Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Metodika pre prácu s databázou

Tím DILEMA

Členovia tímu: Bc. Zuzana Bachárová
Bc. Peter Bakoš
Bc. Martin Baláž
Bc. Marek Bernád
Bc. Imrich Nagy
Bc. Peter Tibenský
Bc. Juraj Volko
Vedúci projektu: Ing. Ján Lang, PhD.
Tím: č. 10 [FIIT DU]
Akademický rok: 2017/2018
Dátum poslednej zmeny: 9.12.2017
Vypracoval: Marek Bernád

1 Úvod

Účelom tejto metodiky je opísať spôsoby práce s databázou a jej spojazdnenia počas lokálneho vývoja a testov od spustenia DB servera až po naplnenie tabuliek údajmi.

2 Postup

Hlavné pokyny pre vytvorenie tabuliek databázy a jej naplnenie sú okrem tohto dokumentu definované aj v triede "*database/seeds/DatabseSeeder.php*". Pre celkové spojazdnenie databázy je potrebné vykonať nasledovné kroky [1. a 2. = inštalácia, 3. = reset DB, nekonzistentná DB, 4. = bežný proces pred prácou na Git vetve, vhodný najmä na produkcii]:

1. V prípade, že ešte nie je dostupný žiadny serverový databázový PSQL klient v pc, je potrebné si stiahnuť najnovšiu verziu EnterpriseDB, PostgreSQL podľa používanej softvérovej platformy - OS tu (pre Windows, Linux treba 64 bitový inšt. súbor): https://www.enterprisedb.com/downloads/postgres-postgresql-downloads#windows

   Pri inštalácií je potrebné zaškrtnúť možnosť spúšťania DB serveru už pri štarte, v opačnom prípade si ho treba spustiť ručne pred každou prácou s databázou takto:
   LINUX, MacOS: https://www.enterprisedb.com/docs/en/9.5/pg/server-start.html
   WINDOWS:https://stackoverflow.com/questions/36629963/how-to-start-postgresql-on-windows

2. Po nainštalovaní aplikačného rámca Laravel do špecifického priečinku v root je potrebné, ak ešte neexistuje, vytvoriť textový súbor *.env*, ktorý je kópiou súboru *.env.example*. Do tohto súboru je potrebné dodať nasledujúce konfiguračné riadky:

   ```
   DB_CONNECTION=pgsql
   DB_HOST=127.0.0.1
   DB_PORT=5432
   DB_DATABASE=dilema
   DB_USERNAME=postgres
   DB_PASSWORD=postgres
   ```

   V prípade, že pri inštalácií databázového servera EDB PostgreSQL je definovaný iný základný databázový port, je potrebné ho v tomto súbore zmeniť na aktuálny, rovnako ako používané meno a heslo. Je nutné, aby súbor *.env* bol definovaný v súbore *gitignore*.

3. Príkazy tohto tretieho kroku sú dôležité v prípade, že je potrebné obnoviť celkovú databázu, alebo v iných špecifických prípadoch ako napr. pri vzniku inak neriešiteľných chýb a nekonzistencie databázy. Pre tento krok **od bodu b)** je podmienkou mať aktuálnu verziu GIT-master vetvy. Následne v priečinku, kde je

nainštalovaný Laravel v root spustím CMD/Terminál alebo použijem terminál vývojového prostredia, napr. v PhpStorm. Je potrebné zadať postupne nasledovné príkazy do terminálu:

a) `php artisan migrate:reset`
   [toto treba spustiť ešte predtým, než si dám pull z mastra alebo svojej branche, rollbackne to všetky tabuľky a odstráni vám problém toho, ak niekto iný zmenil meno nejakej migrácie v mastri, ak si dáte pull skôr, musíte si odstrániť tabuľky z DB ručne a tento krok môžete preskočiť]

b) `composer dump-autoload`
   [možno to chvíľu potrvá, ale toto je potrebné, ak mám starú verziu databázy (a hlavne nové nepushnuté seeds) a dát v priečinku projektu a v serverovej lokálnej databáze, napr. ak som predtým už používal túto databázu]

c) `php artisan migrate:refresh --seed`
   [tento príkaz vykoná migrácie = vytvorí na lokálnom serveri v databáze nové tabuľky a vďaka príkazu --seed ich aj naplní údajmi, dokonca aj základného admin používateľa = netreba registráciu]

   Tento proces bodu 3. je potrebné opakovať vždy, ak idem ďalej pracovať v novej vetve na novej iterácií. Pre základný login cez databázu do systému je vhodné použiť tieto prihlasovacie údaje bez potreby registrácie:

   mail: *fiittim10@gmail.com*
   heslo: *dilema*

4. Príkazy, ktoré je potrebné zadať do terminálu pre priečinok projektu pre update DB:

   a) `composer update`
   b) `php artisan migrate`

2.1 Spôsob práce s tzv. “seeds” *[ref]*

V hlavnom priečinku root pre Laravel, v ktorom je náš projekt, sa nachádza konkrétny priečinok “database/seeds”, v ktorom sa nachádzajú všetky triedy, ktoré svojou funkcionalitou napĺňajú existujúce databázové tabuľky.

Tvorba seedov

1. Na začiatok je potrebné vytvoriť v priečinku “database/seeds” novú triedu, ktorej názov je nasledovný: “**NazovTabulky**Seeder.php”. Teda pri tabuľke user_types máme triedu “UserTypesSeeder.php”.

2. Do tejto vytvorenej triedy, ktorá dedí triedu Seeder, do metódy run() vložím konkrétnu funkciu table triedy DB pre naplnenie tabuľky takto:

```
DB::table('nazov_mojej_tabulky_presne_ako_v_DB') ->insert([
      'nazov_atributu1' => "hodnotu_kt_ulozim_ako_varchar",
      'timestamp_atribut2'    =>    Carbon::now()->format('Y-m-d
H:i:s'),
      'teraz_nepotrebny_atribut_pre_sprint_kt_je_null' => "" ,
      'atribut_len_cislo_napr_FK' =>  34 ,
      'atribut_s_hodnotou_dolaru_v_stringu' => "Mam 10 \$ a som
frajer"
]);
```

   Keďže aj PHP má svoje predefinované symboly ako napr. “$” pre ich vypísanie treba použiť backslash.

3. Do už existujúcej triedy “database/seeds/DatabaseSeeder.php” je na koniec potrebné pridať vždy do metódy run() aj názov tejto novo pridanej triedy, nového seedu, takto:

```
$this->call(NazovTabulkySeeder.php::class);
```

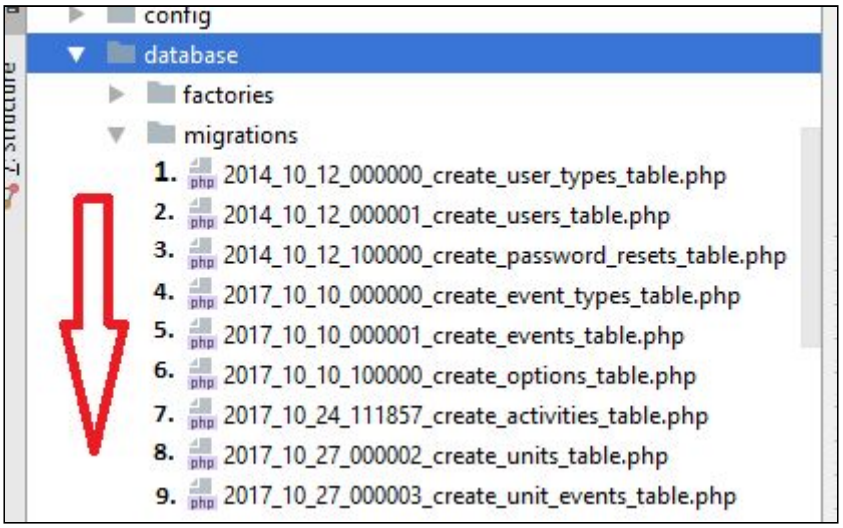
4. Pre otestovanie novo pridaného seedu nestačí spustiť len príkaz *php artisan migrate:refresh --seed,* ale ako prvé treba najprv: *composer dump-autoload* .

2.2 Spôsob práce s tzv. “migrations” *[ref]*

Pre prácu s migráciami je potrebné vedieť ako funguje spustenie migrácie. Už vyššie opísaným príkazom sa nám spustí migrácia, kde nastane proces vykonávania funkcii up() všetkých tried v priečinku “database/migrations”.

2.2.1 Migrácie všeobecne (+ tvorba a mazanie tabuliek)

Čo je najdôležitejšie vedieť pri vytváraní migrácií je vykonanie poradia migrácií, ktoré sa vykonáva od vrchu nadol. Viď obr. 1:



Obr. 1 Poradie vykonania migrácií

Poradie tried v tomto vykonaní určuje ako aj dátum, tak aj šesťciferné číslo za dátumom, podľa ktorého sú triedy usporiadané v poradí, aké vidíme na obrázku. Toto je dôležité preto, aby bolo možné si určovať poradie týchto tried, podľa vzoru: "Year_Month_Day_Num_create_**table_name**_table.php" kde upravujeme najmä Num pre poradie a meno, o aké vytvorenie tabuľky sa jedná. Poradie je dôležité kvôli cudzím "FK" kľúčom, kde nie je možné vytvoriť tabuľku, ktorá sa nedokáže naviazať týmto kľúčom na ID existujúcej. Čiže na vrchu musia byť postupne všetky také triedy migrácie / tabuľky, ktoré sa vykonajú skôr, ako tie, ktoré na ne naviažu svoje cudzie kľúče.

Migrácie môžu slúžiť na vytváranie, mazanie a taktiež aj úpravu(update - alter table) tabuliek.

Názvy tried migrácií:

Pre vytváranie akýchkoľvek tried migrácií je potrebné nazvať triedu migrácie presne ako aj v názve .php súboru migrácie, a to ako:

class Create[TableName]Table extends Migration { … }

alebo

class Update[TableName]Table extends Migration { … }

2.2.1 Migrácie - úprava existujúcich tabuliek *[ref]*

Tieto typy migrácií - migračných súborov je vhodné použiť pre jednorazové použitie po získaní novej verzie z GIT-u pre aktualizovanie aktuálneho stavu spojazdnenej databázy. Tieto migrácie je možné spustiť obyčajným príkazom:

php artisan migrate

Tento príkaz prejde všetky nevykonané - neuplatnené migrácie pre aktuálnu verziu (stav) lokálnej databázy a vykoná ich. Týmto spôsobom sa získava aktuálny stav databázy. Tvorba takýchto migračných tried je vhodná vždy po aktualizovaní databázového modelu, teda po jeho zmene. Pri prvotnej práci s týmto druhom migrácie po spustení "php artisan migrate" môžu vzniknúť chyby pre neaktuálne knižnice, a je potrebné spustiť príkaz "*composer require doctrine/dbal*".

Význam:

- Ak potrebujem upraviť existujúce databázové tabuľky bez straty ich aktuálnych dát
- Šetrenie operácií mazania, znovu vytvárania tabuliek a spúšťania seedov
- Dôležité pre vývoj už bežiaceho systému

Využitie:

- Úprava názvu existujúceho stĺpca databázovej tabuľky
- Pridanie nového stĺpca
- Zmazanie stĺpca
- Pridanie indexov a nastavenie primárneho klúča, tiež cudzieho kľúča
- Zmena názvu tabuľky

Názvoslovie-syntax:

Vzor:

[year]_[month]_[day]_[6-digitOrderNumber]_**update**_[table_name]_**table**.php

Príklad:

2017_11_14_000001_update_event_types_table.php

Dôležité pravidlá vytvárania:

1. Migrácie pre update tabuliek sa z pravidla budú pomenúvať tak, aby bolo ich umiestnenie situované pod tabuľkou, ktorú upravujú (za migráciu vytvorenia samotnej tabuľky). Toto je dôležité pre prípad, že si niekto resetuje-refreshuje databázu, aby nenastala úprava stĺpcov neexistujúcich tabuliek.
2. V obsahu tejto migrácie sú opäť metódy up() a down(). Je konvenciou v metóde up() vytvárať zmeny tabuľky, a na druhej strane v down() jej zmeny vrátiť do pôvodného stavu. Avšak program by chybou končiť nemal, ak metóda down() nie je vyplnená.

Špecifické prípady:

Ak v predošlom šprinte bola vytvorená update migrácia pre danú tabuľku, a chcem opäť použiť túto migráciu pre úpravu napr. ďalšieho stĺpca, je potrebné upraviť túto existujúcu migráciu. Pri opätovnom zavolaní príkazu "*php artisan migrate*" sa rozpozná táto migrácia ako nová a nanovo sa vykoná jej obsah, ktorý ešte doposiaľ neupravil aktuálnu databázu.

Príklad triedy pre tento typ migrácie:

```
1.  class UpdateEventsTable extends Migration
2.  {
3.      public function up()
4.      {
5.          Schema::table('events', function (Blueprint $table) {
6.              $table->string('newColumn')->nullable();
7.          });
8.
9.          Schema::table('events', function (Blueprint $table) {
10.             $table->renameColumn('header','caption');
11.         });
12.     }
13.
14.     public function down()
15.     {
16.         Schema::table('events', function (Blueprint $table) {
17.             $table->dropColumn('newColumn');
18.         });
19.
20.         Schema::table('events', function (Blueprint $table) {
21.             $table->renameColumn('caption','header');
22.         });
23.     }
24. }
```

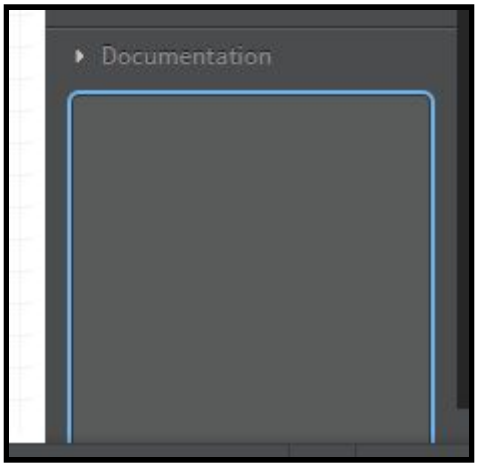
2.3 Spôsob práce so softvérom StarUML a štruktúra modelu

Ak sa vytvorí nová tabuľka alebo akákoľvek UML entita, ale zistíš, že ju už nepotrebuješ, nestačí ju označiť kurzorom a stlačiť DEL, nakoľko zostane stále v source control a vzniká tam zmätok. Natrvalo sa dá odstrániť vyznačením kurzura a kombináciou CTRL+DEL.

Každý vzťah N:N medzi dvoma tabuľkami je potrebné rozdeliť na väzobnú tabuľku s dvoma cudzími kľúčmi. Pri každej novo pridanej tabuľke je potrebné definovať nasledovné údaje:

1. Prvý atribút zvaný "id", ktorý je primárnym kľúčom PK s typom INTEGER
2. Zvyšné atribúty so svojimi typmi
3. Cudzie kľúče
4. Každá tabuľka má povinne atribút v modely uvedený ako timestamp (implementácia v migrácií: `$table->timestamps();`), kde sa toto správanie projektuje do databázy ako vytvorenie dvoch stĺpcov a to: "*created_at*" a "*updated_at*"
5. Pre každý atribút, ktorý má byť unikátny, je dôležité ho tak označiť ako U, kde v DB potom nebudú existovať duplicity hodnoty tohto atribútu

6. Dokumentáciu vpravo dole, kde je potrebné napísať predpokladané príklady obsahu tabuľky (napr. units sú cvičenia, seminár, labáky…, activities sú vyučovacie predmety) a načo slúži:



Obr. 2 Dokumentácia v StarUML

2.3.1 Názvoslovie tabuliek - pravidlá

1. Názov tvorený z malej abecedy
2. Viac slovné názvy sa spájajú podtržníkom (napr. user_types)
3. Názvy píšeme v množnom čísle: users,events, units, activities …
4. Názvy píšeme v anglickom jazyku ako aj ich atribúty

2.4 Povinnosti developera (CRUD) entity DB modelu

Každý, kto pridá novú tabuľku v databázovom fyzickom modeli, je zodpovedný za dokončenie nasledovných úloh:

1. Vytvorenie **modelu** tejto entity v laraveli do priečinka “app”
2. Naviazanie cudzích kľúčov medzi novo pridaným modelom a existujúcimi modelmi v priečinku “app” [**ORM** = belongsTo(), hasMany() ...]
3. Vytvorenie celej **migrácie** pre túto entitu modelu
4. Vytvorenie **seedu** pre naplnenie tejto tabuľky
5. Otestovanie funkčnosti predošlých bodov
6. Vytvoriť zmeny v dokumentácií, ak nastali aj v modeli