LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

OBJEKTRELĀCIJU KARTĒŠANAS IZVEIDE ZEPHIR VALODĀ

BAKALAURA DARBS

Autors: **Ričards Zālītis**
Studenta apliecības nr.: rz11017
Darba vadītājs: M.ekon. Gatis Sprūds

RĪGA 2015

ANOTĀCIJA

Darbā tiek apskatīta Zephir programmēšanas valoda un iespējas, ko tā paver, objektrelācijas kartēšana - kas tā tāda ir un kāpēc tā ir vajadzīga - izstrādātā vienuma funkciju apskats, nepieciešamības pamatojums un to salīdzinājums ar esošajiem risinājumiem.

Risinājums ir izgatavots ar mērķi to izmantot turpmāk praksē, lai atvieglotu un uzlabotu darbu ar datubāzēm, protams, to papildinot ar praksē gūtām zināšanām. Tas ir paredzēts darbā ar MySQL datubāzes vadības sistēmu un izmantot kopā ar PHP, jo risinājums par pamatu izmanto PHP PDO klasi.

Gala produkts ir SimpleORM - strādājošs objektrelācijas kartēšanas risinājums, kas ir izstrādāts Zephir programmēšanas valodā un kas nodrošina nepieciešamās pamatfunkcionalitātes darbā ar MySQL, lai to varētu uzskatīt par labu esam.

Atslēgvārdi: Zephir, objektrelācijas kartēšana, SimpleORM, MySQL, PHP, PDO

ABSTRACT

Takes a look on Zephir programming language and features it opens, answer what is object relational mapping and why is it necessary, look on developed extension functions and necessity of them, extension comparison with existing solutions.

Solution is developed to be used futher in pratice to ease and improve work with databases, of course with improving in mind. It is designed to be used with MySQL database managment systems and PHP because solution will extend PHP PDO class.

The end product is SimpleORM - working object relational mapping solution who has been developed in Zephir to cover all the core functions to be able to work with MySQL so the solution can be considered good.

Keywords: Zephir, object relation mapping, SimpleORM, MySQL, PHP, PDO

SATURS

APZĪMĒJUMU SARAKSTS

Zephir – atklātā pirmkoda programmēšanas valoda, kas ir paredzēta PHP paplašinājumu izstrādei un kurā ir pieejami statiski datu tipi.

ORM - *Object-relational mapping* – objektorientētās programmēšanas tehnikā kā sasaistīt datus starp dažāda veida sistēmām.

PHP - atklātā pirmkoda skriptu valoda, kas tiek izmantota tīmekļu aplikācijās.

MySQL - relāciju datubāzu vadības sistēma.

PDO - *PHP Data Object*s – PHP klase, kas definē datubāzes datu piekļuves abstrakcijas slāni.

VirtualBox – programmatūra, kas nodrošina virtualizāciju.

SimpleORM – Zephir valodā izstrādātais objektrelācijas risinājums.

CRUD – *create, update, delete* – akronīms, kas tiek lietots, lai aprakstītu pamatfunkcijas, kas nepieciešamas datu glabātuvēm.

IEVADS

Programmēšanā bieži nākās darboties ar datubāzēm un programmētāji izstrādā sistēmas, kas nemitīgi pieprasa datus no datubāzes vai sūta datus uz datubāzi un ir konstantā darbībā ar to. Izstrādājot sistēmas, datu nolasīšana nesagādā nekādas galvassāpes, jo parasti datu nolasīšanai ir nemainīgs vaicājums, kas ir neatkarīgs no sistēmas lietotāja ievada. Galvassāpes sākās, kad datubāzē ir jāievieto lietotāja ievadīti dati, jo šie dati ir pilnībā neparedzami un, ja tiem nepievērš pietiekami lielu uzmanību, tad sistēmas datubāze ir apdraudēta, jo to ir iespējams bojāt ar nefiltrētu datu ievadi.

Lai manipulētu datus, programmētājiem ir jāraksta SQL vaicājumi vai cita veida vaicājumi, kuri ir jāizpilda pret datubāzi. Šie vaicājumi ir diezgan pamatīgi un nepārskatāmi, parasti tajos nākas iedziļināties sīkāk, jo nav iespējams pateikt, ko tas dara un kāda ir tā jēga un nākas patērēt vairāk laika nekā vajadzētu. Ja izstrādātā sistēma sastāv tikai no šādiem zema līmeņa vaicājumiem, tad noteikti ir vietas, kur tiek kopēti šie paši vaicājumi, jo nekas nav objektorientēts un sistēmas kods paliek arvien nepārskatāmāks.

Bakalaura darba mērķis ir izveidot vienkāršu, viegli uztveramu un ātri strādājošu objektrelācijas kartēšanu, kas tiks realizēta Zephir programmēšanas valodā, mēģinot atrisināt problēmas ar kurām ir nācies saskarties praksē, datu manipulācija notiktu pēc vienota principa un izskaustu zemu līmeņu SQL vaicājumu rakstīšanu.

Mērķa sasniegšanai izvirzītie uzdevumi:

1. Noskaidrot, kas ir objektrelāciju kartēšana.
2. Apzināt tipiskās problēmas ar ko sastopas praksē.
3. Noskaidrot informāciju par Zephir programmēšanas valodu.
4. Izstrādāt objektrelācijas kartēšanas risinājumu Zephir valodā.
5. Salīdzināt izveidoto risinājumu ar esošajiem risinājumiem.
6. Izdarīt secinājumus par izveidoto objektrelācijas kartēšanu.

1. OBJEKTRELĀCIJU KARTĒŠANA

Datorzinātnēs objektrelāciju kartēšana ir objektorientētās programmēšanas tehnika, lai sasaistītu datus starp dažāda tipa sistēmām, radot virtuālu objektu datubāzi, kura var tikt izmantota izvēlētajā programmēšanas valodā, lai efektīvi darbotos ar otru sistēmu, parasti tā ir kāda no datubāzes sistēmām [1].

Galvenā problēma, ko risina objektrelāciju kartēšana, ir objektu sasaiste ar datubāzes ierakstiem, saglabājot objekta atribūtus un relācijas. Objektu var atkārtoti iegūt ar tiem pašiem atribūtiem un relācījām, jo objekta informācija turpina eksistēt datubāzē pēc programmas darbības beigām. Viena no pamatfunkcijām ir tās saucamais „CRUD" – create, update, delete. Objektus ar kartēšanas palīdzību jāvar:

1. Izveidot un saglabāt.
2. Iegūt atkārtoti ar objekta pēdējām saglabātajām izmaiņām.
3. Veikt izmaiņas objektā un saglabāt tās.
4. Izdzēst objektu no datubāzes.

Ir pieejami daudz un dažādi objektrelācijas kartēšanas risinājumi gan atvērtā koda, gan komerciālie. Lielākā problēma ar šiem risinājumiem ir, ka tiem ir pieejamas daudz par daudz funkciju no kurām lielākā daļa nemaz netiek izmantotas, jo tās ir pārāk specifiskas un lielākoties var iztikt bez tām, dažkārt pat radot lieku noslodzi. Otra problēma ir tā, ka nav zināms kā tiek realizētas funkcijas un kas patiesībā notiek „zem pārsega", bet kopumā ieguvums no objektrelācijas kartēšanas ir milzīgs, tapēc nav nekāds pārsteigums, ka mūsdienās katrs PHP ietvars nāk ar iekļautu objektrelācijas kartēšanu, kas lielākoties ir radīts speciāli šim ietvaram.

Ieguvumi no objektrelācijas kartēšanas:

1. Nav jāraksta zema līmeņa vaicājumi
2. Nav jāuztraucas par datu ievadu, jo ievads tiek filtrēts
3. Objektorientēta pieeja
4. Samazinās koda apjoms
5. Uzlabojās pārskatāmība un uzturamība
6. Samazinās izstrādes laiks

1.1. Tipiskās problēmas

No paša pieredzes, izstrādājot tīmekļa bāzētas sistēmas ar PHP, ir nācies saskarties ar visādiem brīnumiem, kurus ir mērķis izskaust vai mazināt ar izstrādājamo objektrelācijas kartēšanu.

1.1.1. Zema līmeņa vaicājumi

Viena no lielākajām problēmām ar ko ir nācies saskarties, ir zema līmeņa SQL vaicājumi. Apjomīgi vaicājumi ir pierakstīti vienā garā virknē un pēc tam vel caur dažādiem loģiskajiem nosacījumiem mainīti, līdz beigās vispār ir tikai neliela nojausma kādu vaicājumu satur šī virkne. It kā jau nekas briesmīgs, jo kods strādā un dara paredzēto, tikai lasāmība mazinās un grūtāk atkļūdot to visu, bet, ja vaicājums tiktu veidots objektorientēti, tad uzlabotos lasāmība un vieglāk būtu saprast vaicājuma jēgu.

1.1.2. Loģikas dublēšana

Otra diezgan nopietna problēma – visu laiku tiek kopēta daļa no vaicājumu loģikas, kurai būtu jābūt nemainīgai un izplatīta pa visurieni. Problēma ir tajā, ja tomēr ir kļūda vai nākās papildināt šo „dzelžaino" loģiku, tad nākas iet cauri bez maz vai visai sistēmai un koriģēt vaicājumu loģiku, kas nemaz nav viegli izdarāms un parasti paliek nepamanītas vietas.

1.1.3. Neziņa par realizāciju

Mazākā problēma – nav ne jausmas, kā tiek realizētas objektrelācijas funkcijas. Akli izpilda funkcijas, jo tās ir „pareizas" un strādā kā vajag, bet programmēšanā vienmēr pastāv iespēja, ka tomēr ir pieļauta kāda kļūda un, ja pastāv šī kļūda kādā no funkcijām, tad to ir grūti izlabot, jo nav zināšanas par kartēšanas realizāciju. Ir divas lietas, ko darīt šādos gadījumos:

1. Gaidīt ietvara atjauninājumu vai ielāpu.
2. Apgūt realizācijas kodu un mēģināt novērst kļūdu pašam.

Abas metodes ir laikietilpīgas, lai gan var gadīties, ka pirmais variants ir ātrāks, jo ietvara izstrādātāji ir informāti par šo problēmu un uzreiz novērš to.

1.1.4. Datu ievads

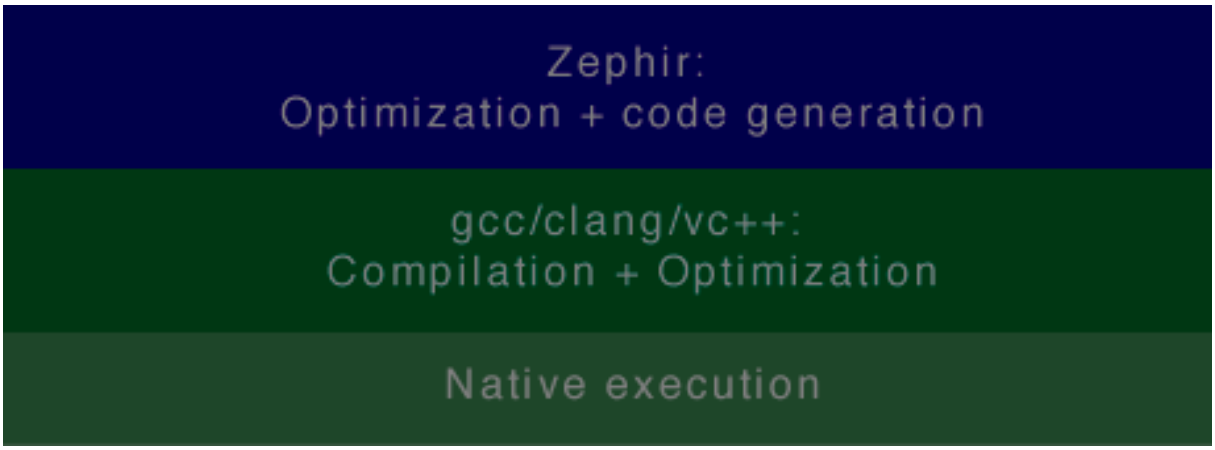
Problēma ar ko saskaras ikviens – lietotāja ievads. Vispār datu ievads ir tāda interesanta lieta, jo uz to nekad nevar paļauties, jo vienmēr kāds tiešām vai netiešām ievadīs informāciju, kas var kaitēt sistēmas darbībai, tāpēc lietotāja ievads vienmēr ir jāfiltrē. Ar ievada filtrāciju var izvairīties no SQL injekcijām, tiesa gan, parasti visos PHP ietvaros, ja tiek lietots ietvara ORM, filtrācija tiek izmantota pašā ORM un par to nav tik daudz jādomā, ja vien neraksta zema līmeņa vaicājumus, kur jāizmanto lietotāja ievadīta informācija.

1.1.5. Nevajadzīga funkcionalitāte

Tāda ne pārāk nopietna problēma, bet tāpat ņemama vērā – nevajadzīga funkcionalitāte. Ļoti bieži gadās, ka izmantojot ietvara ORM, objektiem ir pārāk daudz funkcijas un lielākoties nevajadzīgas, kas nekad netiek izmantotas, galu galā, tāpat tiek izmantotas pamatfunkcijas un pārējās vienkārši traucē pilnvērtīgi apgūt ORM.

1.2. Zephir

Zephir ir atvērtā koda programmēšanas valoda, kas ir domāta, lai atvieglotu PHP papildinājumu izveidi un uzturamību, koncentrējoties uz datu tipiem un atmiņas drošību. Zephir ģenerē C kodu, ko pēc tam kompilē uz mašīnkodu, tādā veidā ir iespējams paslēpt realizāciju.



1.1. att. Zephir kompilācijas shēma[3]

Sakarā ar to, ka Zephir veido C kodu, tad tas ir savietojams ar PHP un līdz ar to Zephir var izmantot jebkuras PHP funkcijas, klases un sintakse ir ļoti līdzīga PHP, tāpēc Zephir valodu var diezgan viegli apgūt jebkurš, kas kādreiz ir darbojies ar PHP.

PHP programmētājs bez problēmām var lasīt Zephir kodu, ātri uztvert kontekstu un salīdzinoši viegli pārveidot PHP kodu uz Zephir kodu. Viena būtiska lieta, kas jāņem vērā, ir, ka, lai mainītu mainīgā vērtību, tad priekšā jāliek atslēgvārds „let", jo mainīgie pēc noklusējuma ir nemainīgi un „let" atslēgvārds tos padara mainīgus.

Sakarā ar to, ka Zephir kompilējās, tad izstrādes laiks palielinās, jo ir jāgaida kamēr kompilējās kods un atkļūdošana uzreiz aizņem vairāk laika. Izstrādājot nopietnas funkcijas ir ieteicams tās vispirms izstrādāt PHP un tad pārnest uz Zephir valodu, jo loģikas kļūdas ir vieglāk novērst PHP, jo tiek izlaists kompilēšanas solis.

Kopumā var teikt, ka Zephir ir kaut kas pa vidu starp C un PHP, izmantojot abu valodu dotās priekšrocības.

1.2.1. Datu tipi

Zephir valodā ir dinamisks datu tips un statiski datu tipi. Dinamiskais datu tips darbojās tieši tāpat kā PHP, tas var pieņemt jebkādu datu tipu bez nekādiem ierobežojumiem. Kopumā Zephir ir pieejami vienpadsmit datu tipi no kuriem desmit ir statiski datu tipi un viens dinamisks.

1.1 tabula

Zephir pieejamie datu tipi

Datu tips	Apraksts
Var	Dinamisks datu tips, var pieņemt jebkādu datu tipu.
Boolean	Reprezentē patiesumvērtības.
Integer	Reprezentē veselus skaitļus ar zīmi, vismaz 16 biti.
unsigned integer	Reprezentē veselus skaitļus bez zīmes, vismaz 16 biti.
float/double	Reprezentē peldošā punkta skaitļus.
Char	Reprezentē mazāko pieejamo mašīnvērtību, kas var pieņemt kādu no pamatsimboliem.
unsiged char	Tas pats, kas char, tikai bez zīmes.
Long	Reprezentē veselus skaitļus ar zīmi, vismaz 32 biti.
unsigned long	Reprezentē veselus skaitļus bez zīmes, vismaz 32 biti.
Array	Datu struktūra, kas var tikt izmantota kā masīvs, vārdnīca, vektors ...
String	Reprezentē simbolu virkni.

Tāpat kā visās programmēšanas valodās šiem tipiem ir ierobežojumi, piemēram, skaitļiem ir noteiktas minimālās un maksimālās vērtības. Statiskie datu tipi piešķir Zephir savu burvību, jo bieži vien PHP pietrūkst skalāru vērtību datu tipi, bet Zephir nav šādas problēmas.

1.2.2. Ierobežojumi

Pašlaik pastāv sava veida ierobežojumi uz Zephir- tas pašlaik nav pieejams uz Windows vides un visa izstrāde ir jāveic Linux vidē. Kompilējot Zephir kodu, gala produkts ir viena datne ar .so paplašinājumu un to uzreiz var pievienot PHP papildinājumu sarakstam un izmantot to, tiesa gan, tikai uz Linux vidēm, bet pārsvarā visi nopietni serveri ir linux vidē un tā nav liela problēma.

Pārnesot PHP kodu uz Zephir, var gadīties, ka kaut kas neiet, jo Zephir nav pieejams atslēgvārds „static” un atrast aizvietojumu šim atslēgvārdam nebūt maz nav viegli.

1.2.3. Programmēšana Zephir

Zephir programmas kodu ir iespējams rakstīt arī uz Windows, jo viss, kas vajadzīgs, lai rakstītu, ir teksta redaktors, bet, lai kompilētu uzrakstīto kodu ir nepieciešama Linux vide uz kuras ir uzlikts tīmekļa serveris un Zephir instalācija.

Vienkāršākais veids kā uzstādīt Zephir ir caur Linux termināli, izpildot sekojošās komandas [1]:

1. sudo apt-get update
2. sudo apt-get install git gcc make re2c php5 php5-json php5-dev libpcre3-dev
3. git clone https://github.com/phalcon/zephir
4. cd zephir
5. ./install-json
6. ./install –c

Kad šīs komandas ir izpildītas, ir jāpārbauda vai viss ir veiksmīgi noritējis, ierakstot terminālī komandu „zephir help”, ja viss ir pareizi uzstādījies, tad terminālī izvadīsies informācija par Zephir.

Tagad var sākt veidot savu PHP paplašinājumu. Pirmais, kas jāizdara – jāatrod vēlamā projektējuma novietne, izmantojot Linux termināli, kad tas izdarīts, termināli jāievada komanda „sudo zephir init _”, kur _ ir papildinājuma nosaukums un Zephir izveidos pamata struktūru, kas nepieciešama, lai pabeigtu papildinājumu. Vērtīgs ieteikums [7]:

1. cd /etc/php5/mods-available
2. sudo cp pdo.ini _.ini
3. sudo nano _.ini
4. Aizvietot extension=pdo.so ar extension=X/_/ext/modules/_.so, kur X – vieta, kur tika rakstīts zephir init _

Kad paplašinājums ir uzrakstīts un gatavs kompilēšanai, izpildīt sekojošās komandas:

1. sudo php5dismod _
2. sudo service apache2 reload
3. sudo zephir build
4. sudo php5enmod _
5. sudo service apache2 reload

Pirmās divas komandas var izlaist pirmajā kompilācijas procesā. Šīs ir pamata komandas, kas jāzina, lai veiksmīgi kompilētu un varētu testēt papildinājumu. Lai pārliecinātos par procesa veiksmīgumu, var ierakstīt komandu: php –m | grep _ un pārliecināties vai paplašinājums ir veiksmīgi pievienots PHP. Izveidotās klases būs uzreiz pieejamas PHP.

2. SimpleORM

Paplašinājuma izstrādes ciklu var iedalīt trijos lielos etapos. Pirmais ir izstrādāt paplašinājumu PHP valodā, jo kā jau tika minēts nodaļā 1.2, tad izstrādes un kļūdu labošanas ātrums PHP valodā ir daudz labāks nekā Zephir valodā. Otrais etaps ir pārveidot PHP kodu uz Zephir kodu un nokompilēt to. Trešais lielais etaps - kļūdu novēršana, tas ir, labot kompilatora dotos kļūdu paziņojumus un ieviest Zephir valodā lietas, kuras vēl nav pieejamas.

Pirmajā izstrādes etapā tika ņemtas vērā 1.1 nodaļā apskatītās tipiskās problēmas ar mērķi novērst tās. Pirmais izstrādes etaps, kā jau paredzēts, norit diezgan raiti un noslēdzās ar paplašinājuma PHP versiju.

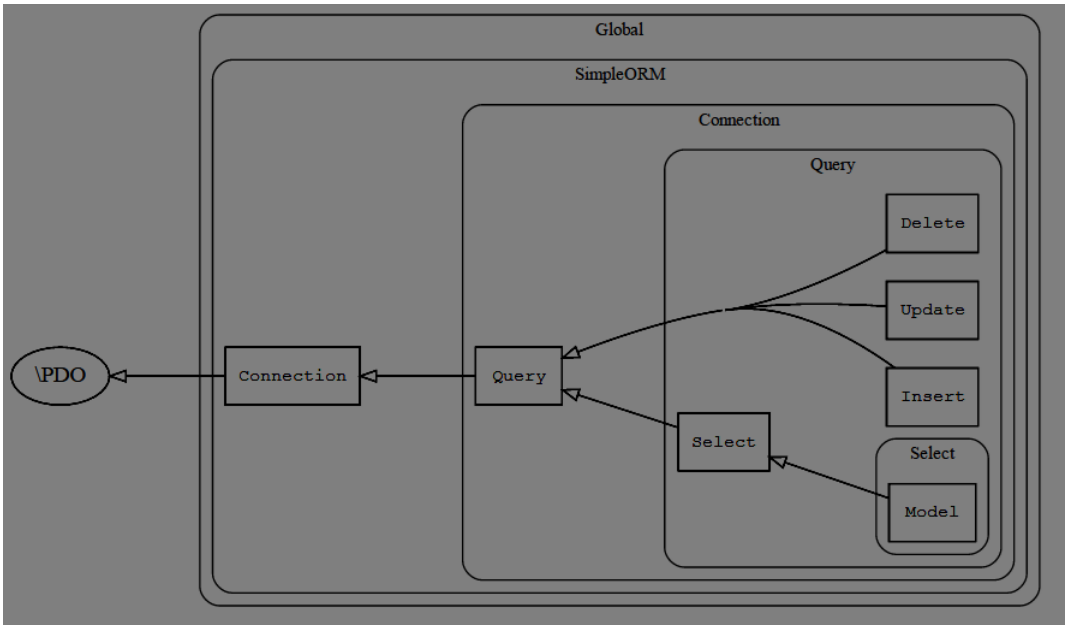
Pirms varēja uzsākt otro izstrādes procesu, nācās izveidot izstrādes vidi. Izstrādes vide tika realizēta windows operetājsistēmā, izmantojot VirtualBox. VirtualBox deva iespēju imitēt Linux vidi uz kuras tika izpildīts 1.2.3 nodaļā aprakstītais Zephir uzstādīšanas process [6].

Otrajā izstrādes procesā PHP versija tika pārveidota uz Zephir valodai atbilstošu sintaksi. Otrais posms arī noritējā samērā ātri, jo PHP kodu ir ļoti viegli pārveidot uz Zephir kodu.

Trešais izstrādes etaps bija vistrakākais un laikietilpīgākais, jo labojumu rezultāts ir redzams tikai pēc paplašinājuma pārkompilēšanas. Viena no lielākajām problēma bija izdomāt kā aizvietot PHP atslēgvārdu „static”, jo Zephir valodā tas vēl nav pieejams [5], bet, galu galā, tas izdevās pateicoties tam, ka Zephir kompilējot vispirms izveido C kodu, kurā varēja skatīties un meklēt kļūdas.

Paplašinājumā visas iekļautās klases un to metodes ir izstrādātas ar domu, ka tās visas ir nepieciešamas un bez tām nevar iztikt. Nosaukums SimpleORM radās ar domu, ka radītais risinājums nesaturēs neko lieku un būs viegli uztverams.

Paplašinājums par pamatu izmanto PHP PDO klasi un tika izstrādātas deviņas klases, lai nodrošinātu nepieciešamo funkcionalitāti un mēģinātu novērst 1.1 nodaļā apskatītās problēmas. Klašu mantošanas diagramma ir apskatāma 2.1 attēlā.



1.2. att. SimpleORM mantošanas diagramma

Diagramma ir automātiski ģenerēta ar phpDocumentor un tapusi pēc pirmā izstrādes posma, kad tika iegūta papildinājuma PHP versija.

2.1. PDO

PDO ir PHP paplašinājums, kas definē datubāžu piekļuves saskarni. PDO nenodrošina datubāzes abstrakcijas līmeni, bet apraksta datu piekļuves abstrakcijas slāni, kas nozīmē, ka neatkarīgi no izvēlētās datubāzes pārvaldības sistēmas, tiks izmantotas vienas un tās pašas funkcijas, lai izpildītu vaicājumus un nolasītu datus [4].

Lai varētu darboties ar datubāzi ir jāizmanto datubāzes veidam speciāli domāts PDO draiveris, jo pats PDO tikai apraksta nepieciešamās funkcijas, bet nenosaka kā tās tiek ieviestas.

Izstrādātais paplašinājums tiek paredzēts darbā ar MySQL datubāzi, kas nozīmē, ka datubāzes piekļuvei SimpleORM izmantos MySQL PDO draiveri un funkcionalitāte tiks nodrošināta šim PDO draiverim.

2.2. \SimpleORM\Connection

Klase ir PDO klases apakšklase. Paredzēta, lai izveidotu savienojumu ar MySQL datubāzi un izmantotu MySQL PDO draivera sniegto funkcionalitāti vaicājumus izpildīšanai un datu nolasīšanai. Klasei ir 3 atribūti, 6 metodes.

2.2.1. Atribūti

Klasei ir 3 atribūti – 2 instances un 1 statisks, no kuriem 2 ir publiski un 1 statisks. Klases atribūtu aprakstu var apskatīt tabulā 2.1.

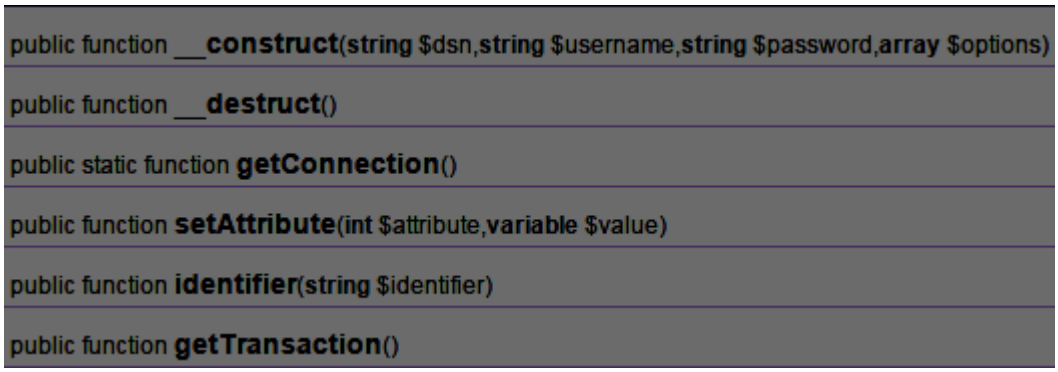
2.1. tabula

\SimpleORM\Connection atribūti

Redzamība	Veids	Atribūts	Apraksts
Publisks	Instances	_transaction	Aktīvā transakcija
Publisks	Instances	_options	Datubāzes savienojuma iestatījumi
Aizsargāts	Statisks	_main_connection	Datubāzes galvenais savienojums

2.2.2. Metodes

Klasei ir 5 publiskas metodes, no kurām 1 ir statiska. Klases metodes var apskatīt attēlā 2.1. un to aprakstus var izlasīt tabulā 2.2.



2.1.att. \SimpleORM\Connection klases metodes

2.2. tabula

\SimpleORM\Connection klases metožu apraksts

Metode	Apraksts
__construct	Publisks klases konsruktors, kas inicializē jaunu klases objektu un izveido jaunu savienojumu ar datubāzi.
__destruct	Publisks klases destruktors, lai atstatītu _transaction atribūtu pirms objekta dzīves cikla beigām.
_getConnection	Atgriež datubāzes galveno savienojumu. Paredzēts, lai ir vienmēr ir iespējams tikt pie datubāzes savienojuma.
setAttribute	Iestata MySQL draivera atribūtu
identifier	Filtrē datubāžu tabulu, kolonu nosaukumus un apliekot apkārt ` tiem.
getTransaction	Iegūst jaunu vai aktīvo datubāzes transackcijas savienojumu

2.3. \SimpleORM\Globals

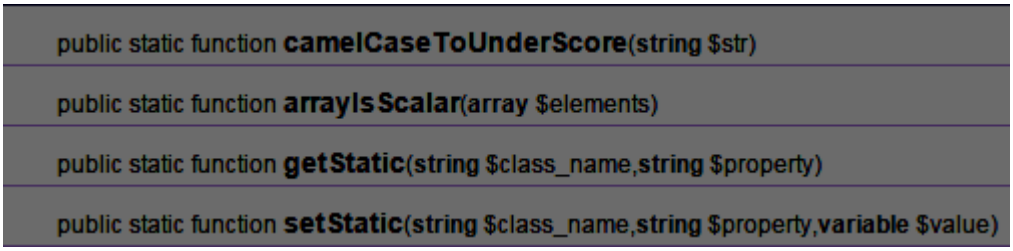
Klase, kas paredzēta globālu funkciju glabāšanai, lai tās vienmēr varētu izsaukt no jebkuras klases. Klasei ir 4 metodes un visas šīs klases metodes vienmēr ir paredzētas statiskas, jo šai klasei nekad nevajadzētu tik inicializētai.

2.3.1. Atribūti

Klasei nav atribūtu un netiek plānoti, ka tādi būs.

2.3.2. Metodes

Klasei ir 4 statiskas metodes. Klases metodes var apskatīt attēlā 2.2. un to aprakstus var izlasīt tabulā 2.3.



2.2.att. \SimpleORM\Globals klases metodes

2.3. tabula

\SimpleORM\Globals klases metožu apraksts

Metode	Apraksts
camelCaseToUnderScore	Pārveido camelCaseToUnderScore uz camel_case_to_under_score. Paredzēts, lai pārveidotu klašu nosaukumus par tabulu nosaukumiem.
arrayIsScalar	Pārbauda vai masīvs ir viendimensionāls un visi elementi ir skalāras vērtības.
getStatic	Nolasa klases statisko atribūtu – aizvietojums atslēgvārdam „static”.
setStatic	Iestata klases statisko atribūtu.

2.4. \SimpleORM\Model

Klase, kas domāta, lai reprezentētu tabulas ierakstus. Klasei ir ietverts CRUD un relāciju atbalsts. Klasei ir 7 atribūti, 33 metodes un 6 klases konstantes. Atribūtu aprakstu var apskatīt 2.4. tabulā, bet metodes var apskatīt attēlā 2.3. un izlasīt to aprakstus var tabulā 2.5.

2.4.1. Atribūti

Klasei ir 7 atribūti – 2 instances, 5 statiskie un visi ir aizsargāti. Klasei ir 6 konstantes, kas reprezentē datu tipu un objekta pašreizējo stāvokli. Detalizētāks atribūtu apraksts ir pieejams zemāk esošajā 2.4. tabulā.

2.4. tabula

\SimpleORM\\Model klases atribūti

Redzamība	Veids	Atribūts	Apraksts
Publisks	Konstante	PARAM_ENUM	Reprezentē MySQL enum datu tipu.

Publisks	Konstante	PARAM_REQUIRED	Reprezentē MySQL NOT NULL kolonas.
Publisks	Konstante	PARAM_UNSIGNED	Reprezentē MySQL unsigned kolonas.
Publisks	Konstante	STATE_NEW	Reprezentē jauna objekta stāvokli.
Publisks	Konstante	STATE_NONE	Reprezentē esoša objekta stāvokli, kurš nav mainīts.
Publisks	Konstante	STATE_DELETED	Reprezentē dzēsta objekta stāvokli.
Aizsargāts	Instances	_connection	Datubāzes savienojums.
Aizsargāts	Instances	_state	Nosaka objekta stāvokli.
Aizsargāts	Statisks	_columns	Tabulas kolonu definīcijas.
Aizsargāts	Statisks	_relations	Tabulas relācijas ar citām tabulām.
Aizsargāts	Statisks	_id_column	Tabulas identifikācijas kolona/-s.
Aizsargāts	Statisks	_unique_keys	Tabulas unikālo kolonu kombinācijas.
Aizsargāts	Statisks	_filters	Tabulas nemainīgie filtri.
Aizsargāts	Statisks	_validated	Vai tabulas definīcija ir pārbaudīta.

Daļa no vēlamās funkcionalitātes vel nav ieviesta, jo tā pašlaik nav tik svarīga un rezultātā ir pieejamas objekta konstantes, kurām pašlaik nav pielietojums, piemēram, PARAM_ENUM, PARAM_ REQUIRED un PARAM_UNSIGNED.

Atribūtam _id_column ir jābūt viendimensionālam masīvam, kas sastāv no primārās atslēgās kolonām, parasti, tas ir masīvs ar vienu elementu „id”. Atribūts _unique_keys, ja iestatīts, jābūt viendimensionālam masīvam, kur katrs elements ir viendimensionāls masīvs, kas sastāv no kolonām, kas veido unikālu kombināciju.

2.4.2. Metodes

2.3.att. \SimpleORM\Model klases metodes

2.5. tabula

\SimpleORM\Model klases metožu apraksts

Metode	Apraksts
getFilter	Iegūst tabulas filtru pēc nosaukuma.
getUniqueKey	Iegūst masīvu ar tabulas kolonam pēc kurām var unikāli identificēt noteiktu ierakstu tabulā.
setType	Iestata mainīgo tipu uz kādu no vērtībām.
getPrefixColumns	Iegūst masīvu ar tabulas kolonu nosaukumiem, kuriem priekšā ir pielikts norādītais priedēklis – pēc noklusējuma tas ir tabulas nosaukums.
getTableName	Iegūst tabulas nosaukumu, ko reprezentē šī klase.
_validateColumns	Pārbauda vai klases atribūts _columns ir pareizi definēts.

_validateID	Pārbauda vai klases atribūts _id_column ir pareizi definēts.
_validateFilters	Pārbauda vai klases atribūts _filters ir pareizi definēts.
_validateDefinition	Pārbauda vai klases nepieciešamie atribūti ir definēti.
__construct	Klases konstruktors.
construct	Statisks klases konstruktors, lai objektus var inicializēt dinamiski.
setConnection	Iestata objekta datubāzes savienojumu.
resetConnection	Noņem datubāzes savienojumu.
checkConnection	Pārbauda datubāzes savienojumu, ja nepieciešams, tad to iestata.
setState	Iestata objekta stāvokli.
insertQuery	Izveido \SimpleORM\Connection\Query\Insert objektu.
_addKey	Izveido \SimpleORM\Connection\Query pievieno identitātes nosacījumus.
updateQuery	Izveido \SimpleORM\Connection\Query\Update objektu.
deleteQuery	Izveido \SimpleORM\Connection\Query\Delete objektu.
insert	Izpilda objekta ievietošanu vai atjaunošanu.
update	Izpilda objekta ievietošanu vai atjaunošanu.
save	Izpilda objekta ievietošanu vai atjaunošanu.
delete	Izpilda objekta dzēšanu.
crud	Izpilda objekta ievietošanu, dzēšanu vai atjaunošanu.
beforeInsert	Metode, kas tiek izsaukta pirms objekta saglabāšanas.
afterInsert	Metode, kas tiek izsaukta pēc objekta saglabāšanas.
beforeUpdate	Metode, kas tiek izsaukta pirms objekta atjauninašanas.
afterUpdate	Metode, kas tiek izsaukta pēc objekta atjauninašanas.
beforeDelete	Metode, kas tiek izsaukta pirms objekta dzēšanas.
afterDelete	Metode, kas tiek izsaukta pēc objekta dzēšanas.
softDelete	Metode, ar kuru iegūst laukus, kas jāiestata, lai objekts tiktu uzskatīts par dzēstu.
_validateRelation	Pārbauda konkrētas relācijas definīciju.
getRelation	Iegūst objekta relāciju pēc nosaukuma.
query	Iegūst \SimpleORM\Connection\Query\Select\Model.

2.5. \SimpleORM\Connection\Query

Klase ir \SimpleORM\Connection apakšklase un paredzēta SQL vaicājumu veidošanai. Galvenais mērķis ir izskaust zema līmeņa SQL rakstīšanu un nodrošināt iespēju vaicājumu veidot objektorientēti. Klasei ir 13 atribūti un 35 metodes. Klases atribūtus var apskatīt attēlā 2.6, bet klases metodes attēlā 2.4. un to aprakstu tabulā 2.7.

2.5.1. Atribūti

Klasei ir 13 atribūti, no kuriem visi ir instances un aizsargāti. Atribūti ir ar aizsargātu redzamību, lai nodrošinātu, ka atribūti netiek sabojāti un programmētājam nāktos izmantot objektorientētu pieeju vaicājuma veidošanai. Detalizēts klases atribūtu saraksts ir pieejams 2.6. tabulā.

2.6. tabula

\SimpleORM\Connection\Query klases atribūti

Redzamība	Veids	Atribūts	Apraksts
Aizsargāts	Instances	columns	Glabā vēlamās kolonas ar kurām darboties.
Aizsargāts	Instances	from	Tabulas nosaukums pret kuru tiek vērsts vaicājums.
Aizsargāts	Instances	binds	Masīvs ar filtrētajām vērtībām.
Aizsargāts	Instances	_joins	Masīvs ar joins nosacījumiem.
Aizsargāts	Instances	_group	Masīvs ar grupēšanas nosacījumiem.
Aizsargāts	Instances	_having	Masīvs ar having nosacījumiem.
Aizsargāts	Instances	_order	Masīvs ar kārtošanas nosacījuemiem.
Aizsargāts	Instances	_connection	Datubāzes savienojums
Aizsargāts	Instances	_limit	Maksimālais atgriežamo ierakstu skaits.
Aizsargāts	Instances	_offset	Nobīdes apjoms.
Aizsargāts	Instances	_type	Vaicājuma tips – SELECT, INSERT, UPDATE, DELTE.
Aizsargāts	Instances	_conditions	Masīvs ar WHERE grupas nosacījumiem.
Aizsargāts	Instances	_filters	Tabulas nemainīgie filtri.
Aizsargāts	Instances	_active_group	Aktīvā WHERE grupa.

2.5.2. Metodes

Klasei ir nepieciešamās metodes, lai būtu iespējams izveidot pietiekami sarežģītus SQL vaicājumus. Lielākoties publiskās instances metodes atgriež aktīvo instanci, nodrošinot iespēju veidot vaicājumus objektorientēti. Klases metodes var apskatīt 2.4. attēlā un detalizētu aprakstu tabulā 2.7.

2.7. tabula

\SimpleORM\Connection\Query klases metožu apraksts

Metode	Apraksts
__construct	Klases konstruktors.
addColumns	Papildina _columns atribūtu
getBinds	Atgriež _binds atribūtu.
_setType	Iestata _type atribūtu.
bindString	Pievieno vēlamo vērtību _binds atribūtam un atgriež tā atslēgu _binds masīvā.
_bindCondition	Izfiltrē nosacījumu.
aliasCondition	Nosacījumā aizvieto tabulas nosaukumu ar aizstājvārdu.
join	Izveido join nosacījumu un pievieno to _joins atribūtam.
open	(
andOpen	AND (
orOpen	OR (
close	)
_condition	"=", ">=", "<=", ">", "<", "LIKE"
condition	_condition
orCondition	OR _condition
andCondition	AND _condition
_in	IN(...,...)

inValues	_in
andIn	AND _in
orIn	OR _in
_between	BETWEEN .. AND ...
between	_between
andBetween	AND _between
orBetween	OR _between
getConditions	Apstrādā _conditions masīvu un ģenerē WHERE nosacījumu ievērojot iekavu secību(open, andOpen, orOpen, close).
_filter	Iegūst klases filtru.
filter	_filter
andFilter	AND _filter
orFilter	OR _filter
limit	LIMIT ...
offset	OFFSET ...
order	... DESC|ASC un pieveno _order atribūtam.
group	Pievieno kolonu _group atribūtam.
getStatement	Atgriež ģenerētu SQL vaicājuma daļu – joins | conditions | group | having | limit | offset.

Ievērībai der zināt, ka HAVING funkcionalitāte pašreizējā paplašinājuma versijā nav līdz galam ieviesta, jo tā tiek reti lietota un patreiz nav nepieciešama. To tiek plānots ieviest nākamajā versijā kopā ar FOR UPDATE nosacījumu un uzlabotiem \SimpleORM\Model klases filtriem. Visu publisko metožu argumenti, kuri tiek lietoti vaicājuma veidošanā tiek filtrēti, lai programmētājam nav nemitīgi jādomā par lietotāja ievada filtrāciju.

Lielākā daļa no klases metodēm, kuras sākās ar „_”, sākumā bija plānotas kā privātas metodes. Privātās metodes un atribūti Zephir valodā tomēr darbojās nedaudz savādāk kā PHP, tapēc vieglākais un ātrākais variants kā novērst kļūdas, bija mainīt metodes redzamību uz aizsargāta.

2.4.att. \SimpleORM\Model\Query klases metodes

2.6. \SimpleORM\Connection\Query\Insert

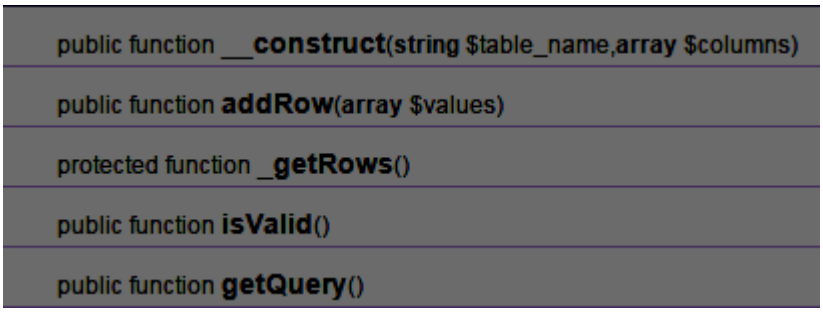
Klase ir \SimpleORM\Connection\Query apakšklase. Klase ir paredzēta INSERT vaicājumu veidošanai. Tāpat kā virsklasei, tai ir nodrošināta objektorientēta pieeja. Tai ir viens atribūts un piecas metodes.

2.6.1. Atribūti

Klasei ir viens aizsargāts atribūts _rows. Tas uzskaita, cik tabulas ierakstu tiks ievietoti tabulā un tiek izmantots, lai noteiktu vai vaicājums ir derīgs un izpildāms.

2.6.2. Metodes

Klasei ir piecas metodes, no kurām viena ir aizsargāta un tai nav statisku metožu. Klases metodes var apskatīt attēlā 2.5. un detalizētu aprakstu var iegūt tabulā 2.8.



2.5.att. \SimpleORM\Model\Quer\Insert klases metodes

2.8. tabula

\SimpleORM\Connection\Query\Insert klases metožu apraksts

Metode	Apraksts
__construct	Klases konstruktors.
addRow	Saglabā tabulas rindas vērtības _binds atribūtā.
_getRows	Iegūst pievienojamo rindu nosacījumu.
isValid	Pārbauda vai ir bijis vismaz viens veiksmīgs addRow izsaukums.
getQuery	Ģenerē INSERT vaicājumu.

2.7. \SimpleORM\Connection\Query\Delete

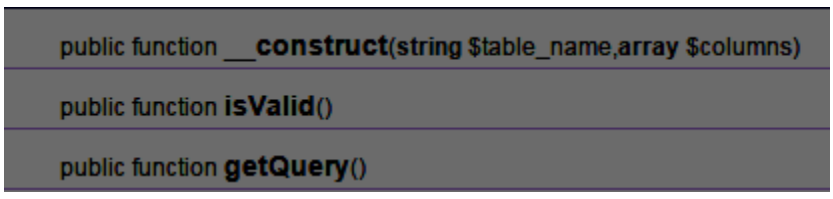
Klase ir \SimpleORM\Connection\Query apakšklase. Klase ir paredzēta DELETE vaicājumu veidošanai. Tāpat kā virsklasei, tai ir nodrošināta objektorientēta pieeja. Tai ir trīs metodes.

2.7.1. Atribūti

Klasei uz doto brīdi nav atribūti.

2.7.2. Metodes

Klasei ir 3 publiskas metodes. Klasei nav statisko metožu. Klases metodes var aspkatīt attēlā 2.6. un iegūt detalizētu aprakstu var tabulā 2.9.



2.6.att. \SimpleORM\Model\Query\Delete klases metodes

2.9. tabula

\SimpleORM\Connection\Query\Delete klases metožu apraksts

Metode	Apraksts
__construct	Klases konstruktors.
isValid	Pārbauda vai vaicājums ir valīds.
getQuery	Ģenerē DELETE vaicājumu.

2.8. \SimpleORM\Connection\Query\Update

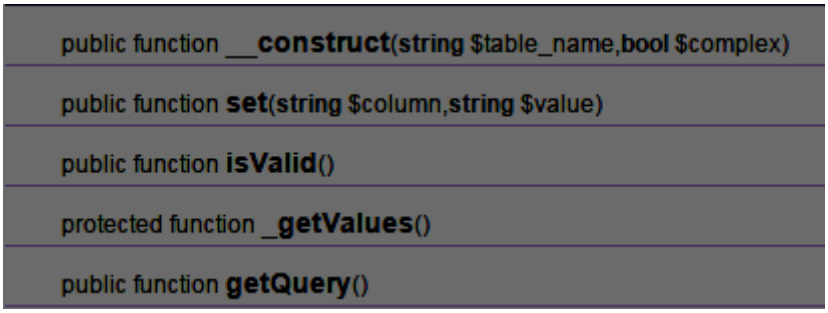
Klase ir \SimpleORM\Connection\Query apakšklase. Klase ir paredzēta UPDATE vaicājumu veidošanai. Tāpat kā virsklasei, tai ir nodrošināta objektorientēta pieeja. Tai ir viens atribūts un piecas metodes.

2.8.1. Atribūti

Klasei ir viens aizsargāts atribūts _complex. Tas paredzēts gadījumos, kad UPDATE vairs nav vienkāršs rindas vērtības pamainīšana, bet satur sarežģītu loģiku, piemēram, CASE vai IF nosacījumus.

2.8.2. Metodes

Klasei ir 5 metodes, no kurām 4 ir publiskas metodes un viena aizsargāta. Klasei nav statisko metožu. Klases metodes var apskatīt attēlā 2.7. un iegūt detalizētu aprakstu var tabulā 2.10.



2.7.att. \SimpleORM\Model\Query\Update klases metodes

2.10. tabula

\SimpleORM\Connection\Query\Update klases metožu apraksts

Metode	Apraksts
__construct	Klases konstruktors.
set	Piefiksē kolonas jauno vērtību un ieraksta to _binds atribūtā.
isValid	Pārbauda vai vaicājums ir derīgs.
_getValues	Iegūst atjaunojamo rindu nosacījumus.
getQuery	Ģenerē UPDATE vaicājumu.

2.9. \SimpleORM\Connection\Query\Select

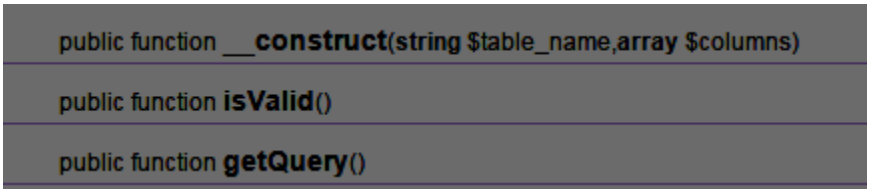
Klase ir \SimpleORM\Connection\Query apakšklase. Klase ir paredzēta SELECT vaicājumu veidošanai. Tāpat kā virsklasei, tai ir nodrošināta objektorientēta pieeja. Tai ir viens atribūts un piecas metodes.

2.9.1. Atribūti

Klasei nav savi atribūti, tā manto virsklases atribūtus.

2.9.2. Metodes

Klasei ir 3 publiskas metodes, tai nav statisku metožu. Klases metodes var aspkatīt attēlā 2.8. un iegūt detalizētu aprakstu var tabulā 2.11.



2.8.att. \SimpleORM\Model\Query\Select klases metodes

2.11. tabula

\SimpleORM\Connection\Query\Select klases metožu apraksts

Metode	Apraksts
__construct	Klases konstruktors.
isValid	Pārbauda vai vaicājums ir valīds.
getQuery	Ģenerē SELECT vaicājumu.

2.10.\SimpleORM\Connection\Query\Select\Model

Klase ir \SimpleORM\Connection\Query\Select apakšklase ar trim aizsargātiem atribūtiem un 8 metodēm, no kurām 4 ir aizsargātas. Klase ir paredzēta SELECT vaicājuma realizācijai, kuru rezultātā tiek iegūts masīvs ar \SimpleORM\Model klases objektiem. Tāpat kā virsklasei, tai ir nodrošināta objektorientēta pieeja.

2.10.1. Atribūti

Klasei ir trīs aizsargāti atribūti un mantotie atribūti. Atribūti ir aizsargāti, lai nav iespējams sabojāt tos. Detalizēts atribūtu apraksts ir tabulā 2.12.

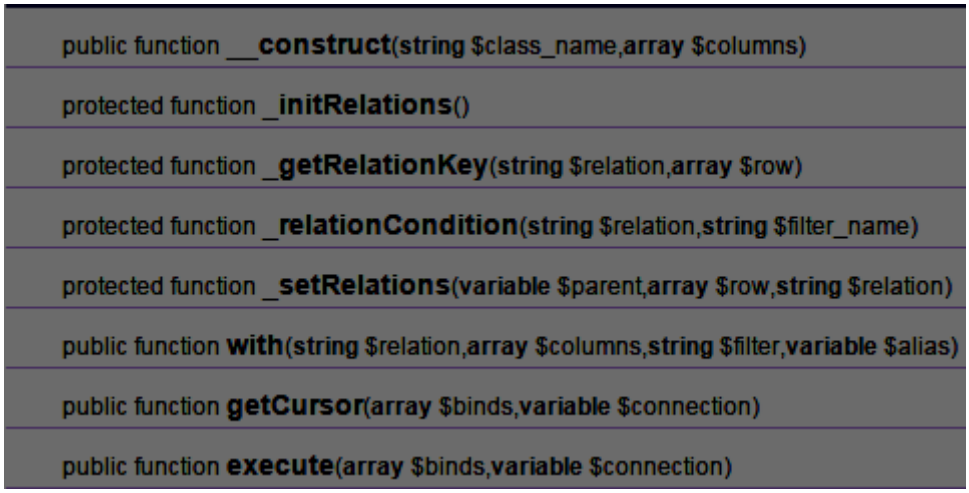
2.12. tabula

\SimpleORM\Connection\Query\Model klases atribūti

Redzamība	Veids	Atribūts	Apraksts
Aizsargāts	Instances	_relations	Glabā vaicājuma izmantotās tabulas un relācijas.
Aizsargāts	Instances	_class	Pamatklase pret kuru tiek veikts pieprasījums.
Aizsargāts	Instances	_reserved	Pamatklases tabulas nosaukums.

2.10.2. Metodes

Klasei ir 8 metodes, no kurām 4 ir aizsargātas. Klases metodes var apskatīt attēlā 2.9. un izlasīt aprakstu var tabulā 2.13.



2.9.att. \SimpleORM\Model\Query\Select \Model klases metodes

2.13. tabula

\SimpleORM\Connection\Query\Select\Model klases metožu apraksts

Metode	Apraksts
__construct	Klases konstruktors.
_initRelations	Sagatavo sākuma stāvokli.
_getRelationKey	Iegūst unikālu relācijas/rindas identifikātoru.
_relationCondition	Sagatavo relācijas JOIN ON nosacījumu.
_setRelations	Sasaista \SimpleORM\Model objektu ar tam atbilstošajām relācijām.
with	Ar \SimpleORM\Model relāciju.
getCursor	Izpilda veidoto vaicājumu un iegūst PDOStatement objektu.
execute	Iegūst masīvu ar \SimpleORM\Model objektiem, kas atbilst norādītajām relācijām.

3. SimpleORM NOVĒRTĒŠANA

Primārais ir pārbaudīt vai SimpleORM nodrošina objektrelācijas pamatfunkcionalitāti un pēc tam salīdzināt izstrādāto Zephir paplašinājumu ar PHP versiju. Tabulā tiek apskatītas svarīgākās pamatfunkcijas, kurām būtu jābūt nodrošinātām risinājumā.

3.1. tabula

Nepieciešamās funkcijas

Funkcija	Apraksts	Ir	Piemērs
CRUD	Iespēja izveidot, atjaunināt un dzēst datubāzes ierakstus izmantojot objektorientētas programmēšanas principus.	+	$obj = new Model(); $obj->save(); $obj->name = „Jānis”; $obj->delete();
Search	Iegūt virtuālo datubāzes ierakstu.	+	$obj = Model::query()->condition(„name”, „=”, „Jānis”)->limit(1)->execute(); $obj = array_pop($obj);
Transakcijas	Transakciju atbalsts, izmaiņas netiek veiktas līdz transakcija netiek pabeigta.	+	$obj = new Model(); $transaction = \SimpleORM\Connection::getTransaction(); $obj->setConnection($transaction); $obj->name = „Līga”; $obj->save(); $transaction->rollback();
FOR UPDATE	Iespēja bloķēt tabulas ierakstus.	-	

Relācijas	Virtuālo objektu relācijas atbalsts.	+	$objects = Model::query()->with(„parts”)->between(„id”, 1, 30)->execute();
			$obj = array_pop($objects);
			While($objects->parts) {
			$part = array_pop($objects->parts);
			$part->delete();
			}

Filtrācija	Ievada filtrācija.	+	

Lai salīdzinātu ātrdarbību SimpleORM ar tā PHP versiju, tika atkārtots atkārtots datubāzes pieprasījums ar relācijām, kas ir apskatāms 3.1. attēls un testu rezultāti ir apskatāmi 3.2. tabulā.

```
$start = 0;
for ( $i = 0; $i < $iterations; $i++ ) {
    \Time::start();
    $rows = \PHP\Models\Events::query()
            ->with( 'langs', null, 'lv' )
            ->with( 'event_dates', null, 'times_all' )
            ->with( 'event_dates.price_groups', null, 'editable' )
            ->group( '`event_dates`.`id`' )
            ->order( '`event_dates`.event_datetime', 'DESC' )
            ->filter( '\Models\Events', 'editable' )
            ->limit( 100 )
            ->execute();
    $start += \Time::end();
}

return $start;
```

3.1. att. Izpildītais SQL pieprasījums

3.2. tabula

Ātrdarbības salīdzinājuma testu rezultāti

Iterāciju skaits	\PHP\SimpleORM vidēji ms	\SimpleORM vidēji ms
10	0.0427	0.063
25	0.0475	Neizpildās
40	0.04575	Neizpildās
55	0.0463	Neizpildās
70	0.0489	Neizpildās
85	0.0495	Neizpildās
100	0.04676	Neizpildās

Var secināt, ka paplašinājumam nepatīk, ka tiek izpildīts viens un tas pats ciklā, vai arī tiek bojāta atmiņa. Katrā ziņā šī problēma ir jāizpēta sīkāk, lai to varētu novērst. Pēc tabulas datiem var secināt, ka pašlaik PHP versija ir par 30% ātrāka, bet tas tāpēc, ka Zephir vel nav apgūts un iespējams kaut kas tiek darīts nepareizi.

REZULTĀTI

Bakalaura darba gaitā tika izstrādāts objektrelācijas risinājums PHP valodā un tas pārnests uz Zephir valodu, kur tas tika kompilēts, iegūstot papildinājumu simpleorm.so, kuru ir iespējams pievienot PHP papildinājumiem un izmantot aprakstīto funkcionalitāti paralēli PHP.

Gūtas iemaņas darbā ar Zephir programmēšanas valdou, kuru rezultātā gūts dziļāks ieskats PHP valodā. Uzrakstīti pāris funkciju izsaukumi C valodā, jo Zephir vel neatbalsta visu PHP funkcionalitāti un nākas rakstīt C kodu.

Papildinājumu ir plānots izmantot tālāk, tapēc pirms to sākt lietot praksē, būs jāveic uzmanīga C koda pārlasīšana un iedziļināšanās tajā, jo, iespējams, varēs atrast vietas, ko var uzlabot un tas dos papildus zināšanas par radīto papildinājumu.

Darbā izvirzītais mērķis tika sasniegts un ir jau zināmi virzieni kā tālāk uzlabot radīto risinājumu.

SECINĀJUMI

Zephir valodā ir iespējams veidot PHP paplašinājumus un to ir viegli realizēt, jo sākumā paplašinājumu var uzrakstīt PHP un tad pārrakstīt Zephir valodā. Zephir galvenais bonuss ir tā savietojamība ar PHP un iespēja paslēpt realizāciju, jo radītais kods tiek kompilēts uz māšīnkodu.

PHP versiju ir diezgan grūti optimizēt, bet Zephir versijai ir daudz iespējas to optimizēt, bet, lai to veiktu ir jāiedziļinās sīkāk Zephir kompilācijas procesos un jāizpēta kā labāk risināt statiskos atribūtus un metodes.

IZMANTOTĀ LITERATŪRA UN AVOTI

1. Object-relational mapping [tiešsaiste] – [atsauce 01.06.2015]. Pieejams: http://en.wikipedia.org/wiki/Object-relational_mapping
2. LZA Terminoloģijas komisija, Akadēmiskā terminu datubāze AkadTerm [tiešsaiste]. Pieejams: http://termini.lza.lv/
3. Zephir Language reference [tiešsaiste]. Pieejams: http://zephir-lang.com/
4. PHP Data Objets [tiešsaiste]. Pieejams: http://php.net/manual/en/book.pdo.php
5. Zephir All discussions [tiešsaiste] – [atsauce 27.06.2015]. Pieejams: http://forum.zephir-lang.com/discussions/new
6. Welcome to VirtualBox.org! [tiešsaiste]. Pieejams: https://www.virtualbox.org/
7. Managing PHP 5.4 Extensions on Ubuntu [tiešsaiste]. Pieejams: http://www.lornajane.net/posts/2012/managing-php-5-4-extensions-on-ubuntu

PIELIKUMI

1. Pielikums. Zephir koda fragmenti

```
namespace SimpleORM;

class Globals
{
    public static function camelCaseToUnderScore( const string! str = null ) -> string
    {
        var value;
        let value = explode( "\\", str ),
            value = array_pop( value ),
            value = lcfirst ( value ),
            value = preg_replace( "/([a-z])([A-Z])/", "$1_$2", value );
        return strtolower( value );
    }

    public static function arrayIsScalar( const array! elements = null ) -> boolean
    {
        var i, key, element;
        let i = 0;

        for key, element in elements {
            if ! is_scalar( element ) || key != i {
                return false;
            }

            let i++;
        }

        return true;
    }

    public static function getStatic( string! class_name, string! property ) -> var
    {
        var value = null;

        %{
            zephir_read_static_property(
                &value,
                Z_STRVAL_P(class_name_param),
                Z_STRLEN_P(class_name_param),
                Z_STRVAL_P(property_param),
                Z_STRLEN_P(property_param) TSRMLS_DC
            );
        }%

        return value;
    }

    public static function setStatic( string! class_name, string! property, var value = null ) -> void
    {
        %{
            zephir_update_static_property(
                Z_STRVAL_P(class_name_param),
                Z_STRLEN_P(class_name_param),
                Z_STRVAL_P(property_param),
                Z_STRLEN_P(property_param),
                &value TSRMLS_DC
            );
        }%
    }
}
```

```
namespace SimpleORM\Connection\Query\Select;

class Model extends \SimpleORM\Connection\Query\Select
{
    protected _relations = null;
    protected _class     = null;
    protected _reserved  = null;

    public function __construct( string! class_name, array! columns = null )
    {
        if ! is_subclass_of( class_name, "\SimpleORM\Model" ) {
            throw new \Exception( "Invalid class provided" );
        }

        let this->_class     = class_name,
            this->_reserved  = call_user_func( [ class_name, "getTableName" ] );
        parent::__construct( this->_reserved, call_user_func( [ class_name, "getPrefixColumns" ], columns ) );
        this->_initRelations();
    }

    protected function _initRelations() -> void
    {
        var relation, relations, key;
        let relation             = new \stdClass(),
            relation->parent     = null,
            relation->type       = null,
            relation->unique_key = call_user_func( [ this->_class, "getUniqueKey" ], true );

        let key                          = this->_reserved,
            relations                    = new \stdClass(),
            relations->{ key }           = relation,
            this->_relations             = relations;
    }

    protected function _getRelationKey( string! relation, array! row ) -> string
    {
        string key = "";
        var column, keys;
        let keys = [];

        if isset this->_relations->{ relation }->unique_key {
            for column in this->_relations->{ relation }->unique_key {
                if isset row[ relation."_".column ] {
                    let keys[] = row[ relation."_".column ];
                }
            }
        }
```

```
        }

        if ! empty keys {
            let key = implode( "_", keys );
        }

        return key;
    }

    protected function _relationCondition( string! relation, string! filter_name ) -> string
    {
        var conditions, reference_table, table, key, value, filter;
        let conditions      = [],
            reference_table = call_user_func( [ this->_relations->{ relation }->reference_model, "getTableName" ] ),
            table           = this->_relations->{ relation }->parent;

        for key, value in this->_relations->{ relation }->foreign_columns {
            let conditions[] = reference_table.".".value." = ".table.".". this->_relations->{ relation }->columns[key];
        }

        if ! empty filter_name {
            let filter = call_user_func( [ this->_relations->{ relation }->reference_model, "getFilter" ], filter_name );

            if ! empty filter {
                let filter               = self::_main_connection->identifier( this->bindCondition( filter ) ),
                    conditions[]         = filter;
            }
        }

        return "(".implode( " AND ", conditions ).")";
    }

    protected function _setRelations( var parent = null, array! row, string! relation ) -> <\SimpleORM\Model>
    {
        var relationship, key, new_object, tmp;
        let relationship = this->_relations->{ relation },
            key          = this->_getRelationKey( relation, row ),
            new_object   = null;

        if ! empty key {
            switch relationship->type {
                case "belongs_to":
                    if ( ! isset( parent->{ relation } ) ) {
                        let parent->{ relation } = call_user_func( [ relationship->reference_model, "construct" ], row, relation );
                    }
```

```
85
86          if ! empty key {
87              switch relationship->type {
88                  case "belongs_to":
89                      if ( ! isset( parent->{ relation } ) ) {
90                          let parent->{ relation } = call_user_func( [ relationship->reference_model, "construct" ], row, relation );
91                      }
92
93                      let new_object = parent->{ relation };
94                      break;
95                  case "has_many":
96                      if ! isset parent->{relation} {
97                          let parent->{relation} = [];
98                      }
99
100                     if ( ! isset( parent->{ relation }[ key ] ) ) {
101                         let tmp = call_user_func( [ relationship->reference_model, "construct" ], row, relation );
102
103                         %{
104                             zephir_update_property_array(parent, Z_STRVAL_P(relation), Z_STRLEN_P(relation), key, tmp TSRMLS_DC);
105                         }%
106                     }
107
108                     let new_object = parent->{ relation }[ key ];
109                     break;
110                 default:
111                     if empty parent {
112                         let parent = call_user_func( [ this->_class, "construct" ], row, this->_reserved );
113                     }
114
115                     let new_object = parent;
116                     break;
117             }
118
119             if ! empty relationship->relations {
120                 for tmp in relationship->relations {
121                     this->_setRelations( new_object, row, tmp );
122                 }
123             }
124         }
125
126         return new_object;
127     }
128
129     public function with( string! relation, array! columns = null, string! filter = null, var alias = null ) -> <\SimpleORM\Connection\Query\Select>
130     {
131         var parts, base_class, tmp;
```

```
131         var parts, base_class, tmp;
132         let parts       = explode( ".", relation ),
133             base_class  = this->_class;
134
135         if count( parts ) == 1 {
136             let parts[1] = parts[0],
137                 parts[0] = this->_reserved;
138         } elseif property_exists( this->_relations, parts[0] ) {
139             let tmp        = parts[0],
140                 base_class = this->_relations->{tmp}->reference_model;
141         } else {
142             throw new \Exception( "Invalid relation provided ".relation );
143         }
144
145         var relationship;
146         let relationship = call_user_func( [ base_class, "getRelation" ], parts[1] );
147
148         if ! empty relationship {
149             settype( relationship, "object" );
150             let relationship->parent     = parts[0],
151                 relationship->unique_key = call_user_func( [ relationship->reference_model, "getUniqueKey" ], true );
152
153             if ! empty alias {
154                 let parts[1] = alias;
155             } else {
156                 let alias = parts[1];
157             }
158
159             if property_exists( this->_relations, parts[1] ) {
160                 throw new \Exception( "Not unique table name ".parts[1]." for relation ".relation );
161             }
162
163             var key;
164             let key             = parts[1],
165                 tmp             = this->_relations,
166                 tmp->{key}      = relationship;
167
168             this->addColumns( call_user_func( [ relationship->reference_model, "getPrefixColumns" ], columns, alias ) );
169             this->join( call_user_func( [ relationship->reference_model, "getTableName" ] ), this->_relationCondition( parts[1], filter ), false, alias
170
171             let tmp = this->_relations,
172                 key = parts[0],
173                 tmp = tmp->{key},
174                 tmp->relations[] = parts[1];
175         }
176
177         return this;
```

```
    public function getCursor( array! binds = null, <\SimpleORM\Connection> connection = null ) -> <\PDOStatement>
    {
        if empty connection {
            let connection = \SimpleORM\Connection::getConnection();
        } elseif ! is_subclass_of( connection, "\\PDO" ) {
            throw new \Exception( "If passed must be instance of PDO" );
        }

        var statement;
        let statement = connection->prepare( this->getQuery(), [ \PDO::ATTR_CURSOR : \PDO::CURSOR_FWDONLY ] );
        statement->setFetchMode( \PDO::FETCH_ASSOC );

        if this->binds && binds {
            let binds = array_merge( this->binds, binds );
        } elseif empty binds {
            let binds = this->binds;
        }

        statement->execute( binds );
        return statement;
    }

    public function execute( array! binds = null, <\SimpleORM\Connection> connection = null ) -> array
    {
        var cursor = null, row = null, key;
        array results = [];
        let cursor    = this->getCursor( binds, connection ),
            row       = cursor->{"fetch"}();

        while row {
            let key = this->_getRelationKey( this->_reserved, row );

            if ( ! isset( results[key] ) ) {
                let results[key] = this->_setRelations( null, row, this->_reserved );
            } else {
                this->_setRelations( results[key], row, this->_reserved );
            }

            let row = cursor->{"fetch"}();
        }

        cursor->closeCursor();
        return results;
    }
}
```

2. Pielikus. PHP koda fragmenti

```
<?php
namespace PHP\SimpleORM\Connection\Query\Select;

class Model extends \PHP\SimpleORM\Connection\Query\Select {
    private $_relations = null;
    private $_class     = null;
    private $_reserved  = null;

    function __construct( $class, array $columns = null )  {
        if ( ! is_subclass_of( $class, "\PHP\SimpleORM\Model" ) ) {
            throw new \Exception( "Invalid class provided" );
        }

        $this->_class     = $class;
        $this->_reserved  = call_user_func( [ $class, "getTableName" ] );
        parent::__construct( $this->_reserved, call_user_func( [ $class, "getPrefixColumns" ], $columns ) );
        $this->_initRelations();
    }

    private function _initRelations() {
        $this->_relations                                = new \stdClass();
        $this->_relations->{ $this->_reserved }          = new \stdClass();
        $this->_relations->{ $this->_reserved }->parent  = null;
        $this->_relations->{ $this->_reserved }->type    = null;
        $this->_relations->{ $this->_reserved }->unique_key = call_user_func( [ $this->_class, "getUniqueKey" ], true );
    }
```

```php
    private function _getRelationKey( $relation, array $row ) {
        $key = null;

        if ( isset( $this->_relations->{ $relation }->unique_key ) ) {
            foreach ( $this->_relations->{ $relation }->unique_key as $column ) {
                if ( isset( $row[ $relation."_".$column ] ) ) {
                    $key[] = $row[ $relation."_".$column ];
                }
            }
        }

        if ( $key ) {
            $key = implode( '_', $key );
        }

        return $key;
    }

    private function _relationCondition( $relation, $filter ) {
        $conditions      = [];
        $reference_table = call_user_func( [ $this->_relations->{ $relation }->reference_model, "getTableName" ] );
        $table           = $this->_relations->{ $relation }->parent;

        foreach ( $this->_relations->{ $relation }->foreign_columns as $key => $value ) {
            $conditions[] = $reference_table.".".$value." = ".$table.".". $this->_relations->{ $relation }->columns[$key];
        }

        if ( $filter ) {
            $filter = call_user_func( [ $this->_relations->{ $relation }->reference_model, "getFilter" ], $filter );

            if ( $filter ) {
                $filter               = $this::$_main_connection->identifier( $this->bindCondition( $filter ) );
                $conditions[]         = $filter;
            }
        }

        return "(".implode( " AND ", $conditions ).")";
    }
```

```php
    private function _setRelations( \PHP\SimpleORM\Model $parent = null, array $row, $relation ) {
        $relationship = $this->_relations->{ $relation };
        $key          = $this->_getRelationKey( $relation, $row );
        $object       = null;

        if ( $key ) {
            switch ( $relationship->type ) {
                case 'belongs_to':
                    if ( ! isset( $parent->{ $relation } ) ) {
                        $parent->{ $relation } = call_user_func( [ $relationship->reference_model, "construct" ], $row, $relation );
                    }

                    $object = $parent->{ $relation };
                    break;
                case 'has_many':
                    if ( ! isset( $parent->{ $relation }[ $key ] ) ) {
                        $parent->{ $relation }[ $key ] = call_user_func( [ $relationship->reference_model, "construct" ], $row, $relation );
                    }

                    $object = $parent->{ $relation }[ $key ];
                    break;
                default:
                    if ( ! $parent ) {
                        $parent = call_user_func( [ $this->_class, "construct" ], $row, $this->_reserved );
                    }

                    $object = $parent;
                    break;
            }

            if ( isset( $relationship->relations ) ) {
                foreach ( $relationship->relations as $relation ) {
                    $this->_setRelations( $object, $row, $relation );
                }
            }
        }

        return $object;
    }
```

```php
    public function with( $relation, array $columns = null, $filter = null, $alias = null ) {
        $parts        = explode( ".", $relation );
        $base_class   = $this->_class;

        if ( count( $parts ) == 1 ) {
            $parts[1] = $parts[0];
            $parts[0] = $this->_reserved;
        } else if ( property_exists( $this->_relations, $parts[0] ) ) {
            $base_class = $this->_relations->{ $parts[0] }->reference_model;
        } else {
            throw new \Exception( "Invalid relation provided ".$relation );
        }

        $relationship = call_user_func( [ $base_class, "getRelation" ], $parts[1] );

        if ( $relationship ) {
            settype( $relationship, 'object' );
            $relationship->parent     = $parts[0];
            $relationship->unique_key = call_user_func( [ $relationship->reference_model, "getUniqueKey" ], true );

            if ( $alias ) {
                $parts[1] = $alias;
            } else {
                $alias = $parts[1];
            }

            if ( property_exists( $this->_relations, $parts[1] ) ) {
                throw new \Exception( "Not unique table name ".$parts[1]." for relation ".$relation );
            }

            $this->_relations->{$parts[1]} = $relationship;
            $this->addColumns( call_user_func( [ $relationship->reference_model, "getPrefixColumns" ], $columns, $alias ) );
            $this->join(
                call_user_func( [ $relationship->reference_model, "getTableName" ] ),
                $this->_relationCondition( $parts[1], $filter ),
                false,
                $alias
            );
            $this->_relations->{$parts[0]}->relations[] = $parts[1];
        }

        return $this;
    }
```

```php
    public function getCursor( array $binds = null, \PHP\SimpleORM\Connection $connection = null ) {
        if ( ! $connection ) {
            $connection = static::getConnection();
        }

        $statement = $connection->prepare( $this->getQuery(), [ static::ATTR_CURSOR => static::CURSOR_FWDONLY ] );
        $statement->setFetchMode( static::FETCH_ASSOC );

        if ( $this->binds and $binds ) {
            $binds = array_merge( $this->binds, $binds );
        } else if ( ! $binds ) {
            $binds = $this->binds;
        }

        $statement->execute( $binds );
        return $statement;
    }

    public function execute( array $binds = null, \PHP\SimpleORM\Connection $connection = null ) {
        $cursor    = $this->getCursor( $binds, $connection );
        $results   = [];

        foreach ( $cursor as $row ) {
            $key = $this->_getRelationKey( $this->_reserved, $row );

            if ( ! isset( $results[$key] ) ) {
                $results[$key] = $this->_setRelations( null, $row, $this->_reserved );
                continue;
            }

            $this->_setRelations( $results[$key], $row, $this->_reserved );
        }

        $cursor->closeCursor();
        return $results;
    }
}
```

3. Pielikums. Virtuālo objektu modeļu piemēri

```php
<?php
namespace Models;

class Events extends \SimpleORM\Model {
    protected static $_id_column    = 'id';
    protected static $_unique_keys = [['id']];
    protected static $_validated    = false;

    protected static $_columns     = [
        'id' => [
            'type' => '\PDO::PARAM_INT',
            'skip' => true
        ],
        'users_id' => [
            'type' => '\PDO::PARAM_INT',
        ],
        'event_type_id' => [
            'type' => '\PDO::PARAM_INT',
        ],
        'super_parent_id' => [
            'type' => '\PDO::PARAM_INT'
        ],
        'organizer_id' => [
            'type' => '\PDO::PARAM_INT'
        ]
    ];

    protected static $_relations = [
        'event_dates' => [
            'reference_model' => '\SimpleORM\EventDates',
            'type'            => 'has_many',
            'columns'         => [ 'id' ],
            'foreign_columns' => [ 'event_id' ]
        ],
        'langs' => [
            'reference_model' => '\SimpleORM\EventsLang',
            'type'            => 'has_many',
            'columns'         => [ 'id' ],
            'foreign_columns' => [ 'object_id' ]
        ],
    ];

    protected static $_filters = [
        'editable'  => [
            '( `events`.`deleted` = 0 )'
        ]
    ];
}
```

```php
<?php
namespace Models;

class EventDates extends \SimpleORM\Model {
    protected static $_id_column    = 'id';
    protected static $_unique_keys = [['id']];
    protected static $_validated    = false;

    protected static $_columns     = [
        'id' => [
            'type' => '\PDO::PARAM_INT',
            'skip' => true
        ],
        'event_id' => [
            'type' => '\PDO::PARAM_INT',
        ],
        'event_datetime' => [
            'type' => '\PDO::PARAM_STR'
        ],
        'plan_id' => [
            'type' => '\PDO::PARAM_STR'
        ],
        'report_status' => [
            'type' => '\PDO::PARAM_STR'
        ]
    ];

    protected static $_relations = [
        'event' => [
            'reference_model' => '\SimpleORM\Models\Events',
            'type'            => 'belongs_to',
            'columns'         => [ 'event_id' ],
            'foreign_columns' => [ 'id' ]
        ],
        'price_groups' => [
            'reference_model' => '\SimpleORM\Models\EventPriceGroups',
            'type'            => 'has_many',
            'columns'         => [ 'id' ],
            'foreign_columns' => [ 'event_date_id' ]
        ]
    ];

    protected static $_filters = [
        'times_all' => [
            'event_dates.deleted = 0 '
        ]
    ];
}
```