korrekt funktioniert. Das liegt daran, dass eine (sehr schnelle) binäre Suche benutzt wird:

```
mysql> select INTERVAL(23, 1, 15, 17, 30, 44, 200);
        -> 3
mysql> select INTERVAL(10, 1, 10, 100, 1000);
        -> 2
mysql> select INTERVAL(22, 23, 30, 44, 200);
        -> 0
```

Wenn Sie eine Zeichenkette, die Groß-/Kleinschreibung nicht berücksichtigt, mit einem der Standard-Operatoren vergleichen (=, <>..., aber nicht LIKE), werden Leerzeichen am Ende ignoriert:

```
mysql> select "a" ="A ";
        -> 1
```

9.2.1.17 Logische Operatoren

Alle logischen Funktionen geben 1 (TRUE), 0 (FALSE) oder NULL (unbekannt, was in den meisten Fällen dasselbe wie FALSE ist) zurück:

9.2.1.18 NOT , !

Logisch NOT. Gibt 1 zurück, wenn das Argument 0 ist, ansonsten 0. Ausnahme: NOT NULL gibt NULL zurück:

```
mysql> select NOT 1;
        -> 0
mysql> select NOT NULL;
        -> NULL
mysql> select ! (1+1);
        -> 0
mysql> select ! 1+1;
        -> 1
```

Das letzte Beispiel gibt 1 zurück, weil der Ausdruck auf dieselbe Art ausgewertet wird wie (!1)+1.

9.2.1.19 OR , ||

Logisch OR. Gibt 1 zurück, wenn eins der Argumente nicht 0 und nicht NULL ist:

```
mysql> select 1 || 0;
        -> 1
mysql> select 0 || 0;
        -> 0
mysql> select 1 || NULL;
        -> 1
```

9.2.1.20 AND , &&

Logisch AND. Gibt 0 zurück, wenn eins der Argumente 0 oder NULL ist, ansonsten 1:

```
mysql> select 1 && NULL;
        -> 0
mysql> select 1 && 0;
        -> 0
```

9.2.1.21 Ablaufsteuerungsfunktionen

9.2.1.22 IFNULL(ausdruck1,ausdruck2)

Wenn ausdruck1 nicht NULL ist, gibt IFNULL() ausdruck1 zurück, ansonsten ausdruck2. IFNULL() gibt einen numerischen oder einen Zeichenketten-Wert zurück, je nachdem, in welchem Zusammenhang es benutzt wird:

```
mysql> select IFNULL(1,0);
        -> 1
mysql> select IFNULL(NULL,10);
        -> 10
mysql> select IFNULL(1/0,10);
        -> 10
mysql> select IFNULL(1/0,'ja');
        -> 'ja'
```

9.2.1.23 NULLIF(ausdruck1,ausdruck2)

Wenn ausdruck1 = ausdruck2 wahr ist, gibt die Funktion NULL zurück, ansonsten ausdruck1. Das ist dasselbe wie CASE WHEN x = y THEN NULL ELSE x END:

```
mysql> select NULLIF(1,1);
        -> NULL
mysql> select NULLIF(1,2);
        -> 1
```

Beachten Sie, dass ausdruck1 in MySQL zweimal ausgewertet wird, wenn die Argumente gleich sind.

9.2.1.24 IF(ausdruck1,ausdruck2,ausdruck3)

Wenn ausdruck1 TRUE ist (ausdruck1 <> 0 und ausdruck1 <> NULL), gibt IF() ausdruck2 zurück, ansonsten ausdruck3. IF() gibt einen numerischen oder einen Zeichenketten-Wert zurück, je nachdem, in welchem Zusammenhang es benutzt wird:

```
mysql> select IF(1>2,2,3);
        -> 3
mysql> select IF(1<2,'ja','nein');
        -> 'ja'
mysql> select IF(strcmp('test','test1'),'nein','ja');
        -> 'nein'
```

ausdruck1 wird als Ganzzahlwert ausgewertet, woraus folgt, dass Sie das Testen auf Fließkomma- oder Zeichenketten-Werte mit einer Vergleichsoperation durchführen sollten:

```
mysql> select IF(0.1,1,0);
        -> 0
mysql> select IF(0.1<>0,1,0);
        -> 1
```

Im ersten Fall gibt IF(0.1) 0 zurück, weil 0.1 in einen Ganzzahlwert umgewandelt wird, wodurch es auf IF(0) getestet wird. Das ist vielleicht nicht das, was Sie erwarten. Im zweiten Fall testet der Vergleich den Original-Fließkommawert, um zu sehen, ob er nicht 0 ist. Das Ergebnis des Vergleichs wird als Ganzzahl

benutzt. Der vorgabemäßige Rückgabewert von IF() (der eine Rolle spielen kann, wenn er in einer temporären Tabelle gespeichert wird), wird in MySQL-Version 3.23 wie folgt berechnet:

Ausdruck	Rückgabewert
ausdruck2 oder ausdruck3 gibt Zeichenkette zurück	Zeichenkette
ausdruck2 oder ausdruck3 gibt Fließkommawert zurück	Fließkommawert
ausdruck2 oder ausdruck3 gibt Ganzzahl zurück	Ganzzahl

9.2.1.25 CASE

CASE wert WHEN [vergleichs-wert] THEN ergebnis [WHEN [vergleichs-wert] THEN ergebnis ...] [ELSE ergebnis] END
CASE WHEN [bedingung] THEN ergebnis [WHEN [bedingung] THEN ergebnis ...] [ELSE ergebnis] END

Die erste Version gibt ergebnis zurück, wo wert=vergleichs-wert. Die zweite Version gibt das Ergebnis für die erste Bedingung zurück, die WAHR ist. Wenn es keinen übereinstimmenden Ergebniswert gab, wird das Ergebnis nach ELSE zurückgegeben. Wenn es keinen ELSE-Teil gibt, wird NULL zurückgegeben:

```
mysql> SELECT CASE 1 WHEN 1 THEN "eins" WHEN 2 THEN "zwei" ELSE
"mehr" END;
       -> "eins"
mysql> SELECT CASE WHEN 1>0 THEN "wahr" ELSE "unwahr" END;
       -> "wahr"
mysql> SELECT CASE BINARY "B" when "a" then 1 when "b" then 2 END;
       -> NULL
```

Der Typ des Rückgabewerts (INTEGER, DOUBLE oder STRING) ist derselbe wie der Typ des ersten zurückgegebenen Werts (der Ausdruck nach dem ersten THEN).

9.2.2 Zeichenketten-Funktionen

Funktionen für Zeichenkettenwerte geben NULL zurück, wenn die Länge des Ergebnisses größer wäre als der max_allowed_packet-Serverparameter. Bei Funktionen, die mit Zeichenkettenpositionen arbeiten, wird die erste Position als 1 gezählt.

9.2.2.1 ASCII(zeichenkette)

Gibt den ASCII-Code-Wert des äußersten linken Zeichens der Zeichenkette zeichenkette zurück. Gibt 0 zurück, wenn zeichenkette die leere Zeichenkette ist. Gibt NULL zurück, wenn zeichenkette NULL ist:

```
mysql> select ASCII('2');
    -> 50
mysql> select ASCII(2);
    -> 50
mysql> select ASCII('dx');
    -> 100
```

Siehe auch ORD()-Funktion.

9.2.2.2 ORD(zeichenkette)

Wenn das äußerste linke Zeichen der Zeichenkette zeichenkette ein Multi-Byte-Zeichen ist, gibt diese Funktion den Code des Multi-Byte-Zeichens zurück, indem der ASCII-Code-Wert des Zeichens in folgendem Format

zurückgegeben wird: ((erstes byte ASCII code)*256+(zweites byte ASCII code))[*256+drittes byte ASCII code...]. Wenn das äußerste linke Zeichen kein Multi-Byte-Zeichen ist, wird derselbe Wert wie bei der ASCII()-Funktion zurückgegeben:

```
mysql> select ORD('2');
    -> 50
```

9.2.2.3 CONV(N,von_basis,zu_basis)

Wandelt Zahlen zwischen verschiedenen Zahlsystemen um. Gibt eine Zeichenkettendarstellung der Zahl N zurück, umgewandelt von Basis von_basis zu Basis zu_basis. Gibt NULL zurück, wenn irgend ein Argument NULL ist. Das Argument N wird als Ganzzahl interpretiert, kann aber als Ganzzahl oder Zeichenkette angegeben werden. Die kleinste Basis ist 2 und die größte Basis 36. Wenn zu_basis eine negative Zahl ist, wird N als vorzeichenbehaftete Zahl betrachtet. Ansonsten wird N als vorzeichenlos behandelt. CONV arbeitet mit 64-Bit-Genauigkeit:

```
mysql> select CONV("a",16,2);
    -> '1010'
mysql> select CONV("6E",18,8);
    -> '172'
mysql> select CONV(-17,10,-18);
    -> '-H'
mysql> select CONV(10+"10"+'10'+0xa,10,10);
    -> '40'
```

9.2.2.4 BIN(N)

Gibt eine Zeichenkettendarstellung des Binärwerts von N zurück, wobei N eine BIGINT-Zahl ist. Das ist äquivalent zu CONV(N,10,2). Gibt NULL zurück, wenn N NULL ist:

```
mysql> select BIN(12);
    -> '1100'
```

9.2.2.5 OCT(N)

Gibt eine Zeichenkettendarstellung des Oktalwerts von N zurück, wobei N eine BIGINT-Zahl ist. Das ist äquivalent zu CONV(N,10,8). Gibt NULL zurück, wenn N NULL ist:

```
mysql> select OCT(12);
    -> '14'
```

9.2.2.6 HEX(N)

Gibt eine Zeichenkettendarstellung des hexadezimalen Werts von N zurück, wobei N eine BIGINT-Zahl ist. Das ist äquivalent zu CONV(N,10,16). Gibt NULL zurück, wenn N NULL ist:

```
mysql> select HEX(255);
    -> 'FF'
```

9.2.2.7 CHAR(N,...)

CHAR() interpretiert die Argumente als Ganzzahlen und gibt eine Zeichenkette zurück, die aus den Zeichen besteht, die durch die ASCII-Code-Werte dieser Ganzzahlen gegeben sind. NULL-Werte werden übersprungen:

```
mysql> select CHAR(77,121,83,81,'76');
    -> 'MySQL'
mysql> select CHAR(77,77.3,'77.3');
    -> 'MMM'
```

9.2.2.8 CONCAT(zeichenkette1,zeichenkette2,...)

Gibt die Zeichenkette zurück, die durch die Verkettung der Argumente entsteht. Gibt NULL zurück, wenn irgend ein Argument NULL ist. Kann mehr als 2 Argumente haben. Ein numerisches Argument wird in die äquivalente Zeichenkettenform umgewandelt:

```
mysql> select CONCAT('My', 'S', 'QL');
    -> 'MySQL'
mysql> select CONCAT('My', NULL, 'QL');
    -> NULL
mysql> select CONCAT(14.3);
    -> '14.3'
```

9.2.2.9 CONCAT_WS(trennzeichen, zeichenkette1, zeichenkette2,...)

CONCAT_WS() steht für CONCAT mit Trennzeichen und ist eine spezielle Form von CONCAT(). Das erste Argument ist das Trennzeichen für die restlichen Argumente. Das Trennzeichen kann eine Zeichenkette sein, so wie die übrigen Argumente. Wenn das Trennzeichen NULL ist, ist das Ergebnis NULL. Die Funktion überspringt jegliche NULLs und leere Zeichenketten nach dem Trennzeichen-Argument. Das Trennzeichen wird zwischen den zu verknüpfenden Zeichenketten hinzugefügt:

```
mysql> select CONCAT_WS(",","Vorname","Zweiter Vorname","Nachname");
    -> 'Vorname,Zweiter Vorname,Nachname'
mysql> select CONCAT_WS(",","Vorname",NULL,"Nachname");
    -> 'Vorname,Nachname'
```

9.2.2.10 LENGTH, OCTET_LENGTH, CHAR_LENGTH, CHARACTER_LENGTH

LENGTH(zeichenkette)
OCTET_LENGTH(zeichenkette)
CHAR_LENGTH(zeichenkette)
CHARACTER_LENGTH(zeichenkette)

Gibt die Länge der Zeichenkette zeichenkette an:

```
mysql> select LENGTH('text');
    -> 4
mysql> select OCTET_LENGTH('text');
    -> 4
```

Beachten Sie, dass bei CHAR_LENGTH() Multi-Byte-Zeichen nur einmal gezählt werden.

9.2.2.11 LOCATE, POSITION

LOCATE(teilzeichenfolge,zeichenkette)
POSITION(teilzeichenfolge IN zeichenkette)

Gibt die Position des ersten Auftretens der Teilzeichenfolge teilzeichenfolge in der Zeichenkette zeichenkette an. Gibt 0 zurück, wenn teilzeichenfolge nicht in zeichenkette enthalten ist:

```
mysql> select LOCATE('bar', 'foobarbar');
    -> 4
mysql> select LOCATE('xbar', 'foobar');
    -> 0
```

Diese Funktion ist Multi-Byte-sicher.

9.2.2.12 LOCATE(teilzeichenfolge,zeichenkette,position)

Gibt die Position des ersten Auftretens der Teilzeichenfolge teilzeichenfolge in der Zeichenkette zeichenkette ab Position position an. Gibt 0 zurück, wenn teilzeichenfolge nicht in zeichenkette enthalten ist:

```
mysql> select LOCATE('bar', 'foobarbar',5);
    -> 7
```
Diese Funktion ist Multi-Byte-sicher.

9.2.2.13 INSTR(zeichenkette,teilzeichenfolge)

Gibt die Position des ersten Auftretens der Teilzeichenfolge teilzeichenfolge in der Zeichenkette zeichenkette an. Das ist dasselbe wie LOCATE() mit zwei Argumenten, ausser dass die Argumente vertauscht sind:
```
mysql> select INSTR('foobarbar', 'bar');
    -> 4
mysql> select INSTR('xbar', 'foobar');
    -> 0
```
Diese Funktion ist Multi-Byte-sicher.

9.2.2.14 LPAD(zeichenkette,laenge,fuellzeichenkette)

Gibt die Zeichenkette zeichenkette zurück, links aufgefüllt mit der Zeichenkette fuellzeichenkette, bis zeichenkette laenge Zeichen lang ist. Wenn zeichenkette länger als laenge ist, wird sie auf laenge Zeichen verkürzt.
```
mysql> select LPAD('hi',4,'??');
    -> '??hi'
```

9.2.2.15 RPAD(zeichenkette,laenge,fuellzeichenkette)

Gibt die Zeichenkette zeichenkette zurück, rechts aufgefüllt mit der Zeichenkette fuellzeichenkette, bis zeichenkette laenge Zeichen lang ist. Wenn zeichenkette länger als laenge ist, wird sie auf laenge Zeichen verkürzt.
```
mysql> select RPAD('hi',5,'?');
    -> 'hi???'
```

9.2.2.16 LEFT(zeichenkette,laenge)

Gibt die äußersten linken laenge Zeichen der Zeichenkette zeichenkette zurück:
```
mysql> select LEFT('foobarbar', 5);
    -> 'fooba'
```
Diese Funktion ist Multi-Byte-sicher.

9.2.2.17 RIGHT(zeichenkette,laenge)

Gibt die äußersten rechten laenge Zeichen der Zeichenkette zeichenkette zurück:
```
mysql> select RIGHT('foobarbar', 4);
    -> 'rbar'
```
Diese Funktion ist Multi-Byte-sicher.

9.2.2.18 SUBSTRING, MID

SUBSTRING(zeichenkette,position,laenge)
SUBSTRING(zeichenkette FROM position FOR laenge)
MID(zeichenkette,position,laenge)

Gibt eine laenge Zeichen lange Teilzeichenfolge der Zeichenkette zeichenkette ab Position position zurück. Die abweichende Form, die FROM benutzt, ist ANSI-SQL92-Syntax:
```
mysql> select SUBSTRING('Heinzholger',5,6);
    -> 'zholge'
```
Diese Funktion ist Multi-Byte-sicher.

SUBSTRING(zeichenkette,position)
SUBSTRING(zeichenkette FROM position)

Gibt eine Teilzeichenfolge der Zeichenkette zeichenkette ab Position position zurück:

```
mysql> select SUBSTRING('Heinzholger',5);
    -> 'zholger'
mysql> select SUBSTRING('foobarbar' FROM 4);
    -> 'barbar'
```

Diese Funktion ist Multi-Byte-sicher.

SUBSTRING_INDEX(zeichenkette,begrenzer,zaehler)

Gibt die Teilzeichenfolge von Zeichenkette zeichenkette vor zaehler Vorkommen des Begrenzers begrenzer zurück. Wenn zaehler positiv ist, wird alle links vom letzten Begrenzer zurückgegeben (von links gezählt). Wenn zaehler negativ ist, wird alles rechts vom letzten Begrenzer (von rechts gezählt) zurückgegeben:

```
mysql> select SUBSTRING_INDEX('www.mysql.com', '.', 2);
    -> 'www.mysql'
mysql> select SUBSTRING_INDEX('www.mysql.com', '.', -2);
    -> 'mysql.com'
```

Diese Funktion ist Multi-Byte-sicher.

9.2.2.19 LTRIM(zeichenkette)

Gibt die Zeichenkette zeichenkette zurück, bei der führende Leerzeichen entfernt wurden:

```
mysql> select LTRIM('  barbar');
    -> 'barbar'
```

9.2.2.20 RTRIM(zeichenkette)

Gibt die Zeichenkette zeichenkette zurück, bei der Leerzeichen am Ende entfernt wurden:

```
mysql> select RTRIM('barbar   ');
    -> 'barbar'
```

Diese Funktion ist Multi-Byte-sicher.

9.2.2.21 TRIM

TRIM([[BOTH | LEADING | TRAILING] [entfernzeichenkette] FROM] zeichenkette)

Gibt die Zeichenkette zeichenkette zurück, bei der alle entfernzeichenkette-Präfixe und / oder -Suffixe entfernt wurden. Wenn keiner der Spezifizierer BOTH, LEADING oder TRAILING angegeben wird, wird BOTH angenommen. Wenn entfernzeichenkette nicht angegeben ist, werden Leerzeichen entfernt:

```
mysql> select TRIM('  bar   ');
    -> 'bar'
mysql> select TRIM(LEADING 'x' FROM 'xxxbarxxx');
    -> 'barxxx'
mysql> select TRIM(BOTH 'x' FROM 'xxxbarxxx');
    -> 'bar'
mysql> select TRIM(TRAILING 'xyz' FROM 'barxxyz');
    -> 'barx'
```

Diese Funktion ist Multi-Byte-sicher.

9.2.2.22 SOUNDEX(zeichenkette)

Gibt eine Soundex-Zeichenkette von zeichenkette zurück. Zwei Zeichenketten, die fast gleich klingen, sollten identische Soundex-Zeichenketten haben. Eine Standard-Soundex-Zeichenkette ist 4 Zeichen lang, aber die SOUNDEX()-Funktion gibt eine beliebig lange Zeichenkette zurück. Sie können SUBSTRING() auf das Ergebnis anwenden, um eine Standard-Soundex-Zeichenkette zu erhalten. Alle nicht alphanumerischen Zeichen in der angegebenen Zeichenkette werden ignoriert. Alle internationalen alphabetischen Zeichen ausserhalb des Wertebereichs A bis Z werden als Vokale behandelt:

```
mysql> select SOUNDEX('Hello');
    -> 'H400'
mysql> select SOUNDEX('Quadratically');
    -> 'Q36324'
```

9.2.2.23 SPACE(N)

Gibt eine Zeichenkette zurück, die aus N Leerzeichen besteht:

```
mysql> select SPACE(6);
    -> '      '
```

9.2.2.24 REPLACE

Replace (zeichenkette,von_zeichenkette,zu_zeichenkette)

Gibt die Zeichenkette zeichenkette zurück, bei der alle Vorkommen der Zeichenkette von_zeichenkette durch die Zeichenkette zu_zeichenkette ersetzt wurden:

```
mysql> select REPLACE('www.mysql.com', 'w', 'Ww');
    -> 'WwWwWw.mysql.com'
```

Diese Funktion ist Multi-Byte-sicher.

9.2.2.25 REPEAT(zeichenkette,zaehler)

Gibt eine Zeichenkette zurück, die aus der Zeichenkette zeichenkette besteht, die zaehler mal wiederholt wurde. Wenn zaehler <= 0 ist, wird eine leere Zeichenkette zurückgegeben. Gibt NULL zurück, wenn zeichenkette oder zaehler NULL sind:

```
mysql> select REPEAT('MySQL', 3);
    -> 'MySQLMySQLMySQL'
```

9.2.2.26 REVERSE(zeichenkette)

Gibt die Zeichenkette zeichenkette in umgedrehter Reihenfolge der Zeichen zurück:

```
mysql> select REVERSE('abc');
    -> 'cba'
```

Diese Funktion ist Multi-Byte-sicher.

9.2.2.27 insert

INSERT(zeichenkette,position,laenge,neue_zeichenkette)

Gibt die Zeichenkette zeichenkette zurück, wobei eine Teilzeichenfolge ab Position position mit laenge Zeichen Länge durch die Zeichenkette neue_zeichenkette ersetzt wurde:

```
mysql> select INSERT('Heinzholger', 6, 4, 'DIET');
    -> 'HeinzDIETer'
```

Diese Funktion ist Multi-Byte-sicher.

9.2.2.28 ELT

ELT(N,zeichenkette1,zeichenkette2,zeichenkette3,...)

Gibt zeichenkette1 zurück, wenn N = 1 ist, zeichenkette2, wenn N = 2 ist usw.. Gibt NULL zurück, wenn N kleiner als 1 oder größer als die Anzahl von Argumenten ist. ELT() ist das Komplement von FIELD():

```
mysql> select ELT(1, 'ej', 'Heja', 'hej', 'foo');
    -> 'ej'
mysql> select ELT(4, 'ej', 'Heja', 'hej', 'foo');
        ➔  'foo'
```

9.2.2.29 FIELD

FIELD(zeichenkette,zeichenkette1,zeichenkette2,zeichenkette3,...)

Gibt den Index von zeichenkette in der Liste zeichenkette1, zeichenkette2, zeichenkette3, ... zurück. Gibt 0 zurück, wenn zeichenkette nicht gefunden wird. FIELD() ist das Komplement von ELT():

```
mysql> select FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
    -> 2
mysql> select FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');
        ➔ 0
```

9.2.2.30 FIND_IN_SET

FIND_IN_SET(zeichenkette,zeichenkettenliste)

Gibt einen Wert 1 bis N zurück, wenn die Zeichenkette zeichenkette in der Liste zeichenkettenliste ist, die aus N Teilzeichenfolgen besteht. Eine Zeichenkettenliste ist eine Zeichenkette, die aus Teilzeichenfolgen zusammen gesetzt ist, die durch `,'-Zeichen getrennt sind. Wenn das erste Argument eine Zeichenketten-Konstante ist und das zweite eine Spalte des Typs SET, wird die FIND_IN_SET()-Funktion optimiert, Bit-Arithmetik zu benutzen! Gibt 0 zurück, wenn zeichenkette nicht in zeichenkettenliste ist oder wenn zeichenkettenliste die leere Zeichenkette ist. Gibt NULL zurück, wenn eines oder beide Argumente NULL sind. Diese Funktion funktioniert nicht korrekt, wenn das erste Argument ein `,' enthält:

```
mysql> SELECT FIND_IN_SET('b','a,b,c,d');
    -> 2
```

9.2.2.31 MAKE_SET

MAKE_SET(bits,zeichenkette1,zeichenkette2,...)

Gibt einen Satz (eine Zeichenkette, die Teilzeichenfolgen enthält, die durch `,' getrennt sind) zurück, der aus Zeichenketten besteht, die das entsprechende Bit in bits gesetzt haben. zeichenkette1 entspricht Bit 0, zeichenkette2 Bit 1 usw. NULL-Zeichenketten in zeichenkette1, zeichenkette2 usw. werden nicht an das Ergebnis angehängt:

```
mysql> SELECT MAKE_SET(1,'a','b','c');
    -> 'a'
mysql> SELECT MAKE_SET(1 | 4,'hallo','liebe','welt');
    -> 'hallo,welt'
mysql> SELECT MAKE_SET(0,'a','b','c');
    -> ''
```

9.2.2.32 EXPORT_SET

EXPORT_SET(bits,an,aus,[trennzeichen,[anzahl_bits]])

Gibt eine Zeichenkette zurück, in der Sie für jedes bit, das in 'bit' gesetzt ist, eine 'an'-Zeichenkette erhalten, und für jedes zurückgesetzte Bit eine 'aus'-Zeichenkette. Jede Zeichenkette wird mit 'trennzeichen' getrennt (vorgabemäßig ','), und nur die 'anzahl_bits' (vorgabemäßig 64) von 'bits' wird benutzt:

```
mysql> select EXPORT_SET(5,'Y','N',',',4)
    -> Y,N,Y,N
```

9.2.2.33 LCASE, LOWER

LCASE(zeichenkette)
LOWER(zeichenkette)

Gibt die Zeichenkette zeichenkette zurück, bei der alle Zeichen in Kleinschreibung gemäß dem aktuellen Zeichensatz-Mapping (Vorgabe ist ISO-8859-1 Latin1) umgewandelt wurden:

```
mysql> select LCASE('HEINZholger');
    -> 'heinzholger'
```

Diese Funktion ist Multi-Byte-sicher.

9.2.2.34 UCASE, UPPER

UCASE(zeichenkette)
UPPER(zeichenkette)

Gibt die Zeichenkette zeichenkette zurück, bei der alle Zeichen in Großschreibung gemäß dem aktuellen Zeichensatz-Mapping (Vorgabe ist ISO-8859-1 Latin1) umgewandelt wurden:

```
mysql> select UCASE('Hej');
    -> 'HEJ'
```

Diese Funktion ist Multi-Byte-sicher.

9.2.2.35 LOAD_FILE(datei)

Liest die Datei datei und gibt den Dateiinhalt als Zeichenkette zurück. Die Datei muss auf dem Server sein, Sie müssen den vollen Pfadnamen zur Datei angeben und Sie müssen die **file**-Berechtigung besitzen. Die Datei muss von allen lesbar sein und kleiner als max_allowed_packet. Wenn die Datei nicht existiert oder aus den oben genannten Gründen nicht gelesen werden kann, gibt die Funktion NULL zurück:

```
mysql> UPDATE tabelle
       SET blob_spalte=LOAD_FILE("/tmp/bild")
       WHERE id=1;
```

.

MySQL konvertiert Zahlen bei Bedarf automatisch in Zeichenketten, und umgekehrt:

```
mysql> SELECT 1+"1";
        -> 2
mysql> SELECT CONCAT(2,' test');
        -> '2 test'
```

Wenn Sie eine Zahl explizit in eine Zeichenkette umwandeln wollen, übergeben Sie sie als Argument an CONCAT().

Wenn in einer Zeichenketten-Funktion eine binäre Zeichenkette als Argument angegeben wird, ist die resultierende Zeichenkette ebenfalls eine binäre Zeichenkette. Eine Zahl, die in eine Zeichenkette umgewandelt wird, wird als binäre Zeichenkette behandelt. Das betrifft nur Vergleichsoperationen.

9.2.3 Zeichenketten-Vergleichsfunktionen

Normalerweise wird ein Vergleich unter Berücksichtigung der Groß-/Kleinschreibung durchgeführt, wenn irgendein Ausdruck in einem Zeichenkettenvergleich abhängig von der verwendeten Groß-/Kleinschreibung ist.

9.2.3.1 ausdruck LIKE muster [ESCAPE 'fluchtzeichen']

Mustervergleich, der den einfachen SQL-Vergleich mit regulären Ausdrücken benutzt. Gibt 1 (TRUE) oder 0 (FALSE) zurück. Bei LIKE können Sie die folgenden zwei Platzhalterzeichen im Muster benutzen:

%	Entspricht einer beliebigen Anzahl von Zeichen, selbst 0 Zeichen
_	Entspricht genau einem Zeichen

```
mysql> select 'David!' LIKE 'David_';
    -> 1
mysql> select 'David!' LIKE '%D%v%';
    -> 1
```

Um auf literale Instanzen des Platzhalterzeichens zu testen, stellen Sie dem Zeichen ein Fluchtzeichen (Escape-Zeichen) voran. Wenn Sie das ESCAPE-Zeichen nicht angeben, wird `\' angenommen:

\%	Entspricht einem %-Zeichen
_	Entspricht einem _-Zeichen

```
mysql> select 'David!' LIKE 'David\_';
    -> 0
mysql> select 'David_' LIKE 'David\_';
    -> 1
```

Um ein anderes Fluchtzeichen (Escape-Zeichen) anzugeben, benutzen Sie die ESCAPE-Klausel:

```
mysql> select 'David_' LIKE 'David|_' ESCAPE '|';
    -> 1
```

Die folgenden beiden Statements zeigen, dass Zeichenketten-Vergleiche die Groß-/Kleinschreibung nicht berücksichtigen, solange nicht einer der Operanden eine binäre Zeichenkette ist: case insensitive unless one of the operands ist a binäre Zeichenkette:

```
mysql> select 'abc' LIKE 'ABC';
    -> 1
mysql> SELECT 'abc' LIKE BINARY 'ABC';
    -> 0
```

LIKE ist bei numerischen Ausdrücken zulässig! (Das ist eine MySQL-Erweiterung zum ANSI-SQL-LIKE.)

```
mysql> select 10 LIKE '1%';
    -> 1
```

HINWEIS: Weil MySQL die C Escape-Syntax in Zeichenketten benutzt (beispielsweise `\n'), müssen Sie jedes `\'-Zeichen, das Sie in LIKE-Zeichenketten benutzen, verdoppeln. Um zum Beispiel nach `\n' zu suchen, geben Sie `\\n' ein. Um nach `\' zu suchen, geben Sie `\\\\' ein (die Backslashes werden einmal vom Parser entfernt und noch einmal, wenn der Mustervergleich durchgeführt wird, so dass letztlich ein einzelner Backslash übrig bleibt).

9.2.3.2 ausdruck NOT LIKE muster [ESCAPE 'fluchtzeichen']

Dasselbe wie NOT (ausdruck LIKE muster [ESCAPE 'fluchtzeichen']).

9.2.3.3 ausdruck REGEXP muster , ausdruck RLIKE muster

Führt einen Mustervergleich eines Zeichenkettenausdrucks ausdruck gegen ein Muster muster durch. Das Muster kann ein erweiterter regulärer Ausdruck sein. Gibt 1 zurück, wenn ausdruck mit muster übereinstimmt, ansonsten 0. RLIKE ist ein Synonym für REGEXP, was aus Gründen der mSQL-Kompatibilität zur Verfügung steht. HINWEIS: Weil MySQL die C-Escape-Syntax in

Zeichenketten benutzt (beispielsweise `\n'), müssen Sie jeden `\', den Sie in Ihren REGEXP-Zeichenketten benutzen, verdoppeln. Ab MySQL-Version 3.23.4 berücksichtigt REGEXP nicht die verwendete Groß-/Kleinschreibung für normale (nicht binäre) Zeichenketten:

```
mysql> select 'Monty!' REGEXP 'm%y%%';
    -> 0
mysql> select 'Monty!' REGEXP '.*';
    -> 1
mysql> select 'new*\n*line' REGEXP 'new\\*.\\*line';
    -> 1
mysql> select "a" REGEXP "A", "a" REGEXP BINARY "A";
    -> 1  0
mysql> select "a" REGEXP "^[a-d]";
    -> 1
```

REGEXP und RLIKE benutzen den aktuellen Zeichensatz (vorgabemäßig ISO-8859-1 Latin1), wenn über den Typ eines Zeichens entschieden wird.

9.2.3.4 ausdruck NOT REGEXP muster ,ausdruck NOT RLIKE muster

Dasselbe wie NOT (ausdruck REGEXP muster).

9.2.3.5 STRCMP(ausdruck1,ausdruck2)

STRCMP() gibt 0 zurück, wenn die Zeichenketten gleich sind, -1, wenn das erste Argument kleiner als das zweite ist (nach der aktuellen Sortierreihenfolge), und ansonsten 1:

```
mysql> select STRCMP('text', 'text2');
    -> -1
mysql> select STRCMP('text2', 'text');
    -> 1
mysql> select STRCMP('text', 'text');
    -> 0
```

9.2.3.6 MATCH (spalte1,spalte2,...) AGAINST (ausdruck)

MATCH ... AGAINST() wird für Volltextsuche benutzt und gibt die Relevanz zurück - ein Ähnlichkeitsmaß zwischen dem Text in den Spalten (spalte1,spalte2,...) und der Anfrage ausdruck. Die Relevanz ist eine positive Fließkommazahl. 0 Relevanz bedeutet keine Ähnlichkeit. Damit MATCH ... AGAINST() funktioniert, muss zuerst ein **FULLTEXT**-Index erzeugt werden.

9.2.4 Groß-/Kleinschreibung

Der BINARY-Operator macht die folgende Zeichenkette zu einer binären Zeichenkette. Das ist eine einfache Möglichkeit, einen Spaltenvergleich zwangsweise in Abhängigkeit von der verwendeten Groß-/Kleinschreibung durchzuführen, selbst wenn die Spalte nicht als BINARY oder BLOB definiert ist:

```
mysql> select "a" = "A";
    -> 1
mysql> select BINARY "a" = "A";
    -> 0
```

Wenn Sie ein Blob ohne Berücksichtigung der Groß-/Kleinschreibung vergleichen wollen, können Sie den Blob jederzeit in Großschreibung umwandeln, bevor Sie den Vergleich durchführen:

```
SELECT 'A' LIKE UPPER(blob_spalte) FROM tabelle;
```

9.2.5 Numerische Funktionen

9.2.5.1 Arithmetische Operationen

Es gibt die üblichen arithmetischen Operatoren. Beachten Sie, dass das Ergebnis im Falle von `-', `+' und `*' mit BIGINT-Genauigkeit (64-Bit) berechnet wird, wenn beide Argumente Ganzzahlen sind!

9.2.5.2 +

Addition:

```
mysql> select 3+5;
    -> 8
```

9.2.5.3 -

Subtraktion:

```
mysql> select 3-5;
    -> -2
```

9.2.5.4 *

Multiplication:

```
mysql> select 3*5;
    -> 15
mysql> select 18014398509481984*18014398509481984.0;
    -> 324518553658426726783156020576256.0
mysql> select 18014398509481984*18014398509481984;
    -> 0
```

Das Ergebnis des letzten Ausdrucks ist falsch, weil die Ganzzahl-Multiplikation den 64-Bit-Wertebereich von BIGINT-Berechnungen überschreitet.

9.2.5.5 /

Division:

```
mysql> select 3/5;
    -> 0.60
```

Division durch 0 erzeugt ein NULL-Ergebnis:

```
mysql> select 102/(1-1);
    -> NULL
```

Eine Division wird nur dann mit BIGINT-Arithmetik berechnet, wenn sie in einem Zusammenhang durchgeführt wird, in dem das Ergebnis in eine Ganzzahl umgewandelt wird!

9.2.5.6 Mathematische Funktionen

Alle mathematischen Funktionen geben im Fehlerfall NULL zurück.

9.2.5.7 -

Unäres Minus. Ändert das Vorzeichen des Arguments:

```
mysql> select - 2;
    -> -2
```

Wenn dieser Operator mit einer BIGINT benutzt wird, beachten Sie, dass der Rückgabewert eine BIGINT ist! Das bedeutet, dass Sie - auf Ganzzahlen, die den Wert -2^63 haben könnten, vermeiden sollten!

9.2.5.8 ABS(X)

Gibt den absoluten Wert von X zurück:

```
mysql> select ABS(2);
    -> 2
mysql> select ABS(-32);
    -> 32
```

Diese Funktion kann bei BIGINT-Werten sicher benutzt werden.

9.2.5.9 SIGN(X)

Gibt das Vorzeichen des Arguments als -1, 0 oder 1 zurück, abhängig davon, ob X negativ, 0 oder positiv ist:

```
mysql> select SIGN(-32);
    -> -1
mysql> select SIGN(0);
    -> 0
mysql> select SIGN(234);
    -> 1
```

9.2.5.10 MOD(N,M) , %

Modulo (wie der %-Operator in C). Gibt den Rest von N dividiert durch M zurück:

```
mysql> select MOD(234, 10);
    -> 4
mysql> select 253% 7;
    -> 1
mysql> select MOD(29,9);
    -> 2
```

Diese Funktion kann bei BIGINT-Werten sicher benutzt werden.

9.2.5.11 FLOOR(X)

Gibt den größten Ganzzahl-Wert zurück, der nicht größer als X ist:

```
mysql> select FLOOR(1.23);
    -> 1
mysql> select FLOOR(-1.23);
    -> -2
```

Beachten Sie, dass der Rückgabewert in eine BIGINT umgewandelt wird!

9.2.5.12 CEILING(X)

Gibt den kleinsten Ganzzahl-Wert zurück, der nicht kleiner als X ist:

```
mysql> select CEILING(1.23);
    -> 2
mysql> select CEILING(-1.23);
    -> -1
```

Beachten Sie, dass der Rückgabewert in eine BIGINT umgewandelt wird!

9.2.5.13 ROUND(X)

Gibt das Argument X zurück, gerundet auf die nächste Ganzzahl:

```
mysql> select ROUND(-1.23);
    -> -1
mysql> select ROUND(-1.58);
    -> -2
mysql> select ROUND(1.58);
    -> 2
```

Beachten Sie, dass das Verhalten von ROUND() abhängig von der C-Bibliothek-Implementation ist, wenn das Argument in der Mitte zwischen zwei Ganzzahlen liegt. Einige runden auf die nächste gerade Zahl, oder immer

nach oben, immer nach unten oder immer Richtung 0. Wenn Sie eine bestimmte Art zu runden brauchen, sollten Sie stattdessen wohldefinierte Funktionen wie TRUNCATE() oder FLOOR() benutzen.

9.2.5.14 ROUND(X,D)

Gibt das Argument X zurück, gerundet auf eine Zahl mit D Dezimalstellen. Wenn D 0 ist, hat das Ergebnis keinen Dezimalpunkt oder Bruchteil:

```
mysql> select ROUND(1.298, 1);
    -> 1.3
mysql> select ROUND(1.298, 0);
    -> 1
```

9.2.5.15 EXP(X)

Gibt den Wert e (die Basis des natürlichen Logarithmus) hoch X zurück:

```
mysql> select EXP(2);
    -> 7.389056
mysql> select EXP(-2);
    -> 0.135335
```

9.2.5.16 LOG(X)

Gibt den natürlichen Logarithmus von X zurück:

```
mysql> select LOG(2);
    -> 0.693147
mysql> select LOG(-2);
    -> NULL
```

Wenn Sie den Logarithmus einer Zahl X zu einer beliebigen Basis B errechnen wollen, benutzen Sie die Formel LOG(X)/LOG(B).

9.2.5.17 LOG10(X)

Gibt den Logarithmus zur Basis 10 von X zurück:

```
mysql> select LOG10(2);
    -> 0.301030
mysql> select LOG10(100);
    -> 2.000000
mysql> select LOG10(-100);
    -> NULL
```

9.2.5.18 POW(X,Y) , POWER(X,Y)

Gibt den Wert X hoch Y zurück:

```
mysql> select POW(2,2);
    -> 4.000000
mysql> select POW(2,-2);
    -> 0.250000
```

9.2.5.19 SQRT(X)

Gibt die nicht negative Quadratwurzel von X zurück:

```
mysql> select SQRT(4);
    -> 2.000000
mysql> select SQRT(20);
    -> 4.472136
```

9.2.5.20 PI()

Gibt den Wert PI zurück. Die vorgabemäßig angezeigte Anzahl von Dezimalstellen ist 5, aber MySQL benutzt intern die volle doppelte Genauigkeit für PI.

```
mysql> select PI();
    -> 3.141593
mysql> SELECT PI()+0.000000000000000000;
    -> 3.141592653589793116
```

9.2.5.21 COS(X)

Gibt den Cosinus von X zurück, wobei X in Radianten angegeben wird:

```
mysql> select COS(PI());
    -> -1.000000
```

9.2.5.22 SIN(X)

Gibt den Sinus von X zurück, wobei X in Radianten angegeben wird:

```
mysql> select SIN(PI());
    -> 0.000000
```

9.2.5.23 TAN(X)

Gibt den Tangens von X zurück, wobei X in Radianten angegeben wird:

```
mysql> select TAN(PI()+1);
    -> 1.557408
```

9.2.5.24 ACOS(X)

Gibt den Arcuscosinus von X zurück, dass heißt den Wert, dessen Cosinus X ist. Gibt NULL zurück, wenn X nicht im Bereich von -1 bis 1 liegt:

```
mysql> select ACOS(1);
    -> 0.000000
mysql> select ACOS(1.0001);
    -> NULL
mysql> select ACOS(0);
    -> 1.570796
```

9.2.5.25 ASIN(X)

Gibt den Arcussinus von X zurück, das heißt den Wert, dessen Sinus X ist. Gibt NULL zurück, wenn X nicht im Bereich von -1 bis 1 liegt:

```
mysql> select ASIN(0.2);
    -> 0.201358
mysql> select ASIN('foo');
    -> 0.000000
```

9.2.5.26 ATAN(X)

Gibt den Arcustangens von X zurück, das heißt den Wert, dessen Tangens X ist:

```
mysql> select ATAN(2);
    -> 1.107149
mysql> select ATAN(-2);
    -> -1.107149
```

9.2.5.27 ATAN2(Y,X)

Gibt den Arcustangens der beiden Variablen X und Y zurück. Das ähnelt der Berechnung des Arcustangens von Y / X, ausser dass die Vorzeichen beider Argumente benutzt werden, um den Quadranten des Ergebnisses zu bestimmen:

```
mysql> select ATAN(-2,2);
    -> -0.785398
mysql> select ATAN(PI(),0);
    -> 1.570796
```

9.2.5.28 COT(X)

Gibt den Cotangens von X zurück:

```
mysql> select COT(12);
    -> -1.57267341
mysql> select COT(0);
    -> NULL
```

9.2.5.29 RAND() , RAND(N)

Gibt eine Zufallszahl (Fließkommawert) im Bereich von 0 bis 1.0 zurück. Wenn ein Ganzzahl-Argument N angegeben wird, wird es als Ausgangswert benutzt:

```
mysql> select RAND();
    -> 0.5925
mysql> select RAND(20);
    -> 0.1811
mysql> select RAND(20);
    -> 0.1811
mysql> select RAND();
    -> 0.2079
mysql> select RAND();
    -> 0.7888
```

Sie können eine Spalte mit RAND()-Werten nicht in einer ORDER BY-Klausel verwenden, weil ORDER BY die Spalte mehrfach auswerten würde. In MySQL-Version 3.23 können Sie jedoch folgendes tun: SELECT * FROM tabelle ORDER BY RAND() Das ist nützlich, um eine Zufallsstichprobe aus SELECT * FROM tabelle1,tabelle2 WHERE a=b AND c<d ORDER BY RAND() LIMIT 1000 zu erhalten. Beachten Sie, dass ein RAND() in einer WHERE-Klausel jedes Mal von Neuem ausgewertet wird, wenn WHERE ausgeführt wird.

9.2.5.30 LEAST(X,Y,...)

Mit zwei oder mehr Argumenten gibt die Funktion das kleinste Argument (das mit dem niedrigsten Wert) zurück. Die Argumente werden nach folgenden Regeln verglichen:

- Wenn der Rückgabewert in einem INTEGER-Zusammenhang benutzt wird oder alle Argumente Ganzzahl-Werte sind, werden sie als Ganzzahlen verglichen.
- Wenn der Rückgabewert in einem REAL-Zusammenhang benutzt wird oder alle Argumente Realzahlen sind, werden sie als Realzahlen verglichen.
- Wenn irgendein Argument eine von der Groß-/Kleinschreibung abhängige Zeichenkette ist, werden die Argumente als Zeichenketten, die von der Groß-/Kleinschreibung abhängen, verglichen.
- In sonstigen Fällen werden die Argumente als Zeichenketten verglichen, die nicht von der Groß-/Kleinschreibung abhängen:

```
mysql> select LEAST(2,0);
    -> 0
mysql> select LEAST(34.0,3.0,5.0,767.0);
    -> 3.0
mysql> select LEAST("B","A","C");
    -> "A"
```

In MySQL-Versionen vor Version 3.22.5 können Sie MIN() statt LEAST benutzen.

9.2.5.31 GREATEST(X,Y,...)

Gibt das größte Argument (das mit dem höchsten Wert) zurück. Die Argumente werden nach denselben Regeln wie bei LEAST verglichen:

```
mysql> select GREATEST(2,0);
    -> 2
mysql> select GREATEST(34.0,3.0,5.0,767.0);
    -> 767.0
mysql> select GREATEST("B","A","C");
    -> "C"
```

In MySQL-Versionen vor Version 3.22.5 können Sie MAX() statt GREATEST benutzen.

9.2.5.32 DEGREES(X)

Gibt das Argument X zurück, von Radianten zu Grad umgewandelt:

```
mysql> select DEGREES(PI());
    -> 180.000000
```

9.2.5.33 RADIANS(X)

Gibt das Argument X zurück, von Grad zu Radianten umgewandelt:

```
mysql> select RADIANS(90);
    -> 1.570796
```

9.2.5.34 TRUNCATE(X,D)

Gibt die Zahl X zurück, auf D Dezimalstellen beschnitten. Wenn D 0 ist, hat das Ergebnis keinen Dezimalpunkt oder Bruchteil:

```
mysql> select TRUNCATE(1.223,1);
    -> 1.2
mysql> select TRUNCATE(1.999,1);
    -> 1.9
mysql> select TRUNCATE(1.999,0);
    -> 1
```

Beachten Sie, dass Dezimalzahlen in Computern normalerweise nicht als exakte Zahlen, sondern als Double-Werte gespeichert werden. Daher können verwirrende Ergebnisse wie im folgenden Beispiel auftreten:

```
mysql> select TRUNCATE(10.28*100,0);
   -> 1027
```

Das Obige passiert, weil 10.28 tatsächlich als etwas wie 10.2799999999999999 gespeichert wird.

9.2.6 Datums- und Zeit-Funktionen

9.2.6.1 DAYOFWEEK(datum)

Gibt den Wochentag-Index zurück. Für datum gilt: 1 = Sonntag, 2 = Montag, ... 7 = Samstag). Diese Index-Werte entsprechen dem ODBC-Standard:

```
mysql> select DAYOFWEEK('1998-02-03');
    -> 3
```

9.2.6.2 WEEKDAY(datum)

Gibt den Wochentag-Index für datum zurück (0 = Montag, 1 = Dienstag, ... 6 = Sonntag):

```
mysql> select WEEKDAY('1997-10-04 22:23:00');
    -> 5
mysql> select WEEKDAY('1997-11-05');
    -> 2
```

9.2.6.3 DAYOFMONTH(datum)

Gibt den Tag des Monats für datum im Bereich 1 bis 31 zurück:

```
mysql> select DAYOFMONTH('1998-02-03');
    -> 3
```

9.2.6.4 DAYOFYEAR(datum)

Gibt den Tag des Jahres für datum im Bereich 1 bis 366 zurück:

```
mysql> select DAYOFYEAR('1998-02-03');
    -> 34
```

9.2.6.5 MONTH(datum)

Gibt den Monat für datum im Bereich 1 bis 12 zurück:

```
mysql> select MONTH('1998-02-03');
    -> 2
```

9.2.6.6 DAYNAME(datum)

Gibt den Namen des Wochentags für datum zurück (auf englisch):

```
mysql> select DAYNAME("1998-02-05");
    -> 'Thursday'
```

9.2.6.7 MONTHNAME(datum)

Gibt den Namen des Monats für datum zurück (auf englisch):

```
mysql> select MONTHNAME("1998-02-05");
    -> 'February'
```

9.2.6.8 QUARTER(datum)

Gibt das Quartal des Jahres für datum im Bereich 1 bis 4 zurück:

```
mysql> select QUARTER('98-04-01');
    -> 2
```

9.2.6.9 WEEK(datum) , WEEK(datum,erste)

Mit einem einzelnen Argument gibt diese Funktion die Woche für datum im Bereich 0 bis 53 zurück (ja, es kann Anfänge der Woche 53 geben), für Orte, in denen Sonntag der erste Wochentag ist. In der Form mit zwei Argumenten gestattet WEEK() es, festzulegen, ob die Woche am Sonntag oder am Montag beginnt. Die Woche beginnt am Sonntag, wenn das zweite Argument 0 ist, und am Montag, wenn das zweite Argument 1 ist:

```
mysql> select WEEK('1998-02-20');
    -> 7
mysql> select WEEK('1998-02-20',0);
    -> 7
mysql> select WEEK('1998-02-20',1);
    -> 8
mysql> select WEEK('1998-12-31',1);
    -> 53
```

9.2.6.10 YEAR(datum)

Gibt das Jahr für datum im Bereich 1000 bis 9999 zurück:

```
mysql> select YEAR('98-02-03');
    -> 1998
```

9.2.6.11 YEARWEEK(datum) , YEARWEEK(datum,erste)

Gibt Jahr und Woche für ein Datum zurück. Das zweite Argument funktioniert genau wie das zweite Argument von WEEK(). Beachten Sie, dass das Jahr sich

in der ersten und letzten Woche des Jahres vom Jahr im Datums-Argument unterscheiden kann:

```
mysql> select YEARWEEK('1987-01-01');
    -> 198653
```

9.2.6.12 HOUR(zeit)

Gibt die Stunde für zeit im Bereich 0 bis 23 zurück:

```
mysql> select HOUR('10:05:03');
    -> 10
```

9.2.6.13 MINUTE(zeit)

Gibt die Minute für zeit im Bereich 0 bis 59 zurück:

```
mysql> select MINUTE('98-02-03 10:05:03');
    -> 5
```

9.2.6.14 SECOND(zeit)

Gibt die Sekunde für zeit im Bereich 0 bis 59 zurück:

```
mysql> select SECOND('10:05:03');
    -> 3
```

9.2.6.15 PERIOD_ADD(P,N)

Zählt N Monate zur Periode P hinzu (im Format YYMM oder YYYYMM). Gibt einen Wert im Format YYYYMM zurück. Beachten Sie, dass das Perioden-Argument P *kein* Datums-Wert ist:

```
mysql> select PERIOD_ADD(9801,2);
    -> 199803
```

9.2.6.16 PERIOD_DIFF(P1,P2)

Gibt die Anzahl von Monaten zwischen den Perioden P1 und P2 zurück. P1 und P2 sollten im Format YYMM oder YYYYMM sein. Beachten Sie, dass die Perioden-Argumente P1 und P2 *keine* Datumswerte sind:

```
mysql> select PERIOD_DIFF(9802,199703);
        ➔ 11
```

9.2.6.17 DATE_ADD, DATE_SUB, ADDDATE, SUBDATE

```
DATE_ADD(datum,INTERVAL ausdruck typ)
DATE_SUB(datum,INTERVAL ausdruck typ)
ADDDATE(datum,INTERVAL ausdruck typ)
SUBDATE(datum,INTERVAL ausdruck typ)
```

Diese Funktionen führen Datumsberechnungen durch. Sie wurden in MySQL-Version 3.22 eingeführt. ADDDATE() und SUBDATE() sind Synonyme für DATE_ADD() und DATE_SUB(). In MySQL-Version 3.23 können Sie + und - anstelle von DATE_ADD() und DATE_SUB() benutzen, wenn der Ausdruck auf der rechten Seite eine DATE oder DATETIME-Spalte ist (siehe Beispiel). datum ist ein DATETIME- oder DATE-Wert, der das Anfangsdatum festlegt. ausdruck ist ein Ausdruck, der den Intervallwert festlegt, der zum Anfangsdatum hinzugezählt oder von diesem abgezogen wird. ausdruck ist eine Zeichenkette; sie kann mit einem `-' für negative Intervalle beginnen. typ ist ein Schlüsselwort, das angibt, wie der Ausdruck interpretiert werden soll. Die verwandte Funktion EXTRACT(typ FROM datum) gibt das 'typ'-Intervall des Datums zurück. Folgende Tabelle zeigt, in welchem Zusammenhang die typ- und ausdruck-Argumente stehen:

typ **wert**	**erwartet** ausdruck **format**
SECOND	Sekunden
MINUTE	Minuten
HOUR	Stunden
DAY	Tage
MONTH	Monate
YEAR	Jahre
MINUTE_SECOND	"Minuten:Sekunden"
HOUR_MINUTE	"Stunden:Minuten"
DAY_HOUR	"Tage Stunden"
YEAR_MONTH	"Jahre-Monate"
HOUR_SECOND	"Stunden:Minuten:Sekunden"
DAY_MINUTE	"Tage Stunden:Minuten"
DAY_SECOND	"Tage Stunden:Minuten:Sekunden"

MySQL erlaubt beliebige Satzzeichen-Begrenzer im ausdruck-Format. Die in der Tabelle gezeigten Begrenzer sind Vorschläge. Wenn das datum-Argument ein DATE-Wert ist und Ihre Berechnungen nur YEAR, MONTH und DAY-Anteile beinhalten (also keine Zeit-Anteile), ist das Ergebnis ein DATE-Wert. Ansonsten ist das Ergebnis ein DATETIME-Wert:

```
mysql> SELECT "1997-12-31 23:59:59" + INTERVAL 1 SECOND;
    -> 1998-01-01 00:00:00
mysql> SELECT INTERVAL 1 DAY + "1997-12-31";
    -> 1998-01-01
mysql> SELECT "1998-01-01" - INTERVAL 1 SECOND;
    -> 1997-12-31 23:59:59
mysql> SELECT DATE_ADD("1997-12-31 23:59:59",
                       INTERVAL 1 SECOND);
    -> 1998-01-01 00:00:00
mysql> SELECT DATE_ADD("1997-12-31 23:59:59",
                       INTERVAL 1 DAY);
    -> 1998-01-01 23:59:59
mysql> SELECT DATE_ADD("1997-12-31 23:59:59",
                       INTERVAL "1:1" MINUTE_SECOND);
    -> 1998-01-01 00:01:00
mysql> SELECT DATE_SUB("1998-01-01 00:00:00",
                       INTERVAL "1 1:1:1" DAY_SECOND);
    -> 1997-12-30 22:58:59
mysql> SELECT DATE_ADD("1998-01-01 00:00:00",
                       INTERVAL "-1 10" DAY_HOUR);
    -> 1997-12-30 14:00:00
mysql> SELECT DATE_SUB("1998-01-02", INTERVAL 31 DAY);
    -> 1997-12-02
```

Wenn Sie einen Intervallwert angeben, der zu kurz ist (nicht alle Intervall-Anteile beinhaltet, die vom typ-Schlüsselwort erwartet werden), nimmt MySQL an, dass Sie den äußersten linken Teil des Intervallwerts ausgelassen haben. Wenn Sie beispielsweise einen typ DAY_SECOND angeben, wird vom Wert von ausdruck erwartet, dass dieser Tages-, Stunden-, Minuten- und Sekunden-Anteile enthält. Wenn Sie einen Wert wie "1:10" angeben, nimmt MySQL an, dass die Tages- und Stunden-Anteile fehlen und der Wert Minuten und Sekunden darstellt. Mit anderen Worten wird "1:10" DAY_SECOND so interpretiert, dass es äquivalent zu "1:10" MINUTE_SECOND ist. Das ist analog zur Weise, wie MySQL TIME-Werte interpretiert, die eher vergangene Zeit als

Tageszeit darstellen. Beachten Sie, dass ein Datumswert automatisch in einen DATETIME-Wert umgewandelt wird, wenn Sie einen DATE-Wert zu etwas hinzuzählen oder von etwas abziehen, das einen Zeit-Anteil hat:

```
mysql> select date_add("1999-01-01", interval 1 day);
    -> 1999-01-02
mysql> select date_add("1999-01-01", interval 1 hour);
    -> 1999-01-01 01:00:00
```

Wenn Sie wirklich falsche Datumsangaben benutzen, ist das Ergebnis NULL. Wenn Sie MONTH, YEAR_MONTH oder YEAR hinzuzählen und das Datumsergebnis einen Tag hat, der größer ist als der höchste Tag für den neuen Monat, wird der Tag auf den höchsten Tag des neuen Monats angepasst:

```
mysql> select DATE_ADD('1998-01-30', Interval 1 month);
    -> 1998-02-28
```

Beachten Sie, dass das Wort INTERVAL und das typ-Schlüsselwort in den vorstehenden Beispielen nicht von der verwendeten Groß-/Kleinschreibung abhängen.

9.2.6.18 EXTRACT(typ FROM datum)

Die EXTRACT()-Funktion benutzt dieselbe Art von Intervalltyp-Spezifikatoren wie DATE_ADD() oder DATE_SUB(), extrahiert aber Anteile aus dem Datum, statt Datumsberechnungen durchzuführen:

```
mysql> SELECT EXTRACT(YEAR FROM "1999-07-02");
    -> 1999
mysql> SELECT EXTRACT(YEAR_MONTH FROM "1999-07-02 01:02:03");
    -> 199907
mysql> SELECT EXTRACT(DAY_MINUTE FROM "1999-07-02 01:02:03");
    -> 20102
```

9.2.6.19 TO_DAYS(datum)

Gibt für ein Datum datum eine Tagesanzahl zurück (die Anzahl von Tagen seit dem Jahr 0):

```
mysql> select TO_DAYS(950501);
    -> 728779
mysql> select TO_DAYS('1997-10-07');
    -> 729669
```

TO_DAYS() ist nicht für die Benutzung mit Werten vor der Einführung des Gregorianischen Kalenders (1582) vorgesehen, weil es nicht die Tage berücksichtigt, die verloren gingen, als der Kalender geändert wurde.

9.2.6.20 FROM_DAYS(N)

Gibt für eine Tagesanzahl N einen DATE-Wert zurück:

```
mysql> select FROM_DAYS(729669);
    -> '1997-10-07'
```

FROM_DAYS() ist nicht für die Benutzung mit Werten vor der Einführung des Gregorianischen Kalenders (1582) vorgesehen, weil es nicht die Tage berücksichtigt, die verloren gingen, als der Kalender geändert wurde.

9.2.6.21 DATE_FORMAT(datum,format)

Formatiert den datum-Wert gemäß der format-Zeichenkette. Folgende Spezifikatoren können in der format-Zeichenkette benutzt werden:

%M	Monatsname auf englisch (January bis December)
%W	Name des Wochentags auf englisch (Sunday bis Saturday)

%D	Tag des Monats mit englischem Suffix (1st, 2nd, 3rd usw.)
%Y	Jahr, numerisch, 4 Ziffern
%y	Jahr, numerisch, 2 Ziffern
%X	Jahr der Woche, wobei Sonntag der erste Tag der Woche ist, numerisch, 4 Ziffern, benutzt mit '%V'
%x	Jahr der Woche, wobei Montag der erste Tag der Woche ist, numerisch, 4 Ziffern, benutzt mit '%v'
%a	Abgekürzter Name des Wochentags auf englisch (Sun..Sat)
%d	Tag des Monats, numerisch (00 bis 31)
%e	Tag des Monats, numerisch (0 bis 31)
%m	Monat, numerisch (01 bis 12)
%c	Monat, numerisch (1 bis 12)
%b	Abgekürzter Monatsname auf englisch (Jan bis Dec)
%j	Tag des Jahrs (001 bis 366)
%H	Stunde (00 bis 23)
%k	Stunde (0 bis 23)
%h	Stunde (01 bis 12)
%I	Stunde (01 bis 12)
%l	Stunde (1 bis 12)
%i	Minuten, numerisch (00 bis 59)
%r	Uhrzeit, 12-Stunden-Format (hh:mm:ss [AP]M)
%T	Uhrzeit, 24-Stunden-Format (hh:mm:ss)
%S	Sekunden (00 bis 59)
%s	Sekunden (00 bis 59)
%p	AM oder PM
%w	Wochentag (0=Sonntag bis 6=Samstag)
%U	Woche (0 bis 53), wobei Sonntag der erste Tag der Woche ist
%u	Woche (0 bis 53), wobei Montag der erste Tag der Woche ist
%V	Woche (1 bis 53), wobei Sonntag der erste Tag der Woche ist. Benutzt mit '%X'
%v	Woche (1 bis 53), wobei Montag der erste Tag der Woche ist. Benutzt mit '%x'
%%	Ein Literal `%'.

Alle anderen Zeichen werden einfach ohne Interpretation ins Ergebnis kopiert:

```
mysql> select DATE_FORMAT('1997-10-04 22:23:00', '%W%M%Y');
    -> 'Saturday October 1997'
mysql> select DATE_FORMAT('1997-10-04 22:23:00', '%H:%i:%s');
    -> '22:23:00'
mysql> select DATE_FORMAT('1997-10-04 22:23:00',
                  '%D%y%a%d%m%b%j');
    -> '4th 97 Sat 04 10 Oct 277'
mysql> select DATE_FORMAT('1997-10-04 22:23:00',
                  '%H%k%I%r%T%S%w');
    -> '22 22 10 10:23:00 PM 22:23:00 00 6'
mysql> select DATE_FORMAT('1999-01-01', '%X%V');
    -> '1998 52'
```

Ab MySQL-Version 3.23 ist das `%'-Zeichen vor Format-Spezifikator-Zeichen erforderlich. In früheren Versionen von MySQL war `%' optional.

9.2.6.22 TIME_FORMAT(zeit,format)

Dieses wird benutzt wie die obige DATE_FORMAT()-Funktion, aber die format-Zeichenkette darf nur die Spezifikatoren enthalten, die Stunden, Minuten und Sekunden handhaben. Andere Spezifikatoren erzeugen einen NULL-Wert oder 0.

9.2.6.23 CURDATE() , CURRENT_DATE

Gibt das Datum von heute im 'YYYY-MM-DD'- oder YYYYMMDD-format zurück, abhängig davon, ob die Funktion in einem Zeichenketten- oder in einem numerischen Zusammenhang benutzt wird:

```
mysql> select CURDATE();
    -> '1997-12-15'
mysql> select CURDATE() + 0;
    -> 19971215
```

9.2.6.24 CURTIME() , CURRENT_TIME

Gibt die aktuelle Zeit als einen Wert im 'HH:MM:SS'- oder HHMMSS-format zurück, abhängig davon, ob die Funktion in einem Zeichenketten- oder in einem numerischen Zusammenhang benutzt wird:

```
mysql> select CURTIME();
    -> '23:50:26'
mysql> select CURTIME() + 0;
    -> 235026
```

9.2.6.25 NOW() , SYSDATE() , CURRENT_TIMESTAMP

Gibt das aktuelle Datum und die aktuelle Zeit als einen Wert im 'YYYY-MM-DD HH:MM:SS'- oder YYYYMMDDHHMMSS-Format zurück, abhängig davon, ob die Funktion in einem Zeichenketten- oder in einem numerischen Zusammenhang benutzt wird:

```
mysql> select NOW();
    -> '1997-12-15 23:50:26'
mysql> select NOW() + 0;
    -> 19971215235026
```

9.2.6.26 UNIX_TIMESTAMP() , UNIX_TIMESTAMP(datum)

Ohne Argument aufgerufen gibt die Funktion einen Unix-Zeitstempel zurück (Sekunden seit '1970-01-01 00:00:00' GMT). Wenn UNIX_TIMESTAMP() mit einem datum-Argument aufgerufen wird, gibt sie den Wert des Arguments als Sekunden seit '1970-01-01 00:00:00' GMT zurück. datum kann eine DATE-Zeichenkette, eine DATETIME-Zeichenkette, ein TIMESTAMP oder eine Zahl im Format YYMMDD oder YYYYMMDD in lokaler Zeit sein:

```
mysql> select UNIX_TIMESTAMP();
    -> 882226357
mysql> select UNIX_TIMESTAMP('1997-10-04 22:23:00');
    -> 875996580
```

Wenn UNIX_TIMESTAMP auf einer TIMESTAMP-Spalte benutzt wird, erhält die Funktion den Wert direkt, ohne implizite ``zeichenkette-zu-unix-zeitstempel''-Umwandlung. Wenn Sie UNIX_TIMESTAMP() einen falschen Wert oder einen Wert ausserhalb des Wertebereichs angeben, gibt sie 0 zurück.

9.2.6.27 FROM_UNIXTIME(unix_zeitstempel)

Gibt das unix_timestamp-Argument als Wert im 'YYYY-MM-DD HH:MM:SS'- oder YYYYMMDDHHMMSS-Format zurück, abhängig davon, ob die Funktion in einem Zeichenketten- oder in einem numerischen Zusammenhang benutzt wird:

```
mysql> select FROM_UNIXTIME(875996580);
    -> '1997-10-04 22:23:00'
mysql> select FROM_UNIXTIME(875996580) + 0;
    -> 19971004222300
```

9.2.6.28 FROM_UNIXTIME(unix_zeitstempel,format)

Gibt das unix_timestamp-Argument als Wert zurück, der wie mit der format-Zeichenkette angegeben formatiert ist. format kann dieselben Spezifikatoren wie die DATE_FORMAT()-Funktion enthalten:

```
mysql> select FROM_UNIXTIME(UNIX_TIMESTAMP(),
                  '%Y%D%M%h:%i:%s%x');
    -> '1997 23rd December 03:43:30 x'
```

9.2.6.29 SEC_TO_TIME(sekunden)

Gibt das sekunden-Argument, umgewandelt in Stunden, Minuten und Sekunden, als Wert im 'HH:MM:SS'- oder HHMMSS-Format zurück, abhängig davon, ob die Funktion in einem Zeichenketten- oder in einem numerischen Zusammenhang benutzt wird:

```
mysql> select SEC_TO_TIME(2378);
    -> '00:39:38'
mysql> select SEC_TO_TIME(2378) + 0;
    -> 3938
```

9.2.6.30 TIME_TO_SEC(zeit)

Gibt das zeit-Argument, umgewandelt in Sekunden, zurück:

```
mysql> select TIME_TO_SEC('22:23:00');
    -> 80580
mysql> select TIME_TO_SEC('00:39:38');
    -> 2378
```

9.2.7 Weitere Funktionen

9.2.7.1 Bit-Funktionen

MySQL benutzt BIGINT-Berechnungen (64-Bit) für Bit-Operationen, so dass diese Operatoren einen maximalen Wertebereich von 64 Bits haben.

9.2.7.2 |

Bitweises OR:

```
mysql> select 29 | 15;
    -> 31
```

9.2.7.3 &

Bitweises AND:

```
mysql> select 29 & 15;
    -> 13
```

9.2.7.4 <<

Verschiebt eine BIGINT-Zahl nach links:

```
mysql> select 1 << 2;
```

```
    -> 4
```

9.2.7.5 >>

Verschiebt eine BIGINT-Zahl nach rechts:

```
mysql> select 4 >> 2;
    -> 1
```

9.2.7.6 ~

Invertiert alle Bits:

```
mysql> select 5 & ~1;
    -> 4
```

9.2.7.7 BIT_COUNT(N)

Gibt die Anzahl von Bits, die im Argument N gesetzt sind, zurück:

```
mysql> select BIT_COUNT(29);
    -> 4
```

9.2.7.8 Verschiedene Funktionen

9.2.7.9 DATABASE()

Gibt den aktuellen Datenbanknamen zurück:

```
mysql> select DATABASE();
    -> 'test'
```

Wenn es keine aktuelle Datenbank gibt, gibt DATABASE() die leere Zeichenkette zurück.

9.2.7.10 USER() , SYSTEM_USER() , SESSION_USER()

Gibt den aktuellen MySQL-Benutzernamen zurück:

```
mysql> select USER();
    -> 'heinzholger@localhost'
```

Ab MySQL-Version 3.22.11 beinhaltet dieser Wert den Client-Hostnamen sowie den Benutzernamen. Sie können nur den Benutzernamen-Anteil wie folgt extrahieren (was funktioniert, ob der Wert nun einen Hostnamen-Anteil hat oder nicht):

```
mysql> select substring_index(USER(),"@",1);
    -> 'heinzholger'
```

9.2.7.11 PASSWORD(zeichenkette)

Berechnet eine Passwort-Zeichenkette aus dem Klartext-Passwort zeichenkette. Diese Funktion wird benutzt, um MySQL-Passwörter zum Speichern in der Password-Spalte der user-Berechtigungstabelle zu verschlüsseln:

```
mysql> select PASSWORD('schlechtespasswort');
    -> '1ccbb34b4e2b2f95'
```

Die PASSWORD()-Verschlüsselung ist nicht umkehrbar. PASSWORD() führt keine Passwort-Verschlüsselung in der Art durch, wie Unix-Passwörter verschlüsselt werden. Sie sollten nicht annehmen, dass Ihr Unix-Passwort und Ihr MySQL-Passwort dasselbe sind. PASSWORD() ergibt denselben verschlüsselten Wert, wie er in der Unix-Passwortdatei gespeichert ist. Siehe ENCRYPT().

9.2.7.12 ENCRYPT(zeichenkette[,salt])

Verschlüsselt zeichenkette unter Benutzung des Unix-crypt()-Systemaufrufs. Das salt-Argument sollte eine Zeichenkette mit zwei Zeichen sein (ab MySQL-Version 3.22.16 darf salt länger als zwei Zeichen sein):

```
mysql> select ENCRYPT("hello");
    -> 'VxuFAJXVARROc'
```

Wenn crypt() auf Ihrem System nicht verfügbar ist, gibt ENCRYPT() immer NULL zurück. ENCRYPT() ignoriert alle ausser den ersten 8 Zeichen von zeichenkette, zumindest auf einigen Systemen. Das wird durch den zugrunde liegenden crypt()-Systemaufruf festgelegt.

9.2.7.13 ENCODE(zeichenkette,passwort_zeichenkette)

Verschlüsselt zeichenkette, indem passwort_zeichenkette als Passwort benutzt wird. Um das Ergebnis zu entschlüsseln, benutzen Sie DECODE(). Das Ergebnis ist eine binäre Zeichenkette derselben Länge wie zeichenkette. Wenn Sie sie in einer Spalte speichern wollen, benutzen Sie eine BLOB-Spalte.

9.2.7.14 DECODE(crypt_zeichenkette,passwort_zeichenkette)

Entschlüsselt die verschlüsselte Zeichenkette crypt_zeichenkette, indem passwort_zeichenkette als Passwort benutzt wird. crypt_zeichenkette sollte eine Zeichenkette sein, die von ENCODE() zurückgegeben wird.

9.2.7.15 MD5(zeichenkette)

Berechnet eine MD5-Prüfsumme für die Zeichenkette. Der Wert wird als eine 32 Stellen lange hexadezimale Zahl zurückgegeben, die zum Beispiel als Hash-Schlüssel benutzt werden kann:

```
mysql> select MD5("testing");
    -> 'ae2b1fca515949e5d54fb22b8ed95575'
```

Das ist ein "RSA Data Sicherheit, Inc. MD5 Message-Digest Algorithm".

9.2.7.16 LAST_INSERT_ID([ausdruck])

Gibt den letzten automatisch erzeugten Wert zurück, der in eine AUTO_INCREMENT-Spalte eingefügt wurde

```
mysql> select LAST_INSERT_ID();
    -> 195
```

Die letzte ID, die erzeugt wurde, wird im Server für jede Verbindung separat gespeichert. Sie wird nicht durch andere Clients geändert. Sie wird nicht einmal geändert, wenn Sie eine andere AUTO_INCREMENT-Spalte mit einem nicht 'magischen' Wert aktualisieren (also einem Wert, der nicht NULL und nicht 0 ist). Wenn Sie viele Zeilen zugleich mit einem Insert-Statement einfügen, gibt LAST_INSERT_ID() den Wert für die erste eingefügte Zeile zurück. Der Grund dafür liegt darin, dass es Ihnen dadurch ermöglicht wird, dasselbe INSERT-Statement auf einfache Weise auf einem anderen Server zu reproduzieren. Wenn ausdruck als Argument zu LAST_INSERT_ID() angegeben wird, wird der Wert des Arguments von der Funktion zurückgegeben, als nächster Wert gesetzt, der von LAST_INSERT_ID() zurückgegeben wird und als nächster auto_increment-Wert benutzt. Damit können Sie Zahlenfolgen emulieren: Erzeugen Sie zuerst die Tabelle:

```
mysql> create table sequenz (id int not null);
mysql> insert into sequenz values (0);
```

Danach kann die Tabelle benutzt werden, um wie folgt Zahlenfolgen zu erzeugen:

```
mysql> update sequenz set id=LAST_INSERT_ID(id+1);
```

Sie können Zahlenfolgen erzeugen, ohne LAST_INSERT_ID() aufzurufen, aber der Nutzen, die Funktion auf diese Art zu benutzen, liegt darin, dass der ID-Wert im Server als letzter automatisch erzeugter Wert gehalten wird. Sie

können die neue ID auf dieselbe Art abrufen, wie Sie jeden anderen normalen AUTO_INCREMENT-Wert in MySQL lesen würden. LAST_INSERT_ID() (ohne Argument) zum Beispiel gibt die neue ID zurück. Die C-API-Funktion mysql_insert_id() kann ebenfalls benutzt werden, um den Wert zu erhalten. Beachten Sie, dass Sie diese Funktion nicht benutzen können, um den Wert von LAST_INSERT_ID(ausdruck) abzurufen, nachdem Sie andere SQL-Statements wie SELECT oder SET ausgeführt haben, weil mysql_insert_id() nur nach INSERT- und UPDATE-Statements aktualisiert wird.

9.2.7.17 FORMAT(X,D)

Formatiert die Zahl X in ein Format wie '#,###,###.##', gerundet auf D Dezimalstellen. Wenn D 0 ist, hat das Ergebnis keinen Dezimalpunkt oder Bruchteil:

```
mysql> select FORMAT(12332.123456, 4);
    -> '12,332.1235'
mysql> select FORMAT(12332.1,4);
    -> '12,332.1000'
mysql> select FORMAT(12332.2,0);
    -> '12,332'
```

9.2.7.18 VERSION()

Gibt eine Zeichenkette zurück, die die MySQL-Serverversion anzeigt:

```
mysql> select VERSION();
    -> '3.23.13-log'
```

Wenn Ihre Versionsnummer mit -log endet, bedeutet das, dass Loggen angeschaltet ist.

9.2.7.19 CONNECTION_ID()

Gibt die Verbindungskennnummer (Thread_id) für die Verbindung zurück. Jede Verbindung hat ihre eigene eindeutige Kennnummer:

```
mysql> select CONNECTION_ID();
    -> 1
```

9.2.7.20 GET_LOCK(zeichenkette,zeitueberschreitung)

Versucht, eine Sperre mit dem Namen, der durch die Zeichenkette zeichenkette angegeben wird, zu erlangen, mit einem Timeout von zeitueberschreitung Sekunden. Gibt 1 zurück, wenn die Sperre erfolgreich erlangt wurde, und 0, wenn der Versuch wegen Zeitüberschreitung abgebrochen wurde, oder NULL, wenn ein Fehler auftrat (wenn zum Beispiel kein Arbeitsspeicher mehr frei ist oder der Thread mit mysqladmin kill gekillt wurde). Eine Sperre wird aufgehoben, wenn Sie RELEASE_LOCK() ausführen, einen neuen GET_LOCK() ausführen oder der Thread beendet wird. Diese Funktion kann benutzt werden, um Applikations-Sperren zu implementieren oder um Datensatz-Sperren zu simulieren. Sie blockiert Anfragen von anderen Clients nach Sperren mit demselben Namen; Clients, die sich auf einen angegebenen Namen für die Sperr-Zeichenkette einigen, können die Zeichenkette benutzen, um kooperatives beratendes Sperren (advisory locking) auszuführen:

```
mysql> select GET_LOCK("lock1",10);
    -> 1
mysql> select GET_LOCK("lock2",10);
    -> 1
mysql> select RELEASE_LOCK("lock2");
    -> 1
mysql> select RELEASE_LOCK("lock1");
```

```
    -> NULL
```
Beachten Sie, dass der zweite RELEASE_LOCK()-Aufruf NULL zurückgibt, weil die Sperre "lock1" automatisch durch den zweiten GET_LOCK()-Aufruf aufgehoben wurde.

9.2.7.21 RELEASE_LOCK(zeichenkette)

Hebt die Sperre auf, die durch die Zeichenkette zeichenkette benannt ist, die mit GET_LOCK() erlangt wurde. Gibt 1 zurück, wenn die Sperre aufgehoben wurde, und 0, wenn die Sperre nicht durch diesen Thread gemacht wurde (in diesem Fall wird die Sperre nicht aufgehoben), oder NULL, wenn die benannte Sperre nicht existiert. Die Sperre existiert nicht, wenn sie nie durch einen Aufruf von GET_LOCK() erlangt wurde oder wenn sie bereits aufgehoben wurde.

9.2.7.22 BENCHMARK(zaehler,ausdruck)

Die BENCHMARK()-Funktion den Ausdruck ausdruck wiederholt zaehler mal aus. Sie kann benutzt werden, um die Zeit zu ermitteln, die MySQL benötigt, um den Ausdruck zu verarbeiten. Der Ergebniswert ist immer 0. Die Funktion ist für die Benutzung im mysql-Client gedacht, der die Ausführungszeiten von Anfragen zum Beispiel wie folgt darstellt:

```
mysql> select BENCHMARK(1000000,encode("hello","goodbye"));
+----------------------------------------------+
| BENCHMARK(1000000,encode("hello","goodbye")) |
+----------------------------------------------+
|                                            0 |
+----------------------------------------------+
1 row in set (4.74 sec)
```

Die berichtete Zeit ist die am Client-Ende verstrichene Zeit, nicht die Prozessorzeit am Server-Ende. Es ist ratsam, BENCHMARK() mehrere Male auszuführen und das Ergebnis unter Berücksichtigung der Last, unter der die Server-Maschine fährt, zu interpretieren.

9.2.7.23 INET_NTOA(ausdruck)

Gibt die Netzwerk-Adresse (4 oder 8 Bytes) für den numerischen Ausdruck zurück:

```
mysql> select INET_NTOA(3520061480);
    ->  "209.207.224.40"
```

9.2.7.24 INET_ATON(ausdruck)

Gibt eine Ganzzahl zurück, die den numerischen Wert einer Netzwerk-Adresse darstellt. Adressen können 4-Byte- oder 8-Byte-Adressen sein:

```
mysql> select INET_ATON("209.207.224.40");
    ->  3520061480
```

Die erzeugte Zahl ist immer in Netzwerk-Byte-Reihenfolge; die obige Zahl wird zum Beispiel errechnet als 209*255^3 + 207*255^2 + 224*255 +40.

9.2.7.25 MASTER_POS_WAIT(log_name, log_position)

Blockiert, bis der Slave während der Replikation die festgelegte Position in der Master-Log-Datei erreicht. Wenn die Master-Information nicht initialisiert wird, wird NULL zurückgegeben. Wenn der Slave nicht läuft, blockiert die Funktion und wartet, bis er gestartet wurde, und geht dann hinter die angegebene Position. Wenn der Slave bereits hinter der angegebenen Position ist, kehrt die Funktion sofort zurück. Der Rückgabewert ist die Anzahl von Log-Events, die sie warten muss, um bis zur angegebenen Position zu kommen, oder

NULL in Fehlerfällen. Nützlich für die Steuerung der Master-Slave-Synchronisation, aber ursprünglich geschrieben, um das Testen der Replikation zu erleichtern.

9.2.8 Funktionen zur Benutzung bei GROUP BY-Klauseln

Wenn Sie in einem Statement eine Gruppierungsfunktion benutzen, die keine GROUP BY-Klausel enthält, ist das gleichbedeutend mit der Gruppierung aller Zeilen.

9.2.8.1 COUNT(ausdruck)

Gibt die Anzahl der Zeilen mit Nicht-NULL-Werten zurück, die durch ein SELECT-Statement abgerufen werden:

```
mysql> select student.student_name,COUNT(*)
        from student,kurs
        where student.student_id=kurs.student_id
        GROUP BY student_name;
```

COUNT(*) ist insofern anders, als es die Anzahl der abgerufenen Zeilen zurückgibt, egal ob sie NULL-Werte enthalten oder nicht. COUNT(*) ist darauf optimiert, das Ergebnis sehr schnell zurückzugeben, wenn es mittels eines SELECT von einer Tabelle abruft, wenn keine weiteren Spalten abgerufen werden und es keine WHERE-Klausel gibt. Beispiel:

```
mysql> select COUNT(*) from student;
```

9.2.8.2 COUNT(DISTINCT ausdruck,[ausdruck...])

Gibt die Anzahl unterschiedlicher Nich-NULL-Werte zurück:

```
mysql> select COUNT(DISTINCT ergebnisse) from student;
```

Bei MySQL erhalten Sie die Anzahl unterschiedlicher Ausdruckskombinationen, die nicht NULL enthalten, indem Sie eine Liste von Ausdrücken angeben. In ANSI-SQL müssten Sie eine Verkettung aller Ausdrücke innerhalb von CODE(DISTINCT ..) angeben.

9.2.8.3 AVG(ausdruck)

Gibt den Durchschnittswert von ausdruck zurück:

```
mysql> select student_name, AVG(test_ergebnis)
        from student
        GROUP BY student_name;
```

9.2.8.4 MIN(ausdruck) , MAX(ausdruck)

Gibt den kleinsten oder größten Wert von ausdruck zurück. MIN() und MAX() können Zeichenketten-Argumente aufnehmen und geben in solchen Fällen den kleinsten oder größten Zeichenketten- Wert zurück.

```
mysql> select student_name, MIN(test_ergebnis), MAX(test_ergebnis)
        from student
        GROUP BY student_name;
```

9.2.8.5 SUM(ausdruck)

Gibt die Summe von ausdruck zurück. Beachten Sie, dass der Rückgabewert NULL ist, wenn die Ergebnismenge keine Zeilen hat!

9.2.8.6 STD(ausdruck) , STDDEV(ausdruck)

Gibt die Standardabweichung von ausdruck zurück. Das ist eine Erweiterung zu ANSI-SQL. Die STDDEV()-Form dieser Funktion wird aus Gründen der Oracle-Kompatibilität zur Verfügung gestellt.

9.2.8.7 BIT_OR(ausdruck)

Gibt das bitweise OR aller Bits in ausdruck zurück. Die Berechnung wird mit 64-Bit-(BIGINT)-Genauigkeit durchgeführt.

9.2.8.8 BIT_AND(ausdruck)

Gibt das bitweise AND aller Bits in ausdruck zurück. Die Berechnung wird mit 64-Bit-(BIGINT)-Genauigkeit durchgeführt.

MySQL hat die Benutzung von GROUP BY erweitert. Sie können Spalten oder Berechnungen im SELECT-Ausdruck angeben, die nicht im GROUP BY-Teil erscheinen. Das steht für *jeden möglichen Wert für diese Gruppe*. Das können Sie benutzen, um bessere Performance zu erzielen, indem Sie Sortieren und Gruppieren unnötiger Bestandteile vermeiden. Zum Beispiel müssen Sie in folgender Anfrage nicht nach kunde.name gruppieren:

```
mysql> select bestellung.kunde_id,kunde.name,max(zahlungen)
       from bestellung,kunde
       where bestellung.kunde_id = kunde.kunde_id
       GROUP BY bestellung.kunde_id;
```

In ANSI-SQL müssten Sie der GROUP BY-Klausel kunde.name hinzufügen. In MySQL ist der Name überflüßig, solange Sie nicht im ANSI-Modus fahren.

Benutzen Sie dieses Feature nicht, wenn die Spalten, die Sie im GROUP BY-Teil auslassen, in der Gruppe nicht eindeutig sind! Sonst erhalten Sie unvorhersagbare Ergebnisse.

In einigen Fällen können Sie MIN() und MAX() benutzen, um einen bestimmten Spaltenwert zu erhalten, selbst wenn er nicht eindeutig ist. Folgendes gibt den Wert von spalte aus der Zeile zurück, die den kleinsten Wert in der sortierung-Spalte enthält:

```
substr(MIN(concat(rpad(sortierung,6,' '),spalte)),7)
```

9.3 Links

- www.mysql.com und www.mysql.de – die Heimat von MySQL
- http://www.mysql.de/products/administrator/index.html - MySQLAdmin
- www.fabforce.net/dbdesigner4/ DBDesigner4
- http://www.phpmyadmin.net/ PhpMyAdmin