Пакет для работы с СУБД Firebird в Laravel

Автор: Денис Симонов
17.09.2016

Оглавление

Пакет для работы с СУБД Firebird в Laravel

В прошлой статье я рассказывал о том, как можно добавить поддержку Firebird в Laravel. На тот момент я не знал о существовании пакета jacquestvanzuydam/laravel-firebird и добавлял поддержку Firebird с нуля. Сделано это было через модификацию файлов ядра Laravel, за что я был справедливо раскритикован. Посмотрев пакет jacquestvanzuydam/laravel-firebird, я понял, что его возможности меня не устраивают, и решил расширить его. В этой статье я хочу описать основные функциональные отличия моего пакета sim1984/laravel-firebird от пакета jacquestvanzuydam/laravel-firebird.

Поддержка автоинкрементных столбцов

Самым важным недостатком оригинального пакета является отсутствие поддержки автоинкрементных столбцов в миграции. В моём пакете поддержка автоинкрементных столбцов реализована двумя способами. Первый способ является классическим для Firebird. В этом способе при создании автоинкрементного столбца автоматически создаётся последовательность (генератор) и BEFORE INSERT триггер. Следующий скрипт на PHP

```
Schema::create('users', function (Blueprint $table) {
    $table->increments('id');
    $table->string('name');
    $table->string('email')->unique();
    $table->string('password');
    $table->rememberToken();
    $table->timestamps();
});
```

Сгенерирует и выполнит следующие SQL операторы.

```
CREATE TABLE "users" (
   "id"              INTEGER NOT NULL PRIMARY KEY,
   "name"            VARCHAR(255) NOT NULL,
   "email"           VARCHAR(255) NOT NULL,
   "password"        VARCHAR(255) NOT NULL,
   "remember_token"  VARCHAR(100),
   "created_at"      TIMESTAMP,
   "updated_at"      TIMESTAMP
 );

ALTER TABLE "users" ADD CONSTRAINT "users_email_unique" UNIQUE ("email");

CREATE SEQUENCE "seq_users";

CREATE OR ALTER TRIGGER "tr_users_bi" FOR "users"
ACTIVE BEFORE INSERT
AS
BEGIN
  IF (NEW."id" IS NULL) THEN
    NEW."id" = NEXT VALUE FOR "seq_users";
END
```

Второй способ работает, начиная с Firebird 3.0. В этом случае вместо последовательности и триггера используется IDENTITY поле. Следующий скрипт на PHP

```
Schema::create('users', function (Blueprint $table) {
    $table->useIdentity(); // only Firebird 3.0
    $table->increments('id');
    $table->string('name');
    $table->string('email')->unique();
    $table->string('password');
    $table->rememberToken();
    $table->timestamps();
});
```

Сгенерирует и выполнит следующие SQL операторы.

```
CREATE TABLE "users" (
   "id"              INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
   "name"            VARCHAR(255) NOT NULL,
   "email"           VARCHAR(255) NOT NULL,
   "password"        VARCHAR(255) NOT NULL,
   "remember_token"  VARCHAR(100),
   "created_at"      TIMESTAMP,
   "updated_at"      TIMESTAMP
 );

ALTER TABLE "users" ADD CONSTRAINT "users_email_unique" UNIQUE ("email");
```

Поддержка INSERT … RETURNING

Грамматика Firebird\Schema\Grammars\FirebirdGrammar расширена методом compileInsertGetId, который предназначен для сборки INSERT запроса с возвратом идентификатора только что добавленной строки.

```
/**
 * Compile an insert and get ID statement into SQL.
 *
 * @param  \Illuminate\Database\Query\Builder  $query
 * @param  array   $values
 * @param  string  $sequence
 * @return string
 */
public function compileInsertGetId(Builder $query, $values, $sequence) {
    if (is_null($sequence)) {
        $sequence = 'id';
    }

    return $this->compileInsert($query, $values) . ' returning ' . $this->wrap($sequence);
}
```

Тут обнаружилось, что INSERT … RETURNING не работает через PDO драйвер с Fireird 3.0. Раньше это проявлялось так https://bugs.php.net/bug.php?id=72931. На последних снапшотах поведение изменилось и теперь драйвер выдаёт ошибку SQLSTATE[HY000]: General error: -502 Cursor is not open. Очевидно, что PDO умеет возвращать данные только из курсоров (косвенно на это намекает метод PDOStatement::fetch()). Интересно, что это работало в firebird 2.5. Значит в API, которое должно быть совместимым, где-то произошли изменения, которые повлияли на работоспособность.

Я решил обмануть PDO, переработав запрос так, чтобы возвращался курсор. Для этого обернём наш оператор INSERT … RETURNING в анонимный PSQL блок (EXECUTE BLOCK). Тут есть одна неприятная особенность. Дело в том, что для поддержки именованных параметров PDO делает замену всех переменных вида :VARNAME на «?».

Это портит содержимое тело анонимного блока. Такая замена работала бы правильно, если бы делалась только между ключевыми словами EXECUTE BLOCK и AS. Другим способом является замена маркера переменных, как это сделано в некоторых других компонентах доступа. К сожалению PDO не имеет возможности изменить маркер переменных. Поэтому пришлось искать способ избегать символа «:» внутри тела блока. Поскольку это надо делать только для Firebird 3.0, я выделил для него отдельную грамматику FirebirdGrammar30, и добавил специальный метод для определения версии Firebird. Кроме того, отдельная грамматика позволит нам лучше использовать новые возможности Firebird 3.0. Приведу код, который фиксит баг с INSERT … RETURNING

```
    /**
     * Fix PDO driver bug for 'INSERT ... RETURNING'
     * See https://bugs.php.net/bug.php?id=72931
     * Reproduced in Firebird 3.0 only
     * Remove when the bug is fixed!
     *
     * @param \Illuminate\Database\Query\Builder $query
     * @param array $values
     * @param string $sequence
     * @param string $sql
     */
    private function fixInsertReturningBug(Builder $query, $values, $sequence, $sql)
    {
        /*
         * Since the PDO Firebird driver bug because of which is not executed
         * sql query 'INSERT ... RETURNING', then we wrap the statement in
         * the block and execute it. PDO may not recognize the colon (:) within
         * a block properly, so we will not use it. The only way I found
         * buyout perform a query via EXECUTE STATEMENT.
         */
        if (!is_array(reset($values))) {
            $values = [$values];
        }
        $table = $this->wrapTable($query->from);
        $columns = array_map([$this, 'wrap'], array_keys(reset($values)));
        $columnsWithTypeOf = [];
        foreach ($columns as $column) {
            $columnsWithTypeOf[] = "  {$column} TYPE OF COLUMN {$table}.{$column} = ?";
        }
        $ret_column = $this->wrap($sequence);

        $columns_str = $this->columnize(array_keys(reset($values)));

        $new_sql = "EXECUTE BLOCK (\n";
        $new_sql .= implode(",\n", $columnsWithTypeOf);
        $new_sql .= ")\n";
        $new_sql .= "RETURNS ({$ret_column} TYPE OF COLUMN {$table}.{$ret_column})\n";
        $new_sql .= "AS\n";
        $new_sql .= "  DECLARE STMT VARCHAR(8191);\n";
        $new_sql .= "BEGIN\n";
        $new_sql .= "  STMT = '{$sql}';\n";
        $new_sql .= "  EXECUTE STATEMENT (STMT) ({$columns_str})\n";

        if (!$query->getConnection()->getPdo()->inTransaction()) {
            // For some unknown reason, there is a ROLLBACK. Probably due to the COMMIT
RETAINING.
            $new_sql .= "  WITH AUTONOMOUS TRANSACTION\n";
        }
        $new_sql .= "  INTO {$ret_column};\n";
        $new_sql .= "  SUSPEND;\n";
        $new_sql .= "END";

        return $new_sql;
    }

    /**
     * Compile an insert and get ID statement into SQL.
     *
     * @param  \Illuminate\Database\Query\Builder  $query
     * @param  array   $values
     * @param  string  $sequence
     * @return string
     */
    public function compileInsertGetId(Builder $query, $values, $sequence)
```

```
    {
        $sql = parent::compileInsertGetId($query, $values, $sequence);
        // Fix PDO driver bug for 'INSERT ... RETURNING'
        // See https://bugs.php.net/bug.php?id=72931
        $sql = $this->fixInsertReturningBug($query, $values, $sequence, $sql);

        return $sql;
    }
```

> **Замечание**
>
> Мне совершенно не нравится данный способ. Надеюсь, что в будущем баг будет исправлен и это временное решение можно будет удалить.

Работа с последовательностями

Иногда возникает потребность работать с последовательностью сгенерированной не с помощью миграций Laravel, например, вы можете работать с уже готовой базой данных. В ряде случае одна и та же последовательность может использоваться несколькими таблицами. Вы можете использовать в своих моделях произвольное имя последовательности, воспользовавшись расширенной моделью Firebird\Eloquent\ Model. В этой модели присутствует дополнительное свойство $sequence, которое сдержит имя необходимой последовательности. Пример:

```
use Firebird\Eloquent\Model;


class Customer extends Model
{
    /**
     * таблица связанная с моделью
     * @var string
     */
    protected $table = 'CUSTOMER';

    /**
     * Первичный ключ модели
     * @var string
     */
    protected $primaryKey = 'CUSTOMER_ID';

    /**
     * Indicates if the model should be timestamped.
     * @var bool
     */
    public $timestamps = false;

    /**
     * имя последовательности для генерации первичного ключа
     * @var string
     */
    protected $sequence = 'GEN_CUSTOMER_ID';
}
```

В этой модели переопределён метод insertAndSetId, который не использует INSERT … RETURNING, а получает следующий номер последовательности и использует его в

обычном запросе INSERT. Использование этой модели позволяет также не использовать не слишком красивое решение INSERT … RETURNING в Firebird 3.0.

Как я уже говорил, последовательности являются полностью самостоятельными объектами метаданных в Firebird, поэтому неплохо бы иметь возможность управлять ими через миграции Laravel. Для этого был написан класс Firebird\Schema\ SequenceBlueprint. Давайте посмотрим как это работает на примере вот такой миграции.

```php
<?php

use Firebird\Schema\Blueprint;
use Firebird\Schema\SequenceBlueprint;
use Illuminate\Database\Migrations\Migration;

class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::createSequence('seq_users_id');

        Schema::create('users', function (Blueprint $table) {
            $table->integer('id')->primary();
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });


        Schema::sequence('seq_users_id', function (SequenceBlueprint $sequence) {
            $sequence->increment(5);
            $sequence->restart(10);
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropSequence('seq_users_id');

        Schema::drop('users');
    }
}
```

Накат такой миграции приведёт к выполнению следующих SQL операторов

```
CREATE SEQUENCE "seq_users_id";

CREATE TABLE "users" (
    "id"              INTEGER NOT NULL,
    "name"            VARCHAR(255) NOT NULL,
    "email"           VARCHAR(255) NOT NULL,
    "password"        VARCHAR(255) NOT NULL,
    "remember_token"  VARCHAR(100),
    "created_at"      TIMESTAMP,
    "updated_at"      TIMESTAMP
);

ALTER TABLE "users" ADD PRIMARY KEY ("id");

ALTER TABLE "users" ADD CONSTRAINT "users_email_unique" UNIQUE ("email");

ALTER SEQUENCE "seq_users_id" RESTART WITH 10 INCREMENT BY 5;
```

Откат миграции выполнит следующие операторы:

```
DROP SEQUENCE "seq_users_id";

DROP TABLE "users";
```

Дополнительные параметры конфигурации

В нашем пакете добавлено два дополнительных параметра конфигурации для настройки подключения:

- role – имя роли с которой произойдёт подключение к базе данных;
- engine_version – версия Firebird. Этот необязательный параметр позволяет не делать дополнительный запрос к серверу для определения версии Firebird.

Пример:

```
    'connections' => [

        'firebird' => [
            'driver' => 'firebird',
            'host' => env('DB_HOST', 'localhost'),
            'port' => env('DB_PORT', '3050'),
            'database' => env('DB_DATABASE', 'examples'),
            'username' => env('DB_USERNAME', 'BOSS'),
            'password' => env('DB_PASSWORD', 'qw897tr'),
            'role' => 'RDB$ADMIN',
            'charset' => env('DB_CHARSET', 'UTF8'),
            'engine_version' => '3.0.0',
        ],

    ],
```

Особенности установки через composer

Поскольку мой пакет является форком пакета jacquestvanzuydam/laravel-firebird, то его установка несколько отличается. Как и при установки оригинального пакета, не забывайте ставить в composer.json параметр minimum-stability равный dev. Далее необходимо добавить ссылку на репозиторий:

```
    "repositories": [
        {
            "type": "package",
            "package": {
                "version": "dev-master",
                "name": "sim1984/laravel-firebird",
                "source": {
                    "url": "https://github.com/sim1984/laravel-firebird",
                    "type": "git",
                    "reference": "master"
                },
                "autoload": {
                    "classmap": [""]
                }
            }
        }
    ],
```

После чего добавьте в параметр require следующую строку:

```
"sim1984/laravel-firebird": "dev-master"
```

Заключение

Я надеюсь, что мой пакет имеет достаточный функционал для разработки приложений с использований СУБД Firebird. Если у вас есть вопросы и предложения по улучшению пакета sim1984/laravel-firebird пишите в личку, обязательно отвечу. В следующей статье я расскажу о том, как создать небольшое приложение с использованием Laravel и СУБД Firebird.