SQL – Tipps und Tricks – Part III

PHP-User-Group Stuttgart

08.02.2012

- Datenbank- und SQL-Performance
- Erkennen warum eine SQL-Abfrage langsam ist
- SQL Tipps und Tricks aus der Praxis

SQL – Tipps und Tricks – Part III

Wer bin ich ?

Thomas Wiedmann

- n+1 Jahre Problemlösungen in der Softwareentwicklung
- Seit „vielen“ Jahren Projekte mit PHP und Oracle PL/SQL bzw. DB2/NT
- Zend Certified PHP Engineer (ZCE)
- IBM Certified Solution Expert - DB2 UDB v7.1 Database Administration
- Autor diverser Fachartikel in der „Toolbox“ und im PHP-Magazin
- Autor des Buches „DB2 – SQL, Programmierung, Tuning“ © 2001
- SQL-Tipps, MySQL-EXPLAIN und Performance in der **SQL-Backstube**

SQL – Tipps und Tricks – Part III

Überblick

- Datenbankperformance – Was ist das?
- SQL Anti-Patterns
- Warum ist eine SQL-Abfrage eigentlich langsam?
- DISTINCT, RAND(), GROUP BY – Vorsicht Falle!
- The End!

Datenbankperformance – Was ist das?

```
CREATE TABLE testdaten (
   id INT NOT NULL,
   …
   PRIMARY KEY (id)
);
```

Welche SQL-Abfrage ist schneller bzw. performanter?

a) SELECT id FROM testdaten WHERE id LIKE '4711'

b) SELECT id FROM testdaten WHERE id = 4711

c)

Datenbankperformance – Was ist das?

```
CREATE TABLE testdaten (
   id INT NOT NULL,
   …
   PRIMARY KEY (id)
);
```

Welche SQL-Abfrage ist schneller bzw. performanter?

3.) SELECT id FROM testdaten WHERE id LIKE '4711'

Query opened in 0,362s [0,361s exec, 0,001s fetch]

2.) SELECT id FROM testdaten WHERE id = 4711

Query opened in 0,002s [0,001s exec, 0,001s fetch]

1.) 

SQL – Tipps und Tricks – Part III

Datenbankperformance – Was ist das?

Datenbanken und SQL sind strongly typed !

Query Optimization Theorie, Prioritäten und Basics:

1. Avoid the query **= Vermeide die Abfrage !**
2. Cache the query
3. Simplify the query
4. Optimize the query

```
MySQL Queries Optimization:
"Zitate von Peter Zaitsev,
Percona (c) Zürich Fun tour November 21 2007"
```

SQL Anti-Patterns

Wie prüft man einen Login gegen eine SQL-Datenbank-Tabelle?

```
CREATE TABLE users (
 user_id INT NOT NULL,
 name VARCHAR(50) NOT NULL,
 passwort VARCHAR(50) NOT NULL,
 PRIMARY KEY(user_id)
);
```

Sollte eigentlich klar sein, aber PHP Programmierer waren schon immer sehr creativ...

ACHTUNG!

SQL Anti-Patterns

Wie prüft man einen Login gegen eine SQL-Datenbank-Tabelle?

Gesehen in einem PHP-Forum, mit dem obligatorischen Hinweis „...funzt nich..“

```
<?php
[...]
$sqlbenutzername = 'SELECT name FROM users';
$sqlpasswort = 'SELECT passwort FROM users';
[...]
while($dbbenutzer = mysql_fetch_row($sqlbenutzername))
 while($dbpasswort = mysql_fetch_row($sqlpasswort)){
  for($i = 0; $i < mysql_num_rows($sqlbenutzername); $i++)
    for($j = 0; $j < mysql_num_rows($sqlpasswort); $j++){
      if($benutzername == $dbbenutzer and $pass == $dbpasswort){
        echo '<p>Sie haben sich erfolgreich angemeldet</p>';
        echo '<a href="willkommen.html">Willkommen</a>';
      }
    }
  }
}
[...]
?>
```

SQL Anti-Patterns

Wie prüft man einen Login gegen eine SQL-Datenbank-Tabelle?

Gesehen in einem PHP-Forum, mit dem obligatorischen Hinweis „...funzt nich..“

```
<?php
[...]
$sqlbenutzername = 'SELECT name FROM users';
$sqlpasswort = 'SELECT passwort FROM users';
[...]
while($dbbenutzer = mysql_fetch_row($sqlbenutzername)){
 while($dbpasswort = mysql_fetch_row($sqlpasswort)){
  for($i = 0; $i < mysql_num_rows($sqlbenutzername); $i++)
    for($j = 0; $j < mysql_num_rows($sqlpasswort); $j++){
      if($benutzer[$i] == $name && $pass[$j] == $dbpasswort){
        echo '<p>Sie haben sich erfolgreich angemeldet</p>';
        echo '<a href="willkommen.html">Willkommen</a>';
      }
    }
  }
}
[...]
?>
```

..oder vielleicht doch eher so:

```
SELECT COUNT(*) FROM users
 WHERE name = :sName
   AND passwort = :sPasswort;
```

SQL Anti-Patterns

Wie ermittelt man am Besten die Anzahl Datensätze dieser Tabelle?

```
CREATE TABLE bigtable (
 big_id BIGINT UNSIGNED NOT NULL,
 ...
 PRIMARY KEY(big_id)
);
```

Sollte eigentlich klar sein, aber PHP Programmierer waren schon immer sehr creativ...

ACHTUNG!

SQL Anti-Patterns

Wie ermittelt man am Besten die Anzahl Datensätze dieser Tabelle?

Gesehen in einem PHP-Forum

```
<?php
[...]
 $result = mysql_query('SELECT * FROM bigtable');

 $aData = array();
 while ($aRow = mysql_fetch_array($result)) {
  $aData[] = $aRow;
 }

 echo 'Anzahl Datensätze '.count($aData);
[...]
?>
```

SQL Anti-Patterns

Wie ermittelt man am Besten die Anzahl Datensätze dieser Tabelle?

Gesehen in einem PHP-Forum

```
<?php
[...]
 $result = mysql_query('SELECT * FROM bigtable');

 $aData = array();
 while ($aRow = mysql_fetch_array($result)) {
  $aData[] = $aRow;
 }

 echo 'Anzahl Datensätze '.count($aData);
[...]
?>
```

..oder vielleicht doch eher so:

```
SELECT COUNT(*) FROM bigtable;
```

SQL Anti-Patterns

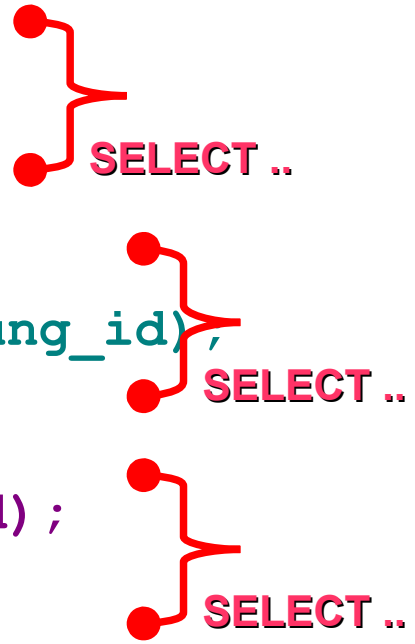
Model/Aktive Record Problematik:

```
$aKunde = Projekt_model_kunde::read($nKunde_id);
if ($aKunde['status']) {
 $nRechnung_id = (int) $_POST['nRechnung_id'];
 $aRechnung = Projekt_model_rechnung::read($nRechnung_id);
 if ($aRechnung['status']) {
  $aRabatt = Projekt_model_rabatt::read($nRabatt_id);
  if ($aRabatt['status']) {
    ... u.s.w. ...
  }
 }
}
```

SQL Anti-Patterns

Model/Aktive Record Problematik:

```
$aKunde = Projekt_model_kunde::read($nKunde_id);
if ($aKunde['status']) {
 $nRechnung_id = (int) $_POST['nRechnung_id'];
 $aRechnung = Projekt_model_rechnung::read($nRechnung_id);
 if ($aRechnung['status']) {
  $aRabatt = Projekt_model_rabatt::read($nRabatt_id);
  if ($aRabatt['status']) {
     ... u.s.w. ...
  }
 }
}
```



SQL Anti-Patterns

Model/Aktive Record Problematik:

```
$aKunde = Projekt_model_kunde::read($nKunde_id);
if ($aKunde['status']) {
 $nRechnung_id = (int) $_POST['nRechnung_id'];
 $aRechnung = Projekt_model_rechnung::read($nRechnung_id);
 if ($aRechnung['status']) {
  $aRabatt = Projekt_model_rabatt::read($nRabatt_id);
  if ($aRabatt['status']) {
    ... u.s.w. ...
  }
 }
}
```

SELECT ..

SELECT ..

SELECT ..

..oder vielleicht doch eher so:

```
SELECT k.*, r.*, ra.*
  FROM kunde k
  JOIN rechnung r
    ON k.kunde_id r.kunde_id
  JOIN rabatt ra
    ON ra.rabbatt_id = …
 WHERE …
```

SQL – Tipps und Tricks – Part III

SQL Anti-Patterns

Wesentliche Hinweise für gesundes SQL- und Datenbanken

1. Know-how
2. g'wusst wie
3. savoir-faire
4. 
5. SQL Backstube (http://www.twiedmann.de/sqlbackstube/feedrss20)

Was macht eine SQL-Abfrage eigentlich langsam?

```
select t1.col1, t2.col2,
 count(t3.id) AS id from tabelle1 t1 join
   tabelle2   t2 on t2.id = t1.id join tabelle3 t3 on
 t3.id = t2.id where  t2.col2 = 4711
and    t1.col1 > 10  group
  by  t1.col1, t2.col2 order
by t3.id
 Limit 100000 , 10;
```

Was macht eine SQL-Abfrage eigentlich langsam?

```
SELECT t1.col1, t2.col2, COUNT(t3.id) AS id
  FROM tabelle1 t1
  JOIN tabelle2 t2
    ON t2.id = t1.id
  JOIN tabelle3 t3
    ON t3.id = t2.id
 WHERE t2.col2 = 4711
   AND t1.col1 > 10
 GROUP BY t1.col1, t2.col2
 ORDER BY t3.id
 LIMIT 100000,10;
```

SQL Formatter
http://www.dpriver.com/pp/sqlformat.htm

Was macht eine SQL-Abfrage eigentlich langsam?

```
SELECT t1.col1, t2.col2, COUNT(t3.id) AS id
  FROM tabelle1 t1
  JOIN tabelle2 t2
    ON t2.id = t1.id
  JOIN tabelle3 t3
    ON t3.id = t2.id
 WHERE t2.col2 = 4711
   AND t1.col1 > 10
 GROUP BY t1.col1, t2.col2
 ORDER BY t3.id
 LIMIT 100000,10;
```

Index
auf beide IDs ?!?

Index
auf beide IDs ?!?

Was macht eine SQL-Abfrage eigentlich langsam?

```
SELECT t1.col1, t2.col2, COUNT(t3.id) AS id
  FROM tabelle1 t1
  JOIN tabelle2 t2
    ON t2.id = t1.id
  JOIN tabelle3 t3
    ON t3.id = t2.id
 WHERE t2.col2 = 4711
   AND t1.col1 > 10
 GROUP BY t1.col1, t2.col2
 ORDER BY t3.id
 LIMIT 100000,10;
```

kein „guter“ Index möglich!!

Order nach anderer Spalte als GROUP?!?

Order muss komplett ausgeführt werden,
dann der komplette Resultset durchsucht werden,
um 10 Datensätze auszugeben?!?

SQL – Tipps und Tricks – Part III

Was macht eine SQL-Abfrage eigentlich langsam?

Immer dann wenn Daten umsortiert oder durchsucht werden müssen (Tablescan), dauert es mal wieder länger.

LIMIT 10000,10 meiden, besser Sub-Select wenn möglich.
(Pagination via OFFSET -- inefficient.)

Grosser RAM und Hardwarepower bringt viel, aber auch mit fundiertem „Performance-Know-How“ läßt sich sehr gutes SQL-Tuning erreichen.

MySQL ab 5.6.3 rüstet auf und spendiert der neuen Version einige Lösungen zu den oben genannten Problemen:

Verbesserungen an MySQLs Optimierer (http://heise.de/-1354352 und http://blogs.oracle.com/MySQL/entry/more_early_access_features_in)

Was macht eigentlich DISTINCT?

```
CREATE TABLE rechnung (
   rechnung_id INT NOT NULL,
   kunde_id INT NOT NULL,
   PRIMARY KEY(rechnung_id)
);

INSERT INTO rechnung VALUES
( 1, 10), (2, 10), (3, 20), (4, 20);

SELECT DISTINCT * FROM rechnung;
```

DISTINCT wird gerne verwendet, wenn in einem JOIN blöder weise doppelte Datensätze raus kommen, man nicht versteht warum und nun irgendwie das Ergebnis „platt“ klopfen möchte.

ABER ...

Was macht eigentlich DISTINCT?

```
SELECT DISTINCT * FROM rechnung;
+--------------+-----------+
| rechnung_id | kunde_id |
+--------------+-----------+
|            1 |        10 |
|            2 |        10 |
|            3 |        20 |
|            4 |        20 |
+--------------+-----------+
4 rows in set (0.00 sec)
```

mehr „Rechenaufwand“ aber gleiches Ergebnis wie SELECT * ..

```
SELECT * FROM rechnung;
+--------------+-----------+
| rechnung_id | kunde_id |
+--------------+-----------+
|            1 |        10 |
|            2 |        10 |
|            3 |        20 |
|            4 |        20 |
+--------------+-----------+
4 rows in set (0.00 sec)
```

Wenn ein Primary Key oder eine andere eindeutige Spalte im DISTINCT enthalten ist, kann dieser die Daten nicht weiter verdichten. MySQL führt den DISTINCT aber trotzdem aus.

Was macht eigentlich DISTINCT?

```
SELECT COUNT(*) FROM testdaten;
+-----------+
| COUNT(*)  |
+-----------+
|  1000000  |
+-----------+
1 row in set (0.34 sec)
```

Bei größeren Datenmengen ist DISTINCT richtig zeitaufwändig.
(Test mit MySQL v.5.5.9)

```
SELECT DISTINCT * FROM testdaten;
[...]
|   100000000 | 4294967295 | 4294967295 |
+-------------+------------+------------+
1000000 rows in set (15.73 sec)
```

```
SELECT * FROM testdaten;
[...]
|   100000000 | 4294967295 | 4294967295 |
+-------------+------------+------------+
1000000 rows in set (0.75 sec)
```

Was macht eigentlich DISTINCT?

```
SELECT DISTINCT * FROM rechnung;
```

..ist identisch zu...

```
SELECT * FROM rechnung
GROUP BY rechnung_id, kunde_id;
+--------------+-----------+
| rechnung_id | kunde_id |
+--------------+-----------+
|            1 |        10 |
|            2 |        10 |
|            3 |        20 |
|            4 |        20 |
+--------------+-----------+
4 rows in set (0.00 sec)
```

DISTINCT ist identisch mit einem GROUP BY über alle beteiligten Spalten

SQL – Tipps und Tricks – Part III

Was macht eigentlich DISTINCT?

Mit DISTINCT werden identische Daten(-sätze) eliminiert. Dabei kommt intern GROUP BY zum Einsatz.

Sinnvolle Beispiele wären:

```
SELECT DISTINCT col2, col3 FROM tabelle1;
SELECT COUNT(DISTINCT col4) FROM tabelle1;
```

Wenig sinnvolle Beispiele wären:

```
SELECT DISTINCT * FROM tabelle1;
SELECT DISTINCT meine_pk_id FROM tabelle1;
```

Zufälligen Datensatz auswählen mit RAND()?

In vielen Online-Shops etc. sollen dem Besucher immer wieder zufällige Artikel oder Waren aus einem Pool von bestimmten Top-Artikel angeboten werden. Dafür finden sich viele Lösungsansätze, wie z. B.

```
SELECT * FROM testdaten
 ORDER BY RAND()
LIMIT 1;
```

RAND() ist nicht als perfekter Zufallsgenerator gedacht, sondern stellt eine gute Möglichkeit dar, ad hoc Zufallszahlen zu Funktionen für die Benutzung in SELECT- und WHERE Klauseln zu erzeugen, die innerhalb derselben MySQL-Version plattformübergreifend funktionieren. © MySQL Handbuch

Aber ...

Zufälligen Datensatz auswählen mit RAND()?

Aber Vorsicht:

Bei einer großen Datenmenge (1 Mio. Sätze) dauert diese Abfrage schon so seine Zeit und allgemein länger als man glaubt.

```
SELECT * FROM testdaten
 ORDER BY RAND()
LIMIT 1;
Query opened in 0,721s [0,717s exec, 0,004s fetch]
```

Es empfiehlt sich also unbedingt, den RAND() über eine kleine "Schlüssel" Tabelle laufen zu lassen. In einem 2. Schritt damit die zufällige ID zu ermitteln und erst mit dieser gefundenen ID den kompletten Datensatz zu lesen.

Zufälligen Datensatz auswählen mit RAND()?

RAND() ist zusammen mit ORDER BY eine MySQL-Speziallösung

```
SELECT * FROM testdaten
 ORDER BY RAND()
LIMIT 1;
+-------------+------------+-------------+
|    86131500 | 4294967295 | 4294967295 |
+-------------+------------+-------------+

SELECT RAND();
+---------------------+
| 0.745993050510057 |
+---------------------+

SELECT * FROM testdaten
 ORDER BY 0.745993050510057
LIMIT 1;
+-------------+----------+-----------+
|         100 |    11100 |     22200 |   <== 1. Row der Tabelle
+-------------+----------+-----------+
```

Zufälligen Datensatz auswählen mit RAND()?

RAND() ist zusammen mit ORDER BY eine MySQL-Speziallösung

```
SELECT * FROM testdaten
 ORDER BY 0.2
LIMIT 1;
+-------------+----------+-----------+
|         100 |    11100 |     22200 |
+-------------+----------+-----------+

SELECT * FROM testdaten
 ORDER BY 0
LIMIT 1;
ERROR 1054 (42S22): Unknown column '0' in 'order clause'

SELECT * FROM testdaten
 ORDER BY 1
LIMIT 1;
+-------------+----------+-----------
|         100 |    11100 |     22200
+-------------+----------+-----------
```

SQL-Standard:
Es wird die Spaltenposition aus der SELECT-Liste angegeben, nach der sortiert werde soll

Zufälligen Datensatz auswählen mit RAND()?

Übliche MySQL Lösung

```
SELECT * FROM testdaten
 ORDER BY RAND()
LIMIT 1;
Query opened in 0,721s [0,717s exec, 0,004s fetch]
```

Einfache Alternativlösung (geringfügig schneller)

```
SELECT col1, col2, col3, RAND()
  FROM testdaten
 ORDER BY 4
LIMIT 1;
Query opened in 0,673s [0,672s exec, 0,001s fetch]
```

Zufälligen Datensatz auswählen mit RAND()?

Mein Lösungvorschlag: Funktioniert auch wenn die ID nicht lückenlos ist.

```
SELECT t.*
  FROM testdaten t
  JOIN ( SELECT FLOOR( MIN(id) +
                       RAND() * MAX(id) +1
                     ) AS id
           FROM testdaten
       ) t
    ON g.id BETWEEN t.id AND t.id + 200
LIMIT 1;
Query opened in 0,268s [0,267s exec, 0,001s fetch]
```

Diese Abfrage ermittelt via SUB-Select zuerst einen kleinen zufälligen ID-Bereich und erst dann wird der komplette Datensatz gelesen. Dies braucht mit 0,268 Sekunden nur ein 1/3 der Zeit.

SQL – Tipps und Tricks – Part III

Zufälligen Datensatz auswählen mit RAND()?

Mit **ORDER BY RAND()** liefert MySQL eine spezielle Syntax, um einen zufälligen Datensatz zu finden.

Aber Vorsicht, um damit eine performante Abfrage zu erreichen, sollten möglichst nur wenige Datensätze im Spiel sein.

Es lohnt sich also eine spezielle kleine „ARTIKEL POOL“ Tabelle zu erstellen und nur darüber den ORDER BY RAND() auszuführen.

Zufälligen Lagerbestand ermitteln mit GROUP BY?

Zufällige Datensätze zu ermitteln kann erwünscht sein, aber einen zufälligen Lagerbestand zu berechnen ist eher nicht erwünscht.

Hierzu ein tückisches Beispiel aus einem MySQL Forum!

```
CREATE TABLE artikel (
 artikel_id INT NOT NULL,
 bezeichnung VARCHAR(20) NOT NULL,
 gruppe_id INT NOT NULL,
 PRIMARY KEY (artikel_id)
);

CREATE TABLE artikel_lager (
 lager_id INT NOT NULL,
 artikel_id INT NOT NULL,
 menge DEC(10,2) NOT NULL,
 PRIMARY KEY (lager_id),

 CONSTRAINT fk_artikel_lager
  FOREIGN KEY (artikel_id)
 REFERENCES artikel (artikel_id)
);
```

Zufälligen Lagerbestand ermitteln mit GROUP BY?

Nun noch den Datenbestand einfügen und korrekt berechnen.

```
INSERT INTO artikel VALUES
( 1, 'Druckerpapier', 1),
( 2, 'Toner', 2),
( 3, 'Briefumschlag', 1);

INSERT INTO artikel_lager VALUES
( 1, 1, 10 ), ( 2, 2, 20 ), ( 3, 3, 30 ),
( 4, 1, 10 ), ( 5, 2, 20 ), ( 6, 3, 30 ),
( 7, 1, 10 ), ( 8, 2, 20 ), ( 9, 3, 30 );
```

```
+------------+------------+
| gruppe_id | auf_lager |
+------------+------------+
|          1 |     120.00 |
|          2 |      60.00 |
+------------+------------+
2 rows in set (0.00 sec)
```

Zufälligen Lagerbestand ermitteln mit GROUP BY?

Hierzu folgende Abfrage – **Was läuft hier falsch?**

```
SELECT a.gruppe_id,
       ( SELECT SUM(l.menge)
           FROM artikel_lager l
          WHERE l.artikel_id = a.artikel_id
       ) AS auf_lager
  FROM artikel a
GROUP BY a.gruppe_id
ORDER BY a.gruppe_id;
+------------+------------+
| gruppe_id | auf_lager |
+------------+------------+
|          1 |      30.00 |  <== sollten 120 (?)
|          2 |      60.00 |
+------------+------------+
2 rows in set (0.00 sec)
```

Zufälligen Lagerbestand ermitteln mit GROUP BY?

Hierzu folgende Abfrage – **Was läuft hier falsch? - GROUP BY !!**

```
SELECT a.gruppe_id,
       SUM(
            ( SELECT SUM(l.menge)
              FROM artikel_lager l
             WHERE l.artikel_id = a.artikel_id
            )
          ) AS auf_lager
  FROM artikel a
GROUP BY a.gruppe_id
ORDER BY a.gruppe_id;
+------------+------------+
| gruppe_id | auf_lager |
+------------+------------+
|          1 |     120.00 |
|          2 |      60.00 |
+------------+------------+
2 rows in set (0.00 sec)
```

SQL – Tipps und Tricks – Part III

MySQL - SQL Modus und GROUP BY

Möglichkeiten MySQL „zu zwingen“ eher ANSI-SQL zu verarbeiten.
(durch Anpassen des SQL-Modus des Servers)

- **ONLY_FULL_GROUP_BY**

Nur Spalten der GROUP BY-Klausel dürfen direkt in der Select-Liste stehen. Für alle anderen müssen Aggregatfunktionen wie MIN(), MAX(), COUNT(), SUM() etc. angewendet werden.

- **ANSI**

Ändert Syntax und Verhalten so, dass eine höhere Kompatibilität mit Standard-SQL erzielt wird. ANSI enthält folgende Modi:
REAL_AS_FLOAT, PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE

mehr Details dazu siehe „5.2.5. Der SQL-Modus des Servers“

SQL – Tipps und Tricks – Part III

MySQL - SQL Modus und GROUP BY

GROUP BY und SQL-Standard

- MySQL erlaubt „falsche“ GROUP BY Konstruktionen
  http://dev.mysql.com/doc/refman/5.0/en/group-by-hidden-columns.html
- SELECT @@global.sql_mode; liefert den Server SQL-Modus
- SELECT @@session.sql_mode; liefert den session SQL-Modus
- sql_mode = 'ONLY_FULL_GROUP_BY' kennen und nutzen
- Mehr zu Server-Modus siehe „5.2.5. Der SQL-Modus des Servers“

SQL – Tipps und Tricks – Part III

MySQL - SQL Modus und GROUP BY

GROUP BY und SQL-Standard

- MySQL erlaubt „falsche“ GROUP BY Konstruktionen
  http://dev.mysql.com/doc/refman/5.0/en/group-by-hidden-columns.html
- SELECT @@global.sql_mode; liefert den Server SQL-Modus
- SELECT @@session.sql_mode; liefert den session SQL-Modus
- sql_mode = 'ONLY_FULL_GROUP_BY' kennen und nutzen
- Mehr zu Server-Modus siehe „5.2.5. Der SQL-Modus des Servers“

The End!