Разработка хранимых процедур и функций на сервере MySQL

Введение

- Хранимые подпрограммы – процедуры, функции и триггеры (procedure, function, trigger).
- Реализация концепции активного сервера и объектно-реляционного подхода.
- Хранимые подпрограммы являются объектами БД и хранятся в базе данных наряду с другими объектами (таблицами, представлениями, индексами и т.д.).
- Хранимые подпрограммы представляют собой набор процедурных команд и команд SQL.
- Обычно процедуру вызывают для того, чтобы выполнить некоторое действие, а функцию – для того, чтобы вычислить некоторое значение.
- Преимущества хранимых подпрограмм:
  - повышение продуктивности
  - улучшение производительности
  - экономия памяти
  - целостность приложений
  - повышение безопасности

Команды для работы с хранимыми подпрограммами

Название	Описание
CREATE PROCEDURE	создание процедуры
CREATE FUNCTION	создание функции
ALTER PROCEDURE	изменение процедуры
ALTER FUNCTION	изменение функции
DROP PROCEDURE	удаление процедуры
DROP FUNCTION	удаление функции
SHOW CREATE PROCEDURE proc_name	показать текст процедуры proc_name
SHOW CREATE FUNCTION func_name	показать текст функции func_name
SHOW PROCEDURE STATUS LIKE ‘proc_name’	показать характеристики процедуры proc_name
SHOW FUNCTION STATUS LIKE ‘func_name’	показать характеристики функции func_name
CALL proc_name()	вызвать процедуру proc_name
DECLARE	определение локальных переменных
SET	изменение значений локальных и глобальных переменных
SELECT … INTO	сохранение значения указанного столбца в переменную
IF	условный оператор if-then-else-end
CASE … WHEN	оператор выбора
LOOP,REPEAT,WHILE	циклы
RETURNS	возвращение значения из функции

Синтаксис создания хранимой процедуры или функции

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    PROCEDURE  sp_name
([proc_parameter[,...]])
    [characteristic ...]  routine_body

CREATE
    [ DEFINER = { user | CURRENT_USER }]
    FUNCTION  sp_name
      ( [func_parameter [,...] ] )
    RETURNS  type
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type

func_parameter:
    param_name  type
```

```
type:
    Any valid MySQL data type

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS  SQL
DATA | MODIFIES  SQL  DATA }
  | SQL  SECURITY { DEFINER | INVOKER }

routine_body:
    Valid  SQL  routine  statement
```

Создание хранимой процедуры или функции (1)

- Для создания хранимой подпрограммы необходимо иметь привилегию CREATE ROUTINE. Если двоичная регистрация допускается, эти инструкции могут также требовать привилегии SUPER.
- Создатель автоматически получает привилегии ALTER ROUTINE и EXECUTE.
- По умолчанию подпрограмма создается в заданной по умолчанию БД. Для создания подпрограммы в другой БД надо указать ее имя как db_name.sp_name
- Когда подпрограмма вызывается, выполняется неявное USE db_name (и отменяется по завершении). Инструкции USE внутри сохраненных подпрограмм недопустима.
- Список параметров подпрограммы заключается в круглые скобки. Если параметров нет, должен использоваться пустой список ().
- Для каждого параметра указывается тип данных (любой, кроме типа COLLATE).
- Каждый параметр имеет одну из трех мод:
  - IN – мода входного параметра (по умолчанию);
  - OUT – мода выходного параметра;
  - INOUT – мода входного и выходного параметра.
- **Обратите внимание**: определение параметра как IN, OUT или INOUT допустимо только для PROCEDURE! Параметры FUNCTION всегда расцениваются как IN.

Создание хранимой процедуры или функции (2)

- Значение по умолчанию (DEFAULT) присваивается в той ситуации, когда при вызове процедуры (функции) фактический параметр не передается.
- Предложение RETURNS может быть определено только для FUNCTION, для которой является обязательным и указывает тип значения, возвращаемого функцией.
- Моды параметров:
  - IN – параметр передает значение в процедуру. Процедура могла бы изменять значение, но модификация не видима вызывающему оператору, когда процедура завершается. Может быть переменной, константой или литералом.
  - OUT – параметр передает значение из процедуры обратно вызывающему оператору. Начальное значение внутри процедуры NULL, и значение видимо вызывающему оператору, когда процедура завершается. Может быть только переменной.
  - INOUT – параметр инициализирован вызывающим оператором, может изменяться процедурой, и последнее изменение, сделанное процедурой, видно вызывающему оператору, когда процедура завершается. Может быть только переменной.
- routine_body состоит из допустимой инструкции процедуры. Если инструкция состоит из нескольких команд, необходимого заключать ее в операторные скобки BEGIN и END.

Создание хранимой процедуры или функции (3)

- Начиная с MySQL 5.0.18, сервер использует тип данных стандартного параметра или функционального возвращаемого значения следующим образом (эти правила также относятся к локальным стандартным переменным, созданным инструкцией DECLARE):
  - Назначения проверены на предмет несоответствия типов данных и переполнение. Преобразование и проблемы переполнения приводит к предупреждениям или ошибкам в строгом режиме (см. *Справочный раздел* в конце данной презентации).
  - Для символьных типов данных, если имеется предложение CHARACTER SET в объявлении, используются определенный набор символов и заданное по умолчанию объединение. Если не имеется никакого такого предложения, используются наборы символов базы данных и объединение, которые были актуальными во время написания подпрограммы (они заданы значениями переменных системы character_set_database и collation_database).
  - Только скалярные значения могут быть присвоены параметрам или переменным. Например, инструкция типа SET x=(SELECT 1,2) глубоко ошибочна.
- Хранимые процедуры (но не функции!) могут содержать инструкции DDL, например, CREATE и DROP.
- Хранимые процедуры (но не функции!) могут содержать SQL-инструкции управления транзакциями (COMMIT, ROLLBACK и т.д.).

Создание хранимой процедуры или функции (4)

characteristic

- COMMENT может использоваться для описания подпрограммы. Эта информация отображается командами SHOW CREATE PROCEDURE и SHOW CREATE FUNCTION.
- DETERMINISTIC / NOT DETERMINISTIC - процедура или функция рассматривается как детерминированная, если она всегда производит тот же самый результат для тех же самых входных параметров, или недетерминированная в противном случае. Значение по умолчанию – NOT DETERMINISTIC . Подпрограмма, которая содержит функцию NOW() (или ее синонимы) или RAND() не детерминирована, но она может быть безопасна для репликации (см. раздел «Репликация»).
- CONTAINS SQL – указывает, что подпрограмма не содержит инструкции, которые читают или записывают данные. Это значение по умолчанию, если ни одна из этих характеристик не указана явно. Примерами таких операторов являются SET @x = 1 или DO RELEASE_LOCK('abc'),
- NO SQL указывает, что подпрограмма не содержит никаких инструкций SQL.
- READS SQL DATA указывает, что подпрограмма содержит инструкции, которые читают данные, но не инструкции, чтобы эти данные записывать.
- MODIFIES SQL DATA указывает, что подпрограмма содержит инструкции, которые могут записывать данные.

(Эти характеристики только консультативные. Сервер не использует их, чтобы ограничить то, какие виды инструкций подпрограмме позволено выполнить.)

Создание хранимой процедуры или функции (5)

characteristic

- SQL SECURITY указывает, должна ли подпрограмма выполняться с правами создателя этой подпрограммы или с правами пользователя, который ее вызывает (исполнителя). Значение по умолчанию: DEFINER. Создатель или исполнитель должен иметь права на обращение к БД, в которой она хранится, а исполнитель – привилегию EXECUTE.
- DEFINER определяет логин MySQL, который нужно использовать при проверке привилегий доступа в стандартном режиме выполнения для подпрограмм, которые имеют характеристику SQL SECURITY DEFINER (начиная с MySQL 5.0.20):
  - Значение user должно быть логином MySQL в формате 'user_name'@'host_name' (тот же самый формат используется в инструкции GRANT). Параметры user_name и host_name обязательны. CURRENT_USER также может быть указан как CURRENT_USER(). Значение по умолчанию для DEFINER – пользователь, который выполняет CREATE PROCEDURE, CREATE FUNCTION или инструкцию (аналогично DEFINER = CURRENT_USER).
  - Если пользователь не имеет привилегии SUPER, единственное допустимое значение user – его собственный логин, определенный буквально или используя CURRENT_USER. Нельзя устанавливать DEFINER к некоторому другому логину.
  - Если пользователь имеет привилегию SUPER, можно определять любой синтаксически допустимый логин. Если он фактически не существует, будет сгенерировано предупреждение.
- Возможно создать подпрограммы с несуществующим значением DEFINER. Ошибка возникает, если подпрограмма выполняется с привилегиями DEFINER, но сам DEFINER не существует во время выполнения.

Ограничения на хранимые функции

- Хранимые функции не могут содержать инструкции DDL, например, CREATE и DROP.
- Хранимые функции не могут содержать SQL-инструкции управления транзакциями (COMMIT, ROLLBACK и т.д.).
- Хранимые функции должны содержать внутри команду RETURN, возвращающую значение того типа, который указан для этой функции в предложении RETURNS.
- Возвращаемое функцией значение должно быть скалярным. Если инструкция RETURN возвращает значение иного типа (например, ENUM или SET), оно будет приведено к соответствующему скалярному типу элемента набора или объединения (число или строка).
- Инструкции, которые возвращают набор результатов, не могут использоваться внутри хранимой функции:
  - Это включает инструкции SELECT, которые не используют INTO, чтобы выбрать значения столбца в переменные, инструкции SHOW и другие инструкции, типа EXPLAIN.
  - Для инструкций, которые могут быть заданы при функциональном определении, но возвращают набор результатов, возникнет ошибка Not allowed to return a result set from a function (ER_SP_NO_RETSET_IN_FUNC).
  - Для инструкций, которые могут быть определены только во время выполнения, происходит ошибка PROCEDURE %s can't return a result set in the given context (ER_SP_BADSELECT).

Примеры создания подпрограмм (1)

Для создания простой **процедуры** в клиенте mysql.exe можно выполнить следующие операторы:

```
DELIMITER //
CREATE PROCEDURE Hello_World()
BEGIN
SELECT 'Hello, world!';
END
//
```

В данном примере первая строка (DELIMITER //) задает последовательность символов, которая будет использоваться в качестве признака завершения ввода. Обычно для этого используется точка с запятой, однако, в приведенном выше примере точка с запятой уже используется в качестве разделителя операторов в теле функции.

Запуск процедуры:

CALL Hello_World;

Примеры создания подпрограмм (2)

Пример создания функции:

```
DELIMITER //
CREATE  FUNCTION  func()
    RETURNS  INTEGER   -- определение типа возвращаемого значения
  BEGIN
    DECLARE  val  INTEGER;  -- описание переменной
    -- извлечение значения id из таблицы table1 в переменную val
    SELECT  id  INTO  val  FROM  table1;
 -- возврат значения val или 0, если val is null
    RETURN  IFNULL(val, 0);
  END
//
DELIMITER ;
```

Выполнение созданной функции:

```
SELECT func();
```

Синтаксис BEGIN ... END используется для записи составных инструкций.

Если запрос SELECT … INTO в подпрограмме возвращает пустой результат, это приведет к ошибке 1329 (No data). Если этот запрос содержит более одной строки, это приведет к ошибке 1172 (Result consisted of more than one row).

Примеры создания подпрограмм (3)

Количество строк, возвращаемых запросом, можно ограничить опцией LIMIT оператора SELECT.
Данная опция имеет два параметра. Первый параметр указывает смещение возвращаемого набора строк относительно начала, второй – количество возвращаемых строк. При использовании опции только с одним параметром он интерпретируется как количество возвращаемых строк от начала результата. Таким образом, совместно с оператором SELECT..INTO можно использовать опцию LIMIT 1.

Следующая процедура выводит наименование самой тяжелой детали:

```
CREATE PROCEDURE Heavy()
  BEGIN
    DECLARE S VARCHAR(20);
    SELECT weight INTO S FROM Parts ORDER BY Weight DESC LIMIT 1;
    SELECT(S);
  END
//
```

В процессе выполнения оператора SELECT..INTO выполняется неявное приведение типа возвращаемого запросом значения типу переменной.

Обратите внимание: использование LIMIT 1 не гарантирует возвращение строго одной строки. Если таблица пустая, то запрос не вернет данных, и возникнет ошибка No data. LIMIT 1 гарантирует возвращение *не более чем* одной строки.

Операторы управления ходом выполнения программы (1)

Оператор присваивания (установки значений):

```
SET <имя переменной> = <выражение>;
```

Разница между простыми и *системными переменными* в том, что системные переменные доступны извне хранимой процедуры. Системную переменную не нужно инициализировать. Разница в простой и системной переменной – в использовании префикса @ в имени системной переменной, например:

```
SET @S='Hello, world!';
```

Условный оператор IF..THEN имеет следующий синтаксис:

```
IF <условие> THEN <оператор  1>
   [ELSEIF <условие> THEN <оператор  2>]  ...
   [ELSE <оператор  3>]
ENDIF;
```

Оператор CASE имеет следующий синтаксис:

```
CASE case_value
   WHEN when_value THEN statement_list
   [WHEN when_value THEN statement_list] ...
   [ELSE statement_list]
ENDCASE;
```

Операторы управления ходом выполнения программы (2)

1. Самым простым оператором цикла является оператор LOOP, имеющий следующий синтаксис:

```
[Метка_начала:] LOOP
    <оператор>
END LOOP  [Метка_конца];
```

Операторы цикла выполняются до тех пор, пока внутри цикла не будет выполнен оператор `LEAVE <Метка_начала>`, который прерывает цикл с указанной меткой.

2. Оператор цикла с постусловием имеет следующий синтаксис:

```
REPEAT
    <оператор>
    UNTIL  <условие_выхода>
END  REPEAT;
```

Цикл будет выполняться до тех пор, пока условие выхода не станет истинным.

3. Оператор цикла с предусловием имеет следующий синтаксис:

```
WHILE  <условие_выполнения>  DO
   <оператор>
END  WHILE;
```

Цикл будет выполняться до тех пор, пока условие выполнения будет истинным.

Курсоры

Курсор представляет собой временную таблицу, получаемую в результате запроса, которая служит для построчной обработки данных. Курсор позволяет в цикле «перебирать» строки, выполняя над ними необходимые действия.
Для объявления курсора используется следующий оператор:

```
DECLARE <имя_курсора> CURSOR FOR <SQL-выражение>;
```

Прежде чем курсор может быть использован, его необходимо открыть:

```
OPEN <имя_курсора>;
```

После открытия указатель курсора устанавливается на первую строку. Для доступа к текущей строке открытого курсора используется оператор:

```
FETCH <имя_курсора> INTO<имя_переменной> [,<имя_переменной>] ...
```

Этот оператор помещает значения строки курсора в переменные, количество и типы данных которых соответствуют схеме (столбцам) курсора. После выполнения оператора FETCH происходит автоматическое продвижение на следующую строку курсора. Если более нет доступных строк (достигнута последняя строка) происходит изменение значения переменной SQLSTATE в 02000. Для обработки этого события необходимо установить обработчик: HANDLER FOR SQLSTATE '02000'.

Следующий оператор закрывает курсор:

```
CLOSE <имя_курсора>;
```

Если оператор закрытия курсора не указан явно, то курсор закрывается автоматически при закрытии соответствующего блока подпрограммы.

Курсоры. Пример

Процедура, изменяющая имя каждой детали с определенным именем на имя, формируемое как «Имя-N», где N – порядковый номер в списке всех деталей «Gasket» в порядке возрастания веса детали. Имя детали передается в качестве параметра.

```
CREATE PROCEDURE Parts_rename(PName VARCHAR(20))
BEGIN
  DECLARE Done INT DEFAULT 0;        DECLARE S VARCHAR(20);
  DECLARE N,I INTEGER;
  DECLARE Cur1 CURSOR FOR
    SELECT Part_ID, Part_name
       FROM Parts
       WHERE Part_name=PName ORDER BY WEIGHT;
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;
  OPEN Cur1;
  SET I=1;
  REPEAT
    FETCH Cur1 INTO N,S;
    IF Done = 0 THEN
       UPDATE Parts SET Part_name=CONCAT(S,'-',I) WHERE Part_ID=N;
    END IF;
    SET I = I+1;
  UNTIL Done END REPEAT;
  CLOSE Cur1;
  END;
```

Обработка ошибок (1)

Некоторые условия могут требовать специфической обработки. Эти условия могут касаться ошибок или общего управления потоком данных внутри подпрограммы.

```
DECLARE condition_name CONDITION FOR
            condition_value
condition_value:
    SQLSTATE [VALUE] sqlstate_value | mysql_error_code
```

Эта инструкция определяет условия, которые нуждаются в специфической обработке. Она сопоставляет имя с определенным условием ошибки. Имя может впоследствии использоваться в инструкции DECLARE HANDLER.
Здесь condition_value может быть значением SQLSTATE или же кодом ошибки MySQL.

```
DECLARE handler_type HANDLER FOR
            condition_value[,...] statement
handler_type:
            CONTINUE | EXIT | UNDO
condition_value:
            SQLSTATE [VALUE] sqlstate_value | condition_name |
            SQLWARNING | NOT FOUND | SQLEXCEPTION | mysql_error_code
```

Инструкция DECLARE ... HANDLER определяет обработчики, с каждым из которых может быть связано одно или большее количество условий (событий). Если одно из этих событий происходит, выполняется указанная инструкция statement . Инструкция может быть простой инструкцией (например, SET var_name = value), или составной инструкцией, записанной с помощью пары BEGIN и END.

Обработка ошибок (2)

Типы обработчиков:

CONTINUE: выполнение текущей подпрограммы продолжается после выполнения инструкции обработчика.

EXIT: выполнение завершается для составной инструкции BEGIN ... END, в которой объявлен данный обработчик. Это истинно, даже если условие происходит во внутреннем блоке.

Инструкция типа UNDO пока не реализованы.

Если возникает условие, для которого никакой обработчик не был объявлен, заданное по умолчанию действие: EXIT.

Параметр condition_value может быть любым из следующих значений:

- Значение SQLSTATE или MySQL-код ошибки.
- Имя условия, предварительно определенное с DECLARE ... CONDITION.
- SQLWARNING (краткая запись для всех кодов SQLSTATE, которые начинаются с 01).
- NOT FOUND (краткая запись для всех кодов SQLSTATE, которые начинаются с 02).
- SQLEXCEPTION (краткая запись для всех кодов SQLSTATE, не охваченных SQLWARNING или NOT FOUND).

Обработка ошибок (3)

```
mysql> CREATE TABLE test.t (s1 int, primary key (s1));
mysql> delimiter //
mysql> CREATE PROCEDURE handlerdemo ()
    -> BEGIN
    -> DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
    -> SET @x = 1;
    -> INSERT INTO test.t VALUES (1);
    -> SET @x = 2;
    -> INSERT INTO test.t VALUES (1);
    -> SET @x = 3;
    -> END;
    -> //
mysql> CALL handlerdemo()//
mysql> SELECT @x//            -- выдаст x=3
```

Пример сопоставляет драйвер с SQLSTATE 23000, который происходит для ошибки дублирования ключа. @x равен 3, это показывает что MySQL перейдет к концу процедуры. Если бы не было строки (`DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;`), MySQL принял заданный по умолчанию путь (EXIT) после второй неудачи INSERT из-за ограничения PRIMARY KEY, а SELECT @x возвратил бы 2. Если необходимо игнорировать условие, можно объявлять обработчик типа CONTINUE для этого и сопоставлять его с пустым блоком. Например:

```
DECLARE CONTINUE HANDLER FOR SQLWARNING BEGIN END;
```

Репликация данных в MySQL (1)

MySQL поддерживает 2 **типа** репликации: Master-Slave и Master-Master.

Основная – Master-Slave: все изменения происходят в Master БД, в Slave БД они попадают в асинхронном режиме. Slave БД используется для чтения.

При выходе из строя Master БД нужно переопределить Slave как Master, и потом заниматься восстановлением Master БД.

Master в дополнение к обычной работе из-за репликации формирует специальный лог-файл (binary log, =binlog), который потом будет рассылаться по сети.

На Slave есть специальный процесс (thread), который скачивает бинарные лог-файлы с master'а (или разных master'ов) и сохраняет их локально (relay log).

Проблемы синхронизации данных на Slave:

- отставание из-за асинхронного режима распространения изменений;
- влияние последовательности применения изменений на результат.

Более подробно о режимах репликации и ее настройке: Документация по MySQL. Глава 19. Репликация.

http://www.rldp.ru/mysql/mysql80/replica.htm#replication-options

Репликация данных в MySQL (2)

В MySQL существуют два основных **механизма** репликации данных:

Операторная репликация (Statement-based replication, SBR) записывает все запросы SQL, сделанные к главной (Master) БД, и воспроизводит их на подчинённых (Slave) серверах.

Достоинства:

1) высокая скорость;
2) небольшой объем передаваемых данных.

Недостатки: для недетерминированных команд не обеспечивает идентичность копий (например, RAND(), NOW() и т.д.).

Построчная репликация (Row-based replication, RBR) подсчитывает все изменения в главной (Master) БД и передает измененные данные на подчиненные узлы (Slave) БД.

Достоинства: поддерживает работу с недетерминированными командами, обеспечивая идентичность копий.

Недостатки:

1) низкая скорость;
2) большой объем передаваемых данных;
3) сложность с определениям границ транзакций.

Проблема, вызванная особенностями реализации MySQL: обработка транзакций зависит от типа таблицы – InnoDB или MyISAM.

Репликация данных в MySQL (3)

Параметры управления репликацией:

```
binlog_format = ROW | STATEMENT
```

ROW – репликация строк

STATEMENT – репликация команд

```
binlog_row_image = Full | Noblob | Minimal
```

Full – полные копии строк до и после обновления

Noblob – полные копии кроме ненужных полей типа BLOB

Minimal – только данные измененных полей + primary key строки

Идентификация транзакции происходит по Global TID – глобальному идентификатору транзакций. Для включения этой возможности д.б. установлены следующие параметры:

```
txn_ID = GTID
gtid_mode = ON
```

Как настроить репликацию:
https://ruhighload.com/%D0%9A%D0%B0%D0%BA+%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B8%D1%82%D1%8C+mysql+master-slave+%D1%80%D0%B5%D0%BF%D0%BB%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8E%3F

Список источников

- Сайт StudFiles. Практическое занятие 4. Разработка хранимых процедур и функций на сервере MySql. https://studfiles.net/preview/5339009/
- Афанасьев Юрий. Раскрываем магию MySQL или о строгости и мягкости MySQL. https://habr.com/post/166411/
- Аксенов Андрей. Как устроена MySQL-репликация. http://highload.guide/blog/mysql_replication_2015.html
- Репликация. http://www.rldp.ru/mysql/mysql80/replica.htm#replication-options

Справочный раздел

Системная переменная sql_mode

- MySQL сохраняет значение переменной системы sql_mode, которая была во время создания подпрограммы, и всегда выполняет подпрограмму именно с этим значением.
- Возможности SQL mode:
  1. Устанавливает строгую или мягкую проверку входных данных.
  2. Включает или отключает следование SQL стандарту.
  3. Обеспечивает лучшую синтаксическую совместимость с другими БД.
- Задание значения sql_mode возможно разными способами:

1. Соответствует значению по умолчанию, кавычки являются обязательными):

```
SET sql_mode = '';
```

2. Установка одного режима sql_mode. Возможно два варианта – с кавычками и без них:

```
SET sql_mode = ANSI_QUOTES;
SET sql_mode = 'TRADITIONAL';
```

3. Установка нескольких режимов sql_mode (указание кавычек является обязательным):

```
SET sql_mode = 'IGNORE_SPACE,ANSI_QUOTES';
SET sql_mode = 'STRICT_ALL_TABLES,ERROR_FOR_DIVIZION_BY_ZERO‘
```

4. Установка режима sql_mode для всего сервера (только для суперпользователя):

```
SET [GLOBAL|SESSION] sql_mode='параметры';
```

5. Запуск сервера с опцией `--sql-mode="<режимы>"`

6. Установка в файле my.cnf (для unix подобных систем) или my.ini (для windows) параметра `sql-mode="<режимы>"`

Переменная sql_mode. Краткий справочник режимов (1)

ANSI_QUOTES

Заставляет сервер интерпретировать двойную кавычку (") точно также, как и обратную кавычку (`), при этом она теряет способность обрамлять строки. Этот режим заставляет MySQL приблизиться к SQL стандарту. Пример:

```
mysql> CREATE TABLE test11 (`order` INT NULL)
ENGINE = InnoDB;
Query OK, 0 rows affected (0.28 sec)

mysql> CREATE TABLE test12 ("order" INT NULL)
ENGINE = InnoDB;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '"order" INT NOT NULL) ENGINE = InnoDB' at line 1

mysql> SET sql_mode = 'ANSI_QUOTES';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE test12 ("order" INT NULL)
ENGINE = InnoDB; Query OK, 0 rows affected (0.08 sec)
```

Переменная sql_mode. Краткий справочник режимов (2)

IGNORE_SPACE
По умолчанию, между функцией и открывающейся круглой скобкой нельзя устанавливать пробелы. Включение этого режима разрешает серверу игнорировать пробелы, но платой за такую вольность станет то, что все функции станут зарезервированными словами, а значит, при совпадении имени столбца с именем функции придётся в обязательном порядке экранировать такой столбец.

ONLY_FULL_GROUP_BY
Генерирует ошибку в запросах, в которых GROUP BY имеет не полный список не агрегированных параметров из SELECT и HAVING.

```
mysql>  SELECT name, address, MAX(age)
            FROM test
            GROUP BY name;
ERROR 1055 (42000): 't.address' isn't in GROUP BY

mysql>  SELECT name, MAX(age) as max_age
            FROM test
            GROUP BY name
            HAVING max_age < 30;
Empty set (0.00 sec) ERROR 1463 (42000): Non-grouping field 'max_age' is used in HAVING clause
```

Переменная sql_mode. Краткий справочник режимов (3)

ERROR_FOR_DIVISION_BY_ZERO
При делении на ноль в строгом режиме генерируется ошибка, а нестрогом — предупреждение при выполнении операторов INSERT или UPDATE. Без этого параметра деление на ноль возвращает предупреждение и вставляет в таблицу NULL.

STRICT_TRANS_TABLES
Включает «строгий режим» для всех таблиц, поддерживающих транзакции, т.е. на InnoDB и BDB. Этот режим возвращает ошибку, вместо предупреждения в следующих случаях:

1. Тип входных данных не соответствует заданному типу. Например, вставка строки в колонку с числовым типом.
2. Число или дата находится вне допустимого диапазона. Диапазон определяется типом данных. Например, для типа unsigned tinyint допустимым диапазоном являются числа от 0 до 255.
3. При вставке данных пропущено значение колонки, для которой не задано значение по умолчанию и имеет атрибут NOT NULL.
4. Длина значения выходит за пределы заданного диапазона. Например, для колонки типа CHAR(5) нельзя вставить строку более 5 символов.
5. Для типов ENUM и SET отсутствует вставляемое или обновляемое значение.

STRICT_ALL_TABLES
STRICT_ALL_TABLES полностью идентично STRICT_TRANS_TABLES, но действие режима уже распространяется на все таблицы MySQL, а не только на транзакционные.

Переменная sql_mode. Краткий справочник режимов (4)

TRADITIONAL

Композитный режим, включает в себя целый набор режимов, в который входит «строгий режим», а также ряд других режимов, налагающих ограничения на входные данные. Заставляет MySQL вести себя как большинство «традиционных» баз данных SQL. Полный перечень режимов, который содержит в себе данный режим:

```
mysql> SET sql_mode = 'TRADITIONAL';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@sql_mode\G
*************************** 1. row ***************************
@@sql_mode:
STRICT_TRANS_TABLES, STRICT_ALL_TABLES, NO_ZERO_IN_DATE,  NO_ZERO_DATE,
ERROR_FOR_DIVISION_BY_ZERO, TRADITIONAL,  NO_AUTO_CREATE_USER,
NO_ENGINE_SUBSTITUTION
```

ANSI

Другой композитный режим, делающий MySQL «ANSI-подобным», т.е. приближенным к стандарту SQL. Включает в себя следующие режимы:
REAL_AS_FLOAT, PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE.

REAL_AS_FLOAT – тип данных real является синонимом float, а не double.
PIPES_AS_CONCAT – разрешает использовать для конкатенации строк (||), вместо логического ИЛИ.

Переменная sql_mode. Краткий справочник режимов (5)

Строгий режим

При работе в строгом режиме (STRICT_TRANS_TABLES, STRICT_ALL_TABLES и композитный TRADITIONAL.) все входные данные проверяются с особой тщательностью и в случае нарушений любых ограничений возникает ошибка.

Ошибка в транзакционных таблицах (InnoDB) вызывает откат транзакции (rollback).

Для **не транзакционных таблиц** всё сложнее. Так, при вставке, обновлении или удалении нескольких строк в случае ошибки **отменяется только последнее действие** (вместо отката всей транзакции).

Генерация ошибки происходит в следующих случаях:

1. Тип вставляемых данных отличается от заданного типа столбца.
2. Опущено значение для столбца, которому не задано значение по умолчанию и имеет атрибут NOT NULL.
3. Для чисел и дат – данные находятся вне диапазона допустимых значений.
4. Для строк – превышение допустимой длины.
5. Для типов ENUM и SET – значение не является допустимым для заданного перечисления.
6. Для столбца, определённого как NOT NULL — вставка NULL.

Более подробно – https://habr.com/post/166411/