MySql-миграции: что это и как реализовать прост php-скриптом

Метки: рубрика "полезное" mysql php

Скачать исходники

Содержание статьи

Когда разрабатываешь веб-приложение не один, а в команде и/или на нескольких маши рано или поздно сталкиваешься с проблемой синхронизации кода проекта и базы данны Для управления кодом есть системы контроля версий, в частности, git, а для СУБД придуманы миграции.

Есть много готовых разнообразных инструментов, которые занимаются миграциями, но!

Очень часто все, что мы хотим - это просто залить в базу изменения, которые сделаны др разработчиком или же самим собой на другой машине. И желательно при этом затратив минимальные усилия, в том числе и на изучение и настройку незнакомой системы. К том далеко не всегда мы располагаем полным доступом к серверу для установки оных инструментов.

Поэтому всех, кому интересно узнать, как самим сделать простую утилиту миграций, наг полсотни строк php-кода, прошу в статью.

Идея миграций

Идея довольно проста: в проекте создаем отдельную папку sql, куда складываем sql-фай миграциями, то есть, со скриптами, которые меняют содержимое базы, а также один php файл, который эти миграции и накатывает.

Нужно учесть 2 вещи: во-первых, каждая миграция должна выполняться строго один раз во-вторых, в строго определенном порядке. Это разумно и обязательно, потому как если прилетают от коллеги 2 миграции, одна из которых создает таблицу users, а другая добавляет в нее тестовых пользователей, то мы хотим выполнить эти скрипты именно в таком порядке и не добавить при этом данные в базу больше одного раза.

Проблему повторных выполнений миграций мы решим, записывая в отдельную таблицу отработавшие скрипты, а порядок выполнения установим четкими правилами именован sql-файлов. Как это решается в коде, увидим чуть позже, а пока займемся подготовкой с тестовых миграций (исходники всех миграций и php-скрипта в конце статьи)

Создаем тестовые sql-скрипты

Пусть у нас есть тестовая база данных под названием test. Мы работаем с ней какое-то в и решили внедрить миграции. Есть разумное правило: самая первая миграция должна

Linux для веб-разработчика

jasmine, unit-тесты

содержать в себе полный дамп уже существующих сущностей в базе. Уточню: миграции помогают не только последовательно расширять уже существующую базу, но еще и нака эту самую базу с нуля, например, для новых людей в команде.

Перед началом выполнения миграций я предполагаю, что указанная база данных уже существует, поэтому в миграции нигде не указывается название базы. На мой взгляд, эт хорошо в том плане, что на одной машине программист может работать с разными базам разными их версиями. Поэтому у нас скрипта создания базы не будет.

В тестовом примере рассмотрим такой ход событий.
Первая миграция накатывает уже существующие данные и создает таблицу versions - он содержит уже выполненные скрипты.
Вторая создает таблицу users с тремя полями: id, email, password.
Третья добавляет в users три тестовых записи.
А четвертая добавляет в users новую колонку active.

Напишем первую миграцию: дамп базы и таблица versions. Пусть на момент внедрения миграций у нас есть таблица goods с парой товаров. Их нужно скинуть в скрипт и в тот же скрипт добавить таблицу versions. В итоге файл будет выглядеть так.

0000_base.sql

```sql
-- Дамп существующей базы --

-- Таблица товаров --
create table `goods` (
    `id` int(10) unsigned not null auto_increment,
    `name` varchar(255) not null,
    `price` int(11) not null,
    primary key (id)
)
engine = innodb
auto_increment = 1
character set utf8
collate utf8_general_ci;

-- Данные из таблицы товаров --
insert into `goods` (`name`, `price`) values
    ('Ноутбук', 30000),
    ('Телефон', 20000);

-- /Дамп существующей базы --


-- Таблица versions --
create table if not exists `versions` (
    `id` int(10) unsigned not null auto_increment,
    `name` varchar(255) not null,
    `created` timestamp default current_timestamp,
    primary key (id)
)
engine = innodb
auto_increment = 1
character set utf8
collate utf8_general_ci;
```

В файле мы видим структуру goods и 2 строчки с данными, а также versions. Эта таблица содержит 3 столбца: первичный ключ id, name - имя файла с миграцией, created - дата ее накатывания. Как Вы догадываетесь, в versions мы будем добавлять строки после накатывания новых миграций. А пока создадим оставшиеся 3 миграции.

0002_add_users.sql

```sql
-- Таблица пользователей --
create table if not exists `users` (
    `id` int(10) unsigned not null auto_increment,
    `email` varchar(255) not null,
    `password` varchar(255) not null,
    primary key (id)
)
engine = innodb
auto_increment = 1
character set utf8
collate utf8_general_ci;
```

0003_insert_data_into_users.sql

```sql
-- Добавляем тестовых пользователей --
insert into `users` (`email`, `password`) values
    ('test1@gmail.com', '111111'),
    ('test2@gmail.com', '222222'),
    ('test3@gmail.com', '333333')
```

0004_add_column_active_to_users.sql

```sql
-- Добавляем колонку active в users --
alter table `users`
    add column `active` tinyint(1) not null default 1 after `pas
```

На заметку: возможно, Вам не удобно писать sql-скрипты руками, Вы привыкли создава заполнять таблицы через phpMyAdmin или другой инструмент. Спешу успокоить, все известные мне утилиты позволяют генерировать такой sql-код автоматически. То есть В можете работать с базой, как удобно, а при подготовке файла-миграции вытащить нужн скрипт из условного phpMyAdmin-а в режиме copy-paste.

Обратим внимание на правила наименования файлов - это важная часть.

0001_base.sql
0002_add_users.sql
0003_insert_data_into_users.sql
0004_add_column_active_to_users.sql

Порядковые номера в начале каждого файла нужны, чтобы правильно отсортировать файлы. Только в этом случае миграции будут выполняться в правильном порядке. После идет краткое описание миграции, которое нужно исключительно для нашего удобства, ч понимать, что делает тот или иной скрипт, не заглядывая в него.

В нашем варианте мы предполагаем, что миграций не будет больше 9999 штук, и номер всегда должен состоять из 4-х цифр. Например, 10-я миграция будет называться 0010_description, а 101-я - 0101_description.

Теперь у нас все готово к написанию php-скрипта для выполнения миграций - migration.p

Пишем php-код для запуска миграций

Подумаем, что нам нужно уметь делать.

Во-первых, получить список всех sql-файлов из папки sql.
Во-вторых, понять, какие из них уже накатились ранее (сравнить этот список с тем, что лежит в таблице versions).
В-третьих, залить содержимое отобранных файлов в базу, заодно записав в versions их названия.

Пожалуй все, остальные операции довольно просты, разберем их по ходу пьесы.
Можно писать код.

Каркас migration.php, базовые константы и функции

Нам нужно написать фунцию подключения к базе данных и общую логику работы скрип
Создадим файл migration.php, кинем его в папку sql рядом с миграциями и напишем в нем следующее:

```php
// Объявляем нужные константы
define('DB_HOST', 'localhost');
define('DB_USER', 'root');
define('DB_PASSWORD', 'root');
define('DB_NAME', 'test');
define('DB_TABLE_VERSIONS', 'versions');


// Подключаемся к базе данных
function connectDB() {
    $errorMessage = 'Невозможно подключиться к серверу базы да
```

```
    $conn = new mysqli(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);
    if (!$conn)
        throw new Exception($errorMessage);
    else {
        $query = $conn->query('set names utf8');
        if (!$query)
            throw new Exception($errorMessage);
        else
            return $conn;
    }
}


// Получаем список файлов для миграций
function getMigrationFiles($conn) {
    // ...
}


// Накатываем миграцию файла
function migrate($conn, $file) {
    // ...
}


// Стартуем

// Подключаемся к базе
$conn = connectDB();

// Получаем список файлов для миграций за исключением тех, кото
$files = getMigrationFiles($conn);

// Проверяем, есть ли новые миграции
if (empty($files)) {
    echo 'Ваша база данных в актуальном состоянии.';
} else {
    echo 'Начинаем миграцию...<br><br>';

    // Накатываем миграцию для каждого файла
    foreach ($files as $file) {
        migrate($conn, $file);
        // Выводим название выполненного файла
        echo basename($file) . '<br>';
    }

    echo '<br>Миграция завершена.';
}
```

Пробежимся по коду.

Сначала объявляем константы для подключения к базе и с названием таблицы versions (
будет упоминаться в коде несколько раз).
Функция connectDB возвращает mysqli-объект для работы с базой.
getMigrationFiles($conn) возвращают список актуальных, еще не выполненых миграций.
migrate($conn, $file) накатывает одну миграцию - по имени файла.

Дальше сама логика работы.

Подключаемся к базе и получаем список актуальных миграций.
Если он пустой, то просто выводим сообщение, что база в актуальном состоянии.
Если файлы миграции имеются, то запускаем для каждого функцию migrate, выводя
название файла для информации. А в конце пишем позитивное сообщение, что миграци
завершена.

Теперь нам нужен код для getMigrationFiles()

```
// Получаем список файлов для миграций
function getMigrationFiles($conn) {
    // Находим папку с миграциями
    $sqlFolder = str_replace('\\', '/', realpath(dirname(__FILE
    // Получаем список всех sql-файлов
    $allFiles = glob($sqlFolder . '*.sql');

    // Проверяем, есть ли таблица versions
    // Так как versions создается первой, то это равносильно то
    $query = sprintf('show tables from `%s` like "%s"', DB_NAME
    $data = $conn->query($query);
    $firstMigration = !$data->num_rows;

    // Первая миграция, возвращаем все файлы из папки sql
    if ($firstMigration) {
```

```
        return $allFiles;
    }

    // Ищем уже существующие миграции
    $versionsFiles = array();
    // Выбираем из таблицы versions все названия файлов
    $query = sprintf('select `name` from `%s`', DB_TABLE_VERSIO
    $data = $conn->query($query)->fetch_all(MYSQLI_ASSOC);
    // Загоняем названия в массив $versionsFiles
    // Не забываем добавлять полный путь к файлу
    foreach ($data as $row) {
        array_push($versionsFiles, $sqlFolder . $row['name']);
    }

    // Возвращаем файлы, которых еще нет в таблице versions
    return array_diff($allFiles, $versionsFiles);
}
```

Почти каждая строка прокомментирована, но поясню еще подробнее.
В переменную $sqlFolder попадает полный абсолютный путь к текущему файлу migration. А соответственно, и к папке sql с миграциями. Замена \\ на / нужна для тех случаев, когд realpath возвращает путь с обратными слешами вместо прямых (например, в windows). Пе \ нужен для экранирования второго.

Дальше функция glob вытащит все файлы из указанной папки по нужной маске *.sql.

Затем нужно понять, не в первый ли раз мы собираемся накатывать миграции. Определи это можно только по наличию таблицы versions в базе. Мы же помним, что она создается самой первой миграции. Если запрос **show tables from test like "versions"** вернет пустой список, значит таблицы еще нет, в переменную $firstMigration запишем true и вернем из функции весь список файлов.

Если же таблица versions существует, то вытаскиваем из нее список файлов - уже выполненные миграции. И возвращаем из функции файлы, которых нет в таблице versio этом поможет array_diff.

Все, список нужных файлов мы получили, осталось понять, как закинуть их содержимое базу - функция migrate.

```
// Накатываем миграцию файла
function migrate($conn, $file) {
    // Формируем команду выполнения mysql-запроса из внешнего ф
    $command = sprintf('mysql -u%s -p%s -h %s -D %s < %s', DB_U
    // Выполняем shell-скрипт
    shell_exec($command);

    // Вытаскиваем имя файла, отбросив путь
    $baseName = basename($file);
    // Формируем запрос для добавления миграции в таблицу versi
    $query = sprintf('insert into `%s` (`name`) values("%s")',
    // Выполняем запрос
    $conn->query($query);
}
```

Из интересного в этой функции только shell_exec($command) - выполнение строки $comm терминале. Залить содержимое sql-файла в базу можно такой командой
mysql -uuser -ppassword -h host -D database < file.sql
Оную команду формируем с помощью sprintf и передаем ее в shell_exec. Собственно мигр накатилась, осталось добавить соответствующее название файла в таблицу versions - эт заурядный запрос вида
insert into `versions` (name) values('0001_base.sql')

И это весь код, который нужен нам для создания простой системы миграций. Теперь создадим пустую базу данных test, закинем, если еще не успели, папку sql куда-нибудь в проект и откроем в браузере sql/migration.php.

На заметку: если Вы не хотите на боевом сайте светить содержимым своих миграций возможным злоумышленникам, то поместите сами sql-миграции в место, недоступное дл просмотра извне. migration.php можно закинуть куда угодно, хоть в корень проекта, толь

придется немного поправить пути к папке с миграциями. Думаю, с этим делом Вы без тру справитесь сами :-)

Итак, migration.php в браузере запущен, если все сделано правильно, то увидим такую картину

```
Начинаем миграцию...

0001_base.sql
0002_add_users.sql
0003_insert_data_into_users.sql
0004_add_column_active_to_users.sql

Миграция завершена.
```

Посмотрите в базу и убедитесь, что нужные таблицы и данные созданы, в versions лежат строки с названиями файлов. Теперь обновите страницу в браузере и увидите сообщени "Ваша база данных в актуальном состоянии." Все правильно, все миграции сделаны, а но пока нет.

Чтобы окончательно убедиться в работоспособности системы, удалите все три таблицы базы и оставьте в папке sql только первую миграцию. Обновите страницу - накатится од миграция, таблица goods и versions с единственной строкой. Добавьте в папку миграции 0002_add_users.sql и 0003_insert_data_into_users.sql, обновите, выполнятся еще 2 миграции базе появится users с данными. И наконец последний файл и снова запуск - в users добав новая колонка active.

Итоги и исходники

Что же у нас получилось?
Все работает, как мы и задумали.
Теперь можно править базу, добавляя все изменения с ней в миграции, не боясь, что кто когда-то будет работать с неактуальной базой.

Главное, не забывать в нужные моменты при изменении базы создавать соответствующ миграции, соблюдая правила именования файлов, и запускать скрипт migration.php.

Как видим, код получился действительно короткий и простой (половину занимают комментарии). И при этом прямые свои обязанности он выполняет - накатывает миграци выполняя ничего лишнего.

Я намеренно не стал усложнять код, добавив обработку ошибок при накатывании скрип Также не рассматривал тему даунгрейда - так называемые обратные миграции, которые идее должны автоматически откатывать неудачные изменения к предыдущим версиям на практике это опасная процедура и ее все равно приходится делать руками.

Если Вы работаете над большим и долгим проектом с постоянным редактированием баз сотней программистов, то такая простейшая система скорее всего не подойдет. Нужна с из упомянутых выше серьезных утилит для работы с миграциями, где доступно больше возможностей и на проработана отказоусточивость.

Но не секрет, что в таких проектах уже давно подобная система используется, Вы може даже не знать, в какой момент сборщик накатывает новые миграции. Кстати, это можно сделать и при помощи моего любимого gulp-a, но это уже другая история, и возможно, те отдельной статьи.

Задавайте вопросы и оставляйте отзывы в комментариях, и конечно, скачивайте исходн

Скачать исходники

Метки: рубрика "полезное" mysql php

Понравилась статья? Поделись с другими!

2