Бесплатная электронная книга

УЧУСЬ MySQL

Free unaffiliated eBook created from
Stack Overflow contributors.

#mysql

1
1: MySQL 2
2
2
Examples 3
3
7
7
8
2: ALTER TABLE 9
9
9
Examples 10
; ; file_per_table 10
ALTER COLUMN OF TABLE 10
ALTER table INDEX 11
11
11
11
MySQL 12
MySQL 12
MySQL 13
MySQL 13
3: ENUM 15
Examples 15
ENUM? 15
TINYINT 15
VARCHAR 16
16
NULL vs NOT NULL 16
4: JSON 18

18

18

Examples 18

JSON 18

JSON 18

JSON 18

JSON 19

CAST JSON 19

Json 19

5: MySQL LOCK TABLE 21

21

21

Examples 21

Mysql 21

23

6: MySQL 25

25

25

Examples 25

25

26

26

7: mysqlimport 28

28

28

Examples 28

28

- 29

29

29

30

csv 30

8: TRIGGERS 31
..... 31
..... 31
..... 31
..... 31
Examples 32
..... 32
..... 32
..... 32
..... 33
Insert 33
..... 33
..... 33
9: UNION 35
..... 35
..... 35
Examples 35
SELECT UNION 35
..... 35
OFFSET 36
..... 36
UNION ALL UNION 36
MySQL 37
10: MySQL 38
Examples 38
root 38
Drop 38
Atomic RENAME & Table Reload 38
11: 39
..... 39
Examples 39
.....

39
BIGINT 39
DOUBLE 40
40
40
(SIN, COS) 40
40
40
41
Arc Cosine ( ) 41
( ) 41
( ) 41
41
42
(ROUND, FLOOR, CEIL) 42
42
42
42
43
(POW) 43
(SQRT) 43
(RAND) 43
43
44
(ABS, SIGN) 44
12: 46
Examples 46
46
(@ host) 46
13: MySQL 5.7+ 48
48
48

Examples....48
....48
root 48
root, «/ var / run / mysqld» UNIX »....49
14: root....52
Examples....52
root, root http....52
15: 53
Examples....53
....53
....53
16: 55
....55
Examples....55
....55
, Javascript....55
- 56
/ 56
Javascript TIMESTAMP....56
17: 58
....58
....58
Examples....59
....59
INSERT, ON DUPLICATE KEY UPDATE....59
....59
....60
INSERT SELECT ( )....61
INSERT AUTO_INCREMENT + LAST_INSERT_ID ()....61
AUTO_INCREMENT....63
18: 65
....65

65

65

Examples 65

SELECT 65

(*) 66

67

SELECT WHERE 67

SELECT LIKE (%) 68

SELECT (AS) 69

SELECT LIMIT 69

DISTINCT 70

SELECT LIKE (_) 71

SELECT CASE IF 72

72

SELECT 73

19: 75

75

75

75

Examples 75

76

MIN 76

COUNT 76

76

Group Concat 76

GROUP BY AGGREGATE 77

20: MyISAM 81

81

Examples 81

= MyISAM 81

21: 82

Examples 82
Un-pivot 82
22: 85
85
Examples 85
LOAD DATA INFILE 85
CSV- MySQL 86
86
86
LOAD DATA INFILE 'fname' REPLACE 86
LOAD DATA INFILE 'fname' IGNORE 87
87
87
23: 88
88
88
Examples 93
- 93
24: JSON 95
95
95
95
95
Examples 96
JSON Array 96
JSON 96
25: 98
Examples 98
MySQL Linux 98
MySQL Windows 99
99

26: 101
..... 101
..... 101
..... 101
Examples 102
..... 102
..... 102
..... 102
..... 102
AUTO_INCREMENT 103
27: 104
..... 104
Examples 104
SHOW VARIABLES 104
SHOW STATUS 105
28: 106
Examples 106
..... 106
. Select Statement 107
29: 109
Examples 109
Disambiguation 109
30: MySQL 110
..... 110
..... 110
Examples 110
..... 111
..... 111
..... 111
: 112
..... 112

31: 113
Examples 113
1064: 113
1175: 113
1215: 114
1045 115
1236 « » 115
2002, 2003 116
1067, 1292, 1366, 1411 - , , ... 116
126, 127, 134, 144, 145 117
139 117
1366 117
126, 1054, 1146, 1062, 24 117
32: Mysql 120
..... 120
Examples 120
..... 120
..... 120
33: 122
..... 122
Examples 122
InnoDB 122
, ... 123
group_concat 123
InnoDB 123
MySQL 124
34: 125
..... 125
Examples 125
- 125
..... 128
35: - 130

Examples 130
130
130
131
133
36: 135
Examples 135
135
135
? 135
136
37: PS1 137
Examples 137
MySQL PS1 137
PS1 MySQL 137
38: SSL 138
Examples 138
Debian 138
CA SSL 138
MySQL 138
SSL- 139
SSL 140
: 140
CentOS7 / RHEL7 140
dbserver 141
. 142
143
144
. 144
39: 146
146

146

Examples....146

....146

....147

....147

....147

....148

....148

....148

....149

+ 149

40: **151**

Examples....151

NULL....151

NULL....151

41: **152**

....152

Examples....152

....152

....152

....152

....153

UPDATE ORDER BY LIMIT....153

UPDATE....154

....154

42: **156**

....156

Examples....156

....156

DATE DATETIME 157

TIMESTAMP 157

?....157

time_zone ? 158
43: 159
Examples 159
Backticks 159
44: 161
161
161
Examples 161
161
LIMIT 161
LIMIT 162
OFFSET : 163
45: 164
164
164
Examples 164
164
, 165
165
46: 167
Examples 167
() 167
167
168
SYSDATE (), NOW (), CURDATE () 168
DateTime 168
169
47: 1055: ONLY_FULL_GROUP_BY: - GROUP BY ... 170
170
170
Examples 171

GROUP BY 171
GROUP BY : 172
GROUP BY SELECT *, ... 172
ANY_VALUE () 173
48: 175
..... 175
..... 175
Examples 175
..... 175
..... 176
49: 177
..... 177
Examples 177
, ... 177
..... 177
..... 178
50: UTF-8 179
Examples 179
..... 179
PHP 179
51: 181
..... 181
..... 181
Examples 181
FULLTEXT 181
BOOLEAN 181
FULLTEXT 182
52: 183
..... 183
..... 183
..... 183
Examples 184

184
185
VIEW 186
186
53: MyISAM InnoDB 187
Examples 187
187
187
54: : 3 id. 188
Examples 188
3 188
55: 189
189
Examples 189
189
JOIN («») 190
- 190
191
3 193
194
56: 195
Examples 195
, NULL 195
57: 198
198
Examples 198
198
199
HASH 200
58: 202
202
Examples 202

REGEXP / RLIKE 202
^ 202
$ ** 202
REGEXP 203
Regex Contain 203
[] 203
| 203
..... 203
59: mysqldump 205
..... 205
..... 205
..... 206
Examples 206
..... 206
..... 207
..... 207
mysqldump 208
gzipped mysqldump 208
Amazon S3 208
MySQL 209
..... 209
60: 211
..... 211
Examples 211
..... 211
61: 213
Examples 213
..... 213
COMMIT, ROLLBACK AUTOCOMMIT 214
JDBC 217
62: 220
Examples 220

220
220
2 , 1- , 2- 10 220
( ) 221
222
63: Mysql 223
Examples 223
223
InnoDB 223
224
64: 226
226
226
Examples 226
, 226
MyDatabase 228
229
229
65: 231
231
231
Examples 231
231
232
232
( ) 233
233
234
235
CREATE TABLE FROM SELECT 235
236

TimeStamp 237
66: 238
..... 238
Examples 238
MySQL 238
..... 238
..... 239
..... 239
67: 240
Examples 240
..... 240
..... 240
, ... 240
..... 241
68: ( ) 242
..... 242
..... 242
Examples 242
..... 242
..... 243
IN, OUT, INOUT 244
..... 245
ResultSets 247
..... 247
69: 248
..... 248
Examples 250
, ... 251
STR_TO_DATE - 251
LOWER () / LCASE () 251
REPLACE () 251
SUBSTRING () 251

UPPER () / UCASE () 252
() 252
CHAR_LENGTH () 252
HEX () 252
70: -- 254
254
Examples 254
254
71: 255
Examples 255
/ 255
VARCHAR (255) - 255
INT AUTO_INCREMENT 256
256
() 257
257
258
258
258
259
() 259
DATE, DATETIME, TIMESTAMP, YEAR TIME 260
72: 261
261
261
Examples 261
Where 261
262
262
262
264
264

264

Multi-table DELETE 265

73: Mysql Docker-Compose 266

Examples 266

..... 266

..... **268**

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: mysql

It is an unofficial and free MySQL ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official MySQL.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с MySQL

замечания

MySQL - это система управления реляционными базами данных с открытым исходным кодом (RDBMS), которая разработана и поддерживается корпорацией Oracle.

MySQL поддерживается на большом количестве платформ, включая версии Linux, OS X и Windows. Он также имеет API для большого количества языков, включая C, C ++, Java, Lua, .Net, Perl, PHP, Python и Ruby.

MariaDB - это вилка MySQL со слегка отличающимся набором функций . Он полностью совместим с MySQL для большинства приложений.

Версии

Версия	Дата выхода
1,0	1995-05-23
3,19	1996-12-01
3,20	1997-01-01
3,21	1998-10-01
3,22	1999-10-01
3,23	2001-01-22
4,0	2003-03-01
4,1	2004-10-01
5.0	2005-10-01
5,1	2008-11-27
5,5	2010-11-01

Версия	Дата выхода
5,6	2013-02-01
5,7	2015-10-01

Examples

Начиная

Создание базы данных в MySQL

```
CREATE DATABASE mydb;
```

Возвращаемое значение:

> Запрос OK, 1 строка затронута (0,05 сек)

Использование созданной базы данных `mydb`

```
USE mydb;
```

Возвращаемое значение:

> Изменена база данных

Создание таблицы в MySQL

```
CREATE TABLE mytable
(
  id              int unsigned NOT NULL auto_increment,
  username        varchar(100) NOT NULL,
  email           varchar(100) NOT NULL,
  PRIMARY KEY     (id)
);
```

`CREATE TABLE mytable` создаст новую таблицу под названием `mytable` .

`id int unsigned NOT NULL auto_increment` создает столбец `id` , этот тип поля присваивает уникальной числовой идентификатор каждой записи в таблице (это означает, что в этом случае две строки не могут иметь одинаковый `id`), MySQL автоматически назначит новый, уникальное значение для поля `id` записи (начиная с 1).

Возвращаемое значение:

> Запрос OK, 0 строк затронуты (0,10 сек)

Вставка строки в таблицу MySQL

```
INSERT INTO mytable ( username, email )
VALUES ( "myuser", "myuser@example.com" );
```

Пример возвращаемого значения:

> Запрос OK, 1 строка затронута (0,06 сек)

Также можно вставлять `strings varchar` aka с помощью одинарных кавычек:

```
INSERT INTO mytable ( username, email )
VALUES ( 'username', 'username@example.com' );
```

Обновление строки в таблице MySQL

```
UPDATE mytable SET username="myuser" WHERE id=8
```

Пример возвращаемого значения:

> Запрос OK, 1 строка затронута (0,06 сек)

Значение `int` может быть вставлено в запрос без кавычек. Строки и Даты должны быть заключены в одинарные кавычки `'` или двойные кавычки `"` .

Удаление строки в таблицу MySQL

```
DELETE FROM mytable WHERE id=8
```

Пример возвращаемого значения:

> Запрос OK, 1 строка затронута (0,06 сек)

Это приведет к удалению строки с `id` 8.

Выбор строк на основе условий в MySQL

```
SELECT * FROM mytable WHERE username = "myuser";
```

Возвращаемое значение:

```
+----+----------+---------------------+
| id | username | email               |
+----+----------+---------------------+
|  1 | myuser   | myuser@example.com  |
+----+----------+---------------------+
```

1 строка в наборе (0,00 сек)

Показать список существующих баз данных

```
SHOW databases;
```

Возвращаемое значение:

```
+--------------------+
| Databases          |
+--------------------+
| information_schema|
| mydb               |
+--------------------+
```

2 ряда в наборе (0,00 сек)

Вы можете думать о «information_schema» как о «основной базе данных», которая обеспечивает доступ к метаданным базы данных.

Показывать таблицы в существующей базе данных

```
SHOW tables;
```

Возвращаемое значение:

```
+-----------------+
| Tables_in_mydb |
+-----------------+
| mytable         |
+-----------------+
```

1 строка в наборе (0,00 сек)

Показать все поля таблицы

```
DESCRIBE databaseName.tableName;
```

или, если уже использовать базу данных:

```
DESCRIBE tableName;
```

Возвращаемое значение:

```
+-----------+-----------------+--------+---------+--------------------+-------+
| Field     | Type            | Null   | Key     | Default            | Extra |
```

```
+-----------+----------------+--------+---------+-------------------+-------+
| fieldname | fieldvaluetype | NO/YES | keytype | defaultfieldvalue |       |
+-----------+----------------+--------+---------+-------------------+-------+
```

Например, `Extra` может содержать `auto_increment` .

`Key` относится к типу ключа, который может повлиять на поле. Первичный (PRI), уникальный (UNI) ...

    n строка в наборе (0,00 сек)

Где n - количество полей в таблице.

Создание пользователя

Во-первых, вам нужно создать пользователя, а затем предоставить права пользователя на определенные базы данных / таблицы. При создании пользователя вам также необходимо указать, с какого пользователя можно подключиться.

```
CREATE USER 'user'@'localhost' IDENTIFIED BY 'some_password';
```

Создает пользователя, который может подключаться только к локальному компьютеру, на котором размещается база данных.

```
CREATE USER 'user'@'%' IDENTIFIED BY 'some_password';
```

Создает пользователя, который может подключаться из любого места (кроме локальной машины).

Пример возвращаемого значения:

    Запрос OK, 0 строк, затронутых (0.00 сек)

Добавление привилегий

Предоставьте общие, основные привилегии пользователю для всех таблиц указанной базы данных:

```
GRANT SELECT, INSERT, UPDATE ON databaseName.* TO 'userName'@'localhost';
```

Предоставьте всем привилегии пользователю для всех таблиц во всех базах данных (внимание с этим):

```
GRANT ALL ON *.* TO 'userName'@'localhost' WITH GRANT OPTION;
```

Как показано выше, `*.*` Предназначено для всех баз данных и таблиц, `databaseName.*`

Предназначено для всех таблиц конкретной базы данных. Также можно указать базу данных и таблицу, например, имя `databaseName.tableName` .

`WITH GRANT OPTION` следует исключить, если пользователь не может предоставить права других пользователей.

Привилегии могут быть **либо**

```
ALL
```

или комбинацию из следующего, каждая из которых разделена запятой (неисчерпывающий список).

```
SELECT
INSERT
UPDATE
DELETE
CREATE
DROP
```

Заметка

Как правило, вам следует избегать использования имен столбцов или таблиц, содержащих пробелы, или использования зарезервированных слов в SQL. Например, лучше избегать имен, таких как `table` или `first name` .

Если вы должны использовать такие имена, поместите их между разделителями `` back-tick``. Например:

```
CREATE TABLE `table`
(
   `first name` VARCHAR(30)
);
```

Запрос, содержащий разделители обратного тика в этой таблице, может быть:

```
 SELECT `first name` FROM `table` WHERE `first name` LIKE 'a%';
```

Примеры информационных схем

Список процессов

Это покажет все активные и спальные запросы в этом порядке, а затем как долго.

```
SELECT * FROM information_schema.PROCESSLIST ORDER BY INFO DESC, TIME DESC;
```

Это немного более подробная информация о временных рамах, поскольку она находится в секундах по умолчанию

```
SELECT ID, USER, HOST, DB, COMMAND,
TIME as time_seconds,
ROUND(TIME / 60, 2) as time_minutes,
ROUND(TIME / 60 / 60, 2) as time_hours,
STATE, INFO
FROM information_schema.PROCESSLIST ORDER BY INFO DESC, TIME DESC;
```

Поиск сохраненной процедуры

Легко выполнять поиск по всем `Stored Procedures` для слов и подстановочных знаков.

```
SELECT * FROM information_schema.ROUTINES WHERE ROUTINE_DEFINITION LIKE '%word%';
```

Прочитайте Начало работы с MySQL онлайн: https://riptutorial.com/ru/mysql/topic/302/начало-работы-с-mysql

глава 2: ALTER TABLE

Синтаксис

- ALTER [IGNORE] TABLE tbl_name [**alter_specification** [, alter_specification] ...] [partition_options]

замечания

```
    alter_specification: table_options
      | ADD [COLUMN] col_name column_definition [FIRST | AFTER col_name ]
      | ADD [COLUMN] (col_name column_definition,...)
      | ADD {INDEX|KEY} [index_name] [index_type] (index_col_name,...) [index_option] ...
      | ADD [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...) [index_option]
...
      | ADD [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY] [index_name] [index_type]
(index_col_name,...) [index_option] ...
      | ADD FULLTEXT [INDEX|KEY] [index_name] (index_col_name,...) [index_option] ...
      | ADD SPATIAL [INDEX|KEY] [index_name] (index_col_name,...) [index_option] ...
      | ADD [CONSTRAINT [symbol]] FOREIGN KEY [index_name] (index_col_name,...)
reference_definition
      | ALGORITHM [=] {DEFAULT|INPLACE|COPY}
      | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
      | CHANGE [COLUMN] old_col_name new_col_name column_definition [FIRST|AFTER col_name]
      | LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
      | MODIFY [COLUMN] col_name column_definition [FIRST | AFTER col_name]
      | DROP [COLUMN] col_name
      | DROP PRIMARY KEY
      | DROP {INDEX|KEY} index_name
      | DROP FOREIGN KEY fk_symbol
      | DISABLE KEYS
      | ENABLE KEYS
      | RENAME [TO|AS] new_tbl_name
      | RENAME {INDEX|KEY} old_index_name TO new_index_name
      | ORDER BY col_name [, col_name] ...
      | CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
      | [DEFAULT] CHARACTER SET [=] charset_name [COLLATE [=] collation_name]
      | DISCARD TABLESPACE
      | IMPORT TABLESPACE
      | FORCE
      | {WITHOUT|WITH} VALIDATION
      | ADD PARTITION (partition_definition)
      | DROP PARTITION partition_names
      | DISCARD PARTITION {partition_names | ALL} TABLESPACE
      | IMPORT PARTITION {partition_names | ALL} TABLESPACE
      | TRUNCATE PARTITION {partition_names | ALL}
      | COALESCE PARTITION number
      | REORGANIZE PARTITION partition_names INTO (partition_definitions)
      | EXCHANGE PARTITION partition_name WITH TABLE tbl_name [{WITH|WITHOUT} VALIDATION]
      | ANALYZE PARTITION {partition_names | ALL}
      | CHECK PARTITION {partition_names | ALL}
      | OPTIMIZE PARTITION {partition_names | ALL}
      | REBUILD PARTITION {partition_names | ALL}
      | REPAIR PARTITION {partition_names | ALL}
      | REMOVE PARTITIONING
```

```
      | UPGRADE PARTITIONING
   index_col_name: col_name [(length)] [ASC | DESC]
   index_type: USING {BTREE | HASH}
   index_option: KEY_BLOCK_SIZE [=] value
      | index_type
      | WITH PARSER parser_name
      | COMMENT 'string'
```

`table_options: table_option [[,] table_option] ...` (see `options`) CREATE TABLE `options`)

`partition_options:` (see `options`) CREATE TABLE `options`)

Ссылка: MySQL 5.7 Справочное руководство / ... / ALTER TABLE Синтаксис / 14.1.8 ALTER TABLE Синтаксис

Examples

Изменение механизма хранения; перестроить таблицу; изменить file_per_table

Например, если `t1` в настоящее время не является таблицей InnoDB, это утверждение изменяет механизм хранения на InnoDB:

```
ALTER TABLE t1 ENGINE = InnoDB;
```

Если таблица уже InnoDB, это приведет к восстановлению таблицы и ее индексов и эффекта, аналогичного `OPTIMIZE TABLE` . Вы можете немного улучшить дисковое пространство.

Если значение `innodb_file_per_table` в настоящее время отличается от значения, действующего при построении `t1` , оно преобразуется в (или из) file_per_table.

ALTER COLUMN OF TABLE

```
CREATE DATABASE stackoverflow;

USE stackoverflow;

Create table stack(
    id_user int NOT NULL,
    username varchar(30) NOT NULL,
    password varchar(30) NOT NULL
);

ALTER TABLE stack ADD COLUMN submit date NOT NULL; -- add new column
ALTER TABLE stack DROP COLUMN submit; -- drop column
ALTER TABLE stack MODIFY submit DATETIME NOT NULL; -- modify type column
ALTER TABLE stack CHANGE submit submit_date DATETIME NOT NULL; -- change type and name of
column
ALTER TABLE stack ADD COLUMN mod_id INT NOT NULL AFTER id_user; -- add new column after
existing column
```

ALTER table добавить INDEX

Чтобы повысить производительность, вы можете добавить индексы в столбцы

```
ALTER TABLE TABLE_NAME ADD INDEX `index_name` (`column_name`)
```

изменение для добавления составных (нескольких столбцов) индексов

```
ALTER TABLE TABLE_NAME ADD INDEX `index_name` (`col1`,`col2`)
```

Изменить значение автоинкремента

Изменение значения автоматического прироста полезно, когда вы не хотите, чтобы пробел в столбце AUTO_INCREMENT после массивного удаления.

Например, у вас появилось много нежелательных (рекламных) строк, размещенных в вашей таблице, вы удалили их, и вы хотите исправить пробел в значениях автоматического увеличения. Предположим, что значение MAX столбца AUTO_INCREMENT равно 100. Вы можете использовать следующее, чтобы исправить значение автоинкремента.

```
ALTER TABLE your_table_name AUTO_INCREMENT = 101;
```

Изменение типа столбца первичного ключа

```
ALTER TABLE fish_data.fish DROP PRIMARY KEY;
ALTER TABLE fish_data.fish MODIFY COLUMN fish_id DECIMAL(20,0) NOT NULL PRIMARY KEY;
```

Попытка изменить тип этого столбца без первоначального удаления первичного ключа приведет к ошибке.

Изменить определение столбца

Изменение определения столбца db может быть использовано, например, для запроса, если у нас есть эта схема db

```
users (
    firstname varchar(20),
    lastname varchar(20),
    age char(2)
)
```

Чтобы изменить тип столбца `age` от `char` до `int` , мы используем следующий запрос:

```
ALTER TABLE users CHANGE age age tinyint UNSIGNED NOT NULL;
```

Общий формат:

```
ALTER TABLE table_name CHANGE column_name new_column_definition
```

Переименование базы данных MySQL

Нет единой команды для переименования базы данных MySQL, но для ее достижения можно использовать простой способ обхода путем резервного копирования и восстановления:

```
mysqladmin -uroot -p<password> create <new name>
mysqldump -uroot -p<password> --routines <old name> | mysql -uroot -pmypassword <new name>
mysqladmin -uroot -p<password> drop <old name>
```

шаги:

1. Скопируйте строки выше в текстовый редактор.
2. Замените все ссылки на `<old name>` , `<new name>` и `<password>` (+ необязательно `root` для использования другого пользователя) с соответствующими значениями.
3. Выполняйте один за другим в командной строке (при условии, что папка «bin» MySQL находится в пути и вводит «y» при появлении запроса).

Альтернативные шаги:

Переименуйте (переместите) каждую таблицу с одной базы данных на другую. Сделайте это для каждой таблицы:

```
RENAME TABLE `<old db>`.`<name>` TO  `<new db>`.`<name>`;
```

Вы можете создавать эти заявления, делая что-то вроде

```
SELECT CONCAT('RENAME TABLE old_db.', table_name, ' TO ',
                           'new_db.', table_name)
    FROM information_schema.TABLES
    WHERE table_schema = 'old_db';
```

Предупреждение. Не пытайтесь делать какие-либо таблицы или базы данных, просто перемещая файлы в файловой системе. Это хорошо работало в старые времена только MyISAM, но в новые дни InnoDB и табличных пространств это не сработает. Особенно, когда «Словарь данных» перемещается из файловой системы в системные таблицы InnoDB, возможно, в следующем крупном выпуске. Перемещение (в отличие от просто `DROPping`) `PARTITION` таблицы InnoDB требует использования «переносных табличных пространств». В ближайшем будущем даже не будет файла.

Обмен именами двух баз данных MySQL

Следующие команды могут использоваться для замены имен двух баз данных MySQL (`<db1>` и `<db2>`):

```
mysqladmin -uroot -p<password> create swaptemp
mysqldump -uroot -p<password> --routines <db1> | mysql -uroot -p<password> swaptemp
mysqladmin -uroot -p<password> drop <db1>
mysqladmin -uroot -p<password> create <db1>
mysqldump -uroot -p<password> --routines <db2> | mysql -uroot -p<password> <db1>
mysqladmin -uroot -p<password> drop <db2>
mysqladmin -uroot -p<password> create <db2>
mysqldump -uroot -p<password> --routines swaptemp | mysql -uroot -p<password> <db2>
mysqladmin -uroot -p<password> drop swaptemp
```

шаги:

1. Скопируйте строки выше в текстовый редактор.
2. Замените все ссылки на `<db1>` , `<db2>` и `<password>` (+ необязательно `root` для использования другого пользователя) с соответствующими значениями.
3. Выполняйте один за другим в командной строке (при условии, что папка «bin» MySQL находится в пути и вводит «y» при появлении запроса).

Переименование таблицы MySQL

Переименование таблицы может выполняться одной командой:

```
RENAME TABLE `<old name>` TO `<new name>`;
```

Следующий синтаксис делает то же самое:

```
ALTER TABLE `<old name>` RENAME TO `<new name>`;
```

Если переименовать временную таблицу, следует использовать версию синтаксиса `ALTER TABLE` .

шаги:

1. Замените `<old name>` и `<new name>` в строке выше соответствующими значениями. *Примечание. Если таблица перемещается в другую базу данных, имя `dbname` . Синтаксис `tablename` можно использовать для `<old name>` и / или `<new name>` .*
2. Выполните его в соответствующей базе данных в командной строке MySQL или в клиенте, таком как MySQL Workbench. *Примечание. Пользователь должен иметь привилегии ALTER и DROP на старой таблице и CREATE и INSERT на новом.*

Переименование столбца в таблице MySQL

Переименование столбца может быть выполнено в одном заявлении, но также как и новое имя, также должно быть указано «определение столбца» (то есть его тип данных и другие

необязательные свойства, такие как обнуление, автоматическое увеличение и т. Д.).

```
ALTER TABLE `<table name>` CHANGE `<old name>` `<new name>` <column definition>;
```

шаги:

1. Откройте командную строку MySQL или клиент, например MySQL Workbench.
2. Выполните следующий оператор: `SHOW CREATE TABLE <table name>;` (заменив `<table name>` на соответствующее значение).
3. Запишите полное определение столбца для столбца, который нужно переименовать *(т.е. все, что появляется после имени столбца, но перед запятой, отделяющей его от следующего имени столбца)* .
4. Замените `<old name>` , `<new name>` и `<column definition>` в строке выше соответствующими значениями и затем выполните их.

Прочитайте ALTER TABLE онлайн: https://riptutorial.com/ru/mysql/topic/2627/alter-table

глава 3: ENUM

Examples

Почему ENUM?

ENUM предоставляет способ предоставления атрибута для строки. Атрибуты с небольшим количеством нечисловых опций работают лучше всего. Примеры:

```
reply ENUM('yes', 'no')
gender ENUM('male', 'female', 'other', 'decline-to-state')
```

Значения: строки:

```
INSERT ... VALUES ('yes', 'female')
SELECT ... --> yes female
```

TINYINT в качестве альтернативы

Допустим, у нас есть

```
type ENUM('fish','mammal','bird')
```

Альтернативой является

```
type TINYINT UNSIGNED
```

плюс

```
CREATE TABLE AnimalTypes (
    type TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(20) NOT NULL  COMMENT "('fish','mammal','bird')",
    PRIMARY KEY(type),
    INDEX(name)
) ENGINE=InnoDB
```

который очень похож на таблицу многих ко многим.

Сравнение и лучше или хуже ENUM:

- (хуже) INSERT: нужно искать `type`
- (хуже) SELECT: нужно подключиться, чтобы получить строку (ENUM дает вам строку без усилий)
- (лучше) Добавление новых типов: просто вставьте в эту таблицу. С ENUM вам нужно сделать ALTER TABLE.

- (тот же) Любой метод (до 255 значений) принимает только 1 байт.
- (смешанный). Также существует проблема целостности данных: `TINYINT` допустит недопустимые значения; тогда как `ENUM` им специальное значение пустой строки (если не включен режим строгого SQL, и в этом случае они отклоняются). Улучшение целостности данных может быть достигнуто с помощью `TINYINT` , сделав его внешним ключом в справочную таблицу: которая с соответствующими запросами / объединениями, но все еще небольшая стоимость достижения другой таблицы. (`FOREIGN KEYs` не являются бесплатными.)

VARCHAR в качестве альтернативы

Допустим, у нас есть

```
type ENUM('fish','mammal','bird')
```

Альтернативой является

```
type VARCHAR(20)  COMENT "fish, bird, etc"
```

Это совершенно открыто, поскольку новые типы тривиально добавляются.

Сравнение и лучше или хуже ENUM:

- (то же самое) INSERT: просто укажите строку
- (хуже?) В INSERT опечатка останется незамеченной
- (тот же самый) SELECT: возвращается фактическая строка
- (хуже) Больше места потребляется

Добавление новой опции

```
ALTER TABLE tbl MODIFY COLUMN type ENUM('fish','mammal','bird','insect');
```

Заметки

- Как и во всех случаях MODIFY COLUMN, вы должны включить `NOT NULL` и любые другие квалификаторы, которые изначально существовали, иначе они будут потеряны.
- *Если* вы добавите в *конец* списка, *а* список будет содержать не более 256 элементов, `ALTER` будет выполняться путем простого изменения схемы. То есть не будет длинной копии таблицы. (У старых версий MySQL такой оптимизации нет).

NULL vs NOT NULL

Примеры того, что происходит, когда NULL и «bad-value» хранятся в столбцах с возможностью NULL, а не с NULL. Также показано использование литья в числовое

значение с помощью `+0` .

```
CREATE TABLE enum (
    e     ENUM('yes', 'no')   NOT NULL,
    enull ENUM('x', 'y', 'z')     NULL
        );
INSERT INTO enum (e, enull)
    VALUES
        ('yes', 'x'),
        ('no',  'y'),
        (NULL,  NULL),
        ('bad-value', 'bad-value');
Query OK, 4 rows affected, 3 warnings (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 3

mysql>SHOW WARNINGS;
+---------+------+-------------------------------------------+
| Level   | Code | Message                                   |
+---------+------+-------------------------------------------+
| Warning | 1048 | Column 'e' cannot be null                 |
| Warning | 1265 | Data truncated for column 'e' at row 4    |
| Warning | 1265 | Data truncated for column 'enull' at row 4 |
+---------+------+-------------------------------------------+
3 rows in set (0.00 sec)
```

Что находится в таблице после этих вставок. Это использует «+0» для приведения в числовое значение, чтобы увидеть, что хранится.

```
mysql>SELECT e, e+0 FROM enum;
+-----+-----+
| e   | e+0 |
+-----+-----+
| yes |   1 |
| no  |   2 |
|     |   0 |  -- NULL
|     |   0 |  -- 'bad-value'
+-----+-----+
4 rows in set (0.00 sec)

mysql>SELECT enull, enull+0 FROM enum;
+-------+---------+
| enull | enull+0 |
+-------+---------+
| x     |       1 |
| y     |       2 |
| NULL  |    NULL |
|       |       0 |  -- 'bad-value'
+-------+---------+
4 rows in set (0.00 sec)
```

Прочитайте ENUM онлайн: https://riptutorial.com/ru/mysql/topic/4425/enum

глава 4: JSON

Вступление

Начиная с MySQL 5.7.8, MySQL поддерживает собственный тип данных JSON, который обеспечивает эффективный доступ к данным в документах JSON (JavaScript Object Notation). https://dev.mysql.com/doc/refman/5.7/en/json.html

замечания

Начиная с MySQL 5.7.8, MySQL поставляется с типом JSON. Многие разработчики сохраняют данные JSON в текстовых столбцах за время журнала, но тип JSON отличается, данные сохраняются в двоичном формате после проверки. Это позволяет избежать накладных расходов при анализе текста при каждом чтении.

Examples

Создайте простую таблицу с первичным ключом и полем JSON

```
CREATE TABLE table_name (
    id INT NOT NULL AUTO_INCREMENT,
    json_col JSON,
    PRIMARY KEY(id)
);
```

Вставьте простой JSON

```
INSERT INTO
    table_name (json_col)
VALUES
    ('{"City": "Galle", "Description": "Best damn city in the world"}');
```

Это просто, поскольку можно заметить, что, поскольку ключи словаря JSON должны быть окружены двойными кавычками, вся вещь должна быть обернута в одинарные кавычки. Если запрос будет выполнен успешно, данные будут сохранены в двоичном формате.

Вставьте смешанные данные в поле JSON.

Это вставляет словарь json, где один из членов представляет собой массив строк в таблицу, которая была создана в другом примере.

```
INSERT INTO myjson(dict)
VALUES('{"opening":"Sicilian","variations":["pelikan","dragon","najdorf"]}');
```

Заметим еще раз, что вам нужно быть осторожным с использованием одиночных и двойных кавычек. Все это должно быть завернуто в одинарные кавычки.

Обновление поля JSON

В предыдущем примере мы видели, как смешанные типы данных могут быть вставлены в поле JSON. Что делать, если мы хотим обновить это поле? Мы собираемся добавить *scheveningen* в массив с именами `variations` в предыдущем примере.

```
UPDATE
    myjson
SET
    dict=JSON_ARRAY_APPEND(dict,'$.variations','scheveningen')
WHERE
    id = 2;
```

Заметки:

1. Массив `$.variations` в нашем словаре json. Символ $ представляет собой json-документацию. Для полного объяснения путей json, распознаваемых mysql, обратитесь к https://dev.mysql.com/doc/refman/5.7/en/json-path-syntax.html
2. Поскольку у нас еще нет примера для запросов с использованием json-полей, в этом примере используется первичный ключ.

Теперь, если мы сделаем `SELECT * FROM myjson` мы увидим

```
+----+-------------------------------------------------------------------------------------------
-+
| id | dict
|
+---+--------------------------------------------------------------------------------------------
+
| 2  | {"opening": "Sicilian", "variations": ["pelikan", "dragon", "najdorf", "scheveningen"]}
|
+----+-------------------------------------------------------------------------------------------
-+
1 row in set (0.00 sec)
```

Данные CAST для типа JSON

Это преобразует действительные строки json в тип MySQL JSON:

```
 SELECT CAST('[1,2,3]' as JSON) ;
 SELECT CAST('{"opening":"Sicilian","variations":["pelikan","dragon","najdorf"]}' as JSON);
```

Создать объект Json и массив

`JSON_OBJECT` создает объекты JSON:

```
SELECT JSON_OBJECT('key1',col1 , 'key2',col2 , 'key3','col3') as myobj;
```

`JSON_ARRAY` создает JSON Array:

```
SELECT JSON_ARRAY(col1,col2,'col3') as myarray;
```

Примечание: myobj.key3 и myarray [2] являются «col3» в качестве фиксированной строки.

Также смешанные данные JSON:

```
SELECT JSON_OBJECT("opening","Sicilian",
"variations",JSON_ARRAY("pelikan","dragon","najdorf") ) as mymixed ;
```

Прочитайте JSON онлайн: https://riptutorial.com/ru/mysql/topic/2985/json

глава 5: MySQL LOCK TABLE

Синтаксис

- LOCK TABLES table_name [READ | ЗАПИСЫВАТЬ]; // Стол блокировки
- РАЗБЛОКИРОВАТЬ ТАБЛИЦЫ; // Разблокировать таблицы

замечания

Блокировка используется для решения проблем параллелизма. Блокировка требуется только при выполнении транзакции, которая сначала считывает значение из базы данных и позже записывает это значение в базу данных. Замки никогда не требуются для самостоятельной операции вставки, обновления или удаления.

Доступны два вида замков

READ LOCK - когда пользователь только читает из таблицы.

WRITE LOCK - когда пользователь выполняет чтение и запись в таблицу.

Когда пользователь удерживает `WRITE LOCK` на таблице, другие пользователи не могут читать или писать в эту таблицу. Когда пользователь держит `READ LOCK` на таблице, другие пользователи также могут читать или удерживать `READ LOCK` , но никто не может писать или удерживать `WRITE LOCK` в этой таблице.

Если основным механизмом хранения данных является InnoDB, MySQL автоматически использует блокировку на уровне строк, так что несколько транзакций могут использовать одну и ту же таблицу одновременно для чтения и записи, не заставляя друг друга ждать.

Для всех систем хранения, отличных от InnoDB, MySQL использует блокировку таблиц.

Более подробная информация о блокировке таблиц См. Здесь

Examples

Замки Mysql

Столбцы могут быть важным инструментом для `ENGINE=MyISAM` , но редко используются для `ENGINE=InnoDB` . Если у вас возникли соблазны использовать блокировки таблиц с InnoDB, вы должны пересмотреть, как вы работаете с транзакциями.

MySQL позволяет клиентским сеансам явно получать блокировки таблиц с целью

взаимодействия с другими сеансами для доступа к таблицам или для предотвращения изменения других сеансов в периоды, когда сеанс требует эксклюзивного доступа к ним. Сеанс может приобретать или выпускать блокировки только для себя. Один сеанс не может получить блокировки для другого сеанса или блокировки релиза, проводимые другим сеансом.

Замки могут использоваться для эмуляции транзакций или для увеличения скорости при обновлении таблиц. Это объясняется более подробно ниже в этом разделе.

Команда: `LOCK TABLES table_name READ|WRITE;`

вы можете назначить только тип блокировки для одной таблицы;

Пример (READ LOCK):

```
LOCK TABLES table_name READ;
```

Пример (WRITE LOCK):

```
LOCK TABLES table_name WRITE;
```

Чтобы увидеть блокировку, применяется или нет, используйте следующую команду

```
SHOW OPEN TABLES;
```

Чтобы очистить / удалить все блокировки, используйте следующую команду:

```
UNLOCK TABLES;
```

ПРИМЕР:

```
LOCK TABLES products WRITE:
INSERT INTO products(id,product_name) SELECT id,old_product_name FROM old_products;
UNLOCK TABLES;
```

Выше примера, любое внешнее соединение не может записывать какие-либо данные в таблицу продуктов, пока не будет разблокирован настольный продукт

ПРИМЕР:

```
LOCK TABLES products READ:
INSERT INTO products(id,product_name) SELECT id,old_product_name FROM old_products;
UNLOCK TABLES;
```

Вышеприведенный пример, любое внешнее соединение не может читать данные из таблицы продуктов, пока не будет разблокирован настольный продукт

Блокировка уровня строки

Если в таблицах используется InnoDB, MySQL автоматически использует блокировку на уровне строк, чтобы несколько транзакций могли одновременно использовать одну и ту же таблицу для чтения и записи, не заставляя друг друга ждать.

Если две транзакции, пытающиеся изменить одну и ту же строку, и оба используют блокировку уровня строки, одна из транзакций ждет другого.

Блокировка уровня строк также может быть получена с помощью `SELECT ... FOR UPDATE` для каждой строки, которая, как ожидается, будет изменена.

Рассмотрим два подключения для подробного объяснения блокировки уровня строк

Соединение 1

```
START TRANSACTION;
SELECT ledgerAmount FROM accDetails WHERE id = 1 FOR UPDATE;
```

В соединении 1 блокировка уровня строки, полученная `SELECT ... FOR UPDATE` .

Соединение 2

```
UPDATE accDetails SET ledgerAmount = ledgerAmount + 500 WHERE id=1;
```

Когда кто-то пытается обновить ту же строку в соединении 2, которая будет ждать соединения 1 для завершения транзакции или сообщения об ошибке, будет отображаться в соответствии с параметром `innodb_lock_wait_timeout` , который по умолчанию составляет 50 секунд.

```
Error Code: 1205. Lock wait timeout exceeded; try restarting transaction
```

Чтобы просмотреть сведения об этой блокировке, запустите `SHOW ENGINE INNODB STATUS`

```
---TRANSACTION 1973004, ACTIVE 7 sec updating
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 360, 1 row lock(s)
MySQL thread id 4, OS thread handle 0x7f996beac700, query id 30 localhost root update
UPDATE accDetails SET ledgerAmount = ledgerAmount + 500 WHERE id=1
------- TRX HAS BEEN WAITING 7 SEC FOR THIS LOCK TO BE GRANTED:
```

Соединение 2

```
UPDATE accDetails SET ledgerAmount = ledgerAmount + 250 WHERE id=2;
1 row(s) affected
```

Но при обновлении некоторой другой строки в соединении 2 будет выполняться без какой-

либо ошибки.

Соединение 1

```
UPDATE accDetails SET ledgerAmount = ledgerAmount + 750 WHERE id=1;
COMMIT;
1 row(s) affected
```

Теперь блокировка строки освобождается, поскольку транзакция завершается в Connection 1.

Соединение 2

```
UPDATE accDetails SET ledgerAmount = ledgerAmount + 500 WHERE id=1;
1 row(s) affected
```

Обновление выполняется без каких-либо ошибок в Connection 2 после того, как Connection 1 освободил блокировку строки, завершив транзакцию.

Прочитайте MySQL LOCK TABLE онлайн: https://riptutorial.com/ru/mysql/topic/5233/mysql-lock-table

глава 6: MySQL Союзы

Синтаксис

- SELECT column_name (s) FROM table1 UNION SELECT column_name (s) FROM table2;
- SELECT column_name (s) FROM table1 UNION ALL SELECT column_name (s) FROM table2;
- SELECT column_name (s) FROM table1 WHERE col_name = "XYZ" UNION ALL SELECT column_name (s) FROM table2 WHERE col_name = "XYZ";

замечания

`UNION DISTINCT` - это то же самое, что `UNION` ; он медленнее, чем `UNION ALL` из-за де-дублирования. Хорошая практика заключается в том, чтобы всегда указывать `DISTINCT` или `ALL` , тем самым сигнализируя, что вы думали о том, что делать.

Examples

Союзный оператор

Оператор UNION используется для объединения набора результатов (*только отдельных значений*) двух или более операторов SELECT.

Запрос: (Чтобы выбрать все разные города (*только отдельные значения*) из таблиц «Клиенты» и «Поставщики»)

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

Результат:

```
Number of Records: 10

City
------
Aachen
Albuquerque
Anchorage
Annecy
Barcelona
Barquisimeto
Bend
Bergamo
Berlin
Bern
```

Союз ВСЕ

UNION ALL, чтобы выбрать все (дублирующие значения) города из таблиц «Клиенты» и «Поставщики».

Запрос:

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

Результат:

```
Number of Records: 12

City
-------
Aachen
Albuquerque
Anchorage
Ann Arbor
Annecy
Barcelona
Barquisimeto
Bend
Bergamo
Berlin
Berlin
Bern
```

СОЮЗ ВСЕ С ГДЕ

UNION ALL для выбора всех (повторяющихся значений) также городов Германии из таблиц «Клиенты» и «Поставщики». Здесь `Country="Germany"` указывается в предложении where.

Запрос:

```
SELECT City, Country FROM Customers
WHERE Country='Germany'
UNION ALL
SELECT City, Country FROM Suppliers
WHERE Country='Germany'
ORDER BY City;
```

Результат:

```
Number of Records: 14
```

город	Страна

Aachen	Германия
Берлин	Германия
Берлин	Германия
Бранденбург	Германия
Cunewalde	Германия
Куксхавен	Германия
Франкфурт	Германия
Frankfurt aM	Германия
Köln	Германия
Лейпциг	Германия
Mannheim	Германия
München	Германия
Münster	Германия
Штутгарт	Германия

Прочитайте MySQL Союзы онлайн: https://riptutorial.com/ru/mysql/topic/5376/mysql-союзы

глава 7: mysqlimport

параметры

параметр	Описание
`--delete -D`	очистите таблицу перед импортом текстового файла
`--fields-optionally-enclosed-by`	определить символ, который цитирует поля
`--fields-terminated-by`	полевой ограничитель
`--ignore -i`	игнорировать проглотившуюся строку в случае дубликатов ключей
`--lines-terminated-by`	определить ограничитель строки
`--password -p`	пароль
`--port -P`	порт
`--replace -r`	перезаписать старую строку ввода в случае дубликатов ключей
`--user -u`	имя пользователя
`--where -w`	указать условие

замечания

`mysqlimport` будет использовать имя импортируемого файла после удаления расширения, чтобы определить таблицу назначения.

Examples

Основное использование

С учетом файла, разделенного вкладкой `employee.txt`

1 `\t` Артур Дент
2 `\t` Марвин
3 `\t` Зафод Библброкс

```
$ mysql --user=user --password=password mycompany -e 'CREATE TABLE employee(id INT, name
VARCHAR(100), PRIMARY KEY (id))'

$ mysqlimport --user=user --password=password mycompany employee.txt
```

Использование настраиваемого поля-разделителя

Учитывая текстовый файл employee.txt

1 | Артур Дент
2 | Marvin
3 | Зафод Библброкс

```
$ mysqlimport --fields-terminated-by='|' mycompany employee.txt
```

Использование настраиваемого разделителя строк

Этот пример полезен для оконных окон:

```
$ mysqlimport --lines-terminated-by='\r\n' mycompany employee.txt
```

Обработка дубликатов ключей

Учитывая таблицу `Employee`

Я бы	название
3	Yooden Vranx

И файл `employee.txt`

1 `\t` Артур Дент
2 `\t` Марвин
3 `\t` Зафод Библброкс

Параметр `--ignore` игнорирует запись с дублирующимися ключами

```
$ mysqlimport --ignore mycompany employee.txt
```

Я бы	название
1	Артур Дент
2	Marvin
3	Yooden Vranx

Параметр `--replace` перезапишет старую запись

```
$ mysqlimport --replace mycompany employee.txt
```

Я бы	название
1	Артур Дент
2	Marvin
3	Зафод Библброкс

Условный импорт

```
$ mysqlimport --where="id>2" mycompany employee.txt
```

Импортируйте стандартный csv

```
$ mysqlimport
    --fields-optionally-enclosed-by='"'
    --fields-terminated-by=,
    --lines-terminated-by="\r\n"
    mycompany employee.csv
```

Прочитайте mysqlimport онлайн: https://riptutorial.com/ru/mysql/topic/5215/mysqlimport

глава 8: TRIGGERS

Синтаксис

- CREATE [DEFINER = {пользователь | CURRENT_USER}] TRIGGER trigger_name trigger_time trigger_event ON tbl_name ДЛЯ КАЖДОГО ROW [trigger_order] trigger_body
- trigger_time: {ПЕРЕД | ПОСЛЕ }
- trigger_event: {INSERT | ОБНОВЛЕНИЕ | УДАЛЯТЬ }
- trigger_order: {FOLLOWS | PRECEDES} other_trigger_name

замечания

Вам нужно обратить внимание на два момента, если вы уже используете триггеры для других БД:

ДЛЯ КАЖДОЙ РУКИ

`FOR EACH ROW` является обязательной частью синтаксиса

Вы не можете сделать *заявление* триггера (один раз по запросу) , как Oracle сделать. Это скорее проблема, связанная с производительностью, чем реальная пропавшая функция

СОЗДАТЬ ИЛИ ЗАМЕНИТЬ ТРИГГЕР

`CREATE OR REPLACE` не поддерживается MySQL

MySQL не разрешает этот синтаксис, вместо этого вы должны использовать следующее:

```
DELIMITER $$

DROP TRIGGER IF EXISTS myTrigger;
$$
CREATE TRIGGER myTrigger
-- ...

$$
DELIMITER ;
```

Будьте осторожны, это **не атомная транзакция** :

- вы потеряете старый триггер, если `CREATE` не `CREATE`
- при большой нагрузке между `DROP` и `CREATE` могут выполняться другие операции, используйте `LOCK TABLES myTable WRITE;` сначала, чтобы избежать несогласованности

данных и `UNLOCK TABLES;` после `CREATE` выпустить таблицу

Examples

Базовый триггер

Создать таблицу

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
Query OK, 0 rows affected (0.03 sec)
```

Создать триггер

```
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
    -> FOR EACH ROW SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.06 sec)
```

Оператор CREATE TRIGGER создает триггер с именем ins_sum, который связан с таблицей учетных записей. Он также включает в себя положения, которые определяют время срабатывания триггера, событие запуска и что делать, когда активируется триггер

Вставить значение

Чтобы использовать триггер, установите переменную аккумулятора (@sum) в ноль, выполните инструкцию INSERT и затем посмотрите, какое значение имеет значение после переменной:

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
+-----------------------+
| Total amount inserted |
+-----------------------+
| 1852.48               |
+-----------------------+
```

В этом случае значение @sum после выполнения инструкции INSERT составляет 14.98 + 1937.50 - 100, или 1852.48.

Drop Trigger

```
mysql> DROP TRIGGER test.ins_sum;
```

Если вы удалите таблицу, все триггеры для таблицы также будут удалены.

Типы триггеров

тайминг

Существуют два модификатора времени действия триггера:

- `BEFORE` триггера активируется запрос,
- `AFTER` запускать огонь после смены.

Событие триггера

Существует три события, к которым можно подключить триггеры:

- `INSERT`
- `UPDATE`
- `DELETE`

Пример примера Insert

```
DELIMITER $$

CREATE TRIGGER insert_date
    BEFORE INSERT ON stack
    FOR EACH ROW
BEGIN
        -- set the insert_date field in the request before the insert
    SET NEW.insert_date = NOW();
END;

$$
DELIMITER ;
```

Пример обновления перед обновлением

```
DELIMITER $$

CREATE TRIGGER update_date
    BEFORE UPDATE ON stack
    FOR EACH ROW
BEGIN
        -- set the update_date field in the request before the update
    SET NEW.update_date = NOW();
END;

$$
DELIMITER ;
```

После удаления примера триггера

```
DELIMITER $$

CREATE TRIGGER deletion_date
    AFTER DELETE ON stack
    FOR EACH ROW
BEGIN
        -- add a log entry after a successful delete
    INSERT INTO log_action(stack_id, deleted_date) VALUES(OLD.id, NOW());
END;

$$
DELIMITER ;
```

Прочитайте TRIGGERS онлайн: https://riptutorial.com/ru/mysql/topic/3069/triggers

глава 9: UNION

Синтаксис

- UNION DISTINCT - отчисления после объединения SELECT
- UNION ALL - без дедуплирования (быстрее)
- UNION - по умолчанию DISTINCT
- SELECT ... UNION SELECT ... - это нормально, но неоднозначно на `ORDER BY`
- (SELECT ...) UNION (SELECT ...) ORDER BY ... - устраняет двусмысленность

замечания

UNION не использует несколько процессоров.

UNION всегда * включает временную таблицу для сбора результатов. * Начиная с 5.7.3 / MariaDB 10.1, некоторые формы UNION предоставляют результаты без использования таблицы tmp (следовательно, быстрее).

Examples

Объединение операторов SELECT с UNION

Вы можете комбинировать результаты двух идентично структурированных запросов с ключевым словом `UNION` .

Например, если вам нужен список всех контактных данных из двух отдельных таблиц, `authors` и `editors` , например, вы можете использовать ключевое слово `UNION` так:

```
select name, email, phone_number
from authors

union

select name, email, phone_number
from editors
```

Использование `union` само по себе будет вытеснять дубликаты. Если вам нужно сохранить дубликаты в своем запросе, вы можете использовать ключевое слово `ALL` так: `UNION ALL` .

СОРТИРОВАТЬ ПО

Если вам нужно отсортировать результаты UNION, используйте этот шаблон:

```
( SELECT ... )
```

```
UNION
( SELECT ... )
ORDER BY
```

Без круглых скобок конечный ORDER BY будет принадлежать последнему SELECT.

Разбиение страницы с помощью OFFSET

При добавлении LIMIT в UNION это шаблон для использования:

```
( SELECT ... ORDER BY x  LIMIT 10 )
UNION
( SELECT ... ORDER BY x  LIMIT 10 )
ORDER BY x  LIMIT 10
```

Поскольку вы не можете предсказать, какие из SELECT (s) будут иметь значение «10», вам нужно получить 10 из каждого, а затем еще больше уничтожить список, повторяя оба `ORDER BY` и `LIMIT` .

Для 4-й страницы из 10 элементов этот шаблон необходим:

```
( SELECT ... ORDER BY x  LIMIT 40 )
UNION
( SELECT ... ORDER BY x  LIMIT 40 )
ORDER BY x  LIMIT 30, 10
```

То есть, собрать 4 страницы в каждом `SELECT` , а затем сделать `OFFSET` в `UNION` .

Объединение данных с разными столбцами

```
SELECT name, caption as title, year, pages FROM books
UNION
SELECT name, title, year, 0 as pages FROM movies
```

При объединении двух наборов записей с разными столбцами, эмулируйте отсутствующие значения по умолчанию.

UNION ALL и UNION

SELECT 1,22,44 UNION SELECT 2,33,55

1	22	44
1	22	44
2	33	55

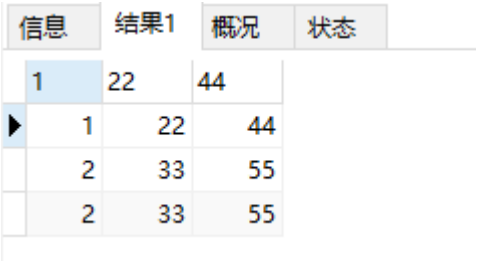
SELECT 1,22,44 UNION SELECT 2,33,55 UNION SELECT 2,33,55

Результат такой же, как и выше.

использовать UNION ALL

когда

SELECT 1,22,44 UNION SELECT 2,33,55 UNION ALL SELECT 2,33,55



1	22	44
1	22	44
2	33	55
2	33	55

Объединение и объединение данных в разные таблицы MySQL с одинаковыми столбцами в уникальные строки и выполнение запроса

Этот ***UNION ALL*** объединяет данные из нескольких таблиц и служит псевдонимом имени таблицы для использования в ваших запросах:

```
SELECT YEAR(date_time_column), MONTH(date_time_column), MIN(DATE(date_time_column)),
MAX(DATE(date_time_column)), COUNT(DISTINCT (ip)), COUNT(ip), (COUNT(ip) / COUNT(DISTINCT
(ip))) AS Ratio
FROM (
    (SELECT date_time_column, ip FROM server_log_1 WHERE state = 'action' AND log_id = 150)
UNION ALL
    (SELECT date_time_column, ip FROM server_log_2 WHERE state = 'action' AND log_id = 150)
UNION ALL
    (SELECT date_time_column, ip FROM server_log_3 WHERE state = 'action' AND log_id = 150)
UNION ALL
    (SELECT date_time_column, ip FROM server_log WHERE state = 'action' AND log_id = 150)
) AS table_all
GROUP BY YEAR(date_time_column), MONTH(date_time_column);
```

Прочитайте UNION онлайн: https://riptutorial.com/ru/mysql/topic/3847/union

глава 10: Администратор MySQL

Examples

Изменение пароля root

```
mysqladmin -u root -p'old-password' password 'new-password'
```

База данных Drop

Полезно для сценариев, чтобы удалить все таблицы и удалить базу данных:

```
mysqladmin -u[username] -p[password] drop [database]
```

Используйте с особой осторожностью.

Для `DROP` базы данных как SQL-скрипта (вам понадобится DROP-привилегия в этой базе данных):

```
DROP DATABASE database_name
```

или же

```
DROP SCHEMA database_name
```

Atomic RENAME & Table Reload

```
RENAME TABLE t TO t_old, t_copy TO t;
```

Никакие другие сеансы не могут обращаться к задействованным таблицам, пока выполняется RENAME TABLE, поэтому операция переименования не подвержена проблемам параллелизма.

Atomic Rename особенно подходит для полной перезагрузки таблицы, не дожидаясь `DELETE` и загрузки до конца:

```
CREATE TABLE new LIKE real;
load `new` by whatever means - LOAD DATA, INSERT, whatever
RENAME TABLE real TO old, new TO real;
DROP TABLE old;
```

Прочитайте Администратор MySQL онлайн: https://riptutorial.com/ru/mysql/topic/2991/администратор-mysql

глава 11: арифметика

замечания

MySQL, на большинстве машин, использует 64-разрядную арифметику с плавающей точкой IEEE 754 для своих вычислений.

В целых контекстах используется целочисленная арифметика.

- `RAND()` не является идеальным генератором случайных чисел. Он в основном используется для быстрого генерации псевдослучайных чисел

Examples

Арифметические операторы

MySQL предоставляет следующие арифметические операторы

оператор	название	пример
`+`	прибавление	`SELECT 3+5;` -> 8 `SELECT 3.5+2.5;` -> 6.0 `SELECT 3.5+2;` -> 5,5
`-`	Вычитание	`SELECT 3-5;` -> -2
`*`	умножение	`SELECT 3 * 5;` -> 15
`/`	разделение	`SELECT 20 / 4;` -> 5 `SELECT 355 / 113;` -> 3,1416 `SELECT 10.0 / 0;` -> NULL
`DIV`	Целостный отдел	`SELECT 5 DIV 2;` -> 2
`%` или `MOD`	Модульное	`SELECT 7 % 3;` -> 1 `SELECT 15 MOD 4` -> 3 `SELECT 15 MOD -4` -> 3 `SELECT -15 MOD 4` -> -3 `SELECT -15 MOD -4` -> -3 `SELECT 3 MOD 2.5` -> 0,5

BIGINT

Если числа в вашей арифметике являются целыми числами, MySQL использует тип данных

`BIGINT` (подписанный 64-разрядный) для выполнения своей работы. Например:

`select (1024 * 1024 * 1024 * 1024 *1024 * 1024) + 1` -> 1,152,921,504,606,846,977

а также

`select (1024 * 1024 * 1024 * 1024 *1024 * 1024 * 1024` -> Ошибка `BIGINT` вне диапазона

DOUBLE

Если какие-либо цифры в вашей арифметике являются дробными, MySQL использует 64-битную арифметику с плавающей точкой IEEE 754 . Вы должны быть осторожны при использовании арифметики с плавающей запятой, поскольку многие числа с плавающей запятой являются, по сути, приблизительными, а не точными значениями .

Математические константы

число Пи

Следующее возвращает значение `PI` отформатированное до 6 знаков после запятой. Фактическое значение хорошее для `DOUBLE` ;

```
SELECT PI();    -> 3.141593
```

Тригонометрия (SIN, COS)

Углы находятся в радианах, а не в градусах. Все вычисления выполняются в 64-битной плавающей точке IEEE 754 . Все вычисления с плавающей точкой подвержены небольшим ошибкам, известным как ошибки машины ε (epsilon) , поэтому не пытайтесь сравнивать их для равенства. Невозможно избежать этих ошибок при использовании с плавающей запятой; они встроены в технологию.

Если вы используете значения `DECIMAL` в тригонометрических вычислениях, они неявно преобразуются в с плавающей запятой, а затем обратно в десятичные.

Синус

Возвращает синус числа X, выраженного в радианах

```
SELECT SIN(PI()); -> 1.2246063538224e-16
```

Косинус

Возвращает косинус X, когда X задан в радианах

```
SELECT COS(PI()); -> -1
```

касательный

Возвращает тангенс числа X, выраженного в радианах. Обратите внимание, что результат очень близок к нулю, но не точно равен нулю. Это пример машины ε.

```
SELECT TAN(PI());   -> -1.2246063538224e-16
```

Arc Cosine (обратный косинус)

Возвращает дуговый косинус X, если X находится в диапазоне от `-1 to 1`

```
SELECT ACOS(1);    -> 0
SELECT ACOS(1.01); -> NULL
```

Дуга Синус (обратный синус)

Возвращает синус дуги X, если X находится в диапазоне от `-1 to 1`

```
SELECT ASIN(0.2); -> 0.20135792079033
```

Дуга Тангенс (обратная касательная)

`ATAN(x)` возвращает тангенс дуги одного числа.

```
SELECT ATAN(2); -> 1.1071487177941
```

`ATAN2(X, Y)` возвращает тангенс дуги двух переменных X и Y. Он аналогичен вычислению дуги тангенса Y / X. Но он численно более устойчив: t правильно работает, когда X близок к нулю, а знаки обоих аргументов используются для определения квадранта результата.

Лучшая практика предлагает писать формулы для использования `ATAN2()` а не `ATAN()` где это возможно.

```
 ATAN2(1,1);    -> 0.7853981633974483 (45 degrees)
 ATAN2(1,-1);   -> 2.356194490192345  (135 degrees)
 ATAN2(0, -1);  -> PI  (180 degrees)  don't try ATAN(-1 / 0)... it won't work
```

Котангенс

Возвращает котангенс X

```
SELECT COT(12); -> -1.5726734063977
```

преобразование

```
SELECT RADIANS(90) -> 1.5707963267948966
SELECT SIN(RADIANS(90)) -> 1
SELECT DEGREES(1), DEGREES(PI()) -> 57.29577951308232, 180
```

Округление (ROUND, FLOOR, CEIL)

Округлить десятичное число до целочисленного значения

Для точных числовых значений (например, `DECIMAL`): если первое десятичное число числа равно 5 или выше, эта функция будет округлять число до следующего целого числа *от нуля* . Если это десятичное место равно 4 или ниже, эта функция будет округляться до следующего целого значения, *ближайшего к нулю* .

```
SELECT ROUND(4.51) -> 5
SELECT ROUND(4.49) -> 4
SELECT ROUND(-4.51) -> -5
```

Для приблизительных числовых значений (например, `DOUBLE`): результат функции `ROUND()` зависит от библиотеки C; на многих системах это означает, что `ROUND()` использует *округление до ближайшего четного* правила:

```
SELECT ROUND(45e-1) -> 4  -- The nearest even value is 4
SELECT ROUND(55e-1) -> 6  -- The nearest even value is 6
```

Завершить номер

Для округления числа используйте `CEIL()` или `CEILING()`

```
SELECT CEIL(1.23)    -> 2
SELECT CEILING(4.83) -> 5
```

Завершить число

Чтобы округлить число, используйте функцию `FLOOR()`

```
SELECT FLOOR(1.99) -> 1
```

FLOOR и CEIL идут в сторону / от -infinity:

```
SELECT FLOOR(-1.01), CEIL(-1.01) -> -2 and -1
SELECT FLOOR(-1.99), CEIL(-1.99) -> -2 and -1
```

Закруглить десятичное число на указанное число знаков после запятой.

```
SELECT ROUND(1234.987, 2) -> 1234.99
SELECT ROUND(1234.987, -2) -> 1200
```

Также обсуждается обсуждение «против» и «5».

Поднимите число до мощности (POW)

Чтобы поднять число `x` до мощности `y` , используйте либо функции `POW()` либо `POWER()`

```
SELECT POW(2,2); => 4
SELECT POW(4,2); => 16
```

Квадратный корень (SQRT)

Используйте функцию `SQRT()` . Если число отрицательное, возвращается `NULL`

```
SELECT SQRT(16); -> 4
SELECT SQRT(-3); -> NULL
```

Случайные числа (RAND)

Создать случайное число

Чтобы создать псевдослучайное число с плавающей запятой между `0` и `1` , используйте функцию `RAND()`

Предположим, у вас есть следующий запрос

```
SELECT i, RAND() FROM t;
```

Это вернет что-то вроде этого

я	RAND ()
1	+0,6191438870682

я	RAND ()
2	+0,93845168309142
3	+0,83482678498591

Случайное число в диапазоне

Чтобы создать случайное число в диапазоне a <= n <= b, вы можете использовать следующую формулу

```
FLOOR(a + RAND() * (b - a + 1))
```

Например, это приведет к случайному числу от 7 до 12

```
SELECT FLOOR(7 + (RAND() * 6));
```

Простой способ случайного возврата строк в таблицу:

```
SELECT * FROM tbl ORDER BY RAND();
```

Это **псевдослучайные** числа.

Генератор псевдослучайных чисел в MySQL не является криптографически безопасным. То есть, если вы используете MySQL для генерации случайных чисел, которые будут использоваться в качестве секретов, определенный противник, который знает, что вы использовали MySQL, сможет угадать ваши секреты легче, чем вы могли бы поверить.

Абсолютная ценность и знак (ABS, SIGN)

Вернуть абсолютное значение числа

```
SELECT ABS(2);   -> 2
SELECT ABS(-46); -> 46
```

`sign` числа сравнивает его с 0.

Знак	Результат	пример
-1	n <0	`SELECT SIGN(42);` -> 1
0	n = 0	`SELECT SIGN(0);` -> 0
1	n> 0	`SELECT SIGN(-3);` -> -1

```
SELECT SIGN(-423421); -> -1
```

Прочитайте арифметика онлайн: https://riptutorial.com/ru/mysql/topic/4516/арифметика

глава 12: Безопасность через ГРАНТ

Examples

Лучшая практика

Ограничить root (и любого другого пользователя с привилегией SUPER) до

```
GRANT ... TO root@localhost ...
```

Это предотвращает доступ с других серверов. Вы должны передать СУПЕР очень мало людей, и они должны осознавать свою ответственность. Приложение не должно иметь SUPER.

Ограничить использование приложений для одной базы данных, используемой им:

```
GRANT ... ON dbname.* ...
```

Таким образом, кто-то, кто взламывает код приложения, не может пройти мимо dbname. Это может быть дополнительно уточнено с помощью любого из них:

```
GRANT SELECT ON dname.* ...    -- "read only"
GRANT ... ON dname.tblname ... -- "just one table"
```

В readonly также могут быть «безопасные» вещи, такие как

```
GRANT SELECT, CREATE TEMPORARY TABLE ON dname.* ...    -- "read only"
```

Как вы говорите, нет абсолютной безопасности. Я хочу сказать, что вы можете сделать несколько вещей, чтобы замедлить хакеров. (То же самое касается честных людей.)

В редких случаях вам может понадобиться приложение, чтобы сделать что-то доступное только для root. это можно сделать с помощью «Хранимой процедуры», в которой установлен `SECURITY DEFINER` (и определяет корень). Это покажет только то, что делает SP, что может быть, например, одним конкретным действием в одной конкретной таблице.

Хост (пользователя @ host)

«Хост» может быть либо именем хоста, либо IP-адресом. Кроме того, он может включать в себя дикие карты.

```
GRANT SELECT ON db.* TO sam@'my.domain.com' IDENTIFIED BY 'foo';
```

Примеры: Примечание: обычно их нужно указывать

```
localhost -- the same machine as mysqld
'my.domain.com' -- a specific domain; this involves a lookup
'11.22.33.44' -- a specific IP address
'192.168.1.%' -- wild card for trailing part of IP address.  (192.168.% and 10.% and 11.% are
"internal" ip addresses.)
```

Использование `localhost` зависит от безопасности сервера. Для лучшей практики `root` должен разрешаться только через localhost. В некоторых случаях это означает одно и то же: `0.0.0.1` и `::1` .

Прочитайте Безопасность через ГРАНТ онлайн: https://riptutorial.com/ru/mysql/topic/5131/безопасность-через-грант

глава 13: Восстановить и сбросить пароль по умолчанию для MySQL 5.7+

Вступление

После MySQL 5.7, когда мы устанавливаем MySQL, иногда нам не нужно создавать учетную запись root или вводить пароль root. По умолчанию при запуске сервера пароль по умолчанию хранится в файле `mysqld.log` . Нам нужно войти в систему, используя этот пароль, и нам нужно его изменить.

замечания

Восстановление и сброс пароля root по умолчанию с помощью этого метода применимо только для MySQL 5.7+

Examples

Что происходит при первом запуске сервера

Учитывая, что каталог данных сервера пуст:

- Сервер инициализируется.
- SSL-сертификат и ключевые файлы создаются в каталоге данных.
- Плагин validate_password установлен и включен.
- Создается учетная запись суперпользователя «root» @ «localhost». Пароль для суперпользователя устанавливается и сохраняется в файле журнала ошибок.

Как изменить пароль root с помощью пароля по умолчанию

Чтобы определить пароль «root» по умолчанию:

```
shell> sudo grep 'temporary password' /var/log/mysqld.log
```

Измените пароль root как можно скорее, войдя с созданным временным паролем и установите пользовательский пароль для учетной записи суперпользователя:

```
shell> mysql -uroot -p

mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass5!';
```

Примечание. Плагин validate_password MySQL установлен по умолчанию. Это потребует,

чтобы пароли содержали по крайней мере одну букву верхнего регистра, одну строчную букву, одну цифру и один специальный символ, а общая длина пароля не менее 8 символов.

сбросить пароль root, когда «/ var / run / mysqld» для файла сокета UNIX не существует »

если я забуду пароль, тогда я получу ошибку.

```
$ mysql -u root -p
```

Введите пароль:

ОШИБКА 1045 (28000): Доступ запрещен для пользователя 'root' @ 'localhost' (с использованием пароля: YES)

Я попытался решить проблему, сначала зная статус:

```
$ systemctl status mysql.service
```

mysql.service - Сервер сообщества MySQL Загружен: загружен (/ lib/systemd/system/mysql.service; включен; предварительный набор поставщика: en Активен: активен (работает) с тех пор 2017-06-08 14:31:33 IST; 38s ago

Затем я использовал код `mysqld_safe --skip-grant-tables &` но я получил ошибку:

> mysqld_safe Directory '/ var / run / mysqld' для файла сокета UNIX не существует.

```
$ systemctl stop  mysql.service
$ ps -eaf|grep mysql
$ mysqld_safe --skip-grant-tables &
```

> Я решил:

```
$ mkdir -p /var/run/mysqld
$ chown mysql:mysql /var/run/mysqld
```

Теперь я использую тот же код `mysqld_safe --skip-grant-tables &` и получаю

> mysqld_safe Запуск mysqld-демона с базами данных из / var / lib / mysql

Если я использую `$ mysql -u root` я получу:

Версия сервера: 5.7.18-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle и / или ее аффилированные лица. Все права защищены.

Oracle является зарегистрированным товарным знаком корпорации Oracle и / или ее филиалов. Другие наименования могут быть торговыми марками их владельцев.

Введите «help»; или '\ h' для получения справки. Введите '\ c', чтобы очистить текущую инструкцию ввода.

MySQL>

Теперь время для смены пароля:

```
mysql> use mysql
mysql> describe user;
```

Чтение информации о таблице для заполнения имен таблиц и столбцов Вы можете отключить эту функцию, чтобы быстрее запустить с помощью -A

Изменена база данных

```
mysql> FLUSH PRIVILEGES;
mysql> SET PASSWORD FOR root@'localhost' = PASSWORD('newpwd');
```

или Если у вас есть учетная запись mysql root, которая может подключаться повсюду, вам также нужно:

```
UPDATE mysql.user SET Password=PASSWORD('newpwd') WHERE User='root';
```

Альтернативный метод:

```
   USE mysql
   UPDATE user SET Password = PASSWORD('newpwd')
   WHERE Host = 'localhost' AND User = 'root';
```

И если у вас есть учетная запись root, доступ к которой можно получить везде:

```
 USE mysql
 UPDATE user SET Password = PASSWORD('newpwd')
 WHERE Host = '%' AND User = 'root';`enter code here
```

теперь нужно `quit` из mysql и остановить / запустить

```
FLUSH PRIVILEGES;
sudo /etc/init.d/mysql stop
sudo /etc/init.d/mysql start
```

теперь снова `mysql -u root -p 'и используйте новый пароль для получения

MySQL>

Прочитайте Восстановить и сбросить пароль по умолчанию для MySQL 5.7+ онлайн: https://riptutorial.com/ru/mysql/topic/9563/восстановить-и-сбросить-пароль-по-умолчанию-для-mysql-5-7plus

глава 14: Восстановление с утраченного пароля root

Examples

Установите пароль root, включите пользователя root для доступа к сокету и http

Решает проблему: доступ запрещен для пользователя root, используя пароль YES Stop mySQL:

```
sudo systemctl stop mysql
```

Перезапустите mySQL, пропустив таблицы грантов:

```
sudo mysqld_safe --skip-grant-tables
```

Авторизоваться:

```
mysql -u root
```

В оболочке SQL посмотрите, существуют ли пользователи:

```
select User, password,plugin FROM mysql.user ;
```

Обновите пользователей (плагин null разрешает для всех плагинов):

```
update mysql.user set password=PASSWORD('mypassword'), plugin = NULL WHERE User = 'root';
exit;
```

В оболочке Unix остановите mySQL без таблиц грантов, затем перезапустите с помощью таблиц привилегий:

```
sudo service mysql stop
sudo service mysql start
```

Прочитайте Восстановление с утраченного пароля root онлайн:
https://riptutorial.com/ru/mysql/topic/9973/восстановление-с-утраченного-пароля-root

глава 15: Временные таблицы

Examples

Создать временную таблицу

Временные таблицы могут быть очень полезны для хранения временных данных. Вариант временных таблиц доступен в MySQL версии 3.23 и выше.

Временная таблица будет автоматически уничтожена при завершении сеанса или закрытии соединения. Пользователь также может отбрасывать временную таблицу.

Одно и то же имя временной таблицы может использоваться во многих соединениях одновременно, поскольку временная таблица доступна и доступна только клиенту, который создает эту таблицу.

Временную таблицу можно создать в следующих типах

```
--->Basic temporary table creation
CREATE TEMPORARY TABLE tempTable1(
       id INT NOT NULL AUTO_INCREMENT,
       title VARCHAR(100) NOT NULL,
       PRIMARY KEY ( id )
    );

--->Temporary table creation from select query
CREATE TEMPORARY TABLE tempTable1
    SELECT ColumnName1,ColumnName2,... FROM table1;
```

Вы можете добавлять индексы при создании таблицы:

```
CREATE TEMPORARY TABLE tempTable1
        ( PRIMARY KEY(ColumnName2) )
    SELECT ColumnName1,ColumnName2,... FROM table1;
```

`IF NOT EXISTS` ключевое слово может использоваться, как указано ниже, чтобы избежать ошибки *«таблица уже существует»* . Но в этом случае таблица не будет создана, если имя таблицы, которое вы используете, уже существует в текущем сеансе.

```
CREATE TEMPORARY TABLE IF NOT EXISTS tempTable1
    SELECT ColumnName1,ColumnName2,... FROM table1;
```

Временная таблица падения

Drop Temporary Table используется для удаления временной таблицы, созданной в текущем сеансе.

```
DROP TEMPORARY TABLE tempTable1

DROP TEMPORARY TABLE IF EXISTS tempTable1
```

Используйте `IF EXISTS` чтобы предотвратить возникновение ошибки для таблиц, которые могут не существовать

Прочитайте Временные таблицы онлайн: https://riptutorial.com/ru/mysql/topic/5757/временные-таблицы

глава 16: Время с точностью до секунды

замечания

Вы должны быть в MySQL версии 5.6.4 или новее, чтобы объявлять столбцы с дробно-временными типами данных.

Например, `DATETIME(3)` даст вам миллисекундное разрешение на ваших временных отметках, а `TIMESTAMP(6)` предоставит вам разрешение по микросекундам на отметке времени в стиле nix.

Прочтите это: http://dev.mysql.com/doc/refman/5.7/en/fractional-seconds.html

`NOW(3)` предоставит вам текущее время с вашей операционной системы сервера MySQL с точностью до миллисекунды.

(Обратите внимание, что внутренняя дробная арифметика MySQL, такая как * 0.001, всегда обрабатывается как плавающая точка с двойной точностью IEEE754, поэтому маловероятно, что вы потеряете точность до того, как Солнце станет звездой белого карлика.)

Examples

Получить текущее время с точностью до миллисекунды

```
SELECT NOW(3)
```

делает трюк.

Получить текущее время в форме, которая похожа на метку времени Javascript.

Временные метки Javascript основаны на почтенном типе данных UNIX `time_t` и показывают количество миллисекунд с `1970-01-01 00:00:00` UTC.

Это выражение получает текущее время как целое число timestamp Javascript. (Он делает это правильно, независимо от текущей настройки time_zone).

```
ROUND(UNIX_TIMESTAMP(NOW(3)) * 1000.0, 0)
```

Если у вас есть значения `TIMESTAMP` хранящиеся в столбце, вы можете получить их как целочисленные временные метки Javascript, используя функцию UNIX_TIMESTAMP ().

```
  SELECT ROUND(UNIX_TIMESTAMP(column) * 1000.0, 0)
```

Если ваш столбец содержит столбцы `DATETIME` и вы извлекаете их как отметки времени Javascript, эти временные метки будут компенсированы смещением часового пояса временного пояса, в котором они хранятся.

Создайте таблицу со столбцами для хранения суб-второго времени.

```
CREATE TABLE times (
     dt DATETIME(3),
     ts TIMESTAMP(3)
 );
```

создает таблицу с полями даты / времени с точностью до миллисекунд.

```
 INSERT INTO times VALUES (NOW(3), NOW(3));
```

вставляет строку, содержащую значения `NOW()` с точностью до миллисекунды в таблицу.

```
 INSERT INTO times VALUES ('2015-01-01 16:34:00.123','2015-01-01 16:34:00.128');
```

вставляет конкретные значения точности миллисекунды.

Обратите внимание, что вы должны использовать `NOW(3)` а не `NOW()` если вы используете эту функцию для вставки высокоточных значений времени.

Преобразование значения даты / времени с точностью до миллисекунды в текст.

`%f` - спецификатор формата дробной точности для функции DATE_FORMAT () .

```
SELECT DATE_FORMAT(NOW(3), '%Y-%m-%d %H:%i:%s.%f')
```

отображает значение типа `2016-11-19 09:52:53.248000` с дробными микросекундами. Поскольку мы использовали `NOW(3)` , последние три цифры во фракции равны 0.

Храните метку времени Javascript в столбце TIMESTAMP

Если у вас есть значение временной метки Javascript, например `1478960868932` , вы можете преобразовать это значение в значение дробного времени MySQL следующим образом:

```
FROM_UNIXTIME(1478960868932 * 0.001)
```

Это простое выражение для хранения вашей метки времени Javascript в таблице MySQL. Сделай это:

```
INSERT INTO table (col) VALUES (FROM_UNIXTIME(1478960868932 * 0.001))
```

(Очевидно, вы захотите вставить другие столбцы.)

Прочитайте Время с точностью до секунды онлайн: https://riptutorial.com/ru/mysql/topic/7850/время-с-точностью-до-секунды

глава 17: ВСТАВИТЬ

Синтаксис

1. INSERT [LOW_PRIORITY | ЗАДЕРЖАННЫЕ | HIGH_PRIORITY] [IGNORE] [INTO] tbl_name [PARTITION (имя_пользователя, ...)] [(col_name, ...)] {VALUES | VALUE} ({expr | DEFAULT}, ...), (...), ... [ON DUPLICATE KEY UPDATE col_name = expr [, col_name = expr] ...]

2. INSERT [LOW_PRIORITY | ЗАДЕРЖАННЫЕ | HIGH_PRIORITY] [IGNORE] [INTO] tbl_name [PARTITION (имя_пользователя, ...)] SET col_name = {expr | DEFAULT}, ... [ON DUPLICATE KEY UPDATE col_name = expr [, col_name = expr] ...]

3. INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE] [INTO] tbl_name [PARTITION (имя_раздела, ...)] [(col_name, ...)] SELECT ... [ON DUPLICATE KEY UPDATE col_name = expr [, col_name = expr] ...]

4. Выражение expr может ссылаться на любой столбец, который был установлен ранее в списке значений. Например, вы можете сделать это, потому что значение для col2 относится к col1, который ранее был назначен:
   INSERT INTO tbl_name (col1, col2) VALUES (15, col1 * 2);

5. Операторы INSERT, использующие синтаксис VALUES, могут вставлять несколько строк. Для этого включите несколько списков значений столбцов, каждый из которых заключен в круглые скобки и разделен запятыми. Пример:
   INSERT INTO tbl_name (a, b, c) ЗНАЧЕНИЯ (1,2,3), (4,5,6), (7,8,9);

6. Список значений для каждой строки должен быть заключен в круглые скобки. Следующий оператор является незаконным, поскольку количество значений в списке не соответствует числу имен столбцов:
   INSERT INTO tbl_name (a, b, c) ЗНАЧЕНИЯ (1,2,3,4,5,6,7,8,9);

7. INSERT ... SELECT Синтаксис
   INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE] [INTO] tbl_name [PARTITION (имя_пользователя, ...)] [(col_name, ...)] SELECT ... [ON DUPLICATE KEY UPDATE col_name = expr, ...]

8. С помощью INSERT ... SELECT вы можете быстро вставить много строк в таблицу из одной или многих таблиц. Например:
   INSERT INTO tbl_temp2 (fld_id) SELECT tbl_temp1.fld_order_id FROM tbl_temp1 WHERE tbl_temp1.fld_order_id> 100;

замечания

Официальный синтаксис INSERT

Examples

Основная вставка

```
INSERT INTO `table_name` (`field_one`, `field_two`) VALUES ('value_one', 'value_two');
```

В этом тривиальном примере `table_name` , где должны быть добавлены данные, `field_one` и `field_two` - это поля для установки данных, а `value_one` и `value_two` - данные, которые нужно делать против `field_one` и `field_two` соответственно.

Рекомендуется перечислить поля, в которые вы вставляете данные, в свой код, как если бы таблица была изменена, а новые столбцы добавлены, ваша вставка будет разбита, если они не будут там

INSERT, ON DUPLICATE KEY UPDATE

```
INSERT INTO `table_name`
    (`index_field`, `other_field_1`, `other_field_2`)
    VALUES
    ('index_value', 'insert_value', 'other_value')
    ON DUPLICATE KEY UPDATE
        `other_field_1` = 'update_value',
        `other_field_2` = VALUES(`other_field_2`);
```

Это будет `INSERT` в `table_name` указанными значениями, но если уникальный ключ уже существует, он обновит `other_field_1` чтобы получить новое значение.
Иногда при обновлении по дубликатному ключу полезно использовать `VALUES()` , чтобы получить исходное значение, которое было передано в `INSERT` вместо того, чтобы напрямую устанавливать значение. Таким образом, вы можете установить разные значения с помощью `INSERT` и `UPDATE` . См. Пример выше, где `other_field_1` установлен в `insert_value` на `INSERT` или `update_value` в `UPDATE` то время как `other_field_2` всегда имеет значение `other_value` .

Важным для вставки в обновлении повторяющихся ключей (IODKU) является схема, содержащая уникальный ключ, который будет сигнализировать о дублировании конфликта. Этот уникальный ключ может быть Первичным ключом или нет. Это может быть уникальный ключ в одном столбце или многостолбцовый (составной ключ).

Вставка нескольких строк

```
INSERT INTO `my_table` (`field_1`, `field_2`) VALUES
    ('data_1', 'data_2'),
    ('data_1', 'data_3'),
    ('data_4', 'data_5');
```

Это простой способ добавить несколько строк одновременно с помощью одной `INSERT` .

Такой тип «пакетной» вставки намного быстрее, чем вставка строк один за другим. Как правило, вставка 100 строк в одну пакетную вставку таким образом в 10 раз быстрее, чем вставка их отдельно.

Игнорирование существующих строк

При импорте больших наборов данных может быть предпочтительным при определенных обстоятельствах пропускать строки, которые обычно приводят к сбою запроса из-за ограничения столбца, например дублирования первичных ключей. Это можно сделать, используя `INSERT IGNORE` .

Рассмотрим следующую примерную базу данных:

```
SELECT * FROM `people`;
--- Produces:
+----+------+
| id | name |
+----+------+
|  1 | john |
|  2 | anna |
+----+------+


INSERT IGNORE INTO `people` (`id`, `name`) VALUES
    ('2', 'anna'), --- Without the IGNORE keyword, this record would produce an error
    ('3', 'mike');

SELECT * FROM `people`;
--- Produces:
+----+--------+
| id |  name  |
+----+--------+
|  1 |  john  |
|  2 |  anna  |
|  3 |  mike  |
+----+--------+
```

Важно помнить, что *INSERT IGNORE* также тихо пропустит и другие ошибки, вот что говорит официальная документация Mysql:

> Преобразования данных, которые будут приводить к ошибкам, прерывают утверждение, если IGNORE не указано. С помощью IGNORE недопустимые значения корректируются до ближайших значений и> вставлены; появляются предупреждения, но утверждение не прерывается.

Примечание. - Раздел ниже добавлен для полноты, но не считается лучшей практикой (это не удастся, например, если в таблицу был добавлен другой столбец).

Если вы укажете значение соответствующего столбца для всех столбцов в таблице, вы

можете игнорировать список столбцов в `INSERT` следующим образом:

```
INSERT INTO `my_table` VALUES
        ('data_1', 'data_2'),
        ('data_1', 'data_3'),
        ('data_4', 'data_5');
```

INSERT SELECT (Вставка данных из другой таблицы)

Это основной способ вставки данных из другой таблицы с помощью инструкции SELECT.

```
INSERT INTO `tableA` (`field_one`, `field_two`)
   SELECT `tableB`.`field_one`, `tableB`.`field_two`
   FROM `tableB`
   WHERE `tableB`.clmn <> 'someValue'
   ORDER BY `tableB`.`sorting_clmn`;
```

Вы можете `SELECT * FROM` , но тогда `tableA` и `tableB` *должны* иметь соответствующее количество столбцов и соответствующие типы данных.

Столбцы с `AUTO_INCREMENT` рассматриваются как в предложении `INSERT` с предложением `VALUES` .

Этот синтаксис позволяет легко заполнять (временные) таблицы данными из других таблиц, тем более, когда данные должны быть отфильтрованы по вставке.

INSERT с AUTO_INCREMENT + LAST_INSERT_ID ()

Когда таблица имеет `AUTO_INCREMENT PRIMARY KEY` , `AUTO_INCREMENT` , обычно один не вставляется в этот столбец. Вместо этого укажите все остальные столбцы, а затем спросите, что такое новый идентификатор.

```
CREATE TABLE t (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    this ...,
    that ...,
    PRIMARY KEY(id) );

INSERT INTO t (this, that) VALUES (..., ...);
SELECT LAST_INSERT_ID() INTO @id;
INSERT INTO another_table (..., t_id, ...) VALUES (..., @id, ...);
```

Обратите внимание, что `LAST_INSERT_ID()` привязан к сеансу, поэтому даже если несколько подключений вставляются в одну и ту же таблицу, каждый получает свой собственный идентификатор.

У вашего клиентского API, вероятно, есть альтернативный способ получения `LAST_INSERT_ID()` без фактического выполнения `SELECT` и передачи значения обратно клиенту вместо того, чтобы оставить его в `@variable` внутри MySQL. Обычно это предпочтительнее.

Более длинный, более подробный, пример

«Нормальным» использованием IODKU является запуск «дублирующего ключа» на основе некоторого ключа `UNIQUE` , а не главного ключа `AUTO_INCREMENT PRIMARY KEY` . Это демонстрирует следующее. Обратите внимание, что он не *содержит* `id` в INSERT.

Настройка для следующих примеров:

```
CREATE TABLE iodku (
    id INT AUTO_INCREMENT NOT NULL,
    name VARCHAR(99) NOT NULL,
    misc INT NOT NULL,
    PRIMARY KEY(id),
    UNIQUE(name)
) ENGINE=InnoDB;

INSERT INTO iodku (name, misc)
    VALUES
    ('Leslie', 123),
    ('Sally', 456);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
+----+--------+------+
| id | name   | misc |
+----+--------+------+
|  1 | Leslie |  123 |
|  2 | Sally  |  456 |
+----+--------+------+
```

Случай IODKU, выполняющий «обновление» и `LAST_INSERT_ID()` извлекает соответствующий `id` :

```
INSERT INTO iodku (name, misc)
    VALUES
    ('Sally', 3333)            -- should update
    ON DUPLICATE KEY UPDATE    -- `name` will trigger "duplicate key"
    id = LAST_INSERT_ID(id),
    misc = VALUES(misc);
SELECT LAST_INSERT_ID();       -- picking up existing value
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                2 |
+------------------+
```

Случай, когда IODKU выполняет «вставку», и `LAST_INSERT_ID()` извлекает новый `id` :

```
INSERT INTO iodku (name, misc)
    VALUES
    ('Dana', 789)          -- Should insert
    ON DUPLICATE KEY UPDATE
    id = LAST_INSERT_ID(id),
    misc = VALUES(misc);
SELECT LAST_INSERT_ID();   -- picking up new value
+------------------+
| LAST_INSERT_ID() |
```

```
+------------------+
|                3 |
+------------------+
```

Результирующее содержимое таблицы:

```
SELECT * FROM iodku;
+----+--------+------+
| id | name   | misc |
+----+--------+------+
|  1 | Leslie |  123 |
|  2 | Sally  | 3333 |  -- IODKU changed this
|  3 | Dana   |  789 |  -- IODKU added this
+----+--------+------+
```

Потерянные идентификаторы AUTO_INCREMENT

Несколько функций «вставки» могут «сжечь» идентификаторы. Вот пример использования InnoDB (другие Двигатели могут работать по-другому):

```
CREATE TABLE Burn (
    id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL,
    name VARCHAR(99) NOT NULL,
    PRIMARY KEY(id),
    UNIQUE(name)
        ) ENGINE=InnoDB;

INSERT IGNORE INTO Burn (name) VALUES ('first'), ('second');
SELECT LAST_INSERT_ID();          -- 1
SELECT * FROM Burn ORDER BY id;
  +----+--------+
  |  1 | first  |
  |  2 | second |
  +----+--------+

INSERT IGNORE INTO Burn (name) VALUES ('second');  -- dup 'IGNOREd', but id=3 is burned
SELECT LAST_INSERT_ID();          -- Still "1" -- can't trust in this situation
SELECT * FROM Burn ORDER BY id;
  +----+--------+
  |  1 | first  |
  |  2 | second |
  +----+--------+

INSERT IGNORE INTO Burn (name) VALUES ('third');
SELECT LAST_INSERT_ID();           -- now "4"
SELECT * FROM Burn ORDER BY id;    -- note that id=3 was skipped over
  +----+--------+
  |  1 | first  |
  |  2 | second |
  |  4 | third  |    -- notice that id=3 has been 'burned'
  +----+--------+
```

Подумайте об этом (примерно) следующим образом: сначала вставка смотрит, сколько строк *может* быть вставлено. Затем возьмите столько значений из auto_increment для этой таблицы. Наконец, вставьте строки, используя идентификаторы по мере необходимости, и

сжигая любые оставшиеся следы.

Единственный раз, когда оставшаяся часть восстанавливается, - это если система выключена и перезапущена. При перезапуске выполняется `MAX(id)` . Это может повторно использовать идентификаторы, которые были сожжены или которые были высвобождены с помощью `DELETEs` с наивысшим id (s).

По существу любой аромат `INSERT` (включая `REPLACE` , который является `DELETE` + `INSERT`) может сжечь идентификаторы. В InnoDB глобальная (не сессия!) Переменная `innodb_autoinc_lock_mode` может использоваться для управления некоторыми из того, что происходит.

Когда «нормализация» длинных строк в `AUTO INCREMENT id` , сжигание может легко произойти. Это *может* привести к переполнению размера `INT` вы выбрали.

Прочитайте ВСТАВИТЬ онлайн: https://riptutorial.com/ru/mysql/topic/866/вставить

глава 18: ВЫБРАТЬ

Вступление

`SELECT` используется для извлечения строк, выбранных из одной или нескольких таблиц.

Синтаксис

- SELECT DISTINCT [выражения] FROM TableName [условия WHERE]; /// Простой выбор
- SELECT DISTINCT (a), b ... совпадает с SELECT DISTINCT a, b ...
- SELECT [ALL | DISTINCT | DISTINCTROW] [HIGH_PRIORITY] [STRAIGHT_JOIN] [SQL_SMALL_RESULT | SQL_BIG_RESULT] [SQL_BUFFER_RESULT] [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS] выражения из таблиц [условия WHERE] [выражения GROUP BY] [условие HAVING] [выражение ORDER BY [ASC | DESC]] [LIMIT [offset_value] number_rows | LIMIT number_rows OFFSET offset_value] [PROCEDURE имя_процесса] [INTO [OUTFILE 'имя_файла' параметры | DUMPFILE ' имя_файла' | @ variable1, @ variable2, ... @variable_n] [FOR UPDATE | LOCK IN SHARE MODE]; /// Полный синтаксис выбора

замечания

Для получения дополнительной информации о `SELECT` MySQL обратитесь к Документам MySQL .

Examples

SELECT по имени столбца

```
CREATE TABLE stack(
    id INT,
    username VARCHAR(30) NOT NULL,
    password VARCHAR(30) NOT NULL
);

INSERT INTO stack (`id`, `username`, `password`) VALUES (1, 'Foo', 'hiddenGem');
INSERT INTO stack (`id`, `username`, `password`) VALUES (2, 'Baa', 'verySecret');
```

запрос

```
SELECT id FROM stack;
```

Результат

```
+------+
| id   |
+------+
|    1 |
|    2 |
+------+
```

ВЫБЕРИТЕ все столбцы (*)

запрос

```
SELECT * FROM stack;
```

Результат

```
+------+----------+----------+
| id   | username | password |
+------+----------+----------+
|    1 | admin    | admin    |
|    2 | stack    | stack    |
+------+----------+----------+
2 rows in set (0.00 sec)
```

Вы можете выбрать все столбцы из одной таблицы в соединении, выполнив:

```
SELECT stack.* FROM stack JOIN Overflow ON stack.id = Overflow.id;
```

Лучшая практика Не используйте `*` если вы не отлаживаете или не извлекаете строки (строки) в ассоциативные массивы, иначе изменения схемы (столбцы ADD / DROP / переупорядочения) могут привести к неприятным ошибкам приложения. Кроме того, если вы укажете список столбцов, который вам нужен в вашем результирующем наборе, планировщик запросов MySQL часто может оптимизировать запрос.

Плюсы:

1. Когда вы добавляете / удаляете столбцы, вам не нужно вносить изменения, когда вы использовали `SELECT *`
2. Короче писать
3. Вы также видите ответы, поэтому `SELECT *` -usage может быть оправданным?

Минусы:

1. Вы возвращаете больше данных, чем вам нужно. Предположим, вы добавили столбец VARBINARY, который содержит 200 тыс. Строк. Вам нужны только эти данные в одном месте для одной записи - с помощью `SELECT *` вы можете в конечном итоге вернуть 2 МБ на 10 строк, которые вам не нужны
2. Явно о том, какие данные используются
3. Указание столбцов означает, что вы получаете сообщение об ошибке при удалении

столбца
4. Обработчик запросов должен выполнить еще одну работу - выяснить, какие столбцы существуют в таблице (спасибо @vinodadhikary)
5. Вы можете найти, где колонка используется более легко
6. Вы получаете все столбцы в соединениях, если используете SELECT *
7. Вы не можете безопасно использовать порядковые ссылки (хотя использование порядковых ссылок для столбцов - это плохая практика сама по себе)
8. В сложных запросах с полями `TEXT` запрос может быть замедлен с помощью менее оптимальной обработки таблицы temp

ВЫБЕРИТЕ ГДЕ

запрос

```
SELECT * FROM stack WHERE username = "admin" AND password = "admin";
```

Результат

```
+------+----------+----------+
| id   | username | password |
+------+----------+----------+
|    1 | admin    | admin    |
+------+----------+----------+
1 row in set (0.00 sec)
```

Запрос с вложенным SELECT в предложении WHERE

Предложение `WHERE` может содержать любой действительный `SELECT` для записи более сложных запросов. Это «вложенный» запрос

запрос

Вложенные запросы обычно используются для возврата одиночных атомных значений из запросов для сопоставлений.

```
SELECT title FROM books WHERE author_id = (SELECT id FROM authors WHERE last_name = 'Bar' AND
first_name = 'Foo');
```

Выбирает все имена пользователей без адреса электронной почты

```
SELECT * FROM stack WHERE username IN (SELECT username FROM signups WHERE email IS NULL);
```

Отказ от ответственности: рассмотрите возможность использования объединений для

повышения производительности при сравнении всего набора результатов.

SELECT с LIKE (%)

```
CREATE TABLE stack
(  id int AUTO_INCREMENT PRIMARY KEY,
   username VARCHAR(100) NOT NULL
);

INSERT stack(username) VALUES
('admin'),('k admin'),('adm'),('a adm b'),('b XadmY c'), ('adm now'), ('not here');
```

«adm» где угодно:

```
SELECT * FROM stack WHERE username LIKE "%adm%";
+----+-----------+
| id | username  |
+----+-----------+
|  1 | admin     |
|  2 | k admin   |
|  3 | adm       |
|  4 | a adm b   |
|  5 | b XadmY c |
|  6 | adm now   |
+----+-----------+
```

Начинается с "adm":

```
SELECT * FROM stack WHERE username LIKE "adm%";
+----+----------+
| id | username |
+----+----------+
|  1 | admin    |
|  3 | adm      |
|  6 | adm now  |
+----+----------+
```

Заканчивается знаком «adm»:

```
SELECT * FROM stack WHERE username LIKE "%adm";
+----+----------+
| id | username |
+----+----------+
|  3 | adm      |
+----+----------+
```

Так же, как символ `%` в предложении `LIKE` соответствует любому количеству символов, символ _ соответствует только одному символу. Например,

```
SELECT * FROM stack WHERE username LIKE "adm_n";
+----+----------+
| id | username |
+----+----------+
```

```
|  1 | admin    |
+----+----------+
```

Замечания по производительности Если есть указатель на `username` , то

- `LIKE 'adm'` выполняет то же самое, что и `= 'adm'
- `LIKE 'adm%` - это «диапазон», аналогичный `BETWEEN..AND.`. Он может эффективно использовать индекс в столбце.
- `LIKE '%adm'` (или любой вариант с *ведущим* шаблоном) не может использовать какой-либо индекс. Поэтому он будет медленным. На таблицах со многими строками он, вероятно, будет настолько медленным, что бесполезен.
- `RLIKE` (`REGEXP`) имеет тенденцию быть медленнее `LIKE` , но имеет больше возможностей.
- Хотя MySQL предлагает индексирование `FULLTEXT` во многих типах таблицы и столбца, эти индексы `FULLTEXT` *не* используются для выполнения запросов с использованием `LIKE` .

SELECT с псевдонимом (AS)

SQL-псевдонимы используются для временного переименования таблицы или столбца. Они обычно используются для улучшения удобочитаемости.

запрос

```
SELECT username AS val FROM stack;
SELECT username val FROM stack;
```

(Примечание: `AS` является синтаксически необязательным.)

Результат

```
+-------+
| val   |
+-------+
| admin |
| stack |
+-------+
2 rows in set (0.00 sec)
```

SELECT с предложением LIMIT

Запрос:

```
 SELECT *
   FROM Customers
  ORDER BY CustomerID
  LIMIT 3;
```

Результат:

Пользовательский ИД	Имя покупателя	Контактное лицо	Адрес	город	Почтовый индекс	Страна
1	Альфредс Футтеркисте	Мария Андерс	Obere Str. 57	Берлин	12209	Германия
2	Ana Trujillo Emparedados y helados	Ана Трухильо	Avda. de la Constitución 2222	México DF	05021	Мексика
3	Антонио Морено Такерия	Антонио Морено	Mataderos 2312	México DF	05023	Мексика

Лучшая практика Всегда используйте `ORDER BY` при использовании `LIMIT` ; в противном случае строки, которые вы получите, будут непредсказуемыми.

Запрос:

```
SELECT *
  FROM Customers
 ORDER BY CustomerID
 LIMIT 2,1;
```

Объяснение:

Когда предложение `LIMIT` содержит два числа, оно интерпретируется как `LIMIT offset,count` . Итак, в этом примере запрос пропускает две записи и возвращает один.

Результат:

Пользовательский ИД	Имя покупателя	Контактное лицо	Адрес	город	Почтовый индекс	Страна
3	Антонио Морено Такерия	Антонио Морено	Mataderos 2312	México DF	05023	Мексика

Замечания:

Значения в предложениях `LIMIT` должны быть константами; они могут не быть значениями столбцов.

ВЫБЕРИТЕ с DISTINCT

Предложение `DISTINCT` после `SELECT` исключает дубликаты строк из набора результатов.

```
CREATE TABLE `car`
(   `car_id` INT UNSIGNED NOT NULL PRIMARY KEY,
    `name` VARCHAR(20),
    `price` DECIMAL(8,2)
);

INSERT INTO CAR (`car_id`, `name`, `price`) VALUES (1, 'Audi A1', '20000');
INSERT INTO CAR (`car_id`, `name`, `price`) VALUES (2, 'Audi A1', '15000');
INSERT INTO CAR (`car_id`, `name`, `price`) VALUES (3, 'Audi A2', '40000');
INSERT INTO CAR (`car_id`, `name`, `price`) VALUES (4, 'Audi A2', '40000');

SELECT DISTINCT `name`, `price` FROM CAR;
+---------+----------+
| name    | price    |
+---------+----------+
| Audi A1 | 20000.00 |
| Audi A1 | 15000.00 |
| Audi A2 | 40000.00 |
+---------+----------+
```

`DISTINCT` работает во всех столбцах, чтобы предоставлять результаты, а не отдельные столбцы. Последнее часто является заблуждением новых разработчиков SQL. Короче говоря, значимость на уровне строки результирующего набора имеет значение, а не отчетность на уровне столбца. Чтобы визуализировать это, посмотрите на «Audi A1» в приведенном выше наборе результатов.

Для более поздних версий MySQL `DISTINCT` имеет последствия с его использованием наряду с `ORDER BY` . Параметр `ONLY_FULL_GROUP_BY` вступает в игру, как показано на следующей странице руководства по MySQL, озаглавленной « Обработка MySQL GROUP BY» .

SELECT с LIKE (_)

Символ _ в шаблоне предложения `LIKE` соответствует одному символу.

запрос

```
SELECT username FROM users WHERE users LIKE 'admin_';
```

Результат

```
+----------+
| username |
+----------+
| admin1   |
| admin2   |
| admin-   |
| adminA   |
+----------+
```

SELECT с CASE или IF

запрос

```
SELECT st.name,
       st.percentage,
       CASE WHEN st.percentage >= 35 THEN 'Pass' ELSE 'Fail' END AS `Remark`
FROM student AS st ;
```

Результат

```
+--------------------------------+
|   name   | percentage | Remark |
+--------------------------------+
|   Isha   |     67     |  Pass  |
|   Rucha  |     28     |  Fail  |
|   Het    |     35     |  Pass  |
|   Ansh   |     92     |  Pass  |
+--------------------------------+
```

Или с IF

```
SELECT st.name,
       st.percentage,
       IF(st.percentage >= 35, 'Pass', 'Fail') AS `Remark`
FROM student AS st ;
```

NB

```
IF(st.percentage >= 35, 'Pass', 'Fail')
```

> Это означает: IF st.percentage> = 35 имеет значение **TRUE, а** затем возвращает `'Pass'` ELSE return `'Fail'`

ВЫБЕРИТЕ МЕЖДУ

Вы можете использовать предложение BETWEEN, чтобы заменить комбинацию условий «больше, чем равных И меньше, чем равных».

Данные

```
+----+-----------+
| id | username  |
+----+-----------+
|  1 | admin     |
|  2 | root      |
|  3 | toor      |
|  4 | mysql     |
|  5 | thanks    |
|  6 | java      |
+----+-----------+
```

Запрос с операторами

```
SELECT * FROM stack WHERE id >= 2 and id <= 5;
```

Похожие запросы: BETWEEN

```
SELECT * FROM stack WHERE id BETWEEN 2 and 5;
```

Результат

```
+----+-----------+
| id | username  |
+----+-----------+
|  2 | root      |
|  3 | toor      |
|  4 | mysql     |
|  5 | thanks    |
+----+-----------+
4 rows in set (0.00 sec)
```

Заметка

> **BETWEEN использует `>=` и `<=` , not `>` и `<` .**

Использование НЕ МЕЖДУ

Если вы хотите использовать негатив, вы можете использовать `NOT` . Например :

```
SELECT * FROM stack WHERE id NOT BETWEEN 2 and 5;
```

Результат

```
+----+-----------+
| id | username  |
+----+-----------+
|  1 | admin     |
|  6 | java      |
+----+-----------+
2 rows in set (0.00 sec)
```

Заметка

> **NOT BETWEEN использует `>` и `<` и не `>=` и `<=`** То есть `WHERE id NOT BETWEEN 2 and 5` совпадает с `WHERE (id < 2 OR id > 5)` .

Если у вас есть индекс в столбце, который вы используете в поиске `BETWEEN` , MySQL может использовать этот индекс для сканирования диапазона.

SELECT с диапазоном дат

```
SELECT ... WHERE dt >= '2017-02-01'
             AND dt  < '2017-02-01' + INTERVAL 1 MONTH
```

Конечно, это можно сделать с `BETWEEN` и включением `23:59:59` . Но эта модель имеет следующие преимущества:

- Вы не предварительно рассчитали дату окончания (которая часто является точной длиной с самого начала)
- Вы не включаете обе конечные точки (как это делает `BETWEEN`), и не вводите «23: 59: 59», чтобы избежать этого.
- Он работает для `DATE` , `TIMESTAMP` , `DATETIME` и даже для `DATETIME(6)` включенных в микросекунду.
- Он заботится о високосных днях, конце года и т. Д.
- Он удобен для индекса (так же `BETWEEN`).

Прочитайте ВЫБРАТЬ онлайн: https://riptutorial.com/ru/mysql/topic/3307/выбрать

глава 19: Группа по

Синтаксис

1. SELECT expression1, expression2, ... expression_n,
2. aggregate_function (выражение)
3. ОТ таблиц
4. [ГДЕ условия]
5. GROUP BY expression1, expression2, ... expression_n;

параметры

параметр	ПОДРОБНОСТИ
expression1, expression2, ... expression_n	Выражения, которые не заключены в агрегированную функцию и должны быть включены в предложение GROUP BY.
aggregate_function	Функция, такая как функции SUM, COUNT, MIN, MAX или AVG.
таблицы	он таблицы, из которых вы хотите получить записи. Должна быть хотя бы одна таблица, перечисленная в предложении FROM.
ГДЕ условия	Необязательный. Условия, которые должны быть выполнены для выбранных записей.

замечания

Предложение MySQL GROUP BY используется в инструкции SELECT для сбора данных по нескольким записям и группировки результатов по одному или нескольким столбцам.

Его поведение частично определяется значением переменной `ONLY_FULL_GROUP_BY` . Когда это разрешено, `SELECT` которые группируются по любому столбцу, не входящему в выход, возвращают ошибку. (Это значение по умолчанию равно 5.7.5 .) Как установка, так и не настройка этой переменной могут вызвать проблемы для наивных пользователей или пользователей, привыкших к другим СУБД.

Examples

ГРУППА С ИСПОЛЬЗОВАНИЕМ СУММЫ Функция

```
SELECT product, SUM(quantity) AS "Total quantity"
FROM order_details
GROUP BY product;
```

Группа с помощью функции MIN

Предположим, что таблица сотрудников, в которой каждая строка является сотрудником, который имеет `name` , `department` и `salary` .

```
SELECT department, MIN(salary) AS "Lowest salary"
FROM employees
GROUP BY department;
```

Это сообщит вам, какой отдел содержит сотрудника с самой низкой зарплатой и какую зарплату. Поиск `name` сотрудника с наименьшей зарплатой в каждом отделе - это другая проблема, выходящая за рамки этого примера. См. «Максимальный размер группы».

ГРУППА ПО ИСПОЛЬЗОВАНИЮ COUNT Функция

```
SELECT department, COUNT(*) AS "Man_Power"
FROM employees
GROUP BY department;
```

ГРУППА ПО ИСПОЛЬЗОВАНИЮ

```
SELECT department, COUNT(*) AS "Man_Power"
FROM employees
GROUP BY department
HAVING COUNT(*) >= 10;
```

Использование `GROUP BY ... HAVING` для фильтрации агрегатных записей аналогично использованию `SELECT ... WHERE` для фильтрации отдельных записей.

Вы также можете сказать, что `HAVING Man_Power >= 10` так как `HAVING` понимает «псевдонимы».

Группа с помощью Group Concat

Группа Concat используется в MySQL для получения конкатенированных значений выражений с более чем одним результатом для каждого столбца. Значит, есть много строк, которые нужно выбрать для одного столбца, например `Name(1):Score(*)`

название	Гол
Адам	A +

название	Гол
Адам	A-
Адам	B
Адам	C +
Билл	D-
Джон	A-

```
SELECT Name, GROUP_CONCAT(Score ORDER BY Score desc SEPERATOR ' ') AS Grades
FROM   Grade
GROUP BY Name
```

Результаты:

```
+------+------------+
| Name | Grades     |
+------+------------+
| Adam | C+ B A- A+ |
| Bill | D-         |
| John | A-         |
+------+------------+
```

GROUP BY с функциями AGGREGATE

Таблица ЗАКАЗОВ

```
+---------+------------+----------+-------+--------+
| orderid | customerid | customer | total | items  |
+---------+------------+----------+-------+--------+
|       1 |          1 | Bob      |  1300 |     10 |
|       2 |          3 | Fred     |   500 |      2 |
|       3 |          5 | Tess     |  2500 |      8 |
|       4 |          1 | Bob      |   300 |      6 |
|       5 |          2 | Carly    |   800 |      3 |
|       6 |          2 | Carly    |  1000 |     12 |
|       7 |          3 | Fred     |   100 |      1 |
|       8 |          5 | Tess     | 11500 |     50 |
|       9 |          4 | Jenny    |   200 |      2 |
|      10 |          1 | Bob      |   500 |     15 |
+---------+------------+----------+-------+--------+
```

- **COUNT**

Возвратите **количество строк** , удовлетворяющих определенным критериям в `WHERE` .

Например: количество заказов для каждого клиента.

```
SELECT customer, COUNT(*) as orders
```

```
FROM orders
GROUP BY customer
ORDER BY customer
```

Результат:

```
+----------+--------+
| customer | orders |
+----------+--------+
| Bob      |      3 |
| Carly    |      2 |
| Fred     |      2 |
| Jenny    |      1 |
| Tess     |      2 |
+----------+--------+
```

- **SUM**

Возвращает **сумму** выбранного столбца.

Например: сумма общей суммы и предметов для каждого клиента.

```
SELECT customer, SUM(total) as sum_total, SUM(items) as sum_items
FROM orders
GROUP BY customer
ORDER BY customer
```

Результат:

```
+----------+-----------+-----------+
| customer | sum_total | sum_items |
+----------+-----------+-----------+
| Bob      |      2100 |        31 |
| Carly    |      1800 |        15 |
| Fred     |       600 |         3 |
| Jenny    |       200 |         2 |
| Tess     |     14000 |        58 |
+----------+-----------+-----------+
```

- **AVG**

Возвращает **среднее** значение столбца числового значения.

Например: средняя стоимость заказа для каждого клиента.

```
SELECT customer, AVG(total) as avg_total
FROM orders
GROUP BY customer
ORDER BY customer
```

Результат:

```
+----------+-----------+
```

```
| customer | avg_total |
+----------+-----------+
| Bob      |       700 |
| Carly    |       900 |
| Fred     |       300 |
| Jenny    |       200 |
| Tess     |      7000 |
+----------+-----------+
```

- **МАКСИМУМ**

Вернуть **максимальное** значение определенного столбца или выражения.

Например: максимальный заказ для каждого клиента.

```
SELECT customer, MAX(total) as max_total
FROM orders
GROUP BY customer
ORDER BY customer
```

Результат:

```
+----------+-----------+
| customer | max_total |
+----------+-----------+
| Bob      |      1300 |
| Carly    |      1000 |
| Fred     |       500 |
| Jenny    |       200 |
| Tess     |     11500 |
+----------+-----------+
```

- **MIN**

Возврат **наименьшего** значения определенного столбца или выражения.

Например: минимальный заказ для каждого клиента.

```
SELECT customer, MIN(total) as min_total
FROM orders
GROUP BY customer
ORDER BY customer
```

Результат:

```
+----------+-----------+
| customer | min_total |
+----------+-----------+
| Bob      |       300 |
| Carly    |       800 |
| Fred     |       100 |
| Jenny    |       200 |
| Tess     |      2500 |
+----------+-----------+
```

Прочитайте Группа по онлайн: https://riptutorial.com/ru/mysql/topic/3523/группа-по

глава 20: Двигатель MyISAM

замечания

На протяжении многих лет InnoDB улучшался до такой степени, что почти всегда лучше, чем MyISAM, по крайней мере поддерживаемые в настоящее время версии. Итог: не используйте MyISAM, за исключением, может быть, таблиц, которые действительно временны.

Одно из преимуществ MyISAM над InnoDB остается: оно на 2x-3x меньше в пространстве, требуемом на диске.

Когда InnoDB впервые появился, MyISAM все еще был жизнеспособным Engine. Но с появлением XtraDB и 5.6, InnoDB стал «лучше», чем MyISAM в большинстве тестов.

Ходят слухи, что следующая основная версия устранит необходимость в MyISAM, сделав по-настоящему временные таблицы InnoDB и переместив системные таблицы в InnoDB.

Examples

ДВИГАТЕЛЬ = MyISAM

```
CREATE TABLE foo (
     ...
) ENGINE=MyISAM;
```

Прочитайте Двигатель MyISAM онлайн: https://riptutorial.com/ru/mysql/topic/4710/двигатель-myisam

глава 21: Динамическая сводная таблица с использованием подготовленного заявления

Examples

Un-pivot динамический набор столбцов на основе условия

Следующий пример - очень полезная основа, когда вы пытаетесь преобразовать данные транзакции в неперевернутые данные для целей BI / отчетности, где размеры, которые должны быть не подвержены повороту, могут иметь динамический набор столбцов.

В нашем примере мы полагаем, что таблица необработанных данных содержит данные оценки сотрудников в виде отмеченных вопросов.

Таблица необработанных данных следующая:

```
create table rawdata

(
 PersonId VARCHAR(255)
,Question1Id INT(11)
,Question2Id INT(11)
,Question3Id INT(11)
)
```

Таблица rawdata является временной таблицей как частью процедуры ETL и может иметь различное количество вопросов. Цель состоит в том, чтобы использовать одну и ту же процедуру без поворота для произвольного количества Вопросов, а именно для столбцов, которые будут не поворачиваться.

Ниже приведен пример игрушечной таблицы:

PersonId	Question1Id	Question2Id	Question3Id
Giannaros	1	3	1
Patra	2	4	3

Известный статический способ отключить данные в MYSQL - это использовать UNION ALL:

```
create table unpivoteddata

(

 PersonId VARCHAR(255)
,QuestionId VARCHAR(255)
```

```
,QuestionValue INT(11)

);

INSERT INTO unpivoteddata SELECT PersonId, 'Question1Id' col, Question1Id
FROM rawdata
UNION ALL
SELECT PersonId, 'Question2Id' col, Question2Id
FROM rawdata
UNION ALL
SELECT PersonId, 'Question3Id' col, Question3Id
FROM rawdata;
```

В нашем случае мы хотим определить способ отказа от произвольного количества столбцов QuestionId. Для этого нам нужно выполнить подготовленный оператор, который представляет собой динамический выбор нужных столбцов. Чтобы иметь возможность выбирать, какие столбцы должны быть не развернуты, мы будем использовать оператор GROUP_CONCAT, и мы выберем столбцы, для которых тип данных имеет значение «int». В GROUP_CONCAT мы также включаем все дополнительные элементы нашей инструкции SELECT для выполнения.

```
set @temp2 = null;

SELECT GROUP_CONCAT(' SELECT ', 'PersonId',',','''',COLUMN_NAME,'''', ' col
',',',COLUMN_NAME,' FROM rawdata' separator ' UNION ALL' ) FROM     INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'rawdata' AND DATA_TYPE = 'Int' INTO     @temp2;

select @temp2;
```

В другом случае мы могли бы выбрать столбцы, имена столбцов которых соответствуют шаблону, например, вместо

```
DATA_TYPE = 'Int'
```

использование

```
COLUMN_NAME LIKE 'Question%'
```

или что-то подходящее, которое можно контролировать через фазу ETL.

Подготовленное заявление завершается следующим образом:

```
set @temp3 = null;

select concat('INSERT INTO unpivoteddata',@temp2) INTO @temp3;

select @temp3;

prepare stmt FROM @temp3;
execute stmt;
deallocate prepare stmt;
```

Таблица неавторизованных данных следующая:

```
SELECT * FROM unpivoteddata
```

PersonId	QuestionId	QuestionValue
Giannaros	Question1Id	1
Patra	Question1Id	2
Giannaros	Question2Id	3
Patra	Question2Id	4
Giannaros	Question3Id	1
Patra	Question3Id	3

Выбор столбцов в соответствии с условием, а затем создание подготовленного оператора - эффективный способ динамически разворачивать данные.

Прочитайте Динамическая сводная таблица с использованием подготовленного заявления онлайн: https://riptutorial.com/ru/mysql/topic/6491/динамическая-сводная-таблица-с-использованием-подготовленного-заявления

глава 22: ЗАГРУЗКА ДАННЫХ

Синтаксис

1. LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'имя_файла'
2. INTO TABLE tbl_name
3. [CHARACTER SET charset]
4. [{FIELDS | COLUMNS} [TERMINATED BY 'string'] [[OPTIONALLY] ENCLOSED BY 'char']]
5. [LINES [STARTING BY 'string]] [TERMINATED by' string ']]
6. [Номер IGNORE {LINES | ЧСТРОК}]
7. [(Col_name_or_user_var, ...)]
8. [SET col_name = expr, ...]

Examples

используя LOAD DATA INFILE для загрузки большого количества данных в базу данных

Рассмотрим следующий пример, предполагая, что для загрузки в вашу базу данных имеется CS;

```
1;max;male;manager;12-7-1985
2;jack;male;executive;21-8-1990
.
.
.
1000000;marta;female;accountant;15-6-1992
```

Создайте таблицу для вставки.

```
CREATE TABLE `employee` ( `id` INT NOT NULL ,
                          `name` VARCHAR NOT NULL,
                          `sex` VARCHAR NOT NULL ,
                          `designation` VARCHAR NOT NULL ,
                          `dob` VARCHAR NOT NULL   );
```

Используйте следующий запрос, чтобы вставить значения в эту таблицу.

```
LOAD DATA INFILE 'path of the file/file_name.txt'
INTO TABLE employee
FIELDS TERMINATED BY ';' //specify the delimiter separating the values
LINES TERMINATED BY '\r\n'
(id,name,sex,designation,dob)
```

Рассмотрим случай, когда формат даты является нестандартным.

```
1;max;male;manager;17-Jan-1985
2;jack;male;executive;01-Feb-1992
.
.
.
1000000;marta;female;accountant;25-Apr-1993
```

В этом случае вы можете изменить формат столбца `dob` прежде чем вставлять это.

```
LOAD DATA INFILE 'path of the file/file_name.txt'
INTO TABLE employee
FIELDS TERMINATED BY ';' //specify the delimiter separating the values
LINES TERMINATED BY '\r\n'
(id,name,sex,designation,@dob)
SET date = STR_TO_DATE(@date, '%d-%b-%Y');
```

В этом примере LOAD DATA INFILE не указаны все доступные функции.

Вы можете увидеть больше ссылок на LOAD DATA INFILE здесь .

Импорт CSV-файла в таблицу MySQL

Следующая команда импортирует CSV-файлы в таблицу MySQL с одинаковыми столбцами при соблюдении правил цитирования и экранирования CSV.

```
load data infile '/tmp/file.csv'
into table my_table
fields terminated by ','
optionally enclosed by '"'
escaped by '"'
lines terminated by '\n'
ignore 1 lines; -- skip the header row
```

Загружать данные с помощью дубликатов

Если вы используете команду `LOAD DATA INFILE` для заполнения таблицы существующими данными, вы часто обнаружите, что импорт `LOAD DATA INFILE` неудачно из-за дубликатов. Существует несколько возможных путей преодоления этой проблемы.

ЛОКАЛЬНЫЕ ДАННЫЕ

Если этот параметр включен на вашем сервере, его можно использовать для загрузки файла, который существует на клиентском компьютере, а не на сервере. Побочным эффектом является то, что повторяющиеся строки для уникальных значений игнорируются.

```
LOAD DATA LOCAL INFILE 'path of the file/file_name.txt'
INTO TABLE employee
```

LOAD DATA INFILE 'fname' REPLACE

Когда используется ключевое слово replace, дубликаты уникальных или первичных ключей приведут к замене существующей строки на новые

```
LOAD DATA INFILE 'path of the file/file_name.txt'
REPLACE INTO TABLE employee
```

LOAD DATA INFILE 'fname' IGNORE

Противоположность `REPLACE` , существующие строки будут сохранены и новые игнорируются. Такое поведение аналогично `LOCAL` описанному выше. Однако файл не должен существовать на клиентском компьютере.

```
LOAD DATA INFILE 'path of the file/file_name.txt'
IGNORE INTO TABLE employee
```

Загрузка через посреднический стол

Иногда игнорирование или замена всех дубликатов не может быть идеальным вариантом. Возможно, вам придется принимать решения на основе содержимого других столбцов. В этом случае лучшим вариантом является загрузка в промежуточную таблицу и перенос оттуда.

```
INSERT INTO employee SELECT * FROM intermediary WHERE ...
```

импорт Экспорт

Импортировать

```
SELECT a,b,c  INTO OUTFILE 'result.txt' FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n' FROM table;
```

экспорт

```
LOAD DATA INFILE 'result.txt' INTO TABLE table;
```

Прочитайте ЗАГРУЗКА ДАННЫХ онлайн: https://riptutorial.com/ru/mysql/topic/2356/загрузка-данных

глава 23: Зарезервированные слова

Вступление

MySQL имеет некоторые специальные имена, называемые *зарезервированными словами* . Зарезервированное слово может использоваться как идентификатор для таблицы, столбца и т. Д., Только если оно завернуто в backticks (`), иначе это приведет к ошибке.

Чтобы избежать таких ошибок, либо не используйте зарезервированные слова в качестве идентификаторов, либо оберните идентификатор нарушения в обратные ссылки.

замечания

Ниже перечислены все зарезервированные слова (из официальной документации):

- ДОСТУПНОЕ
- ДОБАВЛЯТЬ
- ВСЕ
- ALTER
- ПРОАНАЛИЗИРУЙТЕ
- А ТАКЖЕ
- КАК
- ASC
- ASENSITIVE
- ДО
- МЕЖДУ
- BIGINT
- BINARY
- большой двоичный объект
- И ТО И ДРУГОЕ
- ОТ
- ВЫЗОВ
- CASCADE
- ДЕЛО
- МЕНЯТЬ
- CHAR
- ПЕРСОНАЖ
- ПРОВЕРЯТЬ
- СОРТИРОВКА
- КОЛОНКА
- СОСТОЯНИЕ
- CONSTRAINT

- ПРОДОЛЖИТЬ
- ПЕРЕРАБАТЫВАТЬ
- СОЗДАЙТЕ
- ПЕРЕСЕКАТЬ
- ТЕКУЩАЯ ДАТА
- ТЕКУЩЕЕ ВРЕМЯ
- CURRENT_TIMESTAMP
- ТЕКУЩИЙ ПОЛЬЗОВАТЕЛЬ
- КУРСОР
- БАЗА ДАННЫХ
- БАЗ
- DAY_HOUR
- DAY_MICROSECOND
- DAY_MINUTE
- DAY_SECOND
- декабрь
- ДЕСЯТИЧНЫЙ
- DECLARE
- ДЕФОЛТ
- ЗАДЕРЖИВАЕТСЯ
- УДАЛЯТЬ
- DESC
- ОПИСАНИЯ
- DETERMINISTIC
- DISTINCT
- DISTINCTROW
- DIV
- DOUBLE
- DROP
- DUAL
- КАЖДЫЙ
- ELSE
- ELSEIF
- ENCLOSED
- ESCAPED
- СУЩЕСТВУЕТ
- ВЫХОД
- EXPLAIN
- ЛОЖНЫЙ
- FETCH
- FLOAT
- float4
- float8
- ЗА
- FORCE

- ИНОСТРАННЫЕ
- ОТ
- ПОЛНЫЙ ТЕКСТ
- СГЕНЕРИРОВАННЫМИ
- ПОЛУЧИТЬ
- ГРАНТ
- GROUP
- HAVING
- ВЫСОКИЙ ПРИОРИТЕТ
- HOUR_MICROSECOND
- HOUR_MINUTE
- HOUR_SECOND
- ЕСЛИ
- ИГНОРИРУЙТЕ
- В
- ИНДЕКС
- INFILE
- ВНУТРЕННИЙ
- INOUT
- INSENSITIVE
- ВСТАВИТЬ
- INT
- INT1
- INT2
- INT3
- INT4
- INT8
- INTEGER
- ИНТЕРВАЛ
- В
- IO_AFTER_GTIDS
- IO_BEFORE_GTIDS
- ЯВЛЯЕТСЯ
- ITERATE
- ПРИСОЕДИНИТЬСЯ
- KEY
- КЛЮЧИ
- УБИЙСТВО
- ВЕДУЩИЙ
- ПОКИДАТЬ
- ОСТАВИЛ
- ЛАЙК
- ПРЕДЕЛ
- ЛИНЕЙНАЯ

- ЛИНИИ
- НАГРУЗКИ
- МЕСТНОЕ ВРЕМЯ
- LOCALTIMESTAMP
- ЗАМОК
- ДОЛГО
- LONGBLOB
- LONGTEXT
- LOOP
- НИЗКИЙ ПРИОРИТЕТ
- MASTER_BIND
- MASTER_SSL_VERIFY_SERVER_CERT
- МАТЧ
- MAXVALUE
- MEDIUMBLOB
- MEDIUMINT
- MEDIUMTEXT
- MIDDLEINT
- MINUTE_MICROSECOND
- MINUTE_SECOND
- MOD
- модифицирует
- ПРИРОДНЫЙ
- НЕ
- NO_WRITE_TO_BINLOG
- НОЛЬ
- NUMERIC
- НА
- ОПТИМИЗАЦИЯ
- OPTIMIZER_COSTS
- ВАРИАНТ
- ПО ВЫБОРУ
- ИЛИ ЖЕ
- ПОРЯДОК
- ИЗ
- ВНЕШНИЙ
- OUTFILE
- PARTITION
- ТОЧНОСТЬ
- ПЕРВИЧНЫЙ
- ПРОЦЕДУРА
- ПРОДУВКА
- СПЕКТР
- ЧИТАТЬ

- ЧИТАЕТ
- ЧИТАЙ ПИШИ
- РЕАЛЬНЫЙ
- РЕКОМЕНДАЦИИ
- REGEXP
- РЕЛИЗ
- ПЕРЕИМЕНОВАТЬ
- ПОВТОРЕНИЕ
- ЗАМЕНА
- ТРЕБУЕТСЯ
- RESIGNAL
- RESTRICT
- ВЕРНУТЬ
- КЕУОКЕ
- ПРАВО
- RLIKE
- SCHEMA
- SCHEMAS
- SECOND_MICROSECOND
- ВЫБРАТЬ
- Чувствительная
- РАЗДЕЛИТЕЛЬ
- ЗАДАВАТЬ
- ШОУ
- СИГНАЛ
- SMALLINT
- ПРОСТРАНСТВЕННО
- КОНКРЕТНЫЙ
- SQL
- SqlException
- SQLSTATE
- SQLWARNING
- SQL_BIG_RESULT
- SQL_CALC_FOUND_ROWS
- SQL_SMALL_RESULT
- SSL
- ЗАПУСК
- ЗАПОМНЕННАЯ
- STRAIGHT_JOIN
- ТАБЛИЦА
- ОТМЕНЯЛОСЬ
- ЗАТЕМ
- TINYBLOB
- TINYINT

- TINYTEXT
- K
- TRAILING
- СПУСКОВОЙ КРЮЧОК
- ПРАВДА
- UNDO
- UNION
- УНИКАЛЬНАЯ
- ОТКРЫТЬ
- UNSIGNED
- ОБНОВИТЬ
- ИСПОЛЬЗОВАНИЕ
- ИСПОЛЬЗОВАНИЕ
- С ПОМОЩЬЮ
- UTC_DATE
- UTC_TIME
- UTC_TIMESTAMP
- ЦЕННОСТИ
- VARBINARY
- VARCHAR
- VARCHARACTER
- VARYING
- ВИРТУАЛЬНЫЙ
- КОГДА
- ГДЕ
- В ТО ВРЕМЯ КАК
- С
- ЗАПИСЫВАТЬ
- XOR
- ГОД МЕСЯЦ
- ZEROFILL
- СГЕНЕРИРОВАННЫМИ
- OPTIMIZER_COSTS
- ЗАПОМНЕННАЯ
- ВИРТУАЛЬНЫЙ

Examples

Ошибки из-за зарезервированных слов

При попытке выбрать из таблицы, называемой `order` подобным этому

```
select * from order
```

ошибка возрастает:

> Код ошибки: 1064. У вас есть ошибка в синтаксисе SQL; проверьте руководство, соответствующее версии вашего сервера MySQL, для правильного синтаксиса для использования рядом с «порядком» в строке 1

Зарезервированные ключевые слова в MySQL должны быть экранированы с backticks (`)

```
select * from `order`
```

различать ключевое слово и имя таблицы или столбца.

См. Также: Синтаксическая ошибка из-за использования зарезервированного слова в качестве имени таблицы или столбца в MySQL .

Прочитайте Зарезервированные слова онлайн: https://riptutorial.com/ru/mysql/topic/1398/зарезервированные-слова

глава 24: Извлечение значений из типа JSON

Вступление

MySQL 5.7.8+ поддерживает собственный тип JSON. Хотя у вас есть разные способы создания json-объектов, вы можете просматривать и читать их по-разному.

Основной функцией является `JSON_EXTRACT` , поэтому операторы `->` и `->>` более дружелюбны.

Синтаксис

- JSON_EXTRACT (json_doc, путь [...])
- JSON_EXTRACT (json_doc, путь)
- JSON_EXTRACT (json_doc, путь1, путь2)

параметры

параметр	Описание
json_doc	действительный документ JSON
дорожка	путь участников

замечания

Упоминается в MySQL 5.7 Справочное руководство

- Несколько согласованных значений по аргументам пути

  Если возможно, что эти аргументы могут возвращать несколько значений, сопоставленные значения автоматически переносятся в виде массива в порядке, соответствующем путям, которые их создавали. В противном случае возвращаемое значение будет единственным согласованным значением.

- `NULL` Результат, когда:
  - любой аргумент - NULL
  - путь не соответствует

  Возвращает NULL, если любой аргумент равен NULL, или нет путей для определения значения в документе.

Examples

Чтение значения JSON Array

Создайте переменную @myjson как тип JSON (подробнее):

```
SET @myjson = CAST('["A","B",{"id":1,"label":"C"}]' as JSON) ;
```

`SELECT` некоторых членов!

```
SELECT
  JSON_EXTRACT( @myjson , '$[1]' ) ,
  JSON_EXTRACT( @myjson , '$[*].label') ,
  JSON_EXTRACT( @myjson , '$[1].*' ) ,
  JSON_EXTRACT( @myjson , '$[2].*')
;
-- result values:
'\"B\"', '[\"C\"]', NULL, '[1, \"C\"]'
-- visually:
"B", ["C"], NULL, [1, "C"]
```

Операторы извлечения JSON

Извлечь `path` помощью `->` или `->>` Операторы, а `->>` значение UNQUOTED:

```
SELECT
  myjson_col->>'$[1]' , myjson_col->'$[1]' ,
  myjson_col->>'$[*].label' ,
  myjson_col->>'$[1].*' ,
  myjson_col->>'$[2].*'
FROM tablename ;
 -- visuall:
    B, "B" , ["C"], NULL, [1, "C"]
 --^^^ ^^^
```

Таким образом, `col->>path` равен `JSON_UNQUOTE(JSON_EXTRACT(col,path))` :

> Как и в случае ->, оператор - >> всегда выводится в вывод EXPLAIN, как показано в следующем примере:

```
mysql> EXPLAIN SELECT c->>'$.name' AS name
    ->     FROM jemp WHERE g > 2\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: jemp
   partitions: NULL
         type: range
possible_keys: i
          key: i
      key_len: 5
          ref: NULL
```

```
         rows: 2
     filtered: 100.00
        Extra: Using where
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: /* select#1 */ select
json_unquote(json_extract(`jtest`.`jemp`.`c`,'$.name')) AS `name` from
`jtest`.`jemp` where (`jtest`.`jemp`.`g` > 2)
1 row in set (0.00 sec)
```

Читайте о выводе строки inline (+)

Прочитайте Извлечение значений из типа JSON онлайн:
https://riptutorial.com/ru/mysql/topic/9042/извлечение-значений-из-типа-json

глава 25: Изменить пароль

Examples

Изменение пароля пользователя MySQL в Linux

Чтобы изменить пароль пользователя root root:

Шаг 1. Остановите сервер MySQL.

- в Ubuntu или Debian:
  `sudo /etc/init.d/mysql stop`
- в CentOS, Fedora или Red Hat Enterprise Linux:
  `sudo /etc/init.d/mysqld stop`

Шаг 2. Запустите сервер MySQL без системы привилегий.

```
sudo mysqld_safe --skip-grant-tables &
```

или, если `mysqld_safe` недоступен,

```
sudo mysqld --skip-grant-tables &
```

Шаг 3. Подключение к серверу MySQL.

```
mysql -u root
```

Шаг 4. Установите новый пароль для пользователя root.

5,7

```
FLUSH PRIVILEGES;
ALTER USER 'root'@'localhost' IDENTIFIED BY 'new_password';
FLUSH PRIVILEGES;
exit;
```

5,7

```
FLUSH PRIVILEGES;
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('new_password');
FLUSH PRIVILEGES;
exit;
```

Примечание. Синтаксис `ALTER USER` был представлен в MySQL 5.7.6.

Шаг 5: Перезапустите сервер MySQL.

- в Ubuntu или Debian:
  ```
  sudo /etc/init.d/mysql stop
  sudo /etc/init.d/mysql start
  ```
- в CentOS, Fedora или Red Hat Enterprise Linux:
  ```
  sudo /etc/init.d/mysqld stop
  sudo /etc/init.d/mysqld start
  ```

Изменение пароля пользователя MySQL в Windows

Когда мы хотим изменить пароль root в Windows, нам необходимо выполнить следующие шаги:

Шаг 1. Запустите свою командную строку, используя любой из следующих способов:

Perss `Crtl+R` или `Start Menu > Run` а затем введите `cmd` и нажмите « `Start Menu > Run`

Шаг 2. Измените каталог, где установлен `MYSQL` В моем случае это

```
C:\> cd C:\mysql\bin
```

Шаг 3. Теперь нам нужно запустить командную строку `mysql`

```
C:\mysql\bin> mysql -u root mysql
```

Шаг 4: запрос Fire для изменения пароля `root`

```
mysql> SET PASSWORD FOR root@localhost=PASSWORD('my_new_password');
```

Процесс

1. Остановите процесс сервера / демона MySQL (mysqld).
2. Запустите процесс MySQL-сервера с параметром -skip-grant-tables, чтобы он не запрашивал пароль: `mysqld_safe --skip-grant-tables &`
3. Подключитесь к серверу MySQL как пользователь root: `mysql -u root`
4. Изменить пароль:

- (5.7.6 и новее): `ALTER USER 'root'@'localhost' IDENTIFIED BY 'new-password';`
- (5.7.5 и старше, или MariaDB): `SET PASSWORD FOR 'root'@'localhost' = PASSWORD('new-password); flush privileges; quit;`

5. Перезапустите сервер MySQL.

Примечание: это будет работать только в том случае, если вы физически находитесь на одном сервере.

Online Doc: http://dev.mysql.com/doc/refman/5.7/en/resetting-permissions.html

Прочитайте Изменить пароль онлайн: https://riptutorial.com/ru/mysql/topic/2761/изменить-

пароль

глава 26: Индексы и ключи

Синтаксис

- - Создать простой индекс

  CREATE INDEX *index_name* ON *table_name* (*column_name1* [, *column_name2* , ...])

- - Создать уникальный индекс

  CREATE UNIQUE INDEX *index_name* ON *table_name* (*column_name1* [, *column_name2* , ...]

- - Индекс падения

  DROP INDEX **index_name** ON **tbl_name** [**algorithm_option** | **lock_option**] ...

  algorithm_option: ALGORITHM [=] {DEFAULT | INPLACE | COPY}

  lock_option: LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}

замечания

Концепции

Индекс в таблице MySQL работает как индекс в книге.

Допустим, у вас есть книга о базах данных, и вы хотите найти некоторую информацию о хранилище. Без индекса (без какой-либо другой помощи, такой как оглавление) вам придется проходить страницы один за другим, пока не найдете тему (это «полное сканирование таблицы»). С другой стороны, индекс имеет список ключевых слов, поэтому вы проконсультируетесь с индексом и увидите, что хранилище упомянуто на страницах 113-120, 231 и 354. Затем вы можете напрямую переходить на эти страницы без поиска (это поиск с индексом, несколько быстрее).

Конечно, полезность индекса зависит от многих вещей - несколько примеров, используя сравнение выше:

- Если у вас есть книга о базах данных и индексировано слово «база данных», вы можете увидеть, что это упоминается на страницах 1-59, 61-290 и 292-400. Это много страниц, и в таком случае индекс не очень помогает, и, возможно, быстрее будет проходить страницы один за другим. (В базе данных это «низкая избирательность».)
- Для 10-страничной книги нет смысла делать индекс, так как вы можете получить 10-

страничную книгу с префиксом 5 страниц, что просто глупо - просто сканируйте 10 страниц и сделайте с ними ,
- Индекс также должен быть полезен - обычно нет смысла индексировать, например, частоту буквы «L» на странице.

Examples

Создать индекс

```
-- Create an index for column 'name' in table 'my_table'
CREATE INDEX idx_name ON my_table(name);
```

Создать уникальный индекс

Уникальный индекс предотвращает вставку дублированных данных в таблицу. Значения `NULL` могут быть вставлены в столбцы, которые составляют часть уникального индекса (так как по определению значение `NULL` отличается от любого другого значения, включая другое значение `NULL`)

```
-- Creates a unique index for column 'name' in table 'my_table'
CREATE UNIQUE INDEX idx_name ON my_table(name);
```

Индекс падения

```
-- Drop an index for column 'name' in table 'my_table'
DROP INDEX idx_name ON my_table;
```

Создание комбинированного индекса

Это создаст составной индекс обоих ключей, `mystring` и `mydatetime` и ускорит запросы с обоими столбцами в `WHERE` .

```
CREATE INDEX idx_mycol_myothercol ON my_table(mycol, myothercol)
```

Примечание: порядок важен! Если поисковый запрос не включает оба столбца в `WHERE` , он может использовать только самый левый индекс. В этом случае запрос с `mycol` в `WHERE` будет использовать индекс, поиск запроса для `myothercol` **без** также ищу `mycol` **не** будет. Для получения дополнительной информации ознакомьтесь с этим сообщением в блоге .

Примечание. В связи с работой BTREE столбцы, которые обычно запрашиваются в диапазонах, должны иметь самое правое значение. Например, столбцы `DATETIME` обычно запрашиваются как `WHERE datecol > '2016-01-01 00:00:00'` . Индексы BTREE обрабатывают диапазоны очень эффективно, но только если столбец, запрашиваемый как диапазон, является последним в составном индексе.

Кнопка AUTO_INCREMENT

```
CREATE TABLE (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    ...
    PRIMARY KEY(id),
    ...  );
```

Основные примечания:

- Начинается с 1 и увеличивается на 1 автоматически, когда вы не можете указать его на `INSERT` или указать его как `NULL` .
- Иды всегда отличаются друг от друга, но ...
- Не делайте никаких предположений (без пробелов, последовательно генерируемых, не повторно используемых и т. Д.) О значениях id, отличных от уникальных в любой данный момент времени.

Тонкие ноты:

- При перезапуске сервера значение «next» «вычисляется» как `MAX(id)+1` .
- Если последняя операция перед отключением или сбоем заключалась в удалении самого высокого идентификатора, этот идентификатор *может* быть повторно использован (это зависит от двигателя). Таким образом, *не доверяйте auto_increments, чтобы быть постоянно уникальными* ; они уникальны в любой момент.
- Для многомашинных или кластерных решений см. `auto_increment_offset` и `auto_increment_increment` .
- Это нормально, если что-то другое, как `PRIMARY KEY` и просто делать `INDEX(id)` . (Это оптимизация в некоторых ситуациях.)
- Использование `AUTO_INCREMENT` в качестве «ключа `PARTITION` » редко полезно; сделать что-то другое.
- Различные операции *могут* «записывать» значения. Это происходит, когда они предварительно выделяют значения (-ы), а затем не используют их: `INSERT IGNORE` (с ключом dup), `REPLACE` (который `DELETE` плюс `INSERT`) и другие. `ROLLBACK` - еще одна причина для пробелов в идентификаторах.
- В репликации вы не можете доверять идентификаторам, чтобы достигнуть ведомого (-ов) в порядке возрастания. Хотя идентификаторы назначаются в последовательном порядке, операторы InnoDB отправляются `COMMIT` порядке `COMMIT` .

Прочитайте Индексы и ключи онлайн: https://riptutorial.com/ru/mysql/topic/1748/индексы-и-ключи

глава 27: Информация о сервере

параметры

параметры	объяснение
ГЛОБАЛЬНЫЙ	Показывает переменные, поскольку они настроены для всего сервера. Необязательный.
СЕССИЯ	Показывает переменные, настроенные только для этого сеанса. Необязательный.

Examples

Пример SHOW VARIABLES

Чтобы получить все переменные сервера, запустите этот запрос либо в окне SQL вашего предпочтительного интерфейса (PHPMyAdmin или в другом), либо в интерфейсе CLI MySQL

```
SHOW VARIABLES;
```

Вы можете указать, хотите ли вы переменные сеанса или глобальные переменные следующим образом:

Переменные сеанса:

```
SHOW SESSION VARIABLES;
```

Глобальные переменные:

```
SHOW GLOBAL VARIABLES;
```

Как и любая другая команда SQL, вы можете добавить параметры к вашему запросу, такие как команда LIKE:

```
SHOW [GLOBAL | SESSION] VARIABLES LIKE 'max_join_size';
```

Или, используя подстановочные знаки:

```
SHOW [GLOBAL | SESSION] VARIABLES LIKE '%size%';
```

Вы также можете отфильтровать результаты запроса SHOW с помощью параметра WHERE следующим образом:

```
SHOW [GLOBAL | SESSION] VARIABLES WHERE VALUE > 0;
```

Пример SHOW STATUS

Чтобы получить статус сервера базы данных, запустите этот запрос в SQL-окне вашего предпочтительного интерфейса (PHPMyAdmin или другого) или на интерфейсе CLI MySQL.

```
SHOW STATUS;
```

Вы можете указать, хотите ли вы получать статус СЕССИИ или ГЛОБАЛЬНОГО вашего ранга следующим образом: Статус сеанса:

```
SHOW SESSION STATUS;
```

Глобальный статус:

```
SHOW GLOBAL STATUS;
```

Как и любая другая команда SQL, вы можете добавить параметры к вашему запросу, такие как команда LIKE:

```
SHOW [GLOBAL | SESSION] STATUS LIKE 'Key%';
```

Или команда Where:

```
SHOW [GLOBAL | SESSION] STATUS WHERE VALUE > 0;
```

Основное различие между GLOBAL и SESSION заключается в том, что с помощью модификатора GLOBAL команда отображает агрегированную информацию о сервере и всех его соединениях, в то время как модификатор SESSION будет показывать только текущие значения.

Прочитайте Информация о сервере онлайн: https://riptutorial.com/ru/mysql/topic/9924/информация-о-сервере

глава 28: Использование переменных

Examples

Установка переменных

Вот несколько способов установки переменных:

1. Вы можете установить переменную в определенную строку, число, дату, используя SET

   EX: SET @var_string = 'my_var'; SET @var_num = '2' SET @var_date = '2015-07-20';

2. вы можете установить переменную как результат выражения select, используя: =

   EX: Выберите @var: = '123'; (Примечание: вам нужно использовать: = при назначении переменной, не использующей синтаксис SET, потому что в других операторах (select, update ...) для сравнения используется «=», поэтому, когда вы добавляете двоеточие перед « = ", вы говорите:« Это не сравнение, это SET ».)

3. Вы можете установить переменную как результат выражения select, используя INTO

   (Это было особенно полезно, когда мне нужно было динамически выбирать, из каких разделов запрашивать)

EX: SET @start_date = '2015-07-20'; SET @end_date = '2016-01-31';

```
#this gets the year month value to use as the partition names
SET @start_yearmonth = (SELECT EXTRACT(YEAR_MONTH FROM @start_date));
SET @end_yearmonth = (SELECT EXTRACT(YEAR_MONTH FROM @end_date));

#put the partitions into a variable
SELECT GROUP_CONCAT(partition_name)
FROM information_schema.partitions p
WHERE table_name = 'partitioned_table'
AND SUBSTRING_INDEX(partition_name,'P',-1) BETWEEN @start_yearmonth AND @end_yearmonth
INTO @partitions;

#put the query in a variable. You need to do this, because mysql did not recognize my variable
as a variable in that position. You need to concat the value of the variable together with the
rest of the query and then execute it as a stmt.
SET @query =
CONCAT('CREATE TABLE part_of_partitioned_table (PRIMARY KEY(id))
SELECT partitioned_table.*
FROM partitioned_table PARTITION(', @partitions,')
JOIN users u USING(user_id)
WHERE date(partitioned_table.date) BETWEEN ', @start_date,' AND ', @end_date);

#prepare the statement from @query
PREPARE stmt FROM @query;
#drop table
```

```
DROP TABLE IF EXISTS tech.part_of_partitioned_table;
#create table using statement
EXECUTE stmt;
```

Номер строки и группа. Используя переменные в Select Statement

Допустим, у нас есть таблица `team_person` как `team_person` ниже:

```
+======+===========+
| team |    person |
+======+===========+
|   A  |      John |
+------+-----------+
|   B  |     Smith |
+------+-----------+
|   A  |    Walter |
+------+-----------+
|   A  |     Louis |
+------+-----------+
|   C  | Elizabeth |
+------+-----------+
|   B  |     Wayne |
+------+-----------+


CREATE TABLE team_person AS SELECT 'A' team, 'John' person
UNION ALL  SELECT 'B' team,  'Smith' person
UNION ALL  SELECT 'A' team,  'Walter' person
UNION ALL  SELECT 'A' team,  'Louis' person
UNION ALL  SELECT 'C' team,  'Elizabeth' person
UNION ALL  SELECT 'B' team,  'Wayne' person;
```

Чтобы выбрать таблицу `team_person` с дополнительным `row_number` , либо

```
SELECT @row_no := @row_no+1 AS row_number, team, person
FROM team_person, (SELECT @row_no := 0) t;
```

ИЛИ ЖЕ

```
SET @row_no := 0;
SELECT  @row_no := @row_no + 1 AS row_number, team, person
FROM team_person;
```

выведет результат ниже:

```
+============+======+===========+
| row_number | team |    person |
+============+======+===========+
|          1 |   A  |      John |
+------------+------+-----------+
|          2 |   B  |     Smith |
+------------+------+-----------+
|          3 |   A  |    Walter |
+------------+------+-----------+
```

```
|          4 |   A  |     Louis |
+------------+------+-----------+
|          5 |   C  | Elizabeth |
+------------+------+-----------+
|          6 |   B  |     Wayne |
+------------+------+-----------+
```

И, наконец, если мы хотим получить `row_number` группы по колонку `team`

```
SELECT @row_no := IF(@prev_val = t.team, @row_no + 1, 1) AS row_number
   ,@prev_val := t.team AS team
   ,t.person
FROM team_person t,
  (SELECT @row_no := 0) x,
  (SELECT @prev_val := '') y
ORDER BY t.team ASC,t.person DESC;

+============+======+===========+
| row_number | team |    person |
+============+======+===========+
|          1 |   A  |    Walter |
+------------+------+-----------+
|          2 |   A  |     Louis |
+------------+------+-----------+
|          3 |   A  |      John |
+------------+------+-----------+
|          1 |   B  |     Wayne |
+------------+------+-----------+
|          2 |   B  |     Smith |
+------------+------+-----------+
|          1 |   C  | Elizabeth |
+------------+------+-----------+
```

Прочитайте Использование переменных онлайн: https://riptutorial.com/ru/mysql/topic/5013/использование-переменных

глава 29: Кластеризация

Examples

Disambiguation

«Значение кластера MySQL» ...

- NDB Cluster - специализированный, в основном в памяти, движок. Не широко используется.
- Galera Cluster aka Percona XtraDB Cluster aka PXC aka MariaDB с Galera. - очень хорошее решение для высокой доступности для MySQL; он выходит за рамки репликации.

См. Отдельные страницы в этих вариантах «Кластер».

Для «кластеризованного индекса» см. Страницу (ы) на `PRIMARY KEY` .

Прочитайте Кластеризация онлайн: https://riptutorial.com/ru/mysql/topic/5130/кластеризация

глава 30: Клиент MySQL

Синтаксис

- mysql [ОПЦИИ] [имя_базы_данных]

параметры

параметр	Описание
`-D --database=name`	имя базы данных
`--delimiter=str`	установите разделитель инструкций. По умолчанию используется значение ';'
`-e --execute='command'`	выполнить команду
`-h --host=name`	имя хоста для подключения к
`-p --password=name`	password *Примечание: между `-p` и паролем нет пробела*
`-p` (без пароля)	пароль будет запрашиваться для
`-P --port=#`	номер порта
`-s --silent`	тихий режим, производят меньшую мощность. Использовать `\t` качестве разделителя столбцов
`-ss`	например `-s` , но опустить имена столбцов
`-S --socket=path`	указать сокет (Unix) или именованный канал (Windows) для использования при подключении к локальному экземпляру
`--skip-column-names`	опустить имена столбцов
`-u --user=name`	имя пользователя
`-U --safe-updates --i-am-a-dummy`	войдите в систему с переменной `sql_safe_updates=ON` . Это позволит только `DELETE` и `UPDATE` явно использовать ключи
`-V --version`	распечатать версию и выйти

Examples

Базовый логин

Чтобы получить доступ к MySQL из командной строки:

```
mysql --user=username --password=pwd --host=hostname test_db
```

Это можно сократить до:

```
mysql -u username -p password -h hostname test_db
```

Отбрасывая значение `password` MySQL будет запрашивать пароль в качестве первого ввода. Если вы укажете `password` клиент предоставит вам предупреждение «небезопасно»:

```
mysql -u=username -p -h=hostname test_db
```

Для локальных соединений `--socket` может использоваться для указания файла сокета:

```
mysql --user=username --password=pwd --host=localhost --socket=/path/to/mysqld.sock test_db
```

Опущение параметра `socket` приведет к попытке клиента подключиться к серверу на локальной машине. Для подключения к серверу должен быть запущен сервер.

Выполнять команды

Этот пример показывает, как выполнять команды, хранящиеся в строках или файлах сценариев, без необходимости интерактивной подсказки. Это особенно полезно, когда скрипт оболочки должен взаимодействовать с базой данных.

Выполнить команду из строки

```
$ mysql -uroot -proot test -e'select * from people'

+----+-------+--------+
| id | name  | gender |
+----+-------+--------+
|  1 | Kathy | f      |
|  2 | John  | m      |
+----+-------+--------+
```

Чтобы форматировать вывод в виде сетки с `--silent` табуляции, используйте параметр `--silent` :

```
$ mysql -uroot -proot test -s -e'select * from people'

id      name    gender
1       Kathy   f
2       John    m
```

Чтобы опустить заголовки:

```
$ mysql -uroot -proot test -ss -e'select * from people'

1       Kathy   f
2       John    m
```

Выполнить из файла сценария:

```
$ mysql -uroot -proot test < my_script.sql
```

```
$ mysql -uroot -proot test -e'source my_script.sql'
```

Запись вывода в файл

```
$ mysql -uroot -proot test < my_script.sql > out.txt

$ mysql -uroot -proot test -s -e'select * from people' > out.txt
```

Прочитайте Клиент MySQL онлайн: https://riptutorial.com/ru/mysql/topic/5619/клиент-mysql

глава 31: Коды ошибок

Examples

Код ошибки 1064: Синтаксическая ошибка

```
select LastName, FirstName,
from Person
```

Возвращает сообщение:

> Код ошибки: 1064. У вас есть ошибка в синтаксисе SQL; проверьте руководство, соответствующее версии вашего сервера MySQL, для правильного синтаксиса для использования рядом с «от лица» по строке 2.

Получение сообщения «Ошибка 1064» из MySQL означает, что запрос не может быть проанализирован без ошибок синтаксиса. Другими словами, он не может понять смысл запроса.

Цитата в сообщении об ошибке начинается с первого символа запроса, который MySQL не может понять, как разбираться. В этом примере MySQL не может иметь смысла в контексте `from Person` . В этом случае перед `from Person` появляется дополнительная запятая. В запятой говорится, что MySQL ожидает другого описания столбца в предложении `SELECT`

Синтаксическая ошибка всегда говорит `... near '...'` . Вещь в начале цитат очень близка к ошибке. Чтобы найти ошибку, посмотрите на первый токен в кавычках и на последний токен перед кавычками.

Иногда вы получите `... near ''` ; то есть ничего в кавычках. Это означает, что первый символ, который MySQL не может понять, находится в конце или в начале инструкции. Это предполагает, что запрос содержит несбалансированные кавычки (`'` или `"`) или несбалансированные круглые скобки или что вы не закончили утверждение раньше.

В случае хранимой процедуры вы, возможно, забыли правильно использовать `DELIMITER` .

Итак, когда вы получаете Error 1064, посмотрите текст запроса и найдите точку, указанную в сообщении об ошибке. Визуально проверяйте текст запроса прямо вокруг этой точки.

Если вы попросите кого-нибудь помочь вам устранить ошибку 1064, лучше всего предоставить как текст всего запроса, так и текст сообщения об ошибке.

Код ошибки 1175: безопасное обновление

Эта ошибка появляется при попытке обновления или удаления записей без включения `WHERE` , которое использует столбец `KEY` .

Чтобы выполнить удаление или обновление в любом случае - введите:

```
SET SQL_SAFE_UPDATES = 0;
```

Чтобы снова включить безопасный режим, введите:

```
SET SQL_SAFE_UPDATES = 1;
```

Код ошибки 1215: не удается добавить ограничение внешнего ключа

Эта ошибка возникает, когда таблицы недостаточно структурированы для обработки быстрой проверки соответствия требованиям внешнего ключа (`FK`), которые требуется разработчику.

```
CREATE TABLE `gtType` (
  `type` char(2) NOT NULL,
  `description` varchar(1000) NOT NULL,
  PRIMARY KEY (`type`)
) ENGINE=InnoDB;

CREATE TABLE `getTogethers` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `type` char(2) NOT NULL,
  `eventDT` datetime NOT NULL,
  `location` varchar(1000) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_gt2type` (`type`), -- see Note1 below
  CONSTRAINT `gettogethers_ibfk_1` FOREIGN KEY (`type`) REFERENCES `gtType` (`type`)
) ENGINE=InnoDB;
```

Примечание1: такой KEY, как это будет создан автоматически, если это необходимо из-за определения FK в строке, которая следует за ним. Разработчик может пропустить его, и при необходимости добавится KEY (aka index). Пример того, что он пропускает разработчик, показан ниже в `someOther` .

До сих пор так хорошо, пока не позвонил.

```
CREATE TABLE `someOther` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `someDT` datetime NOT NULL,
  PRIMARY KEY (`id`),
  CONSTRAINT `someOther_dt` FOREIGN KEY (`someDT`) REFERENCES `getTogethers` (`eventDT`)
) ENGINE=InnoDB;
```

> Код ошибки: 1215. Невозможно добавить ограничение внешнего ключа

В этом случае он не работает из-за отсутствия индекса в *ссылочной* таблице `getTogethers`

для обработки быстрого поиска `eventDT` . Решить в следующем утверждении.

```
CREATE INDEX `gt_eventdt` ON getTogethers (`eventDT`);
```

Таблица `getTogethers` была изменена, и теперь создание `someOther` будет успешным.

На странице руководства MySQL с использованием ограничений FOREIGN KEY :

> MySQL требует индексов для внешних ключей и ссылочных ключей, чтобы проверки внешнего ключа могли быть быстрыми и не требовать сканирования таблицы. В таблице ссылок должен быть индекс, в котором столбцы внешнего ключа перечислены в качестве первых столбцов в том же порядке. Такой индекс создается в таблице ссылок автоматически, если он не существует.
>
> Соответствующие столбцы внешнего ключа и ссылочного ключа должны иметь похожие типы данных. Размер и знак целочисленных типов должны быть одинаковыми. Длина типов строк не обязательно должна быть одинаковой. Для небинных (символьных) строковых столбцов набор символов и сопоставление должны быть одинаковыми.
>
> InnoDB позволяет внешнему ключу ссылаться на любой индексный столбец или группу столбцов. Однако в ссылочной таблице должен быть индекс, в котором ссылочные столбцы указаны в качестве первых столбцов в том же порядке.

Обратите внимание, что последняя точка выше первых (самых левых) столбцов и отсутствие требования первичного ключа (хотя и очень рекомендуется).

После успешного создания таблицы *реферирования* (дочернего) все ключи, которые были автоматически созданы для вас, видны с помощью следующей команды:

```
SHOW CREATE TABLE someOther;
```

Другие распространенные случаи возникновения этой ошибки включают, как упоминалось выше, в документах, но должны быть выделены:

- По-видимому тривиальные различия в `INT` который подписан, указывая на `INT UNSIGNED` .
- Разработчики испытывают трудности с пониманием многоколоночных (составных) KEYS и первых (самых левых) требований к порядку.

1045 Доступ запрещен

См. Обсуждения в «ГРАНТ» и «Восстановление пароля root».

1236 «невозможное положение» в репликации

Обычно это означает, что Мастер разбился и этот `sync_binlog` был выключен. Решение состоит в том, чтобы `CHANGE MASTER to POS=0` следующего файла binlog (см. Мастер) в Slave.

Причина: Мастер отправляет элементы репликации в подчиненное устройство перед очисткой до его бинарного журнала (когда `sync_binlog=OFF`). Если Мастер выйдет из строя до флеша, ведомый уже логически перемещается за конец файла в binlog. Когда мастер запускается снова, он запускает новый битблог, поэтому ПЕРЕЗАПУСК к началу этого бинарного журнала является наилучшим доступным решением.

Долгосрочное решение - `sync_binlog=ON` , если вы можете позволить себе дополнительный ввод-вывод, который он вызывает.

(Если вы работаете с GTID, ...?)

2002, 2003 Не удается подключиться

Проверьте, заблокирован ли порт 3306 блокировки брандмауэра.

Некоторые возможные диагностические и / или решения

- Действительно ли сервер работает?
- «служба firewalld stop» и «systemctl disable firewalld»
- мастер telnet 3306
- Проверьте адрес `bind-address`
- проверить `skip-name-resolve`
- проверьте розетку.

1067, 1292, 1366, 1411 - Плохая стоимость для числа, даты, по умолчанию и т. Д.

1067 Это, вероятно, связано со значениями по умолчанию `TIMESTAMP` , которые со временем изменились. См. `TIMESTAMP defaults` на странице «Даты и времена». (которого еще нет)

1292/1366 DOUBLE / Integer Проверьте наличие букв или других синтаксических ошибок. Убедитесь, что столбцы выровнены; возможно, вы думаете, что ставите в `VARCHAR` но выровнены с числовым столбцом.

1292 DATETIME Проверьте слишком далеко в прошлом или будущем. Проверяйте между 2:00 и 3:00 утром, когда смена летнего времени изменилась. Проверьте наличие сильного синтаксиса, например, `+00` часовых поясов.

1292 ПЕРЕМЕННЫХ Проверьте допустимые значения для `VARIABLE` вы пытаетесь `SET` .

1292 LOAD DATA Посмотрите на строку, которая является «плохим». Проверьте escape-символы и т. Д. Посмотрите на типы данных.

1411 STR_TO_DATE Неверно отформатированная дата?

126, 127, 134, 144, 145

Когда вы пытаетесь получить доступ к записям из базы данных MySQL, вы можете получить эти сообщения об ошибках. Эти сообщения об ошибках произошли из-за повреждения в базе данных MySQL. Ниже приведены типы

```
MySQL error code 126 = Index file is crashed
MySQL error code 127 = Record-file is crashed
MySQL error code 134 = Record was already deleted (or record file crashed)
MySQL error code 144 = Table is crashed and last repair failed
MySQL error code 145 = Table was marked as crashed and should be repaired
```

Ошибка MySQL, вирусная атака, сбой сервера, неправильное завершение работы, поврежденная таблица - причина этой коррупции. Когда он становится поврежденным, он становится недоступным, и вы больше не можете обращаться к ним. Чтобы получить доступность, лучший способ получить данные из обновленной резервной копии. Однако, если у вас нет обновленной или какой-либо действительной резервной копии, вы можете перейти на восстановление MySQL.

Если тип движка таблицы - `MyISAM` , примените `CHECK TABLE` , а затем `REPAIR TABLE` .

Тогда подумайте серьезно о преобразовании в InnoDB, поэтому эта ошибка больше не повторится.

Синтаксис

```
CHECK TABLE <table name> ////To check the extent of database corruption
REPAIR TABLE <table name> ////To repair table
```

139

Ошибка 139 может означать, что число и размер полей в определении таблицы превышают некоторый предел. обходные:

- Пересмотреть схему
- Нормализовать некоторые поля
- Вертикально разбить таблицу

1366

Обычно это означает, что обработка набора символов не была согласована между клиентом и сервером. См. ... для дальнейшей помощи.

126, 1054, 1146, 1062, 24

(с перерывом). С учетом этих четырех номеров ошибок, я думаю, эта страница охватит около 50% типичных ошибок, которые получат пользователи.

(Да, этот «пример» нуждается в пересмотре.)

24 Не удается открыть файл (слишком много открытых файлов)

`open_files_limit` происходит из настройки ОС. `table_open_cache` должен быть меньше этого.

Это может привести к ошибке:

- Невозможность `DEALLOCATE PREPARE` в хранимой процедуре.
- PARTITIONed table (s) с большим количеством разделов и innodb_file_per_table = ON. Рекомендовать не иметь более 50 разделов в данной таблице (по разным причинам). (Когда «Родные разделы» станут доступными, этот совет может измениться.)

Очевидным обходным решением является увеличение ограничения ОС: разрешить больше файлов, изменить `ulimit` или `/etc/security/limits.conf` или в `sysctl.conf` (kern.maxfiles & kern.maxfilesperproc) или что-то еще (зависит от ОС). Затем увеличьте `open_files_limit` и `table_open_cache` .

Начиная с 5.6.8 `open_files_limit` автоматически `open_files_limit` на основе `max_connections` , но это нормально, чтобы изменить его по умолчанию.

1062 - Повторяющийся ввод

Эта ошибка возникает в основном из-за следующих двух причин

1. *Дублируемое значение* - `Error Code: 1062. Duplicate entry '12' for key 'PRIMARY'`

   Столбец первичного ключа уникален, и он не будет принимать дублируемую запись. Поэтому, когда вы пытаетесь вставить новую строку, которая уже присутствует в вашей таблице, это приведет к ошибке.

   Чтобы решить эту проблему, установите столбец первичного ключа как `AUTO_INCREMENT` . И когда вы пытаетесь вставить новую строку, игнорируйте столбец первичного ключа или вставьте значение `NULL` в первичный ключ.

```
CREATE TABLE userDetails(
  userId INT(10) NOT NULL AUTO_INCREMENT,
  firstName VARCHAR(50),
  lastName VARCHAR(50),
  isActive INT(1) DEFAULT 0,
  PRIMARY KEY (userId) );

--->and now while inserting
INSERT INTO userDetails VALUES (NULL ,'John', 'Doe', 1);
```

2. *Уникальное поле данных* - `Error Code: 1062. Duplicate entry 'A' for key 'code'`

   Вы можете назначить столбец как уникальный, и попытка вставить новую строку с уже существующим значением для этого столбца приведет к этой ошибке.

   Чтобы преодолеть эту ошибку, используйте `INSERT IGNORE` вместо обычного `INSERT` . Если новая строка, которую вы пытаетесь вставить, не дублирует существующую запись, MySQL вставляет ее как обычно. Если запись является дубликатом, ключевое слово `IGNORE` отбрасывает ее, не генерируя никаких ошибок.

```
INSERT IGNORE INTO userDetails VALUES (NULL ,'John', 'Doe', 1);
```

Прочитайте Коды ошибок онлайн: https://riptutorial.com/ru/mysql/topic/895/коды-ошибок

глава 32: Комментарий Mysql

замечания

-- стиль комментария, который требует трейлинг пространства, отличается в поведении от стандарта SQL , который не требует пространств.

Examples

Добавление комментариев

Существует три типа комментариев:

```
 # This comment continues to the end of line

-- This comment continues to the end of line

/* This is an in-line comment */

/*
This is a
multiple-line comment
*/
```

Пример:

```
SELECT * FROM t1; -- this is comment

CREATE TABLE stack(
    /*id_user int,
    username varchar(30),
    password varchar(30)
    */
    id int
);
```

Метод -- требует, чтобы пробел следовал за -- до начала комментария, иначе он будет интерпретироваться как команда и обычно вызывает ошибку.

```
#This comment works
/*This comment works.*/
--This comment does not.
```

Комментирование определений таблиц

```
CREATE TABLE menagerie.bird (
    bird_id INT NOT NULL AUTO_INCREMENT,
    species VARCHAR(300) DEFAULT NULL COMMENT 'You can include genus, but never subspecies.',
```

```
    INDEX idx_species (species) COMMENT 'We must search on species often.',
    PRIMARY KEY (bird_id)
) ENGINE=InnoDB COMMENT 'This table was inaugurated on February 10th.';
```

Использование параметра `=` после `COMMENT` является необязательным. (Официальные документы)

Эти комментарии, в отличие от других, сохраняются вместе с схемой и могут быть получены через `SHOW CREATE TABLE` или из `information_schema` .

Прочитайте Комментарий Mysql онлайн: https://riptutorial.com/ru/mysql/topic/2337/комментарий-mysql

глава 33: Конфигурация и настройка

замечания

Конфигурация происходит одним из трех способов:

- параметры командной строки
- файл конфигурации `my.cnf`
- установка переменных изнутри сервера

`mysqld --long-parameter-name=value --another-parameter` командной строки принимают форму `mysqld --long-parameter-name=value --another-parameter` . Те же параметры можно поместить в `my.conf` конфигурации `my.conf` . *Некоторые* параметры настраиваются с использованием системных переменных из MySQL. Ознакомьтесь с официальной документацией для полного списка параметров.

Переменные могут иметь тире `-` или подчеркивание `_` . Пространства могут существовать вокруг `=` . Большие числа могут быть суффиксами `K` , `M` , `G` для кило-, мега- и гига-. Одна настройка для каждой строки.

Флаги: Обычно `ON` и `1` являются синонимами, а для `OFF` и `0` . Некоторые флаги ничего не имеют после них.

При размещении настроек в `my.cnf` все настройки для *сервера* должны находиться в разделе `[mysqld]` , поэтому не слепо добавлять настройки в конец файла. (Примечание. Для инструментов, которые позволяют нескольким экземплярам mysql делиться одним my.cnf, имена разделов могут отличаться.)

Examples

Производительность InnoDB

Есть сотни настроек, которые можно поместить в my.cnf. Для пользователя Lite MySQL MySQL не имеет значения.

Когда ваша база данных становится нетривиальной, рекомендуется установить следующие параметры:

```
innodb_buffer_pool_size
```

Это должно быть установлено около 70% *доступной* ОЗУ (если у вас есть как минимум 4 ГБ ОЗУ, меньший процент, если у вас есть крошечная виртуальная машина или антикварная машина). Параметр управляет количеством кеша, используемого двигателем

InnoDB. Следовательно, это очень важно для производительности InnoDB.

Параметр, позволяющий вставлять огромные данные

Если вам нужно сохранить изображения или видео в столбце, нам нужно изменить значение по мере необходимости вашим приложением

max_allowed_packet = 10M

M является Mb, G в Gb, K в Kb

Увеличьте ограничение строки для group_concat

`group_concat` используется для конкатенации ненулевых значений в `group` . Максимальная длина результирующей строки может быть задана с помощью параметра `group_concat_max_len` :

```
SET [GLOBAL | SESSION] group_concat_max_len = val;
```

Установка переменной `GLOBAL` обеспечит постоянное изменение, тогда как установка переменной `SESSION` установит значение для текущего сеанса.

Минимальная конфигурация InnoDB

Это минимальная настройка для серверов MySQL с использованием таблиц InnoDB. Используя InnoDB, кеш запросов не требуется. Исправьте дисковое пространство , когда таблица или база данных `DROP` - е изд. Если вы используете SSD, то промывка является избыточной операцией (SDD не являются последовательными).

```
default_storage_engine = InnoDB
query_cache_type = 0
innodb_file_per_table = 1
innodb_flush_neighbors = 0
```

совпадение

Удостоверьтесь, что мы можем создать более чем по умолчанию 4 потока, установив `innodb_thread_concurrency` в бесконечность (0); это позволяет InnoDB решать на основе оптимального исполнения.

```
innodb_thread_concurrency = 0
innodb_read_io_threads = 64
innodb_write_io_threads = 64
```

Использование жесткого диска

Задайте емкость (нормальная нагрузка) и capacity_max (абсолютный максимум) IOPS для

MySQL. Значение по умолчанию 200 отлично подходит для жестких дисков, но в наши дни с SSD, способными к тысячам IOPS, вы, вероятно, захотите настроить этот номер. Для определения IOPS существует множество тестов. Значения выше должны быть почти такими, *если вы используете выделенный сервер MySQL* . Если вы используете какие-либо другие службы на одном компьютере, вы должны распределить их по мере необходимости.

```
innodb_io_capacity = 2500
innodb_io_capacity_max = 3000
```

Использование ОЗУ

Установите RAM для MySQL. Хотя эмпирическое правило составляет 70-80%, это действительно зависит от того, предназначен ли ваш экземпляр для MySQL, и сколько оперативной памяти доступно. Не *теряйте* RAM (т.е. ресурсы), если у вас есть много доступных.

```
innodb_buffer_pool_size = 10G
```

Безопасное шифрование MySQL

Шифрование по умолчанию `aes-128-ecb` использует `aes-128-ecb` электронной кодовой книги (ECB), который является небезопасным и никогда не должен использоваться. Вместо этого добавьте в конфигурационный файл следующее:

```
block_encryption_mode = aes-256-cbc
```

Прочитайте Конфигурация и настройка онлайн: https://riptutorial.com/ru/mysql/topic/3134/конфигурация-и-настройка

глава 34: копирование

замечания

Репликация используется для копирования данных [Backup] с одного сервера базы данных MySQL на один или несколько серверов баз данных MySQL.

Мастер - сервер базы данных MySQL, который служит для копирования данных

Slave - сервер базы данных MySQL, копирует данные, которые обслуживает Master

С MySQL репликация по умолчанию асинхронна. Это означает, что ведомые устройства не должны постоянно подключаться для получения обновлений от ведущего устройства. Например, если ваше подчиненное устройство выключено или не подключено к ведущему устройству, и вы переключите подчиненное устройство или подключитесь к мастеру позднее, то он автоматически синхронизируется с Мастером.

В зависимости от конфигурации вы можете реплицировать все базы данных, выбранные базы данных или даже выбранные таблицы в базе данных.

Форматы репликации

Существует два основных типа форматов репликации

Репликация на основе отчетов (SBR) - которая реплицирует все SQL-запросы. В этом случае мастер записывает операторы SQL в двоичный журнал. Репликация ведущего на ведомое устройство выполняется путем выполнения этих операторов SQL на ведомом.

Репликация на основе строк *(RBR)* - которая реплицирует только измененные строки. В этом случае мастер записывает события в двоичный журнал, который указывает, как изменяются отдельные строки таблицы. Репликация ведущего на ведомое устройство выполняется путем копирования событий, представляющих изменения в строках таблицы, на ведомое устройство.

Вы также можете использовать третью разновидность, *смешанную репликацию (MBR)* . В этом случае используются как ведение журнала на основе инструкций, так и строк. Журнал будет создан в зависимости от того, что наиболее подходит для изменения.

Формат, основанный на утверждениях, был по умолчанию в версиях MySQL старше 5.7.7. В MySQL 5.7.7 и более поздних версиях по умолчанию используется формат на основе строк.

Examples

Мастер-подчиненная настройка репликации

Рассмотрим 2 сервера MySQL для настройки репликации, один - Мастер, а другой - подчиненный.

Мы собираемся настроить Учителя, чтобы он хранил журнал всех действий, выполненных на нем. Мы собираемся настроить Slave-сервер, чтобы он смотрел на журнал в Master и всякий раз, когда происходят изменения в журнале на Master, он должен делать то же самое.

Конфигурация мастера

Прежде всего, нам нужно создать пользователя на Мастере. Этот пользователь будет использоваться Slave для создания соединения с Master.

```
CREATE USER 'user_name'@'%' IDENTIFIED BY 'user_password';
GRANT REPLICATION SLAVE ON *.* TO 'user_name'@'%';
FLUSH PRIVILEGES;
```

Изменение `user_name` и `user_password` в соответствии с вашим именем пользователя и паролем.

Теперь `my.inf` (my.cnf в Linux) должен быть отредактирован. Включите следующие строки в раздел [mysqld].

```
server-id = 1
log-bin = mysql-bin.log
binlog-do-db = your_database
```

Первая строка используется для назначения идентификатора этому серверу MySQL.

Во второй строке говорится, что MySQL начинает записывать журнал в указанный файл журнала. В Linux это можно настроить как `log-bin = /home/mysql/logs/mysql-bin.log` . Если вы запускаете репликацию на сервере MySQL, в котором репликация уже используется, убедитесь, что этот каталог пуст во всех журналах репликации.

Третья строка используется для настройки базы данных, для которой мы собираемся писать журнал. Вы должны заменить `your_database` своим именем базы данных.

Убедитесь, что `skip-networking` не включен и перезапуск сервера MySQL (мастер)

Конфигурация ведомого

Файл `my.inf` должен быть отредактирован в Slave. Включите следующие строки в раздел [mysqld].

```
server-id = 2
master-host = master_ip_address
master-connect-retry = 60
```

```
master-user = user_name
master-password = user_password
replicate-do-db = your_database

relay-log = slave-relay.log
relay-log-index = slave-relay-log.index
```

Первая строка используется для назначения идентификатора этому серверу MySQL. Этот идентификатор должен быть уникальным.

Вторая строка - это IP-адрес главного сервера. Измените это в соответствии с IP-адресом вашей основной системы

Третья строка используется для установки предела повтора в секундах.

В следующих двух строках указывается имя пользователя и пароль для ведомого устройства, с помощью которого он соединяет Master.

Следующая строка задает базу данных, которую необходимо реплицировать.

Последние две строки используются для назначения имен файлов `relay-log` и `relay-log-index` .

Убедитесь, что `skip-networking` не включен и перезапустите сервер MySQL (Slave)

Скопировать данные в подчиненный

Если данные постоянно добавляются к Мастеру, мы должны будем предотвратить доступ к базе данных на Мастере, чтобы ничего не было добавлено. Этого можно добиться, запустив следующую инструкцию в Master.

```
FLUSH TABLES WITH READ LOCK;
```

Если данные не добавляются на сервер, вы можете пропустить вышеуказанный шаг.

Мы собираемся взять резервную копию данных Master с помощью `mysqldump`

```
mysqldump your_database -u root -p > D://Backup/backup.sql;
```

Измените свою `your_database` и каталог резервного копирования в соответствии с настройками. Теперь у вас будет файл под названием `backup.sql` в данном месте.

Если ваша база данных не существует в вашем подчиненном устройстве, создайте ее, выполнив следующие

```
CREATE DATABASE `your_database`;
```

Теперь нам нужно импортировать резервную копию на сервер Slave MySQL.

```
mysql -u root -p your_database  <D://Backup/backup.sql
--->Change `your_database` and backup directory according to your setup
```

Запустить репликацию

Чтобы начать репликацию, нам нужно найти имя файла журнала и место регистрации в Мастере. Итак, запустите в Master

```
SHOW MASTER STATUS;
```

Это даст вам результат, как показано ниже

```
+---------------------+----------+--------------------------------+------------------+
| File                | Position | Binlog_Do_DB                   | Binlog_Ignore_DB |
+---------------------+----------+--------------------------------+------------------+
| mysql-bin.000001    | 130      | your_database                  |                  |
+---------------------+----------+--------------------------------+------------------+
```

Затем запустите следующее в Slave

```
SLAVE STOP;
CHANGE MASTER TO MASTER_HOST='master_ip_address', MASTER_USER='user_name',
   MASTER_PASSWORD='user_password', MASTER_LOG_FILE='mysql-bin.000001', MASTER_LOG_POS=130;
SLAVE START;
```

Сначала мы останавливаем Раба. Затем мы точно скажем, где искать файл главного журнала. Для имени `MASTER_LOG_FILE` и `MASTER_LOG_POS` используйте значения, которые мы получили, выполнив команду `SHOW MASTER STATUS` на Master.

Вы должны изменить IP-адрес мастера в `MASTER_HOST` и соответствующим образом изменить его и пароль.

Раб теперь будет ждать. Статус ведомого устройства можно просмотреть, запустив следующий

```
SHOW SLAVE STATUS;
```

Если вы ранее выполняли `FLUSH TABLES WITH READ LOCK` в Master, освободите таблицы от блокировки, запустив следующие

```
UNLOCK TABLES;
```

Теперь Мастер хранит журнал для каждого действия, выполняемого на нем, а подчиненный сервер просматривает журнал мастера. Всякий раз, когда происходят изменения в журнале Master, Slave реплицирует это.

Ошибки репликации

Всякий раз, когда во время выполнения запроса на ведомом появляется ошибка, MySQL автоматически останавливает репликацию, чтобы идентифицировать проблему и исправить ее. Это главным образом потому, что событие вызвало дублирование ключа или строки не было найдено, и оно не может быть обновлено или удалено. Вы можете пропустить такие ошибки, даже если это не рекомендуется

Чтобы пропустить только один запрос, который висит на подчиненном устройстве, используйте следующий синтаксис

```
SET GLOBAL sql_slave_skip_counter = N;
```

Этот оператор пропускает следующие N событий от мастера. Этот оператор действителен только тогда, когда подчиненные потоки не работают. В противном случае возникает ошибка.

```
STOP SLAVE;
SET GLOBAL sql_slave_skip_counter=1;
START SLAVE;
```

В некоторых случаях это нормально. Но если оператор является частью транзакции с несколькими операторами, он становится более сложным, потому что пропустить оператор создания ошибки приведет к пропущению всей транзакции.

Если вы хотите пропустить больше запросов, которые производят одинаковый код ошибки, и если вы уверены, что пропустить эти ошибки не приведет к непоследовательности вашего ведомого и вы хотите пропустить их все, вы бы добавили строку, чтобы пропустить этот код ошибки в `my.cnf` ,

Например, вы можете пропустить все повторяющиеся ошибки, которые могут возникнуть

```
1062 | Error 'Duplicate entry 'xyz' for key 1' on query
```

Затем добавьте следующее в свой `my.cnf`

```
slave-skip-errors = 1062
```

Вы можете пропустить также другие типы ошибок или все коды ошибок, но убедитесь, что пропуская эти ошибки не приведет к непоследовательности вашего подчиненного. Ниже приведены синтаксис и примеры

```
slave-skip-errors=[err_code1,err_code2,...|all]

slave-skip-errors=1062,1053
slave-skip-errors=all
slave-skip-errors=ddl_exist_errors
```

Прочитайте копирование онлайн: https://riptutorial.com/ru/mysql/topic/7218/копирование

глава 35: Лог-файлы

Examples

Список

- Общий журнал - все запросы - см. VARIABLE general_log
- Медленный журнал - запросы медленнее, чем long_query_time - slow_query_log_file
- Binlog - для репликации и резервного копирования - log_bin_basename
- Журнал ретрансляции - также для репликации
- общие ошибки - mysqld.err
- start / stop - mysql.log (не очень интересно) - log_error
- Журнал изменений InnoDB - iblog *

См. Переменные `basedir` и `datadir` для местоположения по умолчанию для многих журналов

Некоторые журналы включаются и выключаются другими ПЕРЕМЕННЫМИ. Некоторые из них либо записываются в файл, либо в таблицу.

(Примечание для рецензентов: для этого требуется более подробная информация и пояснения.)

Документаторы : укажите расположение и имя по умолчанию для каждого типа журнала, как для Windows, так и для * nix. (Или, по крайней мере, столько, сколько сможете).

Медленный журнал запросов

Журнал медленных запросов состоит из журнальных событий для запросов, `long_query_time` до `long_query_time` . Например, до 10 секунд для завершения. Чтобы увидеть установленный порог времени, выполните следующие действия:

```
SELECT @@long_query_time;
+-------------------+
| @@long_query_time |
+-------------------+
|         10.000000 |
+-------------------+
```

Его можно установить как переменную GLOBAL в файле `my.cnf` или `my.ini` . Или это может быть установлено связью, хотя это необычно. Значение может быть установлено в диапазоне от 0 до 10 (в секундах). Какую ценность использовать?

- 10 настолько высока, что почти бесполезна;
- 2 - компромисс;
- 0,5 и другие фракции;

- 0 захватывает все; это может залить диск опасно быстро, но может быть очень полезно.

Захват медленных запросов включается или выключается. И указанный файл также указан. Ниже приведены следующие понятия:

```
SELECT @@slow_query_log; -- Is capture currently active? (1=On, 0=Off)
SELECT @@slow_query_log_file; -- filename for capture. Resides in datadir
SELECT @@datadir; -- to see current value of the location for capture file

SET GLOBAL slow_query_log=0; -- Turn Off
-- make a backup of the Slow Query Log capture file. Then delete it.
SET GLOBAL slow_query_log=1; -- Turn it back On (new empty file is created)
```

Для получения дополнительной информации см. Страницу руководства MySQL. Журнал медленных запросов

Примечание: приведенная выше информация о включении / выключении замедления была изменена в 5.6 (?); у старой версии был другой механизм.

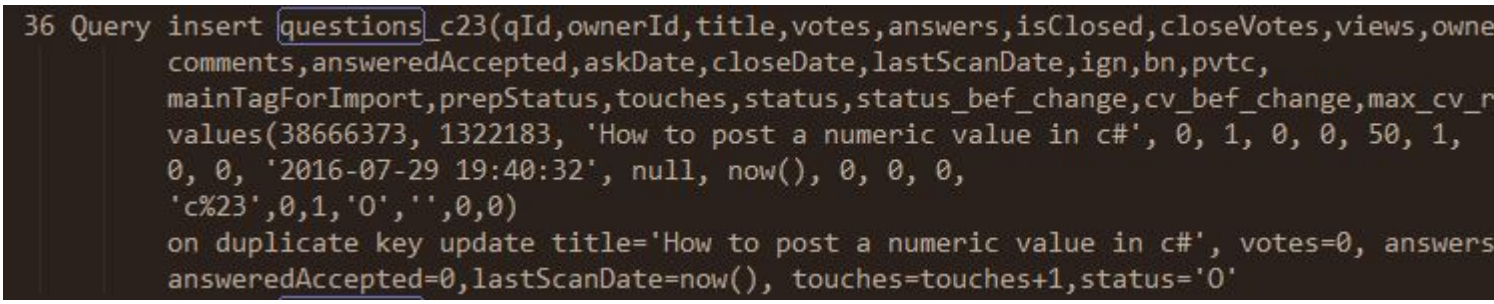
«Лучший» способ увидеть, что замедляет вашу систему:

```
long_query_time=...
turn on the slowlog
run for a few hours
turn off the slowlog (or raise the cutoff)
run pt-query-digest to find the 'worst' couple of queries.  Or mysqldumpslow -s t
```

Общий журнал запросов

Общий журнал запросов содержит листинг общей информации от клиентских подключений, разъединений и запросов. Это бесценно для отладки, но это создает препятствия для работы (цитата?).

Пример просмотра общего журнала запросов приведен ниже:



Чтобы определить, записывается ли в настоящий момент общий журнал:

```
SELECT @@general_log; -- 1 = Capture is active; 0 = It is not.
```

Чтобы определить имя файла захвата:

```
SELECT @@general_log_file; -- Full path to capture file
```

Если полный путь к файлу не отображается, файл существует в `datadir` .

Пример Windows:

```
+----------------------------------------------------------+
| @@general_log_file                                       |
+----------------------------------------------------------+
| C:\ProgramData\MySQL\MySQL Server 5.7\Data\GuySmiley.log |
+----------------------------------------------------------+
```

Linux:

```
+-----------------------------------+
| @@general_log_file                |
+-----------------------------------+
| /var/lib/mysql/ip-ww-xx-yy-zz.log |
+-----------------------------------+
```

Когда изменяются переменные GLOBAL `general_log_file` , новый журнал сохраняется в `datadir` . Однако полный путь больше не может быть отражен путем изучения переменной.

В случае отсутствия записи для файла `general_log_file` в файле конфигурации по умолчанию будет `@@hostname` .log в `datadir` .

Лучшая практика - отключить захват. Сохраните файл журнала в резервном каталоге с именем файла, отражающим дату и время начала и окончания захвата. Удаление предыдущего файла, если *перемещение* файловой системы не произошло из этого файла. Установите новое имя файла для файла журнала и включите захват (все показано ниже). Лучшие практики также включают тщательное определение, даже если вы хотите захватить на данный момент. Как правило, захват включен только для целей отладки.

Типичным именем файла файловой системы для резервного архива может быть:

```
/LogBackup/GeneralLog_20160802_1520_to_20160802_1815.log
```

где дата и время являются частью имени файла как диапазона.

Для Windows обратите внимание на следующую последовательность изменений настроек.

```
SELECT @@general_log; -- 0. Not being captured
SELECT @@general_log_file; -- C:\ProgramData\MySQL\MySQL Server 5.6\Data\GuySmiley.log
SELECT @@datadir; -- C:\ProgramData\MySQL\MySQL Server 5.7\Data\
SET GLOBAL general_log_file='GeneralLogBegin_20160803_1420.log'; -- datetime clue
SET GLOBAL general_log=1; -- Turns on actual log capture. File is created under `datadir`
SET GLOBAL general_log=0; -- Turn logging off
```

Linux похож. Они будут представлять собой динамические изменения. Любой перезапуск

сервера подбирает настройки файла конфигурации.

Что касается файла конфигурации, рассмотрите следующие соответствующие параметры переменной:

```
[mysqld]
general_log_file = /path/to/currentquery.log
general_log      = 1
```

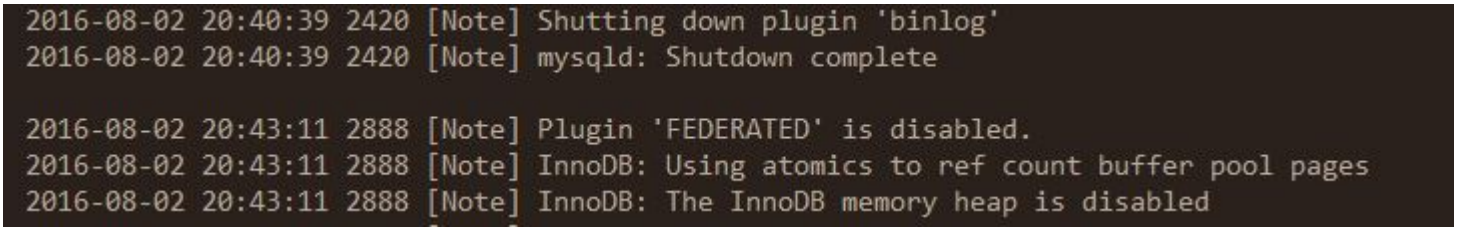
Кроме того, переменную `log_output` можно настроить для вывода `TABLE` , а не только `FILE` . Для этого, пожалуйста, см. « Направления» .

См. Страницу руководства MySQL . Общий журнал запросов .

Журнал ошибок

Журнал ошибок заполняется информацией о запуске и остановке и критическими событиями, встречающимися на сервере.

Ниже приведен пример его содержимого:



Переменная `log_error` содержит путь к файлу журнала для регистрации ошибок.

В отсутствие записи файла конфигурации для `log_error` система будет по умолчанию использовать значения `@@hostname` .err в `datadir` . Обратите внимание, что `log_error` не является динамической переменной. Таким образом, изменения выполняются с помощью изменений файла cnf или ini и перезапуска сервера (или путем просмотра «Промывка и переименование файла журнала ошибок» на странице «Ручная страница» внизу).

Журналы не могут быть отключены для ошибок. Они важны для здоровья системы при устранении неполадок. Кроме того, записи нечасты по сравнению с общим журналом запросов.

GLOBAL-переменная `log_warnings` устанавливает уровень для многословия, который зависит от версии сервера. Следующий фрагмент иллюстрирует:

```
SELECT @@log_warnings; -- make a note of your prior setting
SET GLOBAL log_warnings=2; -- setting above 1 increases output (see server version)
```

`log_warnings` как показано выше, является динамической переменной.

Изменения файла конфигурации в файлах `cnf` и `ini` могут выглядеть следующим образом.

```
[mysqld]
log_error        = /path/to/CurrentError.log
log_warnings     = 2
```

MySQL 5.7.2 расширил многословие уровня предупреждения до 3 и добавил GLOBAL `log_error_verbosity` . Опять же, оно было введено в 5.7.2. Он может быть установлен динамически и проверен как переменная или задан с помощью настроек файла конфигурации `cnf` или `ini` .

Начиная с MySQL 5.7.2:

```
[mysqld]
log_error           = /path/to/CurrentError.log
log_warnings        = 2
log_error_verbosity = 3
```

Пожалуйста, обратитесь к странице руководства по MySQL, озаглавленной « Журнал ошибок», особенно для флеширования и переименования файла журнала ошибок, а *также в* разделе « *Подробный журнал ошибок* » с версиями, связанными с `log_warnings` и `error_log_verbosity` .

Прочитайте Лог-файлы онлайн: https://riptutorial.com/ru/mysql/topic/5102/лог-файлы

глава 36: Наборы символов и сортировки

Examples

декларация

```
CREATE TABLE foo ( ...
    name CHARACTER SET utf8mb4
    ... );
```

соединение

Жизненно важным для использования наборов символов является указание MySQL-серверу, что кодирует байты клиента. Вот один из способов:

```
SET NAMES utf8mb4;
```

Каждый язык (PHP, Python, Java, ...) имеет свой собственный способ, который обычно предпочтительнее `SET NAMES` .

Например: `SET NAMES utf8mb4` вместе с объявленным столбцом `CHARACTER SET latin1` - это будет конвертировать из latin1 в utf8mb4, когда `INSERTing` и конвертировать назад при `SELECTing` .

Какой ХАРАКТЕР УСТАНАВЛИВАЕТСЯ И КОЛЛАМЕНТ?

Есть десятки наборов символов с сотнями сортировок. (Данная сортировка относится только к одному набору символов.) См. Вывод `SHOW COLLATION;` ,

Обычно есть только 4 `CHARACTER SETs` :

```
ascii -- basic 7-bit codes.
latin1 -- ascii, plus most characters needed for Western European languages.
utf8 -- the 1-, 2-, and 3-byte subset of utf8.  This excludes Emoji and some of Chinese.
utf8mb4 -- the full set of UTF8 characters, covering all current languages.
```

Все включают английские символы, которые кодируются одинаково. utf8 - подмножество utf8mb4.

Лучшая практика ...

- Используйте utf8mb4 для любого столбца `TEXT` или `VARCHAR` котором может быть множество языков.
- Используйте ascii (latin1 в порядке) для шестнадцатеричных строк (UUID, MD5 и т. Д.) И простых кодов (country_code, postal_code и т. Д.).

utf8mb4 не существовало до версии 5.5.3, поэтому utf8 был лучшим из доступных до этого.

Вне MySQL «UTF8» означает те же вещи, что и MySQL utf8mb4, а не utf8 MySQL.

Коллажи начинаются с имени кодировки и обычно заканчиваются на `_ci` для « `_ci` и акцентного нечувствительного» или `_bin` для «просто сравнивать биты.

«Последняя» версия utf8mb4 - `utf8mb4_unicode_520_ci` , основанная на Unicode 5.20. Если вы работаете с одним языком, вам может понадобиться, скажем, `utf8mb4_polish_ci` , который немного изменит буквы, основываясь на польских соглашениях.

Установка наборов символов для таблиц и полей

Вы можете установить набор символов как для каждой таблицы, так и для отдельного поля с помощью операторов `CHARACTER SET` и `CHARSET` :

```
CREATE TABLE Address (
    `AddressID`    INTEGER NOT NULL PRIMARY KEY,
    `Street`       VARCHAR(80) CHARACTER SET ASCII,
    `City`         VARCHAR(80),
    `Country`      VARCHAR(80) DEFAULT "United States",
    `Active`       BOOLEAN DEFAULT 1,
) Engine=InnoDB default charset=UTF8;
```

`City` и `Country` будут использовать `UTF8` , поскольку мы устанавливаем это как набор символов по умолчанию для таблицы. `Street` с другой стороны, будет использовать `ASCII` , поскольку мы специально сказали ей сделать это.

Установка правильного набора символов сильно зависит от вашего набора данных, но также может значительно улучшить переносимость между системами, работающими с вашими данными.

Прочитайте Наборы символов и сортировки онлайн: https://riptutorial.com/ru/mysql/topic/4569/наборы-символов-и-сортировки

глава 37: Настройка PS1

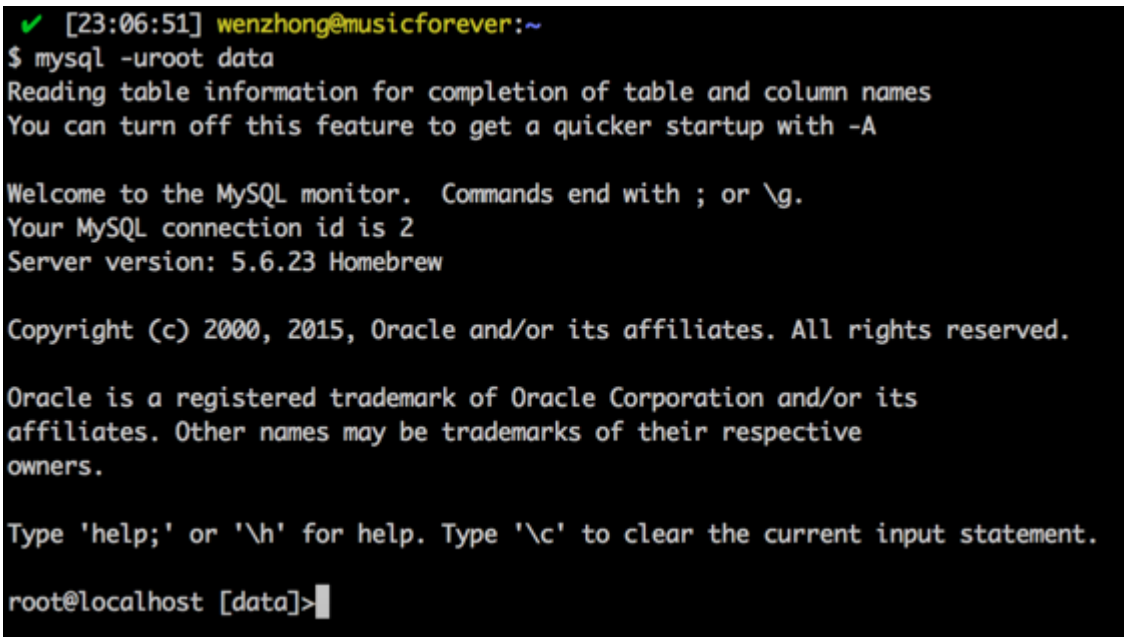
Examples

Настроить MySQL PS1 с текущей базой данных

В .bashrc или .bash_profile добавив:

```
export MYSQL_PS1="\u@\h [\d]>"
```

сделать клиента MySQL PROMPT текущим пользователем @ host [database].



Пользовательский PS1 через конфигурационный файл MySQL

В `mysqld.cnf` или в эквиваленте:

```
[mysql]
prompt = '\u@\h [\d]> '
```

Это приводит к аналогичному эффекту, без необходимости иметь дело с `.bashrc` .

Прочитайте Настройка PS1 онлайн: https://riptutorial.com/ru/mysql/topic/5795/настройка-ps1

глава 38: Настройка подключения SSL

Examples

Настройка для систем на базе Debian

(Предполагается, что MySQL установлен и используется `sudo` .)

Создание ключей CA и SSL

Убедитесь, что установлены OpenSSL и библиотеки:

```
apt-get -y install openssl
apt-get -y install libssl-dev
```

Затем введите и введите каталог для файлов SSL:

```
mkdir /home/ubuntu/mysqlcerts
cd /home/ubuntu/mysqlcerts
```

Чтобы сгенерировать ключи, создайте центр сертификации (CA) для подписания ключей (самозаверяющих):

```
openssl genrsa 2048 > ca-key.pem
openssl req -new -x509 -nodes -days 3600 -key ca-key.pem -out ca.pem
```

Значения, введенные в каждое приглашение, не будут влиять на конфигурацию. Затем создайте ключ для сервера и подпишите с помощью CA:

```
openssl req -newkey rsa:2048 -days 3600 -nodes -keyout server-key.pem -out server-req.pem
openssl rsa -in server-key.pem -out server-key.pem

openssl x509 -req -in server-req.pem -days 3600 -CA ca.pem -CAkey ca-key.pem -set_serial 01 -
out server-cert.pem
```

Затем создайте ключ для клиента:

```
openssl req -newkey rsa:2048 -days 3600 -nodes -keyout client-key.pem -out client-req.pem
openssl rsa -in client-key.pem -out client-key.pem
openssl x509 -req -in client-req.pem -days 3600 -CA ca.pem -CAkey ca-key.pem -set_serial 01 -
out client-cert.pem
```

Чтобы убедиться, что все настроено правильно, проверьте ключи:

```
openssl verify -CAfile ca.pem server-cert.pem client-cert.pem
```

Добавление ключей в MySQL

Откройте файл конфигурации MySQL . Например:

```
vim /etc/mysql/mysql.conf.d/mysqld.cnf
```

В разделе `[mysqld]` добавьте следующие параметры:

```
ssl-ca = /home/ubuntu/mysqlcerts/ca.pem
ssl-cert = /home/ubuntu/mysqlcerts/server-cert.pem
ssl-key = /home/ubuntu/mysqlcerts/server-key.pem
```

Перезапустите MySQL. Например:

```
service mysql restart
```

Тестирование SSL-соединения

Подключитесь таким же образом, передав дополнительные опции `ssl-ca` , `ssl-cert` и `ssl-key` , используя сгенерированный ключ клиента. Например, если предположить, что `cd /home/ubuntu/mysqlcerts` :

```
mysql --ssl-ca=ca.pem --ssl-cert=client-cert.pem --ssl-key=client-key.pem -h 127.0.0.1 -u
superman -p
```

После входа в систему убедитесь, что соединение действительно безопасно:

```
superman@127.0.0.1 [None]> SHOW VARIABLES LIKE '%ssl%';
+---------------+-----------------------------------------+
| Variable_name | Value                                   |
+---------------+-----------------------------------------+
| have_openssl  | YES                                     |
| have_ssl      | YES                                     |
| ssl_ca        | /home/ubuntu/mysqlcerts/ca.pem          |
| ssl_capath    |                                         |
| ssl_cert      | /home/ubuntu/mysqlcerts/server-cert.pem |
| ssl_cipher    |                                         |
| ssl_crl       |                                         |
| ssl_crlpath   |                                         |
| ssl_key       | /home/ubuntu/mysqlcerts/server-key.pem  |
+---------------+-----------------------------------------+
```

Вы также можете проверить:

```
superman@127.0.0.1 [None]> STATUS;
...
SSL:                    Cipher in use is DHE-RSA-AES256-SHA
```

```
...
```

Применение SSL

Это через `GRANT` , используя `REQUIRE SSL` :

```
GRANT ALL PRIVILEGES ON *.* TO 'superman'@'127.0.0.1' IDENTIFIED BY 'pass' REQUIRE SSL;
FLUSH PRIVILEGES;
```

Теперь `superman` *должен* подключиться через SSL.

Если вы не хотите управлять клиентскими ключами, используйте ключ клиента раньше и автоматически используйте это для всех клиентов. Откройте файл конфигурации MySQL , например:

```
vim /etc/mysql/mysql.conf.d/mysqld.cnf
```

В разделе `[client]` добавьте следующие параметры:

```
ssl-ca = /home/ubuntu/mysqlcerts/ca.pem
ssl-cert = /home/ubuntu/mysqlcerts/client-cert.pem
ssl-key = /home/ubuntu/mysqlcerts/client-key.pem
```

Теперь `superman` нужно только ввести следующее для входа через SSL:

```
mysql -h 127.0.0.1 -u superman -p
```

Подключение из другой программы, например, в Python, обычно требует дополнительного параметра для функции соединения. Пример Python:

```
import MySQLdb
ssl = {'cert': '/home/ubuntu/mysqlcerts/client-cert.pem', 'key':
'/home/ubuntu/mysqlcerts/client-key.pem'}
conn = MySQLdb.connect(host='127.0.0.1', user='superman', passwd='imsoawesome', ssl=ssl)
```

Ссылки и дальнейшее чтение:

- https://www.percona.com/blog/2013/06/22/setting-up-mysql-ssl-and-secure-connections/
- https://lowendbox.com/blog/getting-started-with-mysql-over-ssl/
- http://xmodulo.com/enable-ssl-mysql-server-client.html
- https://ubuntuforums.org/showthread.php?t=1121458

Настройка для CentOS7 / RHEL7

В этом примере предполагается два сервера:

1. dbserver (где живет наша база данных)
2. appclient (где живут наши приложения)

FWIW, оба сервера - это принудительное выполнение SELinux.

Сначала войдите в dbserver

Создайте временный каталог для создания сертификатов.

```
mkdir /root/certs/mysql/ && cd /root/certs/mysql/
```

Создание сертификатов сервера

```
openssl genrsa 2048 > ca-key.pem
openssl req -sha1 -new -x509 -nodes -days 3650 -key ca-key.pem > ca-cert.pem
openssl req -sha1 -newkey rsa:2048 -days 730 -nodes -keyout server-key.pem > server-req.pem
openssl rsa -in server-key.pem -out server-key.pem
openssl x509 -sha1 -req -in server-req.pem -days 730  -CA ca-cert.pem -CAkey ca-key.pem -
set_serial 01 > server-cert.pem
```

Переместить сертификаты сервера в / etc / pki / tls / certs / mysql /

Путь к каталогу предполагает CentOS или RHEL (при необходимости настраивается для других дистрибутивов):

```
mkdir /etc/pki/tls/certs/mysql/
```

Обязательно установите разрешения для папки и файлов. mysql требует полного владения и доступа.

```
chown -R mysql:mysql /etc/pki/tls/certs/mysql
```

Теперь настройте MySQL / MariaDB

```
# vi /etc/my.cnf
# i
[mysqld]
bind-address=*
ssl-ca=/etc/pki/tls/certs/ca-cert.pem
ssl-cert=/etc/pki/tls/certs/server-cert.pem
ssl-key=/etc/pki/tls/certs/server-key.pem
# :wq
```

затем

```
systemctl restart mariadb
```

Не забудьте открыть брандмауэр, чтобы разрешать подключения с помощью appclient (используя IP 1.2.3.4)

```
firewall-cmd --zone=drop --permanent --add-rich-rule 'rule family="ipv4" source
address="1.2.3.4" service name="mysql" accept'
# I force everything to the drop zone.  Season the above command to taste.
```

Теперь перезапустите firewalld

```
service firewalld restart
```

Затем войдите в сервер mysql dbserver:

```
mysql -uroot -p
```

Выполните следующие действия, чтобы создать пользователя для клиента. обратите внимание на REQUIRE SSL в заявлении GRANT.

```
GRANT ALL PRIVILEGES ON *.* TO 'iamsecure'@'appclient' IDENTIFIED BY 'dingdingding' REQUIRE
SSL;
FLUSH PRIVILEGES;
# quit mysql
```

Вы должны по-прежнему находиться в / root / certs / mysql с первого шага. Если нет, cd вернется к нему для одной из приведенных ниже команд.

Создание клиентских сертификатов

```
openssl req -sha1 -newkey rsa:2048 -days 730 -nodes -keyout client-key.pem > client-req.pem
openssl rsa -in client-key.pem -out client-key.pem
openssl x509 -sha1 -req -in client-req.pem -days 730 -CA ca-cert.pem -CAkey ca-key.pem -
set_serial 01 > client-cert.pem
```

Примечание . Я использовал одно и то же общее имя для сертификатов сервера и клиента. YMMV.

Убедитесь, что вы все еще / root / certs / mysql / для этой следующей команды

Объединение сертификата сервера и клиента CA в один файл:

```
cat server-cert.pem client-cert.pem > ca.pem
```

Убедитесь, что вы видите два сертификата:

```
cat ca.pem
```

КОНЕЦ СЕВЕРНОЙ РАБОТЫ СЕЙЧАС.

Откройте другой терминал и

```
ssh appclient
```

Как и прежде, создайте постоянный дом для клиентских сертификатов

```
mkdir /etc/pki/tls/certs/mysql/
```

Теперь поместите клиентские сертификаты (созданные на dbserver) на appclient. Вы можете либо скопировать их, либо просто скопировать и вставить файлы по одному.

```
scp dbserver
# copy files from dbserver to appclient
# exit scp
```

Опять же, обязательно установите разрешения для папки и файлов. mysql требует полного владения и доступа.

```
chown -R mysql:mysql /etc/pki/tls/certs/mysql
```

У вас должно быть три файла, каждый из которых принадлежит пользователю mysql:

```
/etc/pki/tls/certs/mysql/ca.pem
/etc/pki/tls/certs/mysql/client-cert.pem
/etc/pki/tls/certs/mysql/client-key.pem
```

Теперь отредактируйте конфигурацию MariaDB / MySQL appclient в разделе `[client]` .

```
vi /etc/my.cnf
# i
[client]
ssl-ca=/etc/pki/tls/certs/mysql/ca.pem
ssl-cert=/etc/pki/tls/certs/mysql/client-cert.pem
ssl-key=/etc/pki/tls/certs/mysql/client-key.pem
# :wq
```

Перезапустить службу mariadb для приложения appclient:

```
systemctl restart mariadb
```

все еще на клиенте здесь

Это должно возвратиться: ssl TRUE

```
mysql --ssl --help
```

Теперь войдите в экземпляр mysql appclient

```
mysql -uroot -p
```

Должно видеть ДА для обеих переменных ниже

```
show variables LIKE '%ssl';
    have_openssl    YES
    have_ssl              YES
```

Первоначально я видел

```
 have_openssl NO
```

Быстрый просмотр mariadb.log показал:

> Ошибка SSL: невозможно получить сертификат из '/etc/pki/tls/certs/mysql/client-cert.pem'

Проблема заключалась в том, что root-client-cert.pem и содержащая папка принадлежали root. Решение заключалось в том, чтобы установить принадлежность / etc / pki / tls / certs / mysql / к mysql.

```
chown -R mysql:mysql /etc/pki/tls/certs/mysql
```

Перезапустите mariadb, если необходимо, с шага, следующего выше

ТЕПЕРЬ МЫ ГОТОВЫ ИСПЫТАТЬ БЕЗОПАСНОЕ СОЕДИНЕНИЕ

Мы все еще здесь.

Попытайтесь подключиться к экземпляру mysql dbserver, используя созданную выше учетную запись.

```
mysql -h dbserver -u iamsecure -p
# enter password dingdingding (hopefully you changed that to something else)
```

С небольшой удачей вы должны войти в систему без ошибок.

Чтобы подтвердить, что вы подключены с поддержкой SSL, введите следующую команду

из приглашения MariaDB / MySQL:

```
\s
```

Это обратная косая черта, ака статус

Это покажет статус вашего соединения, которое должно выглядеть примерно так:

```
Connection id:         4
Current database:
Current user:         iamsecure@appclient
SSL:             Cipher in use is DHE-RSA-AES256-GCM-SHA384
Current pager:         stdout
Using outfile:         ''
Using delimiter:    ;
Server:             MariaDB
Server version:         5.X.X-MariaDB MariaDB Server
Protocol version:    10
Connection:         dbserver via TCP/IP
Server characterset:    latin1
Db     characterset:    latin1
Client characterset:    utf8
Conn.  characterset:    utf8
TCP port:         3306
Uptime:             42 min 13 sec
```

Если вы получили разрешение на отклонения ошибок при попытке подключения, проверьте приведенный выше оператор GRANT, чтобы убедиться, что нет никаких бродячих символов или «отметок».

Если у вас есть ошибки SSL, вернитесь к этому руководству, чтобы убедиться, что шаги упорядочены.

Это работало на RHEL7 и, вероятно, будет работать и на CentOS7. Не удается подтвердить, будут ли эти точные шаги работать в другом месте.

Надеюсь, это спасет кого-то еще немного времени и обострения.

Прочитайте Настройка подключения SSL онлайн: https://riptutorial.com/ru/mysql/topic/7563/настройка-подключения-ssl

глава 39: Настройка производительности

Синтаксис

- Не используйте DISTINCT и GROUP BY в том же SELECT.
- Не разбивайте страницы с помощью OFFSET, «помните, где вы остановились».
- ГДЕ (a, b) = (22,33) не оптимизируется вообще.
- Явно говорю ALL или DISTINCT после UNION - это напоминает, что вы выбираете между быстрым ALL или медленным DISTINCT.
- Не используйте SELECT *, особенно если у вас есть столбцы TEXT или BLOB, которые вам не нужны. В tmp-таблицах и передаче есть накладные расходы.
- Это быстрее, когда GROUP BY и ORDER BY могут иметь точно такой же список.
- Не используйте FORCE INDEX; это может помочь сегодня, но, вероятно, завтра будет больно.

замечания

См. Также дискуссии об ORDER BY, LIKE, REGEXP и т. Д. Примечание: для этого требуется редактирование со ссылками и другими темами.

Поваренная книга по созданию оптимальных индексов .

Examples

Добавьте правильный индекс

Это огромная тема, но это также самая важная проблема «производительности».

Основным уроком для новичков является изучение «составных» индексов. Вот краткий пример:

```
INDEX(last_name, first_name)
```

отлично подходит для них:

```
WHERE last_name = '...'
WHERE first_name = '...' AND last_name = '...'   -- (order in WHERE does not matter)
```

но не для

```
WHERE first_name = '...'   -- order in INDEX _does_ matter
WHERE last_name = '...' OR first_name = '...'   -- "OR" is a killer
```

Правильно установите кеш

`innodb_buffer_pool_size` должно быть около 70% доступной ОЗУ.

Избегайте неэффективных конструкций

```
x IN ( SELECT ... )
```

превратиться в `JOIN`

Когда это возможно, избегайте `OR` .

Не скрывайте индексированный столбец в функции, такой как `WHERE DATE(x) = ...` ; переформулировать как `WHERE x = ...`

Вы можете вообще избегать `WHERE LCASE(name1) = LCASE(name2)` , имея подходящую сортировку.

Не используйте `OFFSET` для «разбивки на страницы», вместо этого «помните, где вы остановились».

Избегайте `SELECT * ...` (кроме отладки).

Примечание для Марии Делевой, Барранка, Батсу: Это владелец места; пожалуйста, удалите эти элементы по мере создания полномасштабных примеров. После того, как вы сделаете то, что сможете, я перейду к описанию остальных и / или брошу их.

Отрицательных

Вот некоторые вещи, которые вряд ли помогут производительности. Они основаны на устаревшей информации и / или наивности.

- InnoDB улучшилась до такой степени, что MyISAM вряд ли будет лучше.
- `PARTITIONing` редко обеспечивает преимущества в производительности; это может даже повредить работе.
- Установка `query_cache_size` больше , чем 100M, как правило , *больно* производительности.
- Увеличение количества значений в `my.cnf` может привести к «замене», что является *серьезной* проблемой производительности.
- «Префиксные индексы» (например, `INDEX(foo(20))`), как правило, бесполезны.
- `OPTIMIZE TABLE` почти всегда бесполезен. (И это связано с блокировкой таблицы.)

Имейте ИНДЕКС

Самая важная вещь для ускорения запроса в любой крошечной таблице - иметь подходящий индекс.

```
WHERE a = 12  --> INDEX(a)
WHERE a > 12  --> INDEX(a)

WHERE a = 12 AND b > 78  --> INDEX(a,b) is more useful than INDEX(b,a)
WHERE a > 12 AND b > 78  --> INDEX(a) or INDEX(b); no way to handle both ranges

ORDER BY x  --> INDEX(x)
ORDER BY x, y  --> INDEX(x,y) in that order
ORDER BY x DESC, y ASC  --> No index helps - because of mixing ASC and DESC
```

Не скрывать функцию

Общей ошибкой является скрыть индексированный столбец внутри вызова функции. Например, этому не может помочь индекс:

```
WHERE DATE(dt) = '2000-01-01'
```

Вместо этого, учитывая `INDEX(dt)` они могут использовать индекс:

```
WHERE dt = '2000-01-01'  -- if `dt` is datatype `DATE`
```

Это работает для `DATE` , `DATETIME` , `TIMESTAMP` и даже `DATETIME(6)` (микросекунды):

```
WHERE dt >= '2000-01-01'
  AND dt  < '2000-01-01' + INTERVAL 1 DAY
```

ИЛИ ЖЕ

В общем, `OR` убивает оптимизацию.

```
WHERE a = 12 OR b = 78
```

не может использовать `INDEX(a,b)` и может или не может использовать `INDEX(a), INDEX(b)` посредством «слияния индекса». Слияние индекса лучше, чем ничего, но только едва.

```
WHERE x = 3 OR x = 5
```

превращается в

```
WHERE x IN (3, 5)
```

который *может* использовать индекс с `x` в нем.

подзапросов

Подзапросы приходят в нескольких вариантах, и у них есть другой потенциал оптимизации. Во-первых, обратите внимание, что подзапросы могут быть либо «коррелированными», либо «некоррелированными». Коррелированный означает, что они зависят от некоторого значения вне подзапроса. Это обычно подразумевает, что подзапрос *должен* быть переоценен для каждого внешнего значения.

Этот коррелированный подзапрос часто очень хорош. Примечание: оно должно возвращать не более 1 значения. Это часто полезно в качестве альтернативы, хотя и не обязательно быстрее, чем `LEFT JOIN` .

```
SELECT a, b, ( SELECT ... FROM t WHERE t.x = u.x ) AS c
    FROM u ...
SELECT a, b, ( SELECT MAX(x) ... ) AS c
    FROM u ...
SELECT a, b, ( SELECT x FROM t ORDER BY ... LIMIT 1 ) AS c
    FROM u ...
```

Обычно это некоррелировано:

```
SELECT ...
    FROM ( SELECT ... ) AS a
    JOIN b ON ...
```

Примечания к `FROM-SELECT` :

- Если он возвращает 1 строку, отлично.
- Хорошая парадигма (опять же «1 строка») для подзапроса `( SELECT @n := 0 )` , тем самым инициализируя `@variable для использования в остальном или запросе.
- Если он возвращает много строк, *а* `JOIN` также `( SELECT ... )` со многими строками, то эффективность может быть ужасной. Pre-5.6, не было индекса, поэтому он стал `CROSS JOIN` ; 5.6+ включает в себя вывод лучшего индекса в временные таблицы, а затем его создание, только для его выброса, когда он заканчивается с помощью `SELECT` .

ПРИСОЕДИНЯЙТЕСЬ + ГРУППА

Общая проблема, которая приводит к неэффективному запросу, выглядит примерно так:

```
SELECT ...
    FROM a
    JOIN b  ON ...
    WHERE ...
    GROUP BY a.id
```

Во-первых, `JOIN` расширяет количество строк; то `GROUP BY` уменьшает количество строк в `a` .

Возможно, не будет никаких хороших вариантов для решения этой проблемы с взрывом-

взломом. Один из возможных вариантов - превратить `JOIN` в коррелированный подзапрос в `SELECT` . Это также исключает `GROUP BY` .

Прочитайте Настройка производительности онлайн:
https://riptutorial.com/ru/mysql/topic/4292/настройка-производительности

глава 40: НОЛЬ

Examples

Использует для NULL

- Данные еще не известны - например, `end_date` , `rating`
- Необязательные данные - например, `middle_initial` (хотя это может быть лучше, чем пустая строка)
- 0/0 - результат определенных вычислений, таких как ноль, деленный на ноль.
- NULL не равно "" (пустая строка) или 0 (в случае целого).
- другие?

Тестирование NULL

- `IS NULL` / `IS NOT NULL` - `= NULL` не работает, как вы ожидаете.
- `x <=> y` - это «нулевое» сравнение.

В тестах `LEFT JOIN` для строк `a` для которых *нет* соответствующей строки в `b` .

```
SELECT ...
    FROM a
    LEFT JOIN b ON ...
    WHERE b.id IS NULL
```

Прочитайте НОЛЬ онлайн: https://riptutorial.com/ru/mysql/topic/6757/ноль

глава 41: ОБНОВИТЬ

Синтаксис

- UPDATE [LOW_PRIORITY] [IGNORE] tableName SET column1 = expression1, column2 = expression2, ... [WHERE conditions]; // Простое обновление одной таблицы
- UPDATE [LOW_PRIORITY] [IGNORE] tableName SET column1 = expression1, column2 = expression2, ... [WHERE conditions] [ORDER BY expression [ASC | DESC]] [LIMIT row_count]; // Обновление с порядком и ограничением
- UPDATE [LOW_PRIORITY] [IGNORE] table1, table2, ... SET column1 = expression1, column2 = expression2, ... [WHERE conditions]; // Обновление нескольких таблиц

Examples

Основное обновление

Обновление одной строки

```
UPDATE customers SET email='luke_smith@email.com' WHERE id=1
```

Этот запрос обновляет содержимое `email` в таблице `customers` в строке `luke_smith@email.com` где значение `id` равно 1. Старое и новое содержимое таблицы базы данных проиллюстрировано ниже слева и справа соответственно:

customers

id	firstname	lastname	email
1	Luke	Smith	luke@example.com
2	Anna	Carey	anna@example.com
3	Todd	Winters	todd@example.com

customers

id	firstname	lastname	email
1	Luke	Smith	luke_smith@email.com
2	Anna	Carey	anna@example.com
3	Todd	Winters	todd@example.com

Обновление всех строк

```
UPDATE customers SET lastname='smith'
```

Этот запрос обновляет содержимое `lastname` для каждой записи в таблице `customers` . Старое и новое содержимое таблицы базы данных показано ниже слева и справа соответственно:

customers

id	firstname	lastname	email
1	Luke	Smith	luke@example.com
2	Anna	Carey	anna@example.com
3	Todd	Winters	todd@example.com

customers

id	firstname	lastname	email
1	Luke	Smith	luke@example.com
2	Anna	Smith	anna@example.com
3	Todd	Smith	todd@example.com

Примечание. В запросе UPDATE необходимо использовать условные предложения (WHERE). Если вы не используете условное предложение, все записи атрибута этой таблицы будут обновлены. В приведенном выше примере новое значение (Смит) последнего имени в таблице клиентов установлено для всех строк.

Обновление с помощью шаблона объединения

Рассмотрим производственную таблицу под названием `questions_mysql` и таблицу `iwtQuestions` (импортированную `iwtQuestions` таблицу), представляющую последнюю партию импортированных данных CSV из `LOAD DATA INFILE` . Рабочий стол усекается перед импортом, данные импортируются, и этот процесс здесь не показан.

Обновите наши производственные данные, используя соединение с импортированными данными рабочей таблицы.

```
UPDATE questions_mysql q -- our real table for production
join iwtQuestions i -- imported worktable
ON i.qId = q.qId
SET q.closeVotes = i.closeVotes,
q.votes = i.votes,
q.answers = i.answers,
q.views = i.views;
```

Псевдонимы `q` и `i` используются для сокращения ссылок на таблицы. Это облегчает разработку и удобочитаемость.

`qId` , первичный ключ, представляет собой идентификатор вопроса Stackoverflow. Четыре столбца обновляются для сопоставления строк из соединения.

UPDATE с ORDER BY и LIMIT

Если предложение `ORDER BY` указано в вашем заявлении SQL SQL, строки обновляются в указанном порядке.

Если предложение `LIMIT` указано в вашем операторе SQL, это ограничивает количество строк, которые могут быть обновлены. Нет предела, если предложение `LIMIT` не указано.

`ORDER BY` и `LIMIT` не могут использоваться для обновления нескольких таблиц.

Синтаксис для MySQL `UPDATE` с `ORDER BY` и `LIMIT` - это,

```
UPDATE [ LOW_PRIORITY ] [ IGNORE ]
tableName
SET column1 = expression1,
    column2 = expression2,
    ...
[WHERE conditions]
[ORDER BY expression [ ASC | DESC ]]
[LIMIT row_count];

---> Example
UPDATE employees SET isConfirmed=1 ORDER BY joiningDate LIMIT 10
```

В приведенном выше примере 10 строк будут обновляться в соответствии с порядком сотрудников `joiningDate` .

Несколько таблиц UPDATE

В нескольких таблицах `UPDATE` он обновляет строки в каждой указанной таблице, которые удовлетворяют условиям. Каждая соответствующая строка обновляется один раз, даже если она соответствует условиям несколько раз.

В нескольких таблицах `UPDATE` , `ORDER BY` и `LIMIT` не могут использоваться.

Синтаксис для нескольких таблиц `UPDATE` ,

```
UPDATE [LOW_PRIORITY] [IGNORE]
table1, table2, ...
    SET column1 = expression1,
        column2 = expression2,
        ...
    [WHERE conditions]
```

Например, рассмотрим две таблицы, `products` и `salesOrders` . В случае, если мы уменьшим количество конкретного продукта из уже заказанного заказа клиента. Затем нам также необходимо увеличить это количество в нашем столбце запасов таблицы `products` . Это можно сделать в одном операторе обновления SQL, как показано ниже.

```
UPDATE products, salesOrders
  SET salesOrders.Quantity = salesOrders.Quantity - 5,
      products.availableStock = products.availableStock + 5
WHERE products.productId = salesOrders.productId
  AND salesOrders.orderId = 100 AND salesOrders.productId = 20;
```

В приведенном выше примере количество «5» будет уменьшено из таблицы `salesOrders` и то же самое будет увеличено в таблице `products` соответствии с условиями `WHERE` .

Массовое ОБНОВЛЕНИЕ

При обновлении нескольких строк с разными значениями гораздо быстрее использовать массовое обновление.

```
UPDATE people
SET name =
  (CASE id WHEN 1 THEN 'Karl'
           WHEN 2 THEN 'Tom'
           WHEN 3 THEN 'Mary'
   END)
WHERE id IN (1,2,3);
```

При массовом обновлении только один запрос может быть отправлен на сервер вместо одного запроса для каждой обновляемой строки. Случаи должны содержать все возможные параметры, рассмотренные в `WHERE` .

Прочитайте ОБНОВИТЬ онлайн: https://riptutorial.com/ru/mysql/topic/2738/обновить

глава 42: Обработка временных зон

замечания

Когда вам нужно обрабатывать информацию о времени для всемирной пользовательской базы в MySQL, используйте тип данных TIMESTAMP в своих таблицах.

Для каждого пользователя сохраните столбец часового пояса пользователя. VARCHAR (64) - хороший тип данных для этого столбца. Когда пользователь регистрируется для использования вашей системы, запросите значение часового пояса. Мое - Атлантическое время, `America/Edmonton` . Может быть, может быть, может быть `Asia/Kolkata` или `Australia/NSW` . Для пользовательского интерфейса для этого параметра предпочтения пользователя, WordPress.org имеет хороший пример.

Наконец, всякий раз, когда вы устанавливаете соединение с вашей хост-программой (Java, php, что угодно) в вашу СУБД от имени пользователя, выполните команду SQL

```
SET SESSION time_zone='(whatever tz string the user gave you)'
```

прежде чем обращаться с любыми пользовательскими данными, связанными с временем. Затем все время `TIMESTAMP` вы установили, будет отображаться в локальное время пользователя.

Это приведет к тому, что все ваши таблицы будут конвертированы в UTC и все время будут переведены на локальный. Он корректно работает для NOW () и CURDATE (). Опять же, для этого вы должны использовать TIMESTAMP, а не DATETIME или DATE.

Убедитесь, что для вашей серверной ОС и часовых поясов MySQL по умолчанию установлено значение UTC. Если вы не сделаете этого, прежде чем вы начнете загружать информацию в свою базу данных, ее будет практически невозможно исправить. Если вы используете поставщика для запуска MySQL, настаивайте на том, что это правильно.

Examples

Получить текущую дату и время в определенном часовом поясе.

Это получает значение `NOW()` по местному времени, в стандартное время Индии, а затем снова в UTC.

```
SELECT NOW();
SET time_zone='Asia/Kolkata';
SELECT NOW();
SET time_zone='UTC';
SELECT NOW();
```

Преобразовать сохраненное значение DATE или DATETIME в другой часовой пояс.

Если у вас есть сохраненный `DATE` или `DATETIME` (в каком-то столбце), он хранился в отношении некоторого часового пояса, но в MySQL часовой пояс *не* сохраняется со значением. Итак, если вы хотите преобразовать его в другой часовой пояс, вы можете, но вы должны знать исходный часовой пояс. Использование `CONVERT_TZ()` делает преобразование. В этом примере показаны строки, проданные в Калифорнии по местному времени.

```
SELECT CONVERT_TZ(date_sold,'UTC','America/Los_Angeles') date_sold_local
  FROM sales
 WHERE state_sold = 'CA'
```

Получить сохраненные значения TIMESTAMP в определенном часовом поясе

Это очень просто. Все значения `TIMESTAMP` сохраняются в универсальном времени и всегда преобразуются в `time_zone` настройку `time_zone` всякий раз, когда они отображаются.

```
SET SESSION time_zone='America/Los_Angeles';
SELECT timestamp_sold
  FROM sales
 WHERE state_sold = 'CA'
```

Почему это? Значения `TIMESTAMP` основаны на почтенном типе данных UNIX `time_t` . Эти временные метки UNIX сохраняются как количество секунд с `1970-01-01 00:00:00` UTC.

Уведомление Значения `TIMESTAMP` хранятся в универсальном времени. Значения `DATE` и `DATETIME` сохраняются в любое местное время, когда они были сохранены.

Какова локальная настройка часового пояса моего сервера?

На каждом сервере установлен глобальный параметр time_zone по умолчанию, настроенный владельцем серверной машины. Вы можете определить текущий часовой пояс таким образом:

```
SELECT @@time_zone
```

К сожалению, это обычно дает значение `SYSTEM` , а это означает, что время MySQL определяется настройкой часового пояса OS сервера.

Эта последовательность запросов (да, это взлом) возвращает вам смещение в минутах между настройкой часового пояса сервера и UTC.

```
CREATE TEMPORARY TABLE times (dt DATETIME, ts TIMESTAMP);
SET time_zone = 'UTC';
INSERT INTO times VALUES(NOW(), NOW());
SET time_zone = 'SYSTEM';
SELECT dt, ts, TIMESTAMPDIFF(MINUTE, dt, ts)offset FROM times;
DROP TEMPORARY TABLE times;
```

Как это работает? Эти две колонки во временной таблице с разными типами данных являются ключом. Типы данных `DATETIME` всегда хранятся в локальном времени в таблицах, а `TIMESTAMP` s - в формате UTC. Таким образом, оператор `INSERT` , выполняемый, когда time_zone установлен в UTC, сохраняет два идентичных значения даты / времени.

Затем оператор SELECT выполняется, когда для параметра time_zone задано локальное время сервера. `TIMESTAMP` s всегда переводится с сохраненной формы UTC в локальное время в операторы SELECT. `DATETIME` - нет. Таким образом, операция `TIMESTAMPDIFF(MINUTE...)` вычисляет разницу между локальным и универсальным временем.

Какие значения time_zone доступны на моем сервере?

Чтобы получить список возможных значений time_zone в экземпляре сервера MySQL, используйте эту команду.

```
SELECT mysql.time_zone_name.name
```

Обычно это показывает список зон зон ZoneInfo, поддерживаемый Полом Эггерт в авторитете Интернет-назначенных номеров . Во всем мире имеется около 600 часовых поясов.

Unix-подобные операционные системы (дистрибутивы Linux, дистрибутивы BSD и современные дистрибутивы Mac OS), например, получают обычные обновления. Установка этих обновлений в операционной системе позволяет экземплярам MySQL, работающим там, отслеживать изменения в часовом поясе и перехода на дневное / стандартное время.

Если вы получаете гораздо более короткий список имен часовых поясов, ваш сервер либо не полностью настроен, либо работает в Windows. Ниже приведены инструкции для администратора вашего сервера для установки и ведения списка ZoneInfo.

Прочитайте Обработка временных зон онлайн: https://riptutorial.com/ru/mysql/topic/7849/обработка-временных-зон

глава 43: Обратные кавычки

Examples

Использование Backticks

Существует много примеров, в которых обратные ссылки используются внутри запроса, но для многих до сих пор неясно, когда и где использовать обратные вызовы `` .

Backticks в основном используются для предотвращения ошибки, называемой « *зарезервированное слово MySQL* ». При создании таблицы в PHPmyAdmin вы иногда сталкиваетесь с предупреждением или предупреждением о том, что вы используете « *зарезервированное слово MySQL* ».

Например, когда вы создаете таблицу с столбцом с именем « `group` », вы получаете предупреждение. Это связано с тем, что вы можете сделать следующий запрос:

```
SELECT student_name, AVG(test_score) FROM student GROUP BY group
```

Чтобы убедиться, что вы не получили ошибку в своем запросе, вам нужно использовать обратные такты, чтобы ваш запрос становился следующим:

```
SELECT student_name, AVG(test_score) FROM student GROUP BY `group`
```

Таблица

Не только имена столбцов могут быть окружены обратными окнами, но также и именами таблиц. Например, когда вам нужно `JOIN` нескольким таблицам.

```
SELECT `users`.`username`, `groups`.`group` FROM `users`
```

Легче читать

Как вы можете видеть, использование обратных ссылок вокруг имен таблиц и столбцов также упрощает чтение запроса.

Например, когда вы используете для записи запросов в нижнем регистре:

```
select student_name, AVG(test_score) from student group by group
select `student_name`, AVG(`test_score`) from `student` group by `group`
```

См. Страницу руководства MySQL под названием « Ключевые слова и зарезервированные

слова» . Те, у кого есть (R), являются Зарезервированными Словами. Остальные - это просто ключевые слова. Зарезервированные требуют особой осторожности.

Прочитайте Обратные кавычки онлайн: https://riptutorial.com/ru/mysql/topic/5208/обратные-кавычки

глава 44: Ограничение и смещение

Синтаксис

- SELECT column_1 [, column_2]
  FROM table_1
  ORDER BY order_column
  LIMIT row_count [OFFSET row_offset]
- SELECT column_1 [, column_2]
  FROM table_1
  ORDER BY order_column
  LIMIT [row_offset,] row_count

замечания

«Предел» может означать «Максимальное количество строк в таблице».

«Смещение» означает выбор из номера `row` (чтобы не путать значение первичного ключа или любое значение данных поля)

Examples

Ограничение и смещение

Учитывая следующую таблицу `users` :

Я бы	имя пользователя
1	User1
2	Пользователь2
3	User3
4	Пользователь4
5	USER5

Чтобы ограничить количество строк в результирующем наборе запроса `SELECT` , предложение `LIMIT` может использоваться вместе с одним или двумя положительными целыми числами в качестве аргументов (включая нуль).

Предложение `LIMIT` с одним аргументом

Когда используется один аргумент, набор результатов будет ограничен только числом, указанным следующим образом:

```
SELECT * FROM users ORDER BY id ASC LIMIT 2
```

Я бы	имя пользователя
1	User1
2	Пользователь2

Если значение аргумента равно `0` , набор результатов будет пустым.

Также обратите внимание, что предложение `ORDER BY` может быть важным, чтобы указать первые строки набора результатов, которые будут представлены (при заказе другим столбцом).

Предложение `LIMIT` с двумя аргументами

Когда в предложении `LIMIT` используются два аргумента:

- **первый** аргумент представляет строку, из которой будут представлены строки результирующего набора, - это число часто упоминается как **смещение** , так как оно представляет строку, предшествующую начальной строке ограниченного результирующего набора. Это позволяет аргументу принимать `0` как значение и, таким образом, принимать во внимание первую строку набора без ограничений.
- **второй** аргумент указывает максимальное количество строк, возвращаемых в результирующий набор (аналогично примеру одного аргумента).

Поэтому запрос:

```
SELECT * FROM users ORDER BY id ASC LIMIT 2, 3
```

Представляет следующий набор результатов:

Я бы	имя пользователя
3	User3
4	Пользователь4
5	USER5

Обратите внимание, что когда аргумент **offset** равен `0` , результирующий набор будет эквивалентен одному аргументу `LIMIT` . Это означает, что следующие два запроса:

```
SELECT * FROM users ORDER BY id ASC LIMIT 0, 2

SELECT * FROM users ORDER BY id ASC LIMIT 2
```

Произведите тот же набор результатов:

Я бы	имя пользователя
1	User1
2	Пользователь2

Ключевое слово `OFFSET` : альтернативный синтаксис

Альтернативный синтаксис для предложения `LIMIT` с двумя аргументами заключается в использовании ключевого слова `OFFSET` после первого аргумента следующим образом:

```
SELECT * FROM users ORDER BY id ASC LIMIT 2 OFFSET 3
```

Этот запрос вернет следующий набор результатов:

Я бы	имя пользователя
3	User3
4	Пользователь4

Обратите внимание, что в этом альтернативном синтаксисе аргументы переключают свои позиции:

- **первый** аргумент представляет количество строк, возвращаемых в результирующий набор;
- **второй** аргумент представляет собой смещение.

Прочитайте Ограничение и смещение онлайн: https://riptutorial.com/ru/mysql/topic/548/ограничение-и-смещение

глава 45: Один ко многим

Вступление

Идея от одного до многих (1: M) касается объединения строк друг с другом, в частности случаев, когда одна строка в одной таблице соответствует многим строкам в другой.

1: M является однонаправленным, то есть в любое время, когда вы запрашиваете отношения 1: M, вы можете использовать строку «один» для выбора «многих» строк в другой таблице, но вы не можете использовать одну строку «много» для выберите более одной строки.

замечания

В большинстве случаев работа с соотношением 1: M требует от нас понимания *первичных ключей* и *внешних ключей* .

Первичный ключ - это столбец в таблице, где любая отдельная строка этого столбца представляет собой единый объект или, выбирая значение в столбце первичного ключа, приводит к точно одной строке. Используя приведенные выше примеры, EMP_ID представляет одного сотрудника. Если вы запрашиваете какой-либо один EMP_ID, вы увидите одну строку, соответствующую соответствующему сотруднику.

Внешний ключ - это столбец в таблице, который соответствует первичному ключу другой другой таблицы. В нашем примере выше MGR_ID в таблице EMPLOYEES является внешним ключом. Обычно для объединения двух таблиц вы присоединяетесь к ним на основе первичного ключа одной таблицы и внешнего ключа в другом.

Examples

Примеры таблиц компании

Рассмотрим компанию, в которой каждый сотрудник, который является менеджером, управляет 1 или более сотрудниками, и каждый сотрудник имеет только 1 менеджера.

Это приводит к двум таблицам:

СОТРУДНИКИ

EMP_ID	ИМЯ	ФАМИЛИЯ	MGR_ID
E01	Джонни	Яблочное	M02

EMP_ID	ИМЯ	ФАМИЛИЯ	MGR_ID
E02	Erin	Macklemore	M01
E03	Colby	Оформление документации	M03
E04	Рон	Sonswan	M01

РУКОВОДИТЕЛИ

MGR_ID	ИМЯ	ФАМИЛИЯ
M01	Громко	McQueen
M02	Властный	Штаны
M03	бочка	Джонс

Получить сотрудников, управляемых одним менеджером

```
SELECT e.emp_id , e.first_name , e.last_name FROM employees e INNER JOIN managers m ON m.mgr_id
= e.mgr_id WHERE m.mgr_id = 'M01' ;
```

Результаты в:

EMP_ID	ИМЯ	ФАМИЛИЯ
E02	Erin	Macklemore
E04	Рон	Sonswan

В конечном счете, для каждого менеджера, которого мы запрашиваем, мы увидим, что 1 или более сотрудников возвращены.

Получить менеджера для одного сотрудника

При просмотре этого примера ознакомьтесь с приведенными выше примерами таблиц.

```
SELECT m.mgr_id , m.first_name , m.last_name FROM managers m INNER JOIN employees e ON e.mgr_id
= m.mgr_id WHERE e.emp_id = 'E03' ;
```

MGR_ID	ИМЯ	ФАМИЛИЯ
M03	бочка	Джонс

Поскольку это является обратным приведенному выше примеру, мы знаем, что для каждого сотрудника, которого мы запрашиваем, мы увидим только одного соответствующего

менеджера.

Прочитайте Один ко многим онлайн: https://riptutorial.com/ru/mysql/topic/9600/один-ко-многим

глава 46: Операции даты и времени

Examples

Сейчас()

```
Select Now();
```

Показывает текущую дату и время сервера.

```
Update `footable` set mydatefield = Now();
```

Это обновит поле `mydatefield` с текущей датой и временем сервера в настроенном часовом поясе сервера, например

```
'2016-07-21 12:00:00'
```

Арифметика даты

```
NOW() + INTERVAL 1 DAY  -- This time tomorrow

CURDATE() - INTERVAL 4 DAY -- Midnight 4 mornings ago
```

Покажите вопросы mysql, которые были заданы от 3 до 10 часов назад (от 180 до 600 минут назад):

```
SELECT qId,askDate,minuteDiff
FROM
(   SELECT qId,askDate,
    TIMESTAMPDIFF(MINUTE,askDate,now()) as minuteDiff
    FROM questions_mysql
) xDerived
WHERE minuteDiff BETWEEN 180 AND 600
ORDER BY qId DESC
LIMIT 50;

+----------+---------------------+------------+
| qId      | askDate             | minuteDiff |
+----------+---------------------+------------+
| 38546828 | 2016-07-23 22:06:50 |        182 |
| 38546733 | 2016-07-23 21:53:26 |        195 |
| 38546707 | 2016-07-23 21:48:46 |        200 |
| 38546687 | 2016-07-23 21:45:26 |        203 |
| ...      |                     |            |
+----------+---------------------+------------+
```

Страницы руководства MySQL для `TIMESTAMPDIFF()` .

Остерегайтесь Не пытайтесь использовать выражения типа `CURDATE() + 1` для арифметики даты в MySQL. Они не возвращают то, что вы ожидаете, особенно если вы привыкли к продукту базы данных Oracle. `CURDATE() + INTERVAL 1 DAY` используйте `CURDATE() + INTERVAL 1 DAY` .

Тестирование по диапазону дат

Хотя очень заманчиво использовать `BETWEEN` ... `AND` ... для диапазона дат, это проблематично. Вместо этого этот шаблон позволяет избежать большинства проблем:

```
WHERE x >= '2016-02-25'
  AND x  < '2016-02-25' + INTERVAL 5 DAY
```

Преимущества:

- `BETWEEN` является «включенным», таким образом, включая окончательную дату или вторую.
- `23:59:59` неудобно и неправильно, если у вас разрешение на микросекунду на `DATETIME` .
- Этот шаблон избегает использования високосных лет и других расчетов данных.
- Он работает, является ли `x` `DATE` , `DATETIME` или `TIMESTAMP` .

SYSDATE (), NOW (), CURDATE ()

```
  SELECT SYSDATE();
```

Эта функция возвращает текущую дату и время как значение в `'YYYY-MM-DD HH:MM:SS'` или `YYYYMMDDHHMMSS` , в зависимости от того, используется ли функция в строковом или числовом контексте. Он возвращает дату и время в текущем часовом поясе.

```
  SELECT NOW();
```

Эта функция является синонимом `SYSDATE()` .

```
  SELECT CURDATE();
```

Эта функция возвращает текущую дату без какого-либо времени в качестве значения в `'YYYY-MM-DD'` или `YYYYMMDD` зависимости от того, используется ли эта функция в строковом или числовом контексте. Он возвращает дату в текущем часовом поясе.

Извлечь дату из заданной даты или выражения DateTime

```
SELECT DATE('2003-12-31 01:02:03');
```

Выход будет:

```
2003-12-31
```

Использование индекса для поиска по дате и времени

Многие таблицы базы данных реального мира имеют много строк с значениями столбца `DATETIME` или `TIMESTAMP` охватывающими много времени, включая годы или даже десятилетия. Часто необходимо использовать предложение `WHERE` для получения некоторого подмножества этого промежутка времени. Например, мы можем захотеть извлечь строки из таблицы с датой 1 сентября-2016.

Неэффективный способ сделать это:

```
WHERE DATE(x) = '2016-09-01'   /* slow! */
```

Он неэффективен, потому что он применяет функцию - `DATE()` - к значениям столбца. Это означает, что MySQL должен проверять каждое значение `x` , и индекс не может использоваться.

Лучшим способом выполнения операции является

```
WHERE x >= '2016-09-01'
  AND x <  '2016-09-01' + INTERVAL 1 DAY
```

Это выбирает диапазон значений `x` лежащих где-нибудь в рассматриваемый день, вплоть до, но *не включая* (следовательно `<`) полночь на следующий день.

Если таблица имеет индекс в столбце `x` , то сервер базы данных может выполнять сканирование диапазона по индексу. Это означает, что он может быстро найти первое соответствующее значение x, а затем последовательно сканировать индекс, пока не найдет последнее соответствующее значение. Сканирование диапазона индекса намного более эффективно, чем полное сканирование таблицы, которое требуется `DATE(x) = '2016-09-01` .

Не искушайтесь использовать это, хотя он выглядит более эффективным.

```
  WHERE x BETWEEN '2016-09-01' AND '2016-09-01' + INTERVAL 1 DAY  /*  wrong! */
```

Он имеет ту же эффективность, что и сканирование диапазона, но он будет выбирать строки со значениями `x` падающими точно в полночь 2 сентября 2016 года, чего вы не хотите.

Прочитайте Операции даты и времени онлайн: https://riptutorial.com/ru/mysql/topic/1882/операции-даты-и-времени

глава 47: Ошибка 1055: ONLY_FULL_GROUP_BY: что-то не в условии GROUP BY ...

Вступление

Недавно новые версии серверов MySQL начали генерировать 1055 ошибок для запросов, которые раньше работали. В этом разделе объясняются эти ошибки. Команда MySQL работает над отказом от нестандартного расширения до `GROUP BY` или, по крайней мере, для того, чтобы затруднить его работу с разработчиками запросов.

замечания

В течение долгого времени MySQL содержал печально известное нестандартное расширение `GROUP BY` , которое позволяет использовать поведение oddball во имя эффективности. Это расширение позволило бесчисленным разработчикам по всему миру использовать `GROUP BY` в производственном коде без полного понимания того, что они делают.

В частности, это плохая идея использовать `SELECT *` в запросе `GROUP BY` , потому что стандартное предложение `GROUP BY` требует перечисления столбцов. К сожалению, многие разработчики этого сделали.

Прочитай это. https://dev.mysql.com/doc/refman/5.7/en/group-by-handling.html

Команда MySQL пытается исправить эту ошибку, не испортив производственный код. Они добавили флаг `sql_mode` в 5.7.5 с именем `ONLY_FULL_GROUP_BY` чтобы заставить стандартное поведение. В недавнем выпуске они по умолчанию включили этот флаг. Когда вы обновили свой локальный MySQL до 5.7.14, флаг включился, и ваш производственный код, зависящий от старого расширения, перестает работать.

Если вы недавно начали получать 1055 ошибок, каковы ваши варианты?

1. исправить оскорбительные SQL-запросы или заставить их авторов сделать это.
2. перейдите к версии совместимого с MySQL совместимого программного обеспечения, которое вы используете.
3. измените `sql_mode` вашего сервера, чтобы избавиться от нового режима `ONLY_FULL_GROUP_BY` .

Вы можете изменить режим, выполнив команду `SET` .

```
SET  sql_mode =
'STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENG
```

должен сделать трюк, если вы сделаете это сразу после того, как ваше приложение подключится к MySQL.

Или вы можете найти файл инициализации в вашей установке MySQL , найти `sql_mode=` и изменить его, чтобы опустить `ONLY_FULL_GROUP_BY` , и перезагрузить сервер.

Examples

Использование и неправильное использование GROUP BY

```
SELECT item.item_id, item.name,     /* not SQL-92 */
       COUNT(*) number_of_uses
 FROM item
 JOIN uses ON item.item_id, uses.item_id
GROUP BY item.item_id
```

покажет строки в таблице, называемой `item` , и покажет количество связанных строк в таблице с именем `uses` . Это хорошо работает, но, к сожалению, это не стандартный SQL-92.

Почему бы и нет? потому что предложение `SELECT` (и предложение `ORDER BY`) в запросах `GROUP BY` должно содержать столбцы, которые

1. указанных в предложении `GROUP BY` , или
2. совокупные функции, такие как `COUNT()` , `MIN()` и тому подобное.

В предложении `SELECT` этого примера упоминается `item.name` , столбец, который не соответствует ни одному из этих критериев. MySQL 5.6 и ранее отклонят этот запрос, если режим SQL содержит `ONLY_FULL_GROUP_BY` .

Этот примерный запрос может быть выполнен в соответствии со стандартом SQL-92, изменив предложение `GROUP BY` , как это.

```
SELECT item.item_id, item.name,
       COUNT(*) number_of_uses
  FROM item
  JOIN uses ON item.item_id, uses.item_id
 GROUP BY item.item_id, item.name
```

Более поздний стандарт SQL-99 позволяет `SELECT` пропускать неагрегированные столбцы из группового ключа, если СУБД может доказать функциональную зависимость между ними и столбцами группового ключа. Поскольку `item.name` функционально зависит от `item.item_id` , исходный пример действителен SQL-99. MySQL получил версию

функциональной зависимости в версии 5.7. Исходный пример работает под `ONLY_FULL_GROUP_BY` .

Неправильная GROUP BY возвращает непредсказуемые результаты: закон Мерфи

```
SELECT item.item_id, uses.category,   /* nonstandard */
       COUNT(*) number_of_uses
  FROM item
  JOIN uses ON item.item_id, uses.item_id
 GROUP BY item.item_id
```

покажет строки в таблице, называемой item, и покажет количество связанных строк в таблице с именем uses. Он также покажет значение столбца, называемого `uses.category` .

Этот запрос работает в MySQL (до `ONLY_FULL_GROUP_BY` флажка `ONLY_FULL_GROUP_BY`). Он использует нестандартное расширение MySQL для `GROUP BY` .

Но у запроса есть проблема: если несколько строк в таблице `uses` соответствуют условию `ON` в предложении `JOIN` , MySQL возвращает столбец `category` только из одной из этих строк. Какая строка? Писатель запроса и пользователь приложения не знают об этом заранее. Формально говоря, это *непредсказуемо* : MySQL может вернуть любое значение, которое он хочет.

Непредсказуемый случайный, с одним существенным различием. Можно ожидать *случайного* выбора время от времени. Поэтому, если выбор был случайным, вы можете обнаружить его во время отладки или тестирования. *Непредсказуемый* результат хуже: MySQL возвращает тот же результат каждый раз, когда вы используете запрос, *пока он этого не сделает.* Иногда это новая версия сервера MySQL, которая приводит к другому результату. Иногда это вызывает растущую таблицу, вызывающую проблему. Что может пойти не так, пойдет не так, и когда вы этого не ожидаете. Это называется законом Мерфи .

Команда MySQL работает над созданием этой ошибки для разработчиков. Более новые версии MySQL в 5,7 последовательности имеют `sql_mode` флаг под названием `ONLY_FULL_GROUP_BY` . Когда этот флаг установлен, сервер MySQL возвращает ошибку 1055 и отказывается запускать такой запрос.

Неправильная команда GROUP BY с SELECT *, и как ее исправить.

Иногда запрос выглядит так: `*` в предложении `SELECT` .

```
SELECT item.*,     /* nonstandard */
       COUNT(*) number_of_uses
 FROM item
 JOIN uses ON item.item_id, uses.item_id
GROUP BY item.item_id
```

Такой запрос должен быть реорганизован для соответствия стандарту `ONLY_FULL_GROUP_BY` .

Для этого нам нужен подзапрос, который правильно использует `GROUP BY` чтобы вернуть значение `number_of_uses` для каждого `item_id` . Этот подзапрос короткий и сладкий, потому что ему нужно только посмотреть таблицу `uses` .

```
                              SELECT item_id, COUNT(*) number_of_uses
                                FROM  uses
                               GROUP BY item_id
```

Затем мы можем присоединиться к этому подзапросу с помощью таблицы `item` .

```
SELECT item.*, usecount.number_of_uses
  FROM item
  JOIN (
                              SELECT item_id, COUNT(*) number_of_uses
                                FROM  uses
                               GROUP BY item_id
       ) usecount ON item.item_id = usecount.item_id
```

Это позволяет сделать предложение `GROUP BY` простым и правильным, а также позволяет использовать спецификатор `*` .

Примечание: тем не менее, мудрые разработчики избегают использования спецификатора `*` в любом случае. Обычно лучше перечислять столбцы, которые вы хотите в запросе.

ANY_VALUE ()

```
SELECT item.item_id, ANY_VALUE(uses.tag) tag,
       COUNT(*) number_of_uses
 FROM item
 JOIN uses ON item.item_id, uses.item_id
GROUP BY item.item_id
```

показывает строки в таблице, называемой `item` , количество связанных строк и одно из значений в связанной таблице, называемой `uses` .

Вы можете рассматривать эту `ANY_VALUE()` как странную своего рода совокупную функцию. Вместо того, чтобы возвращать счет, сумму или максимум, он инструктирует сервер MySQL выбирать произвольно одно значение из рассматриваемой группы. Это способ работы с ошибкой 1055.

Будьте внимательны при использовании `ANY_VALUE()` в запросах в производственных приложениях.

Его действительно следует называть `SURPRISE_ME()` . Он возвращает значение некоторой строки в группе GROUP BY. Какая строка возвращается, является неопределенной. Это означает, что это полностью зависит от сервера MySQL. Формально он возвращает непредсказуемое значение.

Сервер не выбирает случайное значение, это хуже, чем это. Он возвращает одно и то же значение каждый раз, когда вы запускаете запрос, пока он этого не сделает. Он может измениться или нет, когда таблица растет или сжимается, или когда сервер имеет больше или меньше ОЗУ, или когда версия сервера изменяется, или когда Марс находится в ретроградном (что бы это ни значило) или вообще без причины.

Вы были предупреждены.

Прочитайте Ошибка 1055: ONLY_FULL_GROUP_BY: что-то не в условии GROUP BY ... онлайн: https://riptutorial.com/ru/mysql/topic/8245/ошибка-1055--only-full-group-by--что-то-не-в-условии-group-by----

глава 48: Падение таблицы

Синтаксис

- DROP TABLE имя_таблицы;
- DROP TABLE IF EXISTS имя_таблицы; - избежать ошибки в автоматическом скрипте
- ТАБЛИЦА DROP t1, t2, t3; - Несколько таблиц DROP
- ВРЕМЕННАЯ ТАБЛИЦА ВРЕМЕНИ t; - DROP таблица из CREATE TEMPORARY TABLE ...

параметры

параметры	подробности
ВРЕМЕННОЕ	Необязательный. Он указывает, что только операторские сокращения должны быть сброшены только с помощью инструкции DROP TABLE.
ЕСЛИ СУЩЕСТВУЕТ	Необязательный. Если указано, оператор DROP TABLE не будет вызывать ошибку, если одна из таблиц не существует.

Examples

Падение таблицы

Drop Table используется для удаления таблицы из базы данных.

Создание таблицы:

Создание таблицы с именем tbl, а затем удаление созданной таблицы

```
CREATE TABLE tbl(
    id INT NOT NULL AUTO_INCREMENT,
    title VARCHAR(100) NOT NULL,
    author VARCHAR(40) NOT NULL,
    submission_date DATE,
    PRIMARY KEY (id)
);
```

Таблица падения:

```
DROP TABLE tbl;
```

ПОЖАЛУЙСТА, ОБРАТИТЕ ВНИМАНИЕ

Отбрасывающая таблица полностью удалит таблицу из базы данных и всей ее информации, и она не будет восстановлена.

Удаление таблиц из базы данных

```
DROP TABLE Database.table_name
```

Прочитайте Падение таблицы онлайн: https://riptutorial.com/ru/mysql/topic/4123/падение-таблицы

глава 49: Подготовить заявления

Синтаксис

- PREPARE stmt_name FROM prepareable_stmt
- EXECUTE stmt_name [ИСПОЛЬЗОВАНИЕ @var_name [, @var_name] ...]
- {DEALLOCATE | DROP} PREPARE stmt_name

Examples

ПОДГОТОВИТЬ, ВЫПОЛНИТЬ И ДОСТИГАТЬ ПОДГОТОВИТЬ ЗАЯВЛЕНИЯ

PREPARE готовит заявление для выполнения

EXECUTE выполняет подготовленное заявление

DEALLOCATE PREPARE публикует подготовленное заявление

```
SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
PREPARE stmt2 FROM @s;
SET @a = 6;
SET @b = 8;
EXECUTE stmt2 USING @a, @b;
```

Результат:

```
+------------+
| hypotenuse |
+------------+
|         10 |
+------------+
```

В заключение,

```
DEALLOCATE PREPARE stmt2;
```

Заметки:

- Вы должны использовать переменные @variables, а не DECLAREd для `FROM @s`
- Первичное использование Prepare и т. Д. Заключается в том, чтобы «построить» запрос для ситуаций, когда привязка не будет работать, например, вставка имени таблицы.

Построить и выполнить

(Это запрос для хорошего примера, который показывает, как *построить* `SELECT` с помощью `CONCAT` , затем подготовить + выполнить его. Пожалуйста, обратите внимание на использование переменных @variables и DECLAREd - это имеет большое значение, и это то, что новички (включая себя) спотыкаются.)

Изменить таблицу с добавлением столбца

```
SET v_column_definition := CONCAT(
  v_column_name
  ,' ',v_column_type
  ,' ',v_column_options
);

SET @stmt := CONCAT('ALTER TABLE ADD COLUMN ', v_column_definition);

PREPARE stmt FROM @stmt;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
```

Прочитайте Подготовить заявления онлайн: https://riptutorial.com/ru/mysql/topic/2603/подготовить-заявления

глава 50: Подключение к UTF-8 с использованием различного языка программирования.

Examples

ПИТОН

1-я или 2-я строка в исходном коде (чтобы иметь литералы в коде utf8-encoded):

```
# -*- coding: utf-8 -*-
```

Подключение:

```
db = MySQLdb.connect(host=DB_HOST, user=DB_USER, passwd=DB_PASS, db=DB_NAME,
        charset="utf8mb4", use_unicode=True)
```

Для веб-страниц один из них:

```
<meta charset="utf-8" />
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

PHP

В php.ini (это значение по умолчанию после PHP 5.6):

```
default_charset UTF-8
```

При создании веб-страницы:

```
header('Content-type: text/plain; charset=UTF-8');
```

При подключении к MySQL:

```
(for mysql:)   Do not use the mysql_* API!
(for mysqli:)  $mysqli_obj->set_charset('utf8mb4');
(for PDO:)     $db = new PDO('dblib:host=host;dbname=db;charset=utf8', $user, $pwd);
```

В коде не используйте никакие процедуры преобразования.

Для ввода данных,

```
<form accept-charset="UTF-8">
```

Для JSON, чтобы избежать `\uxxxx` :

```
$t = json_encode($s, JSON_UNESCAPED_UNICODE);
```

Прочитайте Подключение к UTF-8 с использованием различного языка программирования. онлайн: https://riptutorial.com/ru/mysql/topic/7332/подключение-к-utf-8-с-использованием-различного-языка-программирования-

глава 51: Полнотекстовый поиск

Вступление

MySQL предлагает поиск FULLTEXT. Он ищет таблицы со столбцами, содержащими текст для лучших совпадений слов и фраз.

замечания

`FULLTEXT` работает странно на таблицах, содержащих небольшое количество строк. Поэтому, когда вы экспериментируете с ним, вам может быть полезно получить таблицу среднего размера в Интернете. Вот таблица книг , с названиями и авторами. Вы можете загрузить его, разархивировать и загрузить в MySQL.

Поиск `FULLTEXT` предназначен для использования с помощью человека. Он предназначен для получения большего количества совпадений, чем обычная операция `WHERE column LIKE 'text%'` .

Поиск `FULLTEXT` доступен для таблиц `MyISAM` . Он также доступен для таблиц `InnoDB` в MySQL версии 5.6.4 или новее.

Examples

Простой поиск FULLTEXT

```
   SET @searchTerm= 'Database Programming';
SELECT MATCH (Title) AGAINST (@searchTerm IN NATURAL LANGUAGE MODE) Score,
       ISBN, Author, Title
  FROM book
 WHERE MATCH (Title) AGAINST (@searchTerm IN NATURAL LANGUAGE MODE)
 ORDER BY MATCH (Title) AGAINST (@searchTerm IN NATURAL LANGUAGE MODE) DESC;
```

Учитывая таблицу с именем `book` с колоннами по имени `ISBN` , «Title» и «Автор», это находит книги , соответствующие термины `'Database Programming'` . Сначала показаны лучшие совпадения.

Чтобы это работало, должен быть доступен полный текст в столбце « `Title` :

```
ALTER TABLE book ADD FULLTEXT INDEX Fulltext_title_index (Title);
```

Простой поиск BOOLEAN

```
   SET @searchTerm= 'Database Programming -Java';
```

```
SELECT MATCH (Title) AGAINST (@searchTerm IN BOOLEAN MODE) Score,
       ISBN, Author, Title
  FROM book
 WHERE MATCH (Title) AGAINST (@searchTerm IN BOOLEAN MODE)
 ORDER BY MATCH (Title) AGAINST (@searchTerm IN BOOLEAN MODE) DESC;
```

Для таблицы, названной `book` с столбцами с именем `ISBN` , `Title` и `Author` , это поиск книг со словами `'Database'` и `'Programming'` в названии, но не слово `'Java'` .

Чтобы это работало, должен быть доступен полный текст в столбце «Название»:

```
ALTER TABLE book ADD FULLTEXT INDEX Fulltext_title_index (Title);
```

Многостолбцовый поиск FULLTEXT

```
   SET @searchTerm= 'Date Database Programming';
SELECT MATCH (Title, Author) AGAINST (@searchTerm IN NATURAL LANGUAGE MODE) Score,
       ISBN, Author, Title
  FROM book
 WHERE MATCH (Title, Author) AGAINST (@searchTerm IN NATURAL LANGUAGE MODE)
 ORDER BY MATCH (Title, Author) AGAINST (@searchTerm IN NATURAL LANGUAGE MODE) DESC;
```

Учитывая таблицу с именем book с столбцами с именем `ISBN` , `Title` и `Author` , это находит книги, соответствующие терминам «Программирование базы данных дат». Сначала показаны лучшие совпадения. Лучшие матчи включают книги, написанные профессором CJ Date.

(Но один из лучших совпадений - это также *«Руководство по дате врача» для знакомства: как добраться от первой даты до идеального помощника* . Это показывает ограничение поиска FULLTEXT: оно не претендует на понимание таких вещей, как части речи или значение индексированных слов.)

Для этого необходимо иметь полнотекстовый индекс в столбцах Title и Author:

```
ALTER TABLE book ADD FULLTEXT INDEX Fulltext_title_author_index (Title, Author);
```

Прочитайте Полнотекстовый поиск онлайн: https://riptutorial.com/ru/mysql/topic/8759/полнотекстовый-поиск

глава 52: ПОСМОТРЕТЬ

Синтаксис

- CREATE VIEW view_name AS SELECT column_name (s) FROM table_name Условие WHERE; /// Простой синтаксис представления вида
- СОЗДАТЬ [ИЛИ ЗАМЕНИТЬ] [АЛГОРИТМ = {НЕ УКАЗАН | MERGE | TEMPTABLE}] [DEFINER = {user | CURRENT_USER}] [SQL SECURITY {DEFINER | INVOKER}] VIEW view_name [(column_list)] AS select_statement [WITH [CASCADED | LOCAL] CHECK OPTION]; /// Полное создание синтаксиса вида
- DROP VIEW [IF EXISTS] [db_name.] View_name; /// Синтаксис вида просмотра

параметры

параметры	подробности
view_name	Имя вида
Оператор SELECT	Операторы SQL должны быть упакованы в представления. Это может быть оператор SELECT для извлечения данных из одной или нескольких таблиц.

замечания

Представления представляют собой виртуальные таблицы и не содержат возвращаемых данных. Они могут спасти вас от написания сложных запросов снова и снова.

- **Перед представлением представления** его спецификация полностью состоит из `SELECT` . Оператор `SELECT` не может содержать подзапрос в предложении FROM.
- **После того, как представление сделано,** оно используется в основном так же, как и таблица, и может быть `SELECT` из таблицы точно так же, как и таблица.

Вы должны создавать представления, когда хотите ограничить несколько столбцов таблицы, от другого пользователя.

- Например: в вашей организации вы хотите, чтобы ваши менеджеры просматривали несколько сведений из таблицы с именем «Продажи», но вы не хотите, чтобы ваши разработчики программного обеспечения могли просматривать все поля таблицы «Продажи». Здесь вы можете создать два разных вида для ваших менеджеров и инженеров-программистов.

Производительность . `VIEWs` - синтаксический сахар. Однако производительность может быть или не быть хуже, чем эквивалентный запрос с выбранным скомпоном просмотра. Оптимизатор пытается сделать это «сбрасывать» для вас, но не всегда успешно. MySQL 5.7.6 предоставляет дополнительные улучшения в Оптимизаторе. Но, несмотря на то, что использование `VIEW` не будет генерировать более *быстрый* запрос.

Examples

Создать представление

привилегии

Оператор CREATE VIEW требует привилегии CREATE VIEW для представления и некоторую привилегию для каждого столбца, выбранного оператором SELECT. Для столбцов, используемых в другом месте в инструкции SELECT, вы должны иметь привилегию SELECT. Если предложение OR REPLACE присутствует, вы также должны иметь привилегию DROP для представления. CREATE VIEW может также требовать привилегии SUPER, в зависимости от значения DEFINER, как описано далее в этом разделе.

При обращении к представлению проверяется проверка привилегий.

Вид принадлежит базе данных. По умолчанию в базе данных по умолчанию создается новое представление. Чтобы создать представление явно в данной базе данных, используйте полное имя

Например:

db_name.view_name

```
mysql> CREATE VIEW test.v AS SELECT * FROM t;
```

Примечание. В базе данных базовые таблицы и представления используют одно и то же пространство имен, поэтому базовая таблица и представление не могут иметь одно и то же имя.

ПРОСМОТР:

- создаваться из множества видов операторов SELECT
- обратитесь к базовым таблицам или другим представлениям
- использовать объединения, UNION и подзапросы
- SELECT не нужно даже ссылаться на какие-либо таблицы

Другой пример

В следующем примере определяется представление, которое выбирает два столбца из

другой таблицы, а также выражение, вычисленное из этих столбцов:

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    3 |    50 |   150 |
+------+-------+-------+
```

ограничения

- Перед MySQL 5.7.7 оператор SELECT не может содержать подзапрос в предложении FROM.
- Оператор SELECT не может ссылаться на системные переменные или определяемые пользователем переменные.
- В хранимой программе оператор SELECT не может ссылаться на параметры программы или локальные переменные.
- Оператор SELECT не может ссылаться на подготовленные параметры оператора.
- Любая таблица или представление, упомянутые в определении, должны существовать. После того, как представление было создано, можно отбросить таблицу или просмотреть, что
  определение относится к. В этом случае использование представления приводит к ошибке. Чтобы проверить определение вида для подобных задач, используйте инструкцию CHECK TABLE.
- Определение не может относиться к ВРЕМЕННОЙ таблице, и вы не можете создайте ВРЕМЕННОЕ представление.
- Вы не можете связать триггер с представлением.
- Псевдонимы для имен столбцов в инструкции SELECT проверяются на максимальную длину столбца 64 символа (не максимальный псевдоним
  длина 256 символов).
- `VIEW` может или не может быть оптимизирован, а также эквивалент `SELECT` . Это вряд ли оптимизирует лучше.

Вид из двух таблиц

Представление наиболее полезно, когда его можно использовать для вывода данных из нескольких таблиц.

```
CREATE VIEW myview AS
SELECT a.*, b.extra_data FROM main_table a
LEFT OUTER JOIN other_table b
ON a.id = b.id
```

В представлениях mysql не реализованы. Если теперь выполнить простой запрос `SELECT *`

`FROM myview` , mysql фактически выполнит LEFT JOIN за сценой.

Созданный вид может быть присоединен к другим представлениям или таблицам

Обновление таблицы через VIEW

`VIEW` очень похож на таблицу. Хотя вы можете `UPDATE` таблицу, вы можете или не сможете обновить представление в этой таблице. В общем случае, если `SELECT` в представлении достаточно сложный, чтобы потребовать временную таблицу, то `UPDATE` не допускается.

Такие вещи, как `GROUP BY` , `UNION` , `HAVING` , `DISTINCT` и некоторые подзапросы, не позволяют обновлять представление. Подробности в *справочном руководстве* .

ПРОСМОТР ПРОСМОТРА

- Создать и удалить представление в текущей базе данных.

```
CREATE VIEW few_rows_from_t1 AS SELECT * FROM t1 LIMIT 10;
DROP VIEW few_rows_from_t1;
```

- Создать и отбросить представление, ссылающееся на таблицу в другой базе данных.

```
CREATE VIEW table_from_other_db AS SELECT x FROM db1.foo WHERE x IS NOT NULL;
DROP VIEW table_from_other_db;
```

Прочитайте ПОСМОТРЕТЬ онлайн: https://riptutorial.com/ru/mysql/topic/1489/посмотреть

глава 53: Преобразование из MyISAM в InnoDB

Examples

Базовая конверсия

```
ALTER TABLE foo ENGINE=InnoDB;
```

Это преобразует таблицу, но не заботится о каких-либо различиях между двигателями. Большинство различий не имеет значения, особенно для небольших таблиц. Но для более занятых столов следует учитывать другие соображения. *Соображения конверсии*

Преобразование всех таблиц в одну базу данных

Чтобы легко преобразовать все таблицы в одну базу данных, используйте следующее:

```
SET @DB_NAME = DATABASE();

SELECT  CONCAT('ALTER TABLE `', table_name, '` ENGINE=InnoDB;') AS sql_statements
FROM    information_schema.tables
WHERE   table_schema = @DB_NAME
AND     `ENGINE` = 'MyISAM'
AND     `TABLE_TYPE` = 'BASE TABLE';
```

> **ПРИМЕЧАНИЕ.** Чтобы работать, вы должны подключиться к своей базе данных для функции `DATABASE()` , иначе она вернет `NULL` . Это в основном относится к стандартному mysql-клиенту, поставляемому с сервером, поскольку позволяет подключаться без указания базы данных.

Запустите эту инструкцию SQL, чтобы получить все таблицы `MyISAM` в вашей базе данных.

Наконец, скопируйте вывод и выполните SQL-запросы от него.

Прочитайте Преобразование из MyISAM в InnoDB онлайн:
https://riptutorial.com/ru/mysql/topic/3135/преобразование-из-myisam-в-innodb

глава 54: ПРИСОЕДИНИТЕСЬ: Присоедините 3 таблицу с тем же именем id.

Examples

Присоедините 3 таблицы к столбцу с тем же именем

```
CREATE TABLE Table1 (
    id INT UNSIGNED NOT NULL,
    created_on DATE NOT NULL,
    PRIMARY KEY (id)
)
CREATE TABLE Table2 (
    id INT UNSIGNED NOT NULL,
    personName VARCHAR(255) NOT NULL,
    PRIMARY KEY (id)
)
CREATE TABLE Table3 (
    id INT UNSIGNED NOT NULL,
    accountName VARCHAR(255) NOT NULL,
    PRIMARY KEY (id)
)
```

после создания таблиц вы можете сделать запрос выбора, чтобы получить идентификатор всех трех таблиц, которые являются одинаковыми

```
SELECT
    t1.id AS table1Id,
    t2.id AS table2Id,
    t3.id AS table3Id
FROM Table1 t1
LEFT JOIN Table2 t2 ON t2.id = t1.id
LEFT JOIN Table3 t3 ON t3.id = t1.id
```

Прочитайте ПРИСОЕДИНИТЕСЬ: Присоедините 3 таблицу с тем же именем id. онлайн: https://riptutorial.com/ru/mysql/topic/9921/присоединитесь--присоедините-3-таблицу-с-тем-же-именем-id-

глава 55: присоединяется

Синтаксис

- `INNER` и `OUTER` игнорируются.
- `FULL` не реализован в MySQL.
- «commajoin» (`FROM a,b WHERE ax=by`) нахмурился; используйте `FROM a JOIN b ON ax=by` вместо.
- FROM JOIN b ON ax = by включает строки, которые соответствуют в обеих таблицах.
- FROM LEFT JOIN b ON ax = by включает все строки из `a` , плюс данные сопоставления из `b` или `NULLs` если нет соответствующей строки.

Examples

Примеры подключения

Запрос на создание таблицы на db

```
CREATE TABLE `user` (
`id` smallint(5) unsigned NOT NULL AUTO_INCREMENT,
`name` varchar(30) NOT NULL,
`course` smallint(5) unsigned DEFAULT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB;

CREATE TABLE `course` (
`id` smallint(5) unsigned NOT NULL AUTO_INCREMENT,
`name` varchar(50) NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

Поскольку мы используем таблицы InnoDB и знаем, что user.course и course.id связаны, мы можем указать отношение внешнего ключа:

```
ALTER TABLE `user`
ADD CONSTRAINT `FK_course`
FOREIGN KEY (`course`) REFERENCES `course` (`id`)
ON UPDATE CASCADE;
```

Присоединиться к запросу (Inner Join)

```
SELECT user.name, course.name
FROM `user`
INNER JOIN `course` on user.course = course.id;
```

JOIN с подзапросом (таблица «Производные»)

```
SELECT x, ...
    FROM ( SELECT y, ... FROM ... ) AS a
    JOIN tbl  ON tbl.x = a.y
    WHERE ...
```

Это будет оценивать подзапрос в временную таблицу, затем `JOIN` к `tbl` .

До 5.6 не может быть указателя в таблице temp. Таким образом, это было потенциально очень неэффективно:

```
SELECT ...
    FROM ( SELECT y, ... FROM ... ) AS a
    JOIN ( SELECT x, ... FROM ... ) AS b  ON b.x = a.y
    WHERE ...
```

С 5.6 оптимизатор определяет лучший индекс и создает его на лету. (У этого есть некоторые накладные расходы, поэтому он все еще не «совершенен».)

Еще одна распространенная парадигма заключается в том, чтобы иметь подзапрос, чтобы инициализировать что-то:

```
SELECT
        @n := @n + 1,
        ...
    FROM ( SELECT @n := 0 ) AS initialize
    JOIN the_real_table
    ORDER BY ...
```

(Примечание: это технически `CROSS JOIN` (декартово произведение), о чем свидетельствует отсутствие `ON` . Однако он эффективен, потому что в подзапросе возвращается только одна строка, которая должна быть сопоставлена с n строками в `the_real_table` .)

Извлечение клиентов с заказами - вариации на тему

Это получит все заказы для всех клиентов:

```
SELECT c.CustomerName, o.OrderID
    FROM Customers AS c
    INNER JOIN Orders AS o
        ON c.CustomerID = o.CustomerID
    ORDER BY c.CustomerName, o.OrderID;
```

Это будет подсчитывать количество заказов для каждого клиента:

```
SELECT c.CustomerName, COUNT(*) AS 'Order Count'
    FROM Customers AS c
    INNER JOIN Orders AS o
        ON c.CustomerID = o.CustomerID
```

```
    GROUP BY c.CustomerID;
    ORDER BY c.CustomerName;
```

Также, считается, но, вероятно, быстрее:

```
SELECT  c.CustomerName,
        ( SELECT COUNT(*) FROM Orders WHERE CustomerID = c.CustomerID ) AS 'Order Count'
    FROM Customers AS c
    ORDER BY c.CustomerName;
```

Перечислите только заказчиков с заказами.

```
SELECT  c.CustomerName,
    FROM Customers AS c
    WHERE EXISTS ( SELECT * FROM Orders WHERE CustomerID = c.CustomerID )
    ORDER BY c.CustomerName;
```

Полная внешняя связь

MySQL не поддерживает `FULL OUTER JOIN` , но есть способы их эмулировать.

Настройка данных

```
-- ----------------------------
-- Table structure for `owners`
-- ----------------------------
DROP TABLE IF EXISTS `owners`;
CREATE TABLE `owners` (
`owner_id` int(11) NOT NULL AUTO_INCREMENT,
`owner` varchar(30) DEFAULT NULL,
PRIMARY KEY (`owner_id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=latin1;

-- ----------------------------
-- Records of owners
-- ----------------------------
INSERT INTO `owners` VALUES ('1', 'Ben');
INSERT INTO `owners` VALUES ('2', 'Jim');
INSERT INTO `owners` VALUES ('3', 'Harry');
INSERT INTO `owners` VALUES ('6', 'John');
INSERT INTO `owners` VALUES ('9', 'Ellie');
-- ----------------------------
-- Table structure for `tools`
-- ----------------------------
DROP TABLE IF EXISTS `tools`;
CREATE TABLE `tools` (
`tool_id` int(11) NOT NULL AUTO_INCREMENT,
`tool` varchar(30) DEFAULT NULL,
`owner_id` int(11) DEFAULT NULL,
PRIMARY KEY (`tool_id`)
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=latin1;
-- ----------------------------
-- Records of tools
-- ----------------------------
INSERT INTO `tools` VALUES ('1', 'Hammer', '9');
```

```
INSERT INTO `tools` VALUES ('2', 'Pliers', '1');
INSERT INTO `tools` VALUES ('3', 'Knife', '1');
INSERT INTO `tools` VALUES ('4', 'Chisel', '2');
INSERT INTO `tools` VALUES ('5', 'Hacksaw', '1');
INSERT INTO `tools` VALUES ('6', 'Level', null);
INSERT INTO `tools` VALUES ('7', 'Wrench', null);
INSERT INTO `tools` VALUES ('8', 'Tape Measure', '9');
INSERT INTO `tools` VALUES ('9', 'Screwdriver', null);
INSERT INTO `tools` VALUES ('10', 'Clamp', null);
```

Что мы хотим видеть?

Мы хотим получить список, в котором мы видим, кто владеет теми инструментами и какие инструменты могут не иметь владельца.

Запросы

Для этого мы можем объединить два запроса с помощью `UNION` . В этом первом запросе мы присоединяемся к инструментам владельцев, используя `LEFT JOIN` . Это добавит всех наших владельцев в наш набор результатов, неважно, действительно ли они владеют инструментами.

Во втором запросе мы используем `RIGHT JOIN` для присоединения инструментов к владельцам. Таким образом, нам удастся получить все инструменты в нашем наборе результатов, если они не принадлежат никому, их столбец владельца просто будет содержать `NULL` . Добавляя `WHERE` -clause, который фильтрует `owners.owner_id IS NULL` мы определяем результат как те наборы данных, которые еще не были возвращены первым запросом, поскольку мы только ищем данные в правой объединенной таблице.

Поскольку мы используем `UNION ALL` набор результатов второго запроса будет привязан к первому запросу.

```
SELECT `owners`.`owner`, tools.tool
FROM `owners`
LEFT JOIN `tools` ON `owners`.`owner_id` = `tools`.`owner_id`
UNION ALL
SELECT `owners`.`owner`, tools.tool
FROM `owners`
RIGHT JOIN `tools` ON `owners`.`owner_id` = `tools`.`owner_id`
WHERE `owners`.`owner_id` IS NULL;

+-------+--------------+
| owner | tool         |
+-------+--------------+
| Ben   | Pliers       |
| Ben   | Knife        |
| Ben   | Hacksaw      |
| Jim   | Chisel       |
| Harry | NULL         |
| John  | NULL         |
| Ellie | Hammer       |
| Ellie | Tape Measure |
| NULL  | Level        |
```

```
| NULL  | Wrench       |
| NULL  | Screwdriver  |
| NULL  | Clamp        |
+-------+--------------+
12 rows in set (0.00 sec)
```

Внутреннее соединение для 3 таблиц

предположим, у нас есть три таблицы, которые можно использовать для простого сайта с тегами.

- Кулачный стол предназначен для сообщений.
- Второе для тегов
- Третий для отношений Tags & Post

кулачный стол "видеоигра"

Я бы	заглавие	reg_date	содержание
1	BioShock Infinite	2016-08-08	....

таблица «тегов»

Я бы	название
1	Йеннифэр из Венгерберга
2	Элизабет

Таблица "tags_meta"

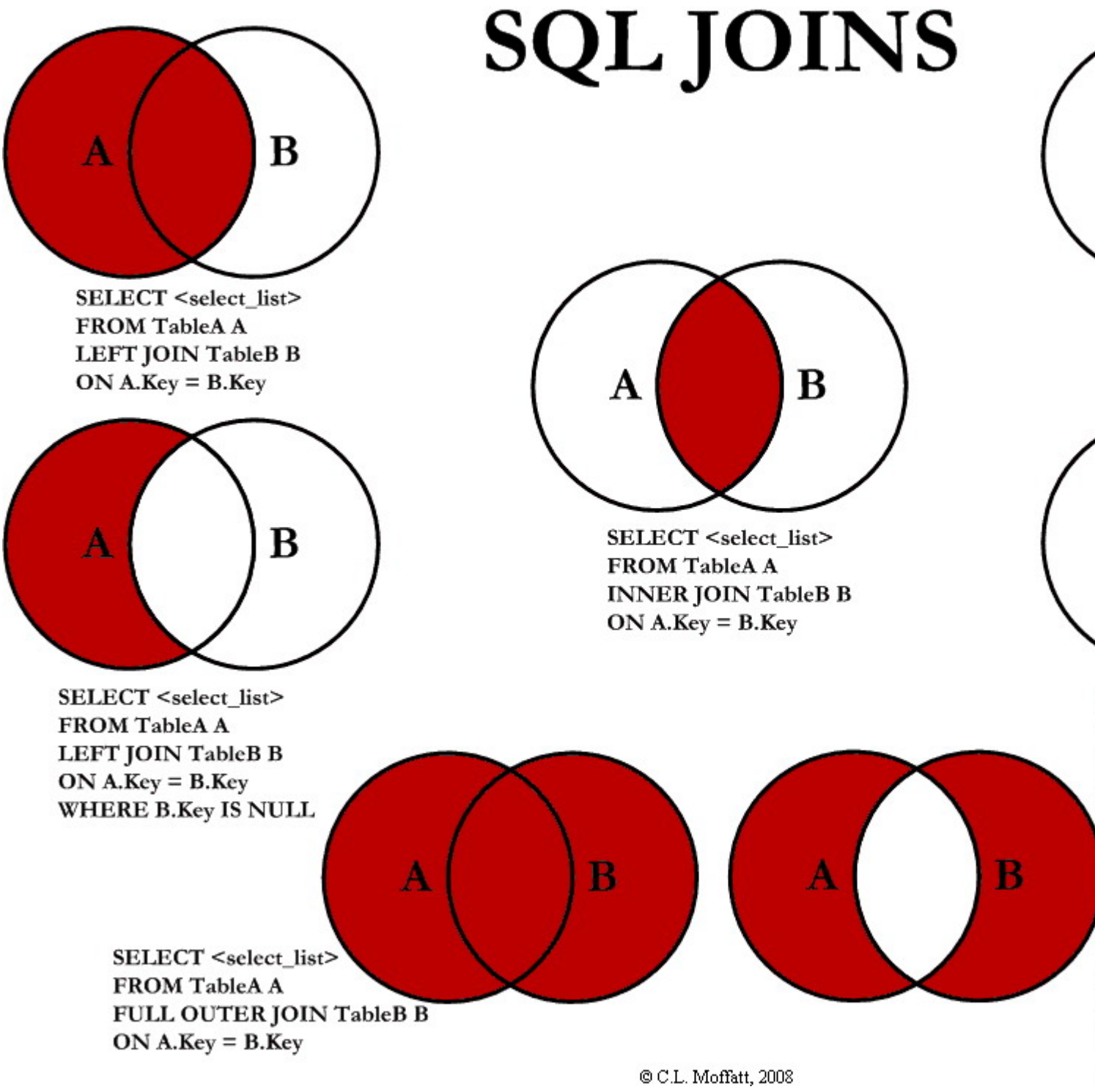
post_id	tag_id
1	2

```
SELECT videogame.id,
    videogame.title,
    videogame.reg_date,
    tags.name,
    tags_meta.post_id
FROM tags_meta
INNER JOIN videogame ON videogame.id = tags_meta.post_id
INNER JOIN tags ON tags.id = tags_meta.tag_id
WHERE tags.name = "elizabeth"
ORDER BY videogame.reg_date
```

этот код может возвращать все сообщения, связанные с этим тегом "#elizabeth"

Соединения визуализируются

Если вы визуально ориентированный человек, эта диаграмма Венна может помочь вам понять разные типы `JOIN` которые существуют в MySQL.



Прочитайте присоединяется онлайн: https://riptutorial.com/ru/mysql/topic/2736/присоединяется

глава 56: Работа с разреженными или отсутствующими данными

Examples

Работа со столбцами, содержащими значения NULL

В MySQL и других диалектах SQL значения `NULL` имеют специальные свойства.

Рассмотрим следующую таблицу, в которой указаны кандидаты на работу, компании, в которых они работали, и дату, когда они покинули компанию. `NULL` указывает, что заявитель все еще работает в компании:

```
CREATE TABLE example
(`applicant_id` INT, `company_name` VARCHAR(255), `end_date` DATE);

+--------------+-----------------+------------+
| applicant_id | company_name    | end_date   |
+--------------+-----------------+------------+
|            1 | Google          | NULL       |
|            1 | Initech         | 2013-01-31 |
|            2 | Woodworking.com | 2016-08-25 |
|            2 | NY Times        | 2013-11-10 |
|            3 | NFL.com         | 2014-04-13 |
+--------------+-----------------+------------+
```

Ваша задача состоит в том, чтобы составить запрос, который возвращает все строки после `2016-01-01` , включая всех сотрудников, которые все еще работают в компании (с датами окончания `NULL`). Этот оператор select:

```
SELECT * FROM example WHERE end_date > '2016-01-01';
```

не содержит никаких строк со значениями `NULL` :

```
+--------------+-----------------+------------+
| applicant_id | company_name    | end_date   |
+--------------+-----------------+------------+
|            2 | Woodworking.com | 2016-08-25 |
+--------------+-----------------+------------+
```

В документации по MySQL сравнения с использованием арифметических операторов <,>, = и <> сами возвращают `NULL` вместо логического `TRUE` или `FALSE` . Таким образом, строка с `NULL` end_date не превышает 2016-01-01 или не меньше 2016-01-01.

Это можно решить, используя ключевые слова IS NULL:

```
SELECT * FROM example WHERE end_date > '2016-01-01' OR end_date IS NULL;

+--------------+-----------------+------------+
| applicant_id | company_name    | end_date   |
+--------------+-----------------+------------+
|            1 | Google          | NULL       |
|            2 | Woodworking.com | 2016-08-25 |
+--------------+-----------------+------------+
```

Работа с NULL становится более сложной, когда задача включает в себя функции агрегации, такие как `MAX()` и предложение `GROUP BY` . Если ваша задача состояла в том, чтобы выбрать самую последнюю использованную дату для каждого заявителя, следующий запрос может показаться логической первой попыткой:

```
SELECT applicant_id, MAX(end_date) FROM example GROUP BY applicant_id;

+--------------+---------------+
| applicant_id | MAX(end_date) |
+--------------+---------------+
|            1 | 2013-01-31    |
|            2 | 2016-08-25    |
|            3 | 2014-04-13    |
+--------------+---------------+
```

Однако, зная, что `NULL` указывает, что заявитель по-прежнему работает в компании, первая строка результата является неточной. Использование `CASE WHEN` обеспечивает обходной путь для проблемы `NULL` :

```
 SELECT
    applicant_id,
    CASE WHEN MAX(end_date is null) = 1 THEN 'present' ELSE MAX(end_date) END
    max_date
FROM example
GROUP BY applicant_id;

+--------------+------------+
| applicant_id | max_date   |
+--------------+------------+
|            1 | present    |
|            2 | 2016-08-25 |
|            3 | 2014-04-13 |
+--------------+------------+
```

Этот результат можно соединить обратно к исходному `example` таблице , чтобы определить компанию , в которой последний работал заявитель:

```
SELECT
  data.applicant_id,
  data.company_name,
  data.max_date
FROM (
  SELECT
    *,
    CASE WHEN end_date is null THEN 'present' ELSE end_date END max_date
```

```
  FROM example
) data
INNER JOIN (
 SELECT
   applicant_id,
   CASE WHEN MAX(end_date is null) = 1 THEN 'present' ELSE MAX(end_date) END max_date
 FROM
   example
 GROUP BY applicant_id
) j
ON data.applicant_id = j.applicant_id AND data.max_date = j.max_date;

+--------------+-----------------+------------+
| applicant_id | company_name    | max_date   |
+--------------+-----------------+------------+
|            1 | Google          | present    |
|            2 | Woodworking.com | 2016-08-25 |
|            3 | NFL.com         | 2014-04-13 |
+--------------+-----------------+------------+
```

Это всего лишь несколько примеров работы со значениями `NULL` в MySQL.

Прочитайте Работа с разреженными или отсутствующими данными онлайн: https://riptutorial.com/ru/mysql/topic/5866/работа-с-разреженными-или-отсутствующими-данными

глава 57: Разметка

замечания

- **RANGE** . Этот тип разбиения присваивает строки разбиениям на основе значений столбцов, попадающих в заданный диапазон.
- **LIST** . Подобно разделу с помощью RANGE, за исключением того, что раздел выбирается на основе столбцов, соответствующих одному из множества дискретных значений.
- **Хэш-разделение** . При таком типе разбиения раздел выбирается на основе значения, возвращаемого определяемым пользователем выражением, которое работает со значениями столбцов в строках, которые должны быть вставлены в таблицу. Функция может состоять из любого выражения, действительного в MySQL, которое дает неотрицательное целочисленное значение. Также доступно расширение для этого типа, `LINEAR HASH` .
- **KEY** . Этот тип разбиения похож на разбиение на разделы с помощью HASH, за исключением того, что предоставляется только один или несколько столбцов, которые будут оцениваться, а сервер MySQL предоставляет собственную функцию хэширования. Эти столбцы могут содержать не целочисленные значения, так как функция хэширования, предоставленная MySQL, гарантирует целочисленный результат независимо от типа данных столбца. Также доступно расширение для этого типа, `LINEAR KEY` .

Examples

Разделение разделов

Таблица, разбитая по диапазону, разбивается на разделы таким образом, что каждый раздел содержит строки, для которых значение выражения секционирования находится в заданном диапазоне. Диапазоны должны быть смежными, но не перекрывающимися, и определяются с помощью оператора `VALUES LESS THAN` . Для следующих нескольких примеров предположим, что вы создаете таблицу, такую как нижеследующая, для хранения записей персонала для цепочки из 20 хранилищ видео, пронумерованных от 1 до 20:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
```

```
    store_id INT NOT NULL
);
```

В зависимости от ваших потребностей эта таблица может быть разделена по диапазону несколькими способами. Один из способов - использовать столбец `store_id` . Например, вы можете решить разбить таблицу 4 путем добавления предложения `PARTITION BY RANGE` , как показано ниже:

```
ALTER TABLE employees PARTITION BY RANGE (store_id) (
    PARTITION p0 VALUES LESS THAN (6),
    PARTITION p1 VALUES LESS THAN (11),
    PARTITION p2 VALUES LESS THAN (16),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

> `MAXVALUE` представляет собой целочисленное значение, которое всегда больше, чем наибольшее возможное целочисленное значение (в математическом языке оно служит наименьшей верхней границей).

основанный на официальном документе MySQL .

Разделение списка

Пересечение списков во многом сходно с разбиением по диапазонам. Как и в разделении по RANGE, каждый раздел должен быть явно определен. Основное различие между двумя типами разбиения состоит в том, что при разбиении списка каждый раздел определяется и выбирается на основе членства значения столбца в одном из наборов списков значений, а не в одном из множества смежных диапазонов ценности. Это делается с помощью `PARTITION BY LIST(expr)` где `expr` - значение столбца или выражение на основе значения столбца и возвращающее целочисленное значение, а затем определение каждого раздела с помощью `VALUES IN (value_list)` , где `value_list` является список целых чисел, разделенных запятыми.

В следующих примерах мы предполагаем, что базовое определение таблицы, которая будет разбита на разделы, предоставляется приведенным здесь инструкцией `CREATE TABLE` :

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
);
```

Предположим, что имеется 20 магазинов видео, распределенных между 4 франшизами, как показано в следующей таблице.

Область, край	Идентификационные номера магазинов
к северу	3, 5, 6, 9, 17
восток	1, 2, 10, 11, 19, 20
запад	4, 12, 13, 14, 18
центральный	7, 8, 15, 16

Чтобы разбить эту таблицу таким образом, чтобы строки для хранилищ, принадлежащих к одному региону, были сохранены в одном разделе

```
ALTER TABLE employees PARTITION BY LIST(store_id) (
    PARTITION pNorth VALUES IN (3,5,6,9,17),
    PARTITION pEast VALUES IN (1,2,10,11,19,20),
    PARTITION pWest VALUES IN (4,12,13,14,18),
    PARTITION pCentral VALUES IN (7,8,15,16)
);
```

основанный на официальном документе MySQL .

Разделение HASH

Разделение по HASH используется в первую очередь для обеспечения равномерного распределения данных между предопределенным количеством разделов. С диапазоном или разбиением списка вы должны явно указывать, в какой раздел необходимо сохранить заданное значение столбца или набор значений столбцов; с хэш-разбиением, MySQL позаботится об этом для вас, и вам нужно указать только значение столбца или выражение на основе значения столбца, которое должно быть хэшировано, и количества разделов, в которые разделена разделенная таблица.

Следующий оператор создает таблицу, которая использует хэширование в столбце store_id и делится на 4 раздела:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY HASH(store_id)
PARTITIONS 4;
```

> Если вы не включаете предложение `PARTITIONS` , количество разделов по умолчанию равно 1.

основанный на официальном документе MySQL .

Прочитайте Разметка онлайн: https://riptutorial.com/ru/mysql/topic/5128/разметка

глава 58: Регулярные выражения

Вступление

Регулярное выражение является мощным способом задания шаблона для сложного поиска.

Examples

REGEXP / RLIKE

Оператор `REGEXP` (или его синоним, `RLIKE`) позволяет сопоставлять шаблоны на основе регулярных выражений.

Рассмотрим следующую таблицу `employee` :

```
+-------------+-------------+-------------+--------------+----------+
| EMPLOYEE_ID | FIRST_NAME  | LAST_NAME   | PHONE_NUMBER | SALARY   |
+-------------+-------------+-------------+--------------+----------+
|         100 | Steven      | King        | 515.123.4567 | 24000.00 |
|         101 | Neena       | Kochhar     | 515.123.4568 | 17000.00 |
|         102 | Lex         | De Haan     | 515.123.4569 | 17000.00 |
|         103 | Alexander   | Hunold      | 590.423.4567 |  9000.00 |
|         104 | Bruce       | Ernst       | 590.423.4568 |  6000.00 |
|         105 | David       | Austin      | 590.423.4569 |  4800.00 |
|         106 | Valli       | Pataballa   | 590.423.4560 |  4800.00 |
|         107 | Diana       | Lorentz     | 590.423.5567 |  4200.00 |
|         108 | Nancy       | Greenberg   | 515.124.4569 | 12000.00 |
|         109 | Daniel      | Faviet      | 515.124.4169 |  9000.00 |
|         110 | John        | Chen        | 515.124.4269 |  8200.00 |
+-------------+-------------+-------------+--------------+----------+
```

Образец ^

Выберите всех сотрудников, чье `FIRST_NAME` начинается с **N.**

запрос

```
SELECT * FROM employees WHERE FIRST_NAME REGEXP '^N'
-- Pattern start with----------------------------^
```

Шаблон $ **

Выберите всех сотрудников, чей `PHONE_NUMBER` заканчивается на **4569** .

запрос

```
SELECT * FROM employees WHERE PHONE_NUMBER REGEXP '4569$'
-- Pattern end with---------------------------------^
```

НЕ REGEXP

Выберите всех сотрудников, `FIRST_NAME` которых *не* начинается с **N.**

запрос

```
SELECT * FROM employees WHERE FIRST_NAME NOT REGEXP '^N'
-- Pattern does not start with---------------^
```

Regex Contain

Выбрать всех сотрудников , чьи `LAST_NAME` содержится **в** и чьи `FIRST_NAME` содержит `.a`

запрос

```
SELECT * FROM employees WHERE FIRST_NAME REGEXP 'a' AND LAST_NAME REGEXP 'in'
-- No ^ or $, pattern can be anywhere ------------------------------------^
```

Любой символ между []

Выберите всех сотрудников, чье `FIRST_NAME` начинается с **A** или **B** или **C.**

запрос

```
SELECT * FROM employees WHERE FIRST_NAME REGEXP '^[ABC]'
-------------------------------------------------^^---^
```

Образец или |

Выберите всех сотрудников, чье `FIRST_NAME` начинается с **A** или **B** или **C** и заканчивается на **r** , **e** или **i** .

запрос

```
SELECT * FROM employees WHERE FIRST_NAME REGEXP '^[ABC]|[rei]$'
-- ------------------------------------------------^^---^^^---^^
```

Подсчет соответствия регулярных выражений

Рассмотрим следующий запрос:

```
SELECT FIRST_NAME, FIRST_NAME REGEXP '^N' as matching FROM employees
```

`FIRST_NAME REGEXP '^N'` равно *1* или *0 в* зависимости от того, что `FIRST_NAME` соответствует `^N`

Чтобы визуализировать это лучше:

```
SELECT
FIRST_NAME,
IF(FIRST_NAME REGEXP '^N', 'matches ^N', 'does not match ^N') as matching
FROM employees
```

Наконец, подсчитайте общее количество совпадающих и несоответствующих строк с помощью:

```
SELECT
IF(FIRST_NAME REGEXP '^N', 'matches ^N', 'does not match ^N') as matching,
COUNT(*)
FROM employees
GROUP BY matching
```

Прочитайте Регулярные выражения онлайн: https://riptutorial.com/ru/mysql/topic/9444/регулярные-выражения

глава 59: Резервное копирование с помощью mysqldump

Синтаксис

- mysqldump -u [имя_пользователя] -p [пароль] [другие параметры] db_name> dumpFileName.sql /// для резервного копирования одной базы данных
- mysqldump -u [имя_пользователя] -p [пароль] [другие параметры] db_name [tbl_name1 tbl_name2 tbl_name2 ...]> dumpFileName.sql /// для резервного копирования одной или нескольких таблиц
- mysqldump -u [имя_пользователя] -p [пароль] [другие параметры] --databases db_name1 db_name2 db_name3 ...> dumpFileName.sql /// Резервное копирование одной или нескольких полных баз данных
- mysqldump -u [имя_пользователя] -p [пароль] [другие параметры] --all-databases> dumpFileName.sql /// Резервное копирование всего сервера MySQL

параметры

вариант	эффект
-	# Параметры входа в сервер
`-h` (`--host`)	Хост (IP-адрес или имя хоста) для подключения. По умолчанию `localhost` (`127.0.0.1`) Пример: `-h localhost`
`-u` (`--user`)	Пользователь MySQL
`-p` (`--password`)	Пароль MySQL. **Важно** . При использовании `-p` между опцией и паролем не должно быть пробелов. Пример: `-pMyPassword`
-	# Параметры дампа
`--add-drop-database`	Добавьте инструкцию `DROP DATABASE` перед каждым оператором `CREATE DATABASE` . Полезно, если вы хотите заменить базы данных на сервере.
`--add-drop-table`	Добавьте инструкцию `DROP TABLE` перед каждым оператором `CREATE TABLE` . Полезно, если вы хотите заменить таблицы на сервере.
`--no-create-db`	`CREATE DATABASE` инструкции `CREATE DATABASE` в дампе. Это полезно, если вы уверены, что базы данных, которые вы сбросили, уже существуют на сервере, где вы загрузите дамп.

вариант	эффект
`-t` (`--no-create-info`)	Запретите все инструкции `CREATE TABLE` в дампе. Это полезно, если вы хотите сбросить только данные из таблиц и будете использовать файл дампа для заполнения идентичных таблиц в другой базе данных / сервере.
`-d` (`--no-data`)	Не записывайте данные таблицы. Это приведет к сбросу только операторов `CREATE TABLE` . Полезно для создания баз данных «шаблонов»
`-R` (`--routines`)	Включите хранимые процедуры / функции в дампе.
`-K` (`--disable-keys`)	Отключайте ключи для каждой таблицы перед вставкой данных и включайте ключи после ввода данных. Это ускоряет вставки только в таблицах MyISAM с неистинными индексами.

замечания

Результат операции `mysqldump` - это слегка комментируемый файл, содержащий последовательные операторы SQL, которые совместимы с версией утилит MySQL, которые были использованы для ее создания (с уделением особого внимания совместимости с предыдущими версиями, но без гарантии для будущих). Таким образом, восстановление базы данных `mysqldump` ed включает выполнение этих операторов. Как правило, этот файл

- `DROP` s первая указанная таблица или представление
- `CREATE` что таблица или представление
- Для таблиц, сбрасываемых данными (т. `--no-data` Без опции `--no-data`)
    - `LOCK` s стол
    - `INSERT` все строки из исходной таблицы в одном выражении
- `UNLOCK TABLES`
- Повторяет вышеуказанное для всех других таблиц и представлений
- `DROP` s первая включенная рутина
- `CREATE` с этой программой
- Повторяет то же самое для всех других процедур

Наличие `DROP` перед `CREATE` для каждой таблицы означает, что если присутствует схема, независимо от того, пуста она или нет, использование файла `mysqldump` для его восстановления будет заполнять или перезаписывать данные в нем.

Examples

Создание резервной копии базы данных или таблицы

Создайте моментальный снимок всей базы данных:

```
mysqldump [options] db_name > filename.sql
```

Создайте моментальный снимок нескольких баз данных:

```
mysqldump [options] --databases db_name1 db_name2 ... > filename.sql
mysqldump [options] --all-databases > filename.sql
```

Создайте снимок одной или нескольких таблиц:

```
mysqldump [options] db_name table_name... > filename.sql
```

Создайте моментальный снимок, *исключая* одну или несколько таблиц:

```
mysqldump [options] db_name --ignore-table=tbl1 --ignore-table=tbl2 ... > filename.sql
```

Расширение файла `.sql` полностью `.sql` от стиля. Любое расширение будет работать.

Указание имени пользователя и пароля

```
> mysqldump -u username -p [other options]
Enter password:
```

Если вам нужно указать пароль в командной строке (например, в скрипте), вы можете добавить его после опции `-p` *без* пробела:

```
> mysqldump -u username -ppassword [other options]
```

Если пароль содержит пробелы или специальные символы, не забудьте использовать экранирование в зависимости от вашей оболочки / системы.

Необязательно расширенная форма:

```
> mysqldump --user=username --password=password [other options]
```

(Объяснение, указывающее пароль в командной строке, не рекомендуется из-за проблем с безопасностью.)

Восстановление резервной копии базы данных или таблицы

```
mysql [options] db_name < filename.sql
```

Обратите внимание, что:

- `db_name` должно быть существующей базой данных;

- ваш аутентифицированный пользователь имеет достаточные привилегии для выполнения всех команд внутри вашего `filename.sql` ;
- Расширение файла `.sql` полностью `.sql` от стиля. Любое расширение будет работать.
- Вы не можете указать имя таблицы для загрузки, даже если вы можете указать, с какого дампа. Это должно быть сделано в `filename.sql` .

В качестве альтернативы, когда в **инструменте командной строки MySQL** вы можете восстановить (или запустить любой другой скрипт) с помощью команды source:

```
source filename.sql
```

или же

```
\. filename.sql
```

mysqldump с удаленного сервера с сжатием

Чтобы использовать сжатие по `--compress` для более быстрой передачи, передайте параметр `--compress` для `mysqldump` . Пример:

```
mysqldump -h db.example.com -u username -p --compress dbname > dbname.sql
```

Важно: если вы не хотите блокировать *источник* db, вы должны также включить `--lock-tables=false` . Но вы не можете получить внутренне согласованное изображение db таким образом.

Чтобы сохранить сжатый файл, вы можете подключиться к `gzip` .

```
mysqldump -h db.example.com -u username -p --compress dbname | gzip --stdout > dbname.sql.gz
```

восстановить gzipped файл mysqldump без разжатия

```
gunzip -c dbname.sql.gz | mysql dbname -u username -p
```

Примечание: `-c` означает запись вывода в stdout.

Резервное копирование непосредственно на Amazon S3 со сжатием

Если вы хотите сделать полную резервную копию большой установки MySql и не располагаете достаточным местным хранилищем, вы можете сбросить и сжать ее непосредственно в ведро Amazon S3. Это также хорошая практика для этого, не имея пароля БД в качестве части команды:

```
mysqldump -u root -p --host=localhost --opt --skip-lock-tables --single-transaction \
```

```
        --verbose --hex-blob --routines --triggers --all-databases |
    gzip -9 | s3cmd put - s3://s3-bucket/db-server-name.sql.gz
```

Вам будет предложено ввести пароль, после чего начнется резервное копирование.

Транслирование данных с одного сервера MySQL на другой

Если вам нужно скопировать базу данных с одного сервера на другой, у вас есть два варианта:

Опция 1:

1. Храните файл дампа на исходном сервере
2. Скопируйте файл дампа на целевой сервер
3. Загрузите файл дампа на целевой сервер

На исходном сервере:

```
mysqldump [options] > dump.sql
```

На целевом сервере скопируйте файл дампа и выполните:

```
mysql [options] < dump.sql
```

Вариант 2:

Если целевой сервер может подключиться к хост-серверу, вы можете использовать конвейер для копирования базы данных с одного сервера на другой:

На целевом сервере

```
mysqldump [options to connect to the source server] | mysql [options]
```

Аналогично, сценарий можно запустить на исходном сервере, нажав на пункт назначения. В любом случае это, вероятно, будет значительно быстрее, чем Вариант 1.

Резервная база данных с хранимыми процедурами и функциями

По умолчанию хранимые процедуры и функции или не сгенерированные `mysqldump` , вам нужно будет добавить параметр `--routines` (или `-R`):

```
mysqldump -u username -p -R db_name > dump.sql
```

При использовании `--routines` создания и изменения времени не поддерживаются, вместо этого вы должны сбросить и перезагрузить содержимое `mysql.proc` .

Прочитайте Резервное копирование с помощью mysqldump онлайн:
https://riptutorial.com/ru/mysql/topic/604/резервное-копирование-с-помощью-mysqldump

глава 60: Сводные запросы

замечания

Создание сводных запросов в MySQL зависит от функции `GROUP_CONCAT()` . Если ожидается, что результат выражения, который создает столбцы сводного запроса, будет большим, значение переменной `group_concat_max_len` должно быть увеличено:

```
set session group_concat_max_len = 1024 * 1024; -- This should be enough for most cases
```

Examples

Создание сводного запроса

MySQL не обеспечивает встроенный способ создания сводных запросов. Однако они могут быть созданы с использованием подготовленных операторов.

Предположим, что таблица `tbl_values` :

Я бы	название	группа	Значение
1	Пит		10
2	Пит	B	20
3	Джон		10

Запрос: создать запрос, который показывает сумму `Value` для каждого `Name` ; `Group` должна быть заголовком столбца, а `Name` должно быть заголовком строки.

```
-- 1. Create an expression that builds the columns
set @sql = (
    select group_concat(distinct
        concat(
            "sum(case when `Group`='", Group, "' then `Value` end) as `", `Group`, "`"
        )
    )
    from tbl_values
);

-- 2. Complete the SQL instruction
set @sql = concat("select Name, ", @sql, " from tbl_values group by `Name`");

-- 3. Create a prepared statement
prepare stmt from @sql;

-- 4. Execute the prepared statement
```

```
execute stmt;
```

Результат:

название		В
Джон	10	НОЛЬ
Пит	10	20

Важно: освободите подготовленное заявление, если оно больше не требуется:

```
deallocate prepare stmt;
```

Пример скрипта SQL

Прочитайте Сводные запросы онлайн: https://riptutorial.com/ru/mysql/topic/3074/сводные-запросы

глава 61: Сделка

Examples

Начать транзакцию

Транзакция представляет собой последовательную группу операторов SQL, таких как выбор, вставка, обновление или удаление, которая выполняется как один рабочий блок.

Другими словами, транзакция никогда не будет полной, если каждая отдельная операция внутри группы не будет успешной. Если какая-либо операция в транзакции завершится с ошибкой, вся транзакция завершится неудачно.

Банковская сделка будет лучшим примером для объяснения этого. Рассмотрите возможность перехода между двумя учетными записями. Для этого вам нужно написать инструкции SQL, которые выполняют следующие

1. Проверьте доступность запрашиваемой суммы на первой учетной записи
2. Вычитайте запрашиваемую сумму с первой учетной записи
3. Внести его во вторую учетную запись

Если кто-то этот процесс терпит неудачу, целое должно быть возвращено в прежнее состояние.

ACID: свойства транзакций

Транзакции имеют следующие четыре стандартных свойства

- **Атомарность:** обеспечивает успешное завершение всех операций в рабочей единице; в противном случае транзакция прерывается в момент сбоя, а предыдущие операции возвращаются в прежнее состояние.
- **Согласованность:** гарантирует, что база данных правильно изменяет состояния при успешной транзакции.
- **Изоляция:** позволяет транзакциям работать независимо друг от друга и прозрачно.
- **Долговечность:** гарантирует, что результат или эффект совершенной транзакции сохранится в случае сбоя системы.

Транзакции начинаются с заявления `START TRANSACTION` или `BEGIN WORK` и заканчиваются либо заявлением `COMMIT` либо `ROLLBACK` . Команды SQL между инструкциями начала и окончания составляют основную часть транзакции.

```
START TRANSACTION;
SET @transAmt = '500';
SELECT @availableAmt:=ledgerAmt FROM accTable WHERE customerId=1 FOR UPDATE;
UPDATE accTable SET ledgerAmt=ledgerAmt-@transAmt WHERE customerId=1;
```

```
UPDATE accTable SET ledgerAmt=ledgerAmt+@transAmt WHERE customerId=2;
COMMIT;
```

С помощью `START TRANSACTION` автокоммит остается отключенным до тех пор, пока вы не закончите транзакцию с помощью `COMMIT` или `ROLLBACK` . Затем режим автосохранения возвращается в прежнее состояние.

`FOR UPDATE` указывает (и блокирует) строку (строки) на время транзакции.

Хотя транзакция остается незафиксированной, эта транзакция будет недоступна для других пользователей.

Общие процедуры, связанные с транзакцией

- Начните транзакцию, выпустив команду SQL `BEGIN WORK` или `START TRANSACTION` .
- Запустите все ваши SQL-операторы.
- Проверьте, все ли выполнено в соответствии с вашим требованием.
- Если да, то выполните команду `COMMIT` , иначе выполните команду `ROLLBACK` чтобы вернуть все в предыдущее состояние.
- Проверьте ошибки даже после `COMMIT` если вы используете или можете в конечном итоге использовать Galera / PXC.

COMMIT, ROLLBACK и AUTOCOMMIT

AUTOCOMMIT

MySQL автоматически фиксирует операторы, которые не являются частью транзакции. Результаты любых `UPDATE` , `DELETE` или `INSERT` не предшествующих `BEGIN` или `START TRANSACTION` , сразу будут видны всем соединениям.

По `AUTOCOMMIT` переменной `AUTOCOMMIT` установлено значение *true* . Это можно изменить следующим образом,

```
--->To make autcommit false
SET AUTOCOMMIT=false;
--or
SET AUTOCOMMIT=0;

--->To make autcommit true
SET AUTOCOMMIT=true;
--or
SET AUTOCOMMIT=1;
```

Просмотр состояния `AUTOCOMMIT`

```
SELECT @@autocommit;
```

COMMIT

Если `AUTOCOMMIT` установлен в значение false и транзакция не выполнена, изменения будут видны только для текущего соединения.

После того, как инструкция `COMMIT` фиксирует изменения в таблице, результат будет видимым для всех подключений.

Мы рассматриваем две связи, чтобы объяснить это

Соединение 1

```
--->Before making autocommit false one row added in a new table
mysql> INSERT INTO testTable VALUES (1);

--->Making autocommit = false
mysql> SET autocommit=0;

mysql> INSERT INTO testTable VALUES (2), (3);
mysql> SELECT * FROM testTable;
+-----+
| tId |
+-----+
|   1 |
|   2 |
|   3 |
+-----+
```

Соединение 2

```
mysql> SELECT * FROM testTable;
+-----+
| tId |
+-----+
|   1 |
+-----+
---> Row inserted before autocommit=false only visible here
```

Соединение 1

```
mysql> COMMIT;
--->Now COMMIT is executed in connection 1
mysql> SELECT * FROM testTable;
    +-----+
    | tId |
    +-----+
    |   1 |
    |   2 |
    |   3 |
    +-----+
```

Соединение 2

```
mysql> SELECT * FROM testTable;
    +-----+
    | tId |
```

```
    +-----+
    |   1 |
    |   2 |
    |   3 |
    +-----+
--->Now all the three rows are visible here
```

ROLLBACK

Если что-то пошло не так в выполнении запроса, `ROLLBACK` используется для отмены изменений. См. Объяснение ниже

```
--->Before making autocommit false one row added in a new table
mysql> INSERT INTO testTable VALUES (1);

--->Making autocommit = false
mysql> SET autocommit=0;

mysql> INSERT INTO testTable VALUES (2), (3);
mysql> SELECT * FROM testTable;
+-----+
| tId |
+-----+
|   1 |
|   2 |
|   3 |
+-----+
```

Теперь мы выполняем `ROLLBACK`

```
--->Rollback executed now
mysql> ROLLBACk;

mysql> SELECT * FROM testTable;
+-----+
| tId |
+-----+
|   1 |
+-----+
--->Rollback removed all rows which all are not committed
```

После выполнения `COMMIT` , `ROLLBACK` не вызовет ничего

```
mysql> INSERT INTO testTable VALUES (2), (3);
mysql> SELECT * FROM testTable;
mysql> COMMIT;
+-----+
| tId |
+-----+
|   1 |
|   2 |
|   3 |
+-----+

--->Rollback executed now
mysql> ROLLBACk;
```

```
mysql> SELECT * FROM testTable;
+-----+
| tId |
+-----+
|   1 |
|   2 |
|   3 |
+-----+
--->Rollback not removed any rows
```

Если значение `AUTOCOMMIT` установлено *верно* , то `COMMIT` и `ROLLBACK` бесполезны

Транзакция с использованием драйвера JDBC

Транзакция с использованием драйвера JDBC используется для контроля того, как и когда транзакция должна совершать транзакции и откатываться. Подключение к серверу MySQL создается с использованием драйвера JDBC

Драйвер JDBC для MySQL можно скачать здесь

Давайте начнем с подключения к базе данных с помощью драйвера JDBC

```
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection(DB_CONNECTION_URL,DB_USER,USER_PASSWORD);
--->Example for connection url "jdbc:mysql://localhost:3306/testDB");
```

Символьные наборы : это указывает, какой набор символов будет использоваться клиентом для отправки операторов SQL на сервер. Он также указывает набор символов, который сервер должен использовать для отправки результатов обратно клиенту.

Это следует упомянуть при создании соединения с сервером. Таким образом, строка подключения должна быть похожа,

```
jdbc:mysql://localhost:3306/testDB?useUnicode=true&amp;characterEncoding=utf8
```

См. Это для более подробной информации о наборах символов и коллаборациях

Когда вы открываете соединение, для режима `AUTOCOMMIT` по умолчанию установлено значение *true* , которое должно быть изменено на значение *false,* чтобы начать транзакцию.

```
con.setAutoCommit(false);
```

Вы всегда должны вызывать `setAutoCommit()` сразу после открытия соединения.

В противном случае используйте `START TRANSACTION` или `BEGIN WORK` чтобы начать новую транзакцию. Используя `START TRANSACTION` или `BEGIN WORK` , не нужно менять `AUTOCOMMIT` *false* . Это автоматически отключится.

Теперь вы можете начать транзакцию. См. Полный пример транзакции JDBC ниже.

```
package jdbcTest;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;


public class accTrans {

    public static void doTransfer(double transAmount,int customerIdFrom,int customerIdTo) {

        Connection con = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;

        try {
            String DB_CONNECTION_URL =
"jdbc:mysql://localhost:3306/testDB?useUnicode=true&amp;characterEncoding=utf8";

            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection(DB_CONNECTION_URL,DB_USER,USER_PASSWORD);

            --->set auto commit to false
            con.setAutoCommit(false);
            ---> or use con.START TRANSACTION / con.BEGIN WORK

            --->Start SQL Statements for transaction
            --->Checking availability of amount
            double availableAmt    = 0;
            pstmt = con.prepareStatement("SELECT ledgerAmt FROM accTable WHERE customerId=?
FOR UPDATE");
            pstmt.setInt(1, customerIdFrom);
            rs = pstmt.executeQuery();
            if(rs.next())
                availableAmt    = rs.getDouble(1);

            if(availableAmt >= transAmount)
            {
                ---> Do Transfer
                ---> taking amount from cutomerIdFrom
                pstmt = con.prepareStatement("UPDATE accTable SET ledgerAmt=ledgerAmt-? WHERE
customerId=?");
                pstmt.setDouble(1, transAmount);
                pstmt.setInt(2, customerIdFrom);
                pstmt.executeUpdate();

                ---> depositing amount in cutomerIdTo
                pstmt = con.prepareStatement("UPDATE accTable SET ledgerAmt=ledgerAmt+? WHERE
customerId=?");
                pstmt.setDouble(1, transAmount);
                pstmt.setInt(2, customerIdTo);
                pstmt.executeUpdate();

                con.commit();
            }
            --->If you performed any insert,update or delete operations before
            ----> this availability check, then include this else part
            /*else { --->Rollback the transaction if availability is less than required
                con.rollback();
```

```
            }*/

        } catch (SQLException ex) {
            ---> Rollback the transaction in case of any error
            con.rollback();
        } finally {
            try {
                if(rs != null)  rs.close();
                if(pstmt != null) pstmt.close();
                if(con != null) con.close();
            }
        }
    }

    public static void main(String[] args) {
        doTransfer(500, 1020, 1021);
        -->doTransfer(transAmount, customerIdFrom, customerIdTo);
    }
}
```

Операция JDBC гарантирует, что все операторы SQL в блоке транзакции выполнены успешно, если какой-либо из SQL-оператора в блоке транзакции не удался, отменил и отменил все операции внутри блока транзакций.

Прочитайте Сделка онлайн: https://riptutorial.com/ru/mysql/topic/5771/сделка

глава 62: События

Examples

Создать мероприятие

У Mysql есть свои функции EVENT, чтобы избежать сложных взаимодействий cron, когда большая часть того, что вы планируете, связана с SQL и меньше связана с файлами. См. Страницу руководства здесь . Подумайте о событиях как хранимых процедурах, которые планируется запустить с повторяющимися интервалами.

Чтобы сэкономить время при отладке связанных с событиями проблем, имейте в виду, что глобальный обработчик событий должен быть включен для обработки событий.

```
SHOW VARIABLES WHERE variable_name='event_scheduler';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| event_scheduler | OFF   |
+-----------------+-------+
```

При его выключении ничего не выйдет. Поэтому включите его:

```
SET GLOBAL event_scheduler = ON;
```

Схема тестирования

```
create table theMessages
(   id INT AUTO_INCREMENT PRIMARY KEY,
    userId INT NOT NULL,
    message VARCHAR(255) NOT NULL,
    updateDt DATETIME NOT NULL,
    KEY(updateDt)
);

INSERT theMessages(userId,message,updateDt) VALUES (1,'message 123','2015-08-24 11:10:09');
INSERT theMessages(userId,message,updateDt) VALUES (7,'message 124','2015-08-29');
INSERT theMessages(userId,message,updateDt) VALUES (1,'message 125','2015-09-03 12:00:00');
INSERT theMessages(userId,message,updateDt) VALUES (1,'message 126','2015-09-03 14:00:00');
```

Вышеуказанные вставки предоставлены для отображения начальной точки. Обратите внимание, что два события, созданные ниже, будут очищать строки.

Создайте 2 события, 1-й трафик ежедневно, 2-й прогон каждые 10 минут

Игнорируйте то, что они на самом деле делают (играя друг против друга). Дело в INTERVAL и планировании.

```
DROP EVENT IF EXISTS `delete7DayOldMessages`;
DELIMITER $$
CREATE EVENT `delete7DayOldMessages`
  ON SCHEDULE EVERY 1 DAY STARTS '2015-09-01 00:00:00'
  ON COMPLETION PRESERVE
DO BEGIN
   DELETE FROM theMessages
   WHERE datediff(now(),updateDt)>6; -- not terribly exact, yesterday but <24hrs is still 1
day

  -- Other code here

END$$
DELIMITER ;
```

...

```
DROP EVENT IF EXISTS `Every_10_Minutes_Cleanup`;
DELIMITER $$
CREATE EVENT `Every_10_Minutes_Cleanup`
  ON SCHEDULE EVERY 10 MINUTE STARTS '2015-09-01 00:00:00'
  ON COMPLETION PRESERVE
DO BEGIN
   DELETE FROM theMessages
   WHERE TIMESTAMPDIFF(HOUR, updateDt, now())>168; -- messages over 1 week old (168 hours)

   -- Other code here
END$$
DELIMITER ;
```

Показать статусы событий (разные подходы)

```
SHOW EVENTS FROM my_db_name; -- List all events by schema name (db name)
SHOW EVENTS;
SHOW EVENTS\G; -- <--------- I like this one from mysql> prompt

*************************** 1. row ***************************
                  Db: my_db_name
                Name: delete7DayOldMessages
             Definer: root@localhost
           Time zone: SYSTEM
                Type: RECURRING
          Execute at: NULL
      Interval value: 1
      Interval field: DAY
              Starts: 2015-09-01 00:00:00
                Ends: NULL
              Status: ENABLED
          Originator: 1
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: utf8_general_ci
*************************** 2. row ***************************
```

```
                  Db: my_db_name
                Name: Every_10_Minutes_Cleanup
             Definer: root@localhost
           Time zone: SYSTEM
                Type: RECURRING
          Execute at: NULL
      Interval value: 10
      Interval field: MINUTE
             Starts: 2015-09-01 00:00:00
               Ends: NULL
             Status: ENABLED
          Originator: 1
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: utf8_general_ci
2 rows in set (0.06 sec)
```

Случайные вещи для рассмотрения

`DROP EVENT someEventName;` - Удаляет событие и его код.

`ON COMPLETION PRESERVE` - Когда мероприятие завершено, сохраните его. В противном случае он будет удален.

События похожи на триггеры. Они не вызываются программой пользователя. Скорее, они запланированы. Таким образом, они преуспевают или терпят неудачу молча.

Ссылка на страницу руководства показывает довольно много гибкости при выборе интервалов, показанных ниже:

> интервал:
>
> ```
> quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |
>           WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
>           DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}
> ```

События - это мощные механизмы, которые обрабатывают повторяющиеся и запланированные задачи для вашей системы. Они могут содержать столько утверждений, DDL и DML-подпрограмм, и сложные объединения, которые вы можете разумно пожелать. См. Страницу руководства MySQL на тему « Ограничения на сохраненные программы» .

Прочитайте События онлайн: https://riptutorial.com/ru/mysql/topic/4319/события

глава 63: Советы по производительности Mysql

Examples

Оптимизация выписок

Ниже приведены некоторые советы, которые следует помнить, когда мы пишем запрос выбора в MySQL, который может помочь нам и сократить время запроса:

1. Всякий раз, когда мы используем, где в большой таблице, мы должны убедиться, что столбец в где предложение индексируется или нет. Пример: - выберите * у сотрудника, где user_id> 2000. user_id, если проиндексирован, то ускорит оценку запроса atlot. Индексы также очень важны во время соединений и внешних ключей.

2. Когда вам понадобится меньший раздел контента, а затем извлечение целых данных из таблицы, попробуйте использовать лимит. Вместо этого пишут Ex: - Выберите * у сотрудника. Если вам нужно только первое 20 сотрудников из lakhs, тогда просто используйте лимит Ex: - Выберите * у сотрудника LIMIT 20.

3. Вы также можете оптимизировать свой запрос, указав имя столбца, которое вы хотите в наборе результатов. Вместо этого пишут Ex: - Выберите * у сотрудника. Просто укажите имя столбца, из которого вам нужны данные, если в таблице много столбцов, и вы хотите иметь данные для нескольких из них. Пример: - Выберите идентификатор, имя сотрудника.

4. Индексный столбец, если вы используете для проверки значения NULL в разделе where. Если у вас есть оператор SELECT * FROM tbl_name WHERE key_col IS NULL; то, если key_col проиндексирован, запрос будет оцениваться быстрее.

Оптимизация структуры хранилища для таблиц InnoDB

1. В InnoDB, имеющий длинный PRIMARY KEY (либо один столбец с длинным значением, либо несколько столбцов, которые образуют длинное составное значение), тратит много дискового пространства. Значение первичного ключа для строки дублируется во всех вторичных индексных записях, указывающих на одну и ту же строку. Создайте столбец AUTO_INCREMENT в качестве первичного ключа, если ваш первичный ключ длинный.
2. Используйте тип данных VARCHAR вместо CHAR для хранения строк переменной длины или для столбцов со многими значениями NULL. Столбец CHAR (N) всегда принимает N символов для хранения данных, даже если строка короче или ее значение равно NULL. Меньшие таблицы лучше подходят для пула буферов и

уменьшают дисковый ввод-вывод.

При использовании формата строки COMPACT (формат InnoDB по умолчанию) и наборов символов переменной длины, таких как utf8 или sjis, столбцы CHAR (N) занимают переменную площадь, но по меньшей мере N байтов.

3. Для больших таблиц или содержащих много повторяющихся текстовых или числовых данных, используйте формат строки COMPRESSED. Меньше дискового ввода-вывода требуется для переноса данных в буферный пул или для полного сканирования таблиц. Прежде чем принимать постоянное решение, измерьте объем сжатия, который вы можете достичь, используя формат COMPRESSED или COMPACT. *Предостережение:* тесты редко показывают лучше сжатия 2: 1, и в buffer_pool для COMPRESSED много накладных расходов.
4. Как только ваши данные достигнут стабильного размера, или растущая таблица увеличится на десятки или несколько сотен мегабайт, подумайте об использовании инструкции OPTIMIZE TABLE для реорганизации таблицы и сжатия любого потерянного пространства. Для реорганизованных таблиц требуется меньше операций ввода-вывода диска для полного сканирования таблиц. Это простой способ, который может повысить производительность, когда другие методы, такие как улучшение использования индекса или кода приложения настройки, нецелесообразны. *Предостережение* . Независимо от размера таблицы, OPTIMIZE TABLE следует выполнять только редко. Это потому, что это дорого, и редко улучшает таблицу настолько, что стоит того. InnoDB неплохо держит свои B + деревья свободными от много потерянного пространства.

   OPTIMIZE TABLE копирует часть данных таблицы и перестраивает индексы. Выгоды связаны с улучшенной упаковкой данных внутри индексов и уменьшением фрагментации в табличных пространствах и на диске. Преимущества варьируются в зависимости от данных в каждой таблице. Вы можете обнаружить, что есть значительная прибыль для некоторых, а не для других, или что прибыль уменьшается со временем, пока вы не оптимизируете таблицу. Эта операция может быть медленной, если таблица большая, или если перестроенные индексы не вписываются в буферный пул. Первый запуск после добавления большого количества данных в таблицу часто намного медленнее, чем более поздние.

Построение составного индекса

Во многих ситуациях составной индекс работает лучше, чем индекс с одним столбцом. Чтобы построить оптимальный составной индекс, заполните его столбцами в этом порядке.

- `=` столбцы (столбцы) из `WHERE` . (например, `INDEX(a,b,...)` для `WHERE a=12 AND b='xyz' ...`)
- `IN` column (s); оптимизатор может проскочить через индекс.
- Один «диапазон» (например, `x BETWEEN 3 AND 9` , `name LIKE 'J%'`). Он не будет

использовать ничего в первом столбце диапазона.
- Все столбцы в `GROUP BY` , чтобы
- Все столбцы в `ORDER BY` по порядку. Работает только в том случае, если все `ASC` или все `DESC` или вы используете 8.0.

Примечания и исключения:

- Не дублируйте столбцы.
- Пропускайте все случаи, которые не применяются.
- Если вы не используете все столбцы `WHERE` , нет необходимости переходить к `GROUP BY` и т. Д.
- Бывают случаи, когда полезно индексировать только столбцы `ORDER BY` , игнорируя `WHERE` .
- Не «спрячь» столбец в функции (например, `DATE(x) = ...` не может использовать `x` в индексе.)
- `text_col(99)` «Префикс» (например, `text_col(99)`) вряд ли будет полезно; может повредить.

Подробнее и советы .

Прочитайте Советы по производительности Mysql онлайн:
https://riptutorial.com/ru/mysql/topic/5752/советы-по-производительности-mysql

глава 64: Создание баз данных

Синтаксис

- CREATE {DATABASE | SCHEMA} [ЕСЛИ НЕ СУЩЕСТВУЕТ] db_name [create_specification] /// Создание базы данных
- DROP {БАЗЫ ДАННЫХ | SCHEMA} [IF EXISTS] db_name /// Чтобы удалить базу данных

параметры

параметр	подробности
СОЗДАТЬ БАЗА ДАННЫХ	Создает базу данных с заданным именем
СОЗДАТЬ СХЕМА	Это синоним `CREATE DATABASE`
ЕСЛИ НЕ СУЩЕСТВУЕТ	Используется, чтобы избежать ошибки выполнения, если указанная база данных уже существует
create_specification	`create_specification` определяют характеристики базы данных, такие как `CHARACTER SET` и `COLLATE` (сортировка базы данных)

Examples

Создание базы данных, пользователей и грантов

Создайте базу данных. Обратите внимание, что сокращенное слово SCHEMA может использоваться как синоним.

```
CREATE DATABASE Baseball; -- creates a database named Baseball
```

Если база данных уже существует, возвращается ошибка 1007. Чтобы обойти эту ошибку, попробуйте:

```
CREATE DATABASE IF NOT EXISTS Baseball;
```

Так же,

```
DROP DATABASE IF EXISTS Baseball; -- Drops a database if it exists, avoids Error 1008
DROP DATABASE xyz; -- If xyz does not exist, ERROR 1008 will occur
```

Из-за вышеперечисленных возможностей ошибки операторы DDL часто используются с `IF EXISTS` .

Можно создать базу данных с установкой CHARACTER SET по умолчанию и сортировкой. Например:

```
CREATE DATABASE Baseball CHARACTER SET utf8 COLLATE utf8_general_ci;

SHOW CREATE DATABASE Baseball;
+----------+-----------------------------------------------------------------+
| Database | Create Database                                                 |
+----------+-----------------------------------------------------------------+
| Baseball | CREATE DATABASE `Baseball` /*!40100 DEFAULT CHARACTER SET utf8 */ |
+----------+-----------------------------------------------------------------+
```

См. Текущие базы данных:

```
SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| ajax_stuff         |
| Baseball           |
+--------------------+
```

Установите текущую активную базу данных и посмотрите информацию:

```
USE Baseball; -- set it as the current database
SELECT @@character_set_database as cset,@@collation_database as col;
+------+-----------------+
| cset | col             |
+------+-----------------+
| utf8 | utf8_general_ci |
+------+-----------------+
```

Вышеприведенные настройки CHARACTER SET и Collation для базы данных.

Создайте пользователя:

```
CREATE USER 'John123'@'%' IDENTIFIED BY 'OpenSesame';
```

Вышеприведенное создает пользователя John123, способного подключаться к любому имени хоста из-за шаблона `%` . Пароль для пользователя установлен в «OpenSesame», который hashed.

И создать другое:

```
CREATE USER 'John456'@'%' IDENTIFIED BY 'somePassword';
```

Покажите, что пользователи были созданы путем изучения специальной базы данных `mysql`

:

```
SELECT user,host,password from mysql.user where user in ('John123','John456');
+---------+------+-------------------------------------------+
| user    | host | password                                  |
+---------+------+-------------------------------------------+
| John123 | %    | *E6531C342ED87 .....................      |
| John456 | %    | *B04E11FAAAE9A .....................      |
+---------+------+-------------------------------------------+
```

Обратите внимание, что на данный момент пользователи созданы, но без каких-либо разрешений использовать базу данных Baseball.

Работа с разрешениями для пользователей и баз данных. Предоставьте права пользователю John123 иметь полные привилегии в базе данных бейсбола и просто права SELECT для другого пользователя:

```
GRANT ALL ON Baseball.* TO 'John123'@'%';
GRANT SELECT ON Baseball.* TO 'John456'@'%';
```

Проверьте вышеуказанное:

```
SHOW GRANTS FOR 'John123'@'%';
+-----------------------------------------------------------------------------------------------
-----------+
| Grants for John123@%
|
+-----------------------------------------------------------------------------------------------
-----------+
| GRANT USAGE ON *.* TO 'John123'@'%' IDENTIFIED BY PASSWORD '*E6531C342ED87
.....................        |
| GRANT ALL PRIVILEGES ON `baseball`.* TO 'John123'@'%'
|
+-----------------------------------------------------------------------------------------------
-----------+

SHOW GRANTS FOR 'John456'@'%';
+-----------------------------------------------------------------------------------------------
-----------+
| Grants for John456@%
|
+-----------------------------------------------------------------------------------------------
-----------+
| GRANT USAGE ON *.* TO 'John456'@'%' IDENTIFIED BY PASSWORD '*B04E11FAAAE9A
.....................        |
| GRANT SELECT ON `baseball`.* TO 'John456'@'%'
|
+-----------------------------------------------------------------------------------------------
-----------+
```

Обратите внимание, что `GRANT USAGE` которое вы всегда увидите, означает просто, что пользователь может войти в систему. Это все, что это значит.

MyDatabase

Вы *должны* создать свою собственную базу данных, а не использовать запись в любую из существующих баз данных. Это, вероятно, будет одной из первых вещей, которые нужно сделать после первого подключения.

```
CREATE DATABASE my_db;
USE my_db;
CREATE TABLE some_table;
INSERT INTO some_table ...;
```

Вы можете ссылаться на свою таблицу, пройдя квалификацию с именем базы данных: `my_db.some_table` .

Системные базы данных

Для использования MySQL используются следующие базы данных. Вы можете прочитать (`SELECT`) их, но вы не должны писать (`INSERT` / `UPDATE` / `DELETE`) таблицы в них. (Есть несколько исключений.)

- `mysql` - репозиторий для информации `GRANT` и некоторые другие вещи.
- `information_schema` - Таблицы здесь являются «виртуальными» в том смысле, что они фактически проявляются в структурах памяти. Их содержимое включает схему для всех таблиц.
- `performance_schema` - ?? [примите, затем отредактируйте]
- другие ?? (для MariaDB, Galera, TokuDB и т. д.)

Создание и выбор базы данных

Если администратор создает вашу базу данных для вас при настройке ваших прав, вы можете начать ее использовать. В противном случае вам нужно создать его самостоятельно:

```
mysql> CREATE DATABASE menagerie;
```

В Unix имена баз данных чувствительны к регистру (в отличие от ключевых слов SQL), поэтому вы всегда должны ссылаться на свою базу данных как на звезду, а не на Menagerie, MENAGERIE или какой-либо другой вариант. Это также верно для имен таблиц. (В Windows это ограничение не применяется, хотя вы должны обращаться к базам данных и таблицам, используя один и тот же буквенный регистр во всем заданном запросе. Однако по целому ряду причин рекомендуемая передовая практика всегда должна использовать тот же буквенный регистр, который использовался, когда база данных была создана.)

Создание базы данных не выбирает ее для использования; вы должны сделать это явно. Чтобы сделать звезду текущей базы данных, используйте это утверждение:

```
mysql> USE menagerie
Database changed
```

Ваша база данных должна быть создана только один раз, но вы должны выбрать ее для использования каждый раз, когда вы начинаете сеанс mysql. Вы можете сделать это, выпустив инструкцию USE, как показано в примере. Кроме того, вы можете выбрать базу данных в командной строке при вызове mysql. Просто укажите его имя после любых параметров подключения, которые вам могут потребоваться. Например:

```
shell> mysql -h host -u user -p menagerie
Enter password: ********
```

Прочитайте Создание баз данных онлайн: https://riptutorial.com/ru/mysql/topic/600/создание-баз-данных

глава 65: Создание таблицы

Синтаксис

- CREATE TABLE table_name (column_name1 data_type (size), column_name2 data_type (size), column_name3 data_type (size), ...); // Создание основной таблицы
- CREATE TABLE имя_таблицы [IF NOT EXISTS] (column_name1 data_type (size), column_name2 data_type (size), column_name3 data_type (size), ...); // Проверка существующих таблиц
- CREATE [TEMPORARY] TABLE table_name [IF NOT EXISTS] (column_name1 data_type (size), column_name2 data_type (size), column_name3 data_type (size), ...); // Создание временной таблицы
- CREATE TABLE new_tbl [AS] SELECT * FROM orig_tbl; // Создание таблицы из SELECT

замечания

Оператор `CREATE TABLE` должен заканчиваться спецификацией `ENGINE` :

```
CREATE TABLE table_name ( column_definitions ) ENGINE=engine;
```

Некоторые параметры:

- `InnoDB` : (по умолчанию, начиная с версии 5.5.5). Это безопасный для транзакций (ACID совместимый) движок. Он имеет транзакционные фиксации и откаты, возможности восстановления при сбое и блокировку на уровне строк.
- `MyISAM` : (по умолчанию до версии 5.5.5) Это простой движок. Он не поддерживает транзакции и внешние ключи, но он полезен для хранения данных.
- `Memory` : сохраняет все данные в ОЗУ для чрезвычайно быстрых операций, но таблица данных будет потеряна при перезапуске базы данных.

Другие варианты двигателя здесь .

Examples

Создание основной таблицы

Оператор `CREATE TABLE` используется для создания таблицы в базе данных MySQL.

```
CREATE TABLE Person (
    `PersonID`      INTEGER NOT NULL PRIMARY KEY,
```

```
    `LastName`      VARCHAR(80),
    `FirstName`     VARCHAR(80),
    `Address`       TEXT,
    `City`          VARCHAR(100)
) Engine=InnoDB;
```

Каждое полевое определение должно иметь:

1. Имя поля: Действительное поле. Не забудьте зафиксировать имена в `-chars. Это гарантирует, что вы можете использовать, например, пробелы в поле имя.
2. Тип данных [Длина]: если поле `CHAR` или `VARCHAR` , обязательно указать длину поля.
3. Атрибуты `NULL` | `NOT NULL` : Если задано `NOT NULL` , любая попытка сохранить значение `NULL` в этом поле не будет выполнена.
4. Подробнее о типах данных и их атрибутах см . Здесь .

`Engine=...` - необязательный параметр, используемый для указания механизма хранения таблицы. Если механизм хранения не указан, таблица будет создана с использованием механизма хранения таблиц по умолчанию сервера (обычно InnoDB или MyISAM).

Установка значений по умолчанию

Кроме того, если это имеет смысл, вы можете установить значение по умолчанию для каждого поля, используя `DEFAULT` :

```
CREATE TABLE Address (
    `AddressID`    INTEGER NOT NULL PRIMARY KEY,
    `Street`       VARCHAR(80),
    `City`         VARCHAR(80),
    `Country`      VARCHAR(80) DEFAULT "United States",
    `Active`       BOOLEAN DEFAULT 1,
) Engine=InnoDB;
```

Если во вставках ни одна `Street` не указана, это поле будет `NULL` при извлечении. Если ни одна `Country` не указана при вставке, она будет по умолчанию «Соединенными Штатами».

Вы можете установить значения по умолчанию для всех типов столбцов, за исключением полей `BLOB` , `TEXT` , `GEOMETRY` и `JSON` .

Создание таблицы с помощью основного ключа

```
CREATE TABLE Person (
    PersonID     INT UNSIGNED NOT NULL,
    LastName     VARCHAR(66) NOT NULL,
    FirstName    VARCHAR(66),
    Address      VARCHAR(255),
    City         VARCHAR(66),
    PRIMARY KEY (PersonID)
);
```

Первичный ключ - это `NOT NULL` одиночный или многоколоночный идентификатор, который однозначно идентифицирует строку таблицы. Создается индекс , и если он явно не объявлен как `NOT NULL` , MySQL будет объявлять их так тихо и неявно.

В таблице может быть только один `PRIMARY KEY` , и каждая таблица рекомендуется иметь. InnoDB автоматически создаст его в своем отсутствии (как видно из документации MySQL), хотя это менее желательно.

Часто `AUTO_INCREMENT INT` также известный как «суррогатный ключ», используется для оптимизации тонких индексов и отношений с другими таблицами. Это значение будет (обычно) увеличиваться на 1 при добавлении новой записи, начиная с значения по умолчанию 1.

Однако, несмотря на свое название, цель не состоит в том, чтобы гарантировать, что значения являются инкрементальными, просто они являются последовательными и уникальными.

Значение `INT` автоматическим инкрементом не будет сброшено до его начального значения по умолчанию, если все строки в таблице будут удалены, если таблица не будет усечена с помощью инструкции `TRUNCATE TABLE` .

Определение одного столбца в качестве основного ключа (встроенное определение)

Если первичный ключ состоит из одного столбца, предложение `PRIMARY KEY` может быть вставлено в строку с определением столбца:

```
CREATE TABLE Person (
    PersonID     INT UNSIGNED NOT NULL PRIMARY KEY,
    LastName     VARCHAR(66) NOT NULL,
    FirstName    VARCHAR(66),
    Address      VARCHAR(255),
    City         VARCHAR(66)
);
```

Эта форма команды короче и легче читать.

Определение первичного ключа с несколькими столбцами

Также возможно определить первичный ключ, содержащий более одного столбца. Это можно сделать, например, на дочерней таблице отношения внешнего ключа. Первичный ключ с несколькими столбцами определяется путем перечисления участвующих столбцов в отдельном предложении `PRIMARY KEY` . Внутренний синтаксис здесь не разрешен, так как только один столбец может быть объявлен как `PRIMARY KEY` . Например:

```
CREATE TABLE invoice_line_items (
    LineNum      SMALLINT UNSIGNED NOT NULL,
    InvoiceNum   INT UNSIGNED NOT NULL,
    -- Other columns go here
    PRIMARY KEY (InvoiceNum, LineNum),
    FOREIGN KEY (InvoiceNum) REFERENCES -- references to an attribute of a table
);
```

Обратите внимание, что столбцы первичного ключа *должны* быть указаны в логическом порядке сортировки, которые *могут* отличаться от порядка, в котором были определены столбцы, как в приведенном выше примере.

Большим индексам требуется больше дискового пространства, памяти и ввода-вывода. Поэтому ключи должны быть как можно меньше (особенно в отношении составленных ключей). В InnoDB каждый «вторичный индекс» включает в себя копию столбцов `PRIMARY KEY` .

Создание таблицы с помощью внешнего ключа

```
CREATE TABLE Account (
    AccountID     INT UNSIGNED NOT NULL,
    AccountNo     INT UNSIGNED NOT NULL,
    PersonID    INT UNSIGNED,
    PRIMARY KEY (AccountID),
    FOREIGN KEY (PersonID) REFERENCES Person (PersonID)
) ENGINE=InnoDB;
```

Внешний ключ: внешний ключ (`FK`) представляет собой либо столбцы, либо столбцы из нескольких столбцов в таблице *ссылок* . Этот `FK` подтверждается, что существует в таблице, на которую делается *ссылка* . Настоятельно рекомендуется, чтобы *ссылочный* ключ таблицы, подтверждающий, что `FK` является основным ключом, но не применяется. Он используется как быстрый поиск в *ссылке,* где он не должен быть уникальным, и на самом деле может быть самым левым индексом.

Внешние отношения ключей включают родительскую таблицу, содержащую центральные значения данных, и дочернюю таблицу с одинаковыми значениями, указывающую на ее родительский элемент. Предложение FOREIGN KEY указано в дочерней таблице. В родительских и дочерних таблицах должен использоваться один и тот же механизм хранения. Они не должны быть ВРЕМЕННЫМИ таблицами.

Соответствующие столбцы внешнего ключа и ссылочного ключа должны иметь похожие типы данных. Размер и знак целочисленных типов должны быть одинаковыми. Длина

типов строк не обязательно должна быть одинаковой. Для небинных (символьных) строковых столбцов набор символов и сопоставление должны быть одинаковыми.

Примечание. Ограничения внешнего ключа поддерживаются в системе хранения данных InnoDB (не MyISAM или MEMORY). БД-настройки с использованием других движков принимают этот оператор `CREATE TABLE` но не будут учитывать ограничения внешнего ключа. (Хотя новые версии MySQL по умолчанию `InnoDB` , но это хорошая практика, чтобы быть явным.)

Клонирование существующей таблицы

Таблица может быть воспроизведена следующим образом:

```
CREATE TABLE ClonedPersons LIKE Persons;
```

Новая таблица будет иметь ту же структуру, что и исходная таблица, включая индексы и атрибуты столбцов.

Помимо создания таблицы вручную, также можно создать таблицу, выбрав данные из другой таблицы:

```
CREATE TABLE ClonedPersons SELECT * FROM Persons;
```

Вы можете использовать любые обычные функции `SELECT` для изменения данных по мере их поступления:

```
CREATE TABLE ModifiedPersons
SELECT PersonID, FirstName + LastName AS FullName FROM Persons
WHERE LastName IS NOT NULL;
```

Первичные ключи и индексы не сохраняются при создании таблиц из `SELECT` . Вы должны обновить их:

```
CREATE TABLE ModifiedPersons (PRIMARY KEY (PersonID))
SELECT PersonID, FirstName + LastName AS FullName FROM Persons
WHERE LastName IS NOT NULL;
```

CREATE TABLE FROM SELECT

Вы можете создать одну таблицу из другой, добавив `SELECT` в конец инструкции `CREATE TABLE` :

```
CREATE TABLE stack (
    id_user INT,
    username VARCHAR(30),
    password VARCHAR(30)
);
```

Создайте таблицу в той же базе данных:

```
-- create a table from another table in the same database with all attributes
CREATE TABLE stack2 AS SELECT * FROM stack;

-- create a table from another table in the same database with some attributes
CREATE TABLE stack3 AS SELECT username, password FROM stack;
```

Создание таблиц из разных баз данных:

```
-- create a table from another table from another database with all attributes
CREATE TABLE stack2 AS SELECT * FROM second_db.stack;

-- create a table from another table from another database with some attributes
CREATE TABLE stack3 AS SELECT username, password FROM second_db.stack;
```

NB

Чтобы создать таблицу, аналогичную другой таблице, которая существует в другой базе данных, вам необходимо указать имя базы данных следующим образом:

```
FROM NAME_DATABASE.name_table
```

Показать структуру таблицы

Если вы хотите увидеть информацию о схеме своей таблицы, вы можете использовать одно из следующих действий:

```
SHOW CREATE TABLE child; -- Option 1

CREATE TABLE `child` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `fullName` varchar(100) NOT NULL,
  `myParent` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `mommy_daddy` (`myParent`),
  CONSTRAINT `mommy_daddy` FOREIGN KEY (`myParent`) REFERENCES `parent` (`id`)
      ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Если используется из командной строки mysql, это менее подробное:

```
SHOW CREATE TABLE child \G
```

Менее описательный способ отображения структуры таблицы:

```
mysql> CREATE TABLE Tab1(id int, name varchar(30));
Query OK, 0 rows affected (0.03 sec)

mysql> DESCRIBE Tab1; -- Option 2
```

```
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | YES  |     | NULL    |       |
| name  | varchar(30) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

И **DESCRIBE,** и **DESC** дают тот же результат.

Чтобы увидеть, как `DESCRIBE` выполняется во всех таблицах в базе данных сразу, см. Этот пример .

Создание таблицы с помощью столбца TimeStamp для отображения последнего обновления

Столбец TIMESTAMP будет отображаться при последнем обновлении строки.

```
CREATE TABLE `TestLastUpdate` (
    `ID` INT NULL,
    `Name` VARCHAR(50) NULL,
    `Address` VARCHAR(50) NULL,
    `LastUpdate` TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)
COMMENT='Last Update'
;
```

Прочитайте Создание таблицы онлайн: https://riptutorial.com/ru/mysql/topic/795/создание-таблицы

глава 66: Создать нового пользователя

замечания

Чтобы просмотреть список пользователей MySQL, мы используем следующую команду:

```
SELECT User,Host FROM mysql.user;
```

Examples

Создание пользователя MySQL

Для создания нового пользователя нам нужно выполнить простые шаги, как показано ниже:

Шаг 1. Вход в `MySQL` как `root`

```
$ mysql -u root -p
```

Шаг 2. Мы увидим приглашение командной строки mysql.

```
mysql> CREATE USER 'my_new_user'@'localhost' IDENTIFIED BY 'test_password';
```

Здесь мы успешно создали нового пользователя, но у этого пользователя не будет никаких `permissions` , поэтому для назначения `permissions` для пользователя используйте следующую команду:

```
mysql> GRANT ALL PRIVILEGES ON my_db.* TO 'my_new_user'@'localhost' identified by
'my_password';
```

Укажите пароль

Основное использование:

```
mysql> CREATE USER 'my_new_user'@'localhost' IDENTIFIED BY 'test_password';
```

Однако для ситуаций, когда не рекомендуется жестко закодировать пароль в открытом тексте, также можно указать напрямую, используя директиву `PASSWORD` , хешированное значение, возвращаемое функцией `PASSWORD()` :

```
mysql> select PASSWORD('test_password'); -- returns *4414E26EDED6D661B5386813EBBA95065DBC4728
mysql> CREATE USER 'my_new_user'@'localhost' IDENTIFIED BY PASSWORD
'*4414E26EDED6D661B5386813EBBA95065DBC4728';
```

Создать нового пользователя и предоставить все привилегии для схемы

```
grant all privileges on schema_name.* to 'new_user_name'@'%' identified by 'newpassword';
```

Внимание: это можно использовать для создания нового пользователя root

Переименование пользователя

```
rename user 'user'@'%' to 'new_name`@'%';
```

Если вы создаете пользователя по ошибке, вы можете изменить его имя

Прочитайте Создать нового пользователя онлайн: https://riptutorial.com/ru/mysql/topic/3508/создать-нового-пользователя

глава 67: СОРТИРОВАТЬ ПО

Examples

Контексты

Предложения в `SELECT` имеют определенный порядок:

```
SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ...
    ORDER BY ...  -- goes here
    LIMIT ... OFFSET ...;

( SELECT ... ) UNION ( SELECT ... ) ORDER BY ...  -- for ordering the result of the UNION.

SELECT ... GROUP_CONCAT(DISTINCT x ORDER BY ... SEPARATOR ...) ...

ALTER TABLE ... ORDER BY ... -- probably useful only for MyISAM; not for InnoDB
```

основной

ЗАКАЗ BY x

`x` может быть любым типом данных.

- `NULL`s предшествуют не-NULL.
- По умолчанию используется `ASC` (от самого низкого до самого высокого)
- Строки (`VARCHAR` и т. Д.) Заказываются в соответствии с `COLLATION` декларации
- `ENUM`s упорядочиваются по порядку его строк.

По возрастанию, по убыванию

```
ORDER BY x ASC  -- same as default
ORDER BY x DESC  -- highest to lowest
ORDER BY lastname, firstname  -- typical name sorting; using two columns
ORDER BY submit_date DESC  -- latest first
ORDER BY submit_date DESC, id ASC  -- latest first, but fully specifying order.
```

- `ASC` = `ASCENDING` , `DESC` = `DESCENDING`
- `NULL`s первом месте даже для `DESC` .
- В приведенных выше примерах `INDEX(x)` , `INDEX(lastname, firstname)` , `INDEX(submit_date)` могут значительно повысить производительность.

Но ... Смешивание `ASC` и `DESC` , как в последнем примере, не может использовать составной индекс. Также не поможет `INDEX(submit_date DESC, id ASC)` - « `DESC` » распознается синтаксически в объявлении `INDEX` , но игнорируется.

Некоторые трюки

```
ORDER BY FIND_IN_SET(card_type, "MASTER-CARD,VISA,DISCOVER") -- sort 'MASTER-CARD' first.
ORDER BY x IS NULL, x  -- order by `x`, but put `NULLs` last.
```

Пользовательский заказ

```
SELECT * FROM some_table WHERE id IN (118, 17, 113, 23, 72)
ORDER BY FIELD(id, 118, 17, 113, 23, 72);
```

Возвращает результат в указанном порядке идентификаторов.

Я бы	...
118	...
17	...
113	...
23	...
72	...

Полезно, если идентификаторы уже отсортированы, и вам просто нужно получить строки.

Прочитайте СОРТИРОВАТЬ ПО онлайн: https://riptutorial.com/ru/mysql/topic/5469/сортировать-по

глава 68: Сохраненные процедуры (процедуры и функции)

параметры

параметр	подробности
ВОЗВРАТ	Задает тип данных, который можно вернуть из функции.
ВЕРНУТЬ	Фактическая переменная или значение после синтаксиса `RETURN` - это то, что возвращается туда, откуда была вызвана функция.

замечания

Сохраненная процедура - это либо процедура, либо функция.

Процедура вызывается с помощью оператора CALL и может только возвращать значения с использованием выходных переменных.

Функция может вызываться изнутри оператора точно так же, как любая другая функция, и может возвращать скалярное значение.

Examples

Создать функцию

Следующая (тривиальная) примерная функция просто возвращает значение константы `INT 12` .

```
DELIMITER ||
CREATE FUNCTION functionname()
RETURNS INT
BEGIN
    RETURN 12;
END;
||
DELIMITER ;
```

Первая строка определяет, для чего должен быть изменен символ разделителя (`DELIMITER ||`), это необходимо установить до того, как функция будет создана иначе, если оставить ее по умолчанию `;` затем первый `;` который находится в теле функции, будет приниматься за конец инструкции `CREATE` , которая обычно не является желаемой.

После запуска `CREATE FUNCTION` вы должны установить для разделителя значение по умолчанию `;` как видно из кода функции в приведенном выше примере (`DELIMITER ;`).

Выполнение этой функции выполняется следующим образом:

```
SELECT functionname();
+----------------+
| functionname() |
+----------------+
|             12 |
+----------------+
```

Несколько более сложный (но все же тривиальный) пример принимает параметр и добавляет к нему константу:

```
DELIMITER $$
CREATE FUNCTION add_2 ( my_arg INT )
  RETURNS INT
BEGIN
  RETURN (my_arg + 2);
END;
$$
DELIMITER ;

SELECT add_2(12);
+-----------+
| add_2(12) |
+-----------+
|        14 |
+-----------+
```

Обратите внимание на использование другого аргумента в директиве `DELIMITER` . Фактически вы можете использовать любую последовательность символов, которая не появляется в `CREATE` , но обычной практикой является использование удвоенного не-алфавитно-цифрового символа, такого как `\\` , `||` или `$$` .

Рекомендуется всегда изменять параметр до и после создания или обновления функции, процедуры или триггера, поскольку некоторые графические интерфейсы не требуют изменения разделителя, тогда как выполнение запросов через командную строку всегда требует установки разделителя.

Создание процедуры с созданной подготовкой

```
DROP PROCEDURE if exists displayNext100WithName;
DELIMITER $$
CREATE PROCEDURE displayNext100WithName
(    nStart int,
    tblName varchar(100)
)
BEGIN
    DECLARE thesql varchar(500); -- holds the constructed sql string to execute
```

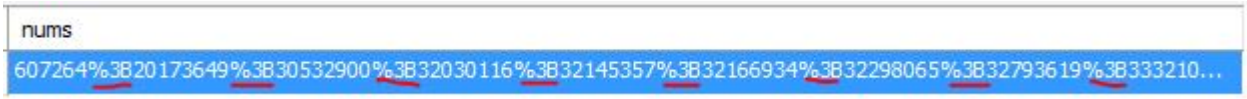
```
    -- expands the sizing of the output buffer to accomodate the output (Max value is at least
4GB)
    SET session group_concat_max_len = 4096; -- prevents group_concat from barfing with error
1160 or whatever it is

    SET @thesql=CONCAT("select group_concat(qid order by qid SEPARATOR '%3B') as nums ","from
(    select qid from ");
    SET @thesql=CONCAT(@thesql,tblName," where qid>? order by qid limit 100 )xDerived");
    PREPARE stmt1 FROM @thesql; -- create a statement object from the construct sql string to
execute
    SET @p1 = nStart; -- transfers parameter passed into a User Variable compatible with the
below EXECUTE
    EXECUTE stmt1 USING @p1;

    DEALLOCATE PREPARE stmt1; -- deallocate the statement object when finished
END$$
DELIMITER ;
```

Создание хранимой процедуры показывает упаковку с DELIMITER, необходимой во многих клиентских инструментах.

Пример вызова:

```
call displayNext100WithName(1,"questions_mysql");
```

Выход образца с разделителем `%3B` (с запятой):



Сохраненная процедура с параметрами IN, OUT, INOUT

```
DELIMITER $$

DROP PROCEDURE IF EXISTS sp_nested_loop$$
CREATE PROCEDURE sp_nested_loop(IN i INT, IN j INT, OUT x INT, OUT y INT, INOUT z INT)
BEGIN
    DECLARE a INTEGER DEFAULT 0;
    DECLARE b INTEGER DEFAULT 0;
    DECLARE c INTEGER DEFAULT 0;
    WHILE a < i DO
        WHILE b < j DO
            SET c = c + 1;
            SET b = b + 1;
        END WHILE;
        SET a = a + 1;
        SET b = 0;
    END WHILE;
    SET x = a, y = c;
    SET z = x + y + z;
END $$
DELIMITER ;
```

Вызывает (CALL) хранимую процедуру:

```
SET @z = 30;
call sp_nested_loop(10, 20, @x, @y, @z);
SELECT @x, @y, @z;
```

Результат:

```
+------+------+------+
|  @x  |  @y  |  @z  |
+------+------+------+
|  10  |  200 |  240 |
+------+------+------+
```

Параметр `IN` передает значение в процедуру. Процедура может изменить значение, но модификация не отображается вызывающему, когда процедура возвращается.

Параметр `OUT` передает значение из процедуры обратно вызывающему абоненту. Его начальное значение равно NULL в пределах процедуры, и его значение отображается вызывающему, когда процедура возвращается.

Параметр `INOUT` инициализируется вызывающим абонентом, может быть изменен процедурой, и любое изменение, сделанное процедурой, отображается вызывающему абоненту при возврате процедуры.

Ссылка: http://dev.mysql.com/doc/refman/5.7/en/create-procedure.html

курсоры

Курсоры позволяют повторять результаты запроса по очереди. Команда `DECLARE` используется для инициализации курсора и связывания его с конкретным SQL-запросом:

```
DECLARE student CURSOR FOR SELECT name FROM studend;
```

Предположим, мы продаем продукты некоторых типов. Мы хотим подсчитать, сколько продуктов каждого типа существует.

Наши данные:

```
CREATE TABLE product
(
  id   INT(10) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  type VARCHAR(50)      NOT NULL,
  name VARCHAR(255)     NOT NULL

);
CREATE TABLE product_type
(
  name VARCHAR(50) NOT NULL PRIMARY KEY
);
CREATE TABLE product_type_count
(
  type  VARCHAR(50)      NOT NULL PRIMARY KEY,
```

```
  count INT(10) UNSIGNED NOT NULL DEFAULT 0
);

INSERT INTO product_type (name) VALUES
  ('dress'),
  ('food');

INSERT INTO product (type, name) VALUES
  ('dress', 'T-shirt'),
  ('dress', 'Trousers'),
  ('food', 'Apple'),
  ('food', 'Tomatoes'),
  ('food', 'Meat');
```

Мы можем достичь цели с помощью хранимой процедуры с помощью курсора:

```
DELIMITER //
DROP PROCEDURE IF EXISTS product_count;
CREATE PROCEDURE product_count()
  BEGIN
    DECLARE p_type VARCHAR(255);
    DECLARE p_count INT(10) UNSIGNED;
    DECLARE done INT DEFAULT 0;
    DECLARE product CURSOR FOR
      SELECT
        type,
        COUNT(*)
      FROM product
      GROUP BY type;
    DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;

    TRUNCATE product_type;

    OPEN product;

    REPEAT
      FETCH product
      INTO p_type, p_count;
      IF NOT done
      THEN
        INSERT INTO product_type_count
        SET
          type  = p_type,
          count = p_count;
      END IF;
    UNTIL done
    END REPEAT;

    CLOSE product;
  END //
DELIMITER ;
```

Когда вы можете вызвать процедуру с помощью:

```
CALL product_count();
```

Результат будет в таблице `product_type_count` :

```
type    | count
----------------
dress   |   2
food    |   3
```

Хотя это хороший пример `CURSOR` , обратите внимание на то, как весь орган процедуры может быть заменен просто

```
INSERT INTO product_type_count
        (type, count)
    SELECT type, COUNT(*)
        FROM product
        GROUP BY type;
```

Это будет работать намного быстрее.

Множественные ResultSets

В отличие от `SELECT` , `Stored Procedure` возвращает несколько наборов результатов. Требуется использовать другой код для сбора результатов `CALL` в Perl, PHP и т. Д.

(Нужен конкретный код здесь или в другом месте!)

Создать функцию

```
DELIMITER $$
CREATE
    DEFINER=`db_username`@`hostname_or_IP`
    FUNCTION `function_name`(optional_param data_type(length_if_applicable))
    RETURNS data_type
BEGIN
    /*
    SQL Statements goes here
    */
END$$
DELIMITER ;
```

RETURNS data_type - любой тип данных MySQL.

Прочитайте Сохраненные процедуры (процедуры и функции) онлайн:
https://riptutorial.com/ru/mysql/topic/1351/сохраненные-процедуры--процедуры-и-функции-

глава 69: Строковые операции

параметры

название	Описание
ASCII ()	Возвращаемое числовое значение самого левого символа
БИН ()	Возвращает строку, содержащую двоичное представление числа
BIT_LENGTH ()	Возвращаемая длина аргумента в битах
СИМВОЛ ()	Возвращает символ для каждого целого числа
CHAR_LENGTH ()	Возвращает количество символов в аргументе
CHARACTER_LENGTH ()	Синоним для CHAR_LENGTH ()
CONCAT ()	Возвращаемая конкатенированная строка
CONCAT_WS ()	Обратный конкатенат с разделителем
ELT ()	Возвращаемая строка с номером индекса
EXPORT_SET ()	Верните строку, такую, что для каждого бита, установленного в битах значения, вы получите строку on и для каждого несоответствующего бита, вы получите строку off
Область ()	Возврат индекса (позиции) первого аргумента в последующих аргументах
FIND_IN_SET ()	Вернуть позицию индекса первого аргумента во втором аргументе
ФОРМАТ()	Возвращает число, отформатированное в указанное число знаков после запятой
FROM_BASE64 ()	Декодировать строку base-64 и вернуть результат
HEX ()	Возвращает шестнадцатеричное представление десятичного или строкового значения

название	Описание
ВСТАВИТЬ ()	Вставьте подстроку в указанной позиции до указанного количества символов
INSTR ()	Возвращает индекс первого вхождения подстроки
LCASE ()	Синонимы для LOWER ()
ОСТАВИЛ()	Вернуть самое левое число символов, как указано
ДЛИНА ()	Возвращает длину строки в байтах
ЛАЙК	Простое сопоставление шаблонов
LOAD_FILE ()	Загрузите указанный файл
LOCATE ()	Вернуть позицию первого вхождения подстроки
НИЖНИЙ ()	Возврат аргумента в нижнем регистре
LPAD ()	Возвращает строковый аргумент, с левой буквой с указанной строкой
LTRIM ()	Удалить ведущие пространства
MAKE_SET ()	Возвращает набор разделенных запятыми строк, которые имеют соответствующий бит в битах
МАТЧ	Выполнение полнотекстового поиска
(MID)	Возвращает подстроку, начиная с указанной позиции
НЕ КАК	Отрицание простого совпадения шаблонов
НЕ REGEXP	Отрицание REGEXP
Октябрь ()	Возвращает строку, содержащую восьмеричное представление числа
OCTET_LENGTH ()	Синоним для LENGTH ()
ORD ()	Возвращать код символа для самого левого символа аргумента
ПОЗИЦИЯ()	Синоним для LOCATE ()
QUOTE ()	Выйти из аргумента для использования в инструкции SQL

название	Описание
REGEXP	Сравнение шаблонов с использованием регулярных выражений
ПОВТОРЕНИЕ()	Повторите строку указанное количество раз
REPLACE ()	Заменить вхождения указанной строки
ЗАДНИЙ ХОД()	Обратить символы в строке
ПРАВО()	Вернуть указанное правое число символов
RLIKE	Синоним для REGEXP
RPAD ()	Добавить строку указанное количество раз
RTRIM ()	Удалить конечные пробелы
SOUNDEX ()	Вернуть строку soundex
ЗВУЧИТ КАК	Сравнить звуки
ПРОСТРАНСТВО()	Возвращает строку указанного количества пробелов
STRCMP ()	Сравните две строки
SUBSTR ()	Вернуть подстроку, как указано
SUBSTRING ()	Вернуть подстроку, как указано
SUBSTRING_INDEX ()	Возвращает подстроку из строки до указанного количества вхождений разделителя
TO_BASE64 ()	Возвращает аргумент, преобразованный в строку base-64
ОТДЕЛКА()	Удалить передние и конечные пробелы
UCASE ()	Синоним для UPPER ()
UNHEX ()	Возвращает строку, содержащую шестнадцатеричное представление числа
ВЕРХНИЙ ()	Преобразование в верхний регистр
WEIGHT_STRING ()	Вернуть строку веса для строки

Examples

Найти элемент в списке, разделенном запятыми

```
SELECT FIND_IN_SET('b','a,b,c');
```

Возвращаемое значение:

    2

```
SELECT FIND_IN_SET('d','a,b,c');
```

Возвращаемое значение:

    0

STR_TO_DATE - преобразовать строку на сегодняшний день

С помощью столбца одного из типов `my_date_field` с именем `my_date_field` со значением, например [строка] `07/25/2016` , следующий оператор демонстрирует использование функции `STR_TO_DATE` :

```
SELECT STR_TO_DATE(my_date_field, '%m/%d/%Y') FROM my_table;
```

Вы можете использовать эту функцию как часть `WHERE` .

LOWER () / LCASE ()

Преобразовать в нижнем регистре строковый аргумент

Синтаксис: LOWER (str)

```
LOWER('fOoBar') -- 'foobar'
LCASE('fOoBar') -- 'foobar'
```

REPLACE ()

Преобразовать в нижнем регистре строковый аргумент

Синтаксис: REPLACE (str, from_str, to_str)

```
REPLACE('foobarbaz', 'bar', 'BAR') -- 'fooBARbaz'
REPLACE('foobarbaz', 'zzz', 'ZZZ') -- 'foobarbaz'
```

SUBSTRING ()

SUBSTRING (или эквивалент: SUBSTR) возвращает подстроку, начиная с указанной позиции и, необязательно, с указанной длиной

Синтаксис: `SUBSTRING(str, start_position)`

```
SELECT SUBSTRING('foobarbaz', 4); -- 'barbaz'
SELECT SUBSTRING('foobarbaz' FROM 4); -- 'barbaz'

-- using negative indexing
SELECT SUBSTRING('foobarbaz', -6); -- 'barbaz'
SELECT SUBSTRING('foobarbaz' FROM -6); -- 'barbaz'
```

Синтаксис: `SUBSTRING(str, start_position, length)`

```
SELECT SUBSTRING('foobarbaz', 4, 3); -- 'bar'
SELECT SUBSTRING('foobarbaz', FROM 4 FOR 3); -- 'bar'

-- using negative indexing
SELECT SUBSTRING('foobarbaz', -6, 3); -- 'bar'
SELECT SUBSTRING('foobarbaz' FROM -6 FOR 3); -- 'bar'
```

UPPER () / UCASE ()

Преобразовать в верхнем регистре аргумент строки

Синтаксис: UPPER (str)

```
UPPER('fOoBar') -- 'FOOBAR'
UCASE('fOoBar') -- 'FOOBAR'
```

ДЛИНА ()

Возвращает длину строки в байтах. Поскольку некоторые символы могут быть закодированы с использованием более одного байта, если вы хотите, чтобы длина в символах отображалась в CHAR_LENGTH ()

Синтаксис: LENGTH (str)

```
LENGTH('foobar') -- 6
LENGTH('fööbar') -- 8 -- contrast with CHAR_LENGTH(...) = 6
```

CHAR_LENGTH ()

Возвращает количество символов в строке

Синтаксис: CHAR_LENGTH (str)

```
CHAR_LENGTH('foobar') -- 6
CHAR_LENGTH('fööbar') -- 6 -- contrast with LENGTH(...) = 8
```

HEX (ул)

Преобразуйте аргумент в шестнадцатеричный. Это используется для строк.

```
HEX('fööbar') -- 66F6F6626172 -- in "CHARACTER SET latin1" because "F6" is hex for ö
HEX('fööbar') -- 66C3B6C3B6626172  -- in "CHARACTER SET utf8 or utf8mb4" because "C3B6" is hex
for ö
```

Прочитайте Строковые операции онлайн: https://riptutorial.com/ru/mysql/topic/1399/строковые-операции

глава 70: Таблица сопоставлений много-ко-многим

замечания

- Отсутствие идентификатора `AUTO_INCREMENT` для этой таблицы. PK - это «натуральный» ПК; нет никакой веской причины для суррогата.
- `MEDIUMINT` - это напоминание о том, что все `INTs` должны быть сделаны настолько маленькими, насколько это безопасно (меньше ⇒ быстрее). Конечно, декларация здесь должна соответствовать определению в связанной с таблицей.
- `UNSIGNED` - почти все INT могут также быть объявлены неотрицательными
- `NOT NULL` - Ну, это правда, не так ли?
- `InnoDB` - более эффективный, чем MyISAM, из-за того, что `PRIMARY KEY` кластеризуется с данными в InnoDB.
- `INDEX(y_id, x_id)` - `INDEX(y_id, x_id) PRIMARY KEY` позволяет эффективно идти в одном направлении; делает другое направление эффективным. Не нужно говорить « `UNIQUE` ; это будет дополнительным усилием для `INSERTs` .
- Во вторичном индексе говорят, что только `INDEX(y_id)` будет работать, потому что он будет подразумевать include `x_id` . Но я предпочел бы сделать более очевидным, что я надеюсь на индекс «покрытия».

Вы *можете* добавить несколько столбцов в таблице; это редко. Дополнительные столбцы могут предоставить информацию о *взаимосвязи,* которую представляет таблица.

Вы *можете* добавить ограничения `FOREIGN KEY` .

Examples

Типичная схема

```
CREATE TABLE XtoY (
    # No surrogate id for this table
    x_id MEDIUMINT UNSIGNED NOT NULL,   -- For JOINing to one table
    y_id MEDIUMINT UNSIGNED NOT NULL,   -- For JOINing to the other table
    # Include other fields specific to the 'relation'
    PRIMARY KEY(x_id, y_id),            -- When starting with X
    INDEX      (y_id, x_id)             -- When starting with Y
) ENGINE=InnoDB;
```

(См. Примечания ниже, для обоснования.)

Прочитайте Таблица сопоставлений много-ко-многим онлайн:
https://riptutorial.com/ru/mysql/topic/4857/таблица-сопоставлений-много-ко-многим

глава 71: Типы данных

Examples

Неявное / автоматическое литье

```
select '123' * 2;
```

Чтобы сделать **умножение** с помощью `2` MySQL, автоматически преобразует строку `123` в число.

Возвращаемое значение:

> 246

Преобразование в число начинается слева направо. Если преобразование невозможно, результат равен `0`

```
select '123ABC' * 2
```

Возвращаемое значение:

> 246

```
select 'ABC123' * 2
```

Возвращаемое значение:

> 0

VARCHAR (255) - или нет

Предлагаемая максимальная длина

Во-первых, я упомянул некоторые общие строки, которые всегда являются шестнадцатеричными или иным образом ограничиваются ASCII. Для этого вы должны указать `CHARACTER SET ascii` (`latin1` в порядке), чтобы он не терял пространство:

```
UUID CHAR(36) CHARACTER SET ascii -- or pack into BINARY(16)
country_code CHAR(2) CHARACTER SET ascii
ip_address CHAR(39) CHARACTER SET ascii -- or pack into BINARY(16)
phone VARCHAR(20) CHARACTER SET ascii -- probably enough to handle extension
postal_code VARCHAR(20) CHARACTER SET ascii -- (not 'zip_code') (don't know the max

city VARCHAR(100) -- This Russian town needs 91:
    Poselok Uchebnogo Khozyaystva Srednego Professionalno-Tekhnicheskoye Uchilishche Nomer
```

```
Odin
country VARCHAR(50) -- probably enough
name VARCHAR(64) -- probably adequate; more than some government agencies allow
```

Почему бы не просто 255? Есть две причины избежать обычной практики использования (255) для всего.

- Когда сложный `SELECT` должен создать временную таблицу (для подзапроса, `UNION` , `GROUP BY` и т. Д.), Предпочтительным выбором является использование механизма `MEMORY` , который помещает данные в ОЗУ. Но `VARCHARs` в процессе превращаются в `CHAR` . Это заставляет `VARCHAR(255) CHARACTER SET utf8mb4` принимать 1020 байт. Это может привести к необходимости разливать на диск, что происходит медленнее.
- В некоторых ситуациях InnoDB будет рассматривать потенциальный размер столбцов в таблице и решить, что он будет слишком большим, прерывая `CREATE TABLE` .

VARCHAR против **TEKCTA**

Рекомендации по использованию для `*TEXT` , `CHAR` и `VARCHAR` , а также некоторые рекомендации:

- Никогда не используйте `TINYTEXT` .
- Почти никогда не используйте `CHAR` - фиксированная длина; каждый символ - это максимальная длина `CHARACTER SET` (например, 4 байта / символ для utf8mb4).
- С `CHAR` используйте `CHARACTER SET ascii` если вы не знаете иначе.
- `VARCHAR(n)` будет усекать при n *символах* ; `TEXT` будет усекаться при некотором количестве *байтов* . (Но, вы хотите усечение?)
- `*TEXT` *может* замедлить сложные `SELECTs` из-за того, как обрабатываются временные таблицы.

INT как AUTO_INCREMENT

Любой размер `INT` может использоваться для `AUTO_INCREMENT` . `UNSIGNED` всегда уместен.

Имейте в виду, что определенные операции «сжигают» `AUTO_INCREMENT` ids. Это может привести к неожиданному разрыву. Примеры: `INSERT IGNORE` и `REPLACE` . Они *могут* предварительно распределить идентификатор, *прежде чем* понимать, что он не понадобится. Это ожидаемое поведение и дизайн в движке InnoDB и не должны препятствовать их использованию.

другие

Уже существует отдельная запись для «FLOAT, DOUBLE, DECIMAL» и «ENUM». Одна страница из типов данных, вероятно, будет громоздкой - я предлагаю «Типы полей» (или ее следует называть «Типы данных»?) - это обзор, а затем разделить на эти страницы тем:

- INTS

- FLOAT, DOUBLE и DECIMAL
- Строки (CHARs, TEXT и т. Д.)
- BINARY и BLOB
- DATETIME, TIMESTAMP и друзья
- ENUM и SET
- Пространственные данные
- Тип JSON (MySQL 5.7.8+)
- Как представлять деньги и другие общие «типы», которые нуждаются в обучении в существующих типах данных

Там, где это необходимо, каждая страница темы должна включать, помимо синтаксиса и примеров:

- Соображения, когда ALTERING
- Размер (байты)
- Контраст с двигателями, отличными от MySQL (с низким приоритетом)
- Соображения при использовании типа данных в PRIMARY KEY или вторичном ключе
- другая передовая практика
- другие проблемы с производительностью

(Я предполагаю, что этот «пример» будет самоликвидироваться, когда мои предложения будут удовлетворены или наложены вето.)

Введение (числовое)

MySQL предлагает несколько различных числовых типов. Они могут быть разбиты на

группа	Типы
Целочисленные типы	`INTEGER` , `INT` , `SMALLINT` , `TINYINT` , `MEDIUMINT` , `BIGINT`
Типы фиксированных точек	`DECIMAL` , `NUMERIC`
Типы с плавающей точкой	`FLOAT` , `DOUBLE`
Тип битового значения	`BIT`

Целочисленные типы

Минимальное значение без знака всегда равно 0.

Тип	Место хранения (Б)	Минимальное значение (Подпись)	Максимальное значение (Подпись)	Максимальное зн (Без знака)
`TINYINT`	1	-2^7 -128	2^7-1 127	2^8-1 255
`SMALLINT`	2	-2^{15} -32768	$2^{15}-1$ 32767	$2^{16}-1$ 65535
`MEDIUMINT`	3	-2^{23} -8388608	$2^{23}-1$ 8388607	$2^{24}-1$ 16777215
`INT`	4	-2^{31} -2147483648	$2^{31}-1$ 2147483647	$2^{32}-1$ 4294967295
`BIGINT`	8	-2^{63} - 9.223.372.036.854.775.808	$2^{63}-1$ 9.223.372.036.854.775.807	$2^{64}-1$ 18.446.744.073.709

Типы фиксированных точек

Типы `DECIMAL` и `NUMERIC` MySQL хранят точные значения числовых данных. Рекомендуется использовать эти типы, чтобы сохранить точную точность, например, за деньги.

Десятичный

Эти значения сохраняются в двоичном формате. В объявлении столбца должны быть указаны точность и масштаб

Precision представляет количество значащих цифр, которые сохраняются для значений.

Масштаб представляет количество цифр, хранящихся *после* десятичной

```
salary DECIMAL(5,2)
```

5 представляет `precision` а 2 - `scale` . В этом примере диапазон значений, которые могут быть сохранены в этом столбце, составляет от `-999.99 to 999.99`

Если параметр масштаба опущен, по умолчанию он равен 0

Этот тип данных может хранить до 65 цифр.

Количество байтов, полученных `DECIMAL(M,N)` составляет *приблизительно* `M/2` .

Типы с плавающей точкой

`FLOAT` и `DOUBLE` представляют собой *приблизительные* типы данных.

Тип	Место хранения	точность	Спектр
FLOAT	4 байта	23 значащих бита / ~ 7 десятичных цифр	10 ^ + / - 38
DOUBLE	8 байт	53 значащих бита / ~ 16 десятичных цифр	10 ^ + / - 308

`REAL` является синонимом `FLOAT` . `DOUBLE PRECISION` является синонимом `DOUBLE` .

Хотя MySQL также разрешает (M, D) квалификатор, *не* использовать его. (M, D) означает, что значения могут быть сохранены с точностью до M суммарных цифр, где D может быть после десятичной. *Числа будут округлены дважды или усечены; это вызовет больше проблем, чем пользы.*

Поскольку значения с плавающей запятой являются приблизительными и не сохраняются в виде точных значений, попытки рассматривать их как точные в сравнении могут привести к проблемам. Обратите внимание, в частности, что значение `FLOAT` редко принимает значение `DOUBLE` .

Тип битового значения

Тип `BIT` полезен для хранения значений битового поля. `BIT(M)` позволяет хранить до M-бит значений, где M находится в диапазоне от `1 to 64`

Вы также можете указать значения с нотой `bit value` .

```
b'111'      -> 7
b'10000000' -> 128
```

Иногда удобно использовать «shift» для построения однобитового значения, например `(1 << 7)` для 128.

Максимальный комбинированный размер всех столбцов BIT в таблице `NDB` равен 4096.

СИМ (п)

`CHAR(n)` - строка *фиксированной* длины `n` *символов* . Если это `CHARACTER SET utf8mb4` , это означает, что он занимает ровно `4*n` *байтов* , независимо от того, какой текст в нем.

Большинство случаев использования `CHAR(n)` включают строки, содержащие английские символы, поэтому должен быть `CHARACTER SET ascii` . (`latin1` будет так же хорошо).

```
country_code CHAR(2) CHARACTER SET ascii,
postal_code  CHAR(6) CHARACTER SET ascii,
uuid    CHAR(39) CHARACTER SET ascii,  -- more discussion elsewhere
```

DATE, DATETIME, TIMESTAMP, YEAR и TIME

Тип данных `DATE` содержит дату, но не временную составляющую. Его формат - `'YYYY-MM-DD'` с диапазоном от «1000-01-01» до «9999-12-31».

Тип `DATETIME` включает время с форматом «YYYY-MM-DD HH: MM: SS». Он имеет диапазон от «1000-01-01 00:00:00» до «9999-12-31 23:59:59».

Тип `TIMESTAMP` - это целочисленный тип, содержащий дату и время с эффективным диапазоном от '1970-01-01 00:00:01' UTC до '2038-01-19 03:14:07' UTC.

Тип `YEAR` представляет год и имеет диапазон от 1901 до 2155.

Тип `TIME` представляет собой время с форматом «HH: MM: SS» и имеет диапазон от «-838: 59: 59» до «838: 59: 59».

Требования к хранению:

```
|-----------|--------------------|----------------------------------------|
| Data Type | Before MySQL 5.6.4 | as of MySQL 5.6.4                      |
|-----------|--------------------|----------------------------------------|
| YEAR      |      1 byte        |  1 byte                                |
| DATE      |      3 bytes       |  3 bytes                               |
| TIME      |      3 bytes       |  3 bytes + fractional seconds storage  |
| DATETIME  |      8 bytes       |  5 bytes + fractional seconds storage  |
| TIMESTAMP |      4 bytes       |  4 bytes + fractional seconds storage  |
|-----------|--------------------|----------------------------------------|
```

Дробные секунды (с версии 5.6.4):

```
|------------------------------|------------------|
| Fractional Seconds Precision | Storage Required |
|------------------------------|------------------|
|              0               |      0 bytes     |
|              1,2             |      1 byte      |
|              3,4             |      2 byte      |
|              5,6             |      3 byte      |
|------------------------------|------------------|
```

См. Типы страниц DATE, DATETIME и TIMESTAMP MySQL , типы хранения данных и дробные секунды во временных значениях .

Прочитайте Типы данных онлайн: https://riptutorial.com/ru/mysql/topic/4137/типы-данных

глава 72: УДАЛЯТЬ

Синтаксис

- DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM table [WHERE conditions] [выражение ORDER BY [ASC | DESC]] [LIMIT number_rows]; /// Синтаксис для строки (-ов) удаления из одной таблицы

параметры

параметр	подробности
НИЗКИЙ ПРИОРИТЕТ	Если предоставлен `LOW_PRIORITY` , удаление будет задерживаться до тех пор, пока в таблице не будет `LOW_PRIORITY` процессов
ИГНОРИРУЙТЕ	Если предоставляется `IGNORE` , все ошибки, возникающие во время удаления, игнорируются
Таблица	Таблица, из которой вы собираетесь удалять записи
ГДЕ условия	Условия, которые должны быть выполнены для записей, подлежащих удалению. Если условий не предусмотрено, все записи из таблицы будут удалены
Выражение ORDER BY	Если предоставляется `ORDER BY` , записи будут удалены в указанном порядке
ПРЕДЕЛ	Он контролирует максимальное количество записей для удаления из таблицы. Учитывая, что `number_rows` будет удален.

Examples

Удалить с предложением Where

```
DELETE FROM `table_name` WHERE `field_one` = 'value_one'
```

Это приведет к удалению всех строк из таблицы, в которой содержимое поля `field_one` для этой строки соответствует значению value_one,

Предложение `WHERE` работает так же, как выбор, поэтому могут использоваться такие вещи, как `>` , `<` , `<>` или `LIKE` .

Примечание. В запросе удаления необходимо использовать условные предложения (WHERE, LIKE). Если вы не используете условные предложения, все данные из этой таблицы будут удалены.

Удалить все строки из таблицы

```
DELETE FROM table_name ;
```

Это удалит все, все строки из таблицы. Это самый простой пример синтаксиса. Он также показывает, что операторы `DELETE` действительно должны использоваться с особой осторожностью, поскольку они могут удалить таблицу, если `WHERE` опущено.

Удаление исключений

```
DELETE FROM `table_name` WHERE `field_one` = 'value_one' LIMIT 1
```

Это работает так же, как пример «Удалить с предложением», но он остановит удаление после ограниченного количества строк.

Если вы ограничиваете строки для удаления таким образом, имейте в виду, что он удалит первую строку, соответствующую критериям. Возможно, это не тот, который вы ожидаете, так как результаты могут возвращаться несортированными, если они явно не упорядочены.

Удаление нескольких таблиц

Оператор `DELETE` MySQL может использовать конструкцию `JOIN` , позволяющую также указать, какие таблицы удалить из. Это полезно для предотвращения вложенных запросов. Учитывая схему:

```
create table people
(    id int primary key,
    name varchar(100) not null,
    gender char(1) not null
);
insert people (id,name,gender) values
(1,'Kathy','f'),(2,'John','m'),(3,'Paul','m'),(4,'Kim','f');

create table pets
(    id int auto_increment primary key,
    ownerId int not null,
    name varchar(100) not null,
    color varchar(100) not null
);
insert pets(ownerId,name,color) values
(1,'Rover','beige'),(2,'Bubbles','purple'),(3,'Spot','black and white'),
(1,'Rover2','white');
```

Я бы	название	Пол
1	Кэти	е
2	Джон	м
3	Павел	м
4	Ким	е

Я бы	OwnerId	название	цвет
1	1	пират	бежевый
2	2	Пузыри	пурпурный
4	1	Rover2	белый

Если мы хотим удалить домашних животных Пола, заявление

```
DELETE p2
FROM pets p2
WHERE p2.ownerId in (
    SELECT p1.id
    FROM people p1
    WHERE p1.name = 'Paul');
```

можно переписать как:

```
DELETE p2    -- remove only rows from pets
FROM people p1
JOIN pets p2
ON p2.ownerId = p1.id
WHERE p1.name = 'Paul';
```

1 строка удалена
Пятно удалено из домашних животных

`p1` и `p2` являются псевдонимами для имен таблиц, особенно полезными для длинных имен таблиц и простоты чтения.

Чтобы удалить и человека, и животное:

```
DELETE p1, p2     -- remove rows from both tables
FROM people p1
JOIN pets p2
ON p2.ownerId = p1.id
WHERE p1.name = 'Paul';
```

Удалено 2 строки

Пятно удалено из домашних животных
Павел исключен из Людей

внешние ключи

Когда оператор DELETE включает таблицы с привязкой ключа foreing, оптимизатор может обрабатывать таблицы в порядке, который не следует за отношением. Добавление, например, внешнего ключа к определению домашних `pets`

```
ALTER TABLE pets ADD CONSTRAINT `fk_pets_2_people` FOREIGN KEY (ownerId) references people(id)
ON DELETE CASCADE;
```

двигатель может попытаться удалить записи от `people` до `pets` , что приведет к следующей ошибке:

```
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails
(`test`.`pets`, CONSTRAINT `pets_ibfk_1` FOREIGN KEY (`ownerId`) REFERENCES `people` (`id`))
```

Решением в этом случае является удаление строки от `people` и использование возможностей `ON DELETE` от `InnoDB` для распространения удаления:

```
DELETE FROM people
WHERE name = 'Paul';
```

Удалено 2 строки
Павел исключен из Людей
Пятно удаляется по каскаду от домашних животных

Еще одно решение - временно отключить проверку входящих ключей:

```
SET foreign_key_checks = 0;
DELETE p1, p2 FROM people p1 JOIN pets p2 ON p2.ownerId = p1.id WHERE p1.name = 'Paul';
SET foreign_key_checks = 1;
```

Основное удаление

```
DELETE FROM `myTable` WHERE `someColumn` = 'something'
```

Предложение `WHERE` является необязательным, но без него все строки удаляются.

УДАЛИТЬ ПРОГРАММЫ

```
TRUNCATE tableName;
```

Это приведет к удалению всех данных и сбросу индекса `AUTO_INCREMENT` . Это намного быстрее, чем `DELETE FROM tableName` на огромном наборе данных. Это может быть очень полезно при разработке / тестировании.

Когда вы ***усекаете*** таблицу, SQL-сервер не удаляет данные, он отбрасывает таблицу и воссоздает ее, тем самым освобождая страницы, поэтому есть возможность восстановить усеченные данные до перезаписанных страниц. (Пространство не может быть немедленно `innodb_file_per_table=OFF` для `innodb_file_per_table=OFF` .)

Multi-table DELETE

MySQL позволяет указать, из какой таблицы должны быть удалены соответствующие строки

```
-- remove only the employees
DELETE e
FROM Employees e JOIN Department d ON e.department_id = d.department_id
WHERE d.name = 'Sales'
```

```
-- remove employees and department
DELETE e, d
FROM Employees e JOIN Department d ON e.department_id = d.department_id
WHERE d.name = 'Sales'
```

```
-- remove from all tables (in this case same as previous)
DELETE
FROM Employees e JOIN Department d ON e.department_id = d.department_id
WHERE d.name = 'Sales'
```

Прочитайте УДАЛЯТЬ онлайн: https://riptutorial.com/ru/mysql/topic/1487/удалять

глава 73: Установите контейнер Mysql с помощью Docker-Compose

Examples

Простой пример с составлением докеров

Это простой пример создания сервера mysql с докере

1. создать **docker-compose.yml** :

Примечание. Если вы хотите использовать один и тот же контейнер для всех своих проектов, вы должны создать PATH в своем HOME_PATH. Если вы хотите создать его для каждого проекта, вы можете создать каталог **докеров** в своем проекте.

```
version: '2'
services:
  cabin_db:
    image: mysql:latest
    volumes:
      - "./.mysql-data/db:/var/lib/mysql"
    restart: always
    ports:
      - 3306:3306
    environment:
      MYSQL_ROOT_PASSWORD: rootpw
      MYSQL_DATABASE: cabin
      MYSQL_USER: cabin
      MYSQL_PASSWORD: cabinpw
```

2. - запустите его:

```
cd PATH_TO_DOCKER-COMPOSE.YML
docker-compose up -d
```

3. - подключение к серверу

```
mysql -h 127.0.0.1 -u root -P 3306 -p rootpw
```

Ура!!

4.- остановить сервер

```
docker-compose stop
```

Прочитайте Установите контейнер Mysql с помощью Docker-Compose онлайн:

https://riptutorial.com/ru/mysql/topic/4458/установите-контейнер-mysql-с-помощью-docker-compose

кредиты

S. No	Главы	Contributors
1	Начало работы с MySQL	A. Raza, Aman Dhanda, Andy, Athafoud, CodeWarrior, Community, Confiqure, Dipen Shah, e4c5, Epodax, Giacomo Garabello, greatwolf, inetphantom, JayRizzo, juergen d, Lahiru Ashan, Lambda Ninja, Magisch, Marek Skiba, Md. Nahiduzzaman Rose, moopet, msohng, Noah van der Aa, O. Jones, OverCoder, Panda, Parth Patel, rap-2-h, rhavendc, Romain Vincent, YCF_L
2	ALTER TABLE	e4c5, JohnLBevan, kolunar, LiuYan , Matas Vaitkevicius, mayojava, Rick James, Steve Chambers, Thuta Aung, WAF, YCF_L
3	ENUM	Philipp, Rick James
4	JSON	A. Raza, Ben, Drew, e4c5, Manatax, Mark Amery, MohaMad, phatfingers, Rick James, sunkuet02
5	MySQL LOCK TABLE	Ponnarasu, Rick James, vijeeshin
6	MySQL Союзы	Ani Menon, Rick James
7	mysqlimport	Batsu
8	TRIGGERS	Blag, e4c5, Matas Vaitkevicius, ratchet, WAF, YCF_L
9	UNION	Mattew Whitt, Rick James, Riho, Tarik, wangengzheng
10	Администратор MySQL	Florian Genser, Matas Vaitkevicius, RationalDev, Rick James
11	арифметика	Barranka, Dinidu, Drew, JonMark Perry, O. Jones, RamenChef, Richard Hamilton, Rick James
12	Безопасность через ГРАНТ	Rick James
13	Восстановить и сбросить пароль по умолчанию для MySQL 5.7+	Lahiru, ParthaSen
14	Восстановление с утраченного пароля root	BacLuc, Jen R

15	Временные таблицы	Ponnarasu, Rick James
16	Время с точностью до секунды	O. Jones
17	ВСТАВИТЬ	0x49D1, AbcAeffchen, Abubakkar, Aukhan, CGritton, Dinidu, Dreamer, Drew, e4c5, fnkr, gabe3886, Horen, Hugo Buff, Ian Kenney, Johan, Magisch, NEER, Parth Patel, Philipp, Rick James, Riho, strangeqargo, Thuta Aung, zeppelin
18	ВЫБРАТЬ	Ani Menon, Asjad Athick, Benvorth, Bhavin Solanki, Chip, Drew, greatwolf, Inzimam Tariq IT, julienc, KartikKannapur, Kruti Patel, Matthis Kohli, O. Jones, Ponnarasu, Rick James, SeeuD1, ThisIsImpossible, timmyRS, YCF_L, ypercube
19	Группа по	Adam, Filipe Martins, Lijo, Rick James, Thuta Aung, WAF, whrrgarbl
20	Двигатель MyISAM	Rick James
21	Динамическая сводная таблица с использованием подготовленного заявления	rpd
22	ЗАГРУЗКА ДАННЫХ	aries12, Asaph, bhrached, CGritton, e4c5, RamenChef, Rick James, WAF
23	Зарезервированные слова	juergen d, user2314737
24	Извлечение значений из типа JSON	MohaMad
25	Изменить пароль	e4c5, Hardik Kanjariya ツ, Rick James, Viktor, ydaetskcoR
26	Индексы и ключи	Alex Recarey, Barranka, Ben Visness, Drew, kolunar, Rick James, Sanjeev kumar
27	Информация о сервере	FMashiro
28	Использование переменных	kolunar, user6655061

29	Кластеризация	Drew, Rick James
30	Клиент MySQL	Batsu, Nathaniel Ford, Rick James
31	Коды ошибок	Drew, e4c5, juergen d, Lucas Paolillo, O. Jones, Ponnarasu, Rick James, WAF, Wojciech Kazior
32	Комментарий Mysql	Franck Dernoncourt, Rick James, WAF, YCF_L
33	Конфигурация и настройка	ChintaMoney, CodeWarrior, Epodax, Eugene, jan_kiran, Rick James
34	копирование	Ponnarasu
35	Лог-файлы	Drew, Rick James
36	Наборы символов и сортировки	frlan, Rick, Rick James
37	Настройка PS1	Eugene, Wenzhong
38	Настройка подключения SSL	4444, a coder, Eugene
39	Настройка производительности	e4c5, RamenChef, Rick James
40	НОЛЬ	Rick James, Sumit Gupta
41	ОБНОВИТЬ	4thfloorstudios, Chris, Drew, Khurram, Ponnarasu, Rick James, Sevle
42	Обработка временных зон	O. Jones
43	Обратные кавычки	Drew, SuperDJ
44	Ограничение и смещение	Alvaro Flaño Larrondo, Ani Menon, animuson, ChaoticTwist, Chris Rasys, CPHPython, Ian Gregory, Matt S, Rick James, Sumit Gupta, WAF
45	Один ко многим	falsefive
46	Операции даты и времени	Abhishek Aggrawal, Drew, Matt S, O. Jones, Rick James, Sumit Gupta
47	Ошибка 1055: ONLY_FULL_GROUP_BY: что-то не в условии	Damian Yerrick, O. Jones

	GROUP BY ...	
48	Падение таблицы	Noah van der Aa, Parth Patel, Ponnarasu, R.K123, Rick James, trf, Tushar patel, YCF_L
49	Подготовить заявления	kolunar, Rick James, winter
50	Подключение к UTF-8 с использованием различного языка программирования.	Epodax, Rick James
51	Полнотекстовый поиск	O. Jones
52	ПОСМОТРЕТЬ	Abhishek Aggrawal, Divya, e4c5, Marina K., Nikita Kurtin, Ponnarasu, R.K123, ratchet, Rick James, WAF, Yury Fedorov, Илья Плотников
53	Преобразование из MyISAM в InnoDB	Ponnarasu, Rick James, yukoff
54	ПРИСОЕДИНИТЕСЬ: Присоедините 3 таблицу с тем же именем id.	FMashiro
55	присоединяется	Artisan72, Batsu, Benvorth, Bikash P, Drew, Matt, Philipp, Rick, Rick James, user3617558
56	Работа с разреженными или отсутствующими данными	Batsu, Nate Vaughan
57	Разметка	Majid, Rick James
58	Регулярные выражения	user2314737, YCF_L
59	Резервное копирование с помощью mysqldump	agold, Asaph, Barranka, Batsu, KalenGi, Mark Amery, Matthew, mnoronha, Ponnarasu, RamenChef, Rick James, still_learning, strangeqargo, Sumit Gupta, Timothy, WAF
60	Сводные запросы	Barranka
61	Сделка	Ponnarasu, Rick James
62	События	Drew, rene
63	Советы по производительности	arushi, RamenChef, Rick James, Rodrigo Darti da Costa

	Mysql	
64	Создание баз данных	Daniel Käfer, Drew, Ponnarasu, R.K123, Rick James, still_learning
65	Создание таблицы	4444, Alex Shesterov, alex9311, andygeers, Aryo, Asaph, Barranka, Benvorth, Brad Larson, CPHPython, Darwin von Corax, Dinidu, Drew, fedorqui, HCarrasko, Jean Vitor, John M, Matt, Misa Lazovic, Panda, Parth Patel, Paulo Freitas, Přemysl Šťastný, Rick, Rick James, Ronnie Wang , Saroj Sasmal, Sebastian Brosch, skytreader, Stefan Rogin, Strawberry, Timothy, ultrajohn, user6655061, vijaykumar, Vini.g.fer, Vladimir Kovpak, WAF, YCF_L, Yury Fedorov
66	Создать нового пользователя	Aminadav, Batsu, Hardik Kanjariya ツ, josinalvo, Rick James, WAF
67	СОРТИРОВАТЬ ПО	Florian Genser, Rick James
68	Сохраненные процедуры (процедуры и функции)	Abhishek Aggrawal, Abubakkar, Darwin von Corax, Dinidu , Drew, e4c5, juergen d, kolunar, llanato, Rick James, userlond
69	Строковые операции	Abubakkar, Batsu, juergen d, kolunar, Rick James, uruloke, WAF
70	Таблица сопоставлений много-ко-многим	Rick James
71	Типы данных	Batsu, dakab, Drew, Dylan Vander Berg, e4c5, juergen d, MohaMad, Richard Hamilton, Rick James
72	УДАЛЯТЬ	Batsu, Drew, e4c5, ForguesR, gabe3886, Khurram, Parth Patel, Ponnarasu, Rick James, strangeqargo, WAF, whrrgarbl, ypercube, Илья Плотников
73	Установите контейнер Mysql с помощью Docker-Compose	Marc Alff, molavec